SABCO
Barez Automation Control
Systems & Electronic Industries

# راهنمای جامع STEP7

## (جلد دوم)

همراه با CD

مشتمل بر:

- برنامه‌نویسی به زبان S7-SCL
- برنامه‌نویسی به زبان S7-Graph
- پیکره بندی سیستم‌های Redundant
- پیکره‌بندی سیستم‌های Fail Safe
- و ...

تالیف:
مهندس محمدرضا ماهر
مهندس نوشین سعیدی

به اهتمام:
شرکت صابکو (دپارتمان آموزش)

بسم اللّه الرحمن الرحیم

# راهنمای جامع STEP7

# ( جلد دوم )

**تالیف:**

**مهندس محمد رضا ماهر**

**مهندس نوشین سعیدی**

**به اهتمام : شرکت صابکو**

بنام آفریننده کاملترین سیستم ها

## پیشگفتار مؤلف

تاخیری که در چاپ کتاب حاضر پیش آمد و علت آن عمدتاً مشغله های کاری زیاد اینجانب بود بنده را در حضور خوانندگان محترم بویژه همکارانی که جلد اول با استقبال ایشان مواجه شده بود شرمسار و سر افکنده ساخت . پیشاپیش از محضر تمامی این عزیزان پوزش میطلبم.

مطالبی که در این مجموعه گردآوری و عرضه شده در ادامه مباحث جلد اول و تکمیل کننده آنست. نحوه استفاده از فانکشن های سیستم ، برنامه نویسی PLC توسط زبان سطح بالا و روش گرافیکی و نهایتاً تشریح سیستم های Fail Safe و Redundant مطالب اصلی کتاب را تشکیل میدهد. تهیه و گردآوری دو موضوع اخیر (فصل ۱۵ و فصل ۱۷ ) توسط سرکار خانم مهندس نوشین سعیدی انجام شده است.

امیدوارم با عرضه این کتاب توانسته باشم گامی هرچند کوچک در جهت انتقال دانش اتوماسیون صنعتی و رفع برخی از مشکلات و ابهامات همکارانم در صنعت برداشته باشم.

برخود لازم میدانم از جناب آقای مهندس علی بارزی که همچون گذشته مشوق اینجانب در تالیف این کتاب بودند و سرکار خانم مهندس طیبه مشکیان که در ویرایش کتاب به اینجانب کمک شایانی نمودند تشکر نمایم .

همچون گذشته در انتظار دریافت نقطه نظرات همه خوانندگان محترم در مورد این مجموعه و کاستی های آن هستم. با ارسال آنها به پست الکترونیکی **reza.maher@gmail.com** یا از طریق شرکت صابکو سرافرازم بفرمایید.

**محمدرضا ماهر**

**مرداد ۸۵**

# فهرست مطالب

صفحه


**۱۲ – فانکشنهای استاندارد**

**۱۲-۱ مقدمه** ۲

**۱۲-۲ فانکشن های سیستم** ۴

۱۲-۲-۱ فانکشنهای Bit Logic ۷

۱۲-۲-۲ فانکشنهای Clock ۹

۱۲-۲-۳ فانکشن های DB_FUNCT ۲۰

۱۲-۲-۴ فانکشن های مربوط به تایمرها و کانتر های IEC ۲۵

۱۲-۲-۵ فانکشن های فعال و غیر فعال سازی وقفه ها ۲۹

۱۲-۲-۶ فانکشن های فیلتر کردن وقفه های سنکرون ۳۱

۱۲-۲-۷ فانکشن های مربوط به وقفه های Time of Day ۳۴

۱۲-۲-۸ فانکشن های مربوط به وقفه های تاخیری Delay Time ۳۹

۱۲-۲-۹ فانکشن های کنترل برنامه PGM_CNTL ۴۰

۱۲-۲-۱۰ فانکشن های کپی کردن محتویات حافظه MOVE ۴۱

۱۲-۲-۱۱ فانکشن های مربوط به آدرس مدول ها ۴۳

۱۲-۲-۱۲ فانکشن های Diagnostic Buffer ۴۵

۱۲-۲-۱۳ فانکشنهای مربوط به CPU های 31x IFM ۴۷

۱۲-۲-۱۴ فانکشن های مربوط به CPU های 31xC ۷۲

**۱۲-۳ فانکشن های IEC** ۱۰۷

۱۲-۳-۱ فانکشنهای تبدیل IEC ۱۰۸

۱۲-۳-۲ فانکشنهای خانواده String از IEC ۱۱۲

۱۲-۳-۳ فانکشنهای خانواده Floating Point از IEC ۱۱۳

۱۲-۳-۴ فانکشنهای خانواده Date and Time از IEC ۱۱۴

**۱۳- برنامه نویسی توسط S7-SCL**

**۱۳-۱ مقدمه** ۱۱۶

**۱۳-۲ آشنایی با محیط نرم افزار S7-SCL** ۱۱۹

**۱۳-۳ شروع کار با S7-SCL** ۱۲۲

**۱۳-٤ ساختار یک برنامه S7-SCL** ۱۲۶

**۱۳-۵ نحوه تعریف بلاک ها** ۱۳۲

**۱۳-٦ نحوه تعریف ثوابت و Label** ۱۳۶

**۱۳-۷ نحوه بکار بردن متغیرهای حافظه CPU** ۱۳۷

**۱۳-۸ Operation و Experession در SCL** ۱۳۹

**۱۳- ۹ دستورات SCL** ۱۴۲

۱۳-۹-۱ دستورات اختصاص مقادیر Value Assignment ۱۴۲

۱۳-۹-۲ دستورات کنترلی ۱۴۸

۱۳-۹-۳ دستورات صدا زدن FB و FC ۱۵۷

۱۳-۹-۴ دستورات کانترها ۱۶۰

۱۳-۹-۵ دستورات تایمرها ۱۶۲

۱۳-۹-۶ فانکشن های استاندارد SCL ۱۶۴

**۱۳-۱۰ مثال هایی از برنامه نویسی SCL** ۱۶۹

**۱۳-۱۱ امکانات Debug در SCL** ۱۹۰

**۱٤-برنامه نویسی توسط S7-Graph**

**۱۴-۱ مقدمه** ۱۹۴

**۱۴-۲ آشنایی با محیط نرم افزار S7-Graph** ۱۹۶

**۱۴-۳ شروع کار با S7-Graph** ۱۹۹

**۱۴-٤ برنامه نویسی در S7-Graph** ۲۱۰

۱۴-۴-۱ برنامه نویسی Action ۲۱۰

۱۴-۴-۲ برنامه نویسی Condinion ۲۲۵

**۱۴-۵ پارمترهای FB در S7_Graph** ۲۲۹

**۱۴-٦ ارائه چند مثال با S7-Graph** ۲۳۹

**۱۴-۷ لیست دستورات در S7-Graph** ۲۵۰

**۱۵ - افزونگی نرم افزاری Software Redundancy**

**۱۵-۱ مقدمه** ۲۶۰

**۱۵-۲ عملکرد کلی در افزونگی نرم افزاری** ۲۶۲

**۱۵-۳ اصول کاربرد افزونگی نرم افزاری** ۲۶۲

**۱۵-٤ بلاک های ویژه** ۲۶۵

**۱۵-۵ نکات مهم در افزونگی نرم افزاری** ۲۷۴

**۱۵-٦ جابجایی بین سیستم و اجزای آن** ۲۷۵

**۱۵-۷ شبکه های قابل استفاده جهت اتصال دو سیستم اصلی و پشتیبان** ۲۷۹

**۱۵-۸ تغییر پیکربندی یا برنامه کاربردی در مد RUN** ۲۸۰

**۱۵-۹ مدول های مناسب جهت استفاده در یک سیستم مبتنی بر افزونگی نرم افزاری** ۲۸۱

**۱۵-۱۰ ارتباط با سایر سیستم ها** ۲۸۳

**۱۵-۱۱ جایگاه وضعیت Standby در افزونگی نرم افزاری** ۲۸۷

۱۵-۱۲ استفاده از OB های مربوط به مدیریت خطا ۲۸۸
۱۵-۱۳ مثالی از پیاده سازی افزونگی نرم افزاری با استفاده از S7-300 ۲۸۹

۱٦ - افزونگی سخت افزاری Hardware Redundancy
۱٦-۱ سیستم H در یک نگاه ۲۹۸
۱٦-۲ I/O Redundancy چیست ؟ ۲۹۹
۱٦-۳ اجزای سخت افزاری سیستم H ۳۰۳
۱٦-٤ روش نصب یک سیستم نمونه ۳۱۶
۱٦-٥ نحوه پیکر بندی در Step7 ۳۱۸
۱٦-٦ نحوه عملکرد سیستم های S7-400H ۳۲۲
۱٦-۷ تغییر در سخت افزار سیستم های H در حین کار ۳۳۶
۱٦-۸ MTBF در سیستم های H ۳۴۸

۱۷ - سیستم های ایمن و مقاوم در برابر خطا
Fail Safe & Fault Tolerant Systems
۱۷-۱ مقدمه ۳۵۶
۱۷-۲ کاربرد سیستم های F و FH ۳۵۷
۱۷-۳ انواع ترکیب بندی ۳۵۹
۱۷-٤ اجزای یک سیستم S7-400F ۳۶۱
۱۷-٥ کار با سیستم های F ۳۶۴
۱۷-٦ پیکربندی سیستم F ۳۶۶
۱۷-۷ راه اندازی و نظارت بر خطاها در سیستم F ۳۷۱
۱۷-۸ پیکر بندی سیستم FH ۳۷۲
۱۷-۹ راه اندازی و نظارت بر خطاها در سیستم FH ۳۷۴
۱۷-۱۰ مکانیزم های ایمنی ۳۷۵

**۱۷-۱۱ واکنش در برابر خطا** ۳۷۷

**۱۷-۱۲ مدهای کاری یک سیستم S7-400F/FH** ۳۷۷

**۱۷-۱۳ نظارت منطقی یا زمانی بر اجرای برنامه** ۳۷۸

**۱۷-۱٤ ارتباطات ایمنی** ۳۸۱

**۱۷-۱۵ نکات پیکربندی سخت افزار و تنظیم پارامترها** ۳۸۴

## ضمیمه ۷ – برنامه های کاربردی با CPU های Compact

**برنامه ۱** : کاربرد CPU312 IFM همراه با SFB29 برای شمارش در فرآیند بطری پر کنی ۳۹۰

**برنامه ۲** : کاربرد CPU312 IFM همراه با SFB29 برای شمارش (تکمیل برنامه ۱ ) ۳۹۴

**برنامه ۳** : کاربرد CPU312 IFM همراه با SFB29 برای شمارش (تکمیل برنامه ۱ و ۲ ) ۳۹۷

**برنامه ٤** : کاربرد CPU312 IFM همراه با SFB30 برای اندازه گیری سرعت شافت ۴۰۱

**برنامه ۵** :کاربرد CPU314 IFM همراه با SFB39 برای کنترل موقعیت در سیستمی با تغذیه کنتاکتوری ۴۰۴

**برنامه ٦** :کاربرد CPU314 IFM همراه با SFB39 برای کنترل برش در سیستم مبتنی بر درایو ۴۱۱

**برنامه ۷**: انجام Positioning با خروجی آنالوگ CPU31xC و توسط SFB44 ۴۱۵

**برنامه ۸** : انجام Positioning با خروجی های دیجیتال CPU31xC و توسط SFB46 ۴۲۹

**برنامه ۹** : انجام عملیات شمارش با CPU31xC و توسط SFB47 ۴۴۳

**برنامه ۱۰** : اندازه گیری سرعت شافت با CPU31xC و توسط SFB48 ۴۵۰

**برنامه ۱۱**: استفاده از PWM برای کنترل دما با CPU31xC و توسط SFB49 ۴۵۷

## ضمیمه ۸ – نحوه تبدیل برنامه S5 به S7 ۴۶۲

## ضمیمه ۹ – نحوه ایجاد STL Source ۴۶۸

## منابع و مراجع ۴۷۳

# فصل دوازدهم - فانکشن های استاندارد

**مشتمل بر :**


**۱۲-۱ مقدمه**

**۱۲-۲ فانکشن های سیستم**

۱۲-۲-۱ فانکشنهای Bit Logic

۱۲-۲-۲ فانکشنهای Clock

۱۲-۲-۳ فانکشن های DB_FUNCT

۱۲-۲-۴ فانکشن های مربوط به تایمرها و کانتر های IEC

۱۲-۲-۵ فانکشن های فعال و غیر فعال سازی وقفه ها

۱۲-۲-۶ فانکشن های فیلتر کردن وقفه های سنکرون

۱۲-۲-۷ فانکشن های مربوط به وقفه های Time of Day

۱۲-۲-۸ فانکشن های مربوط به وقفه های تاخیری Delay Time

۱۲-۲-۹ فانکشن های کنترل برنامه PGM_CNTL

۱۲-۲-۱۰ فانکشن های کپی کردن محتویات حافظه MOVE

۱۲-۲-۱۱ فانکشن های مربوط به آدرس مدول ها

۱۲-۲-۱۲ فانکشن های Diagnostic Buffer

۱۲-۲-۱۳ فانکشنهای مربوط به CPU های 31x IFM

۱۲-۲-۱۴ فانکشن های مربوط به CPU های 31xC

**۱۲-۳ فانکشن های IEC**

۱۲-۳-۱ فانکشنهای تبدیل IEC

۱۲-۳-۲ فانکشنهای خانواده String از IEC

۱۲-۳-۳ فانکشنهای خانواده Floating Point از IEC

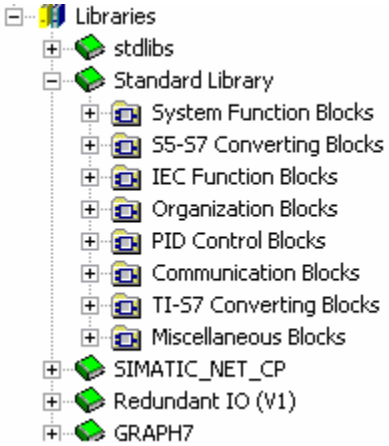
۱۲-۳-۴ فانکشنهای خانواده Date and Time از IEC

## ۱۲-۱ مقدمه

در این فصل هدف آنست که خواننده محترم با فانکشن های استاندارد موجود در برنامه Step7 و نحوه استفاده از آنها در برنامه آشنا شود. منظور از فانکشن های استاندارد فانکشن ها و فانکشن بلاک هایی هستند که در Library نرم افزارStep7 موجود می باشند. این فانکشنها همانطور که در شکل زیر نشان داده شده در زیر مجموعه Standard Libraries در پنجره Program Element از برنامه LAD/STL/FBD ظاهر می شوند.

این فانکشن ها قادر هستند برخی عملیات پیچیده که کاربر در منطق کنترلی سیستم به آنها نیاز پیدا می کند را انجام دهند. اگر چه امکان نوشتن برنامه کنترلی این عملیات توسط کاربر نیز وجود دارد ولی این کار در صورت وجود فانکشن آماده شده از قبل چندان ضرورتی پیدا نمی کند. به هر حال مواردی هم هست که طراح برنامه ترجیح می دهد برای عملیات پیچیده مورد نظر که در پروژه های مختلف تکرار می شود شخصاً فانکشنی را بنویسد و بکار ببرد و از آنجا که دانش فنی آن به خودش اختصاص دارد میتواند آنرا Protect نماید.

صدا زدن فانکشنهای استاندارد هیچ تفاوتی با صدا زدن فانکشنهای معمولی که توسط کاربر نوشته شده ندارد بدیهی است FB ها و SFB ها لازم است همراه بادیتا بلاک (DB) فراخوان شوند .




این فانکشنها Protect بوده و کاربر نمیتواند محتویات برنامه آنها را مشاهده نماید . با مشاهده آیکون آنها در پوشه Block از برنامه Simatic Manager میتوان دید که حفاظت شده می باشند.

در شکل روبرو در زیر مجموعه Standandard Library خانواده های مختلفی از فانکشن های استاندارد لیست شده اند ولی در این فصل فقط دو خانواده از آنها که بیشتر بکار می روند مورد بحث قرار میگیرند یعنی :

- System Function Blocks
- IEC Function Blocks

این دو خانواده بطور تفصیلی مورد بررسی قرار خواهند گرفت . سعی شده مثالهای کوتاه در خلال بحث ذکر شوند ولی نمونه های کاربردی که برنامه آنها مفصل بوده در انتهای کتاب (ضمیمه ۷ ) آورده شده اند.

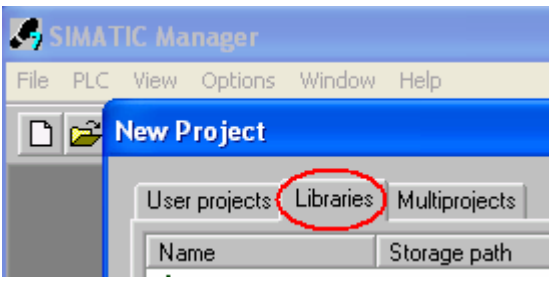
در ارتباط با سایر خانواده های موجود در Standard Library لازم است توجه شود که:

- **S5-S7 Converting Blocks** : این فانکشنها برای تبدیل برنامه S5 به S7 مورد استفاده قرار می گیرند . غالباً نیازی به استفاده مستقیم از آنها نیست در عین حال نحوه تبدیل برنامه S5 به S7 در ضمیمه ۸ آمده است.
- **Organization Blocks** : لیست تمام OB ها که برای مقاصد مختلف بکار می روند در این قسمت آمده است . نحوه استفاده از بسیاری از این OB ها در جلد اول کتاب توضیح داده شد .
- **PID Control Blocks**: در جلد اول کتاب بلاک های اصلی این قسمت تشریح شده اند.
- **TI-S7 Converting Blocks** : این بلاک ها نیز همانطور که از اسمشان پیداست برای تبدیل برنامه PLC های TI (مانند سری 505) به S7 بکار می روند و خارج از بحث این کتاب می باشند.

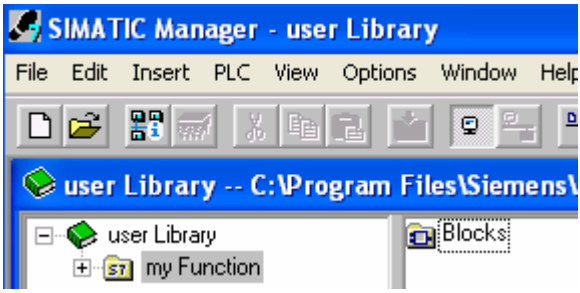
نکته ای که در همین جا لازم است به آن اشاره شود ایجاد Library شخصی است . کاربر در صورت لزوم میتواند Library جدیدی را ایجاد کرده و FC و FB های دلخواهی را به آن اضافه نماید. این کار برای فانکشنهایی که در برنامه های مختلف مورد استفاده قرار می گیرند مفید است . در عین حال به روشی که در فصل های بعدی توضیح خواهیم داد کاربر میتواند در صورت لزوم بلاک های FC و FB خود را protect نموده و Source برنامه را از دسترس دیگر کاربران خارج نماید.

نحوه ایجاد Library جدید توسط کاربر بصورت زیر است:

۱- در برنامه Simatic Manager از منوی **File > New** استفاده کرده ودر پنجره ای که مانند شکل زیر باز می شود قسمت Library را انتخاب کرده سپس اسم دلخواهی به آن اختصاص می دهیم. مشاهده می کنیم که پنجره Library با اسم مورد نظر ظاهر می گردد.



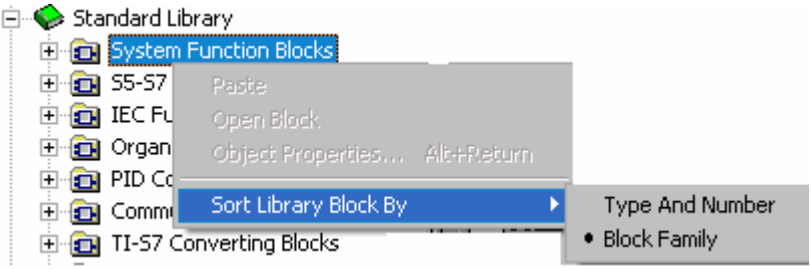
۲- با استفاده از منوی **Insert > Program > S7 Program** به پنجره مزبور S7-Program را اضافه می کنیم. سپس آنرا باز کرده و FC یا FB های مورد نظر را که قبلاً کپی کرده ایم در آن Paste می نماییم.

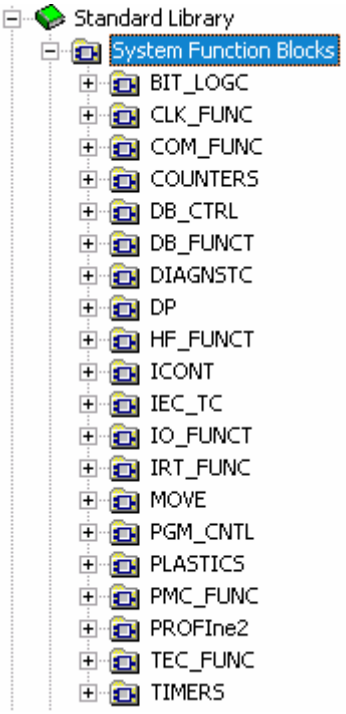



۳-یکبار Simatic Manager را بسته و مجدداً باز می کنیم. اکنون اگر برنامه LAD/STL/FBD را باز کنیم Library مزبور را همراه با بلاکهای داخل آن در پنجره Program Element مشاهده خواهیم کرد. بدینطریق قادر خواهیم شد بلاک مورد نظر را در هر پروژه Step7 که روی این کامپیوتر ایجاد شود استفاده نماییم.

## ۱۲-۲ فانکشن های سیستم

System Functions متشکل از تعدادی SFB و SFC است که برای مقاصد مختلف تعبیه شده اند.به دو روش میتوان لیست آنها را مشاهده نمود یکی بترتیب بر اساس شماره آنها و دیگری بصورت دسته بندی شده براساس عملکرد آنها. بصورت پیش فرض لیست بصورت روش اول ظاهر می گردد ولی با راست کلیک روی System Function Blocks و انتخاب روش مرتب سازی براساس Block Family میتوان آنها را به صورت دسته بندی شده مشاهده نمود.





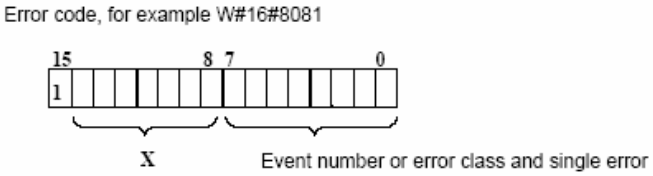

در این فصل تشریح این بلاک ها براساس عملکرد آنها صورت می گیرد. با مرتب سازی به این شیوه همانطور که در شکل روبرو مشاهده می شود دسته های مختلفی از بلاک ها را خواهیم داشت. بجز مواردی که مرتبط با بحث شبکه هستند اهم سایر بلاک ها در صفحات آینده تشریح شده اند.

در تشریح بلاک ها معادل LAD آنها نمایش داده شده ، ورودی و خروجی و عملکرد آنها تشریح شده و غالباً مثالی نیز ارائه گردیده است.

همانطور که خواهیم دید تمام فانکشن های سیستم دارای یک خروجی به نام Ret_Val مخفف Return Value هستند. این خروجی کدی را بر می گرداند که بر اساس آن می توان فهمید فانکشن بدرستی اجرا شده یا خیر واگر با خطا مواجه شده نوع خطا چه بوده است. قبل از تشریح فانکشن ها ابتدا کدهای خطای عمومی آنها را توضیح می دهیم.

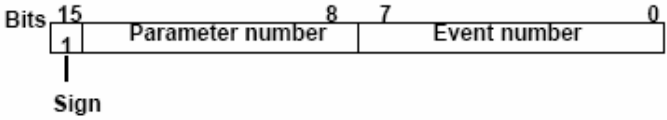
## کدهای خطای عمومی در فانکشنهای SFC

اجرای SFC توسط CPU ممکن است موفقیت آمیز نباشد. در اینحالت به دو طریق میتوان از بروز خطا مطلع شد اول با چک کردن بیت BR از Status Word ، دوم با چک کردن خروجی Ret_VAL فانکشن.وقتی اشکالی وجود ندارد BR=1 و Ret_Val مقداری مثبت است ولی در صورت وقوع خطا BR=0 و Ret_Val مقدار منفی دارد. Ret_VAL دو نوع کد خطا را برمی گرداند یکی کدهای خطای خاص که مربوط به همان SFC خاص است دیگری کدهای خطای عمومی که برای تمامی SFC ها قابل استفاده است. در اینجا صرفاً کد های عمومی خطا معرفی می شود. مقدار Ret_Val در خروجی فانکشن بصورت یک کد هگز ۱۶ بیتی ظاهر می شود که میتوان آن را در یک متغیر Word ریخت. ساختار این کد بصورت شکل زیر است:




همانطورکه در شکل مشاهده می شود بیت 15 همواره یک است و مقدارمنتجه کد خطا منفی است در کنار آن

هشت بیت که با X نمایش داده شده وجود دارد اگر این ۸ بیت صفر باشد نشان دهنده خطای خاص است و اگر بزرگتر از صفر باشد نمایشگر خطای عمومی است.کد خطای عمومی ساختاری شبیه شکل زیر دارد که در آن شماره پارامتر عددی بین 1 تا 111 و شماره Event عددی بین 0 تا 127 است.




با توجه به توضیحات فوق کدهای خطای عمومی بصورت جدول زیر خواهند بود:

| Code (W#16#..) | توضیح |
|---|---|
| 8x7F | خطای داخلی (internal) را نشان می دهد. نمیتوان با تغییر در برنامه از بروز آن جلوگیری کرد. |
| 8x01 | اشکال در فرمت پارمتر از نوع Any |
| 8x22 | خطای خواندن (reading) در هنگام استفاده از پارامتر Any بعلت نادرست بودن رنج آدرس مثلاً در نوع بیت مضربی از ۸ نباشد مانند P#M0.0 BOOL 7 |
| 8x23 | شبیه کد 8x22 ولی برای حالت نوشتن (writing) |
| 8x24 | پارامتر x در هنگام خواندن برای فانکشن نا شناخته بوده است. |
| 8x25 | پارامتر x در هنگام نوشتن برای فانکشن نا شناخته بوده است. |
| 8x26 | شماره تایمر بیش از حد مجاز است |
| 8x27 | شماره کانتر بیش از حد مجاز است |
| 8x28 | پارامتر x که در آدرس بصورت بیتی اشاره شده در هنگام خواندن بصورت 0 نبوده است |
| 8x29 | پارامتر x که در آدرس بصورت بیتی اشاره شده در هنگام نوشتن بصورت 0 نبوده است |
| 8x30 | پارامتر x در یک DB Global که بصورت Read only است قرار گرفته است. |
| 8x31 | پارامتر x در یک DB Instance که بصورت Read only است قرار گرفته است. |
| 8x32 | شماره DB بیش از حد مجاز است. |
| 8x34 | شماره FC بیش از حد مجاز است. |
| 8x35 | شماره FB بیش از حد مجاز است. |
| 8x3A | DB مورد نظر Load نشده است. |
| 8x3C | FC مورد نظر Load نشده است. |
| 8x3E | FB مورد نظر Load نشده است. |
| 8x42 | خطای دسترسی به ناحیه Peripheral Input در هنگام خواندن |
| 8x43 | خطای دسترسی به ناحیه Peripheral Output در هنگام نوشتن |
| 8x44 | پارامتر x نشان می دهد که خطای دسترسی برای خواندن در چندمین نوبت رخ داده است. |
| 8x45 | پارامتر x نشان می دهد که خطای دسترسی برای نوشتن در چندمین نوبت رخ داده است. |

با این مقدمه اکنون به شرح زیر مجموعه فانکشن های سیستم می پردازیم. مجموعه هایی از این خانواده که در اینجا مورد بحث قرار میگیرند عبارتند از :

۱- فانکشن های Bit Logic : برای Set کردن و Reset کردن بیت های خروجی بصورت یک جا

۲- فانکشن های Clock : برای خواندن زمان و تاریخ CPU و تنظیم آن و نیز استفاده از زمان سنج های CPU

۳- فانکشن های DB_FUNCT : برای ایجاد و حذف دیتا بلاک در حافظه و تنظیم ویژگی های آن

۴- فانکشن های مربوط به تایمرها و کانتر های IEC : برای استفاده برای زمانهای بزرگ یا شمارش های بالا

۵- فانکشن های فعال و غیر فعال سازی وقفه ها : برای فعال و غیر فعال سازی انواع وقفه توسط برنامه نویسی

۶- فانکشن های فیلتر کردن وقفه های سنکرون: برای فیلتر کردن خطاهای برنامه نویسی یا دسترسی بهI/O

۷- فانکشن های مربوط به وقفه های Time of Day : برای تنظیم،فعال سازی و غیر فعال سازی وقفهTOD

۸- فانکشن های مربوط به وقفه های تاخیری Delay Time : برای استفاده از این وقفه ها با برنامه نویسی

۹- فانکشن های کنترل برنامه PGM_CNTL : برای انجام عملیاتی مانند توقف یا تاخیر در پردازشCPU

۱۰- فانکشن های کپی کردن محتویات حافظه MOVE : برای انجام کپی حجم زیادی از حافظه

۱۱- فانکشن های مربوط به آدرس مدول ها : برای تبدیل آدرس جغرافیایی کارت ها به آدرس های منطقی یا برعکس

۱۲- فانکشن های Diagnostic Buffer: برای نوشتن پیغام دلخواه در بافر Diagnostic مربوط به CPU

۱۳- فانکشنهای مربوط به CPU های IFM :برای شمارش سریع ،اندازه گیری فرکانس پالس و کنترل موقعیت

۱۴- فانکشن های Tec_Func مربوط به CPU های 31xC: برای شمارش سریع ، اندازه گیری فرکانس پالس، PWM ، کنترل موقعیت و کنترل PID

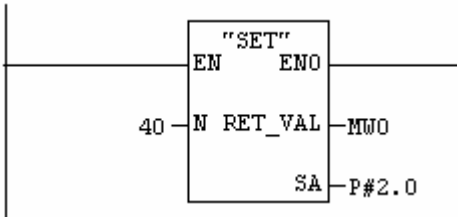
بجز موارد ۱۳ و ۱۴ که کاربرد خاص دارند سایر موارد فانکشنهای عمومی سیستم هستند و در تمام CPU ها قابل استفاده هستند. موارد ۱۳ و ۱۴ علاوه بر ارائه توانایی های CPU های خاص میتواند این دید را به خواننده محترم ارائه کند که چگونه با کارت های FM که برای اموری چون شمارش سریع (counting) و کنترل موقعیت (positioning) بکار میروند و در جلد اول کتاب معرفی شدند کار کند.

### ۱۲-۲-۱ فانکشنهای Bit Logic

این فانکشن ها همانطور که از نامشان مشخص است برای انجام عملیات روی بیت بکار می روند. منظور از بیت در اینجا بیت آدرس خروجی است دو SFC با شماره های SFC79 و SFC80 برای این منظور ارائه شده اند که می توان توسط اولی خروجی ها را Set و توسط دومی خروجی ها را Reset نمود.

**SFC79 با نام سمبلیک "SET"**

توسط این فانکشن میتوان تعداد بیت دلخواهی از یک **آدرس خروجی** را Set کرد. این آدرس خروجی میتواندآدرسهای مربوط Process Image که توسط Q یا QB یا QW یا QD آدرس دهی می شوند یا آدرس های مربوط Peripheral I/O که توسط PQB یا PQW یا PQD آدرس دهی می شوند باشد. شکل بلاک LAD این فانکشن بصورت روبروست.




**ورودی و خروجی های فانکشن:**

**ورودی N** : تعداد بیت خروجی که قرار است Set شوند بصورت عدد Integer. در مثال شکل فوق ۴۰ بیت خروجی که از آدرس Q2.0 شروع می شوند یک می شوند.

**خروجیRet-Val** : خروجی نمایش خطا بصورت Word که توضیحات آن در صفحات قبل ذکر گردید.

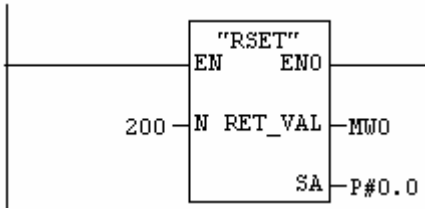
**خروجی SA** : آدرس خروجی که قرار است عمل Set روی آن انجام شود بـصورت Pointer . بعبـارت دیگـر فقط نقطه شروع مشخص می گردد.

نکاتی که در استفاده از این فانکشن لازم است به آنها توجه شود:

- اینکه آدرس خروجی Q یا PQ است در SFC79 مشخص نمی شود چون در این فانکشن صرفاً به آدرس بصورت Pointer اشاره می گردد ولی از آنجا که از دیدگاه CPU آدرس های خروجی منحصر به فرد هستند مثلاً آدرس QB200 و PQB200 یکی محسوب می شوند ازاینرو Set شدن خروجی صرفاً با توجه به آدرس اتفاق می افتد .
- این فانکشن میتواند با یکبار فراخوانی تعداد زیادی بیت (بیش از ۳۲ بیت ) را Set کند در حالیکه اگر با دستورات معمول برنامه نویسی بخواهیم این کار را انجام دهیم حداکثر ۳۲ بیت را با یک دستور Transfer میتوان Set نمود.

**SFC80 با نام سمبلیک "RSET"**

این فانکشن عکس SFC79 را انجام می دهد یعنی برای ری ست کردن بیتهای **آدرسهای خروجی** بکار میرود. از آنجا که ورودی و خروجی و توضیحات آن مشابه SFC79 است از تکرار آنها خودداری می کنیم.

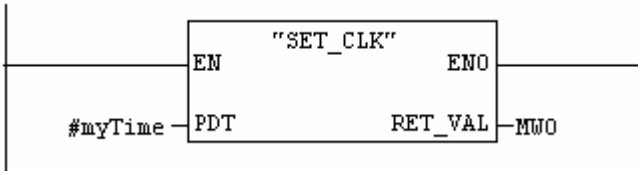


### ۱۲-۲-۲ فانکشنهای Clock

این فانکشن ها مرتبط با تاریخ و زمان CPU و زمان سنج های داخل آن هستند. توسط این فانکشن ها می توان زمان و تاریخ CPU را خواند یا در صورت لزوم آنها را تنظیم کرد بعلاوه می توان از زمان سنج های داخل CPU برای ثبت زمان کارکرد تجهیزات استفاده نمود. این فانکشن ها در اینجا بترتیب تشریح می گردند.

#### SFC0 با نام سمبلیک SET_CLK

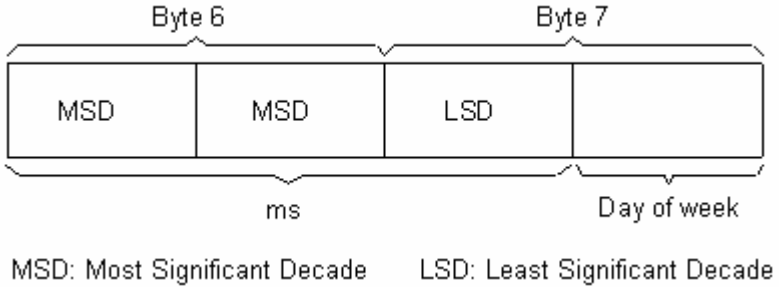
با استفاده از این فانکشن میتوان تاریخ و زمان CPU را تنظیم کرد .**ورودی PDT** تاریخ وزمان را به فرمت DT ( یعنی Date_and_Time ) می گیرد و **خروجی Ret_Val** برای نمایش Error بصورت Word می باشد.



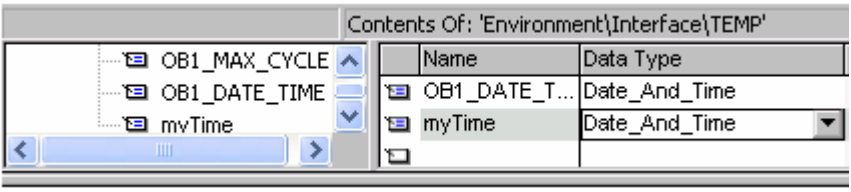

توجه شود که زمان و تاریخ را نمیتوان مستقیما به ورودی PDT داد . DT حجم حافظه ای بیش از ۳۲ بیت را اشغال میکند و دارای فرمتYr-Mo-Day-hr:min:s:ms است . مثال : DT#2005-02-02-08:30:01.999

Date_and_Time بصورت BCD و ۶۴ بیت یا ۸ بایت را مطابق جدول و شکل زیر اشغال می کند:

| Byte | Contents | Range |
|---|---|---|
| 0 | Year | 1990 to 2089 |
| 1 | Month | 01 to 12 |
| 2 | Day | 1 to 31 |
| 3 | Hour | 0 to 23 |
| 4 | Minute | 0 to 59 |
| 5 | Second | 0 to 59 |
| 6 | 2 MSD of ms | 00 to 99 |
| 7 ( 4 MSB ) | LSD of ms | 0 to 9 |
| 7 ( 4 LSB ) | Day of week | 1 to 7 (1 = Sunday ) |



با توجه به این توضیحات از آنجا که طول Date_and_Time بیش از ۳۲ بیت است نمیتوان آنرا در متغیرهای معمول حتی از نوع Double Word بار کرد . یک روش آنست که آنرا در یک متغیر Temp ازنوع Date_and_Time که در جدول Declaration بالای بلاک قابل تعریف است بریزیم سپس از آن استفاده کنیم. در شکل زیر متغیر myTime در بالای بلاک برای این منظور تعریف شده است:




پس از تعریف متغیر فوق در بالای بلاک میتوان آنرا به ورودی PDT فانکشن SFC0 اعمال کرد. ولی هنوز یک مشکل دیگر حل نشده باقی مانده است و آن اینکه چگونه این متغیر را با تاریخ و زمان مورد نظر پر کنیم؟ این مشکل به دو روش قابل حل است:

**روش اول**: با بکار بردن فانکشن های IEC که در بخش ۱۲-۳ مورد بحث قرار گرفته اند. در اینجا همینقدر اشاره می کنیم که توسط فانکشن FC34 از خانواده فانکشن های IEC میتوان زمان و تاریخ را با هم ترکیب کرد و در متغیر Temp از جنس Date_and_Time ریخت. سپس خروجی فانکشن مزبور را برای ورودی PDTفانکشن SFC0 استفاده کرد. در بخش ۱۲-۳ کتاب مراحل فوق با ذکر مثال تشریح شده است .

**روش دوم**: با استفاده از آدرس دهی غیر مستقیم از طریق Adress Register . در این روش یک متغیر Temp از جنس Date_and_time در بالای بلاک تعریف می کنیم و آنرا به ورودی PDT بلاک معرفی می نماییم سپس بایتهای این متغیر را طبق جدول صفحه قبل جداگانه با مقادیر سال و روز و ماه و ... پر می کنیم.

بعنوان مثال فرض کنید که میخواهیم تاریخ CPU را به سال 2009 ماه 7 و روز 25 تغییر دهیم. متغیر myTime را از جنس Date_and_time تعریف کرده سپس برنامه زیر را می نویسیم :

```
LAR1  P##myTime
L     B#16#9
T     B [AR1,P#0.0]

L     B#16#7
T     B [AR1,P#1.0]

L     B#16#25
T     B [AR1,P#2.0]
 CALL  "SET_CLK"
    PDT    :=#myTime
    RET_VAL:=MW0
```

و اگر فرضاً بخواهیم زمان را نیز به ساعت 18:40:20.0 تغییر دهیم کافیست چند سطر زیر را به برنامه فوق اضافه کنیم:

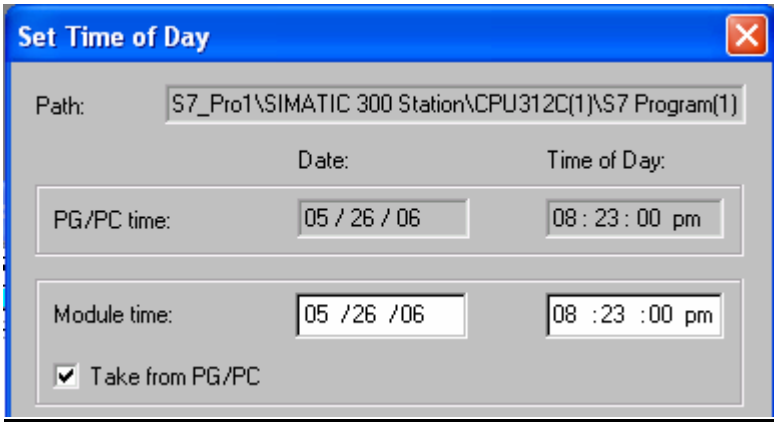
```
L    B#16#18
T    B [AR1,P#3.0]

L    B#16#40
T    B [AR1,P#4.0]

L    B#16#20
T    B [AR1,P#5.0]
```

نکاتی که در استفاده از SFC0 لازم است مد نظر قرار گیرد:

- استفاده ازاین فانکشن در OB1 موجب میشود مرتبا زمان و تاریخ CPU به مقدار فوق عوض شود که روش درستی نیست . از این فانکشن باید بصورت کنترل شده استفاده کرد مثلا در OB راه اندازی یا در صورت وقوع Event خاص.
- به هر حال پس از اجرای برنامه فوق در حالی که به PLC متصل هستیم نتیجه را می توان توسط مسیر زیر در Simatic Manager مشاهده کرد :

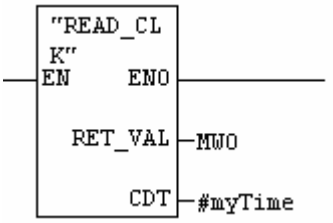
**PLC > Diagnostic/Setting > Set Time of Day**

پنجره ای که باز میشود در شکل زیر نشان داده شده است . بطور مشابه این پنجره دربرنامه های Hwconfig و LAD/STL/FBD نیز از منوی **PLC>Set Time Of Day** قابل مشاهده است.



**SFC1 بانام سمبلیک READ_CLk**

با استفاده از این فانکشن میتوان تاریخ و زمان CPU را خواند . خروجی های این فانکشن همانطور که در شکل روبرو نشان داده شده عبارتند از :




**خروجی CDT**: این خروجی به فرمت #DT است ، یعنی ترکیب تاریخ وزمان (Current Date and Time)

**خروجیRet_Val** : خروجی برای Error ها

نکاتی که در مورد PDT برای SFC0 گفته شد در اینجا برای خروجی CDT نیز صادق است . بدلیل اینکه خواننده محترم ممکن است تا اینجا هنوز با فانکشن های IEC آشنا نباشد ذکر مثال برای این فانکشن را به بخش۱۲-۳ موکول می کنیم و صرفاً مثال را برای روش دوم ذکر می نماییم. در برنامه زیر زمان CPU خوانده شده و در متغیر محلی myTime که از جنس Date_and _Time است ریخته می شود سپس توسط Address Register بایتهای مختلف آن که معرف سال ، ماه ، روز ، ساعت، دقیقه و ثانیه است جدا شده و به متغیرهایی از جنس بایت انتقال داده می شود.

```
CALL  "READ_CLK"
    RET_VAL:=MW100
    CDT    :=#myTime

LAR1   P##myTime
L      B[AR1 , P#0.0]
T      MB0
L      B[AR1 , P#1.0]
T      MB1
L      B[AR1 , P#2.0]
T      MB2
L      B[AR1 , P#3.0]
T      MB3
L      B[AR1 , P#4.0]
T      MB4
L      B[AR1 , P#5.0]
T      MB5
```

اگر بخواهیم میلی ثانیه را نیز داشته باشیم همانطور که در صفحه ۹ ذکر شد میلی ثانیه در ۱۲ بیت سمت چپ از word ششم قرار می گیرد پس ابتدا آنرا با مقدار W#16#FFF0 بیت به بیت AND کرده سپس در متغیری از جنس Word می ریزیم.

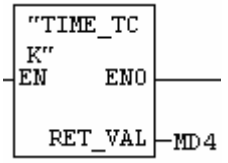
```
L      W#16#FFF0
L      W[AR1 , P#6.0]
AW
T      MW6
```

واگر بخواهیم روز هفته را نیزمشخص کنیم از آنجا که این اطلاعات در چهار بیت سمت راست بایت هفتم وجود دارد (رجوع شود به صفحه ۹) آنرا با مقدار B #16#F بیت به بیت AND کرده سپس در متغیری از جنس بایت می ریزیم.

```
L     B#16#F
L     B[AR1 , P#7.0]
AW
T     MB9
```

**SFC64 با نام سمبلیک TIME-TCK**

با استفاده از فانکشن فوق که به TIME-TCK موسوم است ، می توان زمان CPU را خواند . دقت شود تفاوت آن با SFC1 در اینست که SFC1 تاریخ و زمان را بر می گرداند ولی SFC64 یک زمان سنج است که زمان پردازش CPU را بر حسب میلی ثانیه بر می گرداند.




**خروجی Ret-Val** از جنس TIME می باشد. در CPU زمان سیستم یک شمارنده داخلی است که بطور پریودیک از ۰ تا ماکزیمم ۲۱۴۷۴۸۳۶۴۸ میلی ثانیه می شمارد و سپس مجدداً به صفر برمی گردد. Resolution در S7-400 و CPU 318 یک میلی ثانیه ولی برای سایر اعضای خانواده S7-300 ده میلی ثانیه می باشد . زمان سیستم از مد کاری آن تأثیر می پذیرد بصورت زیر :

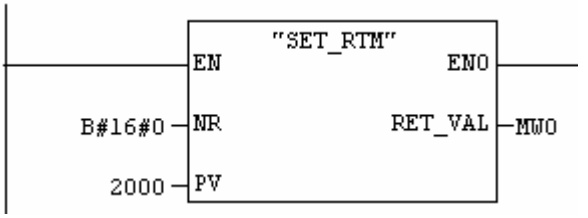
- در مدهای Start up و Run بطور مداوم Update می شود .
- در مد STOP می ایستد و آخرین مقدار را نگه می دارد.
- در راه اندازی Warmو Cold پاک شده واز نو شروع می کند.
- در راه اندازی Hot از مقداری که در لحظه قطع شدن CPU ذخیره شده ادامه می دهد.

**مثال**: برای محاسبه طول مدت پردازش یک Block مثلاً OB1 می توان این فانکشن را در ابتدا وانتهای آن صدا زد و در آخر دو زمان بدست آمده را از یکدیگر کم نمود تا مقدار دقیق زمان استفاده شده بدست آید. طول زمان پردازش Network در MD12 ذخیره می شود که می توان آن را بصورت TIME یا TOD مشاهده نمود.

```
Net work 1 :
        CALL SFC64
                RET-VAL : MD4
Net work 2 :
                Ali: L    30000
                     T    MW0
                     L    ID 0
                     SIN
                     T   QD 0
                     L   MW0
                     Loop ali
Net work 3 :
        CALL SFC64
                RET-VAL : = MD8
Net work 4 :
                     L   MD8
                     L   MD4
                       - D
                     T   MD12
```

**SFC2 با نام سمبلیک SET_RTM**

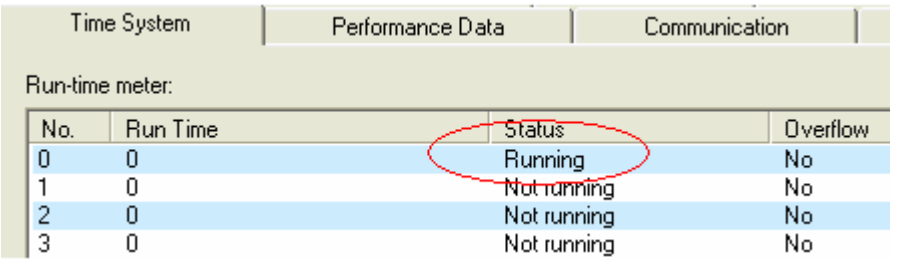
این فانکشن برای Set کردن Run time meter بکار می رود.RTM ها زمان سنج های داخل CPU هستندکه می توانند ساعت کارکرد یک وسیله را در خود ذخیره نمایند. بعنوان مثال فرض کنید وسیله ای مرتباً توسط PLC خاموش و روشن می شود با استفاده از RTM ها میتوان جمع ساعاتی که وسیله روشن بوده را ثبت نمود. این ویژگی کاربرد مهمی در صنایع بویژه از بعد نگهداری و تعمیرات دارد.ماکزیمم هشت RTM که شانزده بیتی هستند می توان استفاده کرد (از شماره ۰ تا شماره ۷) ،بنابراین برای ۸ دستگاه مختلف می توان همزمان این زمان سنجی را انجام داد. این RTM ها چون ۱۶ بیتی هستند حداکثر میتوانند تا ۳۲۷۶۷ ساعت را ثبت کنند.
ثبت ساعات کارکرد تجهیزات از طریق برنامه نویسی معمولی با بکار بردن تایمر نیز امکان پذیر است که محدودیت ۸ وسیله را نیز ندارد ولی استفاده از RTM ها ساده تر است و توصیه می شودکه بدلیل محدودیت تعداد، آنها را برای تجهیزاتی که در درجه اهمیت بالاتری قرار دارند بکار ببرید.




**ورودی NR** : از جنس بایت است که به آن شماره RTM مورد نظر داده می شود مثلا B#16#0 نشان دهنده RTM شماره صفر است همانطور که ذکر شد این شماره میتواند بین 0 تا 7 باشد.

**ورودی PV** : از جنس Integer و مقدار اولیه RTM را مشخص می کند.در مثال فوق RTM شماره ۱ با مقدار اولیه ۲۰۰۰ Set می شود به این معنا که وسیله تاکنون ۲۰۰۰ ساعت کار کرده است.

تعداد و وضعیت RTM ها را می توان توسط پنجره Module Information که از طریق منویPLC در Simatic Manager و سایر زیر برنامه های آن قابل فعال سازی است وسپس در بخش Time System آن مشاهده نمود.




| No. | Run Time | Status | Overflow |
|---|---|---|---|
| 0 | 0 | Running | No |
| 1 | 0 | Not running | No |
| 2 | 0 | Not running | No |
| 3 | 0 | Not running | No |

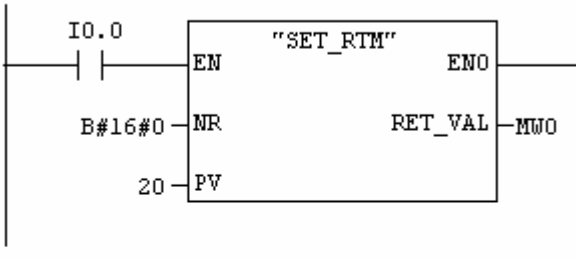
نکته دیگر اینکه استفاده از SFC2 صرفا برای اعمال مقدار اولیه به RTM است و برای روشن کردن RTM باید از SFC3 استفاده نمود. در صورت نیاز به دیدن مقدار RTM نیز از SFC4 استفاده می شود. کاربرد همزمان این فانکشن ها در مثال های بعدی آمده است.

**SFC3 با نام سمبلیک CTRL_RTM**

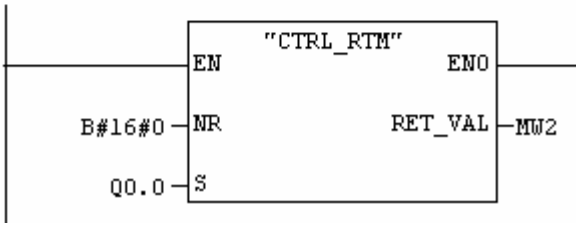
این فانکشن RTM که قبلا با فانکشن SFC2 ست شده و شمارهٔ آن به ورودی NR داده شده را در صورتی که ورودی S یک باشد فعال و در صورتی که این ورودی صفر باشد RTM را غیر فعال می کند.

OB1 : "Main Program Sweep (Cycle)"

**Network 1 :** Title:




**Network 2 :** Title:




در مثال فوق RTM شماره صفر توسط SFC2 با مقدار اولیه ۲۰ تنظیم شده سپس توسط SFC3 براساس وضعیت Q0.0 ساعت کارکرد این خروجی را ثبت مینماید. به نکات زیر باید توجه داشت :

- فانکشن SFC2 فقط برای اعمال مقدار اولیه است بدون وجود آن نیز RTM توسط SFC3 کار میکند .

- فانکشن SFC2 باید بصورت کنترل شده استفاده گردد اگر شرط یا کنتاکتی برای Enable کردن آن بکار نرود یا همیشه فعال بماند در اینصورت RTM اگرچه کار میکند ولی مقدار آن افزایش نمی یابد چون مرتبا با مقدار اولیه PV ست میشود.
- مقدار لحظه ای RTM را صرفا توسط SFC4 میتوان روی خروجی مشاهده یا از آن استفاده نمود.
- فعال بودن RTM را میتوان در قسمت Time پنجره Module Information مانند شکل قبل مشاهده نمود.

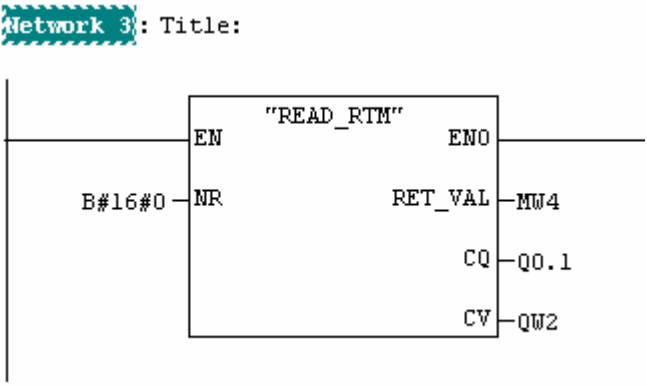
**SFC4 با نام سمبلیک READ_RTM**

برای خواندن وضعیت یک RTM که قبلاً فعال شده وارسال ساعت کارکرد به خروجی مورد نظر ازاین فانکشن استفاده می شود.

**ورودی NR** : شماره RTM به فرمت : شماره#B16

**خروجی CQ** : یک بیت است که نشان می دهد RTM مورد نظر فعال است یا خیر.

**خروجی CV** : یک مقدار Integer است که مقدار فعلی RTM را نشان می دهد.

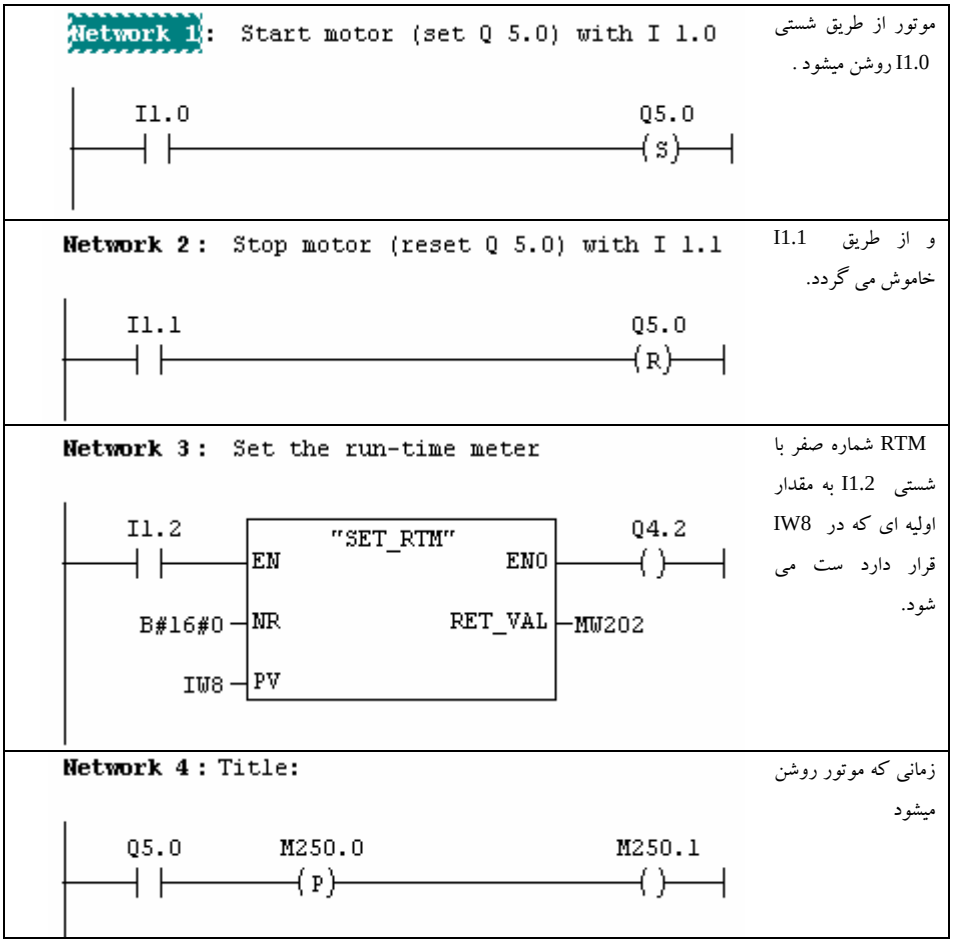



**مثال ۱:**

برنامه قبل را میتوان با اضافه کردن SFC4 مطابق شکل بالا تکمیل کرد . پس از گذشت هر ساعت که Q0.0 کار کند خروجی QW2 یک شماره افزایش می یابد.

**مثال۲:**

موتور Q5.0 از طریق شستی I1.0 روشن و توسط I1.1 خاموش میگردد . می خواهیم ساعات کارکرد موتور را شمارش و ثبت کنیم . برای این منظور از یک RTM استفاده می نماییم که در طول زمان روشن بودن موتور کارکرد آنرا ثبت می کند. نتیجه ساعت ثبت شده بصورت عدد صحیح روی خروجی QW16 نمایش داده میشود.

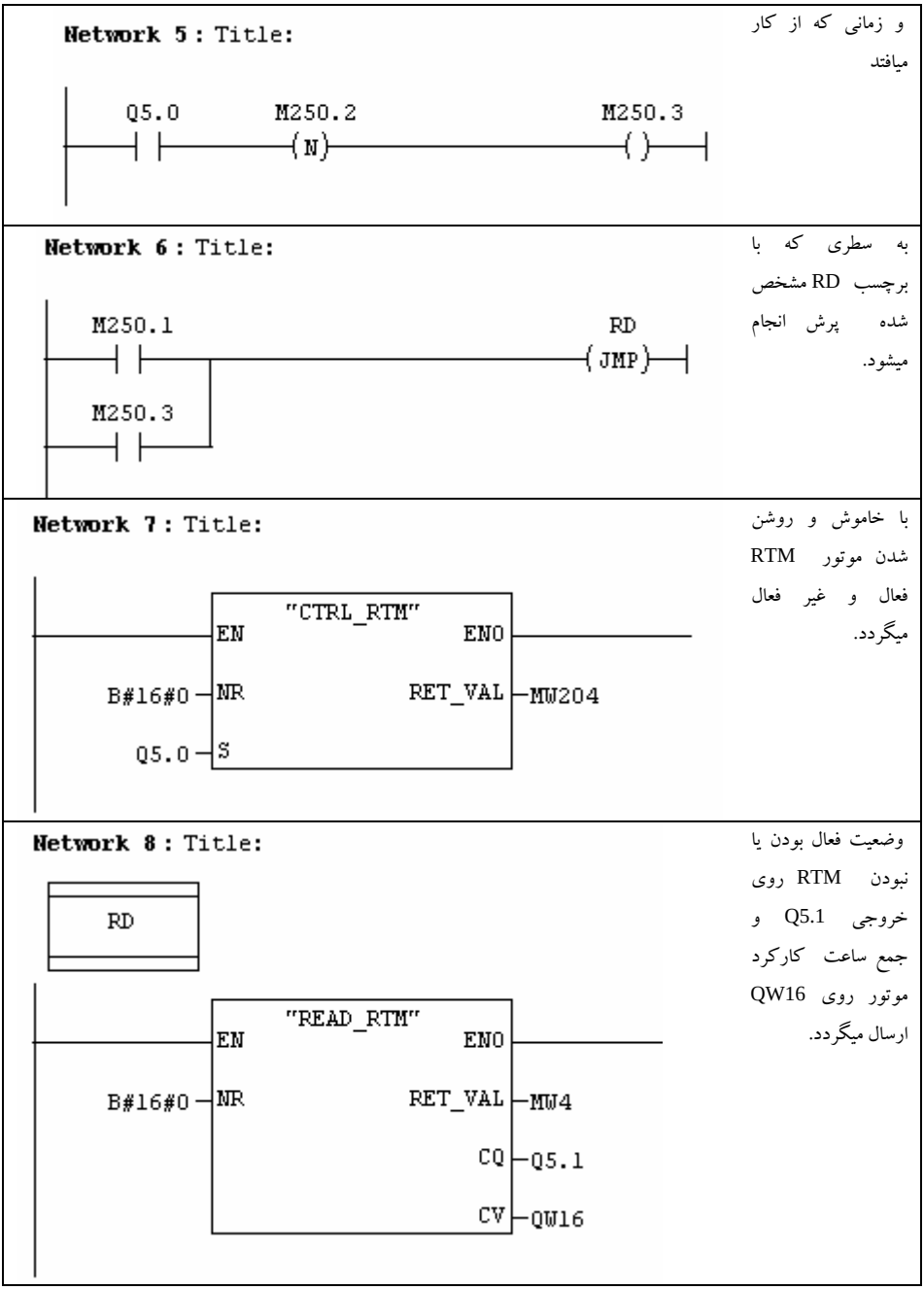



موتور از طریق شستی I1.0 روشن میشود .

و از طریق I1.1 خاموش می گردد.

RTM شماره صفر با شستی I1.2 به مقدار اولیه ای که در IW8 قرار دارد ست می شود.

زمانی که موتور روشن میشود




و زمانی که از کار میافتد

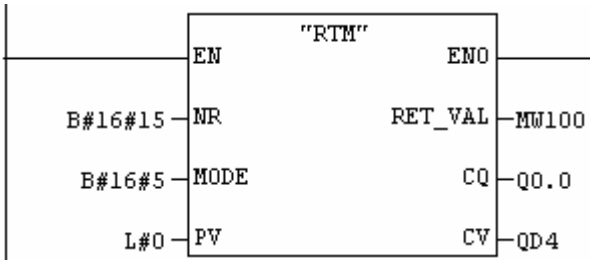
به سطری که با برچسب RD مشخص شده پرش انجام میشود.

با خاموش و روشن شدن موتور RTM فعال و غیر فعال میگردد.

وضعیت فعال بودن یا نبودن RTM روی خروجی Q5.1 و جمع ساعت کارکرد موتور روی QW16 ارسال میگردد.

**SFC101 با نام سمبلیک RTM** ( ۳۲ بیتی )

RTM هایی که با SFC2 و SFC3 و SFC4 کار می کنند ، ۱۶ بیتی هستند، بنابراین فقط می توانند تا ماکزیمم ۳۲۷۶۷ ساعت را ثبت کنند. برای مقادیر بزرگتر لازم است از RTM های ۳۲ بیتی استفاده کرد.RTM های ۳۲ بیتی می توانند دارای شماره ای بین ۰تا ۱۵ باشند.برای Set کردن ، فعال وغیر فعال کردن و خواندن مقدار این RTM ها از SFC101 استفاده می شود که تمام امکانات مورد نیاز را در خود دارد.




**ورودی NR**: شماره RTM که عددی بصورت بایت و بین ۰ تا ۱۵ است.

**ورودی MODE** : یک عدد بصورت بایت و بین 0 تا 6 می تواند باشد ، بصورت زیر:

**0** : در اینحالت وضعیت RTM به خروجی های CQ و CV ارسال می گردد و در صورتی که به CV به مقدار ماکزیمم یعنی 1-(2E31) برسد، متوقف شده وکد مربوط به OverFlow در خروجی Ret-Val ظاهر می گردد. وضعیت در پنجرهٔ Time System نیز قابل مشاهده است.

**1** : RTM شروع به کار می کند (با آخرین مقدار قبلی شمارش شده)

**2** : توقف RTM

**4** : RTM با مقدار PV ست می شود.

**5** : RTM با مقدار PV ست می شود و سپس شروع به کار می کند.

**6** : RTM با مقدار PV ست می شود و سپس متوقف می شود.

**ورودی PV** : مقدار اولیه RTM بصورت عدد صحیح ۳۲ بیتی یعنی از جنس DINT .

**خروجی CQ** : از جنس Bool که اگر یک باشد یعنی RTM فعال و اگر صفر باشد یعنی RTM متوقف است.

**خروجی CV** : از جنس DINT که مقدار فعلی RTM را نشان می دهد.

SFC101 را می توان برای RTM های ۱۶ بیتی نیز بکار برد یعنی آن را جایگزین SFC2 ، SFC3 و SFC4 نمود بدیهی است در این حالت عملکرد RTM ها ۱۶ بیتی خواهد بود.

تا اینجا اهم فانکشن های Clock بحث شد در ادامه به فانکشنهای مرتبط با دیتا بلاک ها می پردازیم.

### ۱۲-۲-۳ فانکشن های DB_FUNCT

این خانواده از فانکشن ها همانطور که از نامشان مشخص است برای کنترل دیتا بلاک بکار می روند اهم SFC های این خانواده عبارتند از :

- SFC22 , SFC85 : برای ایجاد دیتا بلاک
- SFC23 : حذف کردن دیتا بلاک
- SFC24 :تست کردن دیتا بلاک
- SFC25 : فشرده سازی حافظه

**SFC22 با نام سمبلیک CREAT_DB**

با استفاده از این SFC میتوان یک دیتابلاک با شماره دلخواه در حافظه سیستم ایجاد کرد.بدیهی است وقتی SFC در برنامه فراخوان شود DB در حافظه ساخته می شود در حالیکه در حالت Off Line چنین DB در پروژه وجود ندارد.نکاتی که لازمست به آنها توجه شود :

- برای شماره دیتا بلاک یک بازه ( حد پایین و حد بالا ) قابل تعریف است . فانکشن فوق کوچکترین شماره ممکن در بازه فوق را ایجاد می کند.
- برای ایجاد یک DB با شماره خاص لازم است حد پایین وحد بالا را یکسان ومساوی این شماره قرار دهیم.
- در صورتی که این DB با شماره مزبور قبلاً وجود داشته باشد SFC با خطا متوقف می شود.
- حجم حافظه دیتا بلاک نیز قابل تعریف است تعداد بایت که الزاماً بایستی عدد زوج باشد را بعنوان ورودی SFC وارد می کنیم .
- دیتا بلاکی که به این طریق ایجاد می شود حاوی مقادیر اولیه نیست در واقع مقادیر اولیه آن اتفاقی و رندوم می باشند.

شکل بلاک SFC22 با ذکر مثال در صفحه بعد ترسیم شده است.

ورودی Low_Limit : بصورت Word و حد پایین شماره DB را مشخص می کند در مثال ۱۰ داده شده است.

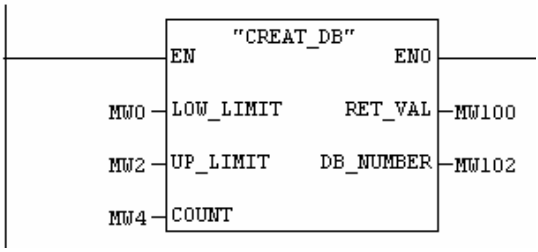
ورودی UP_Limit : بصورت Word و حد بالای شماره DB را مشخص می کند در مثال ۲۰ داده شده است.

ورودی Count : بصورت Word و حجم حافظه داخل DB را مشخص می کند در مثال ۲۰۰ داده شده است.

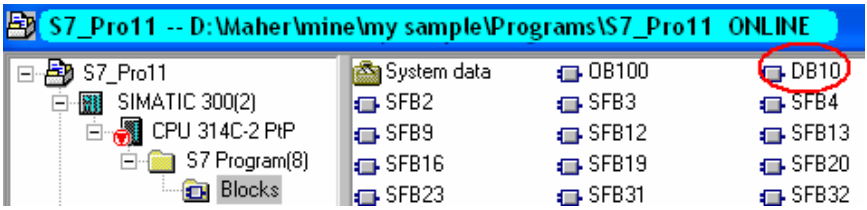
خروجی DB_NUMBER : بصورت Word و شماره DB ایجاد شده را نشان می دهد.

این برنامه اگر در OB1 اجرا شود تمام DB های ۱۰ تا ۲۰ (بشرط موجود نبودن)در حافظه سیستم ایجاد می شوند. پس برای ایجاد فقط یک DB بهتر است این برنامه بصورت کنترل شده یا در OB راه اندازی بکاررود.

```
OB100 :  "Main Program Sweep (Cycle)"
Network 1 : Title:

          L     10
          T     MW     0
          L     20
          T     MW     2
          L     200
          T     MW     4

Network 2 : Title:
```




پس از اجرای برنامه در مد On Line دیتابلاک شماره ۱۰ را خواهیم دید که در حافظه سیستم ایجاد شده است.بدیهی است با خروج از حالت On Line این DB در پروژه Off Line ظاهر نخواهد شد.



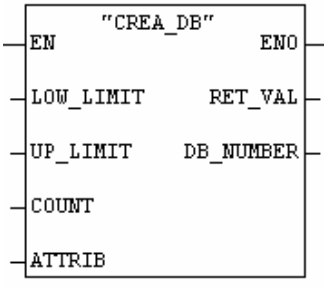

اگر DB مزبور را باز کنیم شکلی شبیه زیر خواهیم داشت همانطور که مشاهده میشود ۲۰۰ بایت برای دیتا منظور شده است:

| Address | Name | Type | Initial value |
|---|---|---|---|
| 0.0 | | STRUCT | |
| +0.0 | STAT0 | ARRAY[-32768..-32569] | |
| *1.0 | | BYTE | |
| =200.0 | | END_STRUCT | |

**SFC85 با نام سمبلیک CREA_DB**

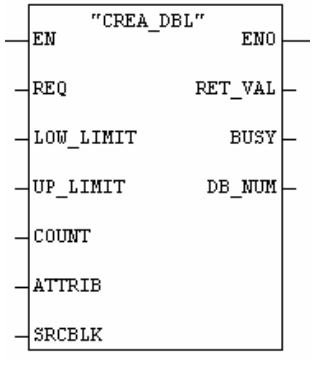
عملکرد این SFC شبیه SFC22 است با این تفاوت که قادر است ویژگی دیتا بلاک را نیز تنظیم کند . این فانکشن ورودی دیگری به نام Attrib دارد که از جنس بایت است و توسط آن میتوان تعریف کرد که دیتا بلاک در بخش Retentive حافظه تعریف شود یا خیر. می دانیم تعریف DB بصورت Retentive از پاک شدن دیتا های آن در صورت قطع و وصل تغذیه جلوگیری می کند. در جلد اول کتاب دیدیم که این ویژگی را میتوان در پارامترهای CPU تنظیم نمود.

ورودی Attrib : اگر B#16#00 داده شود دیتا بلاک Retentive و اگر B#16#04 داده شود .Non_Retent خواهد بود.




**تذکر:**

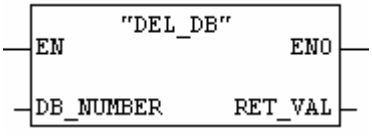
فانکشن SFC82 با نام سمبلیک CREA_DBL نیز در Library موجود است این فانکشن قادر است DB را در Load Memory (مثلاً روی کارت حافظه MMC ) ایجاد کند.بعلاوه با ورودی Attrib آن میتوان علاوه بر تنظیم Retentive بودن یا نبودن تنظیمات دیگر روی پارامتر های DB از جمله Read Only بودن را انجام داد.



**SFC23 با نام سمبلیک DEL_DB**

این فانکشن میتواند دیتابلاک مورد نظر را از Work Memory یا حتی Load Memory پاک کند. بدیهی است DB نباید در آن لحظه توسط برنامه باز شده باشد. این فانکشن میتواند تمام DB هایی که توسط کاربر ایجاد شده و تمام DB هایی که توسط فانکشن های SFC ایجاد شده را پاک کند DB هایی که روی کارت فلش حافظه ذخیره شده اند را نیز توسط این فانکشن میتوان پاک کرد.توصیه می شود که این فانکشن را فقط برای دیتا بلاک های Shared استفاده کنید. استفاده از آن برای DB های نوع Instanse غالباً با Error همراه است.

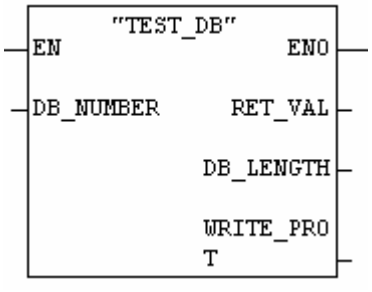
ورودی DB_Number بصورت Word شماره DB مورد نظر را می گیرد.




**SFC24 با نام سمبلیک TEST_DB**

توسط این فانکشن میتوان چک کرد که آیا DB با شماره مورد نظر در Work Memory موجود است یا خیر و در صورت وجود میتوان دید که حجم حافظه آن چقدر است و آیا Read Only است یا خیر.

اگر DB با شماره مزبور موجود نباشد خروجی Ret_Val مقدارهگز 80B1 را بر می گرداند.



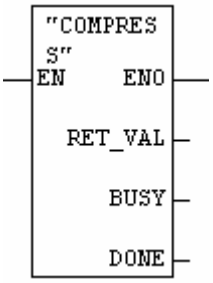

ورودی DB_NUMBER : شماره DB بصورت Word می گیرد.

خروجی DB_LENGHT : حجم بایت حافظه دیتا بلاک را برمی گرداند وبصورت Word است.

خروجی WRITE_PROTECT : بصورت BOOL است و اگر یک شود یعنی DB بصورت Read Only است.

**SFC25 با نام سمبلیک COMPRESS**

با استفاده از این SFC میتوان فضاهای خالی ایجاد شده در حافظه را ساماندهی و فشرده سازی نمود. فضاهای خالی یا GAP در حافظه ( در هر دو قسمت Load و Work ) از پاک کردن و ایجاد مداوم دیتا بلاک ها بوجود می آیند و حافظه موثر سیستم را کاهش می دهند.



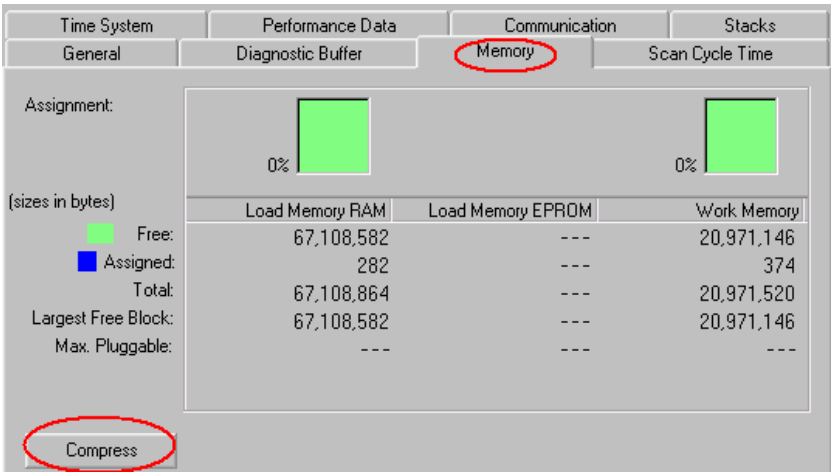

SFC25 با شیفت دادن خانه های حافظه RAM این بخش را فشرده سازی می کند.توجه شود این SFC نمیتواند دیتا بلاک هایی که بیش از ۱۰۰۰ بایت حافظه دارند را جابجا نماید.

خروجی BUSY : بصورت BOOL است و اگر یک شود یعنی فانکشن هنوز فعال است.

خروجی DONE : بصورت BOOL است و اگر یک شود نشان دهنده تکمیل عمل فشرده سازی است.

| فشرده سازی در حال انجام است | فشرده سازی انجام شده است |
|---|---|
| Ret_Val = 0000<br>Busy = 1<br>Done=0 | Ret_Val = 0000<br>Busy = 0<br>Done=1 |

**تذکر**: فشرده سازی از طریق پنجره Module Information که از منوی PLC در Simatic Manager و سایر زیر برنامه های آن قابل دسترس است نیز امکان پذیر می باشد مانند شکل زیر :



### ۱۲-۲-٤ فانکشن های مربوط به تایمرها و کانتر های IEC IEC_TC

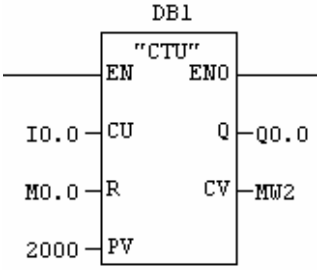
فانکشن های این خانواده تایمرهای و کانتر های خاصی را که طبق استاندارد IEC هستند فعال می کند. این فانکشنها به دو دسته تقسیم می شوند دسته اول تایمرهای IEC هستند که برای زمانهای طولانی تر و دقت بیشتر نسبت به تایمرهای معمولی بکار میروند. دسته دوم کانترهای IEC هستند که میتوانند تا شماره های بیشتری نسبت به کانترهای معمولی بشمارند لیست این فانکشن ها بصورت زیر است:

- SFB0 کانتر افزایشی
- SFB1 کانتر کاهشی
- SFB2 کانتر افزایشی کاهشی
- SFB3 تایمر پالس
- SFB4 تایمر تاخیر در وصل
- SFB5 تایمر تاخیر در قطع

**تذکر**: در Library خانواده دیگری برای فانکشن های IEC وجود دارد که FB یا FC هستند و از فانکشن های سیستم محسوب نمی شوند. این موارد در بخش ۱۲-۳ تشریح خواهند شد.

#### SFB0 با نام سمبلیک CTU

این فانکشن بعنوان یک شمارنده افزایشی عمل می کند و تفاوت آن با شمارنده های معمولی S7 اینست که بصورت Integer می شمارد نه بصورت BCD بنابر این تا 32767 قابل افزایش است در حالیکه در نوع BCD این عدد تا 999 قابل افزایش بود و اگر عدد بزرگتری نیاز بود می بایست از چند شمارنده استفاده می کردیم و وقتی اولی به 999 می رسید دومی را یکی افزایش و اولی را Reset می نمودیم. با فانکشن IEC این مشکل تا حد زیادی بر طرف شده است.




ورودی CU : برای ورودی که قرار است لبه مثبت آن شمارش شود بکار می رود.

ورودی R : برای ریست کردن کانتر به صفر بکار می رود.میتوان بعنوان مثال در OB100 این بیت را صفر کرد.

ورودی PV : مقدار Preset Value بصورت یک عدد Integer برای استفاده در فعال سازی خروجی Q.

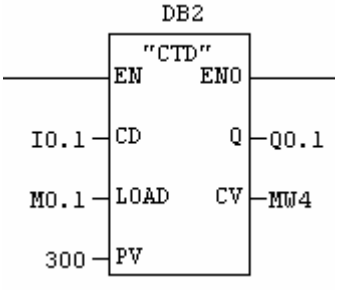
خروجی CV : مقدار فعلی شمارش را بصورت عدد Integer نمایش می دهد.

خروجی Q : این خروجی وقتی CV بزرگتر یا مساوی PV است یک و در غیر اینصورت صفر است.

**SFB1 با نام سمبلیک CTD**

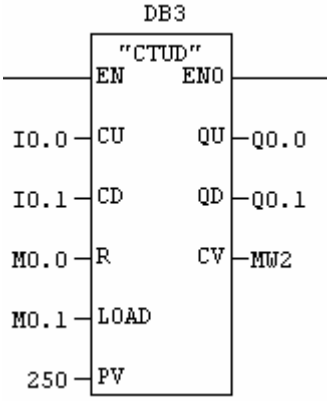
این فانکشن بعنوان یک شمارنده کاهشی عمل می کند و بصورت Integer می شمارد پس با هر لبه پالس در ورودی CD یک شماره کاهش می یابد تا به 32768- برسد. بدیهی است با رسیدن به این نقطه تغییرات لبه پالس ورودی تاثیری روی شماره کانتر ندارد تا زمانی که Reset گردد .

ورودی Load یک بیت است که با رسیدن لبه مثبت آن مقداراولیه PV به کانتر اعمال می شود. سایر ورودی ها و خروجی های آن شبیه کانتر قبلی است.




**SFB2 با نام سمبلیک CTUD**

این فانکشن بعنوان یک شمارنده افزایشی کاهشی عمل می کند بنابراین بازه کاری آن بین 32768- تا 32767+ است. بدلیل تشابه ورودی و خروجی های آن با دو نوع قبلی صرفاً به ارائه شکل LAD اکتفا می کنیم.

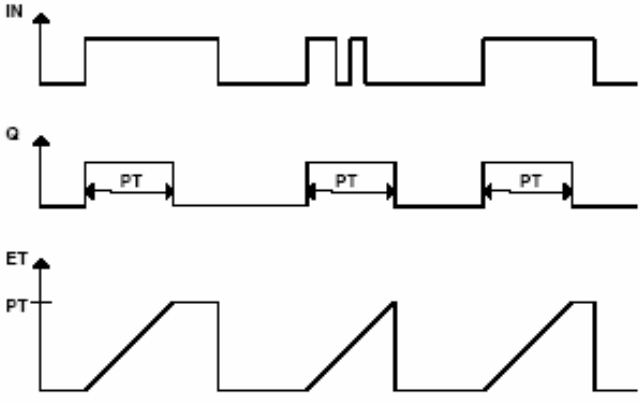


**SFB3 با نام سمبلیک TP**

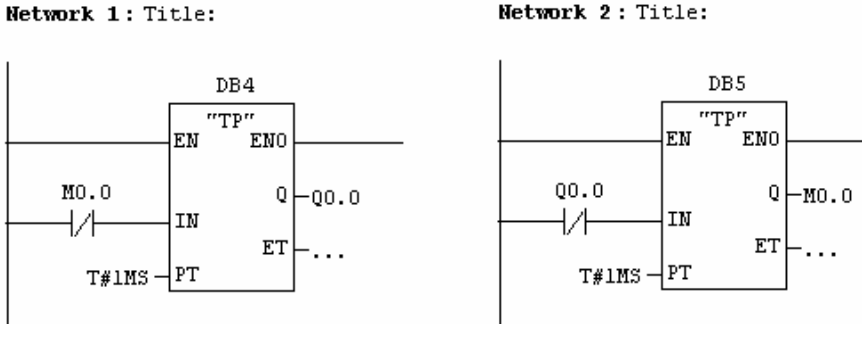
این فانکشن شبیه یک تایمر پالس عمل می کند و زمان را به فرمت Time می گیرد. بنابراین تفاوت آن با تایمرهای معمولی S7 عبارتست از:

- پله های زمان در اینجا 1 ms است در حالیکه در تایمرهای S7 پله ها حداقل 10 ms هستند.
- ماکزیمم زمان در اینجا 24D20H31M23S647MS است در حالیکه در تایمرهای S7 این زمان 2H46M30S0MS بود.

زمان به ورودی PT با فرمت #T داده می شود. در طول زمانی که ورودی IN یک است تایمر عمل زمان سنجی را انجام می دهد.فعال بودن تایمر در خروجی Q و زمان سپری شده در خروجی ET ظاهر می شود.

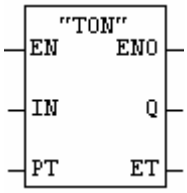


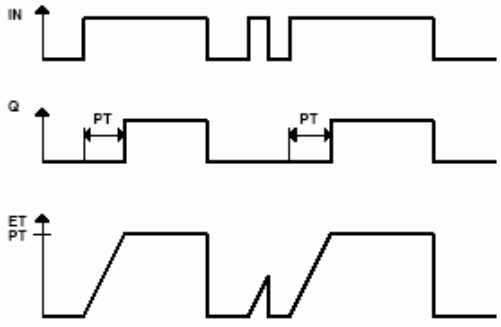

برنامه زیر نحوه ایجاد پالس در خروجی Q0.0 توسط این فانکشن را نشان می دهد. این خروجی یک میلی ثانیه روشن و یک میلی ثانیه خاموش خواهد بود.



**SFB4 با نام سمبلیک TON**

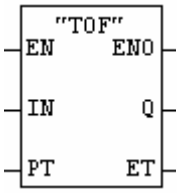
این فانکشن شبیه یک تایمر On Delay یعنی تاخیر در وصل عمل کندیعنی پس از فعال شدن ورودی IN باندازه زمان PT صبر می کند سپس خروجی Q روشن می گردد. تفاوتهای آن با تایمر تاخیر در وصل معمولی S7 همانست که در تایمر پالس ذکر شد.

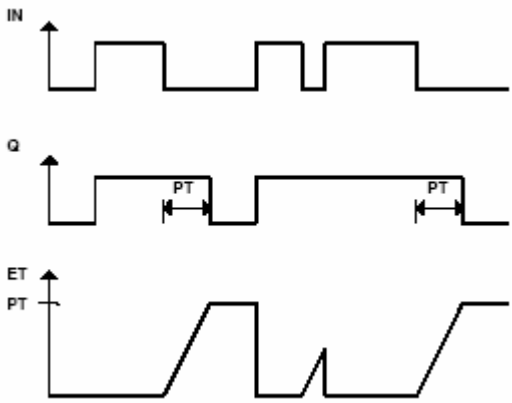



**SFB5 با نام سمبلیک TOF**

این فانکشن شبیه یک تایمر Off Delay یعنی تاخیر در قطع عمل کندیعنی پس از فعال شدن ورودی IN خروجی Q فعال شده و پس از قطع ورودی به اندازه زمان PT صبر می کند سپس خروجی Q خاموش می گردد.



### ۱۲-۲-۵ فانکشن های فعال و غیر فعال سازی وقفه ها IRT_FUNC

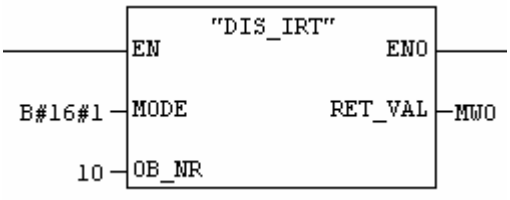
توسط این فانکشنها میتوان وقفه ها را فعال و غیر فعال نمود. فانکشن های این خانواده عبارتند از :

- SFC39 برای غیرفعال کردن وقفه ها
- SFC41 برای غیر فعال سازی وقفه ها با اولویت بالاتر
- SFC40 برای فعال سازی وقفه ها
- SFC42 برای فعال سازی وقفه ها با اولویت بالاتر

**تذکر**: فانکشن های دیگری نیز برای وقفه ها وجود دارند که در خانواده PGM_CNTL تشریح می شوند.

**SFC39 با نام سمبلیک DIS_IRT**

وقتی این فانکشن صدا زده شود و شرایط فعال شدن وقفه ای حتی وقفه های خطای آسنکرون پیش بیاید وقفه غیرفعال شده و سیستم عامل OB مربوط به آن وقفه را صدا نخواهد زد. وقفه ای که به این طریق غیر فعال شود توسط SFC40 میتواند مجدداً فعال گردد.

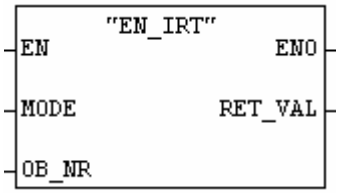



ورودی Mode : در این ورودی کدی بصورت Byte داده می شود و مد کاری وقفه مشخص می گردد:

<table>
<tr><td colspan="2">تمام وقفه های جدید و وقفه های خطای آسنکرون غیر فعال می شوند ولی وقفه های سنکرون غیر فعال نمی گردند. رخداد ها مانند قبل در Diagnostic Buffer ثبت می شوند.</td><td>00</td></tr>
<tr><td colspan="2">تمام وقفه های مربوط به خانواده OB که در ورودی OB_NR آمده غیر فعال می شوند بعنوان مثال اگر OB_NR=30 باشد تمام وقفه های سیکلی OB3x غیر فعال می شوند یا اگر OB_NR=40 باشد تمام وقفه های سخت افزاری OB4x غیر فعال می گردند. رخداد ها مانند قبل در Diagnostic Buffer ثبت می شوند.</td><td>01</td></tr>
<tr><td colspan="2">فقط وقفه خاصی که شماره OB آن در جلوی OB_NR نوشته شده غیر فعال می گردد.</td><td>02</td></tr>
<tr><td rowspan="3">در این ۳ حالت صرفاً کد W#16#5380 به بافر وارد می شود</td><td>شبیه مد 00 ولی با این تفاوت که رخداد ها در بافر CPU ثبت نمیشوند</td><td>80</td></tr>
<tr><td>شبیه مد 01 ولی با این تفاوت که رخداد ها در بافر CPU ثبت نمیشوند</td><td>81</td></tr>
<tr><td>شبیه مد 02 ولی با این تفاوت که رخداد ها در بافر CPU ثبت نمیشوند</td><td>82</td></tr>
</table>

**SFC40 با نام سمبلیک EN_IRT**

این فانکشن عکس SFC39 عمل می کند یعنی وقفه غیرفعال شده را فعال می کند. ورودی های این فانکشن مشابه SFC39 است و نیازی به تکرار آنها نیست.



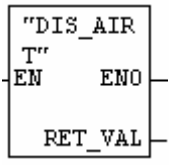

**مثال**:

برنامه ای در OB1 و برنامه ای در OB35 داریم ولی میخواهیم برنامه نوشته شده در Network2 از OB1 تحت تاثیر وقفه سیکلی OB35 قرار نگیرد . برای این منظور قبل از این network فانکشن SFC39 و بعد از آن SFC40 را صدا میزنیم. بصورت زیر:

```
OB1 :
      Network 1:
      CALL  "DIS_IRT"
          MODE   :=B#16#2
          OB_NR  :=35
          RET_VAL:=MW0
      Network2:
        L    PIW   0
        L    10
        -I
        T    MW   100
      Network3:
        CALL  "EN_IRT"
         MODE   :=B#16#2
         OB_NR  :=35
         RET_VAL:=MW2
```

**SFC41 با نام سمبلیک DIS_AIRT و SFC42 با نام سمبلیک EN_AIRT**

این دو فانکشن نیز مشابه SFC39 و SFC40 هستند با این تفاوت که فقط وقفه هایی که اولویت بالاتری نسبت به OB فعلی دارند را غیر فعال یا فعال میسازند از اینرو در ورودی آنها شماره OB یا Mode داده نمی شود. برای یاد آوری اولویت OB ها خواننده محترم را به مباحث وقفه ها در جلد اول کتاب ارجاع می دهیم.



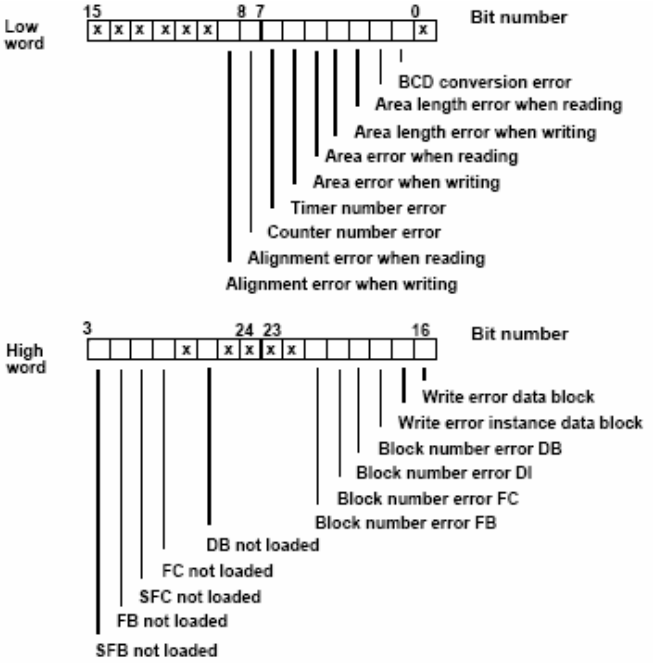
### ۱۲-۲-۶ فانکشن های فیلتر کردن وقفه های سنکرون

توسط این فانکشن ها میتوان وقفه های سنکرون را برای خطا یا خطاهای مورد نظر فیلتر کردیا فیلتر قبلی را غیر فعال نمود. همانطور که در جلد اول کتاب اشاره شد وقفه های سنکرون مربوط به دو دسته خطا هستند:

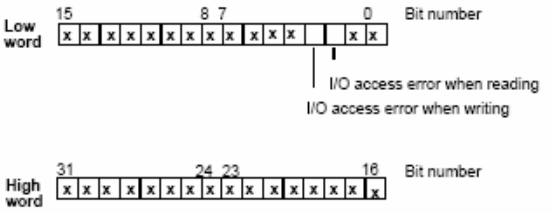
۱. خطا های برنامه نویسی که در اینحالت OB121 توسط سیستم عامل صدا زده می شود.

۲. خطاهای دسترسی به آدرسها که در اینحالت OB122 فراخوان می شود.

خطاهای دسته اول بصورت Bit Patern در ۳۲ بیت مطابق شکل زیر ظاهر می شوند هر خطا یکی از این بیت ها را یک می کند. برای فیلتر کردن خطای مورد نظر کافیست بیت مربوطه را توسط فانکشن یک کنیم.

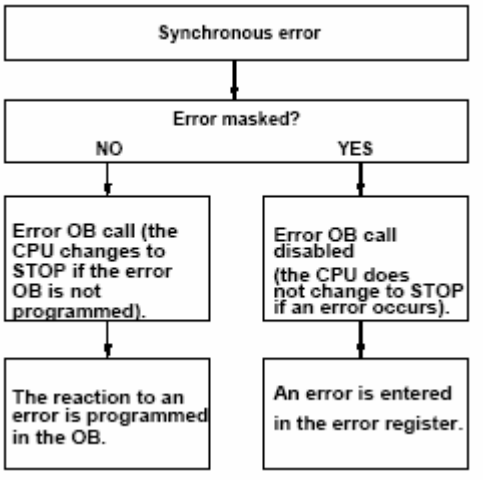



خطاهای دسته دوم بیتهای دیگری از یک Double Word را یک می کنند شکل زیر

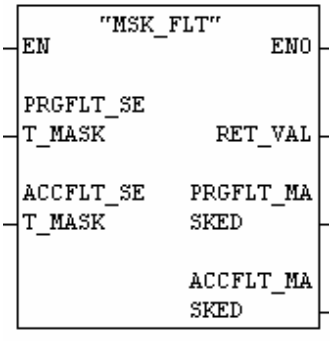


**SFC36 با نام سمبلیک MSK_FLT**

این فانکشن میتواند هر دو دسته از خطاهای سنکرون را فیلتر کند . با صدا زدن این فانکشن اگر در حین اجرای بلاک فعلی خطای سنکرون که درجه اولویتش بالاتر است پیش بیاید سیستم عامل آنرا صدا نخواهد زد و در صورت عدم وجود OB12x نیز CPU متوقف نخواهد شد ولی خطا در بافر CPU ثبت خواهد شد. بایستی توجه داشت که معمولاً بهتر است در هنگام بروز خطا ، کاربر بنحوی از آن مطلع شود وقتی خطا فیلتر نشود و OB12x نیز موجود نباشد CPU متوقف میشود و کاربر مطلع میگردد ولی همانطور که در جلد اول کتاب توضیح داده شد بهترین روش برنامه نویسی وقفه های OB12x و استفاده از کد های خطا برای آگاه کردن اپراتور است. وجود پیغام در بافر Diagnostic وقتی مفید است که کاربر بطور On Line متصل باشد.




شکل LAD این فانکشن بصورت زیر است. ورودی PRGFLT_SET_MASK بصورت Dword است و کد مربوط به خطای سنکرون نوع اول یعنی خطای برنامه نویسی که میخواهیم فیلتر شود به آن داده می شود.




ورودی ACCFLT_SET_MASK: بصورت Dword است و کد مربوط به خطای سنکرون نوع دوم یعنی خطای دسترسی به آدرسها که میخواهیم فیلتر شود به این ورودی داده می شود.

در صورت فیلتر شدن خطاهای فوق کد Dword داده شده در خروجی های فانکشن ظاهر خواهد شد.

**مثال ۱ :**

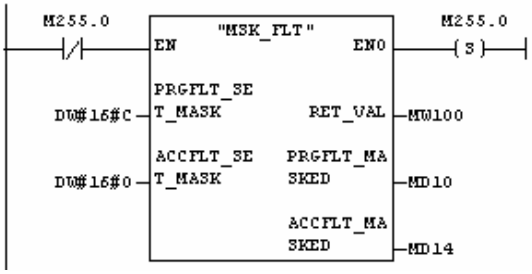
میخواهیم فقط دو خطای سنکرون زیر که هردو از خطاهای برنامه نویسی هستند فیلتر شوند:

- Area length error when reading
- Area length error when writing

با توجه به شکل دو صفحه قبل می بینیم که بیت های ۲ و ۳ از Low Word لازمست یک شوند.

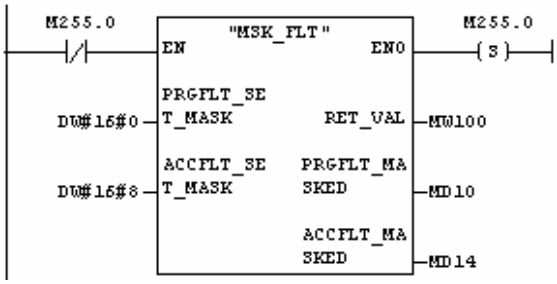
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | = | W#16#C |

و از آنجا که بیت های High Word بدون تغییر می مانند.پس این فیلتر معادل DW#16#C خواهد بود. این مقدار به ورودی اول فانکشن داده می شود و ورودی دوم صفر است.




**مثال ۲ :**

اگر بخواهیم خطای دسترسی I/O درهنگام نوشتن فیلتر شود در اینحالت با توجه به شکل دو صفحه قبل که مربوط به خطای نوع دوم بود کافیست بیت 3 را یک کنیم که معادل 8 هگز خواهد بود.این مقدار به ورودی دوم داده می شود و ورودی اول صفر است.




**SFC37 با نام سمبلیک DMSK_FLT**

توسط این فانکشن میتوان فیلتری که با SFC36 اعمال شده را برداشت.ورودی و خروجی های آن مشابه SFC36 می باشد.

### ۱۲-۲-۷ فانکشن های مربوط به وقفه های Time of Day

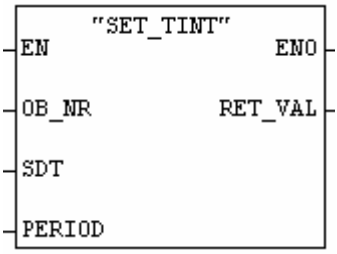
این فانکشنها برای تنظیم ، فعال سازی و غیر فعال سازی وقفه های TOD بکار می روند و عبارتند از :

- SFC28 برای تنظیم وقفه TOD
- SFC30 برای فعال سازی وقفه TOD
- SFC29 برای غیر فعال کردن وقفه TOD
- SFC31 برای نمایش وقفه TOD

بهتر است خواننده محترم مطالب جلد اول در مورد این وقفه را یکبار دیگر مرور کند تا ضمن یادآوری عملکرد این وقفه تفاوت بین کاربرد این فانکشن با تنظیماتی که در HWconfig انجام می شد را بهتر درک نماید. با استفاده از فانکشنهایی که در اینجا مورد بحث قرار می گیرد نیازی به انجام تنظیمات Hwconfig نخواهد بود.

**SFC28 با نام سمبلیک SET_TINT**

توسط این فانکشن میتوان وقفه TOD را تنظیم کرد بعبارت دیگر تنظیماتی که در پنجره وقفه TOD از پارامترهای CPU در Hwconfig انجام می شد توسط این فانکشن نیز امکان پذیر است صدا زدن این فانکشن باعث از بین رفتن وقفه تنظیم شده قبلی برای OB مشخص شده می گردد.البته بایستی توجه داشت که این فانکشن فقط پارامترها را تنظیم می کند و برای فعال کردن آنها لازم است فانکشن SFC30 بدنبال آن صدا زده شود.




ورودی OB_NR : شماره OB مورد نظر به این ورودی داده می شود بعنوان مثال عدد ۱۰ معرف OB10 می باشد.

ورودی SDT : تاریخ و زمان بفرمت Date_And_Time به این ورودی داده می شود. همانطور که قبلاً نیز اشاره شد تاریخ و زمان لازمست ابتدا توسط فانکشن IEC با هم ترکیب شده و به این ورودی داده شوند.

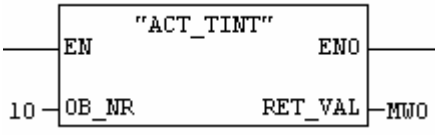
ورودی PERIOD : این ورودی بصورت Word است و دوره اجرای وقفه را مطابق جدول زیر مشخص میکند:

| | |
|---|---|
| W#16#0000 = once | W#16#1202 = weekly |
| W#16#0201 = every minute | W#16#1401 = monthly |
| W#16#0401 = hourly | W#16#1801 = yearly |
| W#16#1001 = daily | W#16#2001 = at month's end |

مثال از کاربرد این فانکشن را همراه با SFC30 در صفحه بعد ببینید.

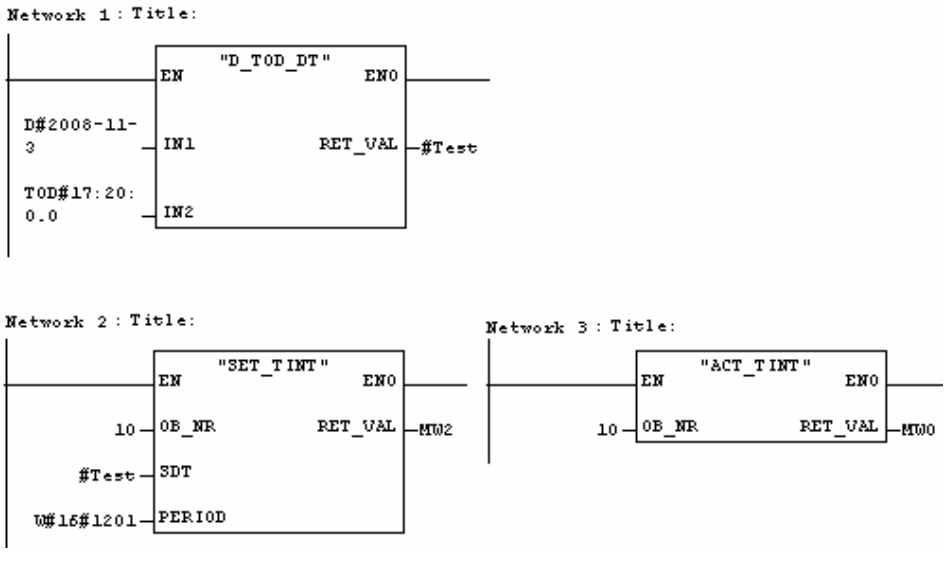
**SFC30 با نام سمبلیک ACT_TINT**

این فانکشن برای فعال سازی وقفه TOD که پارامترهای آن قبلاً با SFC28 تنظیم شده بکارمی رود همانطور که در شکل مشاهده می شود فقط یک ورودی دارد که به آن شماره OB مورد نظر بصورت یک عدد صحیح داده می شود.



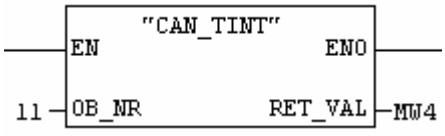

**مثال**

در این مثال فانکشنهای SFC28 و SFC30 همراه با FC3 که از فانکشنهای IEC با نام سمبلیک D_TOD_DT است و بعداً تشریح خواهد شد بکار رفته است. هدف تنظیم وقفه TOD برای OB10 است که در تاریخ 2008-11-03 ساعت 5:20 بعدازظهر اجراشده وبصورت هفتگی تکرار شود. برای اینکار در OB1 یک متغیر Temp از جنس Date_And_Time با نام Test تعریف می کنیم سپس توسط FC3 تاریخ و زمان فوق را با هم ترکیب کرده در Test می ریزیم. در هنگام صدا کردن SFC28 متغیر TEST را به ورودی SDT آن اختصاص می دهیم. پس از تنظیم پارامترها SFC30 را صدا می کنیم.



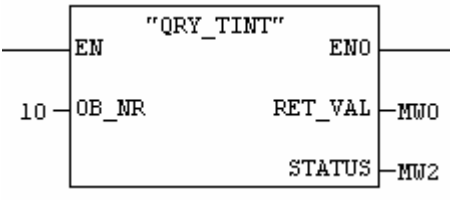
**SFC29 با نام سمبلیک CAN_TINT**

این فانکشن برای غیر فعال سازی وقفه TOD که قبلاً توسط فانکشن SFC30 یا از طریق تنظیمات Hwconfig فعال شده بکارمی رود این فانکشن نیز فقط یک ورودی دارد که به آن شماره OB مورد نظر بصورت یک عدد صحیح داده می شود.




**SFC31 با نام سمبلیک QRY_TINT**

توسط این فانکشن میتوان شرایط وقفه TOD که قبلاً فعال شده را مشاهده نمود. شماره OB مورد نظر بصورت عدد صحیح داده می شود و وضعیت وقفه مطابق با کدهای مندرج در جدول زیر در خروجی Status ظاهر می گردد.




| Status : | | |
|---|---|---|
| **Bit** | **Value** | **Meaning** |
| 0 | 0 | Time-of-day interrupt is enabled by operating system. |
| 1 | 0 | Time-of-day interrupts are accepted. |
| 2 | 0 | Time-of-day interrupt is not activated or has elapsed. |
| 3 | - | - |
| 4 | 0 | Time-of-day interrupt OB is not loaded. |
| 5 | 0 | The execution of the time-of-day interrupt is not disabled by an active test OBfunction. |
| 6 | 0 | Base for the time-of-day interrupt is the basic time |
| | 1 | Base for the time-of-day interrupt is the local time |

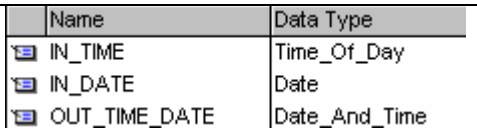
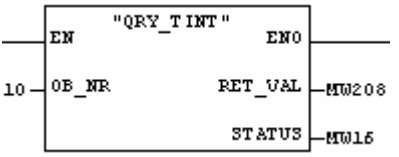
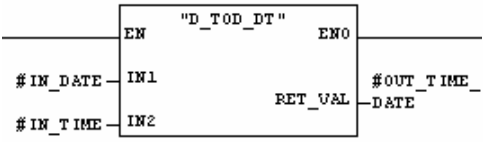
**تذکر**: یکی از کاربردهای فانکشنهای چهار گانه وقفه در جایی است که CPU فقط یک OB1x را ساپورت کند. در اینحالت در تنظیمات Hwconfig فقط میتوان یک وقفه TOD را تنظیم نمود ولی با بکار بردن فانکشن و برنامه نویسی میتوان OB10 را برای وقفه TOD جدید تنظیم نمود.

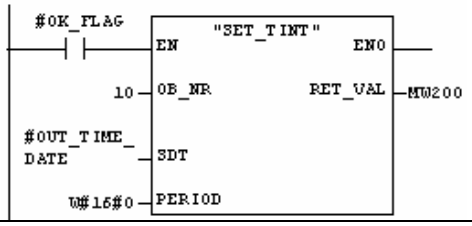
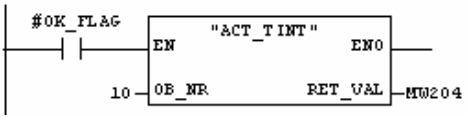
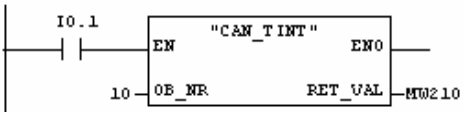
## مثال

میخواهیم خروجی Q4.0 از ۵ صبح دوشنبه 2006-12-11 تا ساعت ۲۰ جمعه 2006-12-15 روشن و پس از آن خاموش گردد. CPU مورد استفاده فقط OB10 را برای وقفه TOD ساپورت می کند.کلید I0.0 برای فعال کردن وقفه و کلید I0.1 برای غیر فعال کردن وقفه بکار می رود.

برنامه به این نحو است که در OB1 ابتدا OB10 را برای دوشنبه سپس در طول هفته فوق OB10 رابرای جمعه تنظیم می کنیم. روشن و خاموش شدن را در OB10 برنامه نویسی می نماییم.

برنامه OB1

| | |
|---|---|
| ابتدا در قسمت Temp متغیرهای روبرو را تعریف کرده سپس برنامه را در Network های زیر می نویسیم. |  <br>Name / Data Type<br>IN_TIME / Time_Of_Day<br>IN_DATE / Date<br>OUT_TIME_DATE / Date_And_Time |
| SFC31 را صدا میزنیم و وضعیت OB10 را در متغیر MW16 می ریزیم بیت M17.2 این متغیراگر صفر باشد نشان دهنده اینست که وقفه OB10 فعال نیست یا سپری شده و بیت M17.4 اگر یک باشد مبین اینست که OB10 به PLC دانلود شده است این بیتها را بعداً استفاده می کنیم. | Network 1:<br>  <br> |
| در این قسمت بسته به وضعیت Q4.0 تاریخ و زمان به متغیرهای Temp داده می شود. اگر Q4.0 خاموش باشدیعنی هنوز وقفه اول برای دوشنبه فعال نشده بنابراین زمان و تاریخ دوشنبه را می دهیم. اگر Q4.0 روشن باشد یعنی وقفه اول فعال شده پس تاریخ و زمان وقفه دوم یعنی جمعه را می دهیم. | Network 2:<br>AN Q 4.0<br>JC MOND<br>L D#2006-12-11<br>T #IN_DATE<br>L TOD#20:0:0.0<br>T #IN_TIME<br>JU WNDL<br>MOND: L D#2006-12-15<br>T #IN_DATE<br>L TOD#5:0:0.0<br>T #IN_TIME<br>WNDL: NOP 0 |
| در اینجا زمان وتاریخ را با استفاده از فانکشن IEC ترکیب کرده و در متغیر Temp می ریزیم. | Network 3:<br>  <br> |

| | |
|---|---|
| Network 4:<br>A I 0.0<br>AN M 17.2<br>A M 17.4<br>= #OK_FLAG | بررسی شرایط فعال کردن وقفه با توجه به بیتهای Status که قبلاً ذکر شد و وضعیت سوئیچ I0.0 یک متغیر Tmep را بعنوان نتیجه اینترلاک ها بکار می بریم. |
| Network 5:<br>  <br> | تنظیم وقفه OB10 با استفاده از SFC28 برای اجرای فقط یکبار در زمان و تاریخ مشخص |
| Network 6:<br>  <br> | فعال کردن وقفه OB10 با استفاده از SFC30 |
| Network 7:<br>  <br> | غیر فعال کردن وقفه OB10 در صورتی که کلید I0.1 زده شود. |

OB10 :

چون OB10 برای دو زمان و تاریخ تنظیم می شود که Q4.0 در اولی بایستی روشن و در دومی خاموش گردد بنابر این قبل از هر چیز لازم است چک شود که برای کدام حالت صدا زده شود اینکا ر را به روشهای مختلف میتوان انجام داد . ساده ترین حالت نوشتن دو سطر برنامه زیر است.

```
AN   Q4.0
=    Q4.0
```

**تذکر** :

برنامه فوق برای تکرار بصورت هفتگی در جلد اول کتاب آمده است.

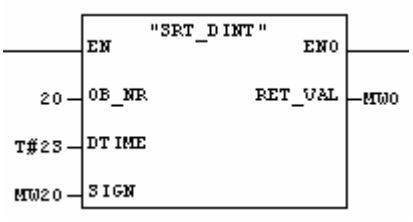
### ۱۲-۲-۸ فانکشن های مربوط به وقفه های تاخیری Delay Time

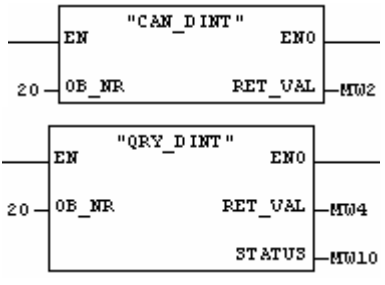
وقفه های تاخیری برای ایجاد تاخیر در پردازش برنامه خاصی توسط CPU بکار می روند و برنامه نویسی آنها در OB2x انجام می شود. سیستم عامل پس از گذشت زمان تاخیر وقفه را اعمال می نماید. در این قسمت فانکشنهایی که برای کار با این وقفه بکار می روند معرفی شده اند.

**SFC32 با نام سمبلیک SRT_DINT**

این فانکشن برای فعال سازی وقفه تاخیری است و پس از گذشت زمانی که در ورودی DTIME تعیین می شود OB مربوطه که شماره آن به ورودی OB_NR داده شده را صدا میزند.فرمت زمان بصورت Time# است که Resolution آن یک میلی ثانیه می باشد.

کاربر میتواند کد دلخواهی را بصورت Word به ورودی Sign این فانکشن بدهد. این کد در هنگامی که OB2x توسط سیستم عامل صدا زده می شود در پارامتر Temp بالای OB به نام OB20_Sign وارد شده و در برنامه OB20 قابل استفاده است.




**SFC33 با نام سمبلیک CAN_DINT**

این فانکشن برای غیرفعال سازی وقفه تاخیری بکار می رود و در ورودی آن فقط شماره OB2x داده می شود.

**SFC34 با نام سمبلیک QRY_DINT**

این فانکشن برای نمایش وضعیت وقفه تاخیری بکار می رود ورودی آن شماره OB2x را می گیرد و وضعیت را در خروجی Status مطابق جدول زیر نشان می دهد.

| Bit | Value | Meaning |
|---|---|---|
| 0 | 0 | Time-delay interrupt is enabled by the operating system. |
| 1 | 0 | New time-delay interrupts are not rejected. |
| 2 | 0 | Time-delay interrupt is not activated or has elapsed. |
| 3 | - | - |
| 4 | 0 | Time-delay interrupt-OB is not loaded. |
| 5 | 0 | The execution of the time-delay interrupt OB is not disabled by an active test function. |

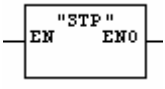
**تذکر**: مثالی از کاربرد وقفه تاخیری در جلد اول کتاب آورده شده است.

### ۹-۲-۱۲ فانکشن های کنترل برنامه PGM_CNTL

فانکشن هایی نیز با عنوان Program Control تعبیه شده اند که در اینجا به دو مورد از آنها اشاره می گردد.فانکشن SFC46 برای متوقف کردن پردازش CPU و فانکشن SFC47 برای ایجاد تاخیر در پردازش بلاک جاری در CPU بکار میرود.

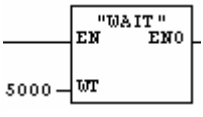
**SFC46 با نام سمبلیک STP**

فراخوانی این فانکشن در برنامه منجر به توقف CPU می گردد. این فانکشن هیچ ورودی یا خروجی ندارد. کاربرمیتواند از طریق برنامه نویسی CPU را متوقف کند مثلاً در OB1x آنرا صدا بزند تا در زمان و تاریخ مشخصی PLC بایستد یا در OB8x تا در صورت بحرانی بودن خطا عمل توقف انجام شودبه هر حال استفاده از آن بستگی به نیاز فرآیند دارد.




**SFC47 با نام سمبلیک Wait**

فراخوانی این فانکشن منجر به تاخیر زمانی در اجرای برنامه باندازه زمان مشخص شده در ورودی Wait می گردد. عدد صحیح به این ورودی داده می شود که مفهوم آن میکروثانیه است پس ماکزیمم تاخیر زمانی ۳۲۷۶۷ میکروثانیه خواهد بود.




لازم است دقت شود که عملکرد این فانکشن با وقفه های تاخیری OB2x متفاوت است در وقفه های تاخیری CPU در حالیکه مشغول اجرای OB1 است با یک تاخیر OB2x را صدا میزند ولی اگر SFC47 در OB1 صدا زده شودبرنامه در همان نقطه دچار تاخیر می گرددو دستور بعدی پس از گذشت زمان تعیین شده اجرا می گردد.بعنوان مثال در برنامه زیر بین خواندن ورودی آنالوگ و ارسال خروجی آنالوگ ۲۰ میلی ثانیه تاخیر ایجاد شده است:

```
L    PIW 252
CALL "WAIT"
 WT:=20000
T    PQW 400
```

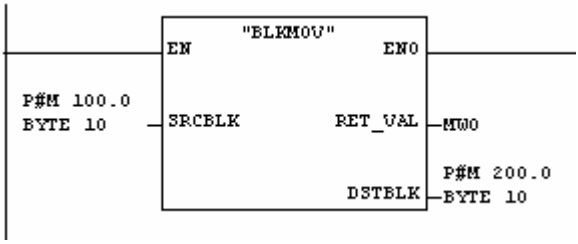
### ۱۲-۲-۱۰ فانکشن های کپی کردن محتویات حافظه MOVE

توسط این فانکشنها میتوان محتویات حافظه را از یک ناحیه آدرس به ناحیه آدرس دیگری کپی کرد.

**SFC20 با نام سمبلیک BLKMOV**

این فانکشن میتواند بخشی از ناحیه حافظه را به بخشی دیگر از حافظه کپی کند. بخشهای دیتابلاک و Memory بیت ها و آدرسهای ورودی(PII) و خروجی (PIQ) در این فانکشن قابل استفاده هستند. تفاوت این بلاک با بلاک MOVE که در LAD و FBD استفاده می شد در این است که MOVE حداکثر ۴ بایت را میتواند منتقل کند در حالیکه SFC20 چنین محدودیتی ندارد بعلاوه متغیرهایی از جنس Array و Struct فقط از طریق SFC20 قابل انتقال هستند.

در این SFC آدرس مبدا بصورت Pointer به ورودی SRCBLK و آدرس مقصد به خروجی DSTBLK داده می شود. در مثال شکل زیر ۱۰ بایت که از MB100 شروع شده به ۱۰ بایت که از آدرس MB200 شروع می شود کپی می گردد.

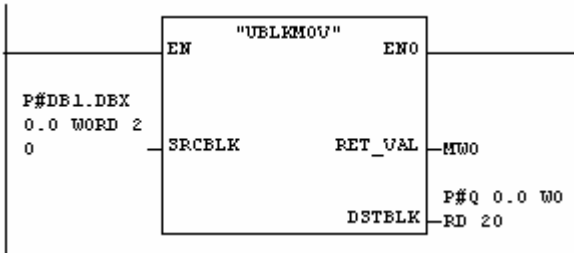



نکاتی در استفاده از SFC20 لازم است به آنها توجه شود:

- آدرس مبدا و مقصد نباید با یکدیگر تداخل داشته باشند.
- اگر آدرس مقصد کوچکتر از مبدا باشد برخی از دیتا ها کپی نخواهند شد.
- اگر آدرس دهی Pointer از جنس Bool انتخاب شود طول آدرس بایستی مضربی از ۸ یعنی بایت باشد در غیر اینصورت SFC20 اجرا نخواهد شد مثلاً نوشتن آدرس P#M0.0 Bool 7 اشکال دارد.
- اگر در حین کار SFC20 توسط وقفه قطع شود پس از بر طرف شدن وقفه عمل کپی را ادامه نخواهد داد .

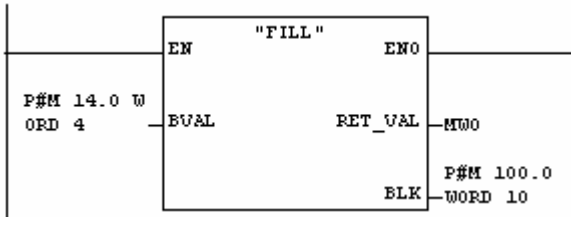
**SFC81 با نام سمبلیک UBLKMOV**

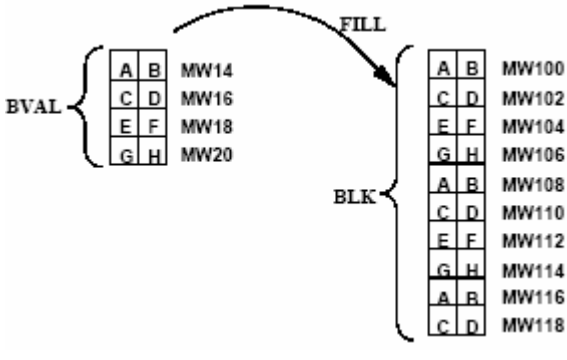
عملکرد این فانکشن مشابه SFC20 است با این تفاوت که وقفه نمیتواند آنرا قطع کند . حرف U در ابتدای اسم آن بر اساس همین عملکرد یعنی Uninterruptible است. لازم است توجه شود که بکار بردن این فانکشن باعث افزایش زمان پاسخ CPU به وقفه ها می گردد.




**SFC21 با نام سمبلیک Fill**

عملکرد این فانکشن شبیه SFC20 است با این تفاوت که اگر ناحیه حافظه مقصد بزرگتر از مبدا باشد کار کپی آنقدر ادامه می یابد تا کل ناحیه مقصد پر شود. برای فهم بیشتر به مثال زیر و نحوه عملکرد آن در شکل دقت کنید.



### ۱۲-۲-۱۱ فانکشن های مربوط به آدرس مدول ها

این فانکشن ها میتوانند آدرس جغرافیایی را به آدرس منطقی یا برعکس تبدیل کنند.منظور از آدرس جغرافیایی یک مدول شماره رک و شماره اسلات محل قرار گیری آنست و منظور از آدرس منطقی (Logical) آدرسی است که CPU از آن استفاده می کند و در برنامه نویسی بکار می رود. آدرس منطقی وقتی در Hwconfig کارت را در اسلات وارد می کنیم در جلوی آن ظاهر می شود. این آدرس غالباً قابل تغییر توسط کاربر نیز هست. آدرس منطقی که بطور پیش فرض به هر مدول داده می شود از رابطه زیر بدست می آید:

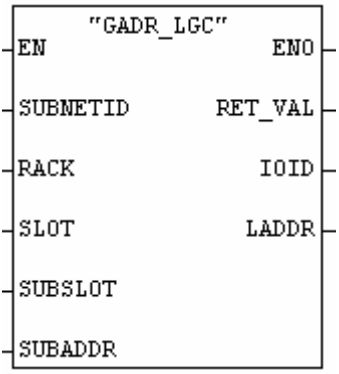
برای کارتهای دیجیتال: ۴ x (۱- شماره اسلات)= آدرس منطقی پیش فرض

برای کارتهای آنالوگ: ۵۱۲ + ۶۴ x (۱- شماره اسلات)= آدرس منطقی پیش فرض

بعنوان مثال وقتی کارت دیجیتال ۱۶ ورودی را در اولین اسلات قرار می دهیم آدرس پیش فرض کارت صفر است و وقتی همین کارت را در اسلات بعدی وارد می کنیم آدرس پیش فرض ۴ خواهد بودکه طبق فرمول فوق محاسبه شده است در حالیکه این ورودی ۲ بایت را بیشتر اشغال نمی کند بعبارت دیگر آدرس های ۲ و ۳ خالی میمانند. با این توضیحات به معرفی فانکشن های این خانواده می پردازیم.

#### SFC5 با نام سمبلیک GADR_LGC

این فانکشن همانطور که از نامش پیداست آدرس جغرافیایی کارت را می گیرد و آدرس منطقی را بر می گرداند. کارت میتواند روی رک اصلی یا رک اضافی یا روی DP Salve در شبکه Profibus قرار گرفته باشد.




**ورودی SUBNETID** : بصورت BYTE اگر کارت روی رک اصلی یا اضافی قرار گرفته باشد به این ورودی مقدار صفر داده می شود و اگر کارت روی شبکه DP قرار گرفته شماره ID مربوط به شبکه را وارد می کنیم شماره ID شبکه معمولاً روی خط Profibus نوشته شده است.

**ورودی RACK** : بصورت Word است اگر کارت در رک قرار گرفته شماره رک و اگر روی DP Slave است شماره Node مربوط به Slave داده می شود.

**ورودی SLOT** : بصورت Word است و شماره اسلات کارت را می گیرد .

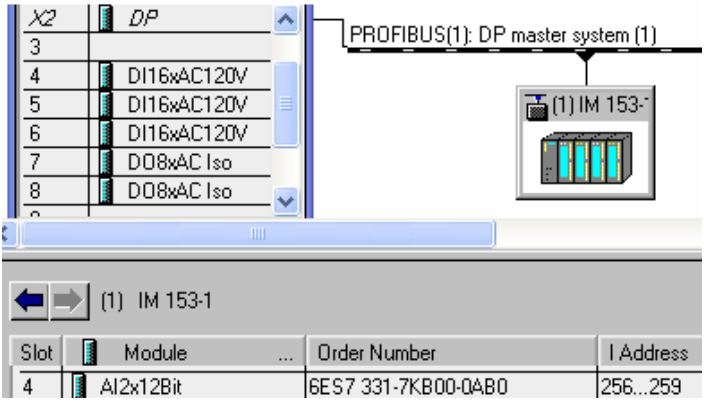
**ورودی SUBSLOT** : بصورت Byte است و در مواردی که Submodule داریم بکار می رود.اگر موجود نباشد صفر داده می شود.

**ورودی SUBADDR** : مقدار آفستی که کاربر در تغییر آدرس اعمال کرده است بصورت Word.

**خروجی LADDR** : در این خروجی آدرس منطقی مدول بصورت Word قابل مشاهده است .

**خروجی IOID** : این خروجی بصورت کد Byte نشان می دهد که آدرس خروجی ورودی است(B#16#54) یا از نوع خروجی(B#16#55)

**مثال**: در شکل زیر فانکشن SFC5 بصورت Online آدرس منطقی کارتی که در اسلات ۵ رک اصلی قرار گرفته و کارتی که روی ET200M در اسلات ۴ قرار گرفته را نشان می دهد.




| کارتی که در اسلات ۴ روی ET200M قرار دارد | کارتی که در اسلات ۵ رک اصلی قرار دارد |
|---|---|
| "GADR_LGC" EN ENO; B#16#1 – SUBNETID; W#16#1 – RACK; W#16#4 – SLOT; B#16#0 – SUBSLOT; W#16#0 – SUBADDR; RET_VAL – MW0 (0); IOID – MB100 (16#54); LADDR – MW4 (16#0100) | "GADR_LGC" EN ENO; B#16#0 – SUBNETID; W#16#0 – RACK; W#16#5 – SLOT; B#16#0 – SUBSLOT; W#16#0 – SUBADDR; RET_VAL – MW2 (0); IOID – MB200 (16#54); LADDR – MW6 (16#0004) |

**SFC49 با نام سمبلیک LGC_GADR**

این فانکشن عکس SFC5 عمل می کند یعنی آدرس منطقی را به آدرس جغرافیایی بر می گرداند.

### ۱۲-۲-۱۲ فانکشن های Diagnostic Buffer

در اینجا یکی از فانکشن های مربوط به Diagnostic Buffer را با ذکر مثال تشریح می کنیم.

#### SFC52 با نام سمبلیک WR_USMSG

با این فانکشن می توان پیغام دلخواهی را در شرایط مورد نظر به بافر Diagnostic در CPU انتقال داد تا در لیست پیام های Diagnostic قابل مشاهده باشد بدینطریق با چک کردن بافر مزبور میتوان فهمید که آیا Event مورد نظر کاربر رخ داده است یا خیر. از آنجا که Event بصورت Time Stamp یعنی همراه با لحظه وقوع در CPU ثبت می شود این اطلاعات میتواند کمک شایانی در برخی از برنامه های کنترلی به کاربر بنماید.

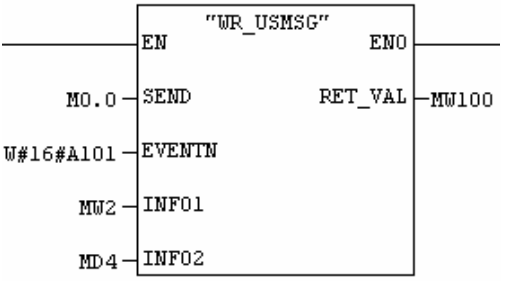
قبل از استفاده از این فانکشن لازم است ابتدا کاربر پیام های خود را در Step 7 معرفی کرده باشد. مسیر زیر:

**Simatic Manager > S7 Program > R.click >Special Object Properties > Messages**

پنجره ای مانند شکل زیرباز می شود که پیام دلخواه را در آن وارد می کنیم. پیام میتواند برای شرایط Incoming یعنی وقوع رخداد یا شرایط Outgoing یعنی برطرف شدن رخداد باشد. لازم است پس از تکمیل این پنجره کدهای داده شده در آن بعنوان Message Number و Error Class را یادداشت کنیم و ازروی آنها ID رخداد را بدست آوریم این IDدر فانکشن SFC51 استفاده خواهد شد. در شکل زیر برای پیام Incoming سطر اول ID= W#16#A101 و برای پیام سطر دوم ID=w#16#A102 خواهد بود عدد 1 بعد از حرف A نشان دهنده Incoming است . پس اگر در سطر اول پیام Outgoing داشتیم کد آن W#16#A001 بود.

| Message identifier | Message type | Error class | Message num | Message ID | Incoming message | Outgoing |
|---|---|---|---|---|---|---|
| WR_USMSG (1) | WR_USMSG | A | 1 | 40961 | interlock 1 is active | |
| WR_USMSG (2) | WR_USMSG | A | 2 | 40962 | interlock 2 is active | |

پس از انجام موارد فوق اکنون بلاک SFC51 را در برنامه صدا میزنیم




ورودی SEND : از جنس BOOL است اگر صفر باشد پیام در بافر Diagnostic و اگر یک باشد در Send Buffer ثبت می شود. بافر Send برای ارسال پیام به سایر ایستگاههایی که روی MPI قرار گرفته اند مانند OP ها بکار می رود.

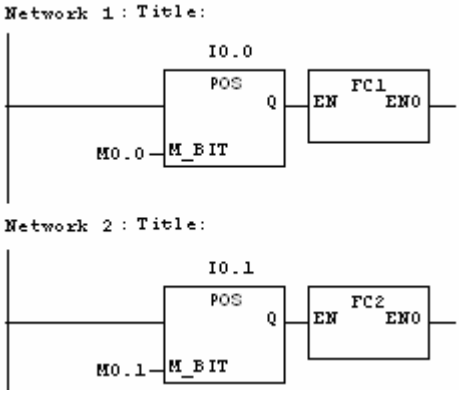
ورودی EVENTN : کد ID که طبق توضیحات قبل مشخص شده می گیرد.

ورودی های Info1 و Info2 برای توضیحات اضافی پیام بکار می روند.

**مثال**

دو اینترلاک I0.0 و I0.1 بدنبال هم فعال می شوند میخواهیم ببینیم از نظر زمانی کدامیک زودتر عمل کرده است. برای این منظور دو پیام مجزا در پنجره Message که در شکل صفحه قبل آورده شد می نویسیم و ID آنها را یادداشت می کنیم. SFC52 را در دو فانکشن FC1 و FC2 صدا میزنیم و در هر کدام یک شماره ID را می دهیم سپس از OB1 با لبه مثبت I0.1 فانکشن FC1 و با لبه مثبت I0.2 فانکشن FC2 را فرا می خوانیم.

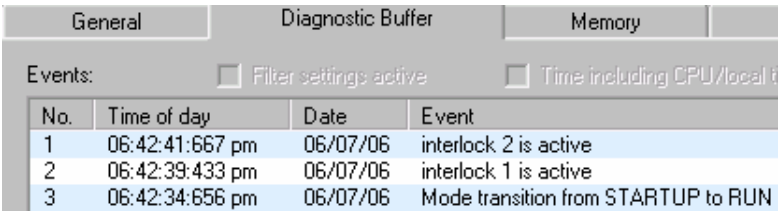
برنامه OB1 بصورت زیر است:




برنامه FC1 و FC2 بصورت زیر می باشد:

| FC2 | FC1 |
|---|---|
| CALL "WR_USMSG"<br>SEND :=FALSE<br>EVENTN :=W#16#A102<br>INFO1 :=MW10<br>INFO2 :=MD12<br>RET_VAL:=MW200 | CALL "WR_USMSG"<br>SEND :=FALSE<br>EVENTN :=W#16#A101<br>INFO1 :=MW10<br>INFO2 :=MD12<br>RET_VAL:=MW200 |

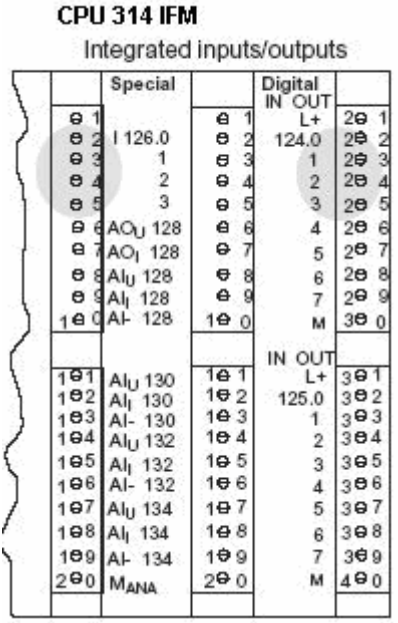
پس از اجرای برنامه با فعال شدن سوئیچهای I0.1 و I0.2 پیام هایی مانند شکل زیر در بافر خواهیم دید.




**تذکر:** اگر در برنامه فوق بجای لبه مثبت از خود کلید برای فعال کردن FC استفاده کنید همه سطرهای بافر با پیغام پر خواهد شد چون CPU دائماً SFC52 را صدا خواهد زد.

### ۱۲-۲-۱۳ فانکشنهای مربوط به CPU های IFM با نام Count

برخی از CPU های S7 بصورت یکپارچه همراه با مدول های I/O خاص عرضه شده و برای مقاصد ویژه مورد استفاده قرار می گیرند. از جمله این CPU ها میتوان به CPU312IFM , CPU314IFM اشاره کرد. کلمه IFM در انتهای کد این CPU ها نشانگر Integrated Function Module می باشد. I/O های کنار این CPU برای کاربردهایی مانند شمارش سریع، اندازه گیری فرکانس و کنترل موقعیت بکار می روند.




این CPU ها با SFB های خاصی بصورت Integrated همراه هستند که فانکشن های آنها را سرویس دهی می کنند. این SFB ها عبارتند از :

- SFB29 HS_Count
- SFB30 Freq_Mes
- SFB38 HSC_A_B
- SFB39 POS

دو مورد آخر صرفاً برای CPU314 IFM کاربرد دارد.

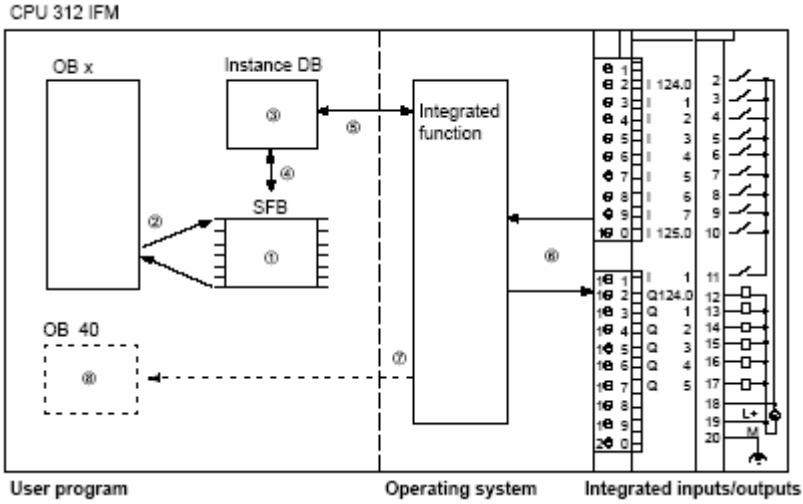
فانکشنهای فوق در این قسمت تشریح میشوند ولی لازم است متذکر شویم که برای کاربرد هر یک از آنها مثالی در ضمیمه ۷ آورده شده و توصیه میشود خواننده پس از مطالعه عملکرد هر فانکشن بلافاصله مثال مربوط به آنرا در ضمیمه بررسی نماید.

بلوک دیاگرام صفحه بعد شکل کلی ارتباط CPU با مدول های یکپارچه کنار آن را نشان می دهد

توجه شود که آدرس I/O ها ثابت بوده و قابل تغییر نمی باشند در پیکر بندی سخت افزار نیز وقتی این CPU را وارد Rack300 در HWconfig می کنیم مشاهده می نماییم که آدرسها از قبل تعیین شده و ثابت هستند. این آدرسها در جدول زیر آورده شده اند.

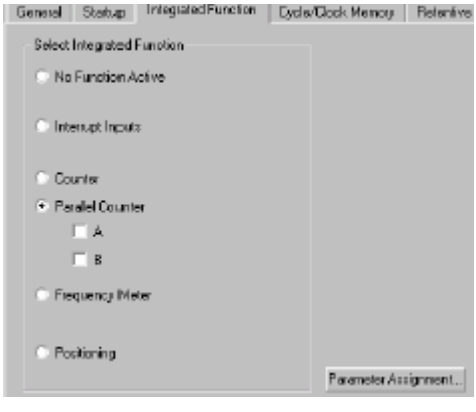
ستون CPU314 IFM جدول را با شکل بالا مقایسه کنید تا اتصالات دقیق تر مشخص گردند.

| CPU 312 IFM | CPU 314 IFM | Function |
|---|---|---|
| I 124.6 | I 126.0 | Digital input up |
| I 124.7 | I 126.1 | Digital input down |
| I 125.0 | I 126.2 | Digital input direction |
| I 125.1 | I 126.3 | Digital input hardware start/stop |
| Q 124.0 | Q 124.0 | Digital output A |
| Q 124.1 | Q 124.1 | Digital output B |



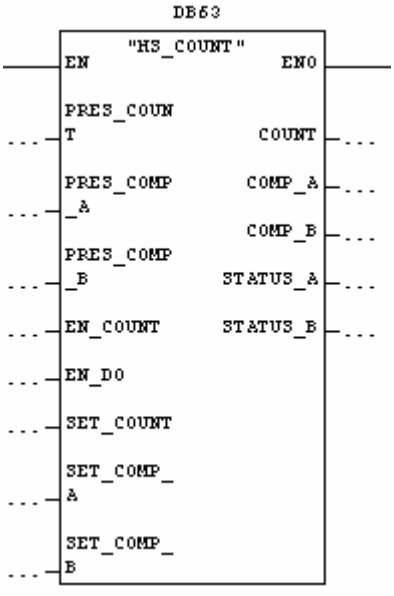

| Slot | Module ... | Order number ... | Fi... | M... | I address | Q addr... |
|---|---|---|---|---|---|---|
| 1 | | | | | | |
| 2 | CPU 314 IFM | 6ES7 314-5AE83-0AB | V1.2 | 2 | 124...135 | 124...129 |
| 3 | | | | | | |

توجه شود که در پیکر بندی سخت افزار برنامه Hwconfig تنظیمات دیگری نیز لازم است برای هر کدام SFB های چهار گانه فوق الذکر بایستی ستینگ مربوطه در پارمترهای CPU در بخش Integrated Function مانند شکل زیر فعال گردد.




با این توضیحات اکنون به تشریح SFB های این خانواده می پردازیم.

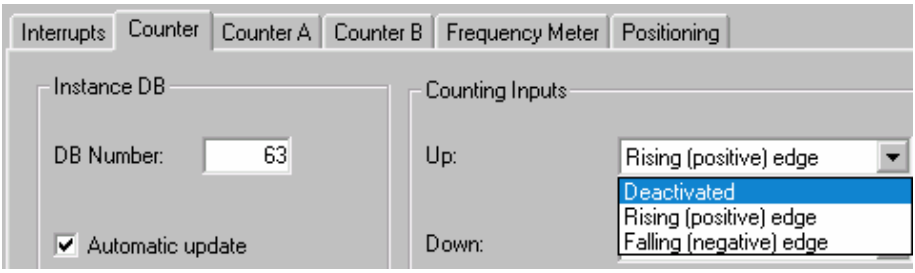
**SFB29 با نام سمبلیک HS_Count**

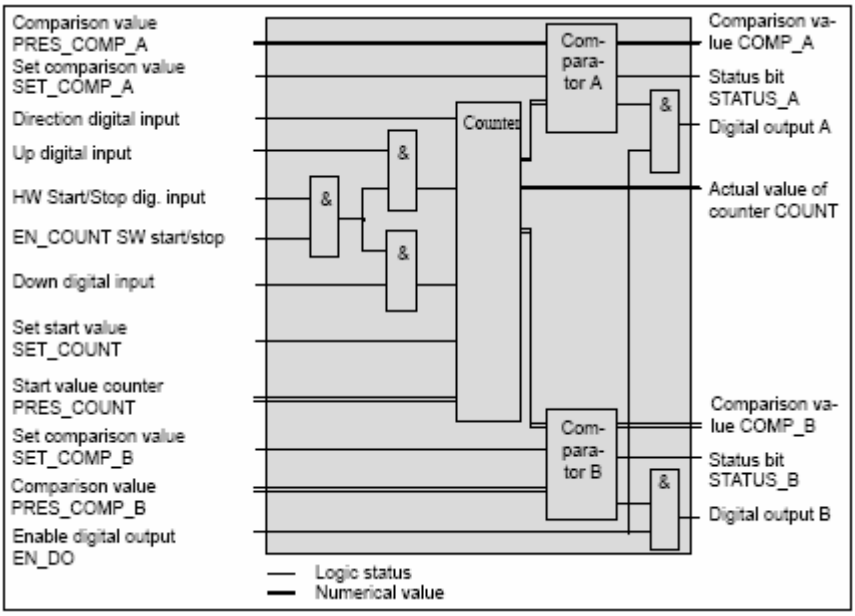



این فانکشن امکان شمارش پالسهایی با فرکانس حداکثر 10 KHZ را برای CPU فراهم میسازد. شمارنده میتواند بصورت افزایشی و کاهشی عمل کند. برای شناخت عملکرد دقیق SFB29 بلوک دیاگرامی در صفحه بعد ترسیم شده است. بطور کلی میتوان گفت که این فانکشن پالسهای سریع را دریافت کرده و بسته به اینکه پالس به کدام ورودی آن متصل باشد شمارنده افزایش یا کاهش می یابد. میتوان شمارنده را با یک مقدار اولیه تنظیم کرد همچنین میتوان مقایسه گرهایی را فعال ساخت تا اگر تعداد پالس به حد معینی رسید خروجی خاصی فعال شود این مطالب در زیر تشریح شده است.

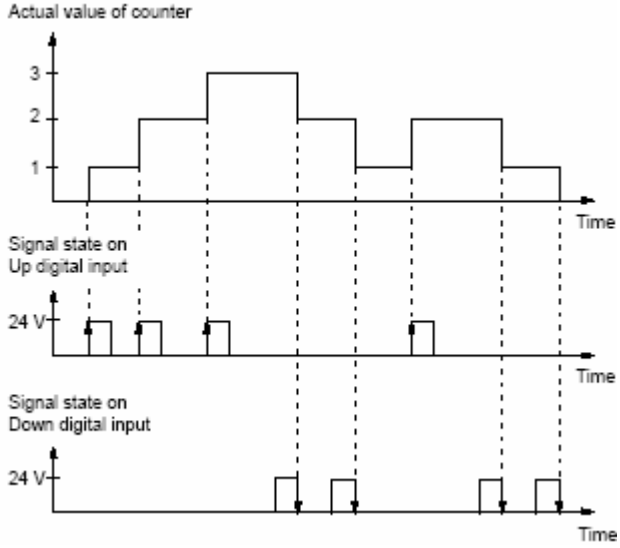
**شمارش پالس**

این کانتر با هر لبه پالس که به ورودی های سخت افزاری ذکر شده در جدول صفحه ۴۷ اعمال شود یک شماره افزایش یا یک شماره کاهش می یابد. بعنوان مثال با لبه مثبت در ورودی I 124.6 مربوط به CPU312IFM مقدار کانتر افزایش پیدا می کند. بعبارت دیگر SFB29 هیچ ورودی برای پالس ها ندارد و پالسها مستقیماً از سخت افزار دریافت می شوند. SFB29 صرفاً به کنترل خروجی ها می پردازد. در پارامترهای این CPU ها در HWconfig بخشی به نامIntegrated Function وجود داردکه در آن عملکرد فانکشن و بسیاری پارامترهای دیگر قابل تنظیم است . در همین قسمت میتوان نوع لبه پالس که بالا رونده یا پایین رونده در نظر گرفته شود را تعیین کرد.






نتیجه شمارش پالس در خروجی Count که بصورت Double Integer است نمایش داده می شود.طبق شکل زیر با رسیدن هر سیگنال Up مقدار Count یکی افزایش و با هر سیگنال Down این مقدار یکی کاهش مییابد.



## روشن و خاموش کردن شمارنده

به دو طریق میتوان شمارنده را فعال یا غیر فعال نمود:

۱. توسط ورودی سخت افزاری که در جدول صفحه ۴۷ آورده شده است. بعنوان مثال با ورودی I125.1 برای CPU312IFM

۲. توسط ورودی EN_Count در فانکشن SFB29

با فعال شدن هر کدام از موارد فوق پالسهای دریافتی در ورودی ها شمارش نخواهند شد.

## دادن مقدار اولیه به شمارنده

مقدار اولیه یا Preset Value را میتوان در ورودی Pres_Count بصورت یک عدد DINT وارد کرد.البته صرفاً وارد کردن این عدد به تنهایی کافی نیست بلکه لازم است لبه مثبتی نیز به ورودی Set_Count داده شود تا کانتر با این مقدار اولیه Set شود.

## تغییر جهت شمارش

در حالت عادی ورودی سخت افزاری Up کانتر را افزایش و ورودی سخت افزاری Down کانتر را کاهش می دهد. میتوان این عمل را بر عکس نمود . اگر به ورودی سخت افزاری Digital input direction (مثلاً ورودی I125.0 در 312IFM) سیگنال صفر اعمال شود جهت شمارش معکوس شده و بارسیدن پالس به ورودی Up مقدار کانتر کاهش می یابد.

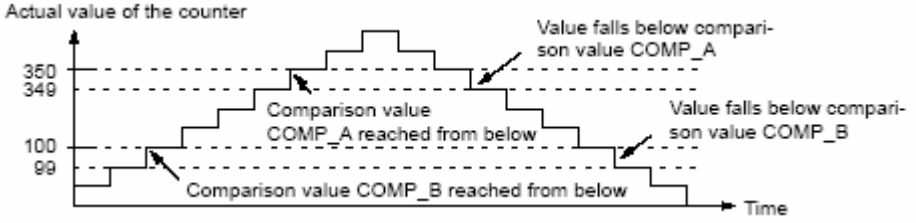
## عملکرد مقایسه گر

کانتر SFB29 دارای دو مقایسه گر داخلی است . این مقایسه گر ها مقدار لحظه ای کانتر را با مقدار از قبل تعیین شده مقایسه و در صورت برقراری شرایط خروجی های خاصی را از SFB فعال می کنند. دو مقایسه گر به نام های A و B وجود داردکه عملکرد آنها مشابه است و میتوان به هر کدام از آنها مقدار جداگانه ای اختصاص داد. Pres_Comp_A برای مقایسه گر A و Pres_Comp_B برای مقایسه گر B مقدار می گیرد. این مقدار بصورت عدد DINT است.برای اعمال این مقادیر به مقایسه گرها لازم است لبه مثبتی به ورودی های Set_Comp_A و Set_Comp_B داده شود.در این شرایط:

- اگر Count>=Comp_A شود بیت خروجی Status_A یک می شود.
- اگر Count>=Comp_B شود بیت خروجی Status_B یک می شود.

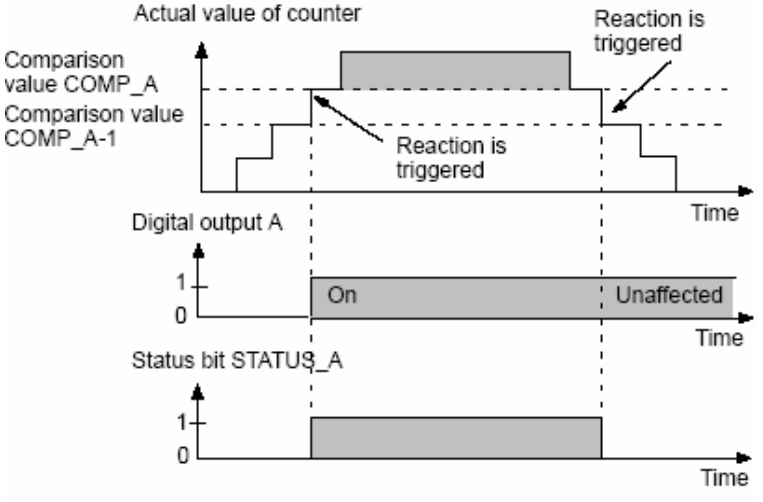
مقادیر Set شده بعنوان مبنا برای مقایسه گرها نیز بصورت DINT در خروجی های Comp_A و Comp_B قابل مشاهده است.

شکل زیر عملکرد مقایسه گر را در حالتی که مقایسه گر A با مقدار ۳۵۰ و مقایسه گر B با مقدار ۱۰۰ تنظیم شده نشان می دهد.نقاطی که در آنها خروجی مقایسه گرها فعال شده اند با فلش نشان داده شده است.




## فعال سازی خروجی ها

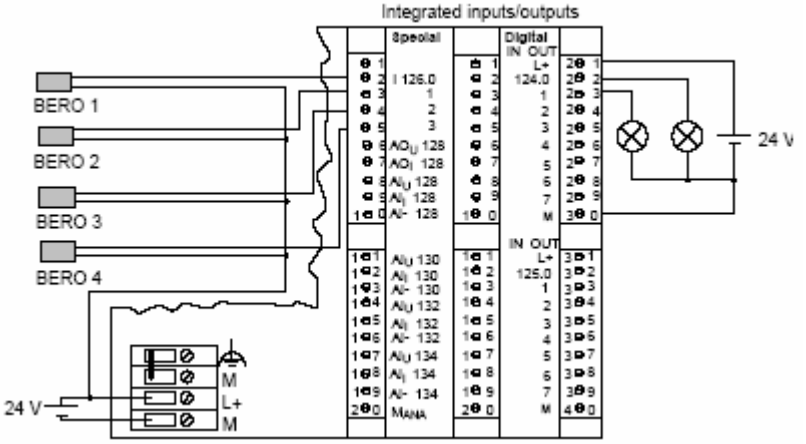
اگر ورودیEN-DO مربوط به SFB29 یک شود در اینصورت خروجی های SFB29 یعنی خروجی های مربوط به مقایسه گر های A و B مستقیماً از مدول Output کنار CPU به فرآیند ارسال می گردد. ولی اگر این ورودی صفر شود خروجی های مذکور شبیه خروجی های دیجیتال معمول میتوانند بکار روند و برنامه نویسی شوند. برای فهم بهتر عملکرد ورودی EN_DO به شکل زیر و وضعیت خروجی Status_A دقت شود:




در صفحات ۳۹۰ و ۳۹۴ و ۳۹۷ کاربردی از این CPU را همراه با SFB29 در یک فرآیند بطری پرکنی نشان میدهد. توصیه میشود خواننده محترم قبل از بررسی فانکشنهای بعدی این برنامه ها را مطالعه نماید.آنچه تا اینجا ذکر شد مربوط به SFB29 بود . در صفحات بعد سایر فانکشنهای CPU های IFM مورد بحث قرار گرفته اند.

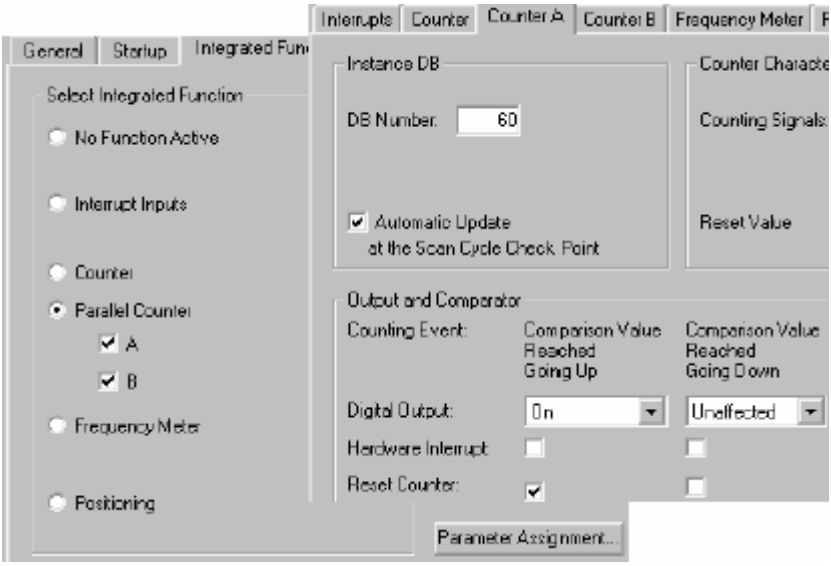
**SFB38 با نام سمبلیک HSC_A_B**

این SFB مخصوص CPU314 IFM می باشد و برای شمارش پالسهایی تا فرکانس 10 KHZ بکار می رود.عملکرد این SFB مشابه SFB29 است از اینرو نیاز به تشریح تفصیلی آن نمی باشد .آنچه که SFB38 را از SFB29 متمایز میسازد اینست که CPU 314 IFM میتوان همزمان دو سری پالس از دو سنسور مختلف دریافت کرد و با استفاده از SFB38 بطور همزمان هر دو شمارش ، مقایسه و خروجی های مستقلی را برای هر کدام فعال نمود.



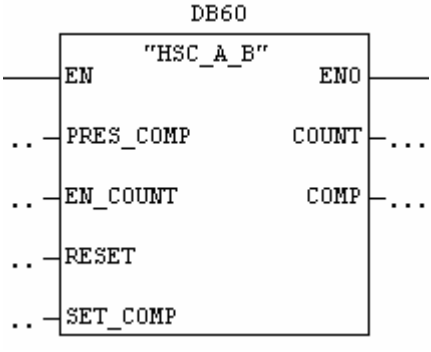

همانطور که در شکل بالا مشاهده می شود دو سنسور به ورودی های I126.0 و I126.1 متصل شده اند که ایندو شمارنده A را بکار می اندازند پالسهای اولی شمارنده را افزایش و پالسهای دومی شمارنده را کاهش می دهد. همینطور سنسورهای ۳ و ۴ که به ورودی های I126.2 و I126.3 متصل شده اند برای افزایش و کاهش شمارنده B استفاده می شوند.خروجی Q124.0 برای شمارنده A و خروجی Q124.1 برای شمارنده B مورد استفاده قرار می گیرد و اگر مقدار شمارش به مقدار مبنای مقایسه برسد خروجی مربوطه روشن می گردد.

قبل از استفاده از این فانکشن لازم است تنظیمات سخت افزاری مربوط به آن در پارامترهای CPU در بخش Integrated Function انجام شده باشد. ابتدا همانطور که در شکل صفحه بعد نشان داده شده Counter A , B را انتخاب کرده سپس با کلیک روی Parameter Assignment پارامترهای مربوط به هر کدام از آنها را تنظیم می کنیم. نکته قابل توجه آنست که برای هر کدام از شمارنده های A و B یک دیتا بلاک جداگانه در نظر گرفته شده است.که پیش فرض شماره های DB60 و DB61 می باشد. پس در برنامه نویسی در هنگام صدا زدن SFB38 اگر همراه با DB60 فراخوان شود شمارنده A بکار خواهد افتاد.




فانکشن SFB29 در LAD بصورت شکل زیر است:

ورودی PRES_COMP مقدار مبنا برای مقایسه است که بصورت یک عدد DINT داده می شود. این مقدار مبنا با لبه مثبتی که به ورودی SET_COMP داده می شود به شمارنده اعمال می گردد. ورودی EN_COUNT برای فعال سازی شمارنده است که یک سیگنال 0 یا 1 می پذیرد. ورودی RESET یک عدد DINT بین -2147483648 تا +2147483647 میتواند باشد . ری ست شدن شمارنده به این مقدار بستگی به تنظیمات سخت افزاری دارد وقتی مقدار شمارنده افزایش یافته و به مقدار مبنای مقایسه برسد شمارنده به مقدار تعیین شده بعنوان Reset Value ریست می گردد.




با توجه به مثال هایی که برای SFB29 در ضمیمه ۷ آمده است عملکرد این فانکشن به دلیل تشابه نیاز به مثال جداگانه ندارد ازاینرو به ادامه بحث در مورد سایرفانکشن های CPU 31x IFM میپردازیم.

**SFB30 با نام سمبلیک FREQ_MES**

این SFB برای اندازه گیری فرکانس پالس هایی تا 10 KHZ که به ورودی های زیر از CPU های IFM داده می شود بکار می رود:

- در CPU312 IFM به ورودی I 124.6
- در CPU314 IFM به ورودی I 126.0

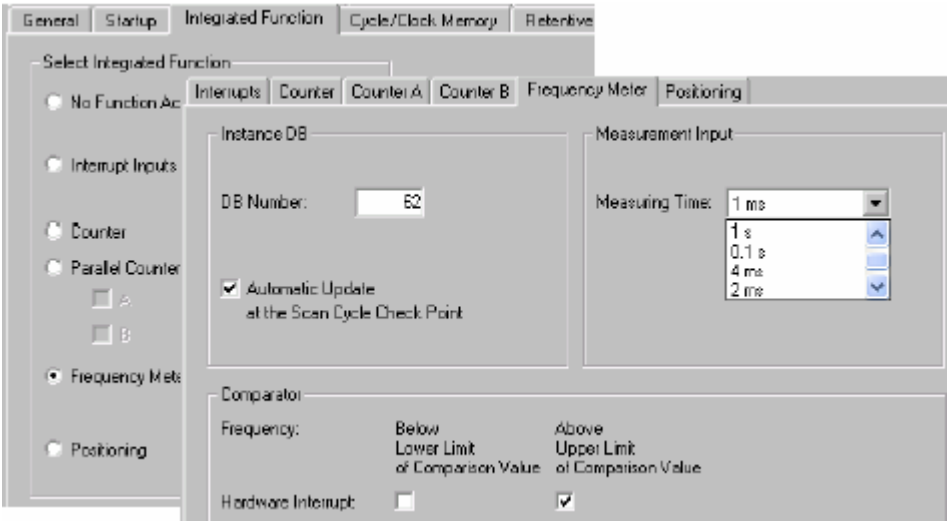
این فانکشن براساس لبه مثبت پالس به یکی از دو روش زیر کار می کند:

روش ۱ : با زمان های نمونه برداری 0.1 S , 1 S , 10 S

روش ۲ : با زمانهای نمونه برداری 1 ms , 2 ms , 4ms

فرکانس پالس با تقسیم تعداد لبه مثبت بر زمان نمونه برداری بدست می آید. بعنوان مثال اگر این زمان ۱ ثانیه انتخاب شود و در مدت یک ثانیه تعداد ۶۵۰۰ پالس برسد در اینصورت فرکانس پالس 6500 HZ خواهد بود.

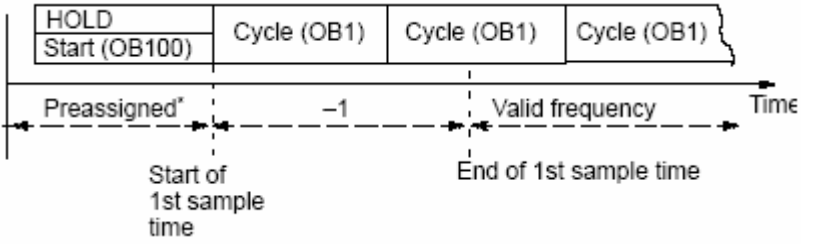
زمان نمونه برداری به فانکشن داده نمی شود بلکه تنظیم آن در پارامترهای CPU در HWconfig در بخش Integrated Function مطابق شکل زیر انجام می شود.ابتدا نوع فانکشن را Frequency Meter انتخاب کرده سپس با کلیک روی Parameter Assignment تنظیم زمان نمونه برداری را انجام می دهیم. در حالتی که از روش ۲ استفاده می کنیم میتوانیم در پارامترهای سخت افزار وقفه سخت افزاری را نیز فعال نماییم که بعداً در این مورد توضیحاتی بیان خواهد شد.



روش ۱ که در آن زمان نمونه برداری بین 0.1s تا 10 s است برای اندازه گیری فرکانسهای بالا بکار می رود دقت آن زیاد و در عین حال کمترین بار گذاری را روی سیکل خواهد داشت.

روش ۲ که زمان نمونه برداری آن بین 1ms تا 4ms است برای اندازه گیری فرکانس های پایین بکار می رود در این روش دقت مناسب بوده و امکان اعمال وقفه سخت افزاری نیز وجود دارد در عین حال بیشترین بار گذاری را روی سیکل اسکن به دنبال دارد.

وقتی CPU استارت می شود یا با مد HOLD شروع بکار می کند در اولین زمان نمونه برداری مقدار فرکانس واقعی نیست و برابر 1- می باشد.در واقع فرکانس از دومین نمونه برداری به بعد محاسبه می گردد.




همانطور که ذکر شد این فانکشن قادر است تا فرکانس 10KHZ را اندازه بگیرد . اگر فرکانس ورودی از این حد بالاتر برود علاوه بر اینکه اعداد نمایش داده شده در خروجی قابل اعتماد نیست ممکن است در برخی شرایط با دخالت Watchdog مربوط به زمان سیکل برای CPU وضعیت Stop پیش بیاید.

میزان Resolution فرکانس اندازه گیری شده با افزایش در زمان نمونه برداری افزایش می یابد مثال های جدول زیر این موضوع را بهتر نشان می دهد.

| Sample Time | Resolution | Example of Positive Edges during 1 Sample Period | Frequency |
|---|---|---|---|
| 0.1 s | فرکانس میتواند در پله های ۱۰ هرتزی محاسبه شود | 900 | 9000 Hz |
| | | 901 | 9010 Hz |
| 1 s | فرکانس میتواند در پله های ۱ هرتزی محاسبه شود | 900 | 900 Hz |
| | | 901 | 901 Hz |
| 10 s | فرکانس میتواند در پله های 0.1 هرتزی محاسبه شود | 900 | 90 Hz |
| | | 901 | 90.1 Hz |

ظاهراً بنظر می رسد که برای داشتن دقت بالا اگر زمان نمونه برداری بزرگ انتخاب شود بهتر است ولی باید توجه داشت که در اینحالت فرکانس اندازه گیری شده در زمانهای طولانی تری (مثلاً هر ۱۰ ثانیه یکبار ) در دسترس خواهد بود. این زمان ممکن است برای سیستم کنترل مناسب نباشد.

در روش اول میزان خطای اندازه گیری با افزایش زمان نمونه برداری کاهش می یابد. جدول زیر میزان این خطا را نشان می دهد روش محاسبه خطا در این روش طبق فرمول زیر می باشد.

$$\text{Max. error in \% of meas. val.} = \frac{0.001\ \text{s} + \dfrac{1}{\text{Frequency in Hz}}}{\text{Sample time in s}} \times 100\ \%$$

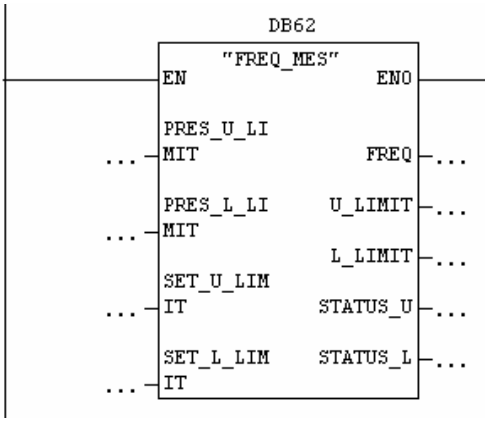
| Frequency | Sample Time | Maximum Measurement Error in % of Measured Value |
|---|---|---|
| 10 kHz | 0.1 s | 1.1 % |
| 10 kHz | 1 s | 0.11 % |
| 10 kHz | 10 s | 0.011 % |

در روش دوم نیز میزان خطا با زمان نمونه برداری نسبت معکوس دارد ولی همانطور که در جدول زیر مشاهده می شود خطا نسبت به روش ۱ بیشتر است. خطا در این روش از فرمول زیر محاسبه می شود:

$$\text{Max. error} = \pm\,\text{frequency in Hz} \times \text{factor in \%} / 100 \pm 0.001\ \text{Hz}$$

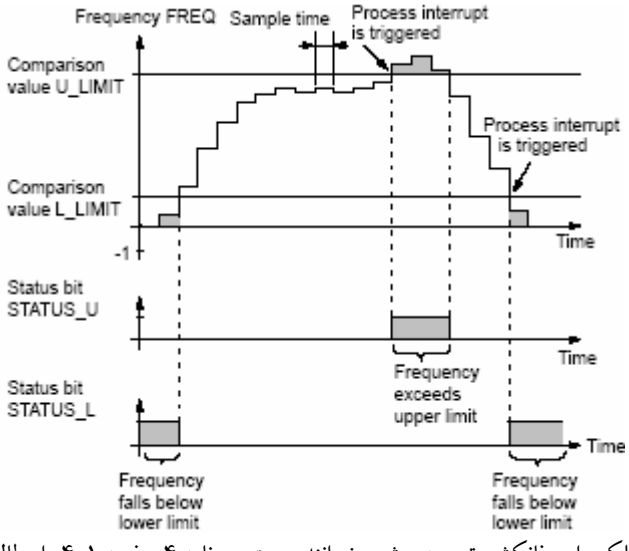
| Frequency | Sample Time | Maximum Measurement Error in % of Measured Value |
|---|---|---|
| 10 kHz | 1 ms | 5 % |
| 10 kHz | 2 ms | 2 % |
| 10 kHz | 4 ms | 1 % |

فرکانس متر SFB30 مجهز به دو مقایسه گر داخلی است یکی برای حد بالا U_Limit و دیگری برای حد پایین L_Limit همانطور که در بلاک این فانکشن( صفحه بعد) مشاهده می شود این حدود به ورودی های Preset بصورت عدد DINT داده می شوندسپس با دادن لبه مثبت به ورودی های Set_U_Limit و Set_L_Limit مقادیر Preset به فانکشن اعمال می شوند.




در خروجی این فانکشن مقدار فرکانس و حد بالا و حد پایین بصورت مقادیر DINT ظاهر می شوند . اگر Frequency > U_Limit باشد خروجی Status_U واگر Frequency < L_Limit باشد خروجی Status_U فعال می شود.

در پارامترهای CPU در صورتی که زمان نمونه برداری از نوع ms باشد میتوان وقفه را همانطور که در شکل قبلی نشان داده شده فعال نمود تا بعنوان مثال وقتی فرکانس از U_Limit بالاتر رفت وقفه سخت افزاری اعمال شده و OB40 اجرا گردد.شکل زیر عملکرد SFB30 را برای شرایط مختلف ترسیم نموده است.

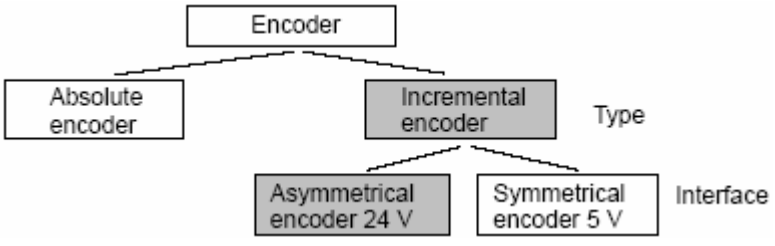



برای فهم بیشتر عملکرد این فانکشن توصیه میشود خواننده محترم برنامه ۴ صفحه ۴۰۱ را مطالعه نماید.

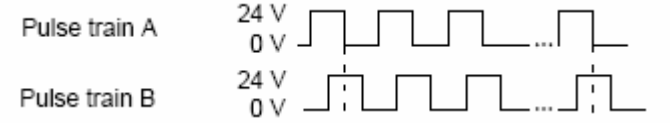
**SFB39 با نام سمبلیک POS**

این فانکشن که مخصوص CPU 314 IFM است برای کنترل موقعیت محوری بصورت حلقه باز بکار می رود و میتواند پالسهایی را از انکودر ۲۴ ولتی Incremental Asymmetrical با فرکانس حداکثر 10 KHZ دریافت کند. این CPU دارای خروجی آنالوگ برای کنترل سرعت درایو و نیز خروجی دیجیتال برای کنترل سرعت در حالتی که موتور بصورت کنتاکتوری کنترل می شود می باشد.

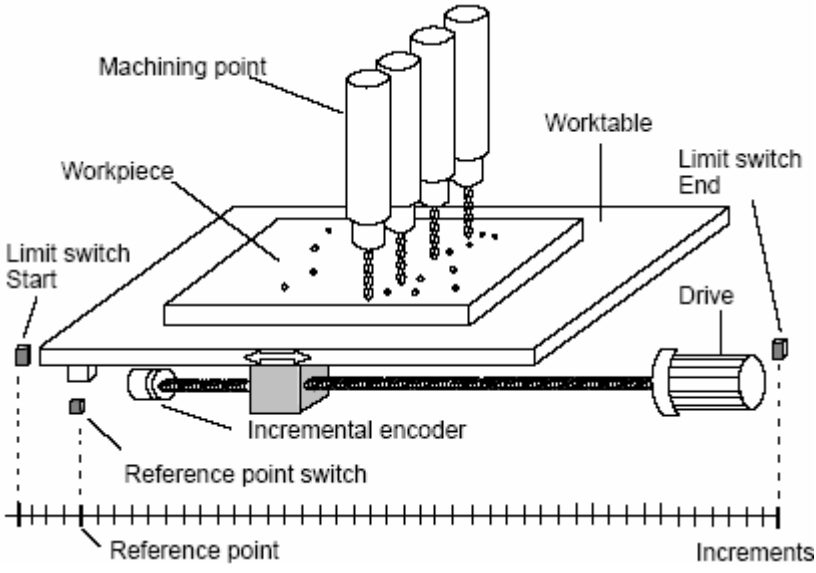
برای فهم درست عملکرد این سیستم در قدم اول آشنایی بادسته بندی انکودرها لازم است. بصورت شکل زیر:




نوعی که برای این فانکشن مناسب است Asymmetrical 24 V می باشد. این انکودر دو قطار پالس به نام های A و B در خروجی خود ظاهر می سازدکه با یکدیگر ۹۰ درجه اختلاف فاز دارند.



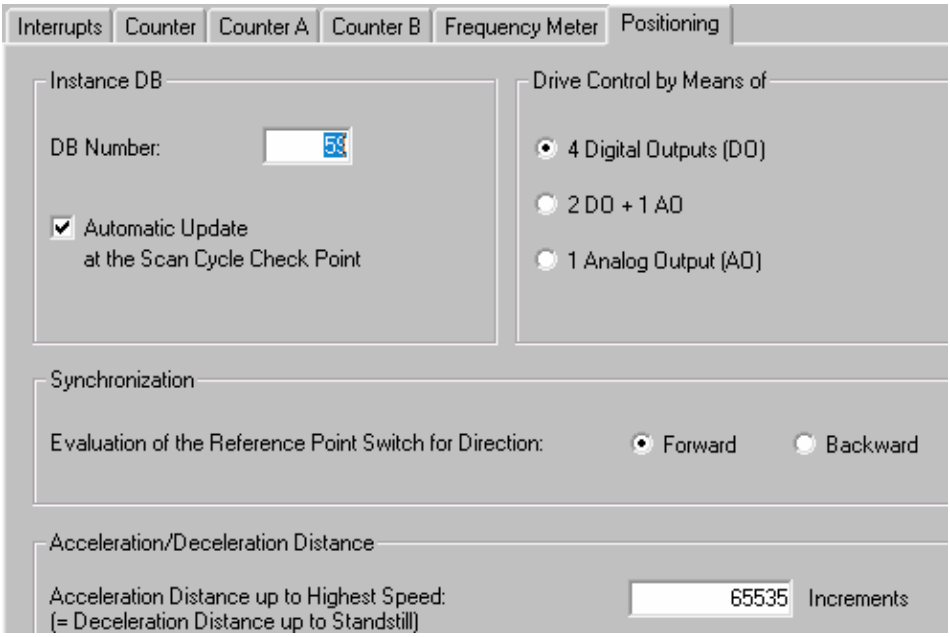

در کنترل موقعیت نیاز به یک نقطه رفرنس وجود دارد تا بقیه موقعیت ها نسبت به آن سنجیده شوند این نقطه توسط انکودر با قطاری از پالس مشخص می گردد. برای فهم بهتر موضوع به ذکر مثالی می پردازیم . همانطور که در شکل صفحه بعد مشاهده می شود قطعه ای لازم است توسط ماشین ابزار سوراخ کاری شود. قطعه روی میز کار قرار می گیرد سپس میز کار توسط موتور و یک شافت مارپیچ بصورت افقی حرکت می کند. ابتدا و انتهای حرکت میز توسط دو لیمیت سوئیچ کنترل می شود. نقطه رفرنس نیز توسط یک سنسور مشخص می گردد وقتی این سنسور یک لبه مثبت بفرستد نشان دهنده آنست که میز در نقطه رفرنس قرار دارد انکودری که روی شافت مارپیچ قرار دارد در نقطه مبنا پالسهایی را میفرستد که بعنوان مبنا تلقی می شوند.




بایستی توجه داشت که دقت در نقطه مبنا بستگی به سرعت حرکت میز و نوع سنسور استفاده شده دارد. بعبارت دیگر ممکن است میز در رفت و برگشت های متوالی در نقطه رفرنس های مختلف با یک اختلاف جزئی بایستد. هر چقدر دقت تکرار سنسور (Repeat Accuracy) بالاتر باشد و سرعت در نزدیکی این نقطه کمتر باشد اختلاف فوق جزئی تر خواهد بود. برای سوئیچ های مکانیکی دقت تکرار 10 میکرو متر و برای سوئیچ های نوری دقت تکرار ۵۰۰ میکرومتر است.

از نیازهای دیگر ماشین کاری مد دستی یا Jog Mode است .این مد دو کاربرد مهم دارد اول اینکه اپراتور در صورت لزوم مثلاً در صورت وقوع اشکال توسط مد دستی میز را به موقعیت مورد نظر هدایت می کند. دوم مسئله سنکرون سازی است که برای CPU 314 IFM مهم است زیرا وقتی این CPU استارت می شودنمیتواند موقعیت را محاسبه کند چون هنوز موقعیت رفرنس را نمی شناسد تا سایر موقعیت ها را نسبت به آن محاسبه کند از اینرو نیاز به شناخت رفرنس دارد که اصطلاحاً به آن سنکرون سازی گفته می شود. برای سنکرون سازی لازم است در حالت Jog Mode محور حرکت کرده تا به نقطه رفرنس برسد و از آنجا سیگنال رفرنس به CPU داده شود.

برای سنکرون سازی لازمست تعیین شود که حرکت نسبت به نقطه رفرنس Forward یا Backward باشد . این تنظیم در پارامترهای CPU در HWconfig مطابق شکل صفحه بعد انجام می گیرد.

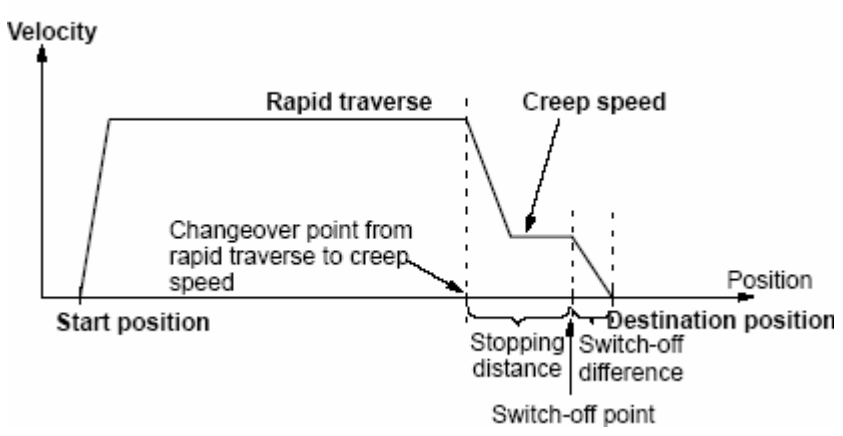



سرعت حرکت محور در حالت Jog Mode قابل تنظیم است و شبیه حالت کار عادی محور است .
بطور کلی کنترل سرعت بستگی به مدار تغذیه موتور دارد ممکن است تغذیه بصورت کنتاکتوری باشد یا اینکه از درایو با کنترل فرکانس (برای موتور آسنکرون یا سنکرون ) استفاده شده باشد. پس دو روش کنترل سرعت برای IFM قابل تعریف است :

۱. روش دو سرعتی که به Rapid Traverse (دوربالا) و Creep Speed (دور پایین) موسوم است.
۲. استفاده از مبدل فرکانس برای کنترل سرعت در رنج دلخواه.

در روش ۱ که سیستم بصورت کنتاکتوری است روش کار مطابق با نمودار صفحه بعد است. این نمودار روش کار را هم برای کار عادی و هم برای حالت Jog Mode نشان می دهد.فرض کنیم نقطه مقصد که قرار است محور تا آنجا حرکت کند و بایستد مشخص است در اینصورت ابتدا محرک با بیشترین سرعت از نقطه شروع حرکت می کند که این حالت همان Rapid traverse است سپس در فاصله مشخصی نسبت به مقصد که این فاصله از قبل مشخص شده محرک به حالت سرعت کم یعنی Creep Speed در می آید. کمی مانده به نقطه

مقصد محرک قطع می شودتا نهایتا محور در موقعیت مورد نظر بایستد. حالت Creep برای دقت بیشتر در رسیدن به نقطه مورد نظر بکار می رود.




نقطه شروع لازم نیست که همان نقطه مبنا باشد و از هر نقطه ای امکان شروع وجود دارد تنها محدودیتی که وجود دارد اینست که اگر فاصله نقطه شروع تا مقصد کوچکتر یا برابر با مسافت Swich off شدن محرک باشد عملاً کنترل موقعیتی اعمال نمی شود.

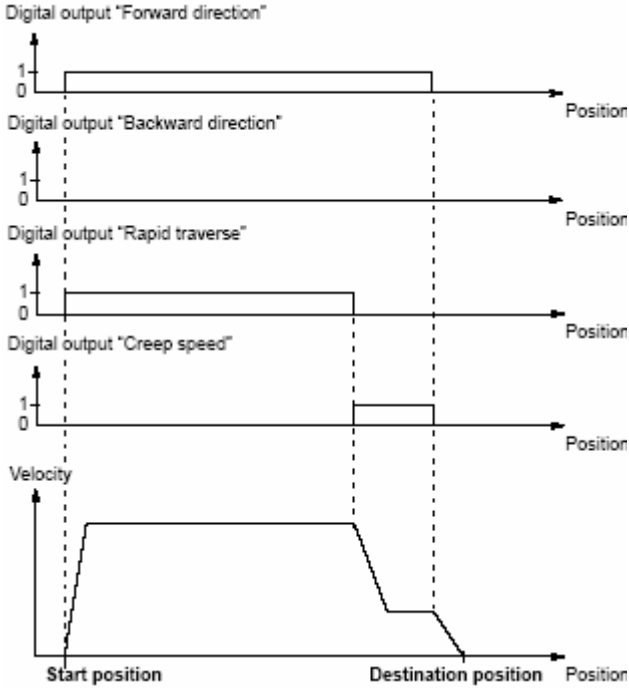
**روش کنترل Rapid/Creep توسط CPU IFM**

در این روش ۴ خروجی دیجیتال برای عملکرد کنترل در نظر گرفته شده است. در پارامترهای Hwconfig نیز همانطور که در شکل قبلی نشان داده شد لازمست گزینه 4 DO فعال شود. این ۴ خروجی عبارتند از :

- یک خروجی دیجیتال برای فعال سازی حالت Rapid
- یک خروجی دیجیتال برای فعال سازی حالت Creep
- یک خروجی دیجیتال برای جهت چرخش Forward
- یک خروجی دیجیتال برای جهت چرخش Backward

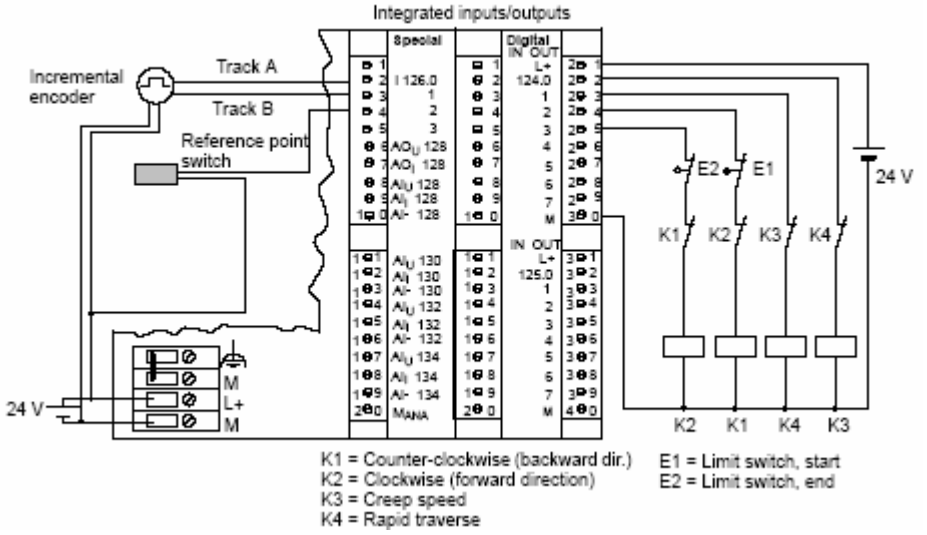
فرض کنیم محرک لازم است بصورت Forward بچرخد دراین حالت سیگنال Forward روشن و سیگنال Backward خاموش است. ابتدا سیگنال Rapid همزمان با Forward فعال شده و به محض رسیدن محور به فاصله معین نسبت به مقصد این سیگنال خاموش و سیگنال Creep فعال می شود.

نمودار صفحه بعد این عملکرد را نشان می دهد.



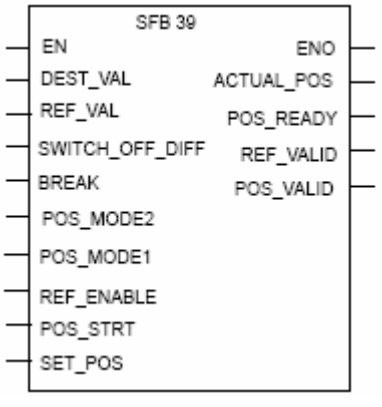

در کنترل موقعیت با سیستم تغذیه کنتاکتوری ورودی و خروجی هایی که روی CPU 314 IFM استفاده می شوند عبارتند از :

- I 126.0 : برای اتصال انکودر Track A
- I 126.1 : برای اتصال انکودر Track B
- I 126.2 : برای اتصال سوئیچ نقطه رفرنس
- Q 124.0 : برای خروجی Creep Speed
- Q 124.1 : برای خروجی Rapid Traverse
- Q 124.2 : برای حرکت Backward
- Q 124.3 : برای حرکت Forward




از نظر الکتریکی کنتاکتور K1 و K2 که مربوط به چپ گرد و راست گرد موتور هستند با یکدیگر اینترلاک دارند با وصل شدن یکی دیگری قطع می شود. همینطور کنتاکتور های K3 و K4 که برای دور کند و دور تند بکار می روند نیز با هم اینترلاک دارند.

پس از انجام اتصالات و تنظیم پارامتر های سخت افزاری فراخوانی SFB39 در برنامه ضروریست .




**ورودی DEST_VAL** : فاصله نقطه مقصد محور بصورت یک عدد DINT

**ورودی REF_VAL** : میتوان مقدار جدید برای فاصله مبنا بصورت یک عدد DINT را به این ورودی داد. این عدد برای سنکرون سازی نرم افزاری بکار می رود . اعمال این عدد به کنترلر با یک شدن ورودی SET_POS صورت می گیرد.

**ورودی SWITCH_OFF_DIFF** : مقدار فاصله از نقطه SWITCH_OFF تا مقصد را بصورت WORD می گیرد.

سه ورودی بالا مقادیر خود را برحسب تعداد پالس انکودر می گیرند. این موضوع در صفحات بعد تشریح خواهد شد.

**ورودی BREAK :** این ورودی بصورت بایت است در سیستم کنتاکتوری برای کنترل حالت Creep یا Rapid بکار می رود.اگر بزرگتر از صفر باشد فقط امکان Creep وجود دارد. در سیستم مبتی بر درایو این ورودی سرعت ماکزیمم را مشخص می کند .

**ورودی POS_MODE2 :**این ورودی که بصورت بیت است حرکت Forward را فعال میسازد.

**ورودی POS_MODE1 :**این ورودی که بصورت بیت است حرکت Backward را فعال میسازد.

**ورودی REF_ENABLE:** بصورت بیت است که برای فعال سازی یا غیر فعال سازی سنکرون سازی نقطه رفرنس بکار می رود.

**ورودی POS_STRT:** بصورت بیت است که برای شروع کار نرمال کنترل موقعیت بکار می رود.

**ورودی SET_POS:** بصورت بیت است و یک شدن آن به معنای پذیرفتن نقطه رفرنس می باشد. بدینطریق سنکرون سازی نرم افزاری اتفاق می افتد.

**خروجی ACTUAL_POS**: این خروجی مقدار واقعی موقعیت را بصورت DINT نشان می دهد.

**خروجی POS_READY**: با کامل شدن عملکرد Positioning یا Jog این خروجی یک می شود و امکان Positioning مجدد فراهم می گردد.لازمست توجه شود که با توجه به منحنی Rapid/Creep در شرایطی که فاصله تا مقصد کمتر یا مساوی فاصله Swith_off باشد در اینصورت خروجی POS_READY بدون تغییر و یک باقی میماند.

**خروجی REF_VALID**: این خروجی نشان دهنده آنستکه سنکرون سازی سخت افزاری انجام شده یا خیر.

**خروجی POS_VALID:** این خروجی نشان دهنده آنست که فانکشن IFM با محورها سنکرون شده است یا خیر . اگر صفر باشد به معنای سنکرون نشدن است در اینحالت عمل Positioning امکان پذیر نیست و فقط امکان Jog Mode وجود دارد.

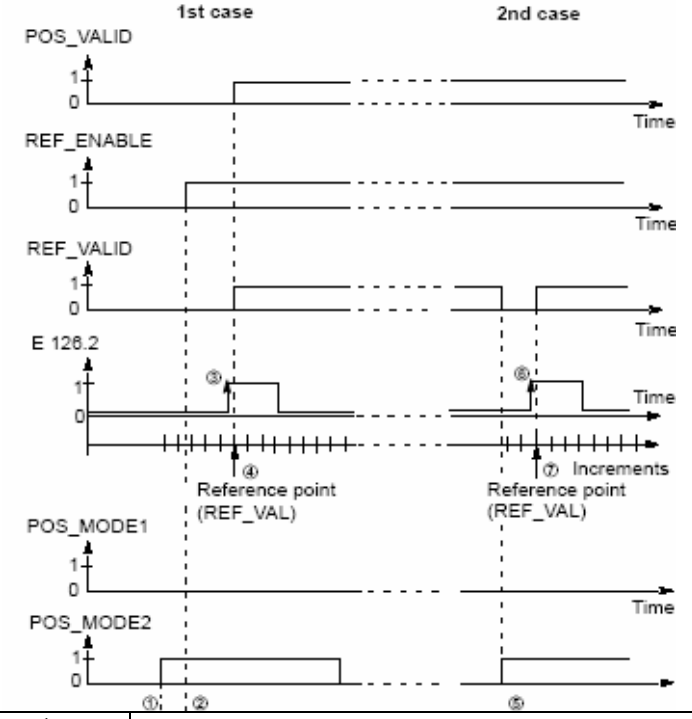
با توجه به نکاتی که در بیان آدرسها ذکر شد دیدیم که دو نوع سنکرون سازی امکان پذیر است:

۱- سنکرون سازی نرم افزاری : این سنکرون سازی توسط SFB39 انجام می شود مقدار داده شده به REF_VAL از طریق ورودی SET_POS بعنوان مبنا به کنترلر اعمال می شود. این عمل در صورتی امکان پذیر است که خروجی POS_READY یک باشد یعنی عمل Positioning کامل شده باشد.پس از انجام سنکرون سازی فوق خروجی POS_VALID یک می شود.

۲- سنکرون سازی سخت افزاری : در این روش مقدار REF_VAL بعنوان نقطه مبنا منظور می شوداگر REF_ENABLE=1 و سیگنال در I126.2 از 0 به 1 برود و جهت حرکت واقعی با آنچه در پارامترهای HWConfig تنظیم شده یکسان باشد.

در شکل زیر دو روش سنکرون سازی فوق نشان داده شده است.اعداد مربوطه در جدول زیر آن آمده است.

Case2 : شروع سنکرون سازی توسط مد Jog Case1 : شروع سنکرون سازی با فعال شدن REF_ENABLE

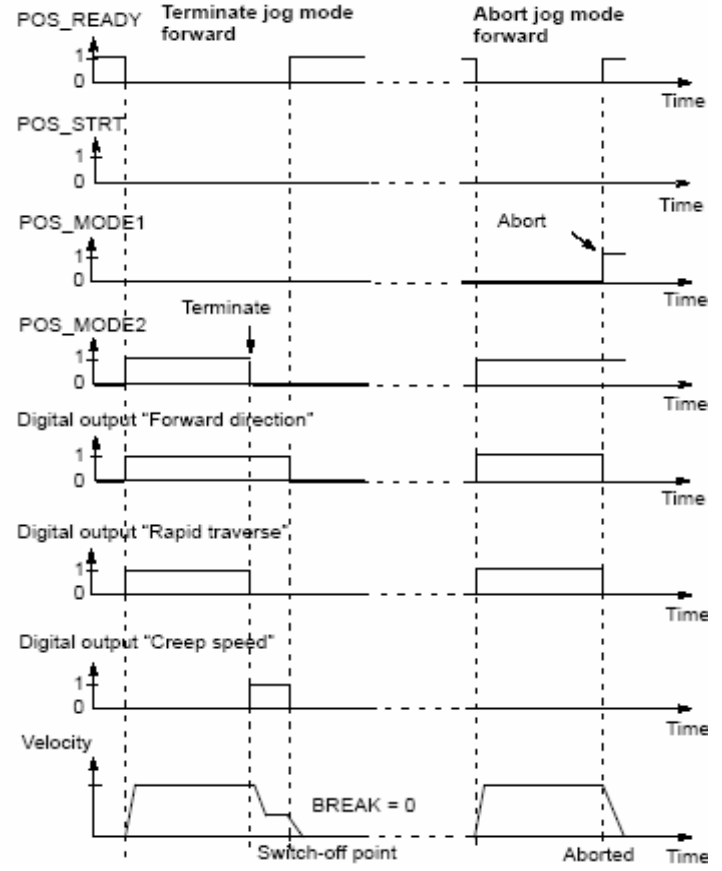



| | | |
|---|---|---|
| Case 1 | 1 | مد Jog بصورت Forward از طریق ورودی POS_MODE2 شروع می شود |
| | 2 | وضعیت سیگنال در ورودی REF_ENABLE از 0 به 1 رفته و REF_VALID=0 |
| | 3 | لبه مثبت در ورودی I126.2 که به سوئیچ رفرنس متصل است اتفاق بیفتد. |
| | 4 | نقطه رفرنس جدید REF_VAL توسط IFM پذیرفته می شود. |
| Case 2 | 5 | POS_VALID و REF_ENABLE دارای وضعیت 1 هستند. مد Jog توسط POS_MODE2 و REF_VALID=0 بصورت Forward شروع می شود. |
| | 6 | لبه مثبت در ورودی I126.2 که به سوئیچ رفرنس متصل است اتفاق بیفتد. |
| | 7 | نقطه رفرنس جدید REF_VAL توسط IFM پذیرفته می شود. |

**انجام Jog Mode**

عملیات Jog Mode یعنی تغییر در موقعیت در بازه ای از 2147483648- تا 2147483647+ برحسب فاصله Increment ( پالس انکودر) . این عملیات تحت شرایط مختلف بصورت Forward یا Backward امکان پذیر است همچنین امکان Terminate و Abort کردن نیز وجود دارد . در حالت Terminate سرعت از وضعیت Creep به وضعیت Switch-Off می رود. ولی در وضعیت Abort بلافاصله تمام خروجیها صفر می شوند.این دو حالت در جدول و شکل زیر نشان داده شده اند.

| Forward | Backward | Terminate | Abort |
|---|---|---|---|
| POS_READY=1<br>POS_MODE1=0<br>POS_MODE2=1<br>POS_STRT=0 | POS_READY=1<br>POS_MODE1=1<br>POS_MODE2=0<br>POS_STRT=0 | POS_MODE1=0<br>POS_MODE2=0 | POS_MODE1=1<br>POS_MODE2=1 |



### اجرای عملکرد Positioning

شرایط مختلف برای مراحل مختلف Positioning در جدول زیر آمده است.

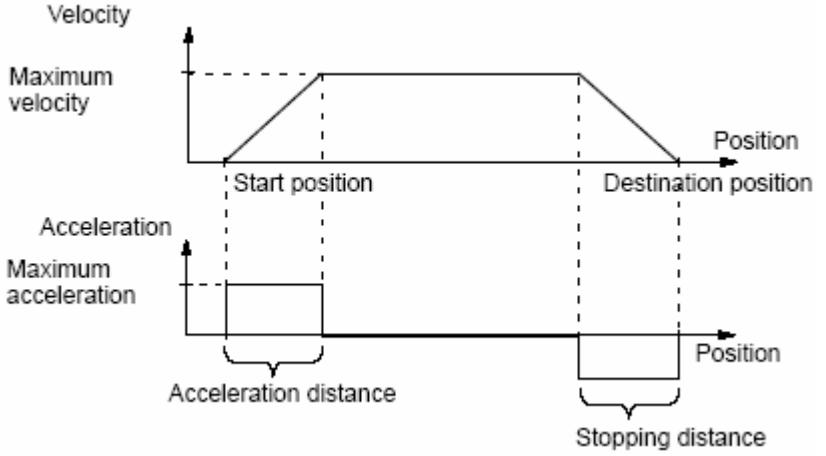
| Positioning Operation | Parameter |
|---|---|
| Start positioning operation | POS_READY = 1<br>Rising edge at POS_STRT<br>POS_MODE1 = 0<br>POS_MODE2 = 0 |
| Positioning operation running | POS_STRT = 1 |
| Terminate | Falling edge at POS_STRT |
| Abort | POS_MODE1 = 1<br>POS_MODE2 = 1 |

تا اینجا مطالبی که بیان شد عمدتاً مربوط به Positioning در سیستم مبتنی بر کنتاکتور و ۲ سرعته بود. در ادامه بحث به نکات تکمیلی مربوط به سیستم مبتنی بر مبدل فرکانس می پردازیم.

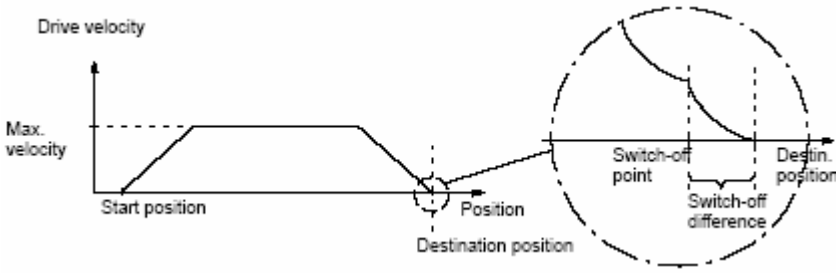
### Positioning با تغذیه سیستم از طریق مبدل فرکانس

مبدل فرکانس برای تغذیه و تغییر دور موتورهای آسنکرون و سنکرون بکار می رود و توسط آن سرعت در رنج وسیعی قابل تغییر است و ۲ حالته نیست بعلاوه شتاب نیز قابل کنترل و تغییر است.

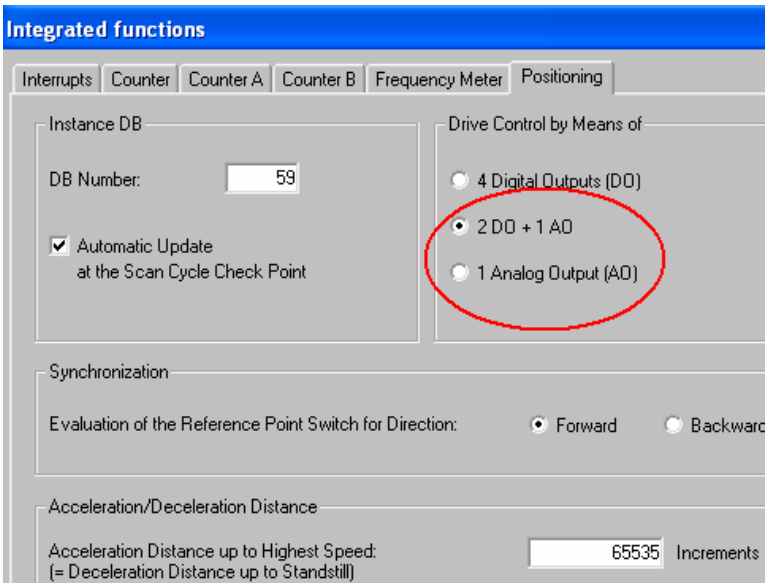
قبل از هر چیز در این روش لازمست حد ماکزیمم سرعت و حد ماکزیمم شتاب مشخص گردد.



در اینجا نیز شبیه سیستم کنتاکتوری مسئله فاصله نقطه Switch-Off تا نقطه مقصد وجود دارد. شکل زیر این موضوع را نشان می دهد.



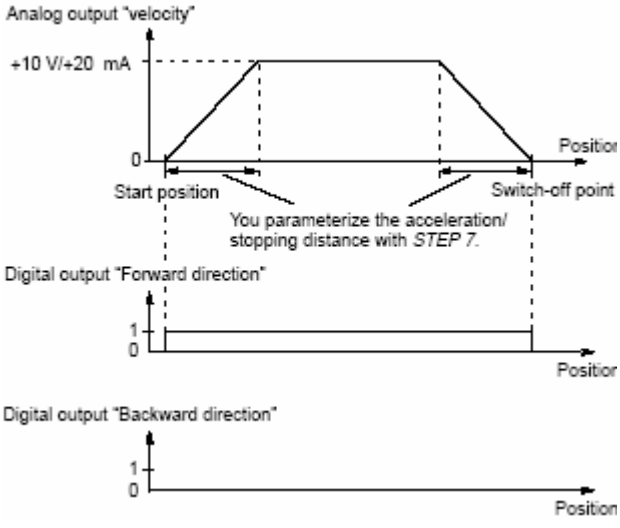

کنترل مبدل فرکانس از طریق IFM به دو طریق زیر انجام می شود . این دو روش در پارامترهای HWconfig تنظیم می گردد.



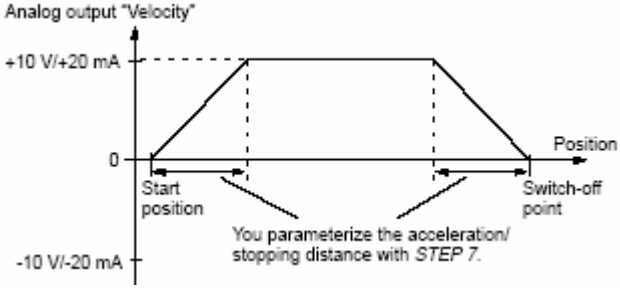

این دو روش همانطور که در شکل مشاهده می شود عبارتند از :

۱- استفاده از یک آنالوگ خروجی همراه با دو دیجیتال خروجی. سیگنال آنالوگ میتواند بین 0-10V یا 0-20mA باشد. این سیگنال سرعت را برای درایو مشخص می کند. خروجی آنالوگ فوق به آدرس PQW128

متصل می گردد. دو خروجی دیجیتال برای جهت چرخش استفاده می شوند Q124.2 برای Backward و خروجی Q124.3 برای Forward بکار می رود.



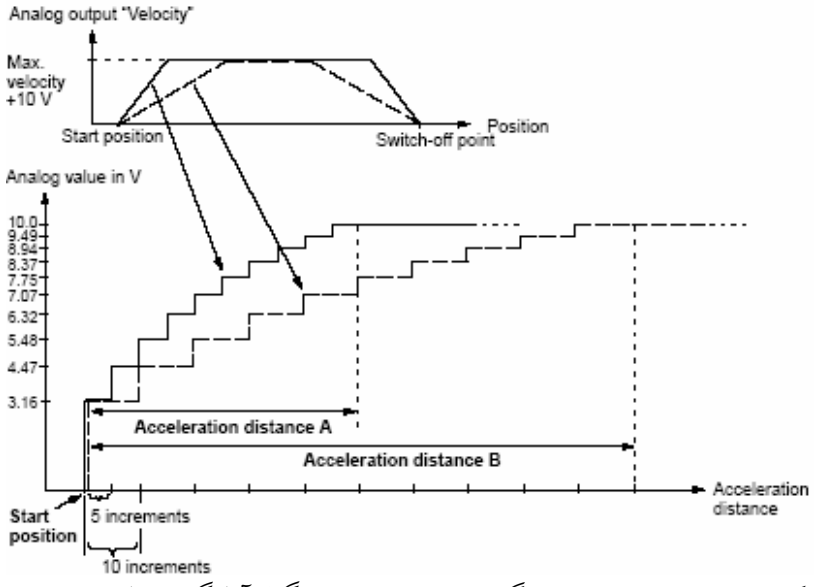

۲- استفاده از یک سیگنال آنالوگ ±10V یا ±20mA که در اینحالت همزمان سرعت و جهت چرخش مشخص می شوند. بدیهی است علامت+ برای چرخش Forward و علامت – برای چرخش backward استفاده می گردد. این خروجی نیز به آدرس PQW128 روی IFM متصل می گردد.




**ماکزیمم شتاب**

میزان شتاب گیری بستگی به پارامتری دارد که در HWConfig (شکل صفحه قبل )داده می شود و عنوان آن Acceleration Distance Up to Highest Speed است.

برای فهم بیشتر موضوع به شکل زیر توجه کنید:




همانطور که مشاهده می شود شیب شتاب گیری با عرض پله های سیگنال آنالوگ ارتباط دارد. این عرض برابر با یک دهم پارامتر تنظیمی در HWConfig است. با توجه به منحنی فوق واضح است که پارامتر تنظیمی در Hwconfig برای منحنی A حدود ۵۰ و برای منحنی B حدود ۱۰۰ بوده است.

**ماکزیمم سرعت**

ماکزیمم سرعت بعنوان v در فرمول زیر داده می شود و از روی آن پارامتر Break محاسبه شده به ورودی SFB39 اعمال می شود.بدینطریق SFB39 میتواند ماکزیمم سرعت را کنترل کرده و به درایو اعمال نماید.

برای ولتاژ رابطه روبرو را داریم که ماکزیمم سرعت بصورت ولتاژ به v داده می شود.

$$v = \frac{10\ \mathrm{V}}{256} \times (256 - \mathrm{BREAK})$$

و برای جریان رابطه روبرو را داریم که ماکزیمم سرعت بصورت میلی آمپر به v داده می شود.

$$v = \frac{20\ \mathrm{mA}}{256} \times (256 - \mathrm{BREAK})$$

برای فهم بیشتر عملکرد این فانکشن خواننده محترم می تواند به برنامه ۵ و ۶ در صفحات ۴۰۴ و ۴۱۱ مراجعه نماید. با این توضیحات بحث در مورد فانکشنهای CPU31x IFM را خاتمه داده و در ادامه به فانکشنهای مربوط به CPU های 31xC می پردازیم. از آنجا که برخی مفاهیم که در این قسمت ذکر شد مانند نقطه رفرنس و Jog Mode و .... در مورد 31xC نیز مشابه است از تکرار آنها خودداری می نماییم.

### ۱۲-۲-۱٤ فانکشن های مربوط به CPU های 31xC

فانکشن های این خانواده با نام **Tec_Func** برای CPU های کمپکت خانواده 300 که در انتهای کد آنها حرف C وجود دارد بکار می رود شامل CPU های زیر:

| CPU | Order no. | Firmware Version | Hardware Version |
|---|---|---|---|
| CPU 312C | 6ES7312-5BD01-0AB0 | V2.0.0 | 01 |
| CPU 313C | 6ES7313-5BE01-0AB0 | V2.0.0 | 01 |
| CPU 313C-2 PtP | 6ES7313-6BE01-0AB0 | V2.0.0 | 01 |
| CPU 313C-2 DP | 6ES7313-6CE01-0AB0 | V2.0.0 | 01 |
| CPU 314C-2 PtP | 6ES7314-6BF01-0AB0 | V2.0.0 | 01 |
| CPU 314C-2 DP | 6ES7314-6CF01-0AB0 | V2.0.0 | 01 |

شکل روبرو CPU314C-2DP را نشان می دهد.

CPU های 31xC فانکشن های زیر را بصورت Built In عرضه می کنند :

- Positioning
- Counting
- Frequency Measurement
- Pulse Width Modulation
- Point to Point Communication
- PID Control

ارتباط P-t-P مرتبط با مباحث شبکه است و در این کتاب مورد بحث قرار نمی گیرد. کنترل PID نیز با سه فانکشن SFB41 و SFB42 و SFB43 انجام می شود که عملکرد آنها مشابه FB41 و FB42 و FB43 است که در جلد اول کتاب تشریح گردید. در اینجا به بیان سایر موارد می پردازیم. باز متذکر میشویم که بهتر است خواننده پس از مطالعه عملکرد هر فانکشن مثال مربوط به آنرا در ضمیمه ۷ ببیند.

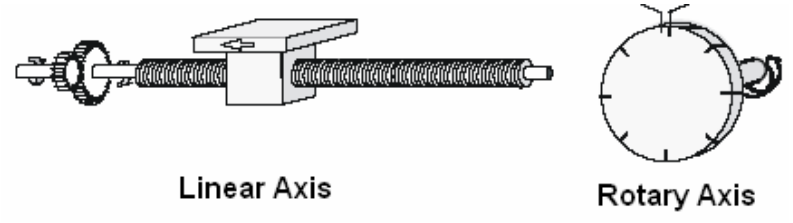
توانایی CPU های 31xC برای ارائه فانکشنهای فوق متفاوت است جدول زیر موضوع را بهتر نشان می دهد.

| | Positioning | Counting | Frequency Meas. | PWM | P-t-P Comm. | PID Control |
|---|---|---|---|---|---|---|
| CPU 312C | - | Yes | Yes | Yes | - | - |
| CPU 313C | - | Yes | Yes | Yes | - | Yes |
| CPU 313C-2 DP | - | Yes | Yes | Yes | - | Yes |
| CPU 313C-2 PtP | - | Yes | Yes | Yes | Yes | Yes |
| CPU 314C-2 DP | Yes | Yes | Yes | Yes | - | Yes |
| CPU 314C-2 PtP | Yes | Yes | Yes | Yes | Yes | Yes |

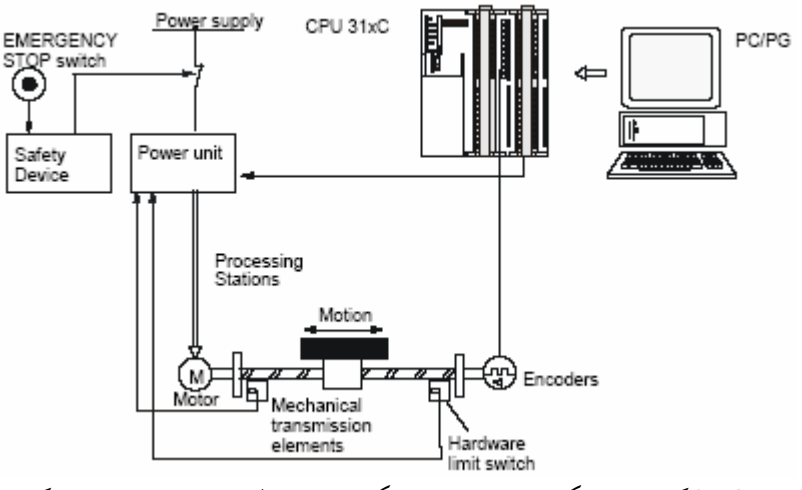
**SFB44 با نام سمبلیک Analog**

بطور کلی برای Positioning در CPU31xC شبیه IFM دو روش وجود دارد که یکی مبتنی بر خروجی آنالوگ و دیگری مبتنی بر خروجی دیجیتال است. SFB44 برای Positioning از طریق خروجی آنالوگ استفاده می شود.این خروجی ها ±20mA , ± 10V , 0-20 mA , 0-10V می باشند.

توسط این خروجی های آنالوگ می توان موتورهای آسنکرون تغذیه شده با مبدل فرکانس یا موتورهای Servo-drive را فرمان داد تا موقعیت کنترل شود. دو نوع محور قابل کنترل است Linear و Rotary




موقعیت محور توسط انکودر Incremental 24V بصورت پالس به CPU اعمال می شود. شکل زیر اجزای سخت افزاری کنترل موقعیت توسط CPU31xC را نشان می دهد.




در CPU های 31xC که تمامی فانکشن ها را ساپورت میکنند دو ترمینال X1 و X2 وجود دارد که برای اتصال ورودی و خروجی هاست نام و شماره پین های کانالهای این دو ترمینال در جدول صفحه بعد آمده است.

توجه شود که موارد فوق شماره آدرس ورودی نیست بلکه شماره اتصال است. در واقع شماره آدرس کانالهای

این CPU برخلاف نوع IFM ثابت نیست و در HWConfig قابل تغییر است.

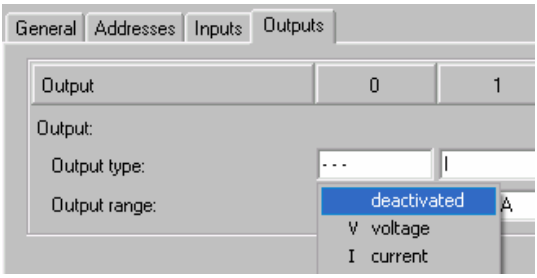
| X1 | | | X2 | | |
|---|---|---|---|---|---|
| Connection | Name/Address | Function | Connection | Name/Address | Function |
| 1 | - | Not connected | 1 | 1 L+ | 24-V supply for inputs |
| 2 | AI 0 (V) | - | 2 | DI+0.0 | Encoder signal A |
| 3 | AI 0 (I) | - | 3 | DI+0.1 | Encoder signal B |
| 4 | AI 0 (C) | - | 4 | DI+0.2 | Encoder signal N |
| 5 | AI 1 (V) | - | 5 | DI+0.3 | Length measurement |
| 6 | AI 1 (I) | - | 6 | DI+0.4 | Reference point switch |
| 7 | AI 1 (C) | - | 7 | DI+0.5 | - |
| 8 | AI 2 (V) | - | 8 | DI+0.6 | - |
| 9 | AI 2 (I) | - | 9 | DI+0.7 | - |
| 10 | AI 2 (C) | - | 10 | - | Not connected |
| 11 | AI 3 (V) | - | 11 | - | Not connected |
| 12 | AI 3 (I) | - | 12 | DI+1.0 | - |
| 13 | AI 3 (C) | - | 13 | DI+1.1 | - |
| 14 | AI R_P | - | 14 | DI+1.2 | - |
| 15 | AI R_N | - | 15 | DI+1.3 | - |
| 16 | AO 0 (V) | V output of converter | 16 | DI+1.4 | - |
| 17 | AO 0 (I) | A output of converter | 17 | DI+1.5 | - |
| 18 | AO 1 (V) | - | 18 | DI+1.6 | - |
| 19 | AO 1 (I) | - | 19 | DI+1.7 | - |
| 20 | MANA | Analog ground | 20 | 1M | Chassis ground |
| 21 | - | Not connected | 21 | 2 L+ | 24-V supply for outputs |
| 22 | DI+2.0 | - | 22 | DO+0.0 | - |
| 23 | DI+2.1 | - | 23 | DO+0.1 | - |
| 24 | DI+2.2 | - | 24 | DO+0.2 | - |
| 25 | DI+2.3 | - | 25 | DO+0.3 | - |
| 26 | DI+2.4 | - | 26 | DO+0.4 | - |
| 27 | DI+2.5 | - | 27 | DO+0.5 | - |
| 28 | DI+2.6 | - | 28 | DO+0.6 | CONV_EN: Converter enable |
| 29 | DI+2.7 | - | 29 | DO+0.7 | CONV_DIR: Direction signal* |
| 30 | 4 M | Chassis ground | 30 | 2 M | Chassis ground |
| | | | 31 | 3 L+ | 24-V supply for outputs |
| | | | 32 | DO+1.0 | - |
| | | | 33 | DO+1.1 | - |
| | | | 34 | DO+1.2 | - |
| | | | 35 | DO+1.3 | - |
| | | | 36 | DO+1.4 | - |
| | | | 37 | DO+1.5 | - |
| | | | 38 | DO+1.6 | - |
| | | | 39 | DO+1.7 | - |
| | | | 40 | 3 M | Chassis ground |
| V : Voltage I: Current C: Common | | | | | |

به مواردی که با رنگ خاکستری نشان داده شده دقت شود چون در بحث های بعدی به آنها خواهیم پرداخت.

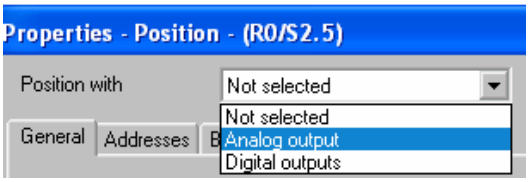
**تنظیمات HWconfig**

برای Positioning تنظیمات زیر لازم است در HWconfig انجام شود:

۱-ابتدا در اسلات زیر CPU روی AI5/AO2 کلیک کرده و وضعیت خروجی AO 0 را Disable کنید .

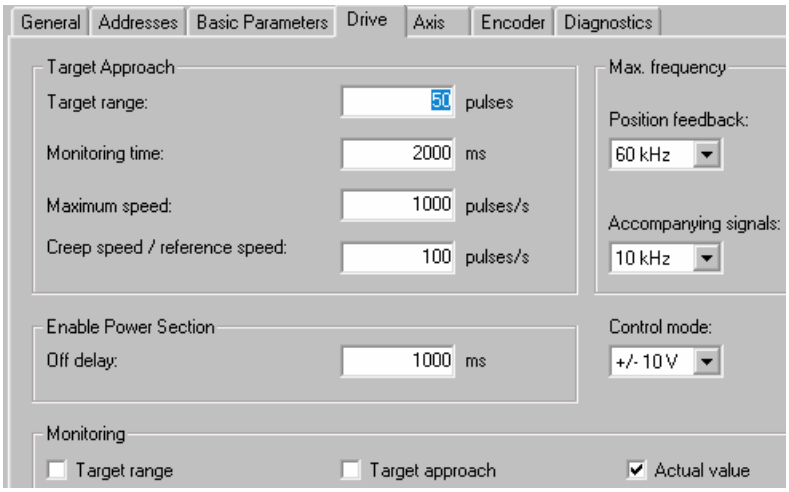



۲-با کلیک روی Position در اسلات زیر CPU نوع کنترل Position را Analog Output انتخاب کنید.




۳-با انجام مرحله ۲ پارامترهای Drive و Axis و Encoder و Diagnostics به پنجره اضافه می شوند.بترتیب آنها را مطابق با نیاز پروسه تنظیم نمایید.

۴-برای پارامتر های درایو پنجره زیر را داریم که با مقادیر پیش فرض پر شده است.




مهمترین تنظیمات در پنجره مزبور عبارتست از :

**Target Range** : عددی بر حسب پالس و حداکثر تا 200,000,000 میتواند بگیرد. این عدد نشاندهنده بازه مسافت برای نقطه مقصد است.

**Monitoring Time** : زمانی است که CPU برای مانیتور کردن مقدار واقعی موقعیت بکار می برد و بین صفر تا 200,000 میلی ثانیه قابل تنظیم است. این زمان بایستی به اندازه کافی باشد تا مطمئن شویم درایو در طول این زمان به گشتاور قفل کننده (Holding Torque) خواهد رسید. بعنوان مثال فرض کنید درایو با مقدار آنالوگ 0.5 V شروع به کار می کند و سرعت ماکزیمم آن 10,000 پالس بر ثانیه و شتاب آن 1000 پالس بر مجذور ثانیه باشد. در این شرایط مقدار 0.5 V معادل 500 پالس بر ثانیه است. با تقسیم این سرعت بر شتاب زمان معادل 0.5 ثانیه بدست می آید. در اینحالت درایو تا زمان 0.5 ثانیه سپری نشود حرکت نمی کند بنابر این زمان Monitoring Time بایستی بزرگتر از 0.5 ثانیه منظور شود.

**Maximum Speed** : عددی بر حسب پالس بر ثانیه است و ماکزیمم سرعت قابل اعمال به درایو را نشان می دهد. این پارامتر لازم است بر حسب پالسهای انکودر محاسبه شود. در محاسبات انکودر ضریب کاهش دور توسط گیربکس نیز بایستی اعمال گردد. سرعت ماکزیمم بر حسب Pulse/s از حاصلضرب زیر بدست می آید:

*۴ x(تعداد پالس انکودر در هر دور) x(ضریب گیربکس) x(سرعت نامی درایو rev/s)*

بعنوان مثال اگر دور نامی درایو 300 rpm ( معادل 50 rev/s) و گیربکس نیز وجود نداشته باشد و تعداد پالس انکودر در هر دور 500 پالس باشد در اینصورت سرعت ماکزیمم برابر است با :

50 x 1 x 500 x 4 = 100,000 pulse/s

**Creep/Reference Speed** : عددی بر حسب پالس بر ثانیه است و میتواند تا یک دهم سرعت ماکزیمم باشد.این پارامتر برای کاهش سرعت در موقعیت ترمز یا رسیدن به نقطه رفرنس بکار می رود.

**Off Delay** : تاخیر از لحظه ای که سیگنال Run برداشته می شود تا زمانی که درایو غیر فعال می شود را نشان می دهد و میتواند بین صفر تا 100,000 میلی ثانیه باشد ( در پله های ۴ میلی ثانیه ای گرد می شود). این پارامتر برای کنترل ترمز مهم است زیرا برای ترمز گیری باید اطمینان از پایین بودن سرعت وجود داشته باشد تا انرژی جنبشی سیستم بتواند توسط ترمز جذب شود.

**Max Frequency** : فرکانس ماکزیمم انکودر (فیدبک) در اینجا قابل تنظیم است و میتواند بین 1 تا 60KHZ باشد. یاد آوری می شود که در نوع IFM این فرکانس حداکثر 10 KHZ بود.

**Control Mode** : برای انتخاب نوع سیگنال آنالوگ خروجی ( ولتاژ یا جریان مورد نظر)

۵- پارامترهای بخش Axis در شکل صفحه بعد نشان داده شده اند و اهم آنها عبارتند از :

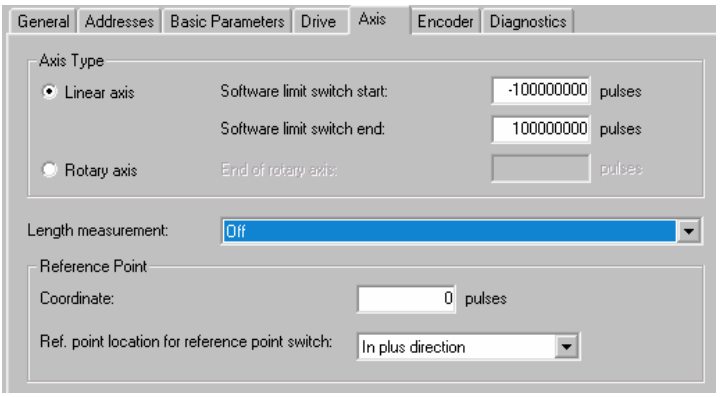
**Axis Type** : در این قسمت با توجه به فرآیند نوع موقعیت سنجی که خطی است یا دورانی را انتخاب می کنیم.

**Software Limit Switch** : در اینجا محدوده کار با توجه به ابعاد محور در نوع خطی تعریف می شود . شروع و انتها بر حسب پالس داده می شوند.

**End of Rotary Axis** : در نوع دورانی این گزینه تعدا پالس را در موقعیت نهایی (که یک دور نزدیک به کامل شدن است ) را برحسب پالس نشان می دهد. آنچه در عمل بعنوان موقعیت نهایی منظور می شود یک پالس کمتر از مقدار وارد شده در اینجاست . بعنوان مثال اگر این مقدار عدد 1000 باشد در چرخش مثبت بین 999 تا 0 و در چرخش منفی بین 0 تا 999 پالس خواهد بود.

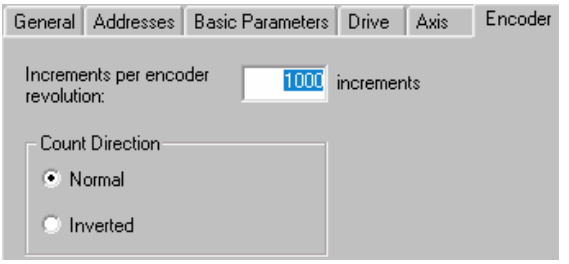
**Length Measurement** : این گزینه امکان اندازه گیری فاصله را در حالت Run فراهم میسازد. در حالت پیش فرض غیر فعال است میتوان آنرا بگونه ای تنظیم کرد تا براساس لبه مثبت یا منفی سیگنال داده شده به ورودی دیجیتال Length Measurement اندازه گیری فاصله انجام شود.

**Reference Point** : برای سنکرون سازی بکار می رود یعنی مشخص می کند که در موقعیت رفرنس چند پالس از انکودر دریافت می شود.بعلاوه در اینجا میتوان مشخص کرد که موقعیت نسبت به نقطه مبنا افزایش یا کاهش یابد (پیش فرض افزایش است)



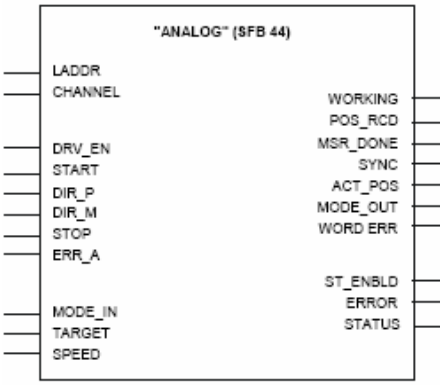

۶- پارامترهای بخش Encoder طبق شکل زیرظاهر می شوند. لازم است با توجه به نوع انکودر تعداد پالسی که در هر دور میفرستد را در اینجا وارد نمود.

**Count Direction** : در حالت نرمال پالسها بصورت صعودی Increment می شوند. درحالت Inverted پالسها بصورت نزولی Decrement خواهند شد.



**صدا زدن SFB44**

پس از انجام تنظیمات HWconfig فانکشن SFB44 را در برنامه صدا میزنیم .

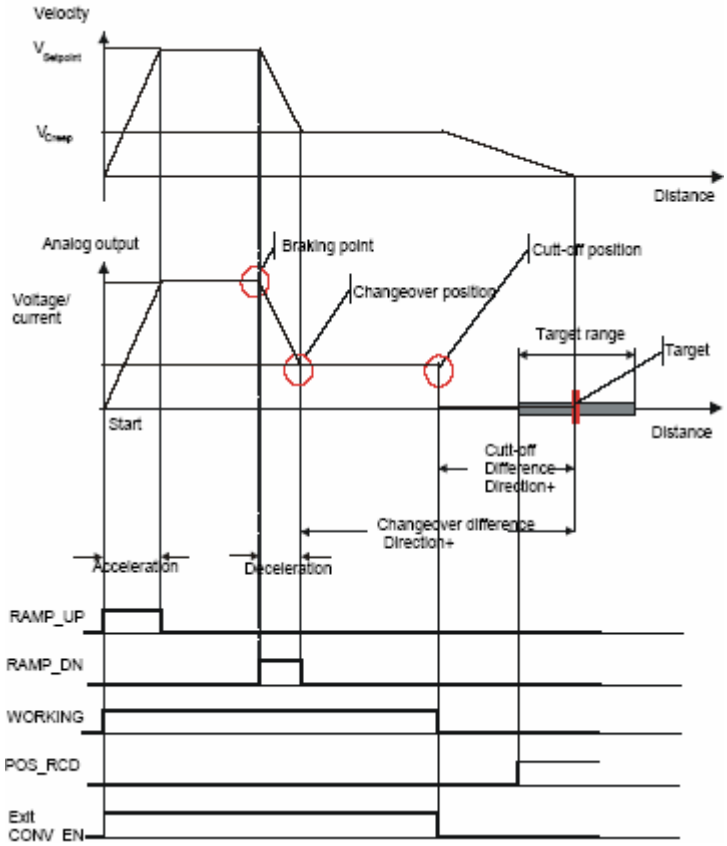



عملکرد این فانکشن در نمودار شکل صفحه بعد آمده است باتوجه به این نمودار می بینیم که:

پس از شروع و رسیدن سیگنال Start با سیگنال RAMP_UP سیستم شتاب مثبت می گیرد.تا اینکه سرعت به نقطه Setpoint برسد. در این نقطه سیگنال شتاب برداشته می شود و سیستم تحت سرعت ثابت حرکت می کند تا به نقطه ترمز برسد. نقطه ترمز توسط توسط CPU محاسبه شده و از آن نقطه به بعد سیگنال RAMP_DN فعال می شود وسرعت سیستم با شتاب منفی کاهش می یابد. تا به نقطه Creep Speed برسد در این نقطه سیگنال شتاب منفی برداشته می شود و سیستم با سرعت کم حرکت می کند . با رسیدن به نقطه Cut-offدرایو قطع می شود و سیگنال خروجی working که تا این لحظه یک بود صفر می شودسپس به اندازه Cut-off Difference که در پارامترها تنظیم شده است حرکت ادامه می یابد تا به نقطه مقصد برسد. پس از رسیدن به نقطه مقصد خروجی POS_REC یک می گردد.

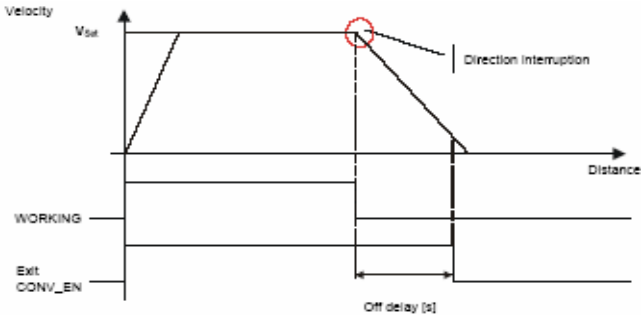
خروجی Conv_En برای فعال یا غیر فعال سازی مبدل ( درایو ) یا کنترل ترمز بکار می رود. این خروجی در نقطه Start فعال و در انتهای Run یعنی در نقطه Cut-off یا وقتی V setpoint صفر شود برداشته می شود. ترمز گیری در این نقطه صورت می گیرد.

در حالتی که از ولتاژ 0-10V یا جریان 0-20mA همراه با سیگنال جهت چرخش استفاده می شود خروجی Conv_Dir نشانگر جهت چرخش است که اگر 0 باشد Forward و اگر 1 باشد Backeard است.




**زمان Off Delay**

طبق شکل زیر از زمانی که سیگنال Run برداشته می شود (خروجی Working) خاموش ولی سیگنال Conv_En به اندازه زمان Off delay وجود دارد پس از آن درایو غیر فعال می شود.



## ورودی های SFB44

| Input Parameters | | | | | |
|---|---|---|---|---|---|
| **Parameters** | **Data type** | **Address (Instance DB)** | **Description** | **Range of values** | **Default** |
| LADDR | WORD | 0 | Submodule I/O address | CPU-specific | 310 hex |
| CHANNEL | INT | 2 | Channel number | 0 | 0 |
| STOP | BOOL | 4.4 | Stopping the run | TRUE/FALSE | FALSE |
| ERR_A | BOOL | 4.5 | ERR_A is used to acknowledge external errors (positive edge) | TRUE/FALSE | FALSE |
| SPEED | DINT | 12 | The axis is accelerated to Vsetpoint. Speed change during run is not possible. | Creep speed up to 1.000.000 pulses/s | 1.000 |

| Input Parameters not Connected to the Blocks (Statical Data) | | | | | |
|---|---|---|---|---|---|
| **Parameters** | **Data type** | **Address (Inst. DB)** | **Description** | **Range of values** | **Default** |
| **ACCEL** | DINT | 30 | Acceleration<br>Change during run is not possible. | 1 to 100,000 $^{2}$ pulses/s | 100 |
| **DECEL** | DINT | 34 | Deceleration<br>Change during run is not possible. | 1 to 100,000 $^{2}$ pulses/s | 100 |
| **CHGDIFF_P** | DINT | 38 | Changeover difference plus | 0 to +10 $^{8}$ Pulses | 1.000 |
| **CUTOFF-DIFF_P** | DINT | 42 | Cut-off difference plus | 0 to +10 $^{8}$ Pulses | 100 |
| **CHGDIFF_M** | DINT | 46 | Changeover difference minus | 0 to +10 $^{8}$ Pulses | 1.000 |
| **CUTOFF-DIFF_M** | DINT | 50 | Cut-off difference minus | 0 to +10 $^{8}$ pulses | 100 |

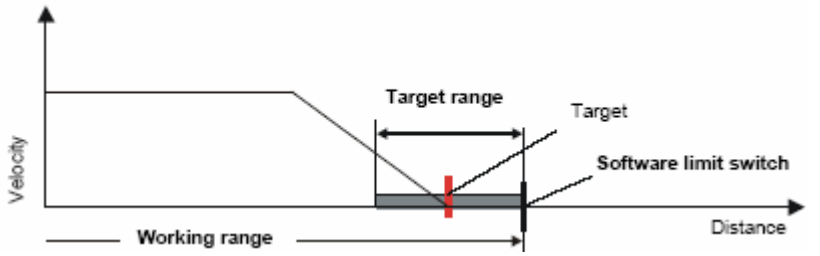
## خروجی های SFB44

| Output Parameters | | | | | |
|---|---|---|---|---|---|
| **Parameters** | **Data type** | **Address (Instance DB)** | **Description** | **Range of values** | **Default** |
| **WORKING** | BOOL | 16.0 | Run is busy | TRUE/FALSE | FALSE |
| **ACT_POS** | DINT | 18 | Actual position value | $-5x10^{8}$ to $+5x10^{8}$ | 0 |
| **MODE_OUT** | INT | 22 | Enabled/configured operating mode | 0, 1, 3, 4, 5 | 0 |
| **ERR** | WORD | 24 | External error:<br>• Bit 2 : missing pulse monitoring<br>• Bit 11: traversing range monitoring<br>• Bit 12: working range monitoring<br>• Bit 13: actual value monitoring<br>• Bit 14: target approach monitoring<br>• Bit 15: target area monitoring<br>• The other bits are reserved | Every bit<br>0 or 1 | 0 |
| **ST_ENBLD** | BOOL | 26.0 | The CPU sets Start Enabled if all of the following conditions are met:<br>• Faultless parameter assignment (PARA = TRUE)<br>• No STOP pending (STOP = FALSE)<br>• No external error occurred (ERR = 0)<br>• Drive Enable is set (DRV_EN = 1)<br>• No positioning run active (WORKING = FALSE). Exception: Jog mode | TRUE/FALSE | TRUE |
| **ERROR** | BOOL | 26.1 | Run start/resume error | TRUE/FALSE | FALSE |

| | | | | | |
|---|---|---|---|---|---|
| **STATUS** | WORD | 28 | Error ID | 0 to FFFF hex | 0 |
| **Output Parameters not Connected to the Blocks (Statical Data)** | | | | | |
| **Parameters** | **Data type** | **Address (Inst. DB)** | **Description** | **Range of values** | **Default** |
| **PARA** | BOOL | 54.0 | Axis is configured | TRUE/FALSE | FALSE |
| **DIR** | BOOL | 54.1 | Current/last sense of direction<br>• FALSE = Forward (plus direction)<br>• TRUE = Reverse (minus direction) | TRUE/FALSE | FALSE |
| **CUTOFF** | BOOL | 54.2 | Drive in cut-off range (starting at the cut-off position, up to the start of the next run) | TRUE/FALSE | FALSE |
| **CHGOVER** | BOOL | 54.3 | Drive in changeover range (after reaching changeover position, up to the start of the next run) | TRUE/FALSE | FALSE |
| **RAMP_DN** | BOOL | 54.4 | Drive is ramped down (starting at the braking point, up to changeover position) | TRUE/FALSE | FALSE |
| **RAMP_UP** | BOOL | 54.5 | Drive is ramped up (from start to final speed) | TRUE/FALSE | FALSE |
| **DIST_TO_GO** | DINT | 56 | Actual distance to go | $-5x10^8$ to $5x10^8$ | 0 |
| **LAST_TRG** | DINT | 60 | Last/actual target<br>• Absolute incremental approach: At run start LAST_TRG = actual absolute target<br>• Relative incremental approach: The distance at run start is the distance specified in LAST_TRG = LAST_TRG of the previous run +/- (TARGET). | $-5x10^8$ to $+5x10^8$ pulses | 0 |
| **BEG_VAL** | DINT | 64.0 | Length Mesurement Initial Value | | 0 |
| **END_VAL** | DINT | 68.0 | Length Mesurement End Value | | 0 |
| **LEN_VAL** | DINT | 72.0 | Length | | 0 |
| **JOB_REQ** | BOOL | 76.0 | Job Request | | FALSE |
| **JOB_DONE** | BOOL | 76.1 | New job Can be Started | | TRUE |
| **JOB_ERROR** | BOOL | 76.2 | Job Error | | FALSE |
| **JOB_ID** | INT | 78.0 | Job Identification Number | | 0 |
| **JOB_STAT** | WORD | 80.0 | Job Error Code | | W#16#0 |
| **JOB_VAL** | DINT | 82.0 | Job Value | | 0 |

## بازه کاری

انجام عملیات در بازه ای بعنوان Software Limit اتفاق می افتد که در HwConfig تنظیم شده است. در صورت خروج از این بازه فقط با Jog mode میتوان آنرا به محدوده درست بر گرداند.



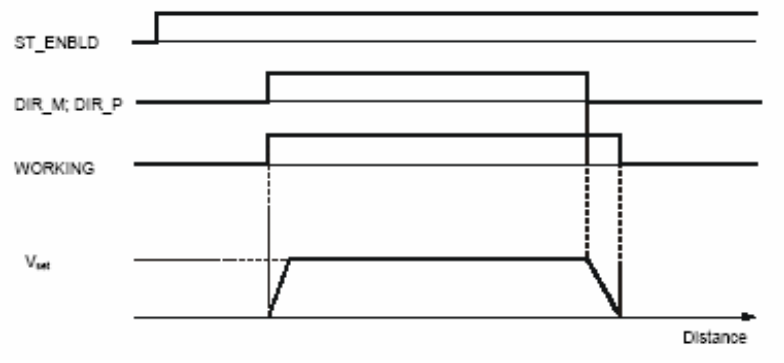

## قطع کردن Run

اگر ورودی Stop در فانکشن یک شود یا اگر ورودی DRV_EN غیر فعال شود عملیات Run قطع می شود.

**انجام Jog Mode**

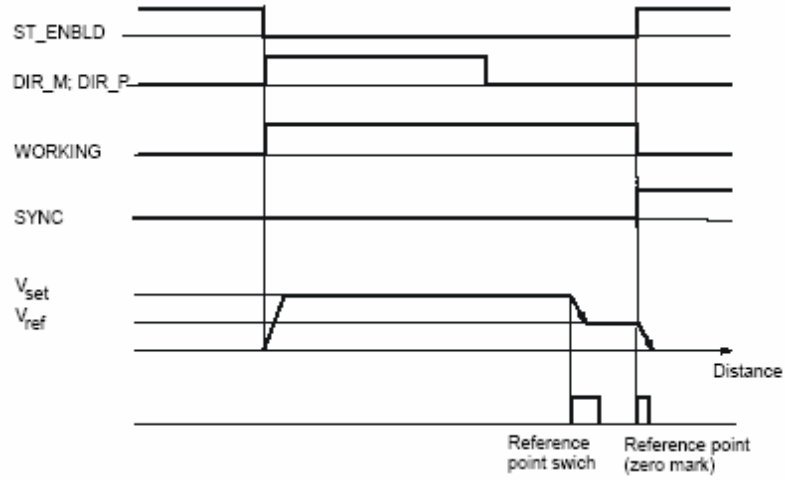
با تنظیم ورودی های روبرو در SFB44 عملیات Jog Mode انجام خواهد شد عملکرد Jog Mode مطابق نمودار زیر است.

| | |
|---|---|
| DRV_EN | True |
| DIR_M یا DIR_P | True |
| MODE_IN | 1 |




**سنکرون سازی با نقطه رفرنس**

نقطه مبنا لازم است به سیستم کنترل شناسانده شود.این نقطه معمولاً توسط یک سوئیچ حس می گردد. ورودی ها برای این عمل شبیه جدول بالا فقط Mode_In=3 می باشد . پس از انجام سنکرون سازی سیگنال خروجی SYNC فعال می گردد.

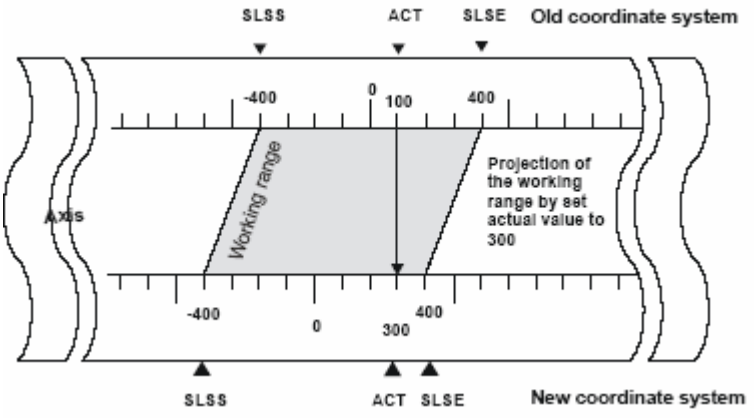


### Set کردن نقطه رفرنس

بدون انجام عملیات سنکرون سازی میتوان مقداری را برای نقطه مبنا تنظیم کرد. این مقدار بصورت تعداد پالس به پارامتر Job_Val از DB_Instance داده می شود و مقدار موقعیت Actual به اندازه اختلاف با Job_Val جابجا می گردد.فرض کنید مقدار موقعیت واقعی 100 پالس است و حدود دامنه کاری طبق تنظیمات Hwconfig بصورت زیر باشد:

Software Limit Switch Start (SLSS) = -400
Software Limit Switch End (SLSE) = 400

اگر در این شرایط Job_Val=300 داده شود در اینصورت تمام مقادیر SLSS و SLSE به اندازه 200(که اختلاف ۳۰۰ با مقدار واقعیست) جابجا می شوند. شکل زیر:



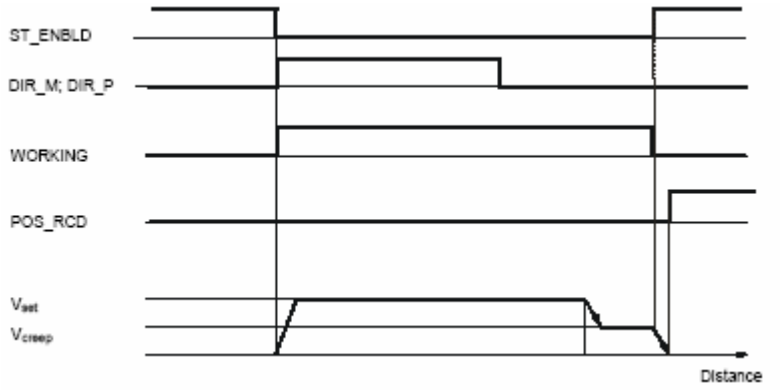

### عملیات Incremental نسبی

در این روش درایو نسبت به آخرین نقطه مقصد (Last_Trg) به اندازه تعیین شده در جهت مشخص شده حرکت می کند. این عملیات برای داشتن دقت بالاتر کاربرد دارد.

پارامترهای ورودی SFB44 شبیه صفحه قبل است با این تفاوت که Mode_In=4 است و فاصله نیز بر حسب پالس به ورودی Target داده می شود. نمودار عملکرد بصورت شکل صفحه بعد است.

### عملیات Incremental مطلق

در این روش مقصد بصورت مطلق و نه نسبت به مقصد قبلی تعیین می گردد در اینحالت Mode_in=5 است.

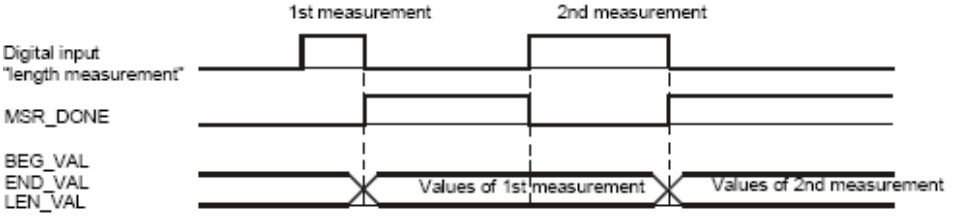



**اندازه گیری طول Length Measurement**

در پارامتر های HWconfig بیان شد که میتوان با فعال کردن تنظیم مربوط به ورودی دیجیتال Length Measurement که بصورت فیزیکی به پین ۵ ترمینال X2 متصل می شود اندازه گیری طول را بر حسب لبه پالس این سوئیچ انجام داد.

طول اندازه گیری شده بر حسب پالس در خروجی Len_Val قابل مشاهده خواهد بود.در اینحالت یک شدن خروجی MSR_Done نشان دهنده اتمام اندازه گیری طول است . خروجی BEG_Val تعداد پالس در نقطه شروع و خروجی END_Val تعداد پالس در نقطه پایان را نشان می دهد.

خروجی های فوق در DB Instance در دسترس هستند.




برنامه ۷ در ضمیمه ۷ صفحه ۴۱۵ کاربرد این فانکشن را نشان می دهد. بعنوان آخرین نکته ذکر می کنیم که در کاربرد عملی همواره قبل از انجام Positioning بترتیب ابتدا Jog Mode سپس سنکرون سازی و پس از آن فاصله Changeover و Cut-off را چک کنید.

در صفحات بعد به شرح کنترل موقعیت توسط فانکشنهای CPU 31xC می پردازیم.

**SFB46 با نام سمبلیک DIGITAL**

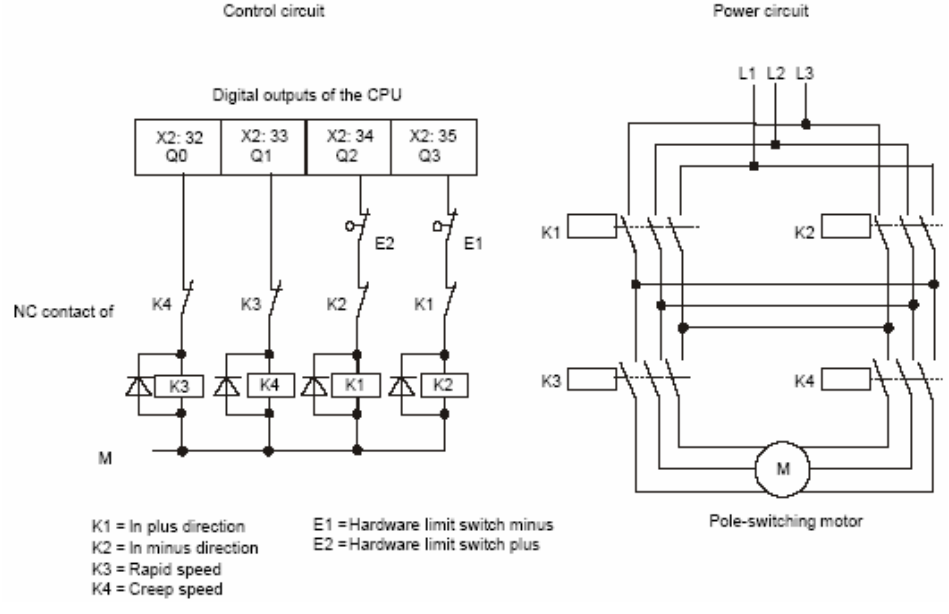
این فانکشن نیز برای Positioning بکار می رود و بجای سیگنال خروجی آنالوگ از ۴ خروجی دیجیتال استفاده می کند. بدلیل مشابه بودن برخی از مفاهیم عملکردی این فانکشن با فانکشن SFB44 ( مانند Jog Mode و سنکرون سازی و Incremental نسبی و مطلق و ....) از تکرار آنها در این بخش خودداری کرده و صرفاً به نکات خاص این فانکشن می پردازیم.

## پین ها و اتصالات

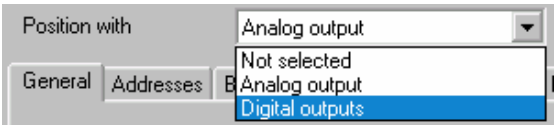
برای Positioning با خروجی دیجیتال از ترمینال X2 کنار CPU و شماره پین های زیر استفاده می شود:

| Terminal | Address | Function |
|---|---|---|
| 32 | DO+1.0 | Digital output Q0 |
| 33 | DO+1.1 | Digital output Q1 |
| 34 | DO+1.2 | Digital output Q2 |
| 35 | DO+1.3 | Digital output Q3 |

خروجی های فوق طبق شکل زیر به مدار تغذیه کنتاکتوری متصل می گردند:

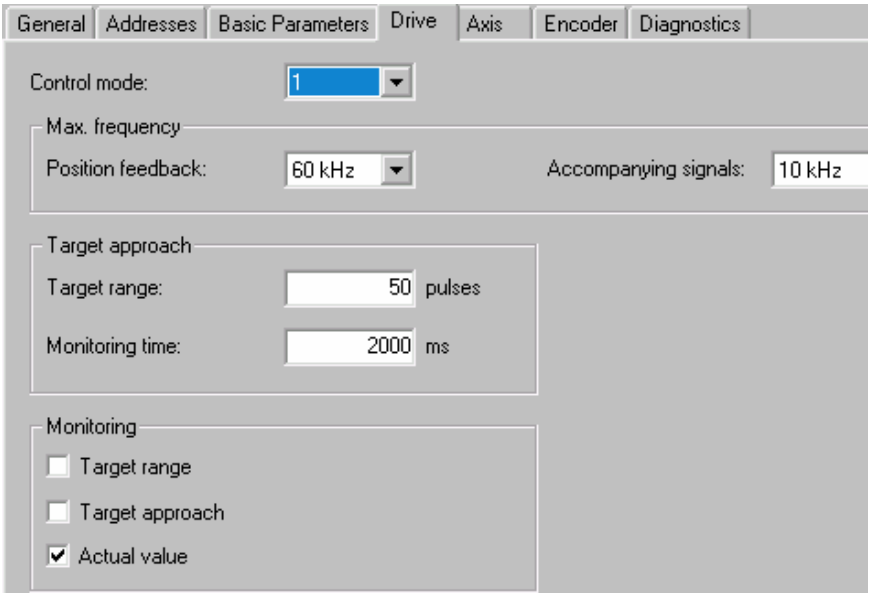


**تنظیمات HWconfig**




۱- با کلیک روی اسلات Position در پنجره ای که باز می شود نوع آن را Digital Output انتخاب می کنیم.

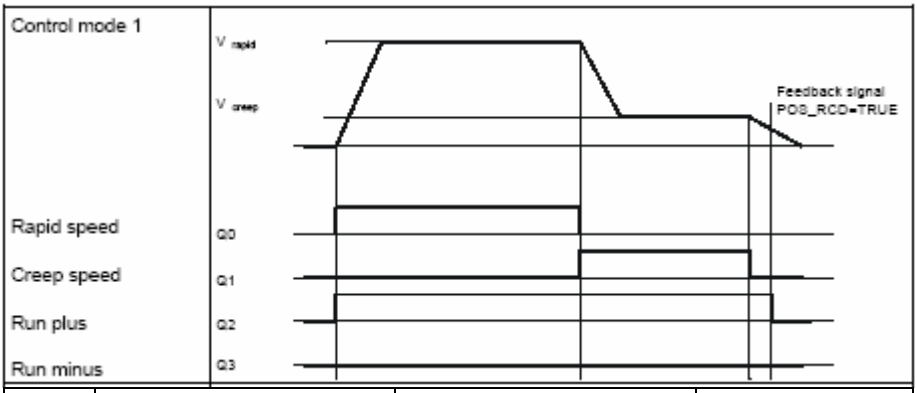
۲- در پنجره Drive مطابق شکل زیر تنظیمات لازم را انجام می دهیم.



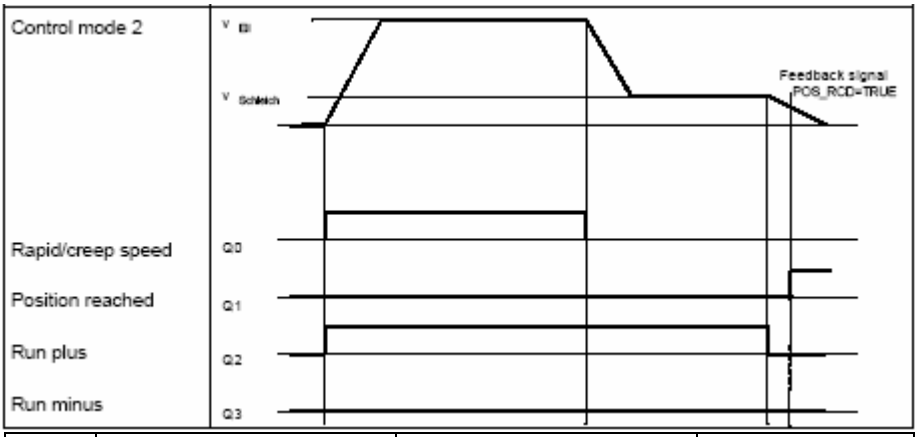

چهار مد کاری برای Drive قابل انتخاب است . وضعیت خروجی ها برای ۴ مد مزبور در جدول زیر آمده و نمودار عملکرد آنها در صفحه بعدهمراه با وضعیت سیگنالها آورده شده است.

| Output | Control mode | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| Q0 | Rapid speed | Rapid/Creep speed | Rapid speed | Rapid speed plus |
| Q1 | Creep speed | Position reached | Creep speed | Creep speed plus |
| Q2 | Run plus | Run plus | Run plus | Rapid speed minus |
| Q3 | Run minus | Run minus | Run minus | Creep speed minus |

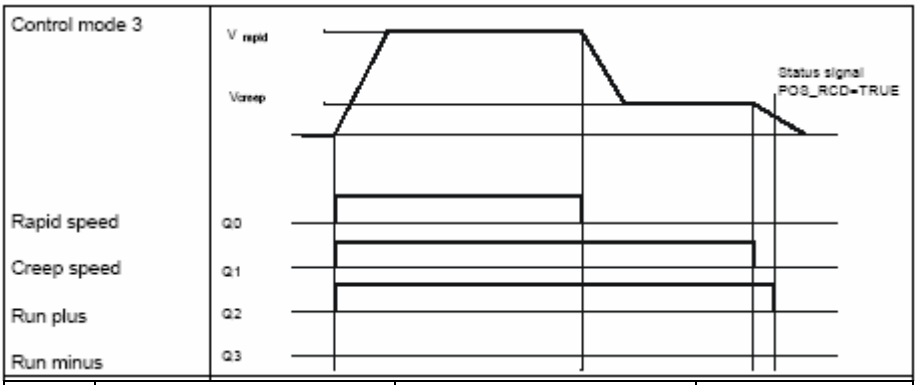
۳- تنظیم سایرپارامترهای پنجره Drive و نیز پنجره های Axis و Encoder مشابه آنچه برای خروجی آنالوگ ذکر شد می باشد.



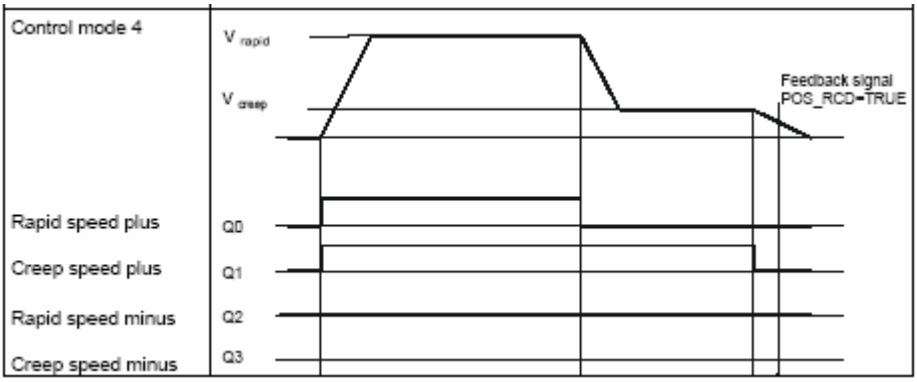

| mode 1 | Rapid speed | | Creep speed | | Position reached (POS_RCD) |
|---|---|---|---|---|---|
| | Plus direction | Direction minus | Plus direction | Direction minus | |
| Q0 | 1 | 1 | 0 | 0 | - |
| Q1 | 0 | 0 | 1 | 1 | - |
| Q2 | 1 | 0 | 1 | 0 | - |
| Q3 | 0 | 1 | 0 | 1 | - |




| mode 2 | Rapid speed | | Creep speed | | Position reached (POS_RCD) |
|---|---|---|---|---|---|
| | Plus direction | Direction minus | Plus direction | Direction minus | |
| Q0 | 1 | 1 | 0 | 0 | 0 |
| Q1 | 0 | 0 | 0 | 0 | 1 |
| Q2 | 1 | 0 | 1 | 0 | 0 |
| Q3 | 0 | 1 | 0 | 1 | 0 |




| mode 3 | Rapid speed | | Creep speed | | Position reached (POS_RCD) |
|---|---|---|---|---|---|
| | Plus direction | Direction minus | Plus direction | Direction minus | |
| Q0 | 1 | 1 | 0 | 0 | - |
| Q1 | 1 | 1 | 1 | 1 | - |
| Q2 | 1 | 0 | 1 | 0 | - |
| Q3 | 0 | 1 | 0 | 1 | - |

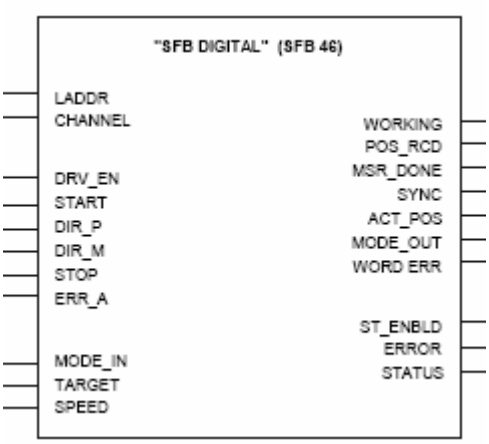


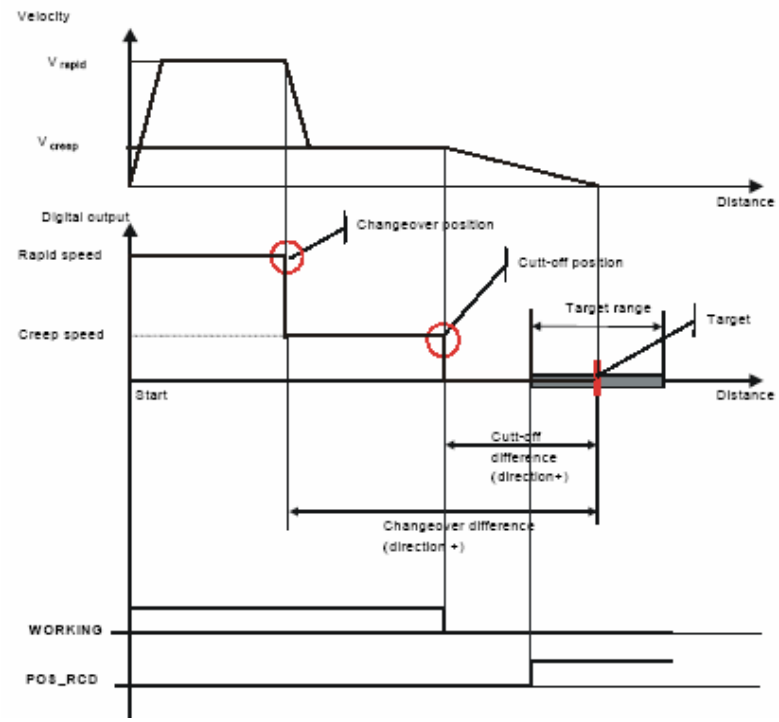

| mode 4 | Rapid speed | | Creep speed | | Position reached (POS_RCD) |
|---|---|---|---|---|---|
| | Plus direction | Direction minus | Plus direction | Direction minus | |
| Q0 | 1 | 0 | 0 | 0 | - |
| Q1 | 1 | 0 | 1 | 0 | - |
| Q2 | 0 | 1 | 0 | 0 | - |
| Q3 | 0 | 1 | 0 | 1 | - |

### صدا زدن SFB46

شکل LAD بلاک SFB46 مطابق شکل روبروست. با تشابهی که بین ورودی و خروجی های این فانکشن با SFB44 وجود دارد از تکرار آنها خودداری می کنیم.

عملکرد Positioning با این فانکشن در شکل زیر ترسیم شده است.




برنامه ۸ در ضمیمه ۷ صفحه ۴۲۹ مثالی از کاربرد این فانکشن را نشان می دهد

**SFB47 با نام سمبلیک Count**

توسط این فانکشن در CPU های 31xC میتوان پالسهایی با فرکانس بالا را دریافت و شمارش کرد. قابلیت CPU های مزبور برای این کار متفاوت و بصورت زیر است:

CPU 312 C : Maximum 10 KHZ
CPU 313 C : Maximum 30 KHZ
CPU 314 C : Maximum 60 KHZ

در نگاه کلی به فانکشن Count میتوان به موارد زیر اشاره کرد:

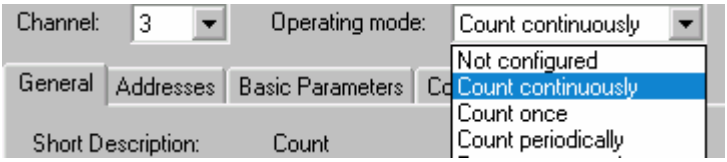
- برای شمارش سه مد وجود دارد شمارش نامحدود ، شمارش Single و شمارش پریودیک
- برای شروع و خاتمه شمارش از Gate Function استفاده می شود.
- برای ذخیره سازی مقدار شماره فعلی در لبه مثبت ورودی دیجیتال از فانکشن Latch استفاده میشود.
- میتوان مقدار مبنایی را برای شمارش به CPU داد تا بر اساس آن سیگنال خروجی دیجیتال یا وقفه سخت افزاری فعال شود.
- میتوان برای جلوگیری از روشن و خاموش شدن خروجی که ناشی از تغییر اندک در سیگنال انکودر است هسیترزیس تعریف نمود.

## پین ها و اتصالات

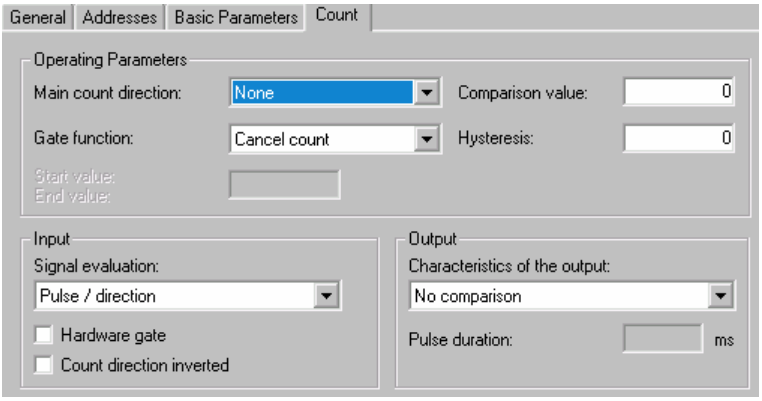
| Terminal | Address | CPU314C : X2 | CPU313C: X2 / X1 | CPU312C : X1 |
|---|---|---|---|---|
| 2 | DI+0.0 | Channel 0: Track A/Pulse | | |
| 3 | DI+0.1 | Channel 0: Track B/Direction | | |
| 4 | DI+0.2 | Channel 0: Hardware gate | | |
| 5 | DI+0.3 | Channel 1: Track A/Pulse | | |
| 6 | DI+0.4 | Channel 1: Track B/Direction | | |
| 7 | DI+0.5 | Channel 1: Hardware gate | | |
| 8 | DI+0.6 | Channel 2: Track A/Pulse | Channel 2: Track A/Pulse | Channel 0: Latch |
| 9 | DI+0.7 | Channel 2: Track B/Direction | Channel 2: Track B/Direction | Channel 1: Latch |
| 12 | DI+1.0 | Channel 2: Hardware gate | Channel 2: Hardware gate | - |
| 13 | DI+1.1 | Channel 3: Track A/Pulse | - | - |
| 14 | DI+1.2 | Channel 3: Track B/Direction | - | Channel 0: Output |
| 15 | DI+1.3 | Channel 3: Hardware gate | - | Channel 1: Output |
| 16 | DI+1.4 | Channel 0: Latch | Channel 0: Latch | - |
| 17 | DI+1.5 | Channel 1: Latch | Channel 1: Latch | - |
| 18 | DI+1.6 | Channel 2: Latch | Channel 2: Latch | - |
| 19 | DI+1.7 | Channel 3: Latch | - | - |
| 22 | DO+0.0 | Channel 0: Output | Channel 0: Output | - |
| 23 | DO+0.1 | Channel 1: Output | Channel 1: Output | - |
| 24 | DO+0.2 | Channel 2: Output | Channel 2: Output | - |
| 25 | DO+0.3 | Channel 3: Output | - | - |

**تنظیمات HWconfig**

۱- با کلیک کردن روی اسلات Count در پنجره ای که ظاهر می شود شماره کانال و نوع شمارش را انتخاب می کنیم .شماره کانال نباید با شماره کانال انتخاب شده برای فانکشن Position یکی باشد.



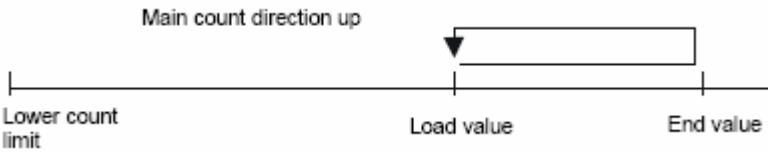

مدهای سه گانه شمارش در صفحات بعد تشریح شده اند. پس از انتخاب یکی از آنها پنجره Count ظاهر می شود که شکلی شبیه زیر دارد:



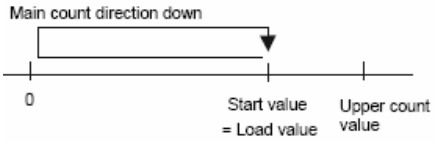

**Main Count Direction** : این گزینه فقط برای مدهای شمارش Single و پریودیک فعال می شود . میتوان None یا Up یا Down را انتخاب کرد.گزینه های Start و End Value برای حالت Up و Down فعال هستند.اگر گزینه None انتخاب شود در اینحالت شمارش میتواند بین حد مینیمم و حد ماکزیمم زیر انجام شود:

- Lower count limit: -2.147.483.648 ($-2^{31}$)
- Upper count limit: +2.147.483.647 ($2^{31}-1$)

اگر گزینه Up انتخاب شود در اینحالت End Value فعال میگردد و میتوان یک حد بالا برای شمارش تعیین کرد . شماره کانتر افزایش مییابد تا اینکه به یکی کمتر از این حد برسدو اتوماتیک به مقدار اولیه پرش میکند.




اگر گزینه Down انتخاب شود در اینحالت Start Value فعال می گردد و میتوان یک حد پایین برای شمارش

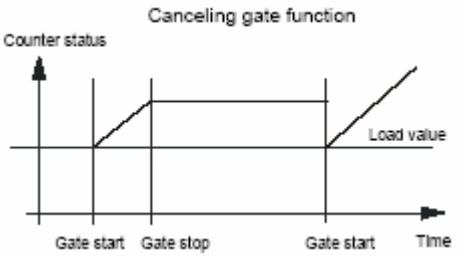


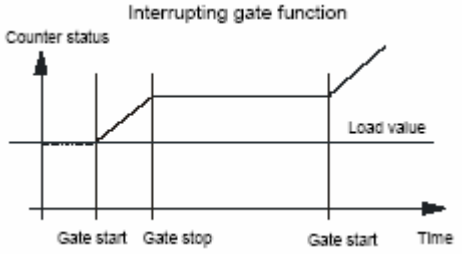

تعریف کرد شماره آنقدر کاهش مییابد تا اینکه به یک برسد در اینحالت اتوماتیک به Start Value پرش میکند عملکرد شمارنده بطور دقیقتر در صفحات بعد مورد بحث قرار گرفته است.

**Gate Function** : روشن و خاموش کردن شمارنده از طریق Gate صورت می گیرد. وقتی Gate بسته شد شمارنده متوقف می گردد. اگر گزینه Cancel انتخاب شود هر بار که Gate باز می شود شمارنده از مقدار اولیه شروع به کار می کند ولی اگر گزینه Stop انتخاب شود شمارنده با باز شدن Gate از همانجایی که متوقف شده بود ادامه می دهد. عملکرد در شکل های صفحه بعد ترسیم شده است.

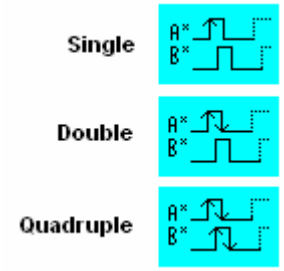
**Comparison Value** : با رسیدن شماره به این عدد خروجی های خاصی فعال می شوند.

**Hysteresis** : برای جلوگیری از نوسان خروجی است و میتواند بین 0 تا 255 باشد . بدیهی است با 0 یا 1 عملاً هیسترزیس اعمال نمی شود.



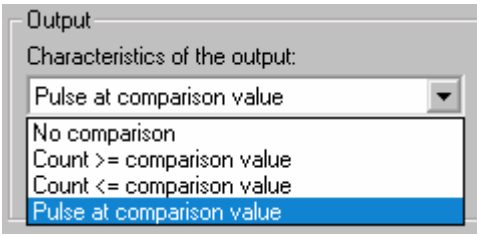




**Signal evaluation** : در این بخش با توجه به نوع پالس انکودر مانند شکل روبرو گزینه مناسب انتخاب میشود.




**Hardware Gate** : این گزینه اگر فعال نباشد یعنی باز و بسته کردن گیت فقط بصورت نرم افزاری ممکن است ولی اگر فعال شود امکان کنترل گیت از طریق ورودی دیجیتال نیز وجود خواهد داشت. در کنترل نرم افزاری این کار توسط ورودی SW_Gate فانکشن انجام می شود.

**Count Direction inverted** :با این گزینه میتوان جهت شمارش را معکوس کرد.

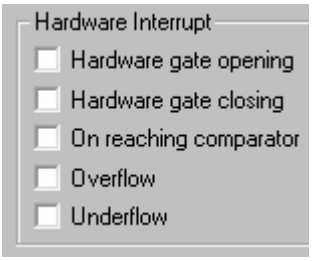



**Charactristics of the output**: در اینجا مشخص می کنیم که با رسیدن شماره به مقدار رفرنس مقایسه که در بالای ان تعیین شده وضعیت خروجی چگونه باشد.یکی از سه وضعیت قابل تعریف است.

۱- روشن شدن شدن خروجی وقتی شماره بزرگتر یا مساوی رفرنس می شود.

۲- روشن شدن شدن خروجی وقتی شماره کوچکتر یا مساوی رفرنس می شود.

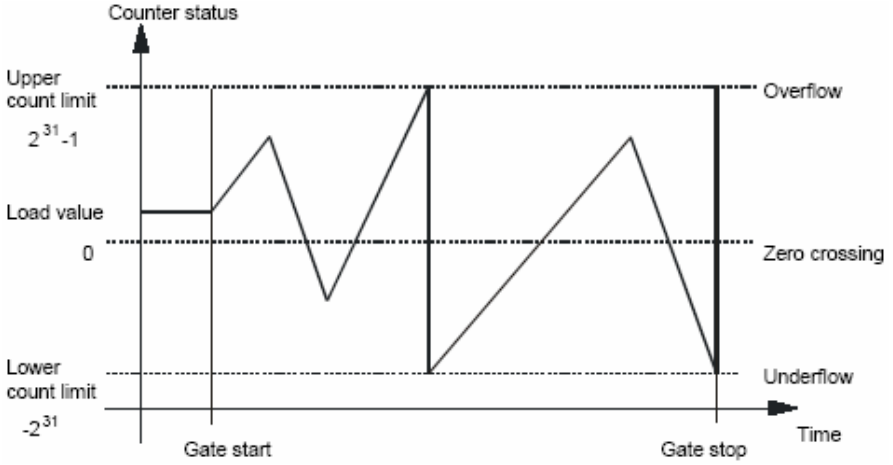
۳- ایجاد پالس وقتی که شماره با رفرنس برابر می شود. در اینحالت duration پالس بر حسب میلی ثانیه در زیر این گزینه مشخص می گردد.




**Hardware Interrupt** : در این قسمت میتوان مشخص کرد که وقفه سخت افزاری تحت چه شرایطی اعمال شود می دانیم که در این شرایط CPU فوراً OB4x را صدا میزند و برنامه آنرا اجرا می کند.این وقفه همانطور که در شکل نشان داده شده میتواند با باز یا بسته شدن گیت یا رسیدن به رفرنس یا در شرایط Overflow و underflow فعال گردد.

## عملکرد شمارش Countinuous

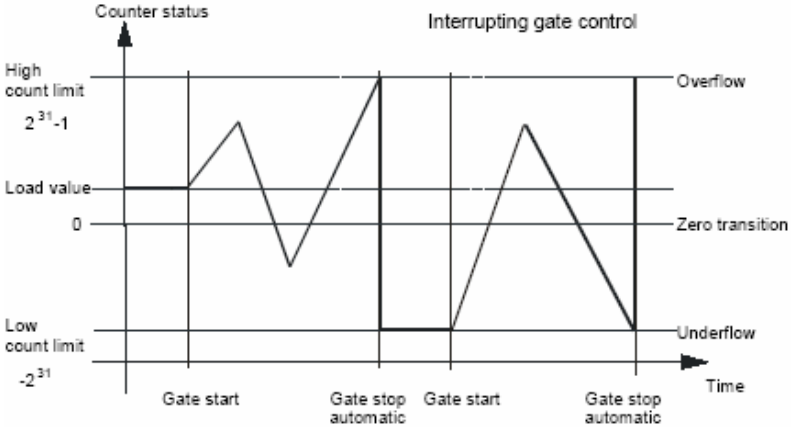
در حالت Continuous شمارش بین دو حد بالاو پایین قابل انجام است. کنترل از طریق Gate انجام می شود.



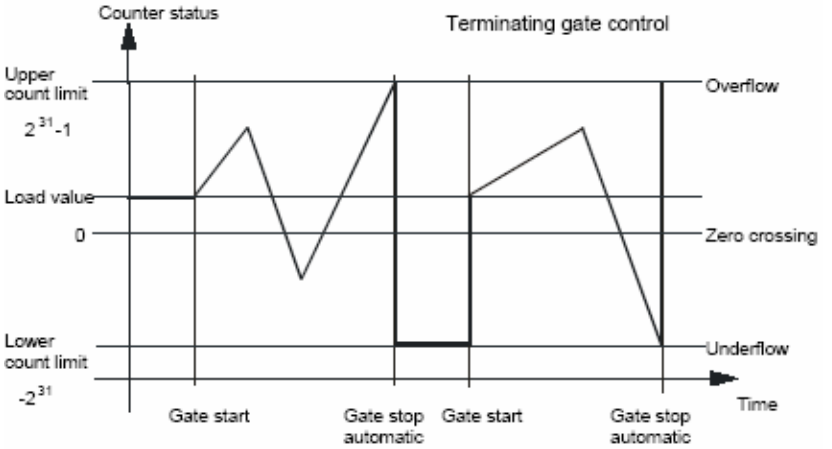
**عملکرد شمارش Single Cycle**

دراین حالت بسته به گزینه Main Count Direction تنظیم شده در HWconfig شرایط متفاوت و بصورت زیراست:

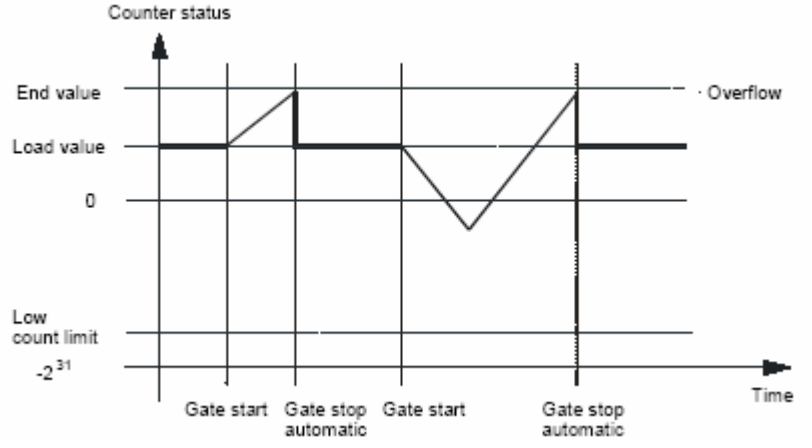
**None** : در این حالت شمارش بین دو حد مینیمم و ماکزیمم شبیه نوع Continuous صورت می گیرد تفاوت در اینست که در Countinuous شماره پس از افزایش وقتی به حد ماکزیمم رسید اتوماتیک به حد مینیمم سوئیچ می شود ولی در اینجا Gate بطور اتوماتیک بسته می شود تا زمانی که توسط لبه مثبت سیگنال جدید باز شودپس از باز شدن نحوه شروع کا ربستگی به پارامتر Gate Function دارد که در Hwconfig تنظیم شده است اگر Stop انتخاب شده باشد با باز شدن از همان نقطه مینیمم شروع بکار می کند. شکل زیر:



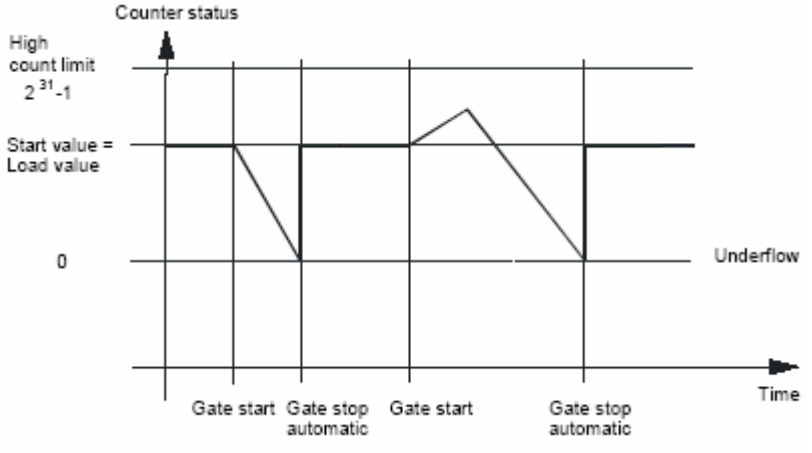

اگر Cancel انتخاب شده باشد با باز شدن گیت شمارش از مقدار اولیه شروع می گردد (شکل زیر)



**UP** : در این حالت شمارش بصورت افزایشی و کاهشی امکان پذیر است. با افزایش شماره و رسیدن به یکی کمتر از End Value که در پارامترهای Hwconfig تعریف شده Gate بطور اتوماتیک بسته می شود تا زمانی که توسط لبه مثبت سیگنال جدید باز شودپس از رسیدن سیگنال وباز شدن Gate شمارش از مقدار اولیه شروع می گردد.



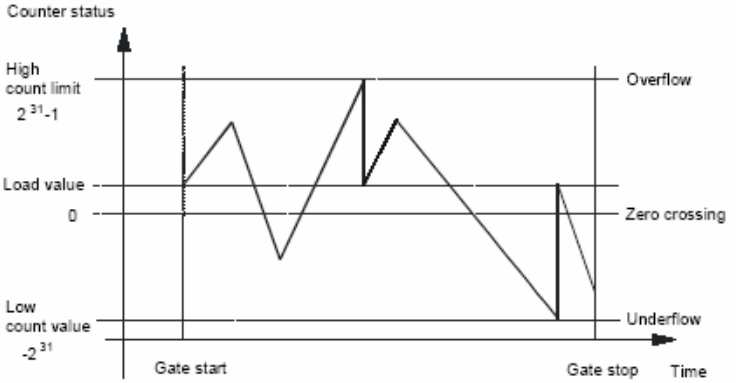

**Down** : در این حالت نیز شمارش بصورت افزایشی و کاهشی امکان پذیر است شمارنده از مقدار Start Value تعیین شده در پارامترهای HWconfig شروع می کند. با کاهش شماره و رسیدن به مقدار یک Gate بطور اتوماتیک بسته می شود تا زمانی که توسط لبه مثبت سیگنال جدید باز شودپس از رسیدن سیگنال وباز شدن Gate شمارش از مقدار Start Value شروع می گردد.



## عملکرد شمارش پریودیک

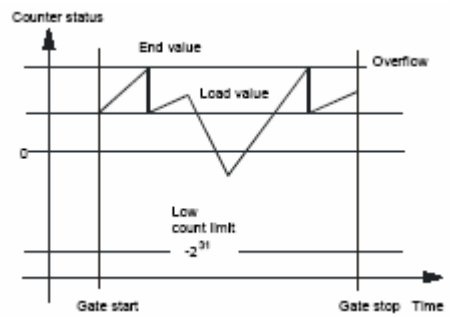
دراین حالت نیز بسته به گزینه Main Count Direction تنظیم شده در HWconfig شرایط متفاوت و بصورت زیر است. کلاً تفاوت این روش با نوع قبلی در ایتست که از مقدار اولیه شروع میکند.

**None** : در این حالت شمارش بین دو حد مینیمم و ماکزیمم شبیه انواع قبلی صورت می گیرد تفاوت در اینست که پس از افزایش وقتی به حد ماکزیمم رسید یا پس از کاهش وقتی به حد مینیمم رسید اتوماتیک به مقدار اولیه سوئیچ می شود.

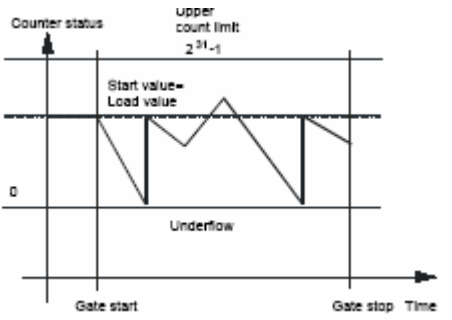



**UP** : در این حالت شمارش بصورت افزایشی و کاهشی امکان پذیر است. با افزایش شماره و رسیدن به یکی کمتر از End Value که در پارامترهای Hwconfig تعریف شده اتوماتیک به مقدار اولیه سوئیچ می شود.

**Down** : در این حالت نیز شمارش بصورت افزایشی و کاهشی امکان پذیر است. با کاهش شماره و رسیدن به یک اتوماتیک به مقدار Start Value که در HWconfig تنظیم شده سوئیچ می شود.




**Up**

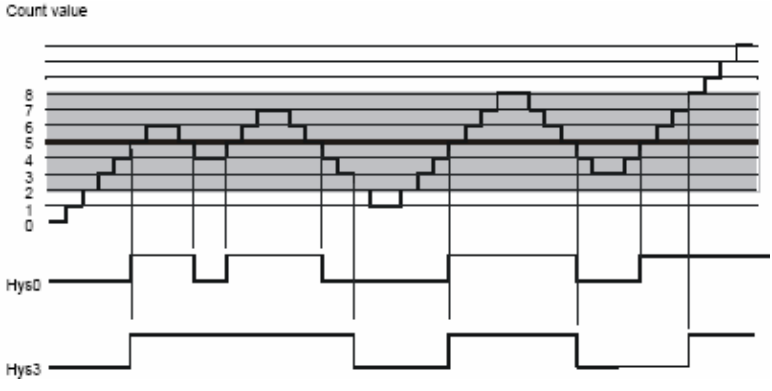



**Down**

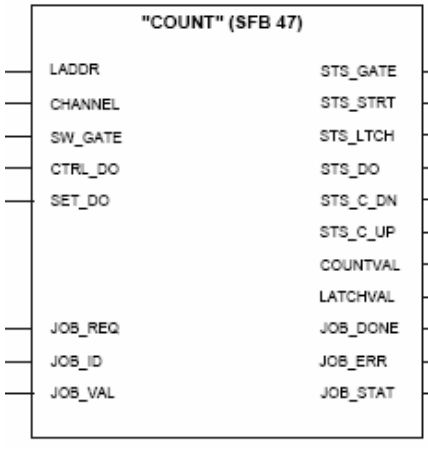
## عملکرد هیسترزیس

هیسترزیس را در پارامترهای HWconfig ذکر کردیم اگر هیسترزیس وجود نداشته باشد وقتی شماره کانتر به مقدار مبنای مقایسه رسید خروجی روشن می شود در این شرایط اگر سیگنال ورودی متناوباً یک شماره بالا و یک شماره پایین برود خروجی مرتباً لازم است روشن و خاموش شود برای جلوگیری از نوسان خروجی در نزدیکی نقطه مقایسه باندی را بعنوان هیسترزیس تعریف می کنند.

شکل زیر عملکرد شمارش را برای دو حالت Hyst=0 و Hyst=3 با فرض عدد 5 برای مقدار مبنای مقایسه نشان می دهد.سیگنال هیسترزیس فقط در محدوده بالا وپایین نقطه مقایسه فعال است.



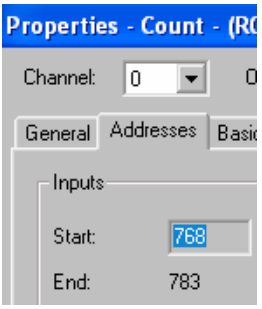

## صدا زدن SFB47




ورودی LADDR : آدرس پایه Submoulde مربوط به شمارش در Hwconfig است و بصورت Word داده می شود. بعنوان مثال در روبرو این آدرس ۷۶۸ که معادل هگز آن W#16#0300 است.

ورودی Channel : آدرس کانال مربوط به Count بصورت عدد Int داده می شود . در شکل روبرو این آدرس 0 است.

ورودی SW_Gate : بصورت Bool است و برای کنترل نرم افزاری Gate بکار می رود. با اعمال لبه مثبت به این ورودی شمارش متوقف میشود




ورودی CTRL_DO : بصورت Bool است و برای کنترل خروجی بکار می رود. از اینجا به بعد هر جا از خروجی نام می بریم منظور خروجی دیجیتال است که شماره پین اتصال آن بسته به کانال و نوع CPU قبلاً ذکر شد. مثلاً پین ۲۲ برای خروجی کانال 0 در CPU313C این خروجی وقتی نتیجه مقایسه درست باشد بشرط اینکه سیگنال CTRL_DO فعال باشد روشن می گردد.

ورودی SET_DO : برای روشن کردن خروجی بدون در نظر گرفتن نتیجه مقایسه

ورودی Job_Req : بصورت Bool است و توسط آن میتوان Job را برای ورودی های بعدی فعال نمود.

ورودی Job_ID : بصورت Word است و بسته به اینکه چه نوع کاری مد نظر است یکی از کدهای زیر را می گیرد بعنوان مثال میتوان با کد W#16#02 مقدار اولیه به شمارنده اختصاص داد.

| Job ID: | Code |
|---|---|
| • Job without function | • 00H |
| • Writes the count value | • 01H |
| • Writes the load value | • 02H |
| • Writes the reference value | • 04H |
| • Writes the hysteresis | • 08H |
| • Writes the pulse width | • 10H |
| • Reads the load value | • 82H |
| • Reads the reference value | • 84H |
| • Reads the hysteresis | • 88H |
| • Reads the pulse width | • 90H |

ورودی Job_VAL : مقداری که توسط job داده می شود بصورت DINT در این ورودی تعریف می شود.

**تذکر** : علاوه بر ورودی های فوق یک ورودی دیگر بنام RES_STS که بعنوان پارامترهای استاتیک در DB Instance تعریف شده و آدرس آن 32.2 می باشد قابل استفاده است. این پارامتربصورت Bool است و با یک شدن بیتهای Status خروجی که در قسمت بعد معرفی می شوند را ری ست می کند.

**خروجی های فانکشن:** این خروجی ها در جدول زیر آمده است.

| Parameter | Data type | Description |
|---|---|---|
| **STS_GATE** | BOOL | Status of the internal gate |
| **STS_STRT** | BOOL | Status of the hardware gate (START input) |
| **STS_LTCH** | BOOL | Status of the latch input |
| **STS_DO** | BOOL | Status of the Output |
| **STS_C_DN** | BOOL | Status of the down-count. |
| **STS_C_UP** | BOOL | Status of the up-count. |
| **COUNTVAL** | DINT | Actual count value |
| **LATCHVAL** | DINT | Actual latch value |

علاوه بر موارد فوق خروجی های دیگر نیز از طریق DB Instance قابل دسترس هستند که عبارتند از:

| Parameters | type | Addr | Description | Parameters | type | Addr | Description |
|---|---|---|---|---|---|---|---|
| **STS_CMP** | BOOL | 26.3 | Comparator status. | **STS_UFLW** | BOOL | 26.6 | Underflow status |
| **STS_OFLW** | BOOL | 26.5 | Overflow status | **STS_ZP** | BOOL | 26.7 | Status of zero mark |

برنامه ۹ در ضمیمه ۷ صفحه ۴۴۳ کاربرد این فانکشن را نشان میدهد.

**SFB48 با نام سمبلیک FREQUENC**

این فانکشن برای اندازه گیری فرکانس پالس هایی که به ورودی CPU های 31xC اعمال می شوند بکار می رود. پالس میتواند از انکودر در یافت شود. فرکانس پالس اندازه گیری شده بصورت mHZ در خروجی این فانکشن قابل دریافت است. ماکزیمم فرکانس پالس قابل اندازه گیری بستگی به نوع CPU داردطبق جدول زیر:

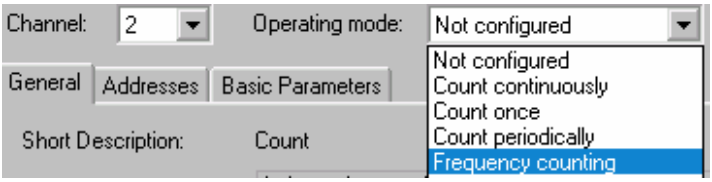
| CPU 312C | CPU 313C | CPU 314C-2 |
|---|---|---|
| 0 to 10 kHz | 0 to 30 kHz | 0 to 60 kHz |

**پین ها و اتصالات**

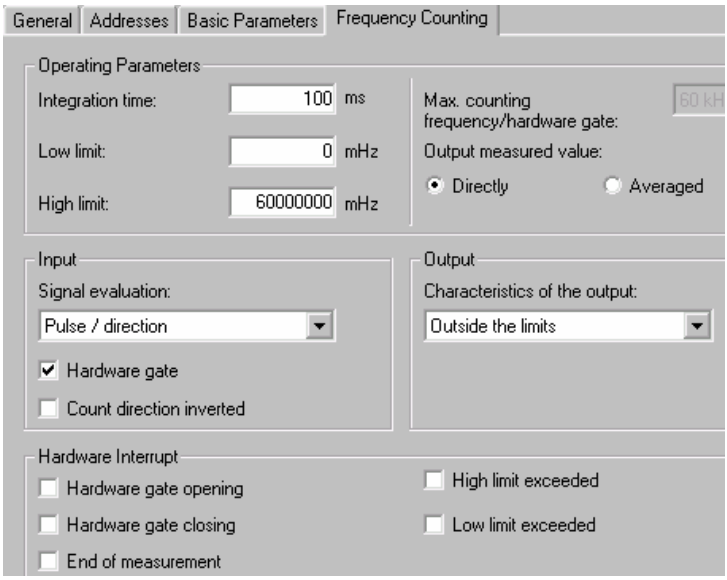
| No | Name/ Address | 314C | 313C | 312C |
|---|---|---|---|---|
| 1 | 1 L+ | 24-V power supply for the inputs | | Not connected |
| 2 | DI+0.0 | Channel 0: Track A/Pulse | | |
| 3 | DI+0.1 | Channel 0: Track B/Direction | | |
| 4 | DI+0.2 | Channel 0: Hardware gate | | |
| 5 | DI+0.3 | Channel 1: Track A/Pulse | | |
| 6 | DI+0.4 | Channel 1: Track B/Direction | | |
| 7 | DI+0.5 | Channel 1: Hardware gate | | |
| 8 | DI+0.6 | Channel 2: Track A/Pulse | | - |
| 9 | DI+0.7 | Channel 2: Track B/Direction | | - |
| 10 | - | n.c. | n.c. | - |
| 11 | - | n.c. | n.c. | - |
| 12 | DI+1.0 /2M | Channel 2: Hardware gate | | Chassis ground |
| 13 | DI+1.1 /1 L+ | Channel 3: Track A/Pulse | - | 24-V supply for outputs |
| 14 | DI+1.2/ DO+0.0 | Channel 3: Track B/Directio | - | Channel 0: Output |
| 15 | DI+1.3 / DO+0.1 | Channel 3: Hardware gate | - | Channel 1: Output |
| 16 | DI+1.4 | - | - | - |
| 17 | DI+1.5 | - | - | - |
| 18 | DI+1.6 | - | - | - |
| 19 | DI+1.7 | - | - | - |
| 20 | 1M | Chassis ground | Chassis ground | Chassis ground |
| 21 | 2 L+ | 24-V power supply for the outputs | | - |
| 22 | DO+0.0 | Channel 0: Output | Channel 0: Output | - |
| 23 | DO+0.1 | Channel 1: Output | Channel 1: Output | - |
| 24 | DO+0.2 | Channel 2: Output | Channel 2: Output | - |
| 25 | DO+0.3 | Channel 3: Output | - | - |
| 26 | DO+0.4 | - | - | - |
| 27 | DO+0.5 | - | - | - |
| 28 | DO+0.6 | - | - | - |
| 29 | DO+0.7 | - | - | - |
| 30 | 2 M | Chassis ground | Chassis ground | - |
| 31 | 3 L+ | 24-V power supply for the outputs | | - |
| 32 | DO+1.0 | - | - | - |
| 33 | DO+1.1 | - | - | - |
| 34 | DO+1.2 | - | - | - |
| 35 | DO+1.3 | - | - | - |
| 36 | DO+1.4 | - | - | - |
| 37 | DO+1.5 | - | - | - |
| 38 | DO+1.6 | - | - | - |
| 39 | DO+1.7 | - | - | - |
| 40 | 3 M | Chassis ground | Chassis ground | - |

**پارامترهای HWconfig**

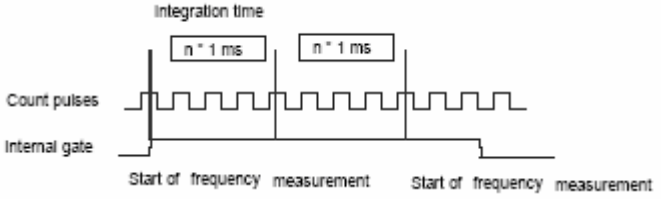
۱- با کلیک روی اسلات Count در پنجره ای مانند شکل زیر کانال و Operating Mode را انتخاب می کنیم. کانالی که برای Frequency Counting استفاده می شود نباید قبلاً برای فانکشن دیگری بکار رفته باشد.




۲- پنجره Frequency counting اضافه می شود که در آن تنظیمات مقتضی بایستی صورت گیرد.



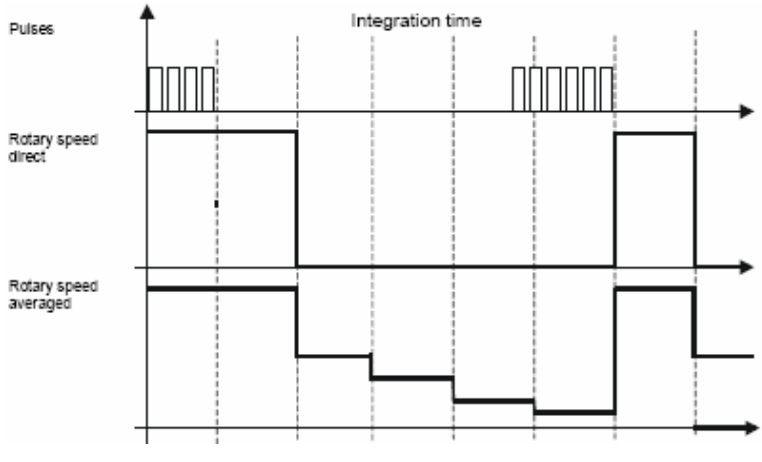

**Integration Time** : منظور بازه زمانی است که در آن فرکانس پالسهای ورودی اندازه گیری می شود و میتواند بین 10 تا 10000ms در پله های 1ms باشد. اندازه گیری فرکانس پس از اتمام هر دوره زمانی مشخص شده ای صورت می گیرد مثلاً در طول 10ms از پالسها نمونه برداری شده و پس از اتمام 100ms فرکانس محاسبه می گردد.




**Low Limit / High Limit** : میتوان دو مقدار را بر حسب mHZ بعنوان نقاط بالا و پایین تعریف کرد تا در صورت رسیدن فرکانس به این نقاط ، خروجی فعال شود یا وقفه سخت افزاری اعمال گردد. مقدار ماکزیمم بسته به نوع CPU متفاوت است مثلاً در نوع 312C معادل 10KHZ یعنی 10,000,000 mHZ میتواند باشد. با رسیدن فرکانس به حد بالا بیت STS_OFLW و با رسیدن به حد پایین بیت STS_UFLW از فانکشن فعال می شود.

**Max Count Frequency** : این مقدار بر حسب KHZ و متناسب با نوع CPU ظاهر می شود.

**Output of the Measured Value** : دوروش برای محاسبه فرکانس قابل استفاده است یک Direct و دیگری Averaged. این دو روش در شکل زیر نشان داده شده است. در نوع Direct فرکانس در هر دوره زمانی مستقل از دوره های دیگر اندازه گیری می شود بنابر این در دوره هایی که پالس وجود ندارد این فرکانس صفر است.
در نوع Averaged اگر در یک یا چند دوره زمانی پالسی دریافت نشود فرکانس این دوره ها صفر نیست بلکه معادل میانگین فرکانس آخرین دوره ای که پالس وجود داشته با دوره هایی که پالس وجود نداشته محاسبه و بعنوان فرکانس در خروجی ظاهر می شود. این وضعیت تا جایی ادامه می یابد که در دوره جدید پالس برسد که در انتهای این دوره فرکانس براساس پالس جدید محاسبه خواهد شد.



**Signal Evaluation** : بسته به نوع وسیله تولید کننده پالس انتخاب می شود. گزینه Pulse/Direction یعنی منبع پالس و جهت سیگنال هر دو به کانالهای ورودی متصل هستند ولی در نوع Rotary Transducer یعنی فقط یک سیگنال وجود دارد.

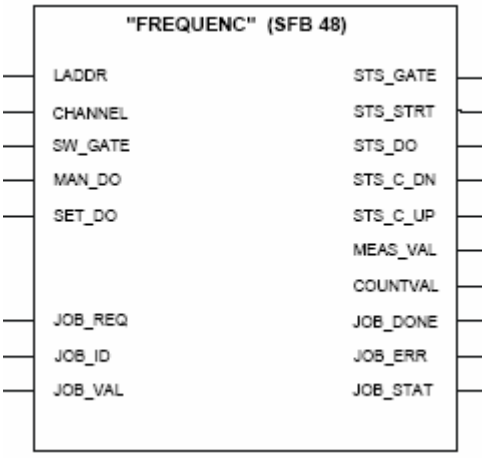
**HW Gate** : اگر گزینه Yes انتخاب شود گیت هم بطریق سخت افزاری و هم بطریق نرم افزاری قابل کنترل است و اگر گزینه No انتخاب شود کنترل گیت صرفاً نرم افزاری است. بدیهی است با انتخاب Yes عمل اندازه گیری فرکانس در صورتی انجام می شود که هر دو گیت HW و SW باز باشند. وضعیت گیت از طریق خروجی STS_Gate فانکشن قابل مشاهده است.

**Output Reaction** : در این قسمت میتوان مشخص کرد که خروجی در چه صورتی فعال شود . بالاتر بودن فرکانس از حد ماکزیمم یا پایین تر بودن آن از حد مینیمم یا کلاً خارج از دو حد فوق بودن آن شرایطی است که در اینجا قابل تنظیم است.

**Hardware Interrupt** : در این قسمت شرایطی که میخواهیم در صورت وقوع آنها وقفه سخت افزاری اعمال شده و OB40 اجرا شود را فعال می کنیم.

## صدا زدن SFB48

شکل زیر بلاک LAD این فانکشن را همراه با ورودی و خروجی های آن نشان می دهد.




### ورودی ها :

ورودی های LADDR و CHANNEL و SW_GATE مشابه مواردی است که برای فانکشن های قبل ذکرشد.

ورودی MAN_DO : بصورت Bool است که اگر فعال شود میتوان بصورت دستی از طریق SET_DO خروجی را فعال کرد. اگر MAN_DO غیر فعال باشدخروجی از مقایسه گر داخلی فرمان می گیرد.

ورودی SET_DO : برای روشن کردن خروجی در مد دستی است.

ورودی JOB_REQ : با لبه مثبت JOB تعریف شده را فعال می کند.

ورودی JOB_ID : بصورت Word است که بسته به نوع کار مورد نظر کدی مطابق جدول زیر می گیرد:

| | |
|---|---|
| Job without function | • 00 hex |
| Write the low limit | • 01 hex |
| Write the high limit | • 02 hex |
| Write the integration time | • 04 hex |
| Read the low limit | • 81 hex |
| Read the high limit | • 82 hex |
| Read ithe ntegration time | • 84 hex |

ورودی JOB_VAL : مقداری که توسط Job برای عمل Write استفاده می شود بصورت DINT به این ورودی داده می شود.

**تذکر**: علاوه بر ورودی های فوق در DB instance نیز ورودی RES_STS بصورت Bool وجود دارد که در صورت فعال شدن میتواند خروجی های STS_OFLW و STS_UFLW را ری ست کند.

## خروجی ها :

خروجی STS_Gate : بصورت Bool و نشان دهنده وضعیت گیت است. یک بودن آن به معنای باز بودن است.

خروجی STS_STRT : بصورت Bool و نشان دهنده نقطه شروع باز شدن گیت سخت افزاری است.

خروجی STS_DO : وضعیت خروجی را نشان می دهد.

خروجی STS_C_DN : اگر فعال باشد یعنی پالسها بصورت کاهشی شمرده می شوند.

خروجی STS_C_UP : اگر فعال باشد یعنی پالسها بصورت افزایشی شمرده می شوند.

خروجی MEAS_VAL : مقدار فرکانس را بصورت DINT نشان می دهد.

خروجی COUNT_VAL : تعداد پالس شمارش شده را بصورت DINT نشان می دهد و در هر دوره زمانی از صفر شروع می شود.

خروجی های مربوط به Job : وضعیت JOB را نشان می دهند.

**تذکر**: خروجی های زیر نیز از DB_Instance در دسترس هستند:

خروجی STS_CMP: این بیت در انتهای دوره زمانی که اندازه گیری کامل شد یک می شود.

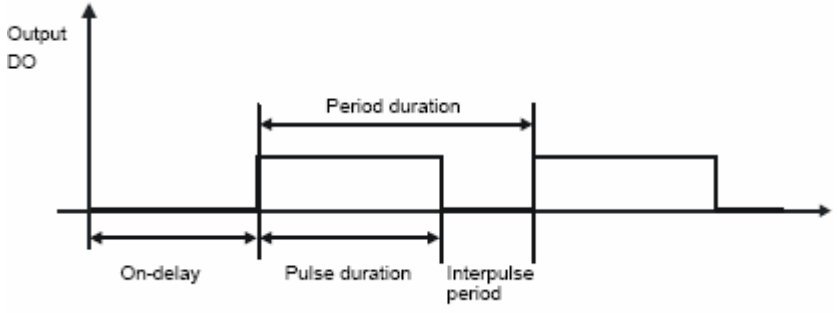
خروجی STS_OFLW : این بیت رسیدن فرکانس به حد ماکزیمم را نشان می دهد.

خروجی STS_UFLW : این بیت رسیدن فرکانس به حد مینیمم را نشان می دهد.

برنامه ۱۰ در ضمیمه ۷ صفحه ۴۵۰ کاربرد این فانکشن را نشان می دهد.

**SFB49 با نام سمبلیک PULSE**

این فانکشن برای مدولاسیون پهنای پالس PWM بکار می رود و توسط آن می توان مقدار OUT_VAL داده شده را به یک توالی از کدهای پالس با فواصل مشخص تبدیل کرد. این پالس در خروجی دیجیتال کنار CPU پس از گذشت زمان تاخیر On Delay طبق شکل زیر ظاهر خواهد شد:



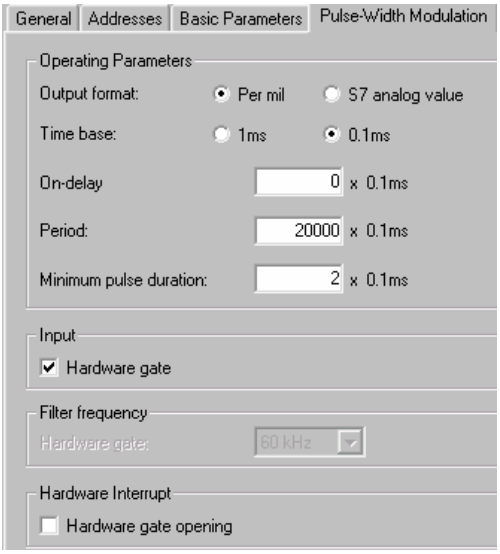

مشخصات فنی توالی پالس در جدول زیر آمده است:

| **Technical data of the pulse sequence** | |
|---|---|
| Output frequency | 0 to 2.5 kHz |
| Minimum pulse width | 200 µs |
| Interpulse accuracy | +/- (Pulse length x 100 ppm) +/- 100 µs ppm = Parts per million |
| Accuracy of the ON delay | 0 to 250 µs |

## پین ها و اتصالات

| | **Name/ Address** | **314C** | **313C** | **312C** |
|---|---|---|---|---|
| 1 | 1 L+ | 24-V supply for inputs | 24-V supply for inputs | Not connected |
| 2 | DI+0.0 | - | - | - |
| 3 | DI+0.1 | 0 / do not use | 0 / do not use | - |
| 4 | DI+0.2 | Channel 0: Hardware gate | Channel 0: Hardware gate | Channel 0: HW gate |
| 5 | DI+0.3 | - | - | - |
| 6 | DI+0.4 | 0 / do not use | 0 / do not use | - |
| 7 | DI+0.5 | Channel 1: HW gate | Channel 1: HW gate | Channel 1: HW gate |
| 8 | DI+0.6 | - | - | - |
| 9 | DI+0.7 | 0 / do not use | 0 / do not use | - |
| 10 | - | n.c. | n.c. | - |
| 11 | - | n.c. | n.c. | - |
| 12 | DI+1.0 /2M | Channel 2: HW gate | Channel 2: HW gate | Chassis ground |
| 13 | DI+1.1 /1L+ | - | - | 24-V supply for the outputs |
| 14 | DI+1.2 / DO+0.0 | 0 / do not use | - | Channel 0: Output |
| 15 | DI+1.3 / DO+0.1 | Channel 3: HW gate | - | Channel 1: Output |
| 16 | DI+1.4 | - | - | - |

| | | | | |
|---|---|---|---|---|
| 17 | DI+1.5 | - | - | - |
| 18 | DI+1.6 | - | - | - |
| 19 | DI+1.7 | - | - | - |
| 20 | 1M | Chassis ground | Chassis ground | Chassis ground |
| 21 | 2 L+ | 24-V supply for the outputs | 24-V supply for the outputs | - |
| 22 | DO+0.0 | Channel 0: Output | Channel 0: Output | - |
| 23 | DO+0.1 | Channel 1: Output | Channel 1: Output | - |
| 24 | DO+0.2 | Channel 2: Output | Channel 2: Output | - |
| 25 | DO+0.3 | Channel 3: Output | - | - |
| 26 | DO+0.4 | - | - | - |
| 27 | DO+0.5 | - | - | - |
| 28 | DO+0.6 | - | - | - |
| 29 | DO+0.7 | - | - | - |
| 30 | 2 M | Chassis ground | Chassis ground | - |
| 31 | 3 L+ | 24-V supply for the outputs | 24-V supply for the outputs | - |
| 32 | DO+1.0 | - | - | - |
| 33 | DO+1.1 | - | - | - |
| 34 | DO+1.2 | - | - | - |
| 35 | DO+1.3 | - | - | - |
| 36 | DO+1.4 | - | - | - |
| 37 | DO+1.5 | - | - | - |
| 38 | DO+1.6 | - | - | - |
| 39 | DO+1.7 | - | - | - |
| 40 | 3 M | Chassis ground | Chassis ground | - |




**تنظیمات HWconfig**

۱- کلیک کردن روی اسلات Count و انتخاب PWM و کانال مربوطه شبیه آنچه برای فانکشن های قبلی توضیح داده شد.

۲- انجام تنظیمات در پنجره Pulse width modulation شکل روبرو با توضیحات زیر:

Output Format: در این قسمت نوع ورودی OUT_VAL مشخص می گردد که میتواند به دو صورت باشد اگر بین ۰ تا ۱۰۰۰ است انتخاب permil و اگر یک سیگنال آنالوگ است S7 Analog انتخاب شود.

پس از انتخاب فرمت محاسبه PWM مطابق فرمول مندرج در جدول بعد انجام می شود.

| Output format | Range of values | Pulse width |
|---|---|---|
| Per mil | 0 to 1.000 | (Output value / 1.000) x Period |
| S7 analog value | 0 to 27.648 | (Output value / 27.648) x Period |

Time Base : مبنای زمان است که برای Resolution پالس و نیز محاسبات بعدی مورد استفاده قرار می گیرد . این مبنا همانطور که در شکل قبل نمایش داده شده میتواند 0.1 ms یا 1ms باشد.

Period : توسط این پارامتر طول توالی Pulse / Interpulse مشخص می گردد.

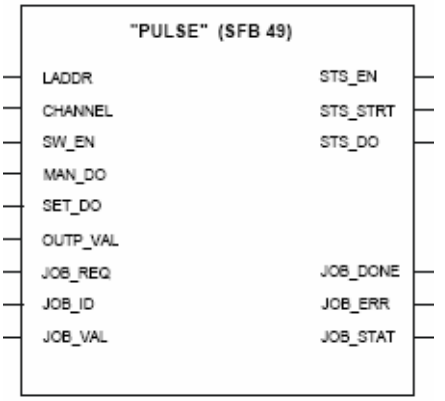
On Delay : این پارامتر زمان تاخیر پالس از نقطه شروع را نشان می دهد.

Minimum Pulse Width : با این پارامتر تمام سیگنالهای Low / High که عرض آنها کمتر از مقدار فوق باشد حذف می شوند.

مقادیر سه پارامتر اخیر برای هر کدام از Time Base ها در جدول زیر آمده است.

| | Timebase: 0.1 ms | Timebase: 1 ms |
|---|---|---|
| **Period** | 4 to 65535 | 1 to 65535 |
| **On delay** | 0 to 65535 | 0 to 65535 |
| **Minimum pulse width** | 2 to period/2 | 0 to period/2 (0 = 0.2 ms) |

## صدا زدن SFB49




بلاک LAD این فانکشن شبیه شکل روبروست. با توجه به توضیحاتی که در مورد فانکشن های قبلی داده شد اکثر ورودی و خروجی های این فانکشن مشخص است . ورودی OUTP_VAL مقداریست که لازمست به پالس PWM تبدیل شود و در مورد فرمت آن توضیح داده شد. ورودی JOB_ID برای انجام عملیات مورد نظر همراه با کدهای مندرج در جدول بکار می رود. برای حالت Write مقدار بصورت DINT به ورودی JOB_VAL داده می شود.

| | | | | | | |
|---|---|---|---|---|---|---|
| Job without function | 00 hex | | | | | |
| Write the period length | 01 hex | | | | Read the period length | 81 hex |
| Write the on delay | 02 hex | | | | Read the on delay | 82 hex |
| Write the minimum pulse width | 04 hex | | | | Read the minimum pulse width | 84 hex |

برنامه ۱۱ در ضمیمه ۷ صفحه ۴۵۰ کاربرد این فانکشن را نشان میدهد.

در اینجا بحث فانکشنهای سیستم را خاتمه داده در ادامه به بحث در مورد فانکشن های IEC می پردازیم.

## ۱۲-۳ فانکشن های IEC

همانطور که در ابتدای این فصل اشاره شد در Library علاوه بر فانکشن های سیستم خانواده ای با عنوان IEC Function Blocks وجود دارند که بسیاری از عملیات پیچیده را ساپورت می کنند. این فانکشنها را میتوان به چهار دسته زیر تقسیم بندی کرد که جزئیات آنها در جدول آمده است

- Convert که تبدیلی را روی نوع دیتا انجام می دهند.
- DT که برای عملیات مقایسه تاریخ و زمان بکار می رود.
- String که برای عملیات روی String کاربرد دارند.
- Floating Point Math برای عملیات روی متغیرهای اعشاری و ۳۲ بیتی.

**تذکر ۱** : این فانکشن ها در صورت استفاده لازم است به PLC دانلود شوند.

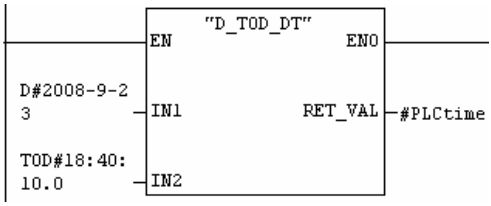
**تذکر ۲**: در هنگام فراخوانی نبایستی FC با همین شماره ( برنامه کاربر ) وجود داشته باشد .

| | | | |
|---|---|---|---|
| | | **String** | |
| | | FC10 EQ_STRNG | مقایسه دو String از نظر مساوی بودن |
| **Convert** | | FC13 GE_STRNG | مقایسه دو String از نظربزرگتر مساوی بودن |
| FC3 D_TOD_DT | ترکیب تاریخ و زمان و تبدیل بصورت DT | FC15 GT_STRNG | مقایسه دو String از نظر بزرگتر بودن |
| FC6 DT_DATE | استخراج تاریخ از متغیر نوع DT | FC19 LE_STRNG | مقایسه دو String ازنظر کوچکتر مساوی |
| FC7 DT_DAY | استخراج روز از متغیر نوع DT | FC24 LT_STRNG | مقایسه دو String از نظر کوچکتر بودن |
| FC8 DT_TOD | استخراج زمان از متغیر نوع DT | FC29 NE_STRNG | مقایسه دو String از نظر مخالف بودن |
| FC33 S5TI_TIM | تبدیل فرمت زمان S5Time به نوع Time | FC21 LEN | بدست آوردن طول String |
| FC40 TIM_S5TI | تبدیل فرمت زمان Time به نوع S5Time | FC20 LEFT | جدا کردن بخشی از String از سمت چپ |
| FC16 I_STRNG | تبدیل INT به STRING | FC32 RIGHT | جدا کردن بخشی از String از سمت راست |
| FC5 DI_STRNG | تبدیل DINT به STRING | FC26 MID | جدا کردن بخشی از وسط String |
| FC30 R_STRNG | تبدیل REAL به STRING | FC2 CONCAT | ترکیب دو String |
| FC38 STRNG_I | تبدیل STRING به INT | FC17 INSERT | وارد کردن به String |
| FC37 STRNG_DI | تبدیل STRING به DINT | FC4 DELETE | پاک کردن قسمتی از String |
| FC39 STRNG_R | تبدیل STRING به REAL | FC31 REPLACE | جایگزین کردن قسمتی از String |
| **Floating Point Math** | | FC11 FIND | پیدا کردن در String |
| FC1 AD_DT_TM | اضافه کردن مدت به زمان | **Date And Time (DT)** | |
| FC35 SB_DT_TM | کم کردن مدت از زمان | FC9 EQ_DT | مساوی بودن |
| FC34 SB_DT_DT | کم کردن دو زمان از یکدیگر | FC12 GE_DT | بزرگتر یا مساوی بودن |
| FC22 LIMIT | محاسبه حدود | FC14 GT_DT | بزرگتر بودن |
| FC25 MAX | محاسبه ماکزیمم | FC18 LE_DT | کوچکتر یا مساوی بودن |
| FC27 MIN | محاسبه مینیمم | FC23 LT_DT | کوچکتر بودن |
| FC36 SEL | انتخاب باینری | FC28 NE_DT | مخالف بودن |

### ۱۲-۳-۱ فانکشنهای تبدیل IEC

**FC3 با نام سمبلیک D_TOD_DT**

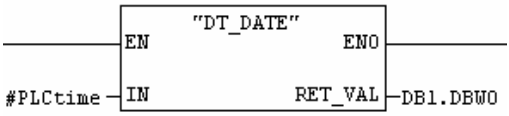
توسط این فانکشن میتوان تاریخ و زمان را با هم ترکیب کرد و در متغیری از جنس Date_And_Time ریخت. ورودی IN1 از جنس Date و ورودی IN2 از جنس Time است ترکیب بصورت DT در خروجی RET_VAL ظاهر می گردد.برای استفاده از خروجی میتوان یک متغیر محلی در بالای بلاک از جنس Date_and_Time تعریف کرد وسپس آنرا به این خروجی معرفی نمود.




**توجه** : از خروجی این فانکشن براحتی میتوان برای تنظیم ساعت CPU استفاده کرد کافی است در Network بعدی فانکشن SFC0 را صدا بزنیم و خروجی فانکشن فوق را به ورودی آن نسبت دهیم.

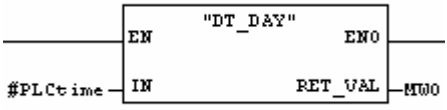
**FC6 با نام سمبلیک DT_DATE**

این فانکشن تاریخ را از Date_And_Time جدا کرده و در خروجی RET_VAL ظاهر میسازد.



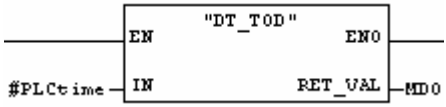

**FC7 با نام سمبلیک DT_DAY**

این فانکشن روز را از Date_And_Time جدا کرده و در خروجی RET_VAL ظاهر میسازد.




**FC8 با نام سمبلیک DT_TOD**

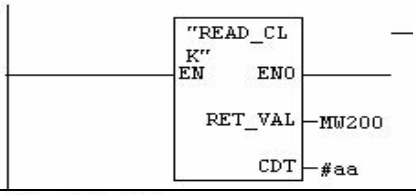
این فانکشن زمان را از Date_And_Time جدا کرده و در خروجی RET_VAL ظاهر میسازد.



**مثال**

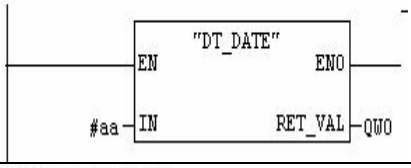
برنامه ای بنویسید که زمان و تاریخ را از CPU بخواند سپس آنها را از هم جدا کرده و هرکدام را روی یک خروجی نشان دهد. در ضمن مشخص کند که چه روزی از هفته است.

Network 1 : Title:



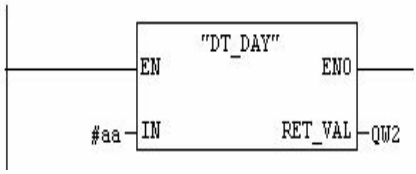

توسط SFC1 تاریخ وزمان CPU خوانده می شود.
و در متغیر aa که از جنس DT است ذخیره می گردد.

Network 2 : Title:




توسط فانکشن **FC6** تاریخ آن جدا شده وبه Qw0 ارسال می شود.

Network 3 : Title:




توسط **FC7** با نام سمبلیک DT-DATE می توان مشخص کرد که چندمین روز هفته است.

یک شنبه = ۱

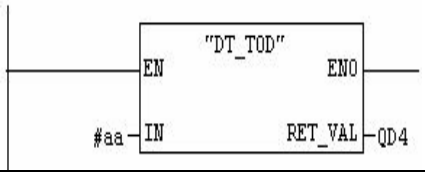
دوشنبه = ۲

.

.

شنبه = ۷

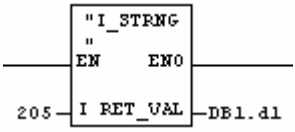
Network 4 : Title:




توسط **FC8** (از IEC) با نام سمبلیک DT-DATE می توان زمان را جدا کرد.

**FC16 با نام سمبلیک I_String**

این فانکشن عدد صحیح Integer را به String یعنی ترکیبی از کاراکترها تبدیل می کند.




در استفاده از String لازم است به نکات زیر توجه شود:

- String را میتوان در جدول متغیرهای محلی یا در دیتا بلاک تعریف کرد.
- در هنگام تعریف String طول آن نیز بایستی مشخص گردد مثلاً در String[10] طول ۱۰ کاراکتر مشخص شده است.
- ماکزیمم طول String میتواند ۲۵۴ کاراکتر یا بعبارت دیگر ۲۵۴ بایت باشد.
- دو بایت اول هر String برای طول آن رزرو شده است. بایت اول طول تعریف شده و بایت دوم طول واقعی را در خود ذخیره می کند.

شکل زیر DB1 را که سطر اول آن از نوع String با طول ۱۰ تعریف شده را نشان می دهد.

| Address | Name | Type | Initial value |
|---|---|---|---|
| 0.0 | | STRUCT | |
| +0.0 | d1 | STRING[10] | '' |
| =12.0 | | END_STRUCT | |

همانطور که مشاهده می شود کل حجم دیتا بلاک 12 بایت است یعنی ۲ بایت بیشتر از دیتای تعریف شده.

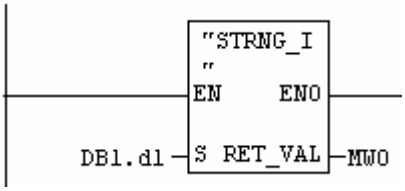
اکنون اگر این دیتا بلاک را همرا با فانکشن نشان داده شده در بالا به PLC دانلود کنیم عدد ۲۰۵ بصورت کاراکتری '+205' در بیتهای دیتا بلاک ذخیره خواهد شد.از آنجا که امکان مشاهده Online متغیر String در دیتا بلاک وجود ندارد بیت ها را بصورت مجزا توسط VAT مانند شکل زیر مانیتور می کنیم.همانطور که مشاهده می شود دو بایت اول طول تعریف شده و طول واقعی String را نشان می دهند.

VAT_1 -- @S7_Pro11\SIMATIC 300(2)\CPU

| | Address | Symbol | Display forma | Status value |
|---|---|---|---|---|
| 1 | DB1.DBB 0 | | DEC | 10 |
| 2 | DB1.DBB 1 | | DEC | 4 |
| 3 | DB1.DBB 2 | | CHARACTER | '+' |
| 4 | DB1.DBB 3 | | CHARACTER | '2' |
| 5 | DB1.DBB 4 | | CHARACTER | '0' |
| 6 | DB1.DBB 5 | | CHARACTER | '5' |

**FC38 با نام سمبلیک String_I**

این فانکشن String را به عدد صحیح Integer تبدیل می کند. بدیهی است در مواردی کاربرد دارد که محتویات String بصورت عدد باشد.




اگر در مثال فوق توسط VAT مقادیری را بصورت کاراکتر به بایت های دوم و سوم از DB1 نسبت دهیم معادل عدد صحیح آنها را در خروجی MW0 خواهیم دید.

VAT_1 -- @S7_Pro11\SIMATIC 300(2)\CPU 314C-2 DP\

| | Address | Symbol | Display forma | Status value | Modify value |
|---|---|---|---|---|---|
| 1 | DB1.DBW 2 | | CHARACTER | '45' | '45' |
| 2 | MW 0 | | DEC | 45 | |

**تذکر**:سایر FC های تبدیل STRING به REAL یا DINT یا تبدیل معکوس نیز بهمین نحو عمل می کنند صرفاً بایستی توجه داشت که:

- در تبدیل String به عدد INT اگر طول String صفر یا بزرگتر از ۶ باشد تبدیل انجام نمی شود.
- در تبدیل String به عدد DINT اگر طول String صفر یا بزرگتر از ۱۱ باشد تبدیل انجام نمی شود.
- در تبدیل String به عدد Real طول String نباید کمتر از ۱۴ کاراکتر بوده و لازمست مطابق ساختار زیر باشد:

| | | |
|---|---|---|
| ±v.nnnnnnnE±xx | ± | Sign |
| | v | 1 digit before the decimal point |
| | n | 7 digits after the decimal point |
| | x | 2 exponential digits |

- در تبدیل Time به S5time اگر زمان بزرگتر از TIME#02:46:30.000 که حد ماکزیمم S5T است باشد در اینصورت S5TIME#999.3 خواهد بود.

### ۱۲-۳-۲ فانکشنهای خانواده String از IEC

این فانکشنها همانطور که در لیست قبلی مشاهده شد برای انجام مقایسه و عملیات دیگر روی String بکار می روند.توسط این فانکشن هاانواع مقایسه را میتوان روی String انجام داد همه فانکشنهای مقایسه دو ورودی دارند که آدرس دو String به آنها داده می شود و خروجی آنها از جنس BOOL است اگر نتیجه مقایسه درست باشد خروجی یک خواهد شد. سایر کارهای دیگر مرتبط با رشته ها مانند جدا کردن بخشی از سمت چپ یا راست یا از وسط و ریختن در String دیگر و نیز مواردی مانند ترکیب رشته ها و بدست آوردن طول رشته قابل انجام است. در اینجا صرفاً مثالی از کاربرد دو فانکشن از این خانواده ارائه می گردد.

برنامه زیر کاربرد فانکشن FC2 برای ترکیب دو String و نیز فانکشن FC21 برای مشخص کردن طول String را نشان می دهد. خروجی برگردانده شده توسط FC21 عدد ۵ می باشد. در این برنامه سه متغیر از جنس Stringبصورت Temp در بالای بلاک تعریف شده اند دقت شود که چگونه توسط برنامه نویسی طول کلی ، طول واقعی و مقادیر به این متغیرها نسبت داده شده است.

| | |
|---|---|
| در این قسمت طول کلی متغیر mystring1 به اندازه ۲۰۰ و طول واقعی آن به اندازه ۱ تعریف شده سپس کاراکتر x به متغیر بار شده است | LAR1 P##myString1<br>L 200<br>T LB [AR1,P#0.0]<br>L 1<br>T LB [AR1,P#1.0]<br>L 'x'<br>T LB [AR1,P#2.0] |
| در این قسمت طول کلی متغیر mystring2 به اندازه 100 و طول واقعی آن به اندازه 4 تعریف شده سپس رشته reza به متغیر بار شده است | LAR2 P##myString2<br>L 100<br>T LB [AR2,P#0.0]<br>L 4<br>T LB [AR2,P#1.0]<br>L 'reza'<br>T LD [AR2,P#2.0] |
| در این قسمت طول کلی متغیر mystring3 به اندازه 300 و طول واقعی آن به اندازه 10 تعریف شده | LAR2 P##mystring3<br>L 300<br>T LB [AR2,P#0.0]<br>L 10<br>T LB [AR2,P#1.0] |
| توسط FC2 دو متغیر اول با هم ترکیب شده و در متغیر سوم ریخته شده اند. | CALL "CONCAT"<br>IN1 :=#myString2<br>IN2 :=#myString1<br>RET_VAL:=#mystring3 |
| توسط FC21 طول متغیر mystring3 بدست آورده شده است. | CALL "LEN"<br>S :=#mystring3<br>RET_VAL:=MW0 |

### ۱۲-۳-۳ فانکشنهای خانواده Floating Point از IEC

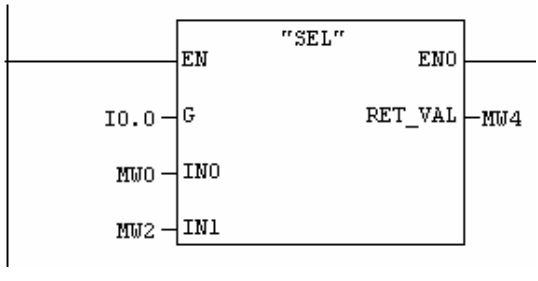
با توجه به لیست قبلی مشاهده می شود که این فانکشن ها عملیات مختلفی را انجام می دهند یکی از کاربرد های آنها در اضافه و کم کردن تاریخ و زمان است.

در مثال زیرکه در OB راه اندازی نوشته شده زمان CPU با SFC1 خوانده می شود سپس توسط FC1 از IEC یک ساعت به آن اضافه شده وزمان جدید مجدداً توسط SFC0 به PLC بار می شود.

```
CALL  "READ_CLK"
   RET_VAL:=MW0
   CDT   :=#oldtime

CALL  "AD_DT_TM"
   T     :=#oldtime
   D     :=T#1H
   RET_VAL:=#newtime

CALL  "SET_CLK"
   PDT   :=#newtime
   RET_VAL:=MW2
```

از دیگر قابلیتهای این خانواده محاسبه مینیمم و ماکزیمم بین چند ورودی است اینکار توسط فانکشن های FC25 و FC27 انجام می شود که سه ورودی همجنس از نوع INT یا DINT یا REAL می گیرند و ماکزیمم یا مینیمم را به خروجی میفرستند. فانکشن دیگری با نام FC22 نیز وجود دارد که برای محدود سازی ورودی بین دو حد بالا و پایین است حدود بالا و پایین و متغیر مورد نظر که لازمست هم جنس باشند به ورودی فانکشن داده می شوندو مقدار محدود شده در خروجی قابل استفاده است.

فانکشن دیگری برای امکان انتخاب بین دو ورودی به نام FC36 وجود دارد که در شکل زیر نشان داده اگر ورودی G صفر باشد ورودی IN0 و اگر G یک باشد IN1 به خروجی اعمال می گردد.



### ۱۲-۳-٤ فانکشنهای خانواده Date and Time از IEC

این فانکشنها برای انجام مقایسه دو تاریخ زمان بکار می روند مساوی بودن یا نبودن و کوچکتر یا بزرگتر بودن توسط آنها قابل مقایسه است.

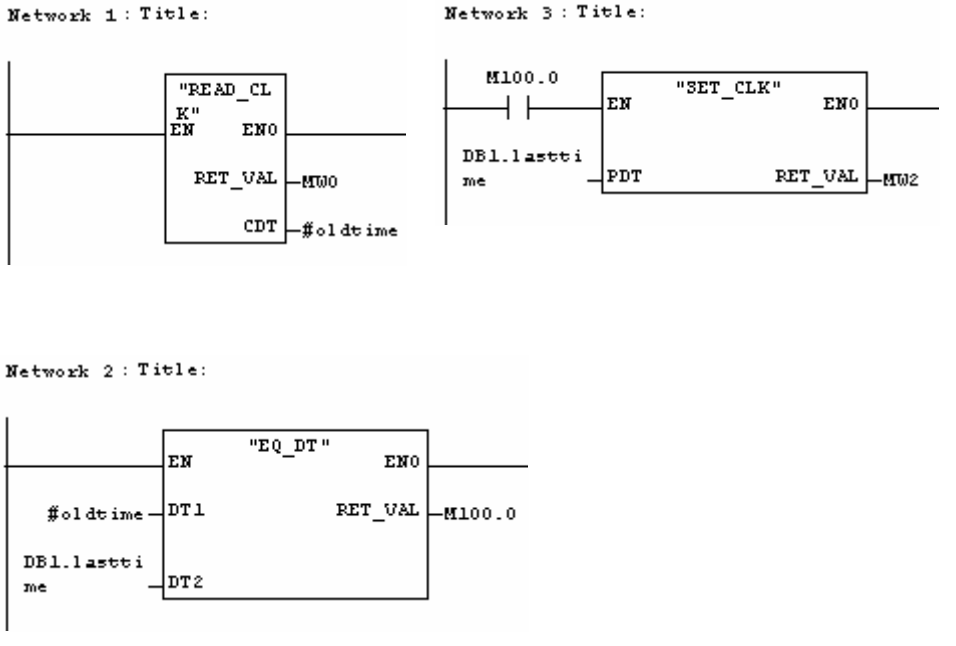
بدلیل تشابه برنامه نویسی این فانکشنها صرفاً یکی از آنها یعنی FC9 که برابر بودن دو تاریخ زمان را چک می کند را با ذکر مثال توضیح می دهیم.

فرض کنید اکنون در شش ماهه دوم سال ۱۳۸۵ شمسی قرار داریم برنامه ای بنویسید که ساعت CPU را از آخرین زمان آخرین روز سال که معادل 2007-3-20 میلادی است یکساعت به جلو ببرد.

ابتدا در یک دیتا بلاک تاریخ و زمان آخرین روز و تاریخ زمان جدید را مانند شکل زیر تعریف می کنیم.

| Address | Name | Type | Initial value |
|---|---|---|---|
| 0.0 | | STRUCT | |
| +0.0 | lasttime | DATE_AND_TIME | DT#07-3-20-23:59:59.000 |
| +8.0 | newtime | DATE_AND_TIME | DT#07-3-21-1:0:0.000 |

در برنامه متغیر محلی از جنس Date_and_Time بنام oldtime تعریف می کنیم و برنامه زیر را می نویسیم:



# فصل سیزدهم - برنامه نویسی توسط S7-SCL

**مشتمل بر :**


**۱۳-۱ مقدمه**

**۱۳-۲ آشنایی با محیط نرم افزار S7-SCL**

**۱۳-۳ شروع کار با S7-SCL**

**۱۳-٤ ساختار یک برنامه S7-SCL**

**۱۳-۵ نحوه تعریف بلاک ها**

**۱۳-٦ نحوه تعریف ثوابت و Label**

**۱۳-۷ نحوه بکار بردن متغیرهای حافظه CPU**

**۱۳-۸ Operation و Experession در SCL**

**۱۳- ۹ دستورات SCL**

۱۳-۹-۱ دستورات اختصاص مقادیر Value Assignment

۱۳-۹-۲ دستورات کنترلی

۱۳-۹-۳ دستورات صدا زدن FB و FC

۱۳-۹-۴ دستورات کانترها

۱۳-۹-۵ دستورات تایمرها

۱۳-۹-۶ فانکشن های استاندارد SCL

**۱۳-۱۰ مثال هایی از برنامه نویسی SCL**

**۱۳-۱۱ امکانات Debug در SCL**

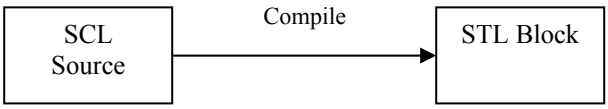
### ۱۳-۱ مقدمه

در جلد اول کتاب سه زبان برنامه نویسی LAD و STL و FBD مورد بحث قرار گرفت و اشاره شد که طبق استاندارد IEC1131-3 دو زبان برنامه نویسی برای PLC ها علاوه بر زبانهای فوق طراحی شده است. یکی از این دو زبان در استاندارد مزبور زبان ST است که مخفف Structured Text یا دستورات ساختار یافته است. این زبان در نرم افزار Step7 با نام S7-SCL عرضه شده که SCL مخفف Structured Control Language میباشد. S7-SCL زبان برنامه نویسی سطح بالاست و دستورات آن به زبان پاسکال نزدیک است. منطق های پیچیده کنترلی در این روش بسهولت برنامه نویسی می شوند و برای عملیات و محاسبات پیچیده ریاضی بسیار مناسب است.

نکته ای که در همین جا لازم است خاطر نشان گردد اینست که بلاک های مختلف برنامه نویسی PLC میتوانند به هر کدام از ۵ زبان فوق الذکر نوشته شوند و الزامی وجود ندارد که همه یک دست باشند. شکل زیر که با انتخاب منوی View > Detail در Simatic Manager ظاهر میشود زبانهای مختلف استفاده شده در بلاک های یک پروژه را نشان میدهد.

| Object name | Symbolic name | Created in language |
|---|---|---|
| OB1 | MAIN | STL |
| OB35 | CYC_INT5 | STL |
| OB100 | COMPLETE RESTART | FBD |
| OB122 | MOD_ERR | STL |
| FB1 | LOOP | GRAPH |
| FC1 | VALV | SCL |

برنامه SCL بصورت Source نوشته میشود. این Source بعد از کامپایل شدن به بلاکSTL تبدیل میشود .آنچه به PLC دانلود میشود بلاک است و Source روی هارد کامپیوتر باقی می ماند. توجه شود که تبدیل یک طرفه است یعنی نمیتوان STL Block را به SCL Source تبدیل کرد.



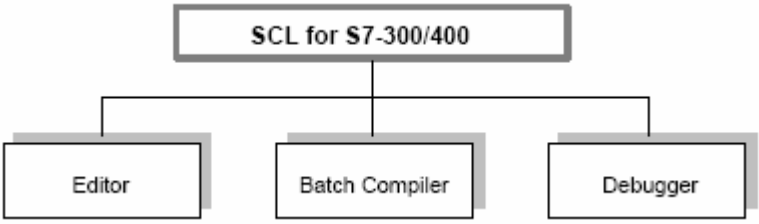

در اینحالت اگر روی بلاک STL دابل کلیک کنیم برنامه Source را خواهیم دید و در عین حال میتوانیم با Open کردن بلاک در زیر برنامه LAD/STL/FBD دستورات STL را مشاهده نماییم توجه گردد که اگر

کوچکترین تغییری در برنامه بدهیم و ذخیره کرده بلاک را ببندیم دیگر نخواهیم توانست با دابل کلیک روی آن Source را مشاهده کنیم.

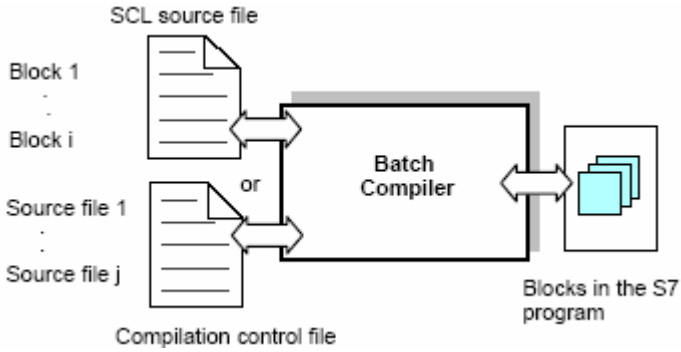
وقتی برنامه Source را در دست داریم میتوانیم با کامپایل مجدد بلاک را بسازیم حتی اگر بلاک را بصورت دستی از پوشه Blocks پاک کنیم خواهیم دید که با کامپایل کردن Source مجدداً بلاک ساخته میشود.

بلاک SCL را میتوان همراه با سایر بلاک هایی که به زبانهای LAD و STL و FBD نوشته شده بکار برد بعبارت دیگر در SCL میتوان تمام بلاک هایی که به زبانهای فوق نوشته شده را صدا زد و بالعکس نیز می توان از بلاک های LAD/STL/FBD بلاک SCL را فراخوان نمود.

نرم افزار SCL متشکل از سه قسمت اصلی است که در شکل زیر نشان داده شده است.

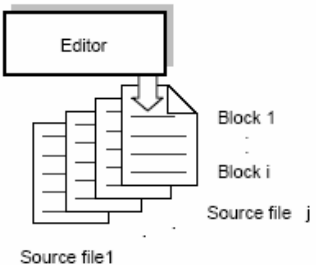



کاربر برنامه خود را در محیط Editor می نویسد برنامه پس از کامپایل کردن توسط Batch Compiler تبدیل به کدهای ماشین میشود . این کدها روی تمام CPU های S7-300 و S7-400 از CPU314 به بعد قابل اجرا هستند. . در برنامه Source اشکالات خط به خط چک نمی شوند بلکه پس از کامپایل کردن لیست میگردند.




Debugger نیز ابزاری است که توسط آن میتوان در محیط Editor برنامه نوشته شده به زبان SCL را خط به خط اجرا نمود و اشکالات منطقی آنرا پیدا کرد.

در محیط Editor میتوان تمام بلاک های OB و FB و FC و DB و UDT را به یکی از دو روش زیر ایجاد کرد که در شکل نیز نمایش داده شده است:

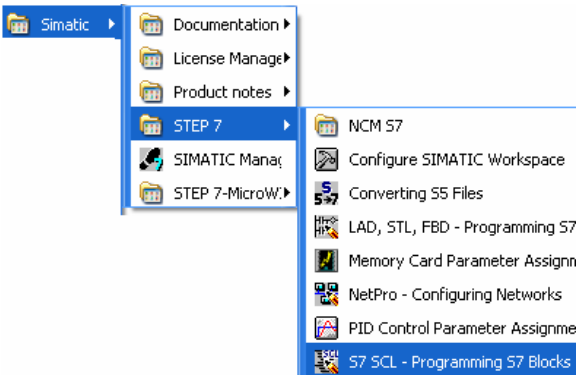



۱- همه بلاک ها در یک فایل Source باشند.

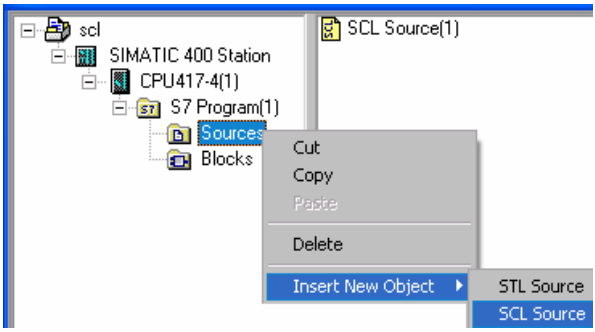
۲- برای هر بلاک یا هر چند بلاک یک فایل Source وجود داشته باشد.

برای نوشتن برنامه در S7-SCL به دو طریق میتوان عمل کرد:

۱- باز کردن برنامه S7-SCL از طریق محیط ویندوز و مسیرزیر مطابق شکل زیر Start> Siamtic > step7

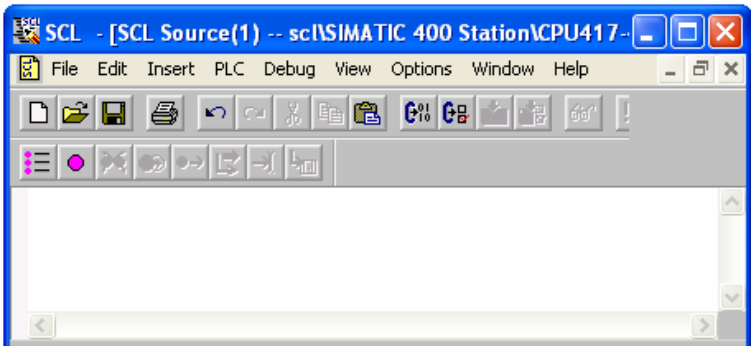



۲- با راست کلیک روی پوشه Source که در پروژه در زیر مجموعه S7 Program ظاهر میشود و انتخاب SCL Source مطابق شکل زیر. در اینحالت پس از انتخاب SCL Source یک فایل Source در پوشه Source ظاهر میشود که نام آن توسط برنامه بصورت پیش فرض داده میشود در عین حال کاربر میتواند نام دلخواه برای آن انتخاب نماید. باکلیک کردن روی نام این فایل برنامه S7-SCL اجرا خواهد شد.

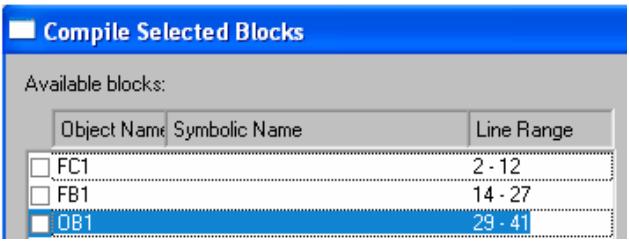


## ۱۳-۲ آشنایی با محیط نرم افزار S7-SCL

پس از اجرای برنامه SCL به یکی از دو طریقی که در صفحه قبل ذکر شد برنامه S7-SCL اجرا شده و پنجره نرم افزار بصورت زیر ظاهر می گردد:




**منوی File** : عمده امکانات این منو برای کاربر آشناست صرفاً در مورد کامپایل که از گزینه های این منو میباشد توضیحاتی داده میشود. فایل Source ممکن است شامل چندین بلاک باشد که بترتیب زیر هم قرار گرفته اند. در اینحالت اگر گزینه File > Compile انتخاب شود تمامی بلاک ها کامپایل میشوند . میتوان برای این منظور از آیکون که در Toolbar بالای نرم افزار موجود است نیز استفاده کرد. گزینه دیگر که برای کامپایل کردن در منوی File وجود دارد File>Compile Selected Block است. با این گزینه که در عین حال از طریق آیکون بالای پنجره نیز قابل استفاده است پنجره ای مانند شکل زیر ظاهر میشود که توسط آن می توان انتخاب کرد کدامیک از بلاکها کامپایل شوند.



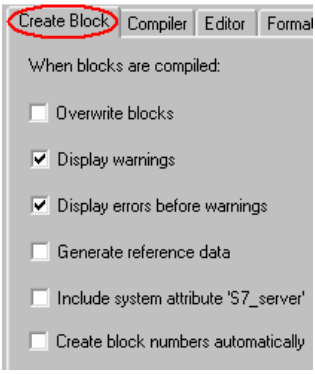

**منوی Edit** : برای کاربر آشناست و نیاز به توضیح ندارد.

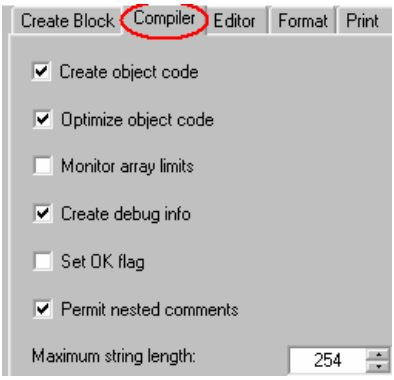
**منوی Insert** : توسط این منو میتوان بلاک دیگری را در برنامه Source صدا زد و یا از قالب های از قبل تهیه شده برای بلاک ها در برنامه Source استفاده کرد . بعلاوه میتوان برخی از قالبهای تهیه شده را برای برخی از دستورات SCL استفاده نمود. این منو در خلال بحث های بعدی تشریح خواهد شد.

**منوی PLC** : این منو مشابه منوی PLC در زیر برنامه های LAD/STL/FBD و HWconfig است و نیاز به توضیح ندارد.

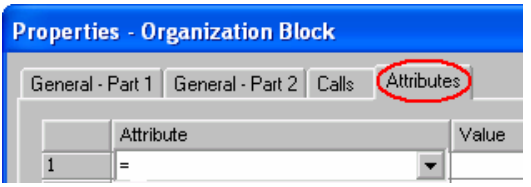
**منوی Debug** : توسط این منومیتوان عملیات Debuging یعنی اجرای خط به خط برنامه را انجام داد که در بخش های بعدی مورد بحث قرار خواهد گرفت.

**منوی Option** : در این منو قسمت های Reference Dataو Symbol Table مشابه آنچه در سایر زیربرنامه های Step7 وجود دارد می باشد.ولی گزینه Customize مربوط به تنظیمات خود SCL است و با انتخاب آن پنجره ای ظاهر میشودکه بخشهای مختلف دارد اگر قسمت های Editor و Format و Print انتخاب شوند عملکرد گزینه های داخل آن با کمی دقت مشخص است که در اینجا از بیان آنها صرفنظر میکنیم و صرفاً به توضیح در مورد دو بخش آن که در شکل های زیر آمده اکتفا می نماییم.




**قسمت Create Block** : مربوط به تنظیمات ایجاد بلاک در هنگام کامپایل است اگر Overwrite blocks فعال شود در هنگام ایجاد بلاک اگر بلاکی به همین نام از قبل وجود داشته باشد بدون اینکه در مورد نوشتن روی آن از کاربر سوال کند آنرا overwrite مینماید. دو گزینه بعدی همانطور که در شکل مشاهده می شود بصورت پیش فرض فعال هستند .این دو گزینه مربوط به Warning ها و Error هایی هستند که در زمان کامپایل در پنجره پایین نرم افزار ظاهر میشوند. Warning ها مانع تبدیل نیستند ولی Error ها تا برطرف نشوند بلاک ساخته نمیشود.گزینه Generate Ref. data اگر فعال باشد موجب میگردد که در زمان ساخته شدن بلاک جدول رفرنس آن نیز ساخته شوند این جدول در جلد اول کتاب توضیح داده شد. گزینه Include System attribute موجب میشود که برخی ویژگی ها که برای ارتباطات و ارسال پیام ها مفید هستند و برای Server مورد استفاده قرار می گیرند به بلاک اضافه شوند. اگر این گزینه را فعال کنیم وسپس با کلیک راست روی




بلاک ساخته شده Properties آن را ببینیم قسمت Attribute را مشاهده خواهیم کرد که به ویژگی ها اضافه شده است.

**قسمت Compiler** که در شکل صفحه قبل نشان داده شد مربوط به تنظیمات کامپایل است و شامل گزینه های مختلف می باشد.گزینه Create Object code بصورت پیش فرض فعال است و کدهای اجرایی را در هنگام کامپایل شدن تولید میکند اگر غیر فعال شود کامپایل کردن بصورت Syntax Check در می آیدیعنی بعد از تکمیل هر خط چک میشود که آیا دستور درست وارد شده یا خیر. گزینه Optimize Object code باعث میشود که کدها بصورت بهینه از نظر حجم حافظه برای PLC تولید شوند و بهتر است همیشه فعال باشد.گزینه Monitor Array Limit اگر فعال شود PLC در حال اجرا چک میکند که آیا آرایه در حد تعیین شده قرار دارد یا خیر و در صورتی که خارج از رنج باشد فلگ OK را بصورت False در می آورد این فلگ توسط گزینه Set OK Flag فعال می شود و امکان استفاده از آن را در برنامه فراهم میسازد. اگرچه هنوز به معرفی دستورات SCL نپرداخته ایم ولی بد نیست در اینجا خواننده با نحوه استفاده از فلگ OK آشنا شود. بطور کلی این فلگ برای چک کردن اینکه دستورات بلاک درست انجام شده یا خیر بکار میرود. در ابتدای بلاک این فلگ true شده و وقتی در یک سطر از برنامه اشکالی پیش بیاید (مثلاً تقسیم بر صفر ) این فلگ False میشود وضعیت فلگ در خروجی ENO بلاک قابل مانیتور کردن است . برنامه زیرنمونه ای از کاربرد فلگOK را نشان میدهد توجه نماییدسطرهایی که با علامت // شروع شده اند دستورات تلقی نمیشوند و Comment هستند.

```
// Set OK flag to TRUE   to check whether
//the action executes correctly.
OK:= TRUE;
Division:= 1 / IN;
IF OK THEN
    // Division was correct.
ELSE
 // Division was not correct.
  END_IF;
```

گزینه دیگر Creat Debug Info است که اگر فعال شود پس از تکمیل کامپایل و دانلود به PLC امکان تست توسط Debugger را فراهم می سازد. گزینه دیگر Maximum String Length است . اگر در برنامه Source از متغیرهای نوع String استفاده شود حداکثر تعداد آنها در اینجا تعریف می شود که بصورت پیش فرض ۲۵۴ است. و آخرین گزینه Permit nested comment است . در برنامه SCL به دو روش میتوان Comment نوشت روش اول Comment در یک سطر است که شبیه مثال قبل کافیست آن سطر با علامت // شروع شود. روش

دوم که برای Comment های طولانی مناسب است استفاده از *) در شروع اولین سطر و استفاده از (* در انتهای آخرین سطر است . مانند

```
(* This is an example of a comment section,
   that can extend over several lines.*)
```

دراینحالت اگر در داخل متن پرانتزهای تودرتو استفاده شود و گزینه فوق فعال نباشد در موقع کامپایل خطا میگیرد.بصورت پیش فرض این گزینه فعال است و در اینحالت مشکلی نخواهد بود.

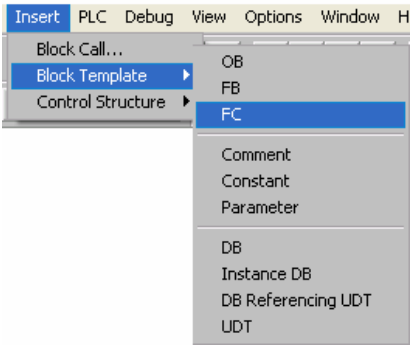
**تذکر**: قسمتهای دیگر که در پنجره Customize وجود دارد مانند Editor و Format Print که برای تنظیمات فونت و رنگ و امثال آنست برا ی `کاربر آشناست و از تشریح آنها صرفنظر کرده صرفاً به رنگهای مختلفی که برای برنامه در محیط SCL بکار میروند اشاره میکنیم این رنگها بصورت پیش فرض مطابق جدول زیر هستند ولی کاربر میتواند آنها را از طریق قسمت Format در پنجره Customize تغییر دهد.

| رنگ | موضوع | مثال |
|---|---|---|
| آبی | Keywords | ORGANIZATION_BLOCK |
| آبی | دیتاهای از قبل تعریف شده | INT |
| آبی | فانکشن های استاندارد | ENO |
| نارنجی | دستورات | BOOL_TO_WORD |
| صورتی | مقادیر ثابت و اعداد | NOT |
| سبز | توضیحات (Comment) | TRUE |
| بنفش | سمبل های اشتراکی | //... or (*...*) |
| سیاه | متن های معمولی | "Motor" |

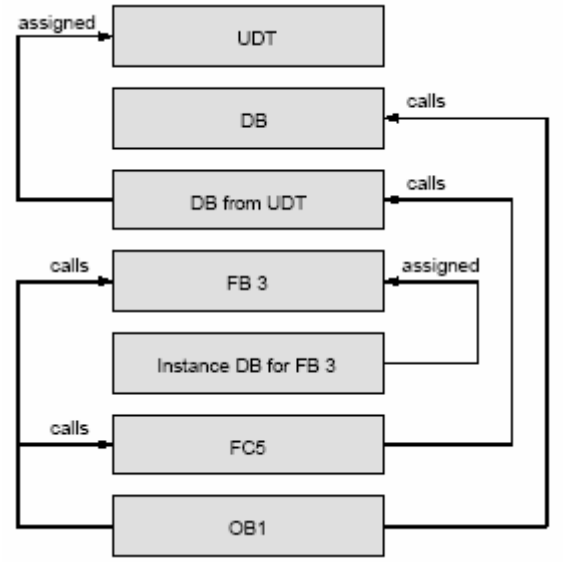
## ۱۳-۳ شروع کار با S7-SCL

قبل از شروع برنامه نویسی و کار با SCL لازم است نکات زیر مد نظر قرار گیرد:

- حروف بزرگ و کوچک در نوشتن دستورات یکسان تلقی میشوند.
- علامت Semicolon لازم است در انتهای تمام دستورات قرار گیرد علامت ;
- تمام انواع بلاک های برنامه نویسی شامل OB و FB و FC قابل ایجاد کردن هستند.
- DB و UDT را نیز می توان توسط SCL ایجاد کرد.
- هر کدام از بلاک های فوق داری ساختار از پیش تعیین شده ای هستند این ساختار را می توان از منوی Insert>Block Tempelet مانند شکل روبرو وارد برنامه نمود.



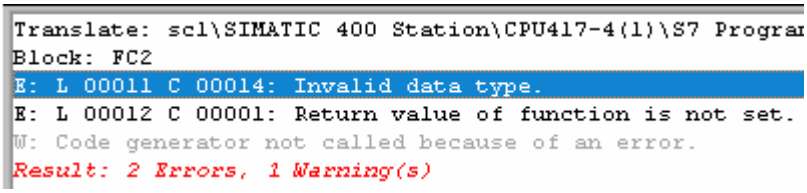
- DB Instance را اگر چه میتوان با SCL ایجاد کرد ولی نیازی به این کار نیست زیرا این DB بطور اتوماتیک در هنگام صدا زدن FB ایجاد میشود.
- ترتیب بلاک ها در برنامه SCL مهم است. بلاکی که فراخوان شده بایستی قبلاً در سطرهای ماقبل تعریف شده باشد.
- DB های اشتراکی لازم است قبل از اتمام بلاک هایی که از آدرس های آن DB استفاده میکنند تعریف شده باشد.
- UDT لازم است قبل از DB یا بلاکی که از آن استفاده میکند تعریف شود.
- نمونه ساختار فراخوانی بلاک ها در شکل زیر آمده است:




- Constant و Parameter که در منوی Insert>Templete مشاهده میشود برای وارد کردن پارامترهای ورودی و خروجی بلاکهای FB و FC استفاده میگردند که بعداً تشریح خواهند شد.
- برای فراخوانی یک بلاک در برنامه میتوان از منوی Insert> call استفاده کرد سپس بلاک مورد نظر را از مسیرآن در پنجره ای که ظاهر میشود انتخاب نمود.
- بلاک های موجود در Library مانند SFC و SFB ها نیز از طریق مسیر فوق الذکر صدا زده میشوند.

- ساختار برخی از دستورات کنترلی مانند IF و Case و For و ... از طریق منوی Insert>Control Structure قابل استفاده است.
- اگر کل بلاک ها در یک فایل Source نوشته شوند بهتر است بترتیب و از طریق Partial Compile کامپایل شوند تابتوان اشکالات آنها را بترتیب رفع نمود.
- اگر بلاک های مختلف در فایلهای Source جداگانه نوشته شوند لازم است این Source ها بترتیب کامپایل شوند بعنوان مثال اگر FC1 از داخل OB1 فراخوان شده بایستی فایل Source مربوط به FC1 را ابتدا کامپایل کرد.
- میتوان چندین فایل Source را بصورت یکجا کامپایل نمود برای این کار بایستی یک فایل SCL Compile Control با راست کلیک کردن روی پوشه Source مانند شکل زیر ایجاد کرده سپس آنرا باز کرده و اسامی فایلهای Source که قرار است بدنبال هم کامپایل شوند را در آن بنویسیم. این فایلها لازم است در پوشه Source موجود باشند.

- پس از کامپایل لیست Error ها و Warning ها در پنجره پایین برنامه ظاهر میشوند. Error با حرف E و Warning با حرف W ظاهر میشود پس از آن شماره سطر و ستون مربوط به Errorبترتیب با حروف L و C نمایش داده میشوند. با دابل کلیک کردن روی هر کدام از Error ها Cursur به آن سطر پرش کرده و با زدن کلید F1 می توان در مورد نوع اشکال اطلاعاتی را بدست آورد. توصیه میشود همیشه اولین Error را چک و برطرف کنید زیرا در بسیاری از حالات با رفع شدن اولین اشکال و کامپایل مجدد بسیاری از اشکالات بعدی نیز رفع میشوند.



- کاراکترهایی که در برنامه SCL بکار می روند میتوانند حروف یا اعداد یا ترکیبی از اینها باشند ولی کاراکترهایی که در جدول زیر آمده است معانی خاص دارند:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| + | - | * | / | = | < | > | [ | ] | ( | ) |
| : | ; | $ | # | " | ' | { | } | % | . | , |

- کلمات زیر رزرو شده هستند و کاربر نمی تواند متغیری به این اسم چه با حروف کوچک و چه با حروف بزرگ در برنامه تعریف کند. این کلمات که Keyword خوانده میشوند در برنامه با رنگ آبی ظاهر میشوند

| | | |
|---|---|---|
| AND | END_CASE | ORGANIZATION_BLOCK |
| ANY | END_CONST | POINTER |
| ARRAY | END_DATA_BLOCK | PROGRAM |
| AT | END_FOR | REAL |
| BEGIN | END_FUNCTION | REPEAT |
| BLOCK_DB | END_FUNCTION_BLOCK | RETURN |
| BLOCK_FB | END_IF | S5TIME |
| BLOCK_FC | END_LABEL | STRING |
| BLOCK_SDB | END_TYPE | STRUCT |
| BLOCK_SFB | END_ORGANIZATION_BLOCK | THEN |
| BLOCK_SFC | END_REPEAT | TIME |
| BOOL | END_STRUCT | TIMER |
| BY | END_VAR | TIME_OF_DAY |
| BYTE | END_WHILE | TO |
| CASE | ENO | TOD |
| CHAR | EXIT | TRUE |
| CONST | FALSE | TYPE |
| CONTINUE | FOR | VAR |
| COUNTER | FUNCTION | VAR_TEMP |
| DATA_BLOCK | FUNCTION_BLOCK | UNTIL |
| DATE | GOTO | VAR_INPUT |
| DATE_AND_TIME | IF | VAR_IN_OUT |
| DINT | INT | VAR_OUTPUT |
| DIV | LABEL | VOID |
| DO | MOD | WHILE |
| DT | NIL | WORD |
| DWORD | NOT | XOR |
| ELSE | OF | Names of the standard functions |
| ELSIF | OK | |
| EN | OR | |

- متغیرها که اصطلاحاً Identifier خوانده می شوند میتوانند ماکزیمم ۲۴ کاراکتر باشند و لازم است با حرف شروع گردند نه با عدد و در وسط اسم آنها فضای خالی یا کاراکترهای خاص وجود نداشته باشد. در جدول زیر نمونه هایی از کاراکترهای مجاز و غیر مجاز آورده شده است:

| نام مجاز | نام غیر مجاز |
|---|---|
| y12<br>Temperature<br>Surface<br>Table | 4th<br>Array<br>S Value |

- از کلماتی که برای بلاک ها ، آدرس ها ، تایمرها و کانتر ها استفاده میشود نمیتوان بعنوان متغیر استفاده کرد بعنوان مثال متغیرهایی مانند T1 یا FB1 نمیتوان استفاده کرد این موارد را Standard Identifier میخوانند که در جدول زیر لیست آنها آمده است . منظور از x شماره یا آدرس می باشد.

| | |
|---|---|
| DBx | SFCx |
| FBx | SFBx |
| FCx | Tx |
| OBx | UDTx |
| SDBx | Cx |

- در هر نقطه از برنامه می توان آدرسهای حافظه CPU مشابه آنچه در برنامه های LAD/STL/FBD استفاده میشد را بکار برد مانند : M , MB , MW , MD , Q , QB , QW , QD , PQW , I , IB ,IW , ID , PIW

## ۱۳-٤ ساختار یک برنامه S7-SCL

همانطور که اشاره گردیدتمام بلاکهای Step7 را میتوان توسط SCL ایجاد کردد هر کدام از بلاک ها در ابتدا و انتها توسط کلمات خاصی مشخص میشوند که در جدول زیر معرفی گردیده اند:

| Identifier | Block Type | Syntax |
|---|---|---|
| Function block | FB | FUNCTION_BLOCK fb_name . . .<br>END_FUNCTION_BLOCK |
| Function | FC | FUNCTION fc_name : function type . . .<br>END_FUNCTION |
| Organization block | OB | ORGANIZATION_BLOCK ob_name . . .<br>END_ORGANIZATION_BLOCK |
| Data block | DB | DATA_BLOCK db_name . . .<br>END_DATA_BLOCK |
| Shared data type | UDT | TYPE udt_name . . .<br>END_TYPE |

در عین حال کاربر نیازی به نوشتن این کلمات ندارد و میتواند با استفاده از منوی Insert>Block Templeteآنها را وارد برنامه نماید. در اینحالت بجای شماره بلاک حروف xxx ظاهر میشود که لازم است کاربر صرفاً شماره

مورد نظر را بجای آنها بنویسد.نکته قابل ذکر دیگر اینست که بجای کلماتی که قبل از نام بلاک ها نوشته میشود میتوان از کلمه Program و بجای عبارت انتهای بلاک میتوان End_Program استفاده کرد. این موضوع بدلیل انطباق با استاندارد IEC1131 منظور شده است بعنوان مثال برای تعریف FB1 میتوان هر کدام از روش های زیر را بکار برد:

| | | |
|---|---|---|
| Program FB1 | یا | Function_Block FB1 |
| End_Program | | End_Function_Block |

بطور کلی هر بلاک از سه قسمت تشکیل شده است . بخش اول برای تعریف ویژگی های بلاک که در ابتدا قرار میگیرد بخش دوم برای تعریف متغیرها ست و نهایتاً بخش برنامه بلاک است این بخشها برای OB1در شکل روبرو نشان داده شده اند.

| Organiazition_Block OB1 |
|---|
| تعریف ویژگی های بلاک |
| تعریف متغیرهای Temp |
| برنامه |
| End_Organization_Block |

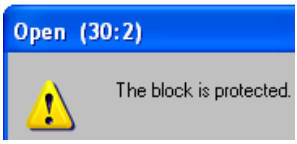
**Attribute بلاک**

ویژگی های بلاک مانند نام بلاک ، توضیحات ، نام تهیه کننده ، شماره Version و امثال آن را می توان در بخش Attribute بلاک تنظیم کرد. برای این ویژگی ها نیز قالبی وجود دارد که از طریق منوی Insert>Block Templete>Comment میتوان آنرا به برنامه بعد از سطر اول مانند شکل روبرو وارد کرد. توضیح این ویژگی ها همراه با مثال در جدول زیر آمده است .

```
FUNCTION_BLOCK FB1

TITLE = 'Block Title'
//Block Comment...
VERSION: '1.0'
AUTHOR: Author
NAME: Name
FAMILY: Family
```

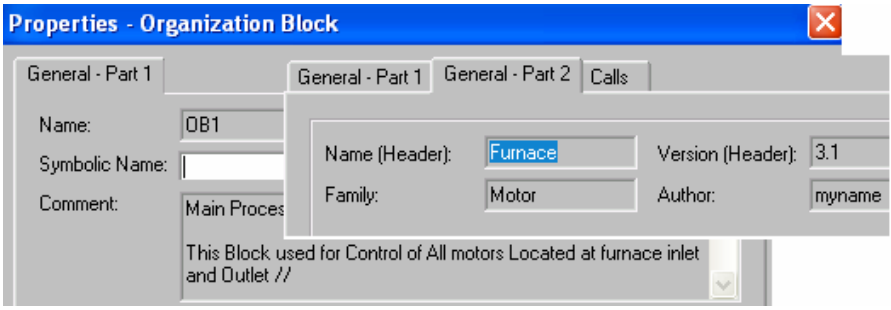
| پارامتر | توضیح | مثال |
|---|---|---|
| **TITLE**='printable characters' | عنوان بلاک که لازم است داخل ' ' قرار گیرد | TITLE='SORT' |
| **VERSION :**'decimal digit string. decimal digit string' | شماره Version بلاک شامل دوعدد که بین آنها با نقطه جدا میشود هر کدام از این اعداد میتوانند بین 0 تا 15 باشند. عدد منتجه برای بلاکها بجز DB داخل ' ' قرار می گیرد . | VERSION : '3.1'<br>//With a DB:<br>VERSION : 3.1 |
| **KOW_HOW_PROTECT** | Protect کردن بلاک | KNOW_HOW_PROTECT |
| **AUTHOR :** | نام شرکت یا قسمت یا تهیه کننده یا سایر اسامی | AUTHOR : myCompany |
| **NAME :** | نام بلاک | NAME : PID |
| **FAMILY :** | نام خانواده ای که بلاک به آن متعلق است تعریف خانواده امکان جستجوی سریع بلاک را فراهم می سازد | FAMILY : Motor |

یکی از موارد مهم در این قسمت عبارت KNOW_HOW_PROTECT است . این عبارت که بعد از version نوشته می شود بعد از کامپایل شدن بلاک را بصورت حفاظت شده (Protect) در می آورد در اینحالت اگر بلاک را توسط برنامه LAD/STL/FBD باز کنیم با پیغام The Block is Protected مانند شکل زیر مواجه میشویم. بعبارت دیگر بلاک اگرچه کار خود را در PLC انجام میدهد ولی کاربر نمی تواند دستورات داخل آن را ببیند و اگر آیکون آنرا در پوشه بلاک Simatic Manager ببینیم مشاهده خواهیم کرد که دارای قفل است. بدیهی است در این حالت طراح پس از پایان برنامه نویسی Source را از داخل پروژه برداشته تا امکان دسترسی به متن برنامه که در واقع دانش فنی اوست وجود نداشته باشد. Protect کردن بلاک فقط توسط این دستور در فایل Source امکان پذیر است. البته علاوه بر SCL Source میتوان برنامه را بصورت STL Source نیز نوشت و با استفاده از همین دستور بلاک را Protect نمود. این روش در ضمیمه ۹ آمده است.



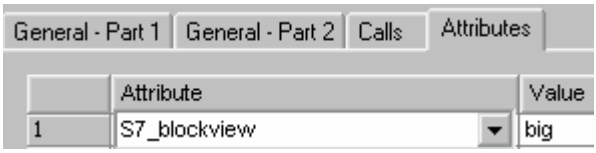

پس از نوشتن ویژگی های دلخواه برای بلاک و کامپایل کردن برنامه می توان با راست کلیک روی بلاک در پوشه Blocks از پنجره Simatic Manager و انتخاب Object Properties آنها در قسمتهای مختلف پنجره ای که ظاهر میشود مشاهده نمود.مثال زیر برنامه SCL را برای OB1 همراه با نتیجه کامپایل نشان میدهد.

```
ORGANIZATION_BLOCK OB1

TITLE = 'Main Process Control'
//This Block used for Control of All motors Located at furnace inlet and Outlet//
Know_how_protect
VERSION: '3.1'
AUTHOR: myname
NAME: Furnace
FAMILY: Motor
BEGIN;
END_ORGANIZATION_BLOCK
```




علاوه بر ویژگی های فوق میتوان System Attribute بلاک را نیز تنظیم نمود. این پارامتر ها را بایستی داخل

{ } نوشت بعنوان مثال اگر عبارت {S7_blockview := 'big'} را در مثال قبلی قبل از Title بنویسیم و کامپایل کنیم سپس با کلیک راست روی بلاک در پوشه Block ویژگیهای بخش Attribute را ببینیم شکلی مانند زیر خواهیم داشت. این ویژگی باعث میشود تا بلاک در محیط CFC با فرمت بزرگ نمایش داده شود.




## تعریف متغیرهای محلی بلاک

همانطور که می دانیم متغیرهای محلی هر بلاک در بخش Declaration بالای بلاک در محیط LAD/STL/FBD تعریف می شوند که نوع آنها بستگی به نوع بلاک دارد بعنوان مثال OB فقط متغیر Temp دارد و نمیتواند Input یا Output داشته باشد. در محیط SCL تمام انواع این متغیرها قابل تعریف است و محل تعریف آنها بلافاصله بعد از بخش ویژگی های بلاک میباشد. جدول زیر نحوه تعریف انواع بلاک ها توسط SCL را نشان میدهد.توجه شود برای تعریف این پارامترها میتوان بجای نوشتن دستورات زیر از منوی Insert>Block Templete و قسمت Parameter یا Constant استفاده کرد .

| Data | Syntax | FB | FC | OB |
|---|---|---|---|---|
| Constants | CONST<br>declaration list<br>END_CONST | X | X | X |
| Labels | LABEL<br>declaration list<br>END_LABEL | X | X | X |
| Temporary Variables | VAR_TEMP<br>declaration list<br>END_VAR | X | X | X |
| Static variables | VAR<br>declaration list<br>END_VAR | X | | |
| Input parameters | VAR_INPUT<br>declaration list<br>END_VAR | X | X | |
| Output parameters | VAR_OUTPUT<br>declaration list<br>END_VAR | X | X | |
| In/out parameters | VAR_IN_OUT<br>declaration list<br>END_VAR | X | X | |

توجه شود که اگر متغیری تعریف نشده باشد و در بخش برنامه نویسی از آن استفاده شود با خطای کامپایل Non-Existense Identifier مواجه خواهیم شد.

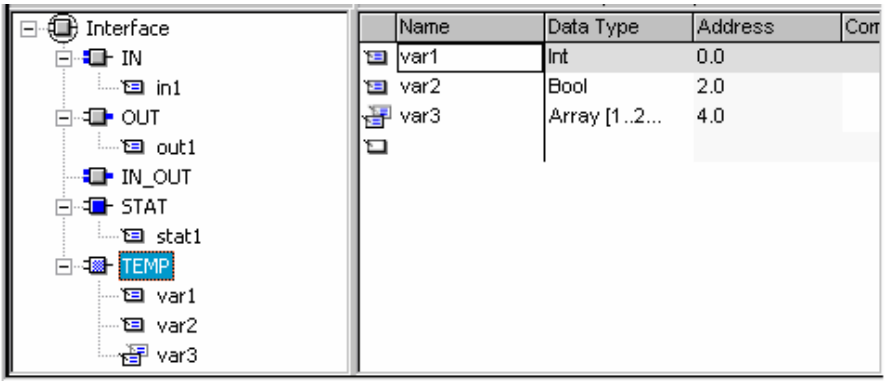
در هنگام تعریف پارامتر بایستی نوع آن نیز مشخص گردد مطابق فرمت زیر:

| نام متغیر | : | نوع متغیر | ; |
|---|---|---|---|

مثال زیر تعریف چند نوع متغیر را یک FB نشان میدهد.

```
FUNCTION_BLOCK FB1

VAR_TEMP
 var1: INT;
 var2: BOOL;
var3: ARRAY[1..200] OF real;
END_VAR

VAR
stat1: DWORD;
END_VAR

VAR_INPUT
in1: TIME;
END_VAR

VAR_OUTPUT
out1: DATE;
END_VAR

;
END_FUNCTION_BLOCK
```

پس از کامپایل کردن برنامه فوق اگر FB1 را توسط LAD/STL/FBD باز کنیم در بالای آن متغیرهای فوق را مانند شکل زیر خواهیم دید.

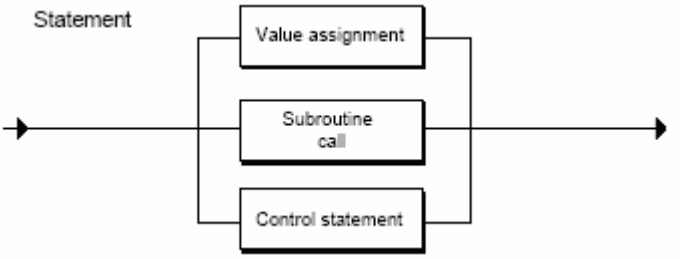



میتوان برای پارامترها Attribute نیز تعریف کرد مثال زیر نحوه تعریف Attribute برای پارامتر In2 را نشان میدهد که میتوان از آن برای برنامه CFC یا تبادل پیام برای Server استفاده کرد.

```
VAR_INPUT
  in2 {S7_server:='alarm_archiv;'
  S7_a_type:='ar_send'}: DWORD;
END_VAR
```

## بخش برنامه

پس از بخش تعریف متغیرها در زیر آن بخش مربوط به برنامه نویسی است که اگر از Tempelete استفاده شده باشد در این بخش کلمه Instructions // را مشاهده میکنیم. این بخش معمولاً و نه الزاماً با دستور Begin شروع شده و به دنبال آن برنامه نوشته میشود. برنامه همانطور که در شکل زیر نمایش داده شده میتواند حاوی سه نوع دستورات زیر باشد:




**Value Assignment** : میتوان در بخش برنامه نویسی مقادیری را به متغیرها یا آدرسهای حافظه CPU اختصاص داد به مثال های زیر توجه شود:

```
Var1:=1240;
In1:= t#2s;
I0.0:=true;
QW0:= w#16#FF00;
```

توجه شود که میتوان در بخش تعریف متغیرها در هنگام تعریف متغیر به آن مقدار اولیه اختصاص داد . مثال زیر

```
Var_temp
Data1: Real : =13.6 ;
Data2 : ARRAY[1..12] OF REAL := 0.0, 10(100.0), 1.0;
End_Var
```

**Subrutine Call** : میتوان FB یا FC یا فانکشنهای سیستم را در برنامه صدا زد همانطور که قبلاً ذکر شد این کار از طریق منوی Insert>Block Call نیز امکان پذیر است. به مثال های زیر توجه شود تفاوت صدا زدن FB و FC بعداً مورد بحث قرار خواهد گرفت. در سطر سوم SFC46 با نام سمبلیک STP صدا زده شده است.

```
FB1.DB11 (TRANSFER:= 10) ;
V2 := FC11 (X1:= FX1, X2 := FX2, Y1 := FY1, Y2 := FY2);
STP();
```

**Control Statement** : میتوان دستورات کنترلی مانند IF یا حلقه For و امثال آنها را در برنامه بکار برد این کار از طریق منوی Insert>Control Structure نیز امکان پذیر است. مثال زیر کاربردی ازدستور IF را نشان می دهد:

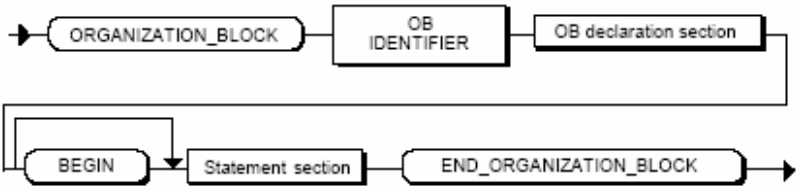
```
IF Q0.0 =false THEN
  Stp( ) ;
End_IF;
```

## ۱۳-۵ نحوه تعریف بلاک ها

ساختار بلاک ها با یکدیگر متفاوت است که بترتیب به آنها اشاره میگردد:

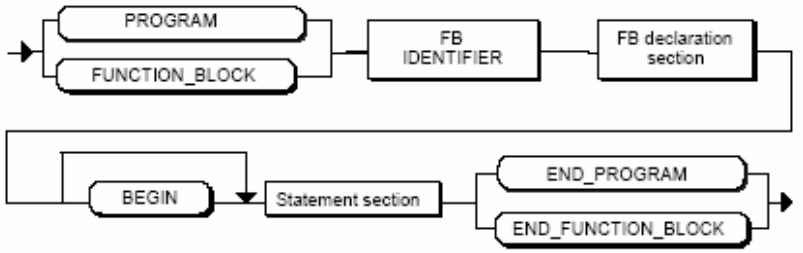
### نحوه تعریف OB

با توضیحاتی که داده شد ساختار این بلاک بر خواننده مشخص گردید . شکل زیر ساختار بلاک OB را نشان میدهد. همانطور که میبینیم استفاده از دستور Begin اختیاری است.



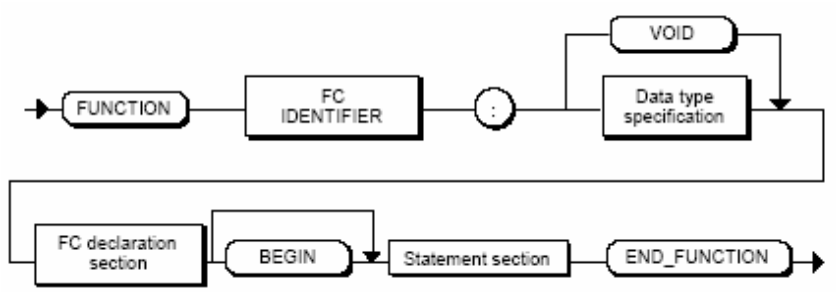

### نحوه تعریف FB

ساختار FB نیز عمدتاً مشابه OB است مطابق شکل زیر:




### نحوه تعریف FC

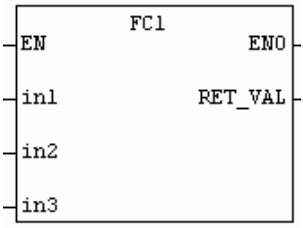
ساختار FC بجز یک مورد در سایر موارد مشابه بلاک های بالاست. همانطور که در شکل زیر می بینیم بعد از نام FC لازم است علامت : قرار گیرد و بدنبال آن یا کلمه Void یا Data Type نوشته شود . تفاوت این دو در صفحه بعد تشریح شده است.



**استفاده از Data Type در FC** : وقتی که در جلوی نام فانکشن نوع دیتا مشخص گردد یعنی فانکشن دارای یک مقدار برگشتی مثلاً از نوع Real است و در موقع صدا زدن آن در بلاک دیگر میتوان از این مقدار برگشتی استفاده کرد. در مثال زیر فانکشن با مقدار برگشتی از نوع Integer تعریف شده سپس در OB1 فراخوان گردیده است توجه شود که در هنگام فراخوانی مقدار خروجی فانکشن به متغیر var1 ریخته شده است.

```
FUNCTION   FC1  :Real

VAR_INPUT
in1,in2,in3: REAL;
END_VAR

 FC1 := in1+in2*(in3**4)
END_FUNCTION
```

```
ORGANIZATION_BLOCK   OB1

VAR_TEMP
var1:  REAL;
END_VAR
var1:= FC1(in1:=2 , in2:=7, in3:=3);
 END_ORGANIZATION_BLOCK
```

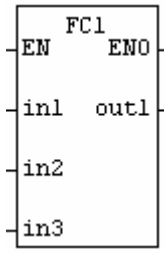



معادلLAD بلاک FC1 وقتی از Data Type استفاده شود

**استفاده از VOID در FC** : وقتی که در جلوی نام فانکشن VOID نوشته شود یعنی خود فانکشن دارای مقدار برگشتی نیست . بعبارت دیگر Ret_Val مانند حالت قبل ندارد. مثال فوق اگر بخواهیم با Void بنویسیم بصورت زیر خواهد بود توجه شود که اولاً یک خروجی برای نتیجه محاسبات در FC بنام Out1 تعریف شده ثانیاً در دستور فراخوانی FC از داخل OB دیگر نمیتوان خود FC را به متغیری اختصاص داد و متغیر Var1 نتیجه محاسبه را از Out1 میگیرد.

```
FUNCTION FC1: void

VAR_INPUT
in1,in2,in3: REAL;
END_VAR

VAR_OUTPUT
  out1:REAL;
END_VAR
 out1 := in1+in2*(in3**4);

END_FUNCTION
```

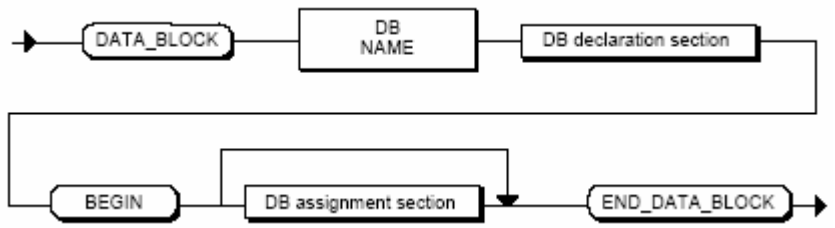
```
ORGANIZATION_BLOCK OB1

VAR_TEMP
var1:REAL;
END_VAR

FC1(in1:=2 , in2:=7, in3:=3 , out1:=var1);
;
END_ORGANIZATION_BLOCK
```




معادلLAD بلاک FC1 وقتی از Void استفاده شود

## نحوه تعریف DB

هر دو نوع DB چه نوع اشتراکی و چه نوع Instance را میتوان توسط SCL ایجاد نمود نحوه ایجاد DB اشتراکی مطابق شکل زیر است در DB های Instance لازم است بعد از نام DB با یک فاصله نام FB که قبلاً ایجاد شده را نیز ذکر کنیم ولی کلاً این DB ها نیاز به تعریف ندارند و کافی است در هنگام صدا زدن FB نام DB را نیز ذکر کنیم تا بصورت اتوماتیک توسط سیستم ایجاد شود.



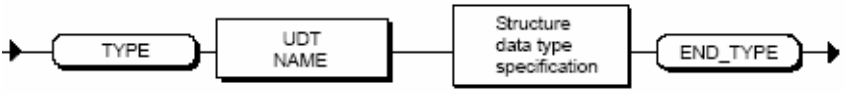

سطرهای DB در قسمت Declaration تعریف میشوند این بخش لازم است با کلمه Struct شروع و با کلمه End_Struct خاتمه یابد و در بین این دو کلمه نام و نوع پارامترها تعریف شوند در همین جا میتوان به آنها مقدار اولیه داد اگرچه برای دادن مقدار اولیه می توان از قسمت Assignment نیز بخصوص برای پارامترهای نوع Array استفاده کرد. مثال زیربرنامه SCL را همراه با شکل بلاک در محیط LAD/STL/FBD نشان می دهد.

```
DATA_BLOCK DB10
STRUCT // Date declaration with initial values
VALUE : ARRAY [1..100] OF INT := 100 (1; (
SWITCH : BOOL := TRUE;
S_WORD : WORD := W#16#FFAA;
S_BYTE : BYTE := B#16#FF;
S_TIME : S5TIME := S5T#1h30m10s;
END_STRUCT
BEGIN // Assignment section
//Value assignment for specific array elements
VALUE [1] := 5;
VALUE [5] := -1;
END_DATA_BLOCK
```

| Address | Name | Type | Initial value |
|---|---|---|---|
| 0.0 | | STRUCT | |
| +0.0 | VALUE | ARRAY[1..100] | 100 (1) |
| *2.0 | | INT | |
| +200.0 | SWITCH | BOOL | TRUE |
| +202.0 | S_WORD | WORD | W#16#FFAA |
| +204.0 | S_BYTE | BYTE | B#16#FF |
| +206.0 | S_TIME | S5TIME | S5T#1H30M10S |
| =208.0 | | END_STRUCT | |

## نحوه تعریف UDT

ساختار UDT مانند شکل زیر است . با تعریف UDT میتوان از آن در DB استفاده کرد یا میتوان پارامترهای ورودی و خروجی و ... یک FB را توسط UDT تعریف کرد. به مثال های زیر توجه کنید.

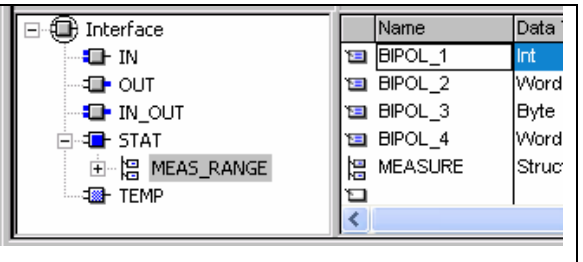



۱- ایجاد UDT برای استفاده از آن در DB در مثال زیر آمده است در اینجا اسم UDT1 بصورت سمبلیک داده شده این اسم لازم است قبلاً در جدول سمبل ها تعریف شده باشد.

```
TYPE MEASVALUES
STRUCT
BIPOL_1 : INT := 5;
BIPOL_2 : WORD := W#16#FFAA;
BIPOL_3 : BYTE := B#16#F1;
BIPOL_4 : WORD := B#(25,25);
MEASURE : STRUCT
BIPOLAR_10V : REAL;
UNIPOLAR_4_20MA : REAL;
END_STRUCT;
END_STRUCT;
END_TYPE

DATA_BLOCK DB11
UDT 1
BEGIN
END_DATA_BLOCK
```

۲-در مثال زیر از UDT مثال فوق برای تعریف پارامترهای Static یک FB استفاده شده است. اگر FB10 را باز کنیم جدول Declaration در آن مطابق شکل زیر خواهد بود.

```
TYPE MEASVALUES
STRUCT
BIPOL_1 : INT := 5;
BIPOL_2 : WORD := W#16#FFAA
BIPOL_3 : BYTE := B#16#F1;
BIPOL_4 : WORD := B#(25,25);
MEASURE : STRUCT
BIPOLAR_10V : REAL;
UNIPOLAR_4_20MA : REAL;
END_STRUCT;
END_STRUCT;
END_TYPE
```




```
FUNCTION_BLOCK FB10
VAR
MEAS_RANGE : MEASVALUES;
END_VAR
BEGIN
MEAS_RANGE.BIPOL_1 := -4;
MEAS_RANGE.MEASURE.UNIPOLAR_4_20MA := 2.7;
END_FUNCTION_BLOCK
```

## ۱۳-٦ نحوه تعریف ثوابت و Label

### ثوابت

انواع مختلف ثوابت اعم از Bit و Byte و Word و DWord و اعداد صحیح و اعشاری و کاراکتر و String و زمان و تاریخ و .... را میتوان در برنامه SCL بکار برد. جدول زیر انواع ثوابت را همراه با فرمت و مثالی از آنها نشان میدهد.

| Data Type | Description | Example in SCL |
|---|---|---|
| BOOL | Bit 1 | FALSE , TRUE ,BOOL#0 ,BOOL#1 BOOL#FALSE ,BOOL#TRUE |
| BYTE | 8-bit hexadecimal number | B#16#00, B#16#FF ,BYTE#0 ,B#2#101 Byte#'ä' ,b#16#f |
| CHAR | 8-bit (1 ASCII character) | 'A' CHAR#49 |
| STRING | Maximum of 254 ASCII characters | 'Address' |
| WORD | 16-bit hexadecimal number<br>16-bit octal number<br>16-bit binary number | W#16#0000 ,W#16#FFFF ,word#16#f<br>WORD#8#177777 ,8#177777<br>W#2#1001_0100 ,WORD#32768 |
| DWORD | 32-bit hexadecimal number<br>32-bit octal number<br>32-bit binary number | DW#16#0000_0000 ,DW#16#FFFF_FFFF<br>Dword#8#37777777777 ,8#37777777777<br>DW#2#1111_0000_1111_0000, dword#32768 |
| INT | 16-bit fixed-point number | -32768 +32767 ,INT#16#3f_ff ,<br>int#-32768 ,<br>Int#2#1111_0000, inT#8#77777 |
| DINT | 32-bit<br>fixed-point number | -2147483648 , +2147483647 ,DINT#16#3fff_ffff ,dint#-1000_0000 ,Dint#2#1111_0000 ,dinT#8#17777777777 |
| REAL | 32-bit floating-point number | Decimal format : 123.4567<br>REAL#1 , real#1.5 ,<br>Exponential format :real#2e4 , +1.234567E+02 |
| S5TIME | 16-bit<br>time value in SIMATIC format | T#0ms , TIME#2h46m30s ,<br>T#0.0s , TIME#24.855134d |
| TIME | 32-bit<br>time value in IEC format | T#-24d20h31m23s647ms ,<br>TIME#24d20h31m23s647ms , T#0.0s ,<br>TIME#24.855134d |
| Date | 16-bit date value | D#1990-01-01 , DATE#2168-12-31 |
| TIME_OF_DAY | 32-bit time of day | TOD#00:00:00 ,TIME_OF_DAY#23:59:59.999 |
| DATE_AND_ TIME | Date and time value | DT#95-01-01-12:12:12.2 , |

مثال زیر کاربرد ثوابت در برنامه SCL را نشان می دهد.

```
Value_2:=2#0101;
Value_3:=8#17;
Value_4:=16#F;
Value_5:=INT#16#3f_ff
NUM4:= -3.4 ;
NUM5:= 4e2 ;
NUM6:= real#1.5;
```

**تذکر** : در استفاده از کاراکتر میتوان بجای کاراکتر کد اسکی آنرا همراه با علامت $ بکاربرد مثال:

```
CHARACTER := '$41' ; //Corresponds to the character 'A'
Blank :='$20'; //Corresponds to the character  '_'
```

همین موضوع برای String نیز صادق است یعنی قبل از هرکد اسکی یک علامت $ قرار میگیرد مثال:

```
MESSAGE1:= '$41$4E' (*character string AN*);
```

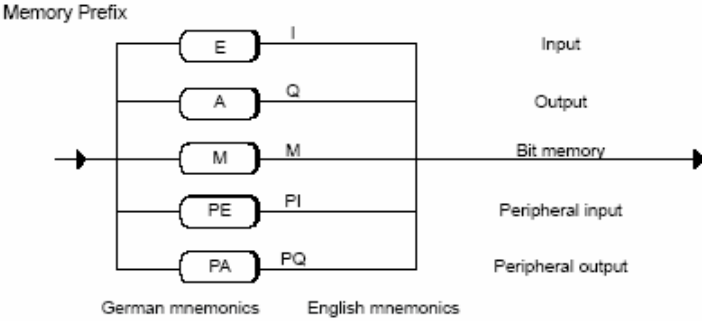
### نحوه تعریف Label

Label یا برچسب برای دستور GOTO در برنامه SCL کاربرد دارد. توسط این دستور می توان از یک سطر به سطر دیگری که با Label مشخص شده پرش نمود. Label ها نیز در ابتدای برنامه در بخش تعریف متغیرها لازمست تعریف شوند . مثال زیر نحوه تعریف Label را نشان می دهد:

```
Label
   Ali , test, err
End_Label
```

## ۱۳-۷ نحوه بکار بردن متغیرهای حافظه CPU

تمام متغیرهای حافظه CPU اعم از ورودی و خروجی و Memory Bit و تایمر و کانتر و دیتا بلاک را میتوان در برنامه SCL استفاده کرد شکل زیر برخی از این موارد را نشان میدهد:




**مثال :**

```
STATUSBYTE :=IB10;
STATUS_3 :=I1.1;
MEASVAL :=IW20;
```

**تذکر ۱:** میتوان بجای آدرس های مطلق از آدرس های سمبلیک نیز استفاده کرد بشرط اینکه سمبل مربوطه قبلاً تعریف شده باشد. مثال زیر این موضوع را نشان می دهد:

**جدول سمبل ها :**

| Symbol | Absolute Address | Data Type | Comments |
|---|---|---|---|
| Motor_contact_1 | I 1.7 | BOOL | Contact switch 1 for Motor A |
| Input1 | IW 10 | INT | Status word |
| New Data | DB1 | DB1 | |

**برنامه :**

```
MEASVAL_1 := Motor_contact_1;
Status_Motor1 := Input1 ;
STATUS_3 :="New data".D1.1;
```

**تذکر ۲ :** امکان دیگری که برای آدرس دهی متغیرهای حافظه وجود دارد و برای برنامه های کاربردی بسیار مفید است آدرس دهی Index می باشد. میتوان بجای استفاده از آدرس آنرا بصورت متغیر داخل کروشه نوشت مثلاً IW[i] که در آن i متغیری از جنس عدد صحیح است . بدیهی است متغیر i لازم است در بالای برنامه تعریف شده باشد. با این شرایط در برنامه به سهولت می توان به آن مقدار داد و آدرس دلخواه را مشخص کرد مثال :

```
MEASVAL_1 :=IW[COUNTER];
OUTLABEL :=I[i,k];
STATUS_1:= DB11.DW[COUNTER];
STATUS_2:= DB12.DX[WNO, BITNO];
STATUS_3:= Database1.DW[COUNTER];
STATUS_4:= Database2.DX[WNO, BITNO];
```

### ۸-۱۳ Operation و Experession در SCL

Operation نوع عملیاتی که بایستی انجام شود را مشخص میکند و به سه دسته تقسیم می گردد :

- عملیات ریاضی که با علاماتی چون -, + , * , / مشخص میگردد.
- عملیات مقایسه که با علاماتی چون =, <, > مشخص می گردد.
- عملیات منطقی که با علاماتی چون And , OR , NOT مشخص می شود.

جزئیات Operation های مختلف در جدول زیر آمده است. در این جدول ستون Precedence ترتیب اجرا وقتی ترکیبی از علامتها بکار رود را مشخص میکند مثلاً در عبارت c+(a*b)**3 ابتدا مقدار داخل پرانتز محاسبه شده سپس توان و بعد از آن عمل جمع انجام می گیرد. بدیهی است اگر عملیات دارای اولویت یکسان باشند ترتیب اجرا از چپ به راست خواهد بود.

| Class | Operation | Symbol | Precedence |
|---|---|---|---|
| **Assignment Operation:** | Assignment | : = | 11 |
| **Arithmetic Operations:** | Power | ** | 2 |
| | Multiplication | * | 4 |
| | Division | / | 4 |
| | Modulo function | MOD | 4 |
| | Integer division | DIV | 4 |
| | Addition | + | 5 |
| | Subtraction | - | 5 |
| **Comparison Operations:** | Less than<br>Greater than<br>Less than or equal to | <<br>><br><= | 6<br>6<br>6 |
| | Greater than or equal to | >= | 6 |
| | Equal to | = | 7 |
| | Not equal to | <> | 7 |
| **Logical Operations:** | Negation | NOT | 3 |
| | And<br>Exclusive or | AND or &<br>XOR | 8<br>9 |
| | Or | OR | 10 |
| **Parentheses :** | Parentheses | ( ) | 1 |

Expression عباراتی هستند که از ترکیب Operation ها و Address ها بوجود می آیند و برای سه منظور زیر استفاده می شوند:

- اختصاص به یک متغیر
- استفاده بعنوان شرط در دستورات کنترلی
- استفاده بعنوان پارامتر برای صدا زدن FC یا FB

چند Expression بعنوان مثال در زیر آورده شده است:

```
IB10              // address
A1 AND (A2)   // logical expression
(A3) < (A4)      // comparison expression
3+3*4/2          //arithmetic expression
A * B + D / C - 3 * VALUE1;
```

نکاتی که لازمست به آنها توجه شود:

- ترتیب اجرا بر حسب اولویت عملیات است که در صفحه قبل ذکر شد.
- ترکیب چند عمل بایستی با پرانتز جدا شود . بعنوان مثال A*-B غلط بوده وصحیح آن A*(-B) است
- تعدا پرانتز های باز و تعداد پرانتزهای بسته شده باید برابر باشند
- عملیات ریاضی نمی توانند با کاراکترها یا عملیات منطقی ترکیب شوند پس موارد ی مانند 'A' + 'B' یا (n <= 0) + (m > 0) نادرست هستند.

## عملیات ریاضی

در Expression هایی که مربوط به عملیات ریاضی هستند بایستی به نوع آدرس ها و نوع نتیجه عملیات توجه داشت طبق جدول زیر که در آن منظور از ANY_INT مقادیر INT و DINT و منظور از ANY_NUM مقادیر INT و DINT و REAL است. توجه شود که اگر یک آدرس از نوع INT و آدرس دیگر از نوع REAL باشد نتیجه بصورت REAL خواهد بود.

| Operation | Identifier | 1st Address | 2nd Address | Result |
|---|---|---|---|---|
| Power | ** | ANY_NUM | ANY_NUM | REAL |
| Multiplication | * | ANY_NUM | ANY_NUM | ANY_NUM |
| | | TIME | ANY_INT | TIME |
| Division | / | ANY_NUM | ANY_NUM | ANY_NUM |
| | | TIME | ANY_INT | TIME |
| Integer division | DIV | ANY_INT | ANY_INT | ANY_INT |
| | | TIME | ANY_INT | TIME |
| Modulo division | MOD | ANY_INT | ANY_INT | ANY_INT |
| Addition | + | ANY_NUM | ANY_NUM | ANY_NUM |
| | | TIME | TIME | TIME |
| | | TOD | TIME | TOD |
| | | DT | TIME | DT |
| Subtraction | - | ANY_NUM | ANY_NUM | ANY_NUM |
| | | TIME | TIME | TIME |
| | | TOD | TIME | TOD |
| | | DATE | DATE | TIME |
| | | TOD | TOD | TIME |
| | | DT | TIME | DT |
| | | DT | DT | TIME |

## عملیات منطقی

Expression هایی که مربوط به عملیات منطقی هستند میتوانند روی مقادیر BOOL,BYTE,WORD و DWORD انجام شوند این موارد در جدول زیر بصورت ANY_BIT نمایش داده شده اند:

| Operation | Identifier | 1st Address | 2nd Address | Result |
|---|---|---|---|---|
| Negation | NOT | ANY_BIT | - | ANY_BIT |
| Conjunction | AND | ANY_BIT | ANY_BIT | ANY_BIT |
| Exclusive disjunction | XOR | ANY_BIT | ANY_BIT | ANY_BIT |
| Disjunction | OR | ANY_BIT | ANY_BIT | ANY_BIT |

مثال های زیر کاربرد عملیات منطقی را نشان می دهد:

```
IF NOT (COUNTER > 5) THEN . . . ;
A := NOT (COUNTER1 = 4) AND (COUNTER2 = 10) ;
WHILE (A >= 9) OR (SCAN <> "n") DO.... ;
Result := IB10 AND 2#11110000 ;
```

## مقایسه

Expression هایی که مربوط به عملیات مقایسه ای هستند میتوانند روی تمام انواع متغیرهای زیر بکار روند

- INT, DINT, REAL
- BOOL, BYTE, WORD, DWORD
- CHAR, STRING
- DT, TIME, DATE, TOD

عملیات مقایسه ای را می توان با عملیات منطقی بصورت ترکیبی بکار برد بعنوان مثال

```
A<B AND A<C
 A<>(B AND C)
NOT (COUNTER1 = 4) AND (COUNTER2 = 10)
```

می توان نتیجه مقایسه را به یک متغیر از جنس Bool اختصاص داد در مثال زیر مقدار متغیر A از نتیجه مقایسه تاثیر می پذیرد و True خواهد شد.

```
A :=   3 <= 4 ;
```

تا اینجا با نحوه نوشتن عبارات و علامات و نکات اولیه برنامه نویسی SCL آشنا شدیم در قسمت بعد به شرح دستورات SCL می پردازیم.

## ۱۳- ۹ دستورات SCL

دستورات SCL به چند دسته زیر تقسیم میشوند:

- دستورات اختصاص مقادیر
- دستورات کنترلی
- صدا زدن FC و FB
- کانترها و تایمرها
- فانکشن های استاندارد SCL

### ۱۳-۹-۱ دستورات اختصاص مقادیر Value Assignment

وقتی به یک متغیر یا آدرس مقداری اختصاص داده میشود مقدار فعلی آن با مقدار جدید که در Expression جلوی آن مشخص شده عوض می شود چه این مقدار ثابت وچه متغیر باشد. فرمت کلی اختصاص مقادیر بصورت زیر است:

| | | | |
|---|---|---|---|
| Varriable | := | Expression | ; |

تمام انواع دیتاهای Elementary و Complex و نیز متغیرهای حافظه و دیتا بلاک ها را میتوان در اختصاص مقادیر بکار برد که بترتیب تشریح شده اند.

### اختصاص مقادیر از نوع Elementary Data

به انواع متغیرهای TIME , BOOL , BYTE,WORD,DWORD , REAL,DINT, INT که از نوع Elementary محسوب می شوند می توان مقدار متناظر اختصاص داد.مثال صفحه بعد این موضوع را نشان میدهد. این مثال همراه با برنامه معادل STL که پس از کامپایل بدست آمده ارائه شده است .

نکته ای که در معادل STL قابل توجه است و آنرا در تمام بلاکهای کامپایل شده SCL می بینیم دستورات ابتدا و انتهای برنامه است. همانطور که مشاهده میشود در ابتدای بلاک RLO یک شده و علاوه بر ذخیره سازی در بیت BR روی متغیر محلی L0.1 نیز ریخته شده است و در انتهای بلاک وقتی عملیات ضرب انجام شده وضعیت Overflow چک شده اگر سرریزی اتفاق بیفتد به سطر I007 پرش کرده و RLO را صفر کرده و آنرا روی متغیر L0.1 ریخته سپس این متغیر را در بیت BR ذخیره کرده است. بدین ترتیب کاربر میتواند پس از فراخوانی این بلاک از بلاک ماقبل وضعیت BR را چک کرده و خطا را بررسی نماید.

| برنامه SCL | معادل STL |
|---|---|
| **FUNCTION_BLOCK FB12**<br>**VAR**<br>SWITCH_1 : INT ;<br>SWITCH_2 : INT ;<br>SETPOINT_1 : REAL ;<br>SETPOINT_2 : REAL ;<br>QUERY_1 : BOOL ;<br>TIME_1 : S5TIME ;<br>TIME_2 : TIME ;<br>DATE_1 : DATE ;<br>TIMEOFDAY_1 : TIME_OF_DAY<br>;<br>**END_VAR**<br>BEGIN<br>SWITCH_1 := -17 ;<br>SETPOINT_1 := 100.1 ;<br>QUERY_1 := TRUE ;<br>TIME_1 := T#1H_20M_10S_30MS<br>;<br>TIME_2 :=<br>T#2D_1H_20M_10S_30MS ;<br>DATE_1 := D#1996-01-10 ;<br>SETPOINT_1 := SETPOINT_2 ;<br>SWITCH_2 := SWITCH_1 ;<br>SWITCH_2 := SWITCH_1 * 3 ;<br>**END_FUNCTION_BLOCK** | SET<br>SAVE<br>= L 0.1<br>L -17<br>T #SWITCH_1<br>L 1.001000e+002<br>T #SETPOINT_1<br># = QUERY_1<br>L S5T#1H20M10S<br>T #TIME_1<br>L T#2D1H20M10S30MS<br>T #TIME_2<br>L D#1996-1-10<br>T #DATE_1<br>L #SETPOINT_2<br>T #SETPOINT_1<br>L #SWITCH_1<br>T #SWITCH_2<br>L #SWITCH_1<br>L 3<br>* I<br>JO I007<br>JU I008<br>I007: CLR<br>= L 0.1<br>I008: T #SWITCH_2<br>CLR<br>A L 0.1<br>SAVE<br>BE |

## اختصاص مقادیر از نوع Struct و UDT

میتوان کل یک Struct را در یک Struct دیگر با ساختار مشابه ریخت مثال:

```
structname_1 := structname_2 ;
```

همچنین می توان مقادیر یا متغیرهایی را به اجزای یک Struct نسبت داد مانند:

```
structname_1.element1 := Value ;
structname_1.element1 := 20.0 ;
structname_1.element1 := structname_2.element1 ;
structname_1.arrname1 := structname_2.arrname2 ;
structname_1.arrname[10] := 100 ;
```

در مثال صفحه بعد متغیرهای Static یک FB از نوع Struct تعریف شده سپس از آنها در برنامه استفاده شده است . معادل STL این برنامه نیز ارائه شده است

| برنامه SCL | معادل STL |
|---|---|
| **FUNCTION_BLOCK FB3**<br>**VAR**<br>AUXVAR : REAL ;<br>MEASVAL : STRUCT //Target structure<br>VOLTAGE:REAL ;<br>RESISTANCE:REAL ;<br>SIMPLEARR : ARRAY [1..2, 1..2] OF INT ;<br>END_STRUCT ;<br>PROCVAL : STRUCT //Source structure<br>VOLTAGE : REAL ;<br>RESISTANCE : REAL ;<br>SIMPLEARR : ARRAY [1..2, 1..2] OF INT ;<br>END_STRUCT ;<br>**END_VAR**<br>BEGIN<br>//Assignment of a complete structure to a complete structure<br>MEASVAL := PROCVAL ;<br>//Assignment of a structure component to a structure<br>//component<br>MEASVAL.VOLTAGE := PROCVAL.VOLTAGE ;<br>//Assignment of a structure component to a variable of the<br>//same type<br>AUXVAR := PROCVAL.RESISTANCE ;<br>//Assignment of a constant to a structure component<br>MEASVAL.RESISTANCE := 4.5;<br>//Assignment of a constant to a single array element<br>MEASVAL.SIMPLEARR[1,2] := 4;<br>**END_FUNCTION_BLOCK** | SET<br>SAVE<br>= L 0.1<br>L #PROCVAL.VOLTAGE<br>T #MEASVAL.VOLTAGE<br>L #PROCVAL.RESISTANCE<br>T #MEASVAL.RESISTANCE<br>L DID [AR2,P#28.0]<br>T DID [AR2,P#12.0]<br>L DID [AR2,P#32.0]<br>T DID [AR2,P#16.0]<br>L #PROCVAL.VOLTAGE<br>T #MEASVAL.VOLTAGE<br>L #PROCVAL.RESISTANCE<br>T #AUXVAR<br>L 4.500000e+000<br>T #MEASVAL.RESISTANCE<br>L 4<br>T #MEASVAL.SIMPLEARR[1, 2]<br>SAVE<br>BE |

## اختصاص مقادیر از نوع Array

آرایه ها می توانند ماکزیمم شش بعدی و همه اجزای آنها از یک نوع باشند. میتوان کل یک آرایه را در یک آرایه همنوع ریخت مانند :

```
arrname_1 := arrname_2 ;
```

و نیز میتوان به اجزای آن مقدار اختصاص داد مانند :

```
arrname_1[ i ] := arrname_2[ j ] ;
arrname_1[ i ] := expression ;
identifier_1 := arrname_1[ i ] ;
```

مثال صفحه بعد با معادل STL نحوه استفاده از آرایه را نشان میدهد.

| برنامه SCL | معادل STL |
|---|---|
| **FUNCTION_BLOCK FB3**<br>**VAR**<br>SETPOINTS :ARRAY [0..127] OF INT;<br>PROCVALS :ARRAY [0..127] OF INT;<br>//Declaration of a matrix (=two-dimensional array)<br>//with 3 rows and 4 columns<br>CRTLLR : ARRAY [1..3, 1..4] OF INT;<br>//Declaration of a vector (=one-dimensional array) with 4<br>//components<br>CRTLLR_1 : ARRAY [1..4] OF INT;<br>**END_VAR**<br>BEGIN<br>//Assignment of a complete array to an array<br>SETPOINTS := PROCVALS;<br>//Assignment of a vector to the second row of the CRTLLR<br>//array<br>CRTLLR[2] := CRTLLR_1;<br>//Assignment of a component of an array to a component of the<br>//CTRLLR array<br>CRTLLR [1,4] := CRTLLR_1 [4];<br>**END_FUNCTION_BLOCK** | SET<br>SAVE<br>= L 0.1<br>L DW#16#10050080<br>T LD 2<br>L DINO<br>T LW 6<br>TAR2<br>+ L#2048<br>T LD 8<br>L DW#16#10050080<br>T LD 12<br>L DINO<br>T LW 16<br>TAR2<br>+ L#0<br>T LD 18<br>TAR2 LD 22<br>UC SFC 20<br>P#L 2.0<br>P#L 26.0<br>P#L 12.0<br>LAR2 LD 22<br>L DID [AR2,P#536.0]<br>T DID [AR2,P#520.0]<br>L DID [AR2,P#540.0]<br>T DID [AR2,P#524.0]<br>L #CRTLLR_1[4]<br>T #CRTLLR[1, 4]<br>A L 0.1<br>SAVE<br>BE |

## اختصاص مقادیر از نوع String

String میتوانند ماکزیمم حاوی ۲۵۴ کاراکتر باشد. میتوان یک متغیر String را به یک متغیر String دیگر اختصاص داد مانند :

```
stringvariable_1 := stringconstant;
stringvariable_1 := stringvariable_2 ;
```

و نیز میتوان کاراکترهایی را به یک String اختصاص داد.

```
DISPLAY_1 := 'Error in module 1' ;
```

مثال صفحه بعد کاربرد String را نشان میدهد.

```
FUNCTION_BLOCK FB4
VAR
DISPLAY_1 : STRING[50] ;
STRUCTURE1 : STRUCT
DISPLAY_2 : STRING[100] ;
DISPLAY_3 : STRING[50] ;
END_STRUCT ;
END_VAR
BEGIN
// Assignment of a constant to a STRING variable
DISPLAY_1 := 'Error in module 1' ;
// Assignment of a structure component to a STRING variable.
DISPLAY_1 := STRUCTURE1.DISPLAY_3 ;
// Assignment of a STRING variable to a STRING variable
If DISPLAY_1 <> STRUCTURE1.DISPLAY_3 THEN
DISPLAY_1 := STRUCTURE1.DISPLAY_3 ;
END_IF;
END_FUNCTION_BLOCK
```

### اختصاص مقادیر از نوع Date_And_Time

این متغیرها ۶۴ بیتی هستند و میتوان به آنها متغیرهای هم نوع یا مقداری از نوع DT اختصاص داد. مثال:

```
FUNCTION_BLOCK FB3
VAR
TIME_1 : DATE_AND_TIME ;
STRUCTURE1 : STRUCT
TIME_2 : DATE_AND_TIME ;
TIME_3 : DATE_AND_TIME ;
END_STRUCT ;
END_VAR
BEGIN
// Assignment of a constant to a DATE_AND_TIME variable
TIME_1 := DATE_AND_TIME#1995-01-01-12:12:12.2 ;
STRUCTURE1.TIME_3 := DT#1995-02-02-11:11:11 ;
// Assignment of a structure component to a DATE_AND_TIME  variable.
TIME_1 := STRUCTURE1.TIME_2 ;
// Assignment of a DATE_AND_TIME variable to DATE_AND_TIME variable
If TIME_1 < STRUCTURE1.TIME_3 THEN
TIME_1 := STRUCTURE1.TIME_3 ;
END_IF ;
END_FUNCTION_BLOCK
```

### اختصاص مقادیر از نوع متغیرهای حافظه CPU

همانطور که قبلاً اشاره شد به تمام متغیرهای حافظه شامل PI ,PQ, M, I,Q میتوان مقدار اختصاص داد این کار به سه روش امکان پذیر است:

- با آدر سهای مطلق

- با آدرس های ایندکس
- با آدرس های سمبلیک

مثال زیر استفاده از آدرس دهی مطلق را با معادل STL نشان می دهد:

```
FUNCTION_BLOCK FB6
Q0.0:= I0.0 AND (I0.1 OR I0.2) ;
END_FUNCTION_BLOCK
```

مثال زیر کاربرد آدرس دهی بصورت ایندکس را نشان میدهد و توسط آن بیتهای صفر از خروجی با آدرسهای بایت 0 تا 7 روشن میشود.

```
FUNCTION_BLOCK FB7
VAR_TEMP
  I:INT;
END_VAR
FOR I:= 0 TO 7 DO
  Q[I,0]:= TRUE;
  END_FOR;
END_FUNCTION_BLOCK
```

## اختصاص مقادیر از نوع دیتا بلاک

در اینجا نیز به همان سه روش که برای متغیرهای حافظه ذکر شد میتوان مقادیر را اختصاص داد مثال زیر :

```
FUNCTION_BLOCK FB3
VAR
CRTLLR_1 : ARRAY [1..4] OF INT ;
STATUSWORD1 : WORD ;
STATUSWORD2 : ARRAY [0..10] OF WORD ;
STATUSWORD3 : INT ;
STATUSWORD4 : WORD ;
ADDRESS : INT ;
END_VAR
VAR_INPUT
ADDRESSWORD : WORD ;
END_VAR
BEGIN
// Assignment of word 1 from DB11 to a variable (simple access)
STATUSWORD1 := DB11.DW1 ;
// The array component in the 1st row and 1st column of the matrix is assigned the //value of the
"NUMBER" variable (structured access):
CRTLLR_1[1] := DB11.NUMBER ;
// Assignment of structure component "NUMBER2"
// of structure "NUMBER1" to the variable status word3
STATUSWORD3 := DB11.NUMBER1.NUMBER2 ;
// Assignment of a word with index address from DB11 to a variable (indexed access)
FOR
ADDRESS := 1 TO 10 BY 1 DO
STATUSWORD2[ADDRESS] := DB11.DW[ADDRESS] ;
// Here the input parameter ADDRESSWORD as number of the
//DB and the index ADDRESS are used to specify the word address within the DB.
STATUSWORD4 :=
WORD_TO_BLOCK_DB(ADDRESSWORD).DW[ADDRESS] ;
END_FOR ;
END_FUNCTION_BLOCK
```

## ۱۳-۹-۲ دستورات کنترلی

دستورات کنترلی به سه دسته تقسیم می شوند:

- دستورات شرطی (انتخابی) شامل دو دستور IF و CASE
- دستورات حلقه شامل FOR و WHILE و REPEAT
- دستورات پرش برنامه شامل CONTINUE و EXIT و GOTO و RETURN

شایان ذکر است دستورات شرطی و حلقه ازمنوی Insert> Control Structure نیز قابل دسترس است.

### دستور IF

فرمت کلی دستور IF بصورت زیر است نحوه عملکرد دستور فوق به این گونه است که اگر شرط ۱ برقرار بود دستور ۱ و اگر شرط ۲ برقرار بود دستور ۲ واگر هیچیک از شرط های ۱ و ۲ برقرار نبود دستور ۳ اجرا میگردد.

```
IF شرط۱ THEN
; دستور۱
ELSIF شرط۲ THEN
; دستور ۲
ELSE
; دستور ۳
END_IF ;
```

نکاتی که در استفاده از دستور IF لازمست به آنها توجه شود:

- میتوان چندین دستور را در هر قسمت زیر هم نوشت تا در صورت برقراری شرط تمام آنها اجرا گردد.
- ELSEIF را میتوان به تعداد دلخواه بکار برد و هر بار یک شرط را چک کرد.
- استفاده از ELSEIF و ELSE الزامی نیست . یک دستور IF میتواند فقط شامل IF….END_IF باشد.

**مثال ۱:** برنامه زیر در صورتی که کلید I0.0 فعال شود تمام ۱۶ خروجی که از آدرس Q0.0 شروع میشود را روشن میکند.همانطور که مشاهده میشود ELSE و ELSEIF بکار نرفته است در شرط نیز نوشتن I0.0 مشابه نوشتن I0.0:=TRUE میباشد. در معادل STL دستور IF با دستور پرش ساخته شده است.

| برنامه SCL | معادل STL |
|---|---|
| IF I0.0 THEN<br>QW0:=W#16#FFFF;<br>END_IF; | SET<br>SAVE<br>= L 20.1<br>A I 0.0<br>JCN A7d0<br>L W#16#FFFF<br>T QW 0<br>A7d0: CLR<br>A L 20.1<br>SAVE<br>BE |

اشکالی که در برنامه قبل هست اینست که با فعال شدن I0.0 خروجی ها روشن شده ولی با قطع کردن سوئیچ خروجی ها خاموش نمیشوند با اضافه کردن ELSE طبق برنامه زیر این مشکل برطرف میشود:

| برنامه SCL | معادل STL |
|---|---|
| IF I0.0 THEN<br>QW0:=W#16#FFFF;<br>ELSE<br>QW0:=W#16#0000;<br>END_IF; | SET<br>SAVE<br>= L 20.1<br>A I 0.0<br>JCN A7d0<br>L W#16#FFFF<br>T QW 0<br>JU A7d1<br>A7d0: L W#16#0<br>T QW 0<br>A7d1: CLR<br>A L 20.1<br>SAVE<br>BE |

شرط در دستور IF متنوع است جدول زیر انواع شرط های ممکن را با مثال نشان می دهد.

| Type | Example |
|---|---|
| Comparison expression | TEMP > 50<br>COUNTER <= 100<br>CHAR1 < 'S' |
| Comparison and logical expression | (ALPHA <> 12) AND NOT BETA |
| Boolean address | I 1.1 |
| Arithmetic expression | ALPHA = (5 + BETA) |

## دستور CASE

این دستور نیز یک دستور شرطی است . در مواردی که برای یک شرط از نوع Integer تعداد انتخاب ها زیاد باشد استفاده از دستور IF طولانی بوده و به دلیل نیاز به ELSEIF های زیاد تا حدودی غیر معقول است . در چنین حالاتی استفاده از دستور CASE توصیه میشود که فرمت آن بصورت زیر است:

```
CASE متغیر یا عبارت OF
  ; دستور اول : مقدار اول
  ; دستور دوم : مقدار دوم
  .
  .
  ;دستور n ام : مقدار n ام
  ELSE
  ;دستور m ام : مقدار m ام
  END_CASE ;
```

Value List

عملکرد CASE به این صورت است که متغیر یا عبارت (Expression) را با مقادیر Value List مقایسه کرده و در صورت پیدا کردن آن دستور جلوی آن مقدار را اجرا میکند و اگر آنرا در لیست پیدا نکرد دستور بعد از ELSE را اجرا می نماید.

نکاتی که در دستور CASE لازمست به آنها توجه شود :

- متغیر یا نتیجه عبارت که بعد از کلمه CASE نوشته می شود بایستی از نوع Integer باشد.
- در Value List می توان در جلوی هر مقدار چندین سطر دستور زیر هم نوشت .
- استفاده از ELSE الزامی نیست اگر ELSE وجود نداشت و متغیر نیز با هیچیک از مقادیر برابر نبود از CASE خارج می شود.
- در Value List می توان چند مقدار را در یک سطر نوشت که با کاما از هم جدا میشوند مانند:

  3,4,7,9 : Q0.0:=true

  در این حالت اگر متغیر با هر کدام از مقادیر فوق برابر بود دستور جلوی آنها اجرا میشود.
- در Value List می توان رنجی از مقادیر را در یک سطر نوشت که با .. مشخص میشوند مانند:

  3..9 : Q0.0:=true

  در این حالت اگر متغیر با مقادیر این رنج ( در مثال فوق اعداد 3 تا 9 ) برابر بود دستور جلوی آنها اجرا میشود.
- مقادیر در Value List همه بایستی از نوع Integer باشند و فقط یکبار تکرار شوند.

**مثال ۱** : از کانال 252 کارت آنالوگ ورودی مقدار فشار خوانده میشود و چهار حد برای آن کنترل می گردد. این حدود عبارتند از : High High و High و Low و Low Low . اگر مقدار فشار به هر کدام از مقادیر فوق برسد بیتهای صفر تا ۳ در DB1 فعال میشوند. توجه شود که نمی توان PIW252 را بجای متغیر بکار برد زیرا از جنس Word است و Integer نیست از اینرو آنرا توسط یک تابع تبدیل به Integer تبدیل کرده و روی متغیر Pressure ریخته ایم .توابع تبدیل در آینده مورد بحث قرار خواهند گرفت.

```
Pressure:= Word_to_Int( PIW252 );
 CASE pressure OF
  27000:  db1.dbx0.0:=1 ;db1.dbx0.1:=0;db1.dbx0.2:=0;db1.dbx0.3:=0;//HH Limit
  20000:  db1.dbx0.0:=0 ;db1.dbx0.1:=1;db1.dbx0.2:=0;db1.dbx0.3:=0;//H  Limit
  7000 :  db1.dbx0.0:=0 ;db1.dbx0.1:=0;db1.dbx0.2:=1;db1.dbx0.3:=0;//L  Limit
    0 :   db1.dbx0.0:=0 ;db1.dbx0.1:=0;db1.dbx0.2:=0;db1.dbx0.3:=1;//LL Limit;
 ELSE:
  db1.dbb0:= b#16#00;
   END_CASE;
```

### دستور FOR

حلقه FOR برای تکرار دستور یا دستورات تا حد مشخص شده ای بکار می رود با فرمت زیر :

```
FOR شاخص حلقه := مقدار اولیه TO مقدار نهایی BY پله های حلقه DO
دستورات ;
END_FOR ;
```

مثال :

```
FOR i:=1 TO 100 BY 1 DO
دستور ;
END_FOR ;
```

نحوه عملکرد بدینگونه است که ابتدا i=1 شده و دستور اجرا می شود سپس حلقه به ابتدا بر می گردد و مقدار I یکی افزایش می یابد و مجدداً دستور اجرا می گردد. این کار آنقدر تکرار می شود تا I به مقدار نهایی یعنی 100 برسد پس از اجرا شدن دستور حلقه به ابتدا بر می گردد و مقدار I برابر 101 می شودکه چون از مقدار نهایی بزرگتر است حلقه خاتمه می یابد. نکاتی که توجه به آنها لازم است :

- شاخص حلقه می تواند از نوع INT یا DINT باشدو بایستی قبلاً تعریف شده باشد.
- ماکزیمم مقدار نهایی می تواند تا حد بزرگترین عدد INT یا عدد DINT باشد.
- مقدار اولیه و مقدار نهایی و پله های حلقه را الزماً نباید عدد ثابت داد این مقادیر میتوانند بصورت متغیر باشند.
- پله های حلقه در جلوی BY مشخص می گردد . اگر عدد یک برای پله های حلقه های مد نظر باشد میتوان کلمه BY و عدد یک را ننوشت و حلقه با مقدار پیش فرض پله 1+ اجرا میشود.
- کاربرد BY کلاً در مواردی است که پله حلقه عددی بغیر از یک باشد مثلاً اگر بخواهیم هر بار شاخص حلقه ۲ واحد افزایش یابد BY 2 بکار می بریم مانند:

```
FOR i:= 1 TO 100 BY 2 DO
دستور ;
END_FOR;
```

در این مثال بار اول i=1 و بار دوم i=3 و همینطور هر بار ۲ واحد افز ایش مییابد آخرین باری که حلقه اجرا میشود i=99 است زیرا بعد از آن i=101 شده که از مقدار نهایی بزرگتر است و حلقه خاتمه مییابد.

- با اختصاص مقدار منفی به BY می توان حلقه کاهشی ایجاد کرد بدیهی است در اینحالت مقدار اولیه بایستی از مقدار نهایی بزرگتر باشد مثال:

```
FOR i:= 100 TO 1 BY -2 DO
دستور ;
END_FOR;
```

- تمام دستورات از جمله دستورات کنترلی مانند IF و CASE را می توان در داخل حلقه بکار برد.
- میتوان دو یا چند حلقه تودرتو ایجاد کرد در مثال زیر هربار که مقدار I یکی افزایش می یابد حلقه داخلی ۲۰۰ بار تکرار می شود بنابراین دستور ۲ عملاً ۲۰۰۰۰ بار اجرا خواهد شد.

```
FOR   i:= 1  TO  100     DO
 دستور۱ ;
FOR j:= 1 to 200 DO
دستور۲ ;
END_FOR;
END_FOR;
```

- نکته مهمی که لازم است بدان توجه شود اینست که حلقه FOR در یک سیکل اسکن CPU اجرا میشود بنابراین بزرگ بودن آن با استفاده از حلقه های تودرتو منجر به افزایش سیکل اسکن خواهد شد. مثال زیر حلقه FOR را همراه با معادل STL آن نشان میدهد همانطور که مشاهده میشود تا زمانی که به مقدار نهایی نرسیده برنامه توسط پرش JU داخل لوپ است و سیکل اسکن جدید شروع نخواهد شد.

| برنامه SCL | معادل STL |
|---|---|
| ORGANIZATION_BLOCK OB1<br><br>VAR_TEMP<br>chk,I:INT;<br>END_VAR<br>FOR i:= 1 TO 30000 do<br>  chk:=chk+1;<br>END_FOR;<br><br>END_ORGANIZATION_BLOCK | SET<br>  SAVE<br>  =  L  24.1<br>  L  1<br>  T  #I<br>**A7d0**: L  #I<br>  L  30000<br>  <=I<br>  JCN  A7d1<br>  L  #chk<br>  L  1<br>  +I<br>  JO  I007<br>  JU  I008<br>I007: CLR<br>  =  L  24.1<br>I008: T  #chk<br>  L  #I<br>  L  1<br>  +I<br>  JO  I009<br>  JU  I00a<br>I009: CLR<br>  =  L  24.1<br>I00a: T  #I<br>  **JU  A7d0**<br>A7d1: CLR<br>  A  L  24.1<br>  SAVE<br>  BE |

یکی از کاربردهای مهم حلقه FOR برای آدرس دهی بصورت INDEX است مثال زیر این موضوع را بهتر نشان میدهد:

**مثال :** دو سیستم مشابه هرکدام ۸ نقطه برای اندازه گیری مقدار دما دارند این هشت نقطه برای هر سیستم بصورت مجزا به یک کارت AI متصل شده است آدرس پایه کارت اول 252 و آدرس پایه کارت دوم 400 است. برنامه PLC این نقاط را نظیر به نظیر چک کرده و اگر اختلافی وجود داشت بیت متناظر از خروجی صفر را روشن میکند بعنوان مثال برای اختلاف بین مقادیر دو کانال اول، خروجی Q0.0 و برای اختلاف بین مقادیر کانال دوم، خروجی Q0.1 روشن می گردد.

```
ORGANIZATION_BLOCK OB1

VAR_TEMP
I:INT;
END_VAR

FOR i:= 0 TO 14 BY 2  DO
IF PIW[252+I]<> PIW[400+I] THEN
   Q[0,I/2]:=TRUE;
ELSE
      Q[0,I/2]:=FALSE;
END_IF;
END_FOR;

END_ORGANIZATION_BLOCK
```

**مثال**: محاسبه تعداد سوئیچ های فعال از یک کارت DI

به یک کارت ورودی دیجیتال ۱۶ ورودی با آدرس پایه صفر تعداد ۱۶ سوئیچ متصل است میخواهیم در هر لحظه تعداد سوئیچهای بسته شده شمارش شده و در MW0 ذخیره شود. برای اینکار ابتدا کل ۱۶ ورودی را بصورت word می خوانیم و آنرا به عدد صحیح تبدیل میکنیم . با تقسیم متوالی این عدد بر ۲ تعداد یک ها را در باقیمانده تقسیم می شماریم تعداد این یک ها معادل تعداد سوئیچ های بسته شده است.متغیر C در این برنامه برای باقیمانده و متغیر S برای جمع کردن باقیمانده ها بکار رفته است .نهایتاً تعداد سوئیچ های بسته شده که بصورت مقدار Integer در متغیر S ذخیره شده را با تبدیل به Word به MW0 انتقال می دهیم.

```
ORGANIZATION_BLOCK OB1
VAR_TEMP
a,b,c,s,i:INT;
END_VAR
a:= word_to_int(iw0);
s:=0;
FOR i:= 1 TO 16   DO
b:=a/2;
c:=a-b*2;
s:=s+c;
a:=b;
END_FOR;
Mw0:=int_to_word(s);
END_ORGANIZATION_BLOCK
```

**مثال :** برنامه زیر در یک آرایه که عناصر آن از جنس String هستند بدنبال عبارت 'KEY' می گردد و در صورت یافتن آن M0.0 را یک می کند.

```
FUNCTION_BLOCK FB1
VAR
I: INT ;
IDWORD: ARRAY [1..50] OF STRING;
END_VAR
BEGIN
FOR I := 1 TO 50 BY 2 DO
IF IDWORD [I] = 'KEY' THEN
M0.0:=true;
EXIT;
END_IF;
END_FOR;
END_FUNCTION_BLOCK
```

**دستور WHILE**

فرمت این دستور بصورت زیر است و عملکرد آن به اینگونه است که تازمانی که شرط برقرار است حلقه ادامه دارد و دستور یا دستورات اجرا می شود و وقتی شرط برقرار نبود حلقه به اتمام می رسد.

```
WHILE   شرط   DO
دستور;
END_WHILE ;
```

**مثال :**

```
I := 1 ;
WHILE  I <= 50  OR  I0.0= TRUE   DO
I := I + 2;
END_WHILE ;
```

نکاتی که لازم است بدانها توجه شود:

- حلقه WHILE بایستی بصورت کنترل شده استفاده گردد از آنجا که این حلقه در یک سیکل اسکن انجام میشود اگر مدت زمانی که شرط برقرار است طولانی باشد و به حد ماکزیمم سیکل برسد CPU متوقف خواهد شد. در مثال زیر اگر سوئیچ I0.0 برای زمان طولانی روشن باشد CPU متوقف میگردد.

```
WHILE  I 0.0=TRUE DO
I := I + 2;
END_WHILE ;
```

- نتیجه شرط بصورت منطقی چک میشود یعنی TRUE یا FALSE بودن آن بررسی میشود و حلقه در صورتی ادامه می یابد که نتیجه شرط TRUE باشد.

### دستور REPEAT

فرمت این دستور بصورت زیر و عملکرد آن به اینگونه است که تازمانی که شرط برقرار نیست حلقه ادامه دارد و دستور یا دستورات اجرا می شود و وقتی شرط برقرار شد حلقه به اتمام می رسد.

```
REPEAT
 دستور ;
UNTIL  شرط
END_REPEAT;
```

**مثال :**

```
I := 1 ;
REPEAT
I := I + 2 ;
UNTIL I > 50  AND  I0.0=FALSE
END_REPEAT ;
```

این مثال شبیه مثال WHILE عمل میکند ولی شرط آن معکوس است . بطور کلی منطق مورد نظر را می توان با هر دو نوع حلقه پیاده کرد و انتخاب به عهده کاربر است. در اینجا نیز اگر حلقه کنترل شده استفاده نشود میتواند CPU را متوقف کند . حلقه های WHILE و REPEAT بر خلاف FOR ممکن است بدون پایان باشند.

### دستور CONTINUE

این دستور در داخل حلقه های FOR , WHILE , REPEAT می تواند بکار رود و به تنهایی نوشته میشود. کاربرد آن موجب میشود که برنامه دستورات بعد از CONTINUE را انجام ندهد و به ابتدای لوپ برگردد . در مثال زیر در صورت فعال شدن سوئیچ I0.0 دستور بعد از END_IF اجرا نشده و برنامه لوپ بعدی را شروع می کند.

```
FOR I:= 1 TO 32000 DO
  IF I0.0=TRUE THEN
    CONTINUE;
  END_IF;
     QW2:=INT_TO_WORD(I);
 END_FOR;
```

**دستور EXIT**

این دستور نیز در داخل حلقه های FOR , WHILE , REPEAT می تواند بکار رود و به تنهایی نوشته میشود. کاربرد آن موجب میشود که برنامه از حلقه بیرون بیاید پس برخلاف CONTINUE لوپ قطع میشود و ادامه نمی یابد. در مثال زیر در صورت فعال شدن سوئیچ I0.0 لوپ خاتمه می یابد و اولین دستور بعد از END_FOR اجرا خواهد شد..

```
FOR I:= 1 TO 32000 DO
  IF I0.0=TRUE THEN
    EXIT;
  END_IF;
     QW2:=INT_TO_WORD(I);
  END_FOR;
```

**دستور GOTO**

این دستور در هر جای برنامه می تواند بکار رود و منجر میشود که برنامه از آنجا به سطری که با Label مشخص شده پرش کند. Label لازم است جلوی GOTO بکار رود:

```
GOTO    Label
```

به نکات زیر توجه شود :

- پرش می تواند به سطرهای ماقبل یا سطرهای مابعد انجام شود.
- نام Label بایستی در ابتدای بلاک تعریف شده باشد.
- از یک بلاک نمیتوان به بلاک دیگر پرش انجام داد پس Label باید در همان بلاک باشد.
- Label در بلاک بایستی منحصر بفرد باشد نمیتوان چند سطر با Label مشابه داشت ولی میتوان با چند دستور GOTO از هر نقطه دلخواهی به Label پرش نمود.
- پرش به داخل حلقه امکان پذیر نیست ولی پرش از داخل حلقه به بیرون ممکن است.

**مثال:**

```
LABEL
LAB1, LAB2, LAB3 ;
END_LABEL
BEGIN
IF A > B THEN
GOTO LAB1 ;
ELSIF A > C THEN
GOTO LAB2 ;
END_IF ;
// . . .
LAB1: INDEX := 1 ;
GOTO LAB3 ;
LAB2: INDEX := 2 ;
// . . .
LAB3:          // …….
```

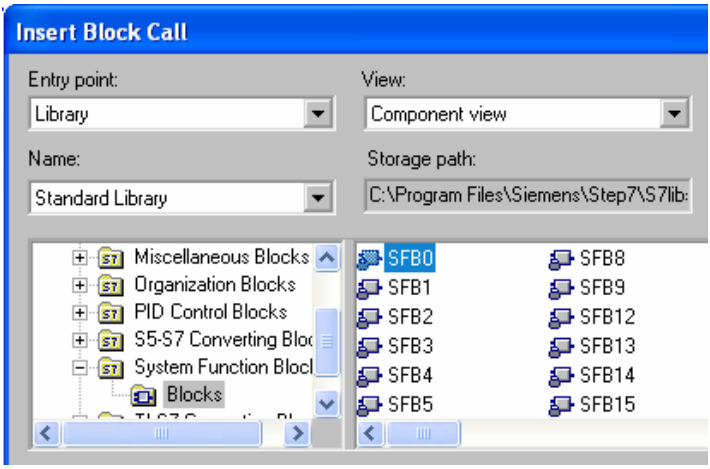
**دستور RETURN**

این دستور در هر جای برنامه بکار رود منجر به این میشود که بلاک جاری در آن نقطه قطع شده و به بلاک ماقبل برگردد. بطور پیش فرض در انتهای تمام بلاک ها RETURN وجود دارد ولی با بکار بردن آن در وسط برنامه امکان برگشت از بلاک قبل از اتمام آن نیز وجود دارد . کاربرد دستور RETURN در OB باعث بازگشت به سیستم عامل میشود . بعنوان مثال میتوان از برنامه یک وقفه قبل از اتمام اجرای آن بازگشت نمود.

## ۱۳-۹-۳ دستورات صدا زدن FB و FC

تمام FB ها و FC ها را می توان در برنامه SCL صدازد حتی اگر به زبانی غیر از SCL ایجادشده باشند مانند LAD یا FBD یا STL . همینطور تمام SFC ها و SFB ها را میتوان در SCL فراخوان کرد.صدا زدن اینها می تواند براساس اسم واقعی یا سمبل آنها باشد بشرط آنکه سمبل قبلاً تعریف شده باشد.

استفاده از منوی Insert> Block Call ساده ترین راه برای صدا زدن موارد فوق است. بنابراین اگر FB یا FC توسط SCL نوشته شده اند بصورت Partial آنها را کامپایل کرد تا امکان صدا زدن آنها در بلاک های بعدی توسط منوی Insert>Block call میسر شود. شکل زیر نحوه صدا زدن SFB0 از طریق منوی مزبور را نشان می دهد:




بدیهی است برای صدا زدن FC یا FB که در پروژه کاربر ایجاد شده بایستی در پنجره فوق بجای Library انتخاب Project را داشته باشیم.

### صدا زدن FB و SFB

همانطور که میدانیم FB و SFB لازم است همراه با دیتا بلاک صدا زده شوند در SCL کافیست ابتدا نام FB را نوشته سپس با قرار دادن یک نقطه نام دیتا بلاک را بنویسیم (مثلاً FB1.DB1 ) این DB حتی اگر وجود نداشته باشد در هنگام کامپایل توسط SCL ایجاد خواهد شد. پس از نوشتن نام FB و DB لازم است پارامترهای FB را داخل پرانتز بکار ببریم به ورودی ها مقدار و به خروجی ها آدرس اختصاص دهیم . خروجی ها را میتوان بعد از صدا زدن FB از DB مربوط به آن نیز دریافت کرد . در مثال زیر ابتدا FB ایجاد شده سپس از بلاک دیگر صدا زده شده و نکات فوق در آن بکار رفته است:

```
FUNCTION_BLOCK FB17
VAR_INPUT
FINALVAL: INT; //Input parameter
END_VAR
VAR_IN_OUT
IQ1 : REAL; //In_out parameter
END_VAR
VAR_OUTPUT
CONTROL: BOOL;//Output parameter
END_VAR
VAR
INDEX: INT;
END_VAR
CONTROL :=FALSE;
FOR INDEX := 1 TO FINALVAL DO
IQ1 :=IQ1*2;
IF IQ1 > 10000 THEN
CONTROL := TRUE;
END_IF;
END_FOR;
END_FUNCTION_BLOCK
```

در ادامه FB17 را که دارای نام سمبلیک TEST است را همراه با DB10 که دارای نام سمبلیک TEST_1 است صدا زده ایم توجه شود که خروجی را بکار نبرده ایم.

```
ORGANIZATION_BLOCK OB1
VAR_TEMP
variable1:REAL;
END_VAR
FB17.DB10 (FINALVAL:=10, IQ1:=VARIABLE1);
END_ORGANIZATION_BLOCK
```

برای استفاده از خروجی FB میتوانیم آن را از FB یا بطور مستقیم از DB بگیریم:

RESULT:= DB10.CONTROL; یا FB17.DB12 (INP_1:=DB10.CONTROL);

در ضمن میتوانستیم صدا زدنFB با DB را بصورت سمبلیک انجام دهیم مانند TEST.TEST_1

### صدا زدن FC

از نظر کلی صدا زدن FC مشابه FB است ولی اولاً نیازی به DB ندارد ثانیاً در هنگام ایجاد FC نیاز به تعریف مقدار برگشتی است که در FB این نیاز وجود نداشت. نحوه تعریف مقدار برگشتی در صفحه ۱۳۵ توضیح داده شد. در اینجا یک مثال را برای دو حالت بررسی میکنیم فانکشن مختصات دو نقطه را میگیرد و فاصله آنها را محاسبه میکند:

- حالت اول FC دارای مقدار برگشتی از نوع REAL است.
- حالت دوم FC دارای مقدار برگشتی نیست ( VOID )

| حالت اول | حالت دوم |
|---|---|
| **FUNCTION FC1 : REAL**<br>VAR_INPUT<br>X1: REAL;<br>X2: REAL;<br>Y1: REAL;<br>Y2: REAL;<br>END_VAR<br>BEGIN<br>FC1:= SQRT( (X2-X1)**2 + (Y2-Y1)**2 );<br>**END_FUNCTION** | **FUNCTION FC1 : VOID**<br>VAR_INPUT<br>X1: REAL;<br>X2: REAL;<br>Y1: REAL;<br>Y2: REAL;<br>END_VAR<br>VAR_OUTPUT<br>D: REAL;<br>BEGIN<br>D:= SQRT( (X2-X1)**2 + (Y2-Y1)**2 );<br>**END_FUNCTION** |
| **ORGANIZATION_BLOCK OB1**<br>VAR_TEMP<br>DISTANCE: REAL;<br>END_VAR<br>DISTANCE:=FC1(X1:=2.1 ,X2:=3.8 , Y1:=20.0 , Y2:=-9.2);<br><br>**END_ORGANIZATION_BLOCK** | **ORGANIZATION_BLOCK OB1**<br>VAR_TEMP<br>DISTANCE: REAL;<br>END_VAR<br>FC1(X1:=2.1 ,X2:=3.8 , Y1:=20.0 , Y2:=-9.2 , D:= DISTANCE);<br><br>**END_ORGANIZATION_BLOCK** |

نکته مهم دیگر استفاده از پارامترهای EN و ENO است این دو پارامتر هم برای FC و هم برای FB در هنگام صدا زدن میتواند بکار رود . با استفاده از EN میتوان صدا زدن FC یا FB را به شرایط خاصی نسبت داد کافیست در هنگام فراخوانی به این پارامترها مقداری اختصاص داده شود. مثال زیر صدا زدن FC1 را در بلاک دیگر با مشروط کردن EN به ورودی I0.0 نشان میدهد:

```
FC1( EN:= I0.0 , X1:=2.1 ,X2:=3.8 , Y1:=20.0 , Y2:=-9.2 , D:= DISTANCE);
```

پارامتر ENO از وضعیت فلگ OK تاثیر می پذیرد و در صورت اجرای درست FC یا FB پارامتر ENO یک خواهد شد. میتوان در بلاک ماقبل از این خروجی برای کنترل صحت اجرای فانکشن استفاده کرد مانند:

```
IF ENO THEN
// Everything OK
ELSE
// Error occurred, so error handling required
END_IF;
```

### ۱۳-۹-٤ دستورات کانترها

شمارنده ها در SCL بصورت یک فانکشن صدا زده میشوند مشابه شمارنده های LAD/STL/FBD در اینجا نیز سه شمارنده وجود دارد که هر کدام فانکشن خاصی دارند و نام آنها شبیه نام بلاکهای LAD و FBD است:

| نام فانکشن | شرح |
|---|---|
| S_CU | شمارنده افزایشی |
| S_CD | شمارنده کاهشی |
| S_CUD | شمارنده افزایشی کاهشی |

در هنگام صدا زدن فانکشن های فوق لازم است پارامترهای ورودی و خروجی آن مشخص شده و مقدار دهی شود. برای فانکشن S_CUD این پارامترها بصورت جدول زیر هستند بدیهی است برای کانتر افزایشی ورودی CD و برای کانتر کاهشی ورودی CU بکار نخواهد رفت.

| Parameter | Data Type | شرح |
|---|---|---|
| C_NO | C+INT | شماره کانتر |
| CD | BOOL | ورودی افزایش دهنده |
| CU | BOOL | ورودی کاهش دهنده |
| S | BOOL | ورودی برای ست کردن مقدار اولیه به کانتر |
| PV | WORD | مقدار اولیه برحسب BCD که میتواند بین 0 تا 999 باشد بفرمت 16#<value |
| R | BOOL | ورودی برای ریست کردن کانتر |
| Q | BOOL | خروجی نشان دهنده فعال بودن یا نبودن کانتر |
| CV | WORD | خروجی نشان دهنده مقدار لحظه ای کانتر بصورت هگز |

بدلیل تشابه این پارامترها با پارامترهای کانترهای LAD و FBD وبا توجه به توضیحاتی که در مورد آنها در جلد اول کتاب داده شده از تکرار آن مباحث در اینجا صرفنظر می نماییم .نمونه ای از نحوه فراخوانی سه نوع کانتر در زیر آمده است :

```
FUNCTION_BLOCK FB1

VAR
CurrVal, binVal: WORD;
actFlag: BOOL;
END_VAR
```

```
CurrVal :=S_CD (C_NO:=C10, CD:=TRUE, S:=TRUE, PV:=100,R:=FALSE, CV:=binVal,Q:=actFlag);
CurrVal :=S_CU (C_NO:= C11, CU:=M0.0, S:=M0.1, PV:=16#110,R:=M0.2, CV:=binVal,Q:=actFlag);
CurrVal :=S_CUD(C_NO:= C12, CD:=I0.0, CU:=I0.1, S:=I0.2&I0.3, PV:=120, R:=FALSE,
CV:=binVal,Q:=actFlag);
END_FUNCTION_BLOCK
```

در مثال زیر کانتر کاهشی با لبه پالس I0.0 کار می کند . مقدار اولیه 89 پس از فعال شدن سوئیچ I0.2 به کانتر اعمال میگردد. کانتر توسط I0.3 میتواند ری ست گردد مقدار لحظه ای کانتر بصورت هگز در متغیر BIN_VALUE ریخته شده که با تبدیل آن به عدد صحیح نتیجه در خروجی RESULT فانکشن قابل استفاده است.

```
FUNCTION_BLOCK FB1

VAR_INPUT
MYCOUNTER : COUNTER ;
END_VAR
VAR_OUTPUT
RESULT : INT ;
END_VAR

VAR
BCD_VALUE : WORD ; // Count value BCD coded
BIN_VALUE : WORD ; // Count value binary
INITIALVALUE: WORD ;
END_VAR

BEGIN
Q0.0 := 1 ;
INITIALVALUE := 16#0089 ;
BCD_VALUE := S_CD (C_NO := MYCOUNTER,
CD := I0.0 ,
S := I0.2 ,
PV := INITIALVALUE,
R := I0.3 ,
CV := BIN_VALUE ,
Q := Q0.7) ;
RESULT := WORD_TO_INT (BIN_VALUE) ;
QW4 := BCD_VALUE ;

END_FUNCTION_BLOCK
```

### ۱۳-۹-۵ دستورات تایمرها

تایمر ها نیز در SCL بصورت یک فانکشن صدا زده میشوند که نام و نوع پارامترهای آنها شبیه بلاکهای LAD و FBD است بنابراین در اینجا نیز ۵ نوع تایمر مطابق جدول زیر وجود دارد:

| Timer Function | Explanation |
|---|---|
| S_PULSE | pulse timer |
| S_PEXT | extended pulse timer |
| S_ODT | on-delay timer |
| S_ODTS | retentive on-delay timer |
| S_OFFDT | off-delay timer |

پس از صدا زدن فانکشنهای فوق بایستی پارامترهای آنها ذکر شده و به آنها مقدار یا متغیر اختصاص داده شود . جدول بعد این پارامترها را نشان میدهد که با توجه به شباهت آنها با پارامترهای تایمر در LAD و FBD که در جلد اول کتاب مورد بحث قرار گرفته اند از تشریح آنها خودداری می کنیم.

| Parameter | TYPE | شرح |
|---|---|---|
| T_NO | T+ int | شماره تایمر |
| S | BOOL | ورودی فعال کننده تایمر |
| TV | S5TIME | زمان تایمر |
| R | BOOL | ورودی ریست کننده تایمر |
| Q | BOOL | خروجی نمایش فعال بودن یا نبودن تایمر |
| BI | WORD | خروجی نمایش مقدار زمان باقیمانده بصورت هگز |

مثال زیر نحوه صدا زدن پنج نوع تایمر فوق الذکر را در برنامه SCL نشان می دهد:

```
FUNCTION_BLOCK FB2

VAR
CurrTime : S5time;
BiVal : word;
ActFlag : bool;
END_VAR

CurrTime :=S_ODT (T_NO:= T10, S:=TRUE, TV:=T#1s, R:=FALSE, BI:=biVal,Q:=actFlag);
CurrTime :=S_ODTS (T_NO:= T11, S:=M0.0, TV:= T#1s, R:=M0.1, BI:=biVal,Q:=actFlag);
CurrTime :=S_OFFDT(T_NO:= T12, S:=I0.1 & actFlag, TV:= T#1s, R:=FALSE, BI:=biVal,Q:=actFlag);
CurrTime :=S_PEXT (T_NO:= T13, S:=TRUE, TV:= T#1s, R:=I0.0, BI:=biVal,Q:=actFlag);
CurrTime :=S_PULSE(T_NO:= T14, S:=TRUE, TV:= T#1s, R:=FALSE, BI:=biVal,Q:=actFlag);

END_FUNCTION_BLOCK
```

نحوه عملکرد تمامی تایمرهای فوق در جلد اول کتاب به تفصیل بحث شده و نیازی به تکرار ندارد. مثال زیر نمونه دیگری از کاربرد تایمر در SCL را نشان می دهد که در آن پارامترها بصورت متغیر داده شده اند:

```
FUNCTION_BLOCK TIMER

VAR_INPUT
mytime : TIMER ;
END_VAR
VAR_OUTPUT
result : S5TIME ;
END_VAR
VAR
set وreset : BOOL ;
bcdvalue : S5TIME ;//Time base and time remaining in BCD
binvalue : WORD ; //Time value in binary
initialvalue : S5TIME ;
END_VAR
Q0.0 := 1;
set := I0.0 ;
reset := I0.1;
initialvalue := T#25S ;
bcdvalue := S_PEXT (T_NO := mytime , S := set ,TV := initialvalue ,R := reset ,BI := binvalue ,
Q := Q0.7) ;
result := bcdvalue ;
QW4 := binvalue ;

END_FUNCTION_BLOCK
```

در مثال زیر ۴ تایمر در یک حلقه For با زمانهای مختلف تنظیم شده اند:

```
FUNCTION_BLOCK FB3

VAR_INPUT
MY_TIMER: ARRAY [1..4] of STRUCT
T_NO: INT;
TV : WORD;
END_STRUCT;
END_VAR
.
.
FOR I:= 1 TO 4 DO
S_ODT(T_NO:=MY_TIMER[I].T_NO, S:=true,
TV:= MY_TIMER[I].TV);
END_FOR;
.
.
END_FUNCTION_BLOCK
```

### ۱۳-۹-٦ فانکشن های استاندارد SCL

با فانکشن های استاندارد برنامه LAD/STL/FBD در فصل قبل آشنا شدیم این فانکشن ها اعم از فانکشنهای سیستم و فانکشن های IEC در SCL به روشی که ذکر شد یعنی از طریق منوی Insert>Block Call قابل فراخوانی هستند علاوه بر آنها در SCL فانکشن هایی در SCL وجود دارد که خاص خود آن هستند قبل از اینکه به این فانکشن های خاص بپردازیم مثالی از کاربردهای فانکشن CONCAT و RIGHT و LEFT که از فانکشنهایIEC هستند و نام اصلی آنها بترتیب FC2 و FC32 و FC20 می باشد را در SCL مطرح می نماییم . در این برنامه برای شرایط مختلف ۵ وسیله یعنی Motor و Valve و Press و Weldingstation و Burner در شرایط مختلف مانند بروز مشکل یا روشن شدن یا تحت تعمیر بودن پیغام مناسب تولید میگردد. همانطور که ملاحظه میشود نام وسایل و شرایط مختلف در یک دیتا بلاک بصورت String ذخیره شده سپس در برنامه بسته به شرایط با هم ترکیب شده وپیغام ایجاد شده در سطرهای دیگری از دیتا بلاک نوشته میشود.جمعاً ۲۰ پیغام قابل تولید است.

```
DATA_BLOCK Messagetexts
STRUCT
Index : int;
textbuffer : array [0..19] of string[34];
HW : array [1..5] of string[16]; //5 different devices
statuses : array [1..5] of string[12]; // 5 different statuses
END_STRUCT
BEGIN
Index :=0;
HW[1] := 'Motor ';
HW[2] := 'Valve ';
HW[3] := 'Press ';
HW[4] := 'Weldingstation ';
HW[5] := 'Burner ';
Statuses[1] := ' problem';
Statuses[2] := ' started';
Statuses[3] := ' temperature';
Statuses[4] := ' repaired';
Statuses[5] := ' maintained';
END_DATA_BLOCK
```

فانکشن زیر برای ترکیب کلمات و ایجاد پیغام بکار رفته است:

```
FUNCTION Textgenerator : bool
VAR_INPUT
unit : int; //Index of the device text
no : int; // ID no. of the device
status : int;
value : int;
END_VAR
VAR_TEMP
text : string[34];
```

```
i : int;
END_VAR
//initialization of the temporary variables
text := '';
Textgenerator := true;

Case unit of
1..5 : case status of
1..5 : text := concat( in1 := Messagetexts.HW[unit],
in2 := right(l:=2,in:=I_STRNG(no)));
text := concat( in1 := text,
in2 := Messagetexts.statuses[status]);
if value <> 0 then
text := concat( in1 := text,
in2 := I_STRNG(value));
end_if;
else Textgenerator := false;
end_case;
else Textgenerator := false;
end_case;
i := Messagetexts.index;
Messagetexts.textbuffer[i] := text;
Messagetexts.index := (i+1) mod 20;
END_FUNCTION
```

از OB1 فانکشن قبلی با مقادیر مناسب صدا زده میشود بعنوان مثال اگر پیغام "Motor 12 started" مد نظر باشد کافیست به ورودی error فانکشن عدد ۱ را بدهیم تا Motor را انتخاب کند سپس به ورودی no شماره موتور یعنی عدد ۱۲ را بدهیم و پس از آن به ورودی Status عدد ۲ را نسبت بدهیم تا Statrted را از دیتا بلاک بردارد. سه عدد فوق از MW0 و IW2 و MW2 خوانده میشوند بنابراین روی آنها تبدیل Word به Integer انجام گرفته است.

```
Organization_block  OB1
Var_temp
Opsy_ifx : array [0..20] of byte;
error: BOOL;
End_var;

if %M10.0 <> %M10.1 then
error := Textgenerator (unit := word_to_int(MW0),
no := word_to_int(IW2),
status := word_to_int(MW2),
value := 0);
%M10.1:=M10.0;
end_if;
end_organization_block
```

### فانکشن های استاندارد خاص SCL

این فانکشن ها را میتوان به ۳ دسته زیر تقسیم کرد:

- فانکشن های تبدیل
- فانکشن های عددی
- فانکشن های شیفت و چرخش

### فانکشن های تبدیل در SCL

در دستورات SCL وقتی دو مقدار یا متغیر با هم مقایسه میشوند یا روی آنها عملیاتی انجام میگیرد لازم است همنوع باشند از اینرو بعضاً نیاز به تبدیل بویژه وقتی از آدرسهای CPU استفاده میشود وجود دارد. بعنوان مثال مقایسه MW0 با عدد صحیح ۲۳ با خطا در هنگام کامپایل مواجه میشود و ناچاریم ابتدا تبدیل Word به Int را روی MW0 توسط فانکشن WORD_TO_INT انجام دهیم.

فانکشنهای تبدیل به دو کلاس A و B تقسیم می شوند فانکشنهای کلاس A برای آدرسهایی بکار میرود که هردو در یک دسته قرار میگیرند به عنوان مثال در تبدیل byte به word چون بایت در word قرار میگیرد هم دسته هستند این تبدیل نیاز به ذکر نام فانکشن ندارد و توسط SCL بطور خودکار انجام میشود.بدیهی است بعنوان مثال در تبدیل Byte به Word بیت های سمت چپ با صفر پر میشوند. مثال زیر نشان میدهد که در مقایسه دو مقدار هم دسته نیاز به استفاده از فانکشن تبدیل نیست:

```
VAR
D1 : BYTE ;
D2 : WORD ;
END_VAR
BEGIN
IF (D1 <> D2) THEN
// statement
END_IF
```

به همین نحو اگر دو عدد غیر همنوع مثلاً صحیح و اعشاری با هم مقایسه شوند تبدیل بطور خودکار توسط SCL انجام می گیرد. لیست فانکشنهای کلاس A در جدول زیر آمده است:

| Function Name | Function Name |
|---|---|
| BOOL_TO_BYTE | CHAR_TO_STRING |
| BOOL_TO_DWORD | DINT_TO_REAL |
| BOOL_TO_WORD | INT_TO_DINT |
| BYTE_TO_DWORD | INT_TO_REAL |
| BYTE_TO_WORD | WORD_TO_DWORD |

فانکشنهای کلاس B فانکشن هایی هستند که روی متغیرها یا مقادیر غیر هم دسته بکار میروند مثلاً تبدیل Word به Integer یا بر عکس . این تبدیل توسط SCL بطور خودکار انجام نمیشود و کاربر بایستی فانکشن مربوطه را در برنامه بکار ببرد. این فانکشنها در جدول زیر آورده شده اند . از روی نام فانکشن میتوان عملکرد آن را شناخت .

| Function Name | Function Name |
|---|---|
| BYTE_TO_BOOL | DWORD_TO_WORD |
| BYTE_TO_CHAR | INT_TO_CHAR |
| CHAR_TO_BYTE | INT_TO_WORD |
| CHAR_TO_INT | REAL_TO_DINT |
| DATE_TO_DINT | REAL_TO_DWORD |
| DINT_TO_DATE | REAL_TO_INT |
| DINT_TO_DWORD | STRING_TO_CHAR |
| DINT_TO_INT | TIME_TO_DINT |
| DINT_TO_TIME | TOD_TO_DINT |
| DINT_TO_TOD | WORD_TO_BOOL |
| DWORD_TO_BOOL | WORD_TO_BYTE |
| DWORD_TO_BYTE | WORD_TO_INT |
| DWORD_TO_DINT | WORD_TO_BLOCK_DB |
| DWORD_TO_REAL | BLOCK_DB_TO_WORD |

در استفاده از آدرسهای حافظه CPU نیاز به استفاده از فانکشنهای تبدیل INT_TOWORD و WORD_TO_INT وجود دارد. بعنوان مثال وقتی میخواهیم مقدار عدد صحیح ۲۳ را در MW0 ذخیره کنیم دستور SCL به شکل زیر خواهد بود :

```
MW0:=INT_TO_WORD(23);
```

اگر معادل STL دستور فوق را پس از کامپایل در بلاک مربوطه ببینیم بصورت زیر است:

```
L    W#16#17
T    QW    0
```

خواننده محترم میتواند کاربرد این فانکشنها را در مثال های بخش بعد بهتر بررسی نماید.

### فانکشن های عددی در SCL

این فانکشنها یک ورودی دارند و تابع ریاضی خاصی را روی آن ورودی اعمال می کنند. لیست این فانکشنها در جدول زیر همراه با نحوه صدا زدن آنها با ذکر مثال آمده است .

| فانکشن | مثال | نتیجه |
|---|---|---|
| ABS | RESULT := ABS (-5) ; | //5 |
| SQR | RESULT := SQR (23); | //529 |
| SQRT | RESULT := SQRT (81.0); | //9 |
| EXP | RESULT := EXP (4.1); | //60.340 ... |
| EXPD | RESULT := EXPD (3); | //1_000 |
| LN | RESULT := LN (2.718 281) ; | //1 |
| LOG | RESULT := LOG (245); | //2.389_166 ... |
| ACOS | RESULT := ACOS (0.5); | //1.047_197 (=PI / 3) |
| ASIN | | |
| ATAN | | |
| COS | | |
| SIN | PI := 3. 141 592 ;<br>RESULT := SIN (PI / 6) ; | //0.5 |
| TAN | | |

### فانکشن های شیفت و چرخش در SCL

این فانکشنها دو ورودی دارند ورودی IN که مقدار را میگیرد و میتواند بصورتDWORD,WORD,BYTE BOOL, باشد ورودی دوم آن N است که یک عدد صحیح میباشد و تعداد شیفت یا چرخش را مشخص می کند. لیست این فانکشن ها همراه با مثال در جدول زیر آمده است .

| فانکشن | عملکرد | مثال | نتیجه |
|---|---|---|---|
| ROL | چرخش چپ | RESULT := ROL<br>(IN:=BYTE#2#1101_0011, N:=5); | //2#0111_1010<br>//(= 122 decimal) |
| ROR | چرخش راست | RESULT := ROR<br>(IN:=BYTE#2#1101_0011, N:=2); | //2#1111_0100<br>//(= 244 decimal) |
| SHL | شیفت چپ | RESULT := SHL<br>(IN:=BYTE#2#1101_0011, N:=3); | //2#1001_1000<br>//(= 152 decimal) |
| SHR | شیفت راست | RESULT := SHR<br>(IN:=BYTE#2#1101_0011, N:=2); | //2#0011_0100<br>//(= 52 decimal) |

## ۱۳-۱۰ مثال هایی از برنامه نویسی SCL

**مثال ۱** : محاسبه مقدار میانگین بین تعداد نامشخصی از متغیر های حافظه

مقادیری بصورت عدد اعشاری در متغیرهای حافظه از آدرس MD0 به بعد ریخته شده اند و نیاز است که مقدار میانگین آنها محاسبه شود.

همانطور که ملاحظه میشود محاسبه میانگین توسط FC1 انجام شده و نتیجه در MD100 ریخته میشود تعداد اعداد از بلاک ماقبل یعنی OB1 به FC1 داده میشود. در FC1 ابتدا مقادیر در داخل یک حلقه FOR که از صفر شروع شده ولی مقدار نهایی آن متغیر است و بستگی به تعداد اعداد دارد با هم جمع میشوند. برای متغیرهای حافظه از آدرس دهی ایندکس استفاده شده است بصورت MD[i] که مقدار I هر بار ۴ واحد افزایش می یابد . برای اینکه امکان جمع بصورت عددی فراهم شود توسط فانکشن تبدیل متغیر MD[i] از Dword به Real تبدیل گردیده است. در داخل حلقه نتیجه جمع در متغیر محلی Sum ریخته شده و پس از خروج از حلقه با تقسیم کردن sum بر تعداد اعداد میانگین بدست آمده است. نهایتاً با تبدیل معکوس یعنی Real به Dword مقدار میانگین به MD100 داده شده است.

```
FUNCTION FC1: VOID

VAR_TEMP
k,i:INT;
sum,avg:REAL;
END_VAR

VAR_INPUT
n:INT;
END_VAR

BEGIN
sum:=0;
k:=4*(n-1);
FOR i:= 0 TO k  BY 4 DO
   sum:= sum+DWORD_TO_REAL(MD[i]);
END_FOR;
avg:=(sum/n);

MD100:=REAL_TO_DWORD(avg);

 END_FUNCTION

ORGANIZATION_BLOCK OB1

VAR_TEMP
END_VAR
FC1(n := 10);

END_ORGANIZATION_BLOCK
```

**مثال ۲** : محاسبه مقدار ماکزیمم بین ۸ ورودی آنالوگ

ترموکوپل هایی به ۸ کانال یک کارت ورودی آنالوگ متصل هستند که مقدار دما را از نقاط مختلف یک فرآیند در اختیار PLC قرار می دهند.آدرس این کانالها در Hwconfig از 252 شروع می شود. حساسیت پروسه به گونه ای است که همواره لازمست نسبت به مقدار ماکزیمم دمای ۸ نقطه مزبور واکنش نشان داده شود اگر دمای ماکزیمم بین 1000 درجه تا1100 درجه باشد لازم است خروجی آلارم Q0.0 فعال و اگر بین 1100 درجه تا1200 درجه باشد لازم است خروجی آلارم Q0.1 فعال گردد. مقدار ماکزیمم دما نیز در اولین آدرس دیتا بلاک DB1 ذخیره میشود.

برای محاسبه مقدار ماکزیمم از حلقه FOR و آدرس دهی ایندکس بصورت PIW[i] برای ورودی های آنالوگ استفاده شده است. پس از یافتن ماکزیمم مقدار در DB1 ذخیره شده و با استفاده از دستور CASE شرایط مورد نظر چک شده اند . توجه شود که در دستور مزبور شرایط براساس A/D شده سیگنال آنالوگ نوشته میشوند که برای دما ۱۰ برابر میباشد.

```
ORGANIZATION_BLOCK OB1

VAR_TEMP
mx,i: INT;
END_VAR
mx:=0;

FOR i:= 0 TO 16 BY 2 DO
IF WORD_TO_INT(piw[i+252]) >mx THEN
   mx:=WORD_TO_INT(piw[i+252]);
   END_IF;

END_FOR;
db1.dbw0:=INT_TO_WORD(mx);


CASE mx OF
   10000..11000: Q0.0:=true;
   11010..12000 :Q0.1:=true;
ELSE:
   Q0.0:=false;
   Q0.1:=false;

   END_CASE;
 ;
END_ORGANIZATION_BLOCK
```

**مثال ۳** : ایجاد Setpoint متغیر در بازه های زمانی خاص

در فرآیندی برای انجام عملیات حرارتی روی محصول لازم است Setpoint کنترلر طبق نمودار خاصی تغییر کند تا محصول با ویژگی مورد نظر بدست آید. در این مثال دمای اولیه ۴۰ درجه است و محصول بمدت 2.5 ساعت در سه فاز بصورت زیر تحت عملیات حرارتی قرار میگیرد:

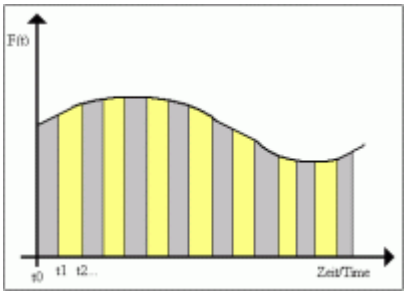
- فاز اول: در مدت یکساعت دما بصورت خطی از ۴۰ به ۴۰۰ افزایش یابد.
- فاز دوم : در مدت یکساعت دما در ۴۰۰ درجه ثابت بماند.
- فاز سوم : در مدت نیم ساعت دما از ۴۰۰ درجه به ۴۰ درجه کاهش یابد.

منطق برنامه بدینصورت است که مقدار دمای مبنا بصورت A/D شده در MW0 و مقدار زمان برحسب ثانیه در متغیر MW2 بصورت عدد صحیح ریخته میشود. ابتدا در بلاک راه اندازی OB100 مقدار دما ۴۰ درجه و مقدار زمان با صفر تنظیم میگردد سپس در داخل OB35 که زمان اجرای پریودیک آن هر یک ثانیه یکبار تنظیم شده منطق اصلی برنامه نوشته میشود. در این بلاک زمان هر بار یک واحد افزایش می یابد سپس توسط دستور CASE شرایط هر سه فاز چک میشود اگر زمان بین صفر تا 3600 ثانیه باشد دما لازم است هر یک ثانیه 0.1 درجه افزایش یابد تا در انتهای این فاز به ۴۰۰ درجه برسد و اگر زمان بین 3600 تا 7200 ثانیه باشد فاز دوم شروع میشود که مقدار مبنای دما ثابت و برابر ۴۰۰ درجه است و چنانچه زمان بین 7200 تا 9000 ثانیه باشد لازم است هر بار که OB35 اجرا میشود یعنی هر یک ثانیه یک بار دما 0.2 درجه کاهش یابد . پس از رسیدن به ۴۰ درجه PLC توسط SFC46 متوقف میگردد.

```
ORGANIZATION_BLOCK OB100
mw0:=INT_TO_WORD(400);
mw2:=INT_TO_WORD(0);
END_ORGANIZATION_BLOCK

ORGANIZATION_BLOCK OB35
VAR_TEMP
   a,b:INT;
END_VAR
a:=WORD_TO_INT(mw0);
b:=WORD_TO_INT(mw2);
CASE b OF
   0..3600: a:=a+1;
   3601..7200: a:=a;
   7201..9000 : a:=a-2;
ELSE
   stp();
END_CASE;
mw0:=INT_TO_WORD(a);
mw2:=INT_TO_WORD(b);
END_ORGANIZATION_BLOCK
```

**مثال ٤** : انتگرال گیری از یک متغیر با تغییرات نامشخص

فرض کنید در یک فرآیند موادی بطور مداوم روی نوار نقاله متحرکی ریخته شده و سپس به قسمت بعدی فرآیند منتقل میشوند . نوار نقاله متحرک بصورت لحظه ای وزن موادی که انتقال میدهد را به PLC میفرستد. برای داشتن مقدار کل وزن مواد منتقل شده نیاز به انتگرال گیری در بازه زمانی مورد نظر داریم. روش انتگرال گیری در برنامه PLC با جمع زدن مقادیر در واحد زمان است. میتوان این کار را به روشهای مختلفی انجام داد. میدانیم وقتی تابع تغییرات یک متغیر در دست نباشد برای انتگرال گیری از آن میتوان تغییرات آن متغیر را در به فواصل زمانی کوچک بصورت ذوزنقه های کوچک تخمین زد و با جمع کردن این ذوزتقه ها مقدار انتگرال یا سطح زیر منحنی را بدست آورد شکل زیر:




مقدار انتگرال = 0.5 * (f(t1) + f(t0))*(t1-t0) + 0.5 * (f(t2) + f(t1))*(t2-t1)+ ……..

در برنامه زیر از متغیری که به ورودی IN داده شده است طبق روش فوق انتگرال گیری می شود. برای زمان از فانکشن SFC64 با نام سمبلیک TIME_TCK که در فصل ۱۲ مورد بحث قرار گرفت استفاده شده است. زمان CPU توسط این فانکشن مرتباً خوانده میشود سپس با تبدیل آن از میلی ثانیه به ثانیه و سپس تبدیل به عدد DINT مبنای محاسبه قرار میگیرد.انتگرال گیری با ورودی Enable فعال و مقدار انتگرال به خروجی OUT داده میشود. یک ورودی نیز برای Reset کردن مقدار انتگرال در نظر گرفته شده است.

```
FUNCTION_BLOCK FB1  // Integral calculation /
VAR_INPUT
   IN : REAL;
   RESET : BOOL;
   ENABLE : BOOL;
END_VAR
VAR_OUTPUT
   OUT : REAL;
   RESET_ACTIV : BOOL;
END_VAR
VAR
   OUT_LOW : REAL;
   LAST_IN : REAL;
   LAST_OUT : REAL;
```

```
   LAST_TIME : REAL;
   ACTUAL_TIME : REAL;
   X : REAL;
   n : INT;
END_VAR

// Reset of values
RESET_ACTIV := RESET;
IF RESET = TRUE THEN
   OUT := 0.0;
   OUT_LOW := 0.0;
   LAST_OUT := 0.0;
   LAST_TIME := ACTUAL_TIME;
   ACTUAL_TIME := TIME_TO_DINT(TIME_TCK()) / 1000.0;
   X := 0.0;
   n := 0;
ELSIF ENABLE = FALSE THEN
   n := 0;
ELSE
   //First Integral Cyclus
   IF n = 0 THEN
         ACTUAL_TIME := TIME_TO_DINT(TIME_TCK()) / 1000.0;
         LAST_TIME := ACTUAL_TIME;
         LAST_IN := IN;
         n:=1;
   ELSE

      // Input
      ACTUAL_TIME := TIME_TO_DINT(TIME_TCK()) / 1000.0;

      // Overflow Correction
      IF ACTUAL_TIME < LAST_TIME THEN
         X := (ACTUAL_TIME - LAST_TIME + 2147483.647) * (IN + LAST_IN) / 2;
      ELSE
         X := (ACTUAL_TIME - LAST_TIME) * (IN + LAST_IN) / 2;
      END_IF;
      LAST_TIME := ACTUAL_TIME;
      LAST_IN   := IN;

      // Integral Calculation
      LAST_OUT := OUT;
      OUT := LAST_OUT + X;
      OUT_LOW :=  (OUT - LAST_OUT) - X + OUT_LOW;
      IF OUT_LOW <> 0 THEN
         IF ABS(OUT/OUT_LOW) < 10000000 THEN
            LAST_OUT := OUT;
            OUT := OUT  - OUT_LOW;
            OUT_LOW := (OUT - LAST_OUT) + OUT_LOW;
         END_IF;
      END_IF;
END_IF;

END_IF;
END_FUNCTION_BLOCK
```

**مثال ۵** : انتگرال گیری از یک متغیر با تغییرات مشخص با کاربرد OB3x

اگر تابع تغییرات متغیر در واحد زمان مشخص باشد به سهولت میتوان از آن با تقریب ذکر شده برای سطح زیر منحنی در صفحه قبل انتگرال را بدست آورد. فرض کنید تابع تغییرات بر حسب زمان بصورت زیر است:

$$Y= \ln(t)+ \sqrt{(t^3 +t+2)}$$

برای محاسبه انتگرال کافیست هر بار مقدار زیر را محاسبه کرده با مقدار لحظه قبل جمع کنیم:

$$0.5 * ( f(t0)+f(t1)* \Delta t$$

که در آن t1 لحظه فعلی و t0 لحظه قبلی است .اگر این تابع را در OB35 که زمان اجرای پریودیک آن 1 ثانیه تنظیم شده بکار ببریم بنابراین فاصله $\Delta t$ ثابت و برابر ۱ است پس مقدار زیر را در هر بار فراخوانی OB35 با مقدار قبلی جمع میکنیم.

$$0.5 * ( f(t0)+f(t1)$$

برنامه زیر این انتگرال گیری را انجام می دهد با فعال شدن سوئیچ I0.0 انتگرال گیری متوقف می شود.

```
ORGANIZATION_BLOCK ob35

VAR_TEMP
   t:int;
   y:real;
END_VAR

LABEL
   result;
END_LABEL

IF i0.0 THEN
   GOTO result;
  ELSE
y:=y+(LN(t)+SQRT(t**3 + t+2));
   t:=t+1;
END_IF;
result: ;

END_ORGANIZATION_BLOCK
```

**مثال ٦** : برنامه نویسی OB80 در SCL

همانطور که در جلد اول کتاب توضیح داده شد OB8x ها برای Error Handling بکار میروند و با استفاده از متغیر های Temp بالای آنها میتوان برنامه نویسی لازم را انجام داد و خطا را قبل از وقوع مدیریت کرد.

در SCL نیز می توان این OB ها را ایجاد و برنامه نویسی کرد تنها نکته ای که وجود دارد نحوه تعریف متغیرهای temp است که ممکن است کاربر آنها را نشناسد یا تعریف آنها در بخش Var_temp وقت گیر باشد. برای این کار فایلهایی با نام تمام OB ها حاوی متغیرهای Temp در مسیر زیر قرار دارد که با هر Editor قابل باز کردن هستند:

**Drive:\Siemens\Step7\S7DATA\S7wiz**

فایل مورد نظر را باز کرده و متغیرهای temp را از آن کپی نموده و در بخش Var_temp از برنامه SCL قرار دهید سپس میتوانید از آنها در برنامه نویسی استفاده نمایید.

با توجه به توضیحات فوق OB80 توسط SCL بصورت زیر ایجاد و برنامه نویسی شده است:

```
ORGANIZATION_BLOCK OB80

VAR_TEMP
  OB80_EV_CLASS : BYTE ;    //16#35, Event class 3, Entering event state, Internal fault event
  OB80_FLT_ID : BYTE ;    //16#XX, Fault identifcation code
  OB80_PRIORITY : BYTE ;    //26 (Priority of 1 is lowest)
  OB80_OB_NUMBR : BYTE ;    //80 (Organization block 80, OB80)
  OB80_RESERVED_1 : BYTE ;    //Reserved for system
  OB80_RESERVED_2 : BYTE ;    //Reserved for system
  OB80_ERROR_INFO : WORD ;    //Error information on event
  OB80_ERR_EV_CLASS : BYTE ;    //Class of event causing error
  OB80_ERR_EV_NUM : BYTE ;    //Number of event causing error
  OB80_OB_PRIORITY : BYTE ;    //Priority of OB causing error
  OB80_OB_NUM : BYTE ;    //Number of OB causing error
  OB80_DATE_TIME : DATE_AND_TIME ;    //Date and time OB80 started
END_VAR

IF OB80_FLT_ID=b#16#01 THEN
   m0.0:=true ;
ELSE
   m0.0:=false;
   END_IF;


END_ORGANIZATION_BLOCK
```

**مثال ۷** : مرتب کردن دیتاهای یک دیتابلاک

در سطرهای DB1 تعدادی از اعداد صحیح ریخته شده اند میخواهیم آنها را از کوچک به بزرگ مرتب کنیم برنامه مرتب سازی در FC1 نوشته شده و این FC از OB1 صدا زده میشود . در هنگام فراخوانی تعداد اعداد توسط OB1 به FC1 داده میشود.برای تست کردن این برنامه در DB1 ده مقدار وارد کنید سپس آنرا بصورت Online ببینید و در مد Online مقادیر ستون Actual Value را تغییر داده دانلود کنیدو نتیجه را مشاهده نمایید.

```
FUNCTION FC1: void

VAR_TEMP
temp1,i,low,swap:INT;
END_VAR

VAR_INPUT
n:INT;
END_VAR

LABEL
   a001,a002;
END_LABEL

i:=2*n-2;
low:=0;
a002:swap:=0;
a001:low:=low+1;
IF WORD_TO_INT(db1.dbw[i-2])>WORD_TO_INT(db1.dbw[i]) THEN
   temp1:=WORD_TO_INT(db1.dbw[i-2]);
      db1.dbw[i-2]:=db1.dbw[i];
      db1.dbw[i]:=INT_TO_WORD(temp1);
      swap:=1;
ELSE
   i:=i-2;
END_IF;

IF i>= low THEN
   GOTO a001;
END_IF;
mw0:=INT_TO_WORD(i);

   IF swap<>0 THEN
      GOTO a002;
      END_IF;
END_FUNCTION

ORGANIZATION_BLOCK OB1
VAR_TEMP
END_VAR
fc1(n:=10);
END_ORGANIZATION_BLOCK
```

**مثال ۸** : راه اندازی مجدد تایمر با زمان باقیمانده از قبل

تایمر پالس با زمان مشخصی تنظیم شده و با فعال شدن کلید I0.0 شروع بکار میکند تا زمان آن سپری شده و قطع شود. میخواهیم اگر قبل از سپری شدن زمان کلید I0.0 قطع شد پس از وصل مجدد تایمر با زمان باقیمانده قبلی ادامه دهد بعنوان مثال اگر ۲ ثانیه زمان تایمر باقیمانده بود و کلید قطع شد با وصل مجدد کلید تایمر با زمان ۲ ثانیه شروع بکار کند.

```
ORGANIZATION_BLOCK OB1

VAR_TEMP
 // Reserved
 Info: ARRAY[0..19] OF BYTE;

currtime,t_remain:S5TIME;
aa,bival:WORD;
END_VAR
LABEL
   next;
   END_LABEL

IF i0.0=false THEN
   aa:=bival;
   t_remain:=currtime;
   IF bival<>16#0 THEN
      m0.0:=1;
ELSE
   m0.0:=false;
   END_IF;

END_IF;

IF m0.0=true THEN
   CurrTime :=S_PULSE(T_NO:= T14, S:=i0.0, TV:= T_remain, R:=i0.1,
BI:=biVal,Q:=q0.0);
ELSE
   CurrTime :=S_PULSE(T_NO:= T14, S:=i0.0, TV:= T#20s, R:=i0.1,
BI:=biVal,Q:=q0.0);
   END_IF;

END_ORGANIZATION_BLOCK
```

**مثال ۹** : آرشیو سازی یک ورودی آنالوگ

میخواهیم مقدار دما را در یک پروسه آرشیو نماییم. ترموکوپل دما را به ورودی کارت آنالوگ با آدرس PIW260 می فرستد. هر ۲ ثانیه یکبار از مقدار دما نمونه برداری شده و نتیجه در یک دیتا بلاک ذخیره میشود. دیتا بلاک دارای ۱۰۰ سطر از نوع Integer است و تا ۱۰۰ مقدار را آرشیو می کند وقتی نمونه جدید می رسد نمونه های قبلی یک سطر به پایین شیفت پیدا میکنند یعنی سطر شماره ۱۰۰ از بین میرود و سطر ۹۹ به جای آن می نشیند وهمینطور سایر سطرها به پایین جابجا میشوند تا نهایتاً نمونه جدید در سطر اول دیتا بلاک قرار میگیرد. برنامه در OB35 نوشته میشود و در پارامترهای CPU زمان اجرای این OB هر ۲ ثانیه یکبار تنظیم می گردد . برای داشتن مقدار دما لازم است آنچه از کارت AI خوانده و تبدیل شده بر 10 تقسیم شود.

```
ORGANIZATION_BLOCK OB35
VAR_TEMP
m1,i: INT;
END_VAR
   m1:=word_to_int(piw260) / 10;
   db1.dbw0:= INT_TO_WORD(m1);
FOR i:= 18 TO 2  BY -2 DO
 db1.dbw[i]:=  db1.dbw[i-2];
END_FOR;
END_ORGANIZATION_BLOCK
```

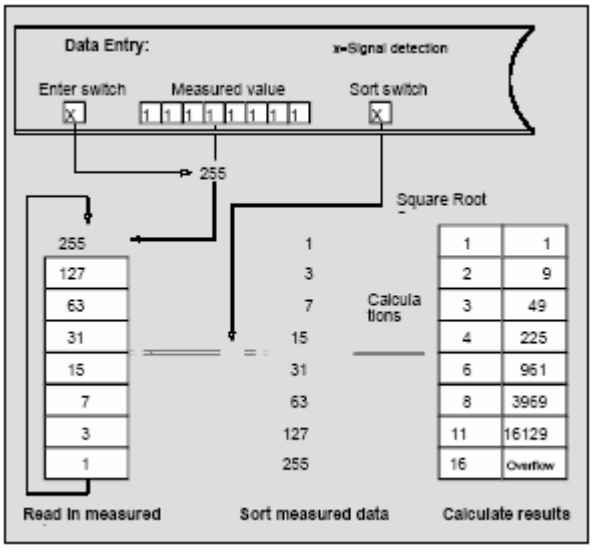
پس از کامپایل این برنامه بلاک ساخته شده را همراه با DB1 (که لازم است حاوی ۱۰۰ سطر باشد) به PLC دانلود کنید سپس محتویات ستون Actual Value دیتا بلاک را در حالت Online ببینید.

اگر بخواهیم فقط در صورت تغییر دما مقدار جدید در دیتا بلاک ذخیره شود برنامه بصورت زیر در می آید:

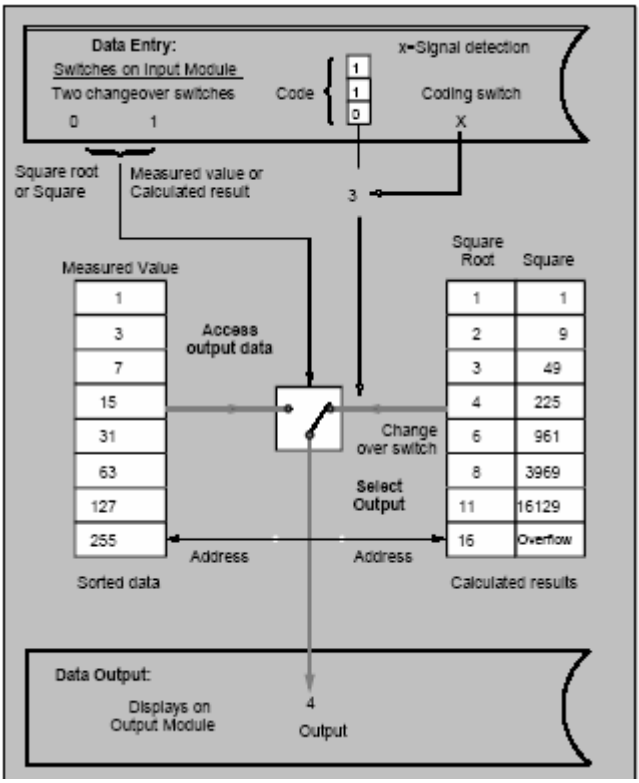
```
ORGANIZATION_BLOCK OB35
VAR_TEMP
m1,i: INT;
END_VAR
LABEL
   nochange;
END_LABEL
IF piw260=mw4 THEN
   GOTO nochange;
   ELSE
   m1:=word_to_int(piw260) / 10;
   db1.dbw0:= INT_TO_WORD(m1);
FOR i:= 18 TO 2  BY -2 DO
 db1.dbw[i]:=  db1.dbw[i-2];
END_FOR;
END_IF;
mw4:=piw260;
nochange: ;
END_ORGANIZATION_BLOCK
```

**مثال ۱۰** : ذخیره سازی و نمایش مقدار اندازه گیری شده

این مثال کاربرد جامعی از فانکشن ها و دستورات SCL را نشان میدهد. ۸ سوئیچ ورودی به کانالهای یک کارت DI از آدرس I1.0 تا I1.7 یعنی بایت ۱ متصل هستند و متناسب با وضعیت پروسه تغییر می یابند در هر لحظه وضعیت این ۸ سوئیچ معرف یک عدد دسیمال بین 0 تا 255 است . با فعال شدن سوئیچ ورودی Enter Switch که به ورودی I0.0 متصل است مقدار فعلی ورودی در یک آرایه ذخیره میشود. آرایه ۸ عنصری است و بصورت یک بافر عمل میکند.وقتی که سیگنال از سوئیچ Sort Switch که به ورودی I0.1 وصل است برسد ابتدا عناصر آرایه بصورت صعودی مرتب شده سپس جذر یا توان دوم تک تک مقادیر محاسبه و ذخیره میشوند.اینکه نتیجه جذر باشد یا ریشه دوم بستگی به وضعیت سوئیچ I0.2 دارد.



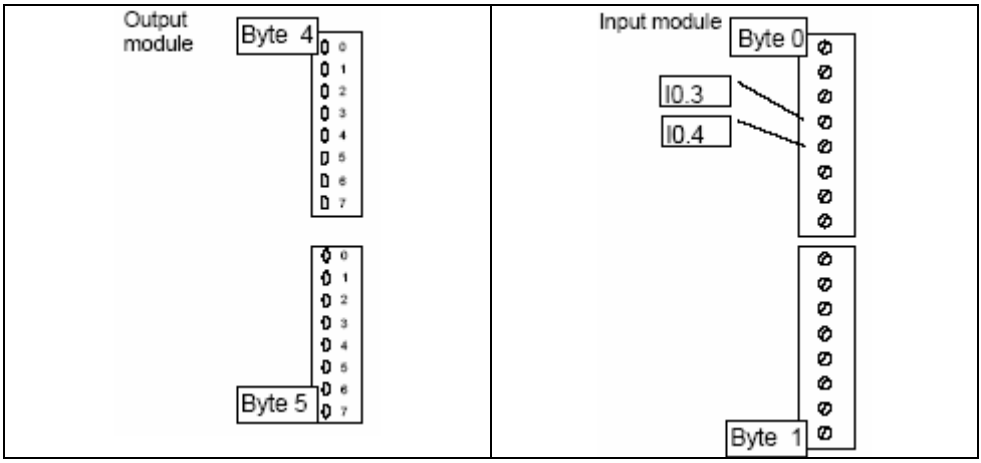

خروجی روی کانالهای یک کارت Digital Output نمایش داده میشود. خروجی میتواند مقدار اندازه گیری فعلی باشد که روی کارت DI در آن لحظه وجود دارد یا انتخاب مقداری از لیست مقادیر در آرایه(جذر یا توان ۲ ) باشد که مرتب شده اند. اینکه کدامیک از این دو روی خروجی نمایش داده شوند بستگی به وضعیت سیگنال ورودی I0.3 دارد. سه کانال دیگر از کارت DI نمایشگر اینست که کدامیک از مقادیر آرایه روی خروجی نمایش داده شود این سه کانال I0.4 , I0.5 , I0.6 هستند بعنوان مثال اگر هر سه صفر باشند عنصر شماره صفر آرایه و وقتی هر سه یک باشند آخرین عنصر آرایه به خروجی منتقل می شود .

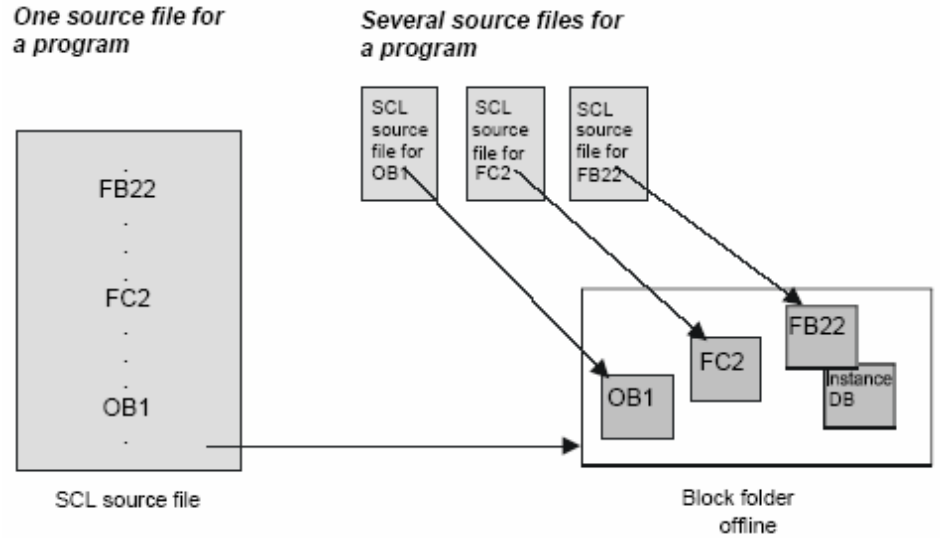



شکل روبرو وضعیت سوئیچ ها برای انتقال عنصر شماره سه آرایه یعنی عدد ۴ به خروجی را نشان می دهد.

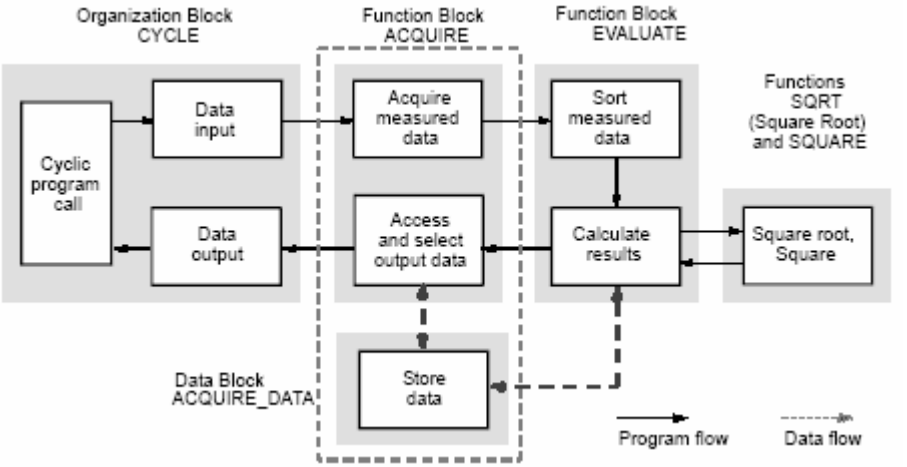
برای انتقال نتیجه به خروجی که ممکن است توان دوم و عددی بزرگ باشد تمام ۱۶ کانال کارت خروجی یعنی بایت ۴ و بایت ۵ در نظر گرفته شده است. بدیهی است بایت ۴ فقط برای توان ۲ بکار میرود و در سایر حالات که عدد کمتر یا مساوی ۲۵۵ است ۸ بیت مربوط به بایت ۵ کفایت می کند.

شکل زیر کانالهای کارتهای ورودی و خروجی را نشان میدهد:






شکل فوق ساختار برنامه را نشان میدهد FC2 فانکشنی است که مقدار را از FB میگیرد و عمل جذر یا توان ۲ را انجام میدهد سپس نتیجه را به FB بر می گرداند.دو FB یکی بنام ACQUIRE برای عملیاتی چون ذخیره سازی مقادیر و دیگری با نام EVALUATE برای مرتب سازی و محاسبه جذر یا توان دوم بکار میرود . FB آنچه نهایتاً انتخاب شده را در اختیار OB قرار میدهد تا به خروجی بفرستد. شکل زیر عملکرد دقیقتر بلاک ها را نشان می دهد.



برای اسامی بلاکها و ورودی خروجی ها اسامی سمبلیک در نظر گرفته شده که در جدول سمبلها طبق شکل روبرو تعریف شده اند.

پارامترهایی که بین فانکشنها تبادل میشود و بعنوان متغیر محلی بلاک ها تعریف می شوند در جدول زیر آورده شده اند:

| | Symbol | Address |
|---|---|---|
| 1 | Coding | IW 0 |
| 2 | Coding switch | I 0.7 |
| 3 | CYCLE | OB 1 |
| 4 | Entry | IB 1 |
| 5 | EVALUATE | FB 20 |
| 6 | Function switch | I 0.2 |
| 7 | Input 0.0 | I 0.0 |
| 8 | Output | QW 4 |
| 9 | Output switch | I 0.3 |
| 10 | ACQUIRE | FB 10 |
| 11 | ACQUIRE_DATA | DB 10 |
| 12 | Sorting switch | I 0.1 |
| 13 | SQUARE | FC 41 |

| Parameter Name | Data Type | Declaration Type | Description |
|---|---|---|---|
| **FUNCTION BLOCK Acquire** | | | |
| measval_in | INT | VAR_INPUT | Measured value |
| newval | BOOL | VAR_INPUT | Switch for entering measured value in ring buffer |
| resort | BOOL | VAR_INPUT | Switch for sorting and evaluating measured data |
| funct_sel | BOOL | VAR_INPUT | Selector switch for square root or square |
| selection | WORD | VAR_INPUT | Code for selecting output value |
| newsel | BOOL | VAR_INPUT | Switch for reading in code |
| result_out | DWORD | VAR_OUTPUT | Output of calculated result |
| measval_out | DWORD | VAR_OUTPUT | Output of measured value |
| **FUNCTION BLOCK Evaluate** | | | |
| sortbuffer | ARRAY[..] OF REAL | VAR_IN_OUT | Measured value array, corresponds to ring buffer |
| calcbuffer | ARRAY[..]OF STRUCT | VAR_OUTPUT | Array for results: Structure with "square root" and "square" components of type INT |
| **FUNCTION SQRT / SQR** | | | |
| value | REAL | VAR_INPUT | Input for SQRT |
| SQRT | REAL | Function value | Square root of input value |
| value | INT | VAR_INPUT | Input for SQUARE |
| SQUARE | INT | Function value | Square of input value |

### فانکشن SQUARE

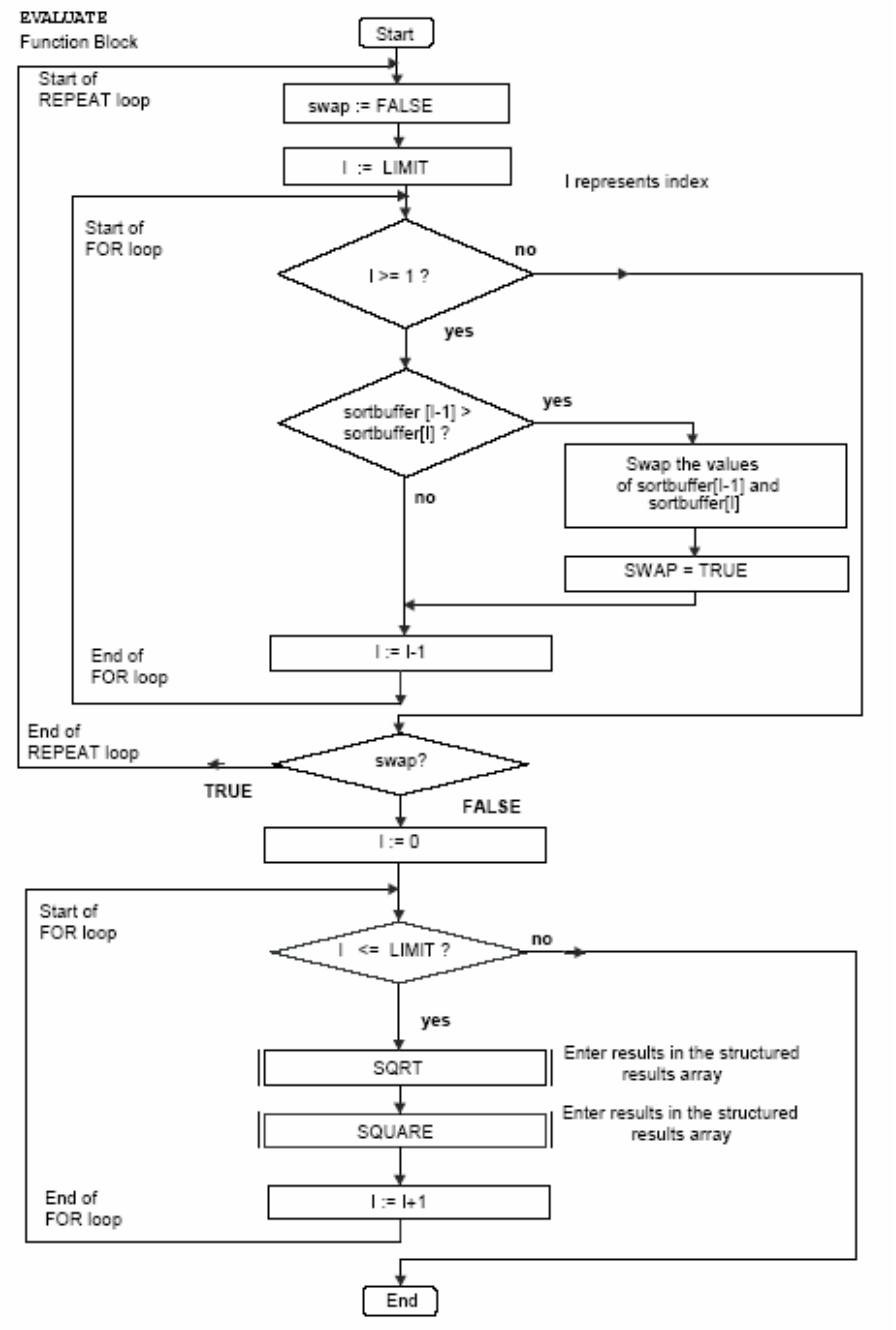
برنامه این فانکشن بصورت زیر است:

```
FUNCTION SQUARE : INT
(*********************************************************
This function returns as its function value the square of the
input value or if there is overflow, the maximum value that
can be represented as an integer.
*********************************************************)
VAR_INPUT
value : INT;
END_VAR
BEGIN
IF value <= 181 THEN
SQUARE := value * value; //Calculation of function
value
ELSE
SQUARE := 32_767; // If overflow, set maximum value
END_IF;
END_FUNCTION
```

### فانکشن بلاک Evaluate

عملکرد این FB طبق فلوچارت صفحه بعد است که در آن نحوه مرتب سازی بصورت حبابی وسایر مراحل آورده شده است . برنامه آن بصورت زیر و در ادامه فلوچارت ارائه گردیده است.

```
FUNCTION_BLOCK EVALUATE

(*******************************************************************************
 Part 1 :  Sort cyclic buffer with measured values
 Part 2 :  Trigger calculation of results
*******************************************************************************)
CONST
 LIMIT := 7;
END_CONST

VAR_IN_OUT
 sortbuffer : ARRAY[0..LIMIT] OF INT;
END_VAR

VAR_OUTPUT
 calcbuffer  : ARRAY[0..LIMIT] OF
   STRUCT
     root  : INT;
     square : INT;
   END_STRUCT;
END_VAR

VAR_TEMP
 swap      : BOOL;
 index, help   : INT;
 valuer, resultno: REAL;
END_VAR
```


EVALUATE
Function Block
Start
Start of
REPEAT loop
swap := FALSE
I := LIMIT
I represents index
Start of
FOR loop
I >= 1 ?
no
yes
sortbuffer [I-1] >
sortbuffer[I] ?
yes
Swap the values
of sortbuffer[I-1] and
sortbuffer[I]
no
SWAP = TRUE
I := I-1
End of
FOR loop
End of
REPEAT loop
swap?
TRUE
FALSE
I := 0
Start of
FOR loop
I <= LIMIT ?
no
yes
SQRT
Enter results in the structured
results array
SQUARE
Enter results in the structured
results array
End of
FOR loop
I := I+1
End

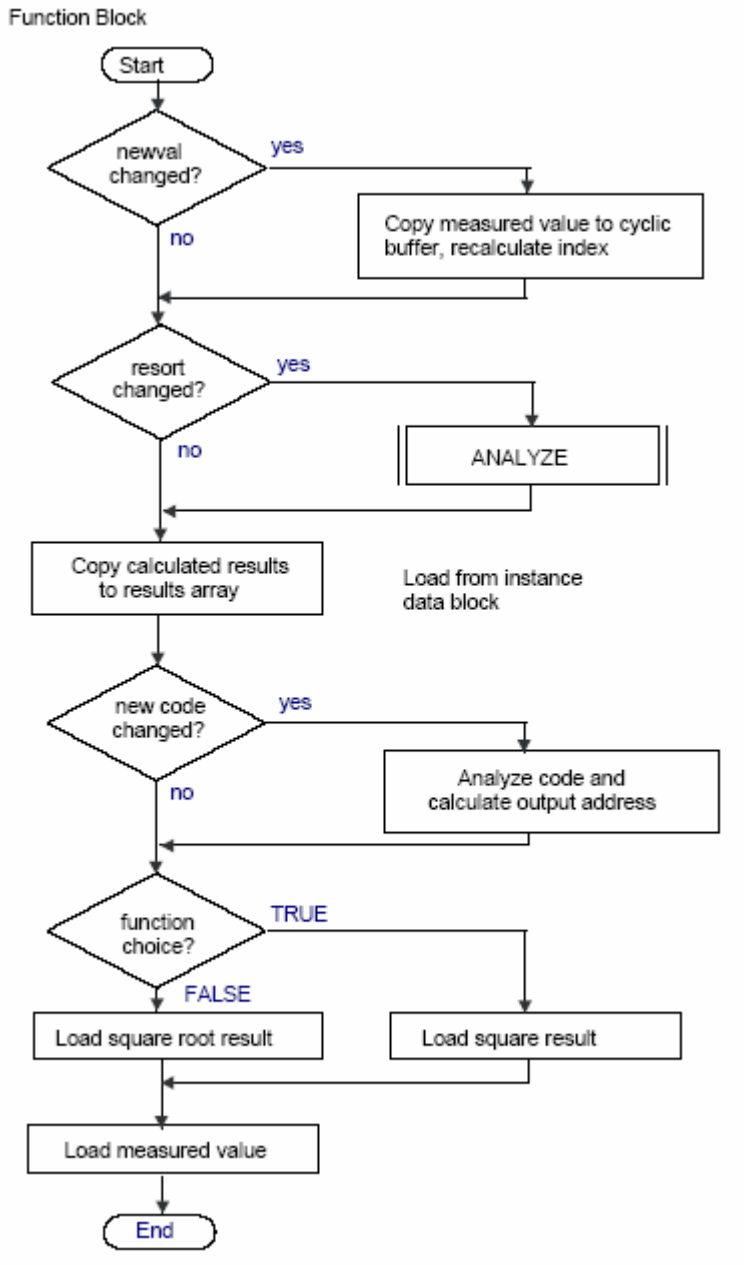
```
BEGIN
(* Part 1  Sorting : ****************************************************
        according to "Bubble Sort" procedure: swap pairs of values
        until measured value buffer is sorted. *)
REPEAT
  swap := FALSE;

    FOR index := LIMIT TO 1 BY -1 DO
      IF sortbuffer[index-1] > sortbuffer[index] THEN
        help                 := sortbuffer[index];
        sortbuffer[index]      := sortbuffer[index-1];
        sortbuffer[index-1]    := help;
        swap                 := TRUE;
      END_IF;
    END_FOR;

  UNTIL NOT swap
END_REPEAT;
(* Part 2  Calculation : ************************************************
        calculates square root using standard function SQRT and
        forms square using function SQUARE. *)

FOR index := 0 TO LIMIT BY 1 DO
  valuer   := INT_TO_REAL(sortbuffer[index]);
  resultno := SQRT(valuer);
  calcbuffer[index].root   := REAL_TO_INT(resultno);
  calcbuffer[index].square := SQUARE(sortbuffer[index]);
END_FOR;
END_FUNCTION_BLOCK
```

### فانکشن بلاک ACQIURE

عملکرد این FB طبق فلوچارت صفحه بعد است که در آن نحوه ذخیره سازی مقادیر و نتایج نشان داده شده و برنامه آن بصورت زیر و در ادامه فلوچارت آورده شده است.

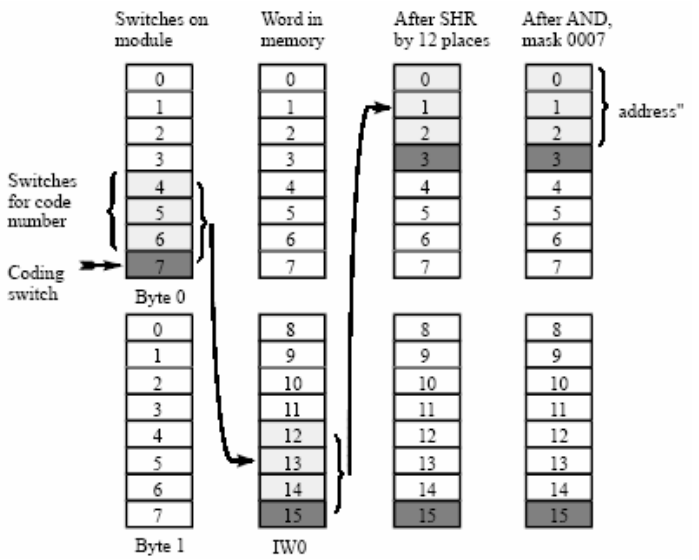
**FUNCTION_BLOCK RECORD**

```
(*****************************************************************************
  Part 1 :  Record measured values   Part 2 :  Trigger sorting and calculating  Part 3 :  Evaluate coding and
prepare output
*****************************************************************************)
CONST
  LIMIT := 7;
  NUMBER := LIMIT + 1;
END_CONST
VAR_INPUT
  measvalue_in : INT ; // New measured value
  newvalue     : BOOL; // Enter measured value in cyclic buffer "measvalues"
  newsort      : BOOL; // Sort measured values
  funcselec    : BOOL; // Select calculation function root/square
  newselec     : BOOL; // Enter output address
  selection    : WORD; // Output address
END_VAR
```


Function Block
Start
newval changed?
yes
no
Copy measured value to cyclic buffer, recalculate index
resort changed?
yes
no
ANALYZE
Copy calculated results to results array
Load from instance data block
new code changed?
yes
no
Analyze code and calculate output address
function choice?
TRUE
FALSE
Load square root result
Load square result
Load measured value
End

```
VAR_OUTPUT
  result_out    : INT; // Calculated value
  measvalue_out : INT; // Corresponding measured value
END_VAR

VAR
  measvalues    : ARRAY[0..LIMIT] OF INT := 8(0);
  resultbuffer  : ARRAY[0..LIMIT] OF
  STRUCT
    root        : INT;
    square      : INT;
  END_STRUCT;
  Index           : INT  := 0;
  oldvalue        : BOOL := TRUE;
  oldsort         : BOOL := TRUE;
  oldselec        : BOOL := TRUE;
  address         : INT  := 0;     //Converted output address
  eval_instance   : EVALUATE;      //Define local instance
END_VAR

BEGIN
(* Part 1 : Record measured values ****************************************
          If "newvalue" is changed, the measured value is entered.
          The instruction MOD realizes a cyclic buffer for measured values.*)

IF newvalue <> oldvalue THEN
  Index                := Index     MOD NUMBER;
  measvalues[Index    ] := measvalue_in;
  Index                := Index     + 1;
END_IF;

oldvalue := newvalue;

(* Part 2 : Trigger sorting and calculating ****************************
          If "newsort" is changed, triggers sorting of cyclic buffer
          and calculation with the measured values. Results are
          stored in a new array, "calcbuffer". *)

IF newsort <> oldsort THEN
  Index     := 0;                          //Reset cyclic buffer Index
  eval_instance(sortbuffer := measvalues);   //Call EVALUATE
END_IF;

oldsort        := newsort;
resultbuffer   := eval_instance.calcbuffer;  //Square and square root

(* Part 3 : Evaluate coding and prepare output **********************
          If "newselec" is changed, the coding for addressing the array
          element for the output is recalculated: the relevant bits
          of "selection" are masked and converted to integers.
          Depending on the switch setting of "funcselec", either "root" or
          "square" is prepared for output. *)

IF newselec <> oldselec THEN
```

```
  address := WORD_TO_INT(SHR(IN := selection, N := 12) AND 16#0007);
END_IF;

oldselec := newselec;

IF funcselec THEN
  result_out    := resultbuffer[address].square;
ELSE
  result_out    := resultbuffer[address].root;
END_IF;
measvalue_out   := measvalues[address]; //Measured value display
END_FUNCTION_BLOCK
```




با دقت به برنامه فانکشن بلاک ACQUIRE می بینیم که دستورات به سه قسمت تقسیم شده اند:

- ذخیره سازی مقادیر : در این قسمت اگر پارامتر ورودی newval با مقدار قبلی oldval متفاوت باشد مقدار جدید در بافر ذخیره میشود.
- شروع مرتب سازی : در این قسمت اگر پارامتر ورودی resort در مقایسه با oldsort تغییر پیدا کند فانکشن بلاک EVALUATE برای عمل مرتب سازی صدا زده میشود.
- بررسی نحوه نمایش خروجی: در این قسمت بسته به وضعیت Coding مقدار مربوطه روی خروجی نمایش داده میشود. بیتهای 12 تا 14 از IW0 شامل Coding هستند و وقتی لبه مثبت در سوئیچ متصل به بیت 15 آشکار شد عمل Coding انجام می شود. طبق شکل فوق آدرس با شیفت راست

و سپس Mask کردن بیت های مربوطه بصورت AND بدست میآید. این آدرس برای ارسال نتایج یا مقادیر به خروجی استفاده می شود. اینکه جذر یا توان ۲ به خروجی منتقل شود بستگی به وضعیت funct_sel دارد.

**بلاک OB1**

این بلاک وظایف زیر را بعهده دارد:

- صدا زدن فانکشن بلاک ACQUIRE و اختصاص ورودی و خروجی های آن.
- خواندن مقادیر برگشتی از فانکشن بلاک فوق
- ارسال خروجی ها به کارت خروجی

```
ORGANIZATION_BLOCK CYCLE
(*********************************************************
CYCLE is like an OB1, i.e. it is called cyclically by the S7
system.
Part 1 : Function block call and transfer of
the input values Part 2 : Reading in of the output values
and output
with output switchover
*********************************************************)
VAR_TEMP
systemdata : ARRAY[0..20] OF BYTE; // Area for OB1
END_VAR
BEGIN
(* Part 1 :
************************************************)
ACQUIRE.ACQUIRE_DATA(
measval_in:= WORD_TO_INT(input),
newval := "Input 0.0", //Input switch as signal
identifier
resort := Sort_switch,
funct_sel:= Function_switch,
newsel := Coding_switch,
selection := Coding);
(* Part 2 :
************************************************)
IF Output_switch THEN
//Output changeover
Output := ACQUIRE_DATA.result_out;
//Square root or square
ELSE
Output := ACQUIRE_DATA.measval_out; //Measured value
END_IF;
END_ORGANIZATION_BLOCK
```

### ۱۳-۱۱ امکانات Debug در SCL

اشکالات Syntax در هنگام کامپایل کردن برنامه مشخص میشوند و تا رفع نشوند بلاک مربوطه ساخته نمیشود ولی اشکالات منطقی برنامه ممکن است Error خاصی تولید نکنند حتی پس از دانلود نیز CPU خطایی را گزارش نکند ولی با این وجود نتیجه مورد نظر بدست نیاید.برای آشکار سازی این خطا ها از ابزار های Debug استفاده میشود. برای کار با این ابزار ها در SCL لازم است تنظیمات زیر از منوی Option>customize در بخش Compiler فعال شده باشند:

- Create Object Code
- Create Debug Info

پس از انجام تنظیمات فوق و دانلود کردن برنامه به PLC امکانات Debug در مد Online در حالتی که CPU در مد Run یا Run-P کار میکند قابل دسترس خواهد بود. Debug دو ابزار زیر را در اختیار می گذارد:

۱. Monitor برای دیدن مقادیر لحظه ای متغیرها در S7-300 و S7-400

۲. Breakpoint برای اجرای خط به خط برنامه فقط در S7-400

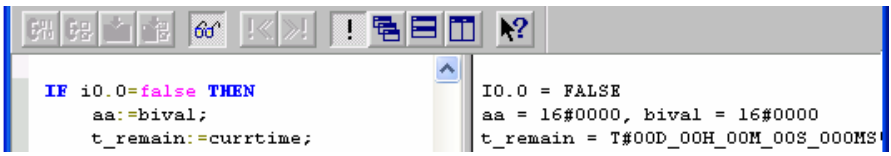
خوانندگان محترم می توانند برای تست این دو ابزار با سیمولاتور CPU را از نوع 414 یا 416 انتخاب نمایند.

### Monitor کردن برنامه SCL

با استفاده از امکان مانیتورینگ میتوان وضعیت لحظه ای متغیرها و آدرس ها را در برنامه SCL مشاهده کرد . قبل از هر چیز بایستی توجه داشت که Monitoring در حالت Online ممکن است سیکل اسکن CPU را افزایش دهد . این امر بستگی به انتخاب های زیر از منوی Debug>operation دارد:

- Test Operation : در این حالت تمام امکانات مانیتورینگ برای کاربر فراهم است ولی سیکل اسکن را تا حد زیادی افزایش می دهد.
- Process Operation : در اینحالت امکانات مانیتورینگ در حد محدود توسط CPU فراهم شده و سیکل اسکن در حد معمول حفظ می گردد.

با در نظر گرفتن نکات فوق کاربر میتواند Monitor را از طریق منوی Debug>Monitor یا آیکون عینک شکل بالای پنجره فعال نماید. پنجره به دوقسمت مانند شکل زیر تقسیم شده که در سمت راست مقادیر لحظه ای متغیر ها نمایش داده میشود.برای اتمام مانیتور میتوان روی آیکون عینک یا منوی View>Monitor کلیک کرد و سپس از منوی Debug> Finish Debuging به آن خاتمه داد.

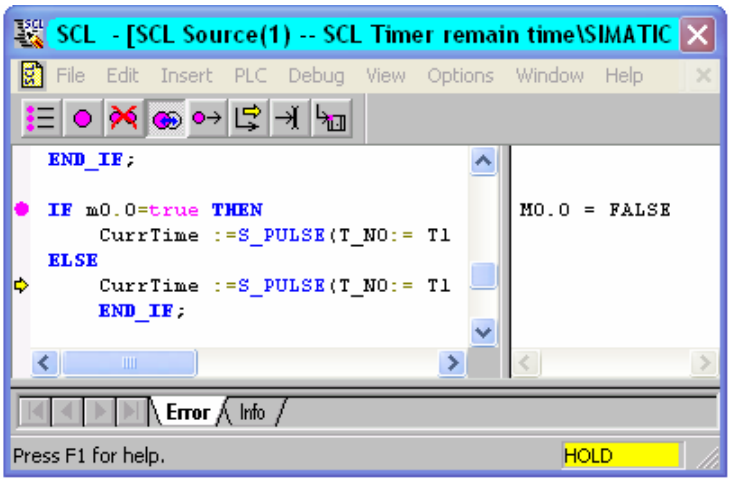


### استفاده از Breakpoint در برنامه SCL

با استفاده از Breakpoint میتوان CPU را در مد HOLD قرار داد و برنامه را خط به خط اجرا کرد اصول کار در اینجا مشابه آنچه برای برنامه STL در جلد اول کتاب ذکر شد می باشد با این تفاوت که در SCL امکان Breakpoint فقط برای S7-400 وجود دارد بعلاوه تعداد نقاط Breakpoint نیز در SCL محدودیت دارد بعنوان مثال برای CPU414 ماکزیمم ۴ نقطه Breakpoint میتوان قرار داد.

برای استفاده از BreakPoint پس از رعایت پیش نیازهای ذکر شده قبلی مراحل زیر را انجام دهید:

۱. ابزارهای Breakpoint را با استفاده از منوی View>Breakpoint Bar فعال کنید تا در بالای پنجره قابل دسترس باشند.
۲. با ماوس در اول سطری که میخواهید برنامه در آنجا متوقف شود کلیک کنید.
۳. منوی Debug> Set Breakpoint را انتخاب کرده مشاهده می شود که دایره قرمز رنگی در سمت چپ سطر مورد نظر ظاهر میشود . اینکار با استفاده از آیکون بالای پنجره نیز امکان پذیر است.
۴. تمام نکات و ملاحظات ایمنی را در نظر داشته باشید.
۵. با منوی Debug> Breakpoint Active یا با استفاده از آیکون به همین نام که در بالای پنجره است Breakpoint را فعال کنید. مشاهده میشود CPU به مد HOLD وارد میشود.
۶. با استفاده از Next Statement میتوانید سطر به سطر برنامه را اجرا کنید.
۷. برای برگشت به مد عادی ابتدا Delete All Breakpoint را انتخاب و سپس روی Resume کلیک کنید.



# فصل چهاردهم - برنامه نویسی توسط S7-Graph

**مشتمل بر :**

**۱۴-۱ مقدمه**

**۱۴-۲ آشنایی با محیط نرم افزار S7-Graph**

**۱۴-۳ شروع کار با S7-Graph**

**۱۴-٤ برنامه نویسی در S7-Graph**

۱۴-۴-۱ برنامه نویسی Action

۱۴-۴-۲ برنامه نویسی Condinion

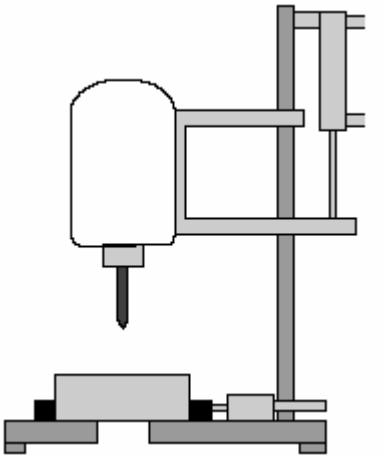
**۱۴-۵ پارمترهای FB در S7_Graph**

**۱۴-٦ ارائه چند مثال با S7-Graph**

**۱۴-۷ لیست دستورات در S7-Graph**

### ۱۴-۱ مقدمه

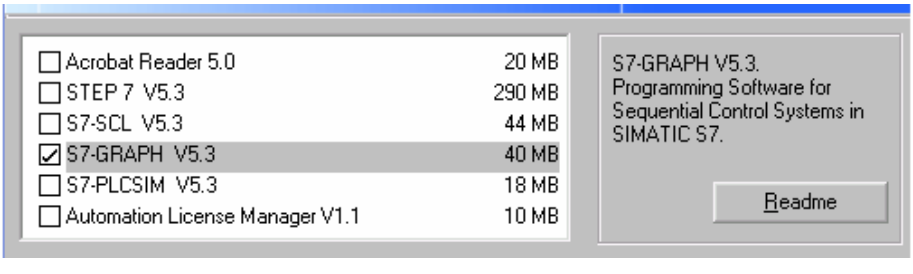
تا اینجا با ۴ زبان برنامه نویسی PLC آشنا شدیم . طبق استاندارد IEC1131-3 زبان برنامه نویسی دیگری که برای PLC ها بکار میرود SFC نام دارد ( مخفف Sequential Function Control ) که بصورت گرافیکی بوده و بیشتر برای کنترل ترتیبی بکار میرود. منظور از کنترل ترتیبی مواردی است که کنترلر لازم است اجزای سیستم تحت کنترل را به ترتیبی منطقی روشن یا خاموش نماید. بعنوان مثال در ماشین ابزار اتوماتیک یا کنترل نوار نقاله ها روش کار بصورت ترتیبی است. شکل زیر یک دریل برقی را همراه با قطعه کار که قرار است روی آن سوراخ کاری انجام شود نشان میدهد . ترتیب کار به اینصورت است که :



۱- سیستم روشن میشود.
۲- قطعه کلمپ میشود.
۳- موتور پمپ خنک کاری بکار می افتد.
۴- دریل پایین می آید تا به نقطه نهایی که با لیمیت سوئیچ مشخص شده برسد.
۵- نیم ثانیه در نقطه نهایی می ماند.
۶- دریل به بالا بر می گردد تا به نقطه اولیه که با لیمیت سوئیچ دیگری مشخص شده برسد.
۷- کلمپ باز میشود موتور دریل و موتور پمپ خنک کاری از کار می افتد.

در این توالی تا مرحله قبلی انجام نشده نبایستی مرحله بعد آغاز شود. این نمونه ای از کنترل ترتیبی است.

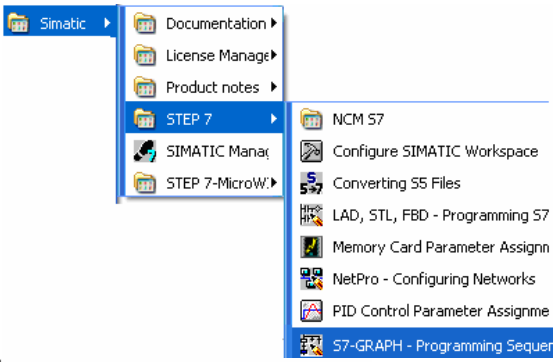
زیمنس برای کنترل ترتیبی و طبق استاندارد IEC1131-3 دو نرم افزار ارائه نموده است یکی S7-Graph و دیگری S7-Higraph . نرم افزار S7-Graph هم اکنون با نسخه Proffessional برنامه Step 7 عرضه میشود و کاربر در هنگام نصب Step 7 میتواند مانند شکل زیر آنرا انتخاب نماید ولی S7 –Higraph بصورت جداگانه ارائه میگردد. ما در این قسمت صرفاً به تشریح S7-Graph می پردازیم.



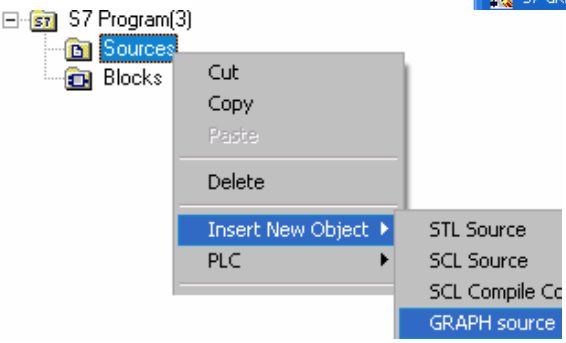
در استفاده از S7-Graph قبل از هر چیز بایستی توجه داشت که در بین بلاک های برنامه نویسی فقط FB را میتوان به این روش برنامه نویسی کرد پس این روش برای بلاک های OB و FC قابل کاربرد نیست.

S7-Graph به سه طریق زیر قابل اجرا و استفاده است:

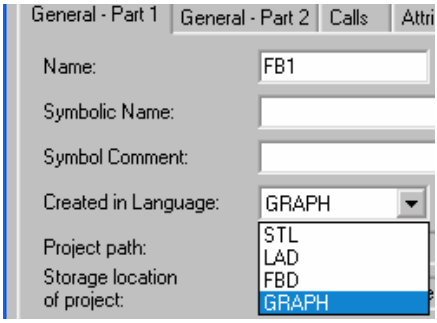
۱- از طریق Windows ومسیر **Start> Siamtic > step7** مطابق شکل زیر:




۲- از طریق ایجاد فایل Source مطابق شکل روبرو و باز کردن این فایل



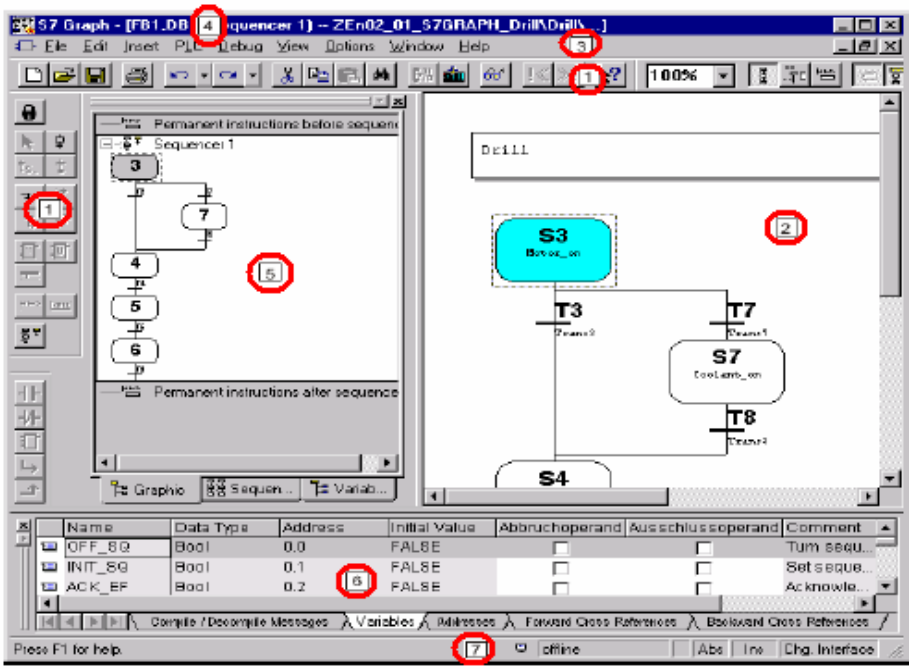

۳- از طریق ایجاد یک FB در پوشه Blocks و انتخاب زبان Graph مانند شکل روبرو سپس باز کردن آن



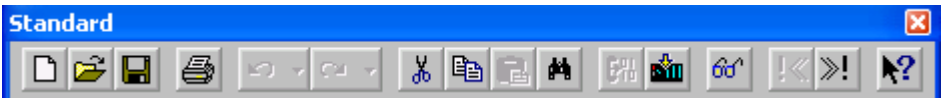

دربین روشهای فوق روش ۳ مناسب تر است و کاربر با سهولت بیشتر میتواند سایر کارهای مرتبط با پروژه را انجام دهد. در هر صورت با انتخاب هر کدام از روشهای مزبور برنامه S7-Graph اجرا میشود که در صفحات بعد تشریح شده است.

## ۱۴-۲ آشنایی با محیط نرم افزار S7-Graph

پس از اجرای برنامه S7-Graph به یکی از سه طریق ذکر شده پنجره برنامه به شکلی شبیه زیر ظاهر میشود. برای توصیف بهتر بخش های این برنامه شماره گذاری شده اند.برنامه در بخش شماره ۲ ترسیم میشود.

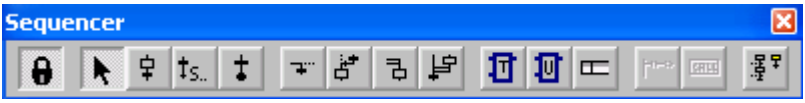



**بخش ۱** : این بخش که در بالا و نیز سمت چپ پنجره ظاهر میشود همان Toolbars برنامه است. بخش بالا به دو قسمت تقسیم میشود یکی قسمت Standard که ابزاری مانند شکل زیر است و خواننده محترم با بسیاری از کلیدهای آن آشناست :




قسمت دوم View نام دارد و همانطور که از اسمش مشخص است برای تغییر نحوه مشاهده محیط برنامه یعنی بخش ۲ بکار میرود فانکشن کلیدهای Zoom در آن مشخص است و سایر کلید ها در خلال بحث های آینده معرفی خواهند شد.

قسمت سوم Sequencer نام دارد که معمولاً در سمت چپ پنجره برنامه ظاهر میشود و دارای المانهایی بصورت زیر است. این المانها در محیط اصلی برنامه یعنی بخش شماره ۲ بکار میروند و بعداً تشریح میشوند.




قسمت چهارم LAD/FBD نام دارد که معمولاً در سمت چپ در زیر قسمت سوم فوق الذکر ظاهر میشود.از این المانها نیز در محیط اصلی برنامه یعنی بخش ۲ استفاده میشود. بصورت پیش فرض المانها بصورت FBD هستند ولی از طریق منوی View میتوان نوع آن را به LAD تغییر داد. در هر صورت المانها شبیه یکی از دو شکل زیر ظاهر خواهند شد:

**بخش ۲**: برنامه کنترل ترتیبی در این قسمت ترسیم و نوشته میشود. وقتی برای بار اول برنامه Graph را باز میکنیم معمولاً شامل یک مستطیل به نام Step 1 که با S1 نیز نشان داده میشود و یک خط در زیر آن به نام Transition 1 که با T1 نیز نشان داده میشود میباشد.

**بخش ۳** : این بخش Menu Bar برنامه است برخی از منوها مانند Edit , Window نیاز به توضیح ندارند برخی مانند منوی Insert همان امکانات بخش ۱ را در اختیار میگذارند این امکانات همراه با گزینه های سایر منوها در حین بحث ذکر خواهند شد.

**بخش ٤** : این بخش Title Bar برنامه است نام پروژه، شماره FB و DB و اطلاعاتی از این قبیل در این بخش نشان داده می شود.

**بخش ٥** : به این پنجره Overview گفته می شود و در آن میتوان وضعیت کل Step ها که تشکیل یک Sequencer یا توالی را میدهند مشاهده کرد. این پنجره معمولاً در سمت چپ ظاهر میشود ولی میتوان آن را از طریق آیکون که از اجزای بخش ۱ می باشد جابجا یا مخفی کرد.

**بخش ٦** : به این پنجره Detail گفته می شود که در پایین برنامه ظاهر میشود و در آن میتوان وضعیت متغیرها یا آدرسهای استفاده شده را دید . در هنگام ذخیره سازی برنامه لیست Error ها و Warning ها در این قسمت نمایش داده می شود.

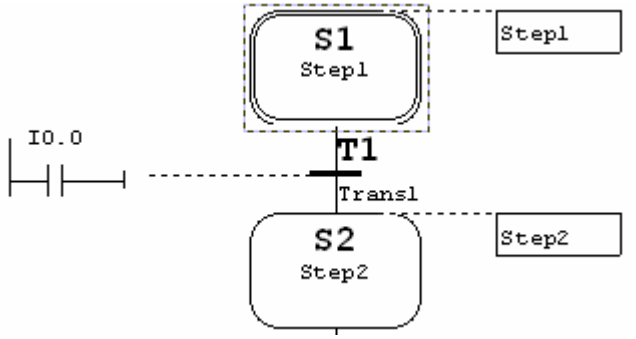
**بخش ۷** : به این قسمت Status Bar گفته می شود وضعیت CPU در حالت Online در این قسمت با آیکونهایی طبق جدول صفحه بعد ظاهر می شود.

| Symbol | Display | Description | With | Statusرنگ |
|---|---|---|---|---|
| | offline | You are editing a block stored on the PG/PC. | S7-300 and S7-400 | خاکستری |
| | DISCONNECTE D | The PLC connection is offline. | S7-300 and S7-400 | سفید |
| | CONNECTED | You are editing a block that is located on a PLC (SIMATIC S7). | S7-300 | سبز |
| | RUN/RUN-P | Module status | S7-400 | سبز |
| | STOP | Module status | S7-400 | قرمز |
| | HALT | Module status | | زرد |
| | FRCE | A force job is busy. | S7-400 | زرد |

## ۱۴-۳ شروع کار با S7-Graph

از سه روشی که برای ایجاد برنامه Graph ذکر شد بطور معمول برنامه Graph یا توسط FB یا بصورت Source File ایجاد می شود. تفاوتی که بین این دو وجود دارد اینست که FB را در صورتی می توان ذخیره کرد که فاقد Error باشد چون قبل از ذخیره سازی کامپایل میشود ولی Source اگر دارای Error هم باشد قابل ذخیره سازی است توسط Source می توان برنامه را نوشت و ذخیره کرد در فرصت مناسب دیگر آنرا کامپایل و Error ها را برطرف نمود.

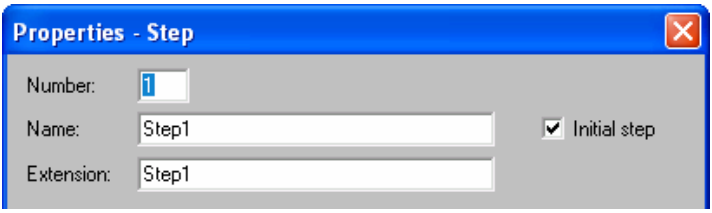
پس از اینکه برنامه S7-Graph به یکی از روشهای فوق باز شد در محیط اصلی آن شکل یک Step و یک Transition نمایش داده میشود. مفهوم Step که بصورت یک باکس مستطیل شکل نمایش داده می شود مرحله ای است که PLC به آن وارد میشود و در آنجا دستور یا دستورات خاصی را اجرا میکند. CPU آنقدر در این مرحله می ماند تا شرایط گذر یا Transition که در زیر Step با یک خط افقی نمایش داده شده بر آورده شود.بعنوان مثال در شکل زیر در صورتی CPU از مرحله ۱ عبور کرده و وارد مرحله ۲ می شود که سوئیچ I0.0 بسته شود.

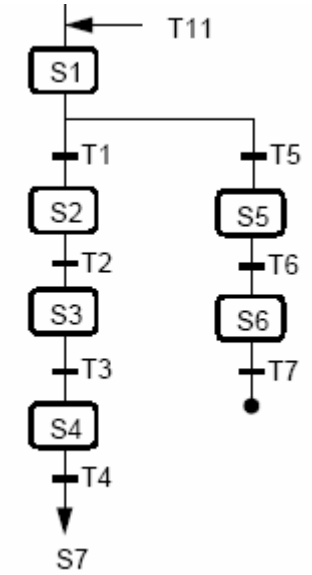



در یک برنامه کاربردی تعداد Step ها و Transsitionها معمولاً زیاد است با استفاده از Toolbar سمت چپ یا منوی Insert میتوان به دفعات و به تعداد دلخواه Step و .Trans اضافه کرد برای این کار با ماوس روی آخرین Trans موجود در برنامه کلیک کرده سپس باآیکون یا از منوی Insert>Step+Transition یا کلید Ctrl+1 آنها را اضافه می نماییم. نکات مهمی که بایستی به آنها توجه داشت:

- وقتی گفته میشود CPU در Step فعلی می ماند تا Trans بعد از آن برآورده شود منظور این نیست که برنامه در لوپ قرار گرفته است همانطور که بعداً معادل STL این برنامه را خواهیم دید در واقع وقتی یک Step فعال است مرتباً در هر سیکل اسکن از روی Step های بعدی پرش انجام میشود.

- به مجموعه ای از Step ها و Trans هایی که بدنبال هم قرار گرفته اند یک توالی یا Sequencer گفته میشود . در بین Step های یک توالی یک Step بایستی اولیه (Initial) باشد. با روشن شدن PLC پس از انجام راه اندازی وقتی وارد FB میشود همیشه بسراغ Step اولیه می رود و دستورات آنرا اجرا می نماید. بطور پیش فرض همیشه Step1 اولیه است ولی میتوان Step دیگری را بعنوان اولیه تعریف کرد .با کلیک راست روی Step و انتخاب Object Properties پنجره ای مانند شکل زیر نمایش داده میشود اگر Initial Step در این پنجره فعال شده باشد بعنوان Step اولیه تلقی میگردد. بایستی توجه داشت که اگر هیچکدام از Step ها اولیه نباشند برنامه FB که بصورت Graph نوشته شده اجرا نخواهد شد و اگر بیش از یک Step بعنوان اولیه تعریف شود در هنگام ذخیره سازی با پیغام خطای Too many initial step مواجه خواهیم شد.

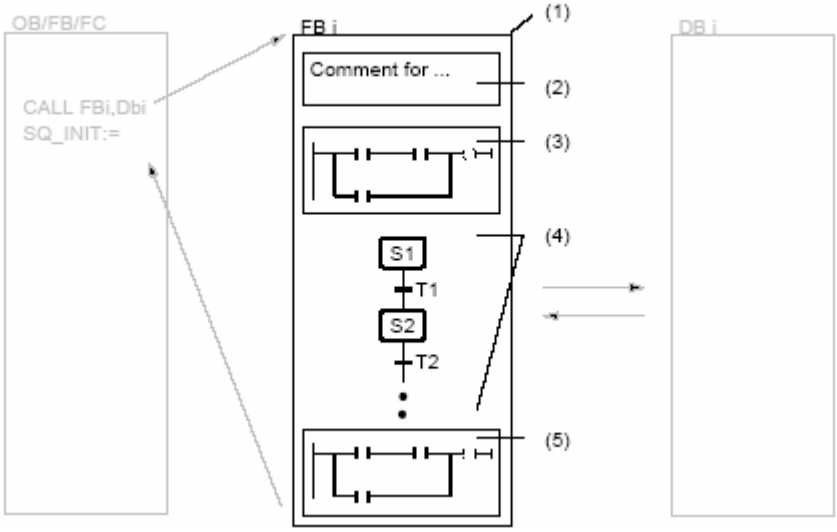



- در یک توالی میتوان شاخه های موازی ایجاد کرد .
- انتهای شاخه در یک توالی نمیتواند بلا تکلیف رها شود بایستی وضعیت ادامه پردازش در این نقطه مشخص گردد.همانطور که خواهیم دید میتوان انتهای شاخه را بست یا از آنجا به Step دیگری پرش نمود. شکل روبرو یک توالی را نشان میدهد که دارای دو شاخه موازیست انتهای یکی از شاخه ها بسته شده و از انتهای دیگری پرش انجام شده است.
- در یک برنامه Garaph میتوان بیش از یک توالی داشت CPU آنها را همزمان اجرا مینماید.
- Step میتواند فاقد دستور باشد همینطور Transition میتواند فاقد شرط باشد که در اینصورت CPU از آن بهStep بعدی عبور خواهد کرد.

## شکل کلی یک برنامه Graph

درشکل زیر شماتیک کلی یک برنامه Graph نشان داده شده است:




با توجه به شکل فوق نکاتی که قابل توجه میباشند عبارتند از :

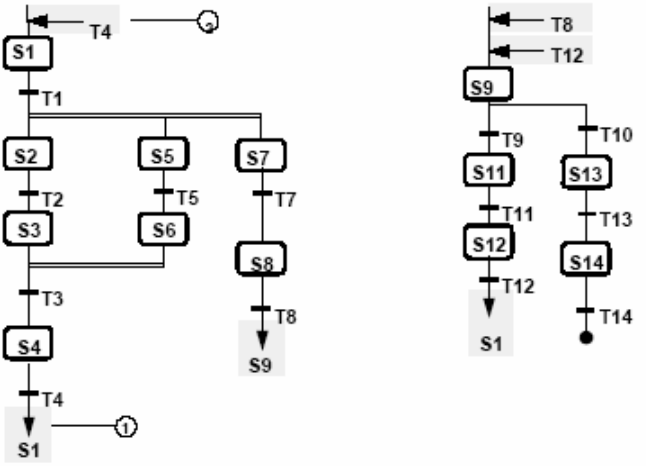
- FB حاوی برنامه Graph را میتوان از هر کد بلاک دیگر مانند OB یا FC یا FB دیگر صدا زد. بدیهی است در هنگام صدا زدن نوشتن نام یک DB لازم است.
- ساختار یک برنامه FB مانند شکل شامل چند قسمت است . بخش 2 برای نوشتن Comment یا توضیحات برنامه است. بخش 3 و 5 که Permanent Instructions نام دارد برای نوشتن دستورات دائمی است این دستورات در تمامی Step ها اجرا میشوند در حالیکه دستورات Step فقط در همان Step اجرا می گردند.دستورات دائمی که در بخش 3 نوشته شوند قبل از توالی و دستورات دائمی که در بخش 5 نوشته شوند بعد از توالی اجرا می شوند. بخش 4 محلی است که در آن توالی یعنی برنامه اصلی ترسیم میگردد.
- در توالی Step و Transition معمولاً بصورت جفت بکار میروند . در یک برنامه Graph ماکزیمم 256 جفت از ایندو میتوان بکار برد خواه در یک توالی خواه در چندین توالی
- شماره Step و Transition اتوماتیک توسط برنامه داده میشود با اینهمه میتوان آنها را تغییر داد بایستی توجه داشت که در برنامه Graph شماره ها مهم نیستند بلکه ترتیب چیدن المانها مهم است.

**Jump و Stop**

همانطور که گفته شد انتهای هر توالی یا هر شاخه ای از توالی نباید بلاتکلیف رها شود وبه یکی از دوطریق زیر بایستی مشخص گردد:

۱. پرش یا Jump که پس از کلیک کردن روی Transition میتوان آیکون را از پنجره سمت چپ انتخاب یا از منوی Insert>Jump استفاده کرد تا فلشی در زیر خط Trans ظاهر شود سپس در کنار فلش آدرس Step مورد نظر را نوشت این Step میتواند هر Step در همان توالی یا Step از یک توالی دیگر باشد این دو حالت در شکل زیر نشان داده شده است. حتی میتوان به Step ماقبل Trans نیز پرش کرد اگر چه این کار معقول نمی باشد ولی از دیدگاه برنامه امکان پذیر است و برنامه در لوپ نیز قرار نمی گیرد. پس از مشخص کردن پرش در کنار Jump آدرس Step مقصد نوشته میشود و در کنار Step مقصد آدرس Transition مبدا اتوماتیک ظاهر میگردد.

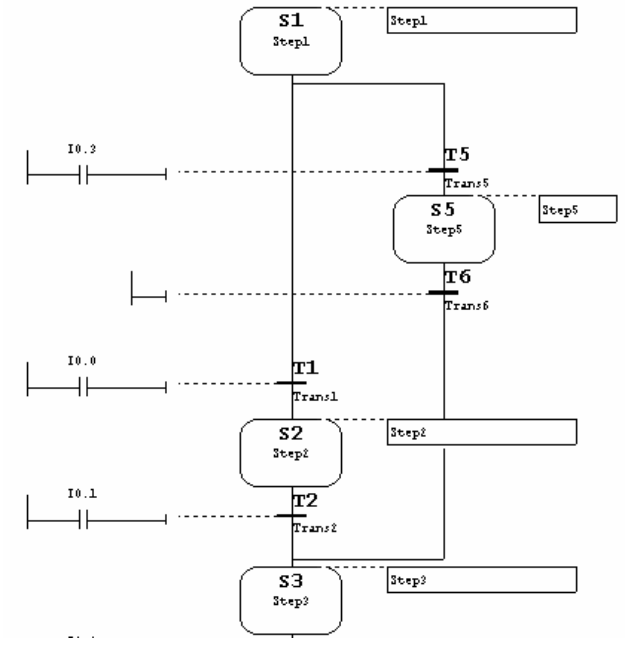
۲. توقف یا Stop که پس از کلیک کردن روی Transition میتوان آیکون را از پنجره سمت چپ انتخاب یا از منوی Insert>Branch Stop استفاده کرد تا یک دایره توپر در زیر خط Trans ظاهر شود مانند دایره بعد از T14 در شکل زیر. در اینحالت برنامه پس از عبور از Trans و رسیدن به این نقطه خاتمه می یابد توجه شود که منظور از خاتمه یافتن برنامه توقف پردازش CPU نیست کار CPU کماکان ادامه می یابد ولی عملاً هیچ Step دیگری اجرا نمیشود مانند اینکه در برنامه مدام از ابتدا به انتها پرش انجام شود.



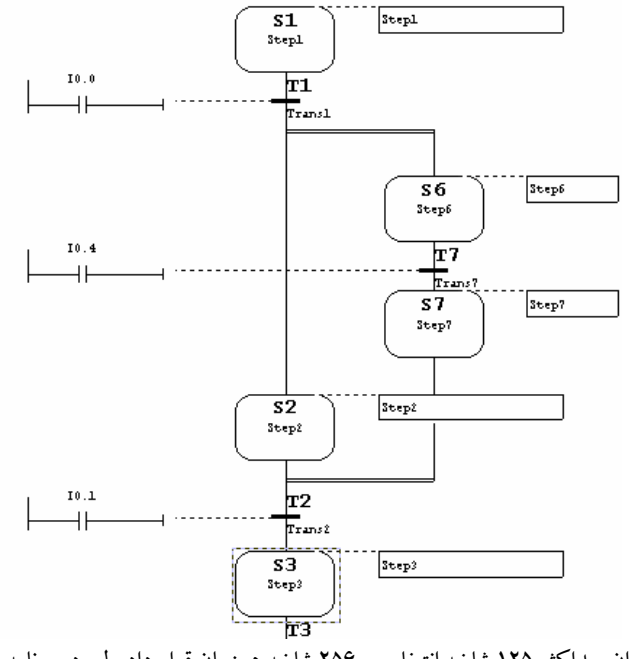
## شاخه های موازی

با استفاده از شاخه موازی میتون کنترل را به مسیرهای دیگر نیز هدایت کرد کلاً دو نوع شاخه موازی وجود دارد

۱. شاخه موازی انتخابی Alternative Branch : این شاخه با کلیک کردن روی Step و سپس استفاده از آیکون از پنجره سمت چپ یا از طریق منوی Insert قابل ایجاد است. پس از کلیک کردن روی این آیکون یک خط موازی همراه با یک Transition زیر Step ایجاد میشود مانند شکل زیر . میتوان بعد از این Transitoin مجدداً Step و Trans به شاخه موازی اضافه کرد . در شکل زیر برنامه در Step1 آنقدر میماند تا یکی از دو کلید I0.3 یا I0.0 فعال شود هر کدام زودتر فعال شد برنامه همان شاخه موازی را انتخاب میکند و Step های شاخه دیگر اجرا نمیشوند. اگر فرض شود که هر دو سوئیچ فوق همزمان در یک لحظه فعال شوند برنامه Step های هر دو شاخه را اجرا میکند. نکته دیگری که شایان توجه است انتهای مسیر موازی است که به سه صورت میتواند باشد یکی پرش به Step دیگر دوم توقف در آنجا و سوم امتداد تا زیر Step مربوط به شاخه موازی دیگر مانند شکل با استفاده از آیکون در این حالت برنامه پس از اجرای Step5 مستقیماً وارد Step3 میشود.



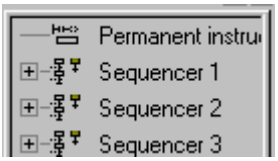
۲. شاخه موازی همزمان Simultaneus Branch : این شاخه با کلیک کردن روی Transision و سپس استفاده از آیکون از پنجره سمت چپ یا از طریق منوی Insert قابل ایجاد است. پس از کلیک کردن روی این آیکون یک خط موازی همراه با یک Step زیر Transition ایجاد میشود مانند شکل زیر . میتوان بعد از این Step مجدداً Trans و Step به شاخه موازی اضافه کرد . در شکل زیر برنامه در Step1 آنقدر میماند تا کلید I0.0 فعال شود سپس بطور همزمان وارد Step6 و Step2 میشود ولی تا کلید I0.4 زده و Step7 تکمیل نشود از T2 گذر نمیکند بعبارت دیگر شرط عبور از T2 آنست که اجرای هر دو شاخه موازی تکمیل شده باشد. شبیه نوع قبلی میتوان در اینحالت نیز شاخه موازی را بست یا به Step دیگر از آنجا پرش نمود اینکار لازم است بعد از Transition و نه بعد از Step انجام شود. روش سوم نیز ایجاد انشعاب به مسیر موازی دیگر با استفاده از آیکون مانند شکل زیر است:




در یک برنامه میتوان حداکثر ۱۲۵ شاخه انتخابی و ۲۵۶ شاخه همزمان قرار داد ولی در برنامه های کاربردی ماکزیمم ۲۰ یا ۳۰ شاخه کفایت میکند.

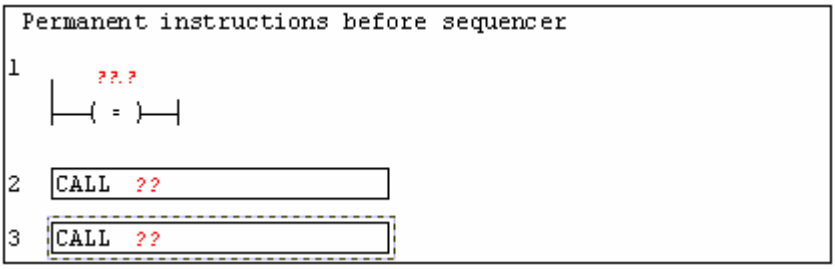
## ایجاد توالی جدید

در برنامه Graph میتوان همزمان چندین Sequencer داشت که مستقل از هم کار کنند و هر کدام Step های خاص خود را داشته باشند و همینطور هر کدام Step اولیه مجزایی دارا باشند. در عین حال میتوان از یک توالی به توالی دیگر پرش کرد. برای ایجاد یک توالی جدید میتوان از آیکون در پنجره سمت چپ یا از منوی Insert > Sequencer استفاده نمود. با اضافه کردن توالی جدید میتوان آنرا در پنجره Overview مانند شکل زیر مشاهده نمود و با کلیک روی توالی مورد نظر جزئیات Step و Trans آن را دید.

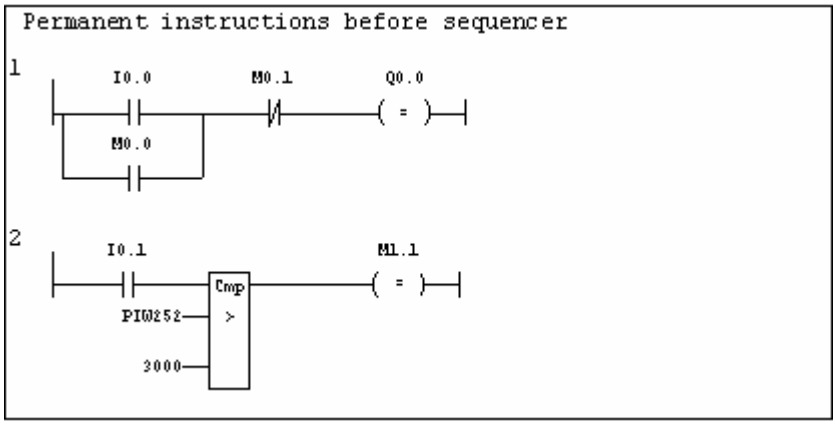



## دستورات دائمی

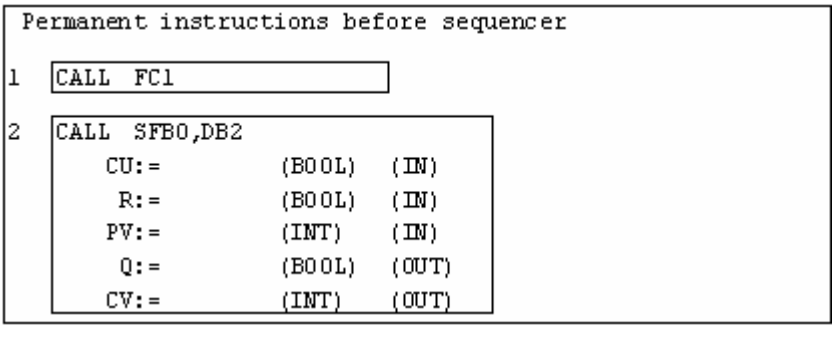
دستورات دائمی یا Permanent Instruction همانطور که ذکرشد دستوراتی هستند که در هر سیکل اسکن صرفنظر از اینکه برنامه در کدام Step قرار دارد یکبار اجرا میشوند دستورات بخش Permanent بالای بلاک در ابتدای سیکل اسکن و دستورات Permanent پایین بلاک در انتهای سیکل اجرا می گردند. با کلیک کردن روی این قسمت یک مستطیل خالی ظاهر میشود که میتوان آنرا برنامه ریزی کرد. برای برنامه نویسی در این قسمت میتوان از آیکونهای که در پنجره سمت چپ فعال میشوند یا از منوی Insert>Permanent استفاده کرد. یکی از این آیکون ها برای Condition بکار میرود و توسط آن میتوان شرایط مختلف را بصورت منطقی ترکیب نمود آیکون دیگر Block Call است که توسط آن میتوان یک FC یا FB را صدا زد. از این دو آیکون به دفعات می توان استفاده کرد و هر بار Condition یا Call جدیدی را اضافه کرد هرکدام از اینها برای خود شماره منحصر به فردی مانند شکل زیر دارند.



در قسمت Condition میتوان المانهای LAD یا FBD را اضافه کرد انتخاب نوع المان که LAD باشد یا FBD از پنجره View امکان پذیر است. صرفاً میتوان یکی از سه المان AND و OR و Comparator را استفاده کرد. ماکزیمم تعداد این المانها در کل یک بخش Permanent میتواند 32 عدد باشد.برای اضافه کردن المان مورد نظر روی خط Condition کلیک کرده سپس از پنجره LAD/FBD استفاده مینماییم. شکل زیر نمونه ای از برنامه نویسی Condition را نشان می دهد:

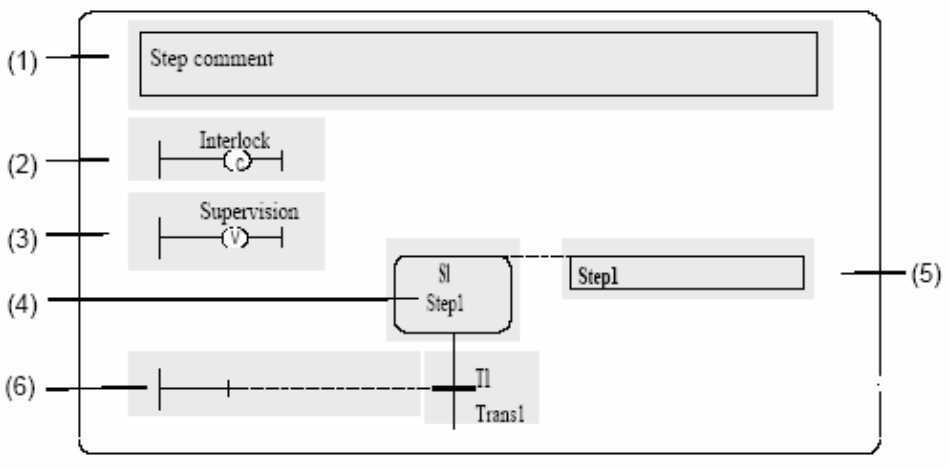



در قسمت CALL میتوان تمام FC ها و FB ها حتی اگر به زبانی غیر از Graph نوشته شده باشند را صدا زد بعلاوه SFC ها و SFB ها نیز از همین طریق قابل فراخوانی هستند.بدیهی است اگر فانکشن های فوق دارای ورودی و خروجی باشند لازم است به پارامترهای آنها مقدار یا آدرس اختصاص یابد.



### جزئیات هر Step

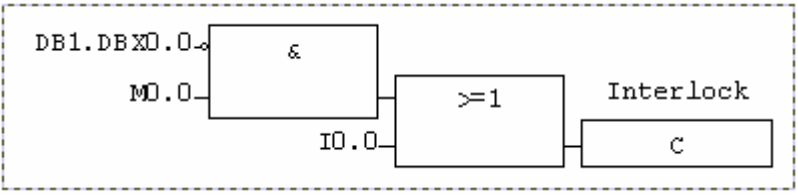
در پنجره Graph معمولاً ساختار توالی قابل مشاهده است برای مشاهده جزئیات هر Step روی آن Step کلیک کرده سپس از آیکون در بالای پنجره یا از منوی View > Single Step استفاده می نماییم. بدینطریق Step با جزئیات کامل مانند شکل زیر نمایش داده می شود. برای برگشت به حالت قبلی یعنی مشاهده کل توالی آیکون یا منوی View > Sequencer را بکار میبریم.



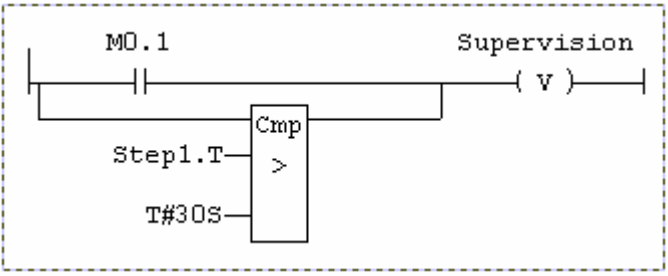

همانطور که مشاهده میشود هر Step بصورت منفرد دارای بخشهای زیر است :

**۱- بخش Comment** : در این بخش توضیحات مربوط به همان Step را میتوان نوشت و با بخش Block Comment که مربوط به توضیحات توالی است متفاوت است. بخش Comment Step را میتوان توسط آیکون از بالای پنجره یا بااستفاده از منوی View > Display With > Comment فعال یا غیر فعال نمود.

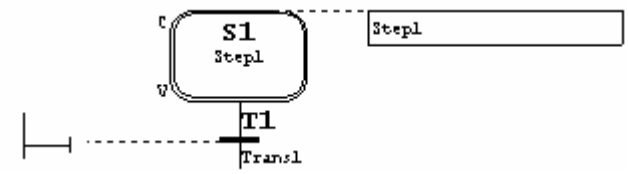
**۲- بخش Interlock** : در این قسمت شرایط Interlock مربوط به Step نوشته میشود. همانطور که بعداً شرح داده خواهد شد از شرایط Interlock میتوان برای دستورات برنامه نویسی Step استفاده کرد و متناسب با آن دستوراتی را اجرا کرد پس از این حیث با شرایط Transition متفاوت است زیرا Trans فقط شرایط گذر از Step را در بر میگیرد. شکل بعد نمونه ای از شرایط بخش Interlock را بصورت FBD نشان میدهد.




**۳- بخش Supervision** : در این قسمت میتوان یک Step را تحت نظارت قرار داد تا بعنوان مثال اگر زمان اجرای آن از حدی بیشتر شد یا کلید خاصی فعال شد آلارمی ظاهر شود. شرایط Supervision را مانند شرایط Interlock میتوان در دستورات برنامه نویسی Step بکار برد که در ادامه به آن خواهیم پرداخت. شکل زیر نمونه ای از شرایط ایجاد شده در Supervision را بصورت LAD نشان میدهد.




**تذکر ۱** : وقتی برای یک Step اینترلاک تعریف شود حرف C و وقتی برای آن Supervision تعریف شود حرف V در کنار آن مانند شکل زیر نشان داده میشود . این حروف در حالت عادی که Step فاقد موارد فوق است وجود ندارند.




**تذکر ۲:** تمام آدرس هایی که در بخش های Interlock و Supervision و Transition و سایر قسمتها بکار میرود در پنجره پایین برنامه در قسمت Addresses قابل مشاهده است شکل بعد :

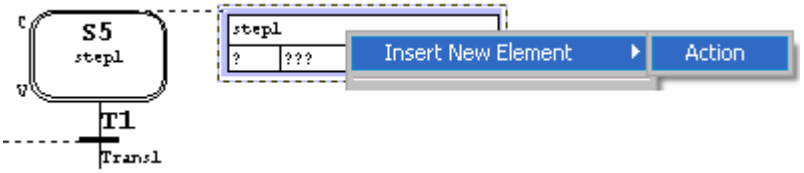
| Address /... | Symbol | Type | Comment |
|---|---|---|---|
| S001.T | start.T | TIME | |
| M0.1 | | BOOL | |
| M0.0 | | BOOL | |
| I0.0 | "start" | BOOL | |
| DB1.DBX0.0 | | BOOL | |

Compile / Decompile Messages | Variables | Addresses

**تذکر ۳** : آدرسها را میتوان بصورت سمبلیک یا مطلق مشاهده کرد اینکار توسط آیکون یا با استفاده از منوی View > Diplay with>symbols انجام میشود.

**٤- بخش Step** : در این قسمت شماره و نام Step مشخص است . شماره در بالا و نام در زیر آن نوشته میشود. نام را میتوان با کلیک کردن روی آن به هر اسم دلخواهی که معرف عملکرد Step باشد تغییر داد. همینطور با کلیک کردن روی شماره میتوان آنرا عدد دلخواه دیگری گذاشت. مثلا S1 را به S6 تغییر داد. لازم است توجه شود که در یک توالی شماره Step ها و Transition ها معرف ترتیب اجرا نیست بلکه ترتیب اجرا بر اساس آرایش انجام شده بین المانهای Graph انجام میگیرد.

**٥ – بخش دستورات Step** : در این قسمت دستوراتی که قرار است اجرا شوند نوشته میشوند . با کلیک راست روی این قسمت و انتخاب Insert New Element > Action مانند شکل زیر خواهیم دید که یک سطر با دو ستون زیر Step ظاهر میشود . به همین روش به تعداد دلخواه میتوان سطر اضافه کرد . هر سطر یک عملیات خاص را انجام میدهد. نحوه نوشتن دستور در این سطر ها بعداً خواهد آمد.




**٦- بخش Transition** : در این قسمت شماره و نام Trans مشخص میشود شبیه Step می توان اینها را نیز تغییر داد.

## ۱۴-۴ برنامه نویسی در S7-Graph

بطور کلی برنامه نویسی در S7-Graph به دو دسته زیر تقسیم میشود:

۱. **برنامه نویسی Action** : این برنامه نویسی در بخش Step انجام میشود و عملیاتی که در آن Step لازم است اجرا شود را مشخص میکند . این دستورات مفصل بوده و فرمت آنها خاص S7-Graph است اگر چه بعضاً شباهتی با دستورات STL دارند.

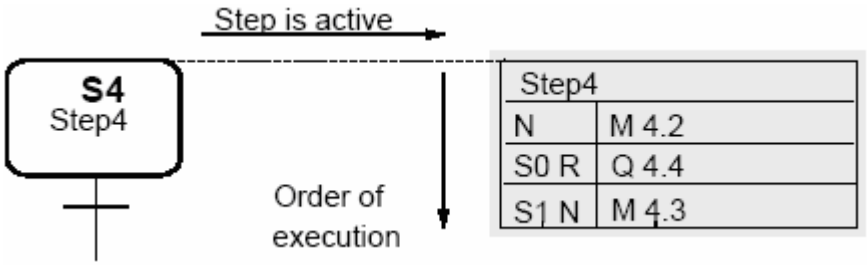
۲. **برنامه نویسی Condition** : این برنامه نویسی در بخش های Transtition و Permanent و Interlock و Supervision انجام میشود و عمدتاً حاوی شرایطی است که توسط المانهای LAD/FBD ایجاد می گردند.

در مشاهده یک توالی در محیط Graph میتوان باکس ها و المانهای مربوط به برنامه نویسی را از نظر مشاهده فعال یا غیر فعال نمود این کار با آیکون یا از طریق منوی **View>Display with>Condition and Action** انجام پذیر است.

در این بخش ابتدا دستورات Action را مورد بحث قرار داده و در انتها به تشریح دستورات Condition می پردازیم.

### ۱۴-۴-۱ برنامه نویسی Action

در قسمت قبل روش اضافه کردن سطرهای Action به بخش دستورات Step ذکر گردید . با اضافه کردن سطرهای مورد نظر دستورات را در ستونهای هر سطر مینویسم. هر سطر معرف یک نوع عملیات است . در ستون سمت چپ، دستور و در ستون سمت راست ،آدرس نوشته میشود . شکل زیر نمونه ای از این دستورات را نشان میدهد. وقتی Step فعال گردید دستورات بترتیب از بالا به پایین اجرا می گردند.نوشتن دستور در Step الزامی نیست . میتوان آنرا خالی گذاشت این موضوع در هنگام کامپایل صرفاً منجر به Warning میشود.



دستورات بخش Action را می توان به زیر مجموعه های زیر تقسیم کرد :

۱. دستورات استاندارد
۲. دستورات مبتنی بر Event
۳. کانترها
۴. تایمرها
۵. فانکشن های محاسباتی

## ۱- دستورات Action استاندارد

این دستورات در جدول زیر لیست شده اند و در ادامه تشریح خواهند شد. این دستورات به دو حالت اجرا میشوند. حالت اول بدون اینترلاک یعنی بدون نظر گرفتن اینترلاک در اینجا دستور به تنهایی نوشته میشود مانند N یا S و ... حالت دوم با اینترلاک که در این حالت اجرای دستورات مشروط به برآورده شدن شرایط اینترلاک خواهد بود. در اینجا ذکر کلمه C بعد از دستور لازم است. در جدول زیر [ ] حالت Optional را برای اینترلاک نشان می دهد.

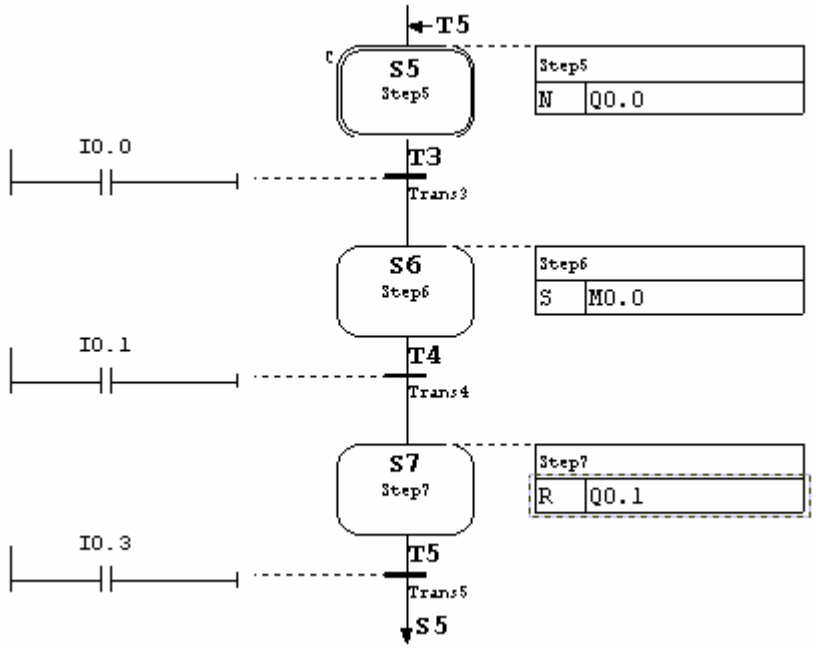
| دستور | آدرس | مفهوم دستور |
|---|---|---|
| N [C] | Q,I,M,D | وقتی Step فعال شد آدرس ذکر شده یک میشود با عبور از Step این آدرس به صفر بر میگردد. |
| S [C] | Q,I,M,D | وقتی Step فعال شد آدرس ذکر شده یک میشود با عبور از Step این آدرس یک باقی می ماند . (حالت Latching) |
| R [C] | Q,I,M,D | وقتی Step فعال شد آدرس ذکر شده صفر میشود با عبور از Step این آدرس صفر باقی می ماند . |
| D [C] | Q,I,M,D<br>T#<const> | n ثانیه بعد از فعال شدن Step آدرس ذکر شده یک می شود و تا زمانی که Step فعال است یک باقی می ماند( شبیه یک تایمر تاخیر در وصل) . اگر n ثانیه کوتاهتر از زمان فعال بودن Step باشد این دستور عمل نمی کند. زمان با فرمت #T در سطر بعدی این دستور نوشته میشود |
| L [C] | Q,I,M,D<br>T#<const> | وقتی Step فعال است به اندازه n ثانیه آدرس ذکر شده یک و پس از آن صفر میشود ( شبیه یک تایمر پالس) . زمان با فرمت #T در سطر بعدی این دستور نوشته میشود |
| CALL [C] | FB, FC, SFB, SFC | در طول مدتی که Step فعال است بلاک ذکر شده صدا زده میشود |

لازمست ذکر شود که اگر حرف C در یک Step که دارای اینترلاک نیست بعد از دستور استفاده شود مشکلی پیش نخواهد آمد و فقط یک Warning در هنگام کامپایل ظاهر خواهد شد.

در صفحات بعد برای دستورات فوق مثالهایی ذکر شده است.

**مثال ۱ :**

در مثال شکل زیر در هیچکدام از Step ها اینترلاک بکار نرفته است . S5 بعنوان Initial میباشد . با روشن شدن PLC برنامه وارد Step5 میگردد و بلافاصله خروجی Q0.0 روشن می گردد . برنامه در این Step آنقدر می ماند تا شستی I0.0 فعال شود پس از آن وارد Step6 میگردد. با عبور از Step5 خروجی Q0,0 خاموش میشود و با ورود به Step6 فلگ M0.0 یک میشود اگر در این مرحله شستی I0.1 فعال شود با عبور از S6 فلگ M0.0 یک باقی میماند و تا ری ست نشود در Step های بعدی نیز یک بودنش حفظ میشود. با ورود به S7 خروجی Q0.1 که فرض شده قبلاً یک بوده ری ست میگردد. برنامه با فعال شدن شستی I0.3 مجدداً به S5 بر میگردد.




اگر برنامه فوق بصورت Source نباشدو در FB نوشته شود با ذخیره سازی برنامه فوق عمل کامپایل نیز انجام میشود و خواهیم دید که فاقد Error و Warning است. اکنون اگر FB را توسط برنامه LAD/STL/FBD باز کنیم برنامه ای مانند صفحه بعد خواهیم دید. کاربرد زیاد دستور jump در این برنامه نشان میدهد که چگونه وقتی یک Step فعال است برنامه بدون قید و شرط از روی سایر Step ها پرش می کند.

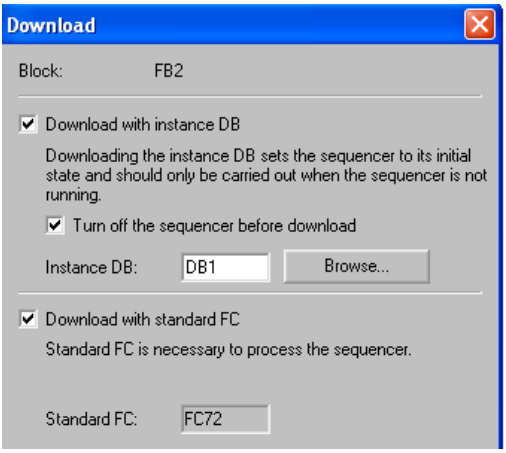
```
NETWORK 1
    L    DIB  104;
    L    B#16#0;
    T    DIB  104;
    TAK  ;
    L    DIB  105;
    TAK  ;
    JL   G7BE;
    JU   USER;
    JU   G7BE;
    JU   T;
    JU   G7BE;
    JU   G7BE;
    JU   M1;
    JU   G7BE;
    JU   M2;
G7BE: BE   ;
USER: LAR1  P#254.0;
    L    W#16#1A6;
    UC   "G7_STD_3";
    BE   ;
T:   TAK  ;
    JL   M1;
    JU   G7BE;
    JU   T1;
    JU   T2;
    JU   T3;
M1:  TAK  ;
    JL   M2;
    JU   G7BE;
    JU   S1;
    JU   S2;
    JU   S3;
M2:  L    5;
    T    #G7S[1].SNO;
    L    6;
    T    #G7S[2].SNO;
    L    7;
    T    #G7S[3].SNO;
    L    3;
    T    #G7T[1].TNO;
    L    4;
    T    #G7T[2].TNO;
    L    5;
    T    #G7T[3].TNO;
    BE   ;
NETWORK 2
    CLR  ; // G7T_0 0000
NETWORK 3
T1:  NOP  0; // Trans3 0003
    A    I0.0;
    =    #CRIT[0];
    NOP  1;
    A    #CRIT[0];
    BE   ;
NETWORK 4
T2:  NOP  0; // Trans4 0004
    A    I    0.1;
    =    #CRIT[0];
    NOP  1;
    A    #CRIT[0];
    BE   ;
NETWORK 5
T3:  NOP  0; // Trans5 0005
    A    I    0.3;
    =    #CRIT[0];
    NOP  1;
    A    #CRIT[0];
    BE   ;
NETWORK 6
    CLR  ; // G7S_0 0000

NETWORK 7
S1:  A    #G7S[1].X; // step5 0005
    =    Q    0.0;
    BE   ;
NETWORK 8
S2:  A    #G7S[2].X; // Step6 0006
    S    M    0.0;
    BE   ;
NETWORK 9
S3:  A    #G7S[3].X; // Step7 0007
    R    Q    0.1;
    BE   ;
```

برای تست کردن برنامه S7-Graph قبلی توسط سیمولاتور یا PLC مراحل زیر را انجام می دهیم:

۱- FB را در S7-Graph ذخیره میکنیم و از عدم وجود Error در هنگام کامپایل مطمئن می گردیم.

۲- روی آیکون Downlod بالای برنامه کلیک کرده و مشاهده میکنیم که پنجره ای مانند شکل بعد ظاهر میشود . در این پنجره نام یک دیتا بلاک و نیز نام فانکشن FC72 که از فانکشن های زیمنس است آورده شده و در اولین دانلود هردو علامت خورده اند. در دانلود های بعدی معمولاً نیازی به

انتخاب DB و FC72 نمی باشد.بعلاوه بهتر است در هنگام دانلود مجدد PLC در حالت Stop باشد.



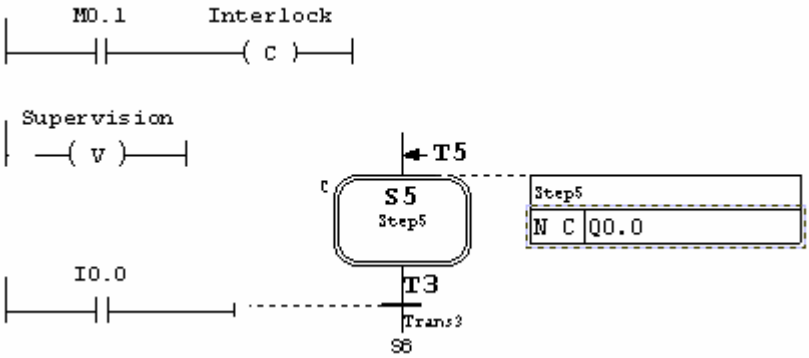

۳- OB1 را باز کرده و در آن FB را همراه با DB فوق الذکر صدا زده سپس OB1 را ذخیره و به **PLC** دانلود میکنیم. توجه شود که پارامترهایی که در زیر **FB** هنگام فراخوانی مشاهده میشوند می توانند خالی باشند و الزامی به دادن مقدار به آنها نیست . این پارامترها بعداً تشریح می گردند.

۴- PLC را روشن کرده و روی آیکون عینک شکل مانیتور در S7-Graph کلیک میکنیم مشاهده خواهیم کرد که Step5 با رنگ سبز روشن میشود که نشان دهنده فعال شدن این Step است .

۵- وضعیت خروجی Q0.0 را مشاهده کرد سپس با فشار دادن شستی های I0.0 و I0.1 و I0.3 بقیه مراحل برنامه را تست می نماییم.

**مثال ۲ :**

در برنامه قبل به یکی از Step ها مثلاً Step5 اینترلاک اختصاص میدهیم سپس در بخش دستورات Step بجای N دستور N C را بکار می بریم . مانند شکل زیر:

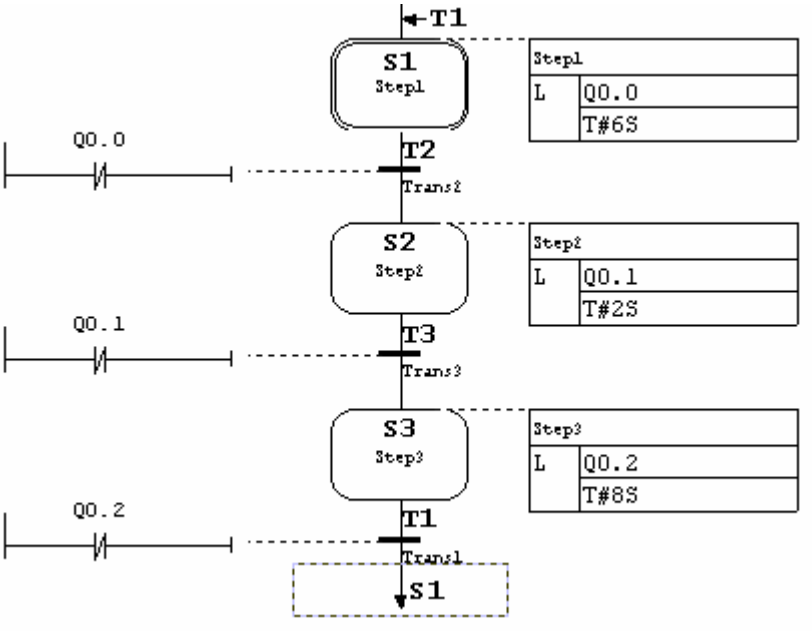


اگر این برنامه را ذخیره و به PLC دانلود و سپس مانیتور کنیم می بینیم که با روشن شدن PLC این Step فعال و با رنگ قرمز نمایش داده میشود که نشان دهنده وجود اینترلاک است. در این شرایط هنوز خروجی Q0.0 روشن نشده و در صورتی که شرایط اینترلاک بر آورده شود یعنی M0.1 یک شود Step با رنگ سبز فعال شده و خروجی مزبور یک می گردد.

**مثال ۳ :**

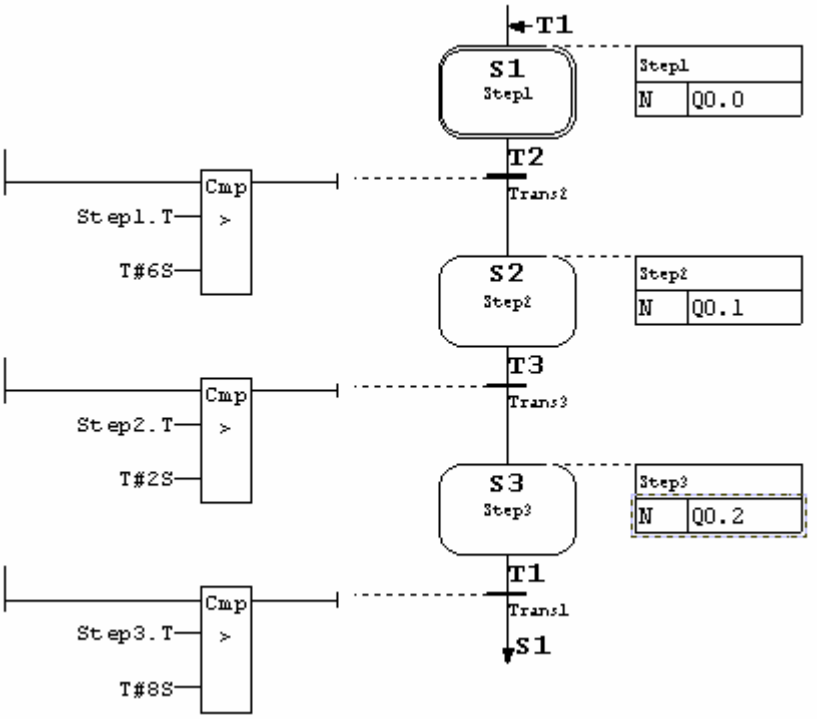
برنامه زیر نمونه ای از کاربرد دستور L را نشان می دهد. در این برنامه با ورود به Step1 خروجی Q0.0 روشن میگردد و به اندازه ۶ ثانیه روشن می ماند وپس از آن خاموش می گردد از آنجا که وضعیت NOT این خروجی بعنوان شرط در Transition 1 آورده شده بنابر این در طول زمان روشن بودن خروجی شرط برقرار نیست ولی به محض خاموش شدن آن شرط برآورده شده و به Step2 گذر میکند در Step2 نیز همین منطق حاکم است فقط خروجی Q0.1 در مدت ۲ ثانیه روشن وپس از آن خاموش میگردد . در Step 3 نیز خروجی Q0.2 در مدت ۸ ثانیه روشن سپس خاموش می گردد.

دستور L اگر چه مانند تایمر پالس عمل میکند ولی اگر معادل STL برنامه FB را ببینیم مشاهده خواهیم کرد که از تایمرهای معمول استفاده نشده و متغیر محلی از جنس Time برای زمان سنجی بکار رفته است.



**مثال ٤:**

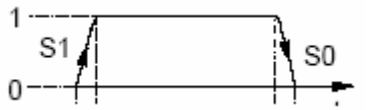
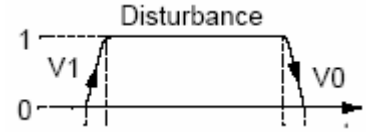
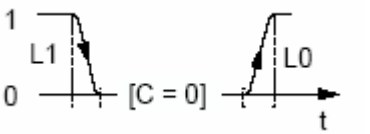
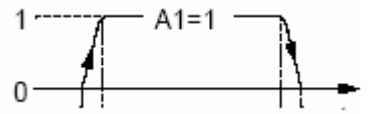
برنامه فوق را میتوان بصورت دیگری نیز نوشت بجای استفاده از زمان سنجی در Step این کار را در Transition انجام می دهیم . همانطور که در برنامه شکل زیر ملاحظه میشود در Step ها فقط دستور N بکار رفته و در Transition ها از یک مقایسه گر برای کنترل مدت زمان مکث Step استفاده شده است. ورودی مقایسه گر یکی مقدار زمان بافرمت #T است و ورودی دیگر زمان Step که بصورت S1.t و امثال آن بکار میرود . منظور از S1.t یا Step1.t مدت زمان فعال بودن Step است که در حالت online نیز در بالای Step نشان داده میشود.



### ۲- دستورات Action مبتنی بر Event

دستورات Action می تواند بطور منطقی با رخداد (Event) ترکیب شود . منظور از Event در اینجا وقوع شرایطی مانند ورودبه Step یا خروج از Step یا وقوع Interlock و Supervision و امثال آنست. بدیهی است دستور Action مورد نظر فقط در هنگام وقوع Event اجرا خواهد شد.

بطور کلی Event های مورد استفاده برای Action به ۴ دسته طبق جدول زیر تقسیم میشوند.

| نوع Event | فرمت | مفهوم | شکل |
|---|---|---|---|
| **Step** | S1 | ورود به Step ( فعال شدن) |   |
| | S0 | خروج از Step ( غیرفعال شدن) | |
| **Supervision** | V1 | وقوع خطای Supervision |   |
| | V0 | رفع خطای Supervision | |
| **Interlock** | L1 | رفع شرایط اینترلاک |   |
| | L0 | وقوع شرایط اینترلاک | |
| | C | برآورده شدن شرایط اینترلاک | |
| **Message and Registration** | A1 | پیغام Acknowledge شده |   |
| | R | Registeration فعال شده | |

در هر کدام از حالات فوق فرمت ذکر شده برای Event قبل از دستور Action به شرحی که در جداول صفحه بعد آمده ذکر میشود بعنوان مثال مفهوم Action زیر آنست که در هنگام خروج از Step خروجی Q0.0 را خاموش کن:

| | |
|---|---|
| S0 R | Q0.0 |

دستورات Action استاندارد بجز دستورات D و L را میتوان همراه با Event بکار برد جدول زیر این دستورات را همراه با Event های مربوطه نشان میدهد. همانطور که می بینیم دستورات Action در اینحالت نیز میتوانند همراه با حرف C ( برای اینترلاک) یا بدون آن صدا زده شوند و [ ] به معنای این انتخاب است.

| Event | Instruction | Address |
|---|---|---|
| S1, V1, A1, R1 | N[C], R[C], S[C] | Q,I,M,D |
| | CALL[C] | FB, FC, SFB, SFC |
| S0, V0, L0, L1 | N, R, S | Q,I,M,D |
| | CALL | FB, FC, SFB, SFC |

چند حالت را بصورت نمونه بررسی میکنیم :

| | |
|---|---|
| S 1 S | Q0.0 |

دستور روبرو باعث میشود تا صرفاً در هنگام ورود به Step خروجی Q0.0 ست شود . وقتی Event بکار نرود یعنی دستور فقط بصورت S Q0.0 نوشته شود در اینحالت فرمان ست کردن مرتباً تا زمانی که Step فعال است صادر میشود ولی در مثال روبرو این فرمان فقط یکبار صادر می شود.

| | |
|---|---|
| S 0 CALL | FC1 |

این دستور باعث میشود تا صرفاً در هنگام خروج از Step فانکشن FC1 صدا زده شود ( یعنی فقط یکبار ) . ولی دستور CALL FC1 که بدون Event نوشته شود مرتباً تا زمانی که Step فعال است فانکشن را صدا میزند

| | |
|---|---|
| L0 R | Q0.0 |

این دستور باعث میشود تا صرفاً وقتی شرایط اینترلاک برآورده شد خروجی Q0.0 ری ست شود . وقتی Event بکار نرود یعنی دستور فقط بصورت R Q0.0 نوشته شود فرمان ری ست کردن مرتباً اعمال می شود.

| | |
|---|---|
| S1 R C | Q1.0 |

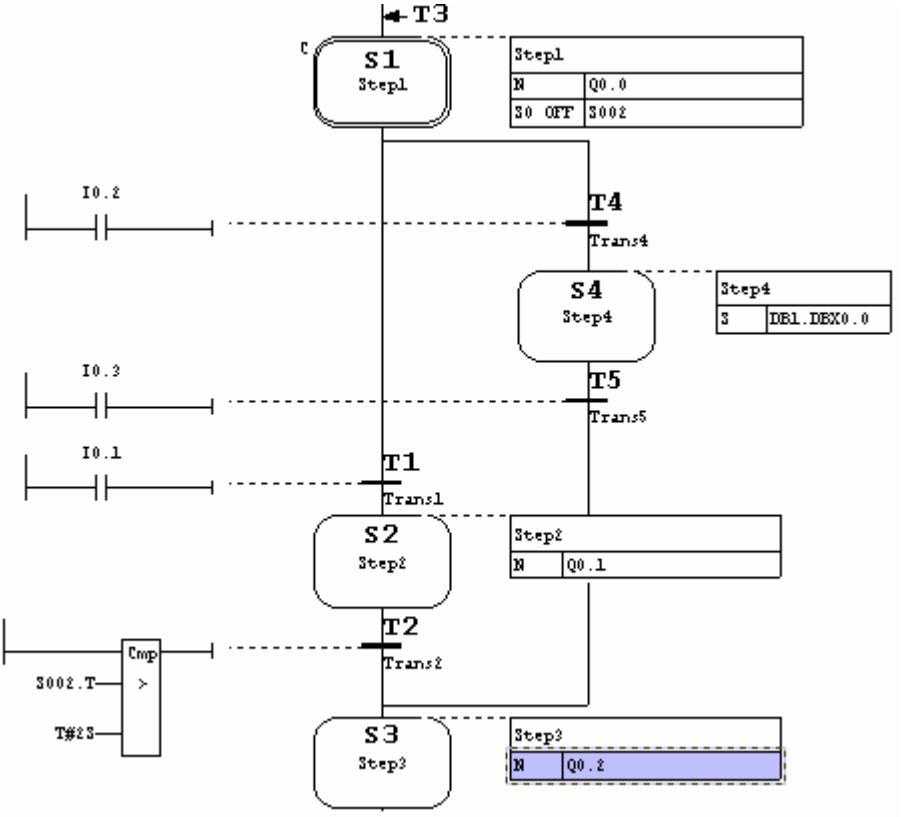
با این دستور Q1.0 وقتی که Step فعال شود و شرایط اینترلاک بر آورده گردد ری ست میشود .

در مورد خطای Supervision و استفاده از V0 و V1 در بخش دستورات Condition توضیح لازم خواهد آمد.

علاوه بر دستوراتی که گفته شد دستورات دیگری مبتنی بر Event وجود دارند که توسط آنها میتوان یک Step یا تمامی Step ها را در صورت وقوع Event فعال یا غیر فعال نمود . دستور OFF و ON که بعد از Event بکار میرود برای این منظور است. جدول بعد:

| Event | Instruction | Address Identifier |
|---|---|---|
| S1, V1, A1, R1 | ON[C], OFF[C] | S |
| S1, V1 | OFF[C] | S_ALL |
| S0, V0, L0, L1 | ON, OFF | S |
| L1 | OFF | S_ALL |

با نوشتن S_ALL در بخش Address تمامی Step ها را میتوان ON یا OFF کرد ولی در صورتی که یک Step خاص مورد نظر باشد لازم است بعد از حرف S شماره آن Step ذکر گردد.




در مثال فوق وقتی برنامه در Step1 قرار دارد وضعیت دو سوئیچ I0.1 و I0.2 را چک می کند با فعال شدن I0.2 برنامه از مسیر موازی Step4 و سپس Step3 را ادامه میدهد ولی از آنجا که درهنگام خروج از Step1 توسط دستور ذکر شده Step2 غیر فعال شده بنابراین با بازگشت به Step1 اگر کلید I0.1 فعال شود برنامه Step2 را اجرا نخواهد کرد.

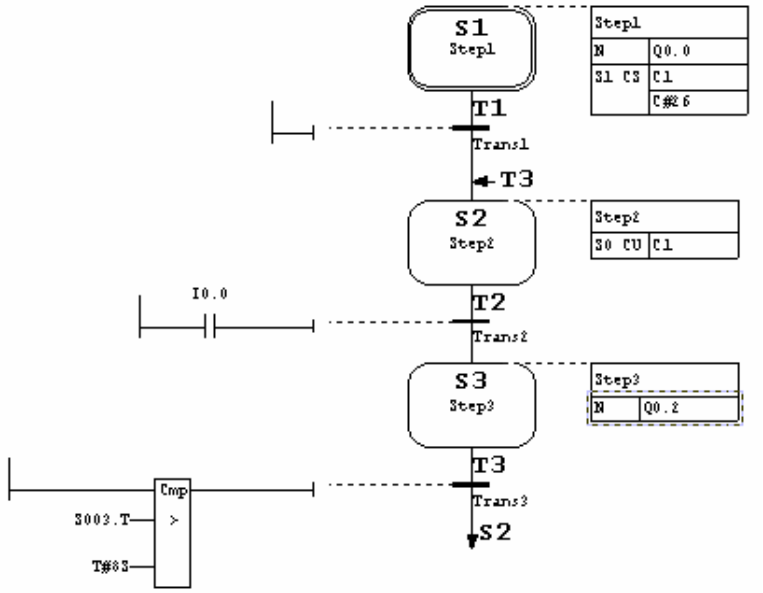
## ۳- دستورات کانترها

کانترها مبتنی بر Event هستند و طبق جدول زیر میتوان آنها را با Event های ذکر شده صدا زد . استفاده از اینترلاک نیز در آنها Optional است وبه همین دلیل حرف C داخل [ ] قرار گرفته است.

دستور CU برای افزایش کانتر و دستور CD برای کاهش کانتر است . برای ست کردن کانتر با مقدار اولیه دستور CS بکار میرود که در اینحالت مقدار اولیه بصورت BCD یعنی با فرمت C# داده میشود. برای ری ست کردن کانتر دستور CR بکار میرود.

| Event | Instruction | Address |
|---|---|---|
| S1, S0, L1, L0, V1, V0, A1, R1 | CS[C] | C<br><initial counter value> |
| S1, S0, L1, L0, V1, V0, A1, R1 | CU[C] | C |
| S1, S0, L1, L0, V1, V0, A1, R1 | CD[C] | C |
| S1, S0, L1, L0, V1, V0, A1, R1 | CR[C] | C |

در مثال زیر به محض روشن شدن سیستم و فعال شدن Step1 کانتر C1 با مقدار اولیه ۲۶ ست میشود . و بلافاصله وارد Step2 میگردد و منتظر میماند تا I0.0 فعال شود تا اینجا مقدار کانتر هنوز ۲۶ است . با فعال شدن I0.0 در هنگام خروج از Step2 مقدار کانتر یکی افزایش می یابد. در انتهای برنامه مجدداً به Step2 پرش انجام شده تا مجدداً براساس وضعیت I0.0 افزایش یابد اگر پرش به S1 بود مجددا کانتر با مقدار ۲۶ ست میگردید.



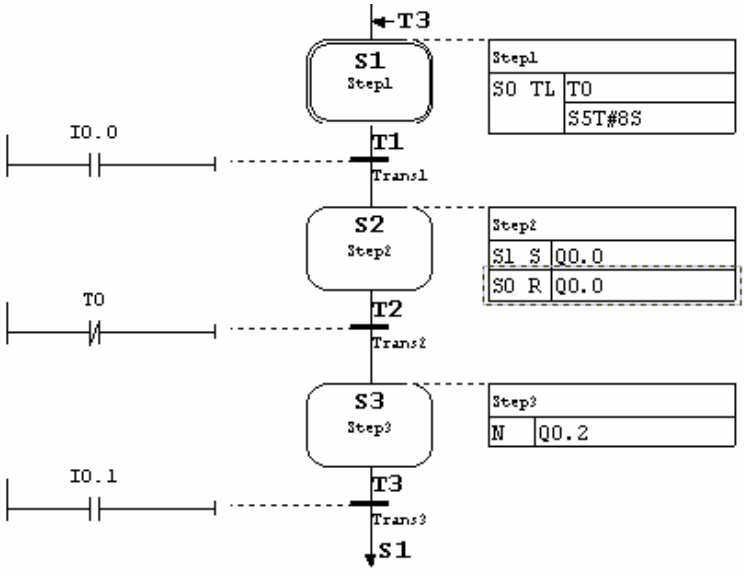
## ٤- دستورات تایمرها

تایمرها نیز مبتنی بر Event هستند و طبق جدول زیر میتوان آنها را با Event های ذکر شده صدا زد . استفاده از اینترلاک [C] نیز در آنها Optional است. در اینجا فقط دو نوع تایمر وجود دارد TL شبیه تایمر Extended Pulse و TD شبیه تایمر On-Delay عمل میکند. زمان در زیر تایمر با فرم S5time یعنی S5t# بکار میرود. دستور TR برای Reset کردن تایمر استفاده میگردد.

| Event | Instruction | Address |
|---|---|---|
| S1, S0, L1, L0, V1, V0, A1, R1 | TL[C] | T<br><Time> |
| S1, S0, L1, L0, V1, V0, A1, R1 | TD[C] | T<br><Time> |
| S1, S0, L1, L0, V1, V0, A1, R1 | TR[C] | T |

در بخش دستورات استاندارد دیدیم که دستورات L و D نیز شبیه تایمر عمل میکردند ولی دو دستور فوق مبتنی بر Event نبودند.

در مثال زیر تایمر T0 که بصورت TL است پس از اینکه I0.0 فعال شد در هنگام خروج از Step1 با زمان ۸ ثانیه فعال میشود و برنامه وارد Step2 می گردد. در هنگام ورود به Step2 خروجی Q0.0 ست می شود . این Step تا زمانی که تایمر کار میکند فعال است با از کار افتادن تایمر Step غیر فعال شده و در این لحظه خروجی Q0.0 ری ست میگردد.



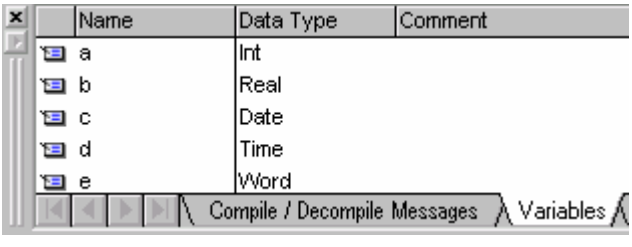
## ۵- دستورات محاسباتی

دستورات محاسباتی میتوانند با Event یا بدون Event اجرا شوند.دستوری که استفاده میشود همانطور که در جدول زیر ملاحظه می شود دستور N است . همراه با N میتوان حرف C را نیز برای اینترلاک بکار برد اگر چه اختیاری است. دستورات محاسباتی به سه دسته تقسیم میشوند:

۱. اختصاص یک مقدار یا یک متغیر به متغیر دیگر A:=B
۲. استفاده از فانکشنهای محاسباتی مانند جذر و سینوس و ... A:= Func(B)
۳. انجام عملیات اصلی جمع و ضرب و منها و تقسیم A:= B <operator> C

| Event | Instruction | Assignment |
|---|---|---|
| -- | N[C] | A:=B |
| | | A:=func(B) |
| | | A:=B<operator>C |
| S0, S1, V0, V1, L0, L1, A1, R1 | N[C] | A:=B |
| | | A:=func(B) |
| | | A:=B<operator>C |

نکته ای که لازمست ذکر شود نحوه تعریف متغیرهای محلی است . در پنجره پایین برنامه در قسمت Varriable با کلیک کردن روی Interface بخشهای In و OUT و STAT و ... مربوط به FB ظاهر میشوند که میتوان متغیرهای محلی دلخواه را از نوع مورد نظر در این قسمت ها تعریف کرد. شکل زیر :




بدینطریق میتوان از متغیرهای محلی همانند آدرسهای دیگر برای عملیات محاسباتی استفاده کرد مثالهای صفحه بعد کاربرد دستورات محاسباتی را بهتر نشان میدهد.

| | |
|---|---|
| متغیر B از جنس Time بصورت محلی تعریف شده و مقدار ۵ ثانیه به آن اختصاص داده شده است. | N B:=T#5S |
| متغیر A بصورت محلی از جنس Integer تعریف شده هر بار که برنامه وارد این Step میشود به A یک عدد اضافه میشود بدین طریق تعداد دفعاتی که Step اجرا شده قابل پیگیری است | S1 N A:=A+1 |
| متغیر محلی C از جنس Word تعریف شده و نتیجه AND دو مقدار MW0 و مقدار FFF0 در هنگام خروج از Step روی آن ریخته شده است | S0 N C:=MW0 AND W#16#FFF0 |
| متغیر محلی E از جنس Real تعریف شده و در صورت برآورده شدن شرایط اینترلاک مقدار سینوس MD0 روی آن ریخته می شود. | N C E:= SIN(MD0) |

انواع عملیات ممکن روی متغیرها در جدول زیر آورده شده اند:

| Assignments with Operator | Comment |
|---|---|
| A := B + C | +I, +D, +R |
| A := B - C | -I, -D, -R |
| A := B ∗C | ∗I, ∗D, ∗R |
| A := B / C | /I, /D, /R |
| A := B MOD C | Modulo: Only for data type DINT |
| A := B AND C | AND operation (STL instructions: AW, AD) |
| A := B OR C | OR operation (STL instructions: OW, OD) |
| A := B XOR C | EXCLUSIVE OR operation (STL instructions: XOW, XOD) |
| A := B SHL C | Shift left, 0<=C<=255 (STL instructions: SLW, SLD) |
| A := B SHR C | Shift right, 0<=C<=255 (STL instructions: SRW, SRD) |
| A := B SSR C | Shift right with sign, 0<=C<=255 (STL instructions: SSI, SSD) |
| A := B ROL C | Rotate left, 0<=C<=255 (STL instructions: RLD) |
| A := B ROR C | Rotate right, 0<=C<=255 (STL instructions: RRD) |

انواع فانکشن های قابل استفاده در جدول زیر آورده شده اند:

| Built-in Function | Comment |
|---|---|
| A := BCD_TO_NUM(B) | BCD to INT or DINT (STL instructions: BTI, BTD) |
| A := NUM_TO_BCD(B) | INT or DINT to BCD (STL instructions: ITB, DTB) |
| A := INT_TO_DINT(B) | INT to DINT (STL instruction: ITD) |
| A := DINT_TO_REAL(B) | DINT to REAL (STL instruction: DTR) |
| A := ROUND(B) | REAL to DINT (STL instruction: RND) |
| A := TRUNC(B) | REAL to DINT, truncate remainder (STL instruction: TRUNC) |
| A := NEGR(B) | REAL negation |
| A := ABS(B) | REAL absolute value |
| A := SQR(B) | REAL square |
| A := SQRT(B) | REAL square root |
| A := LN(B) | REAL logarithm to base e |
| A := EXP(B) | REAL exponent to base e |
| A := SIN(B) | REAL sine |
| A := ASIN(B) | REAL arc sine |
| A := COS(B) | REAL cosine |
| A := ACOS(B) | REAL arc cosine |
| A := TAN(B) | REAL tangent |
| A := ATAN(B) | REAL arc tangent |
| A := NEG(B) | Negation (two's complement) (STL instructions: NEGI, NEGD, NEGR) |
| A := NOT(B) | One's complement (STL instructions: INVI, INVD) |
| A := SWAP(B) | Swap bytes (STL instructions: TAD, TAW) |
| A := RLDA(B) | Rotate 32 bits left by 1 bit via CC1 (STL instruction: RLDA) |
| A := RRDA(B) | Rotate 32 bits right by 1 bit via CC1 (STL instruction: RRDA) |

### ۱۴-۴-۲ برنامه نویسی Condinion

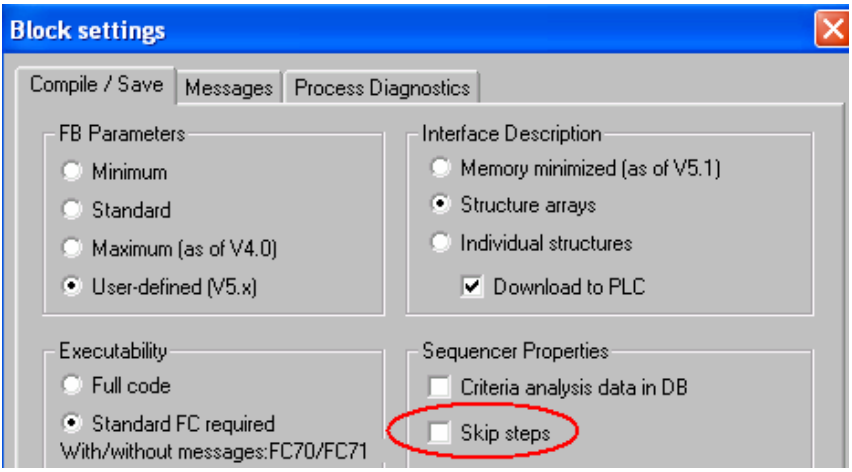
با توضیحاتی که تاکنون داده شده خواننده محترم با Condition و شرایط مورد استفاده در آن آشنا شده است در این بخش با تفصیل بیشتر Condition را مورد بحث قرار می دهیم و نکات خاص مربوط به آن را متذکر می شویم .

Condition توسط المانهای LAD یا FBD و در یکی از ۴ بخش زیر مورد استفاده قرار میگیرد:

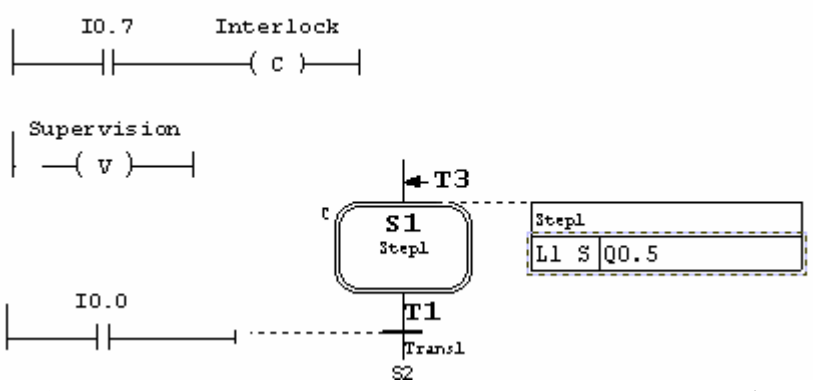
۱. بخش Transition
۲. بخش Interlock
۳. بخش Supervision
۴. بخش Permanent Instruction

نکاتی که لازمست به آنها توجه شود :

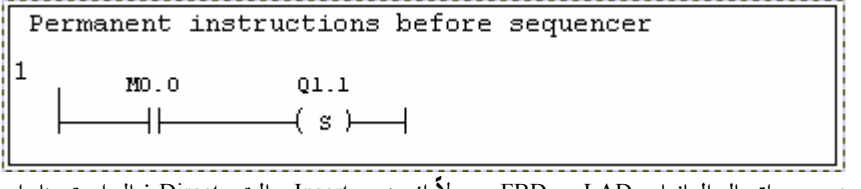
- حداکثر ۳۲ عدد المان LAD یا FBD میتوان در بخش های فوق بکار برد.
- اگر بخش Transition فاقد Condition باشد برنامه از آن Step عبور می کندو در هنگام کامپایل صرفاً warning تولید میشود.
- اگر Condition قبل و بعد از یک Step یکسان باشد در اینصورت با فعال شدن شرایط Step فعال نمیماند و سریعاً به Step بعدی گذر میکند در این شرایط اجرا یا عدم اجرای Step بستگی به تنظیمی دارد که از طریق منوی Option> Block Setting انجام میشود مطابق شکل زیر. اگر Skip Step غیر فعال باشد دستورات Step در حین گذر نیز اجرا میشود ولی در صورت فعال بودن، این دستورات اجرا نخواهند شد.بصورت پیش فرض این گزینه غیر فعال است.



- اگر اینترلاک برنامه نویسی نشود ولی دستور Step همراه با اینترلاک نوشته شود مشکلی پیش نخواهد آمد و در هنگام کامپایل صرفاً Warning تولید می شود.
- وقتی شرایط اینترلاک برقرار نباشد منجر به Disturbance Error میشود و در اینحالت Event با وضعیت L1 تولید شده و برنامه Step اجرا نمیگردد. با بکار بردن L1 میتوان آلارمی را تولید کرد بعنوان مثال در برنامه زیر وقتی کلید I0.7 فعال نباشد Error فوق بوجود می آید و خروجی Q0.5 که نشان دهنده وقوع این اشکال است فعال میگردد.




- فقط در بخش Permanent Instruction میتوان نتیجه را روی خروجی (Coil) انتقال داد. در این بخش علاوه بر = میتوان از S و R نیز استفاده کرد:

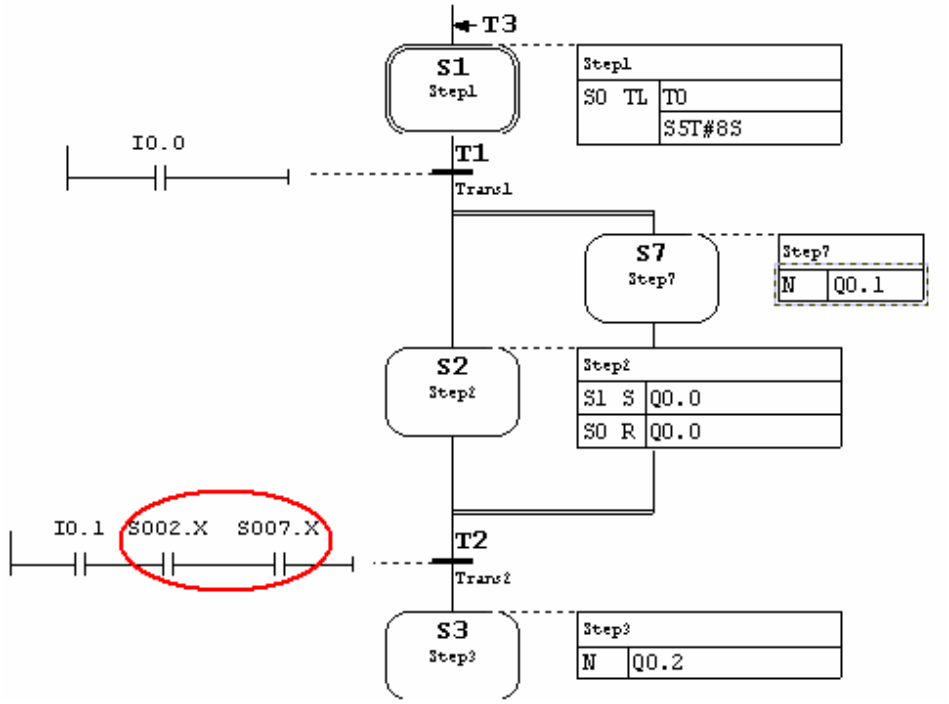



- در اتصال المانهای LAD و FBD معمولاً از منوی Insert حالت Direct فعال است بنابراین ابتدا بایستی با ماوس روی موقعیت مورد نظر سپس روی المان مورد نظر کلیک نماییم. اگر از منوی Insert حالت Drag and Drop انتخاب شود میتوان المان را برداشت و در محل مورد نظر قرار داد.
- انتخاب LAD و FBD از منوی View امکان پذیر است.
- صرفاً المانهای AND و OR و Comparator را می توان استفاده کرد.
- Comparator را برای تمام حالات مقایسه می توان بکار برد کافیست یکی از علامت های =< , > == , <> , <= ,< را روی آن تایپ نمود.

- تمام مقادیر هم نوع مانند TIME , REAL , DINT , INT را میتوان توسط Comparator مورد مقایسه قرار داد.
- از پارامترهای خاص Step و Transition میتوان برای برنامه نویسی Condition استفاده کرد این پارامترها در جدول زیر لیست شده اند . قبلاً پارامتر Step.t را در یکی از مثالها بکار بردیم.

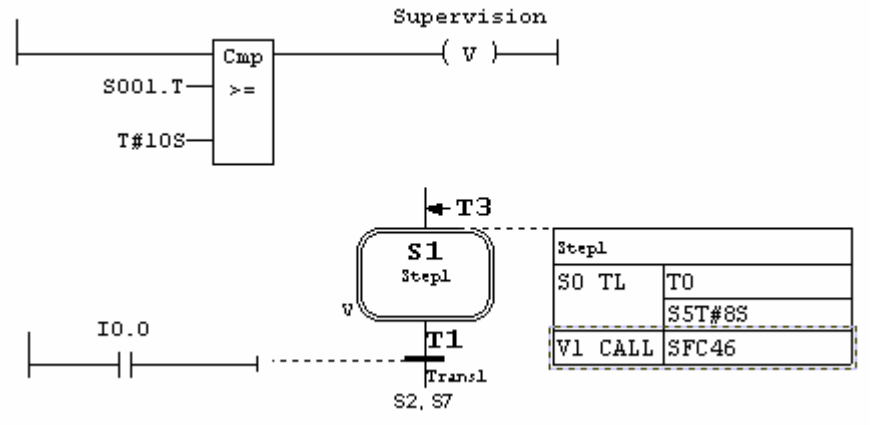
| آدرس | مفهوم | محل استفاده |
|---|---|---|
| Si.T | زمان فعال بودن Step یا آخرین مدت فعال بودن Step | Comparator, assignment |
| Si.U | کل زمان فعال بودن Step بدون زمان مربوط به Disturbance ناشی از اینترلاک یا Supervision | Comparator, assignment |
| Si.X | نشان دهنده اینکه Step مورد نظر فعال است یا خیر | contact |
| Transi.TT | نشان دهنده اینکه آیا شرایط Transition مورد نظر بر آورده شده یا خیر | contact |

برای کنترل زمان CPU از SFC64 استفاده میکند که در برنامه Graph اتوماتیک در پوشه Blocks ظاهر می گردد. در برنامه زیر در صورتی S3 اجرا میشود که قبل از آن S2 و S7 فعال شده باشند.



**نحوه استفاده از Supervision**

در برخی از پروسه ها مهم است که مدت زمان یک Step از حد معینی بیشتر نشود و گرنه ممکن است محصول از بین برود یا دستگاه آسیب ببیند. این کنترل با اعمال نظارت یا Supervision امکان پذیر است . در برنامه مثال زیر با مشاهده Step مورد نظر بصورت Single Step در قسمت Supervision با استفاده از یک مقایسه گر زمان اجرای Step را با مقدار 10 ثانیه مقایسه کرده ایم اگر قبل از ۱۰ ثانیه با فعال شدن سوئیچ I0.0 گذر از Step اتفاق بیفتد شرایط نرمال است گویی که هیچ نظارتی وجود ندارد ولی اگر زمان به ۱۰ ثانیه برسد و I0.0 فعال نگردد در اینصورت خطای Supervision اتفاق می افتد و در حالت On line میبینیم که Step با رنگ قرمز روشن میشود و دیگر فعال و غیرفعال شدن I0.0 تاثیری ندارد و Step گذر نمیکند .در این شرایط میتوان برنامه Step را وابسته به رخداد Supervision نمود . همانطور که قبلاً ذکر شد وقوع Supervision با V1 و برطرف شدن آن با V0 چک میشود پس میتوان در برنامه Step مثلاً در صورت وقوع این خطا دستور داد تا مثلاً PLC با فانکشن SFC46 خاموش شود یا دستور خاص دیگری اجرا شود. لازم است توجه شود که توقف یک توالی بدلیل خطای نظارتی تاثیری روی کار پردازش سایر توالی های موجود در برنامه ندارد.




همانطور که در شکل فوق می بینیم حرف V در کنار Step ظاهر شده و بیانگر اینست که Step تحت نظارت است. نکته ای که هنوز روشن نشده اینست که در صورت وقوع Supervision با توجه به اینکه برنامه در همان Step می ماند چه باید کرد تا گذر اتفاق بیفتد . برای گذر لازم است Acknowledge اعمال شود یعنی مثلاً اپراتور توسط سیستم مانیتورینگ با فشردن یک کلید خطا را Acknowledge کند در اینصورت یک بیت از حافظه CPU که متصل به آن کلید است یک شده و میتوان با استفاده از این بیت Step را عبور داد . برای این

کار به OB1 یا بلاکی که در آن FB فوق صدا زده شده مراجعه می کنیم و می بینیم که در هنگام فراخوانی پارامترهای مختلفی مانند روبرو در زیر FB آمده است. یکی از این پارامتر ها ACK_EF می باشد.

ورودی ACK_EF به یک بیت از دیتا بلاک متصل شده که این بیت توسط اپراتور صفر یا یک میشود اگر برنامه را بصورت On Line ببینیم مشاهده میکنیم که با یک شدن این بیت خطای Supervision برطرف شده و برنامه به شرایط عادی باز میگردد.

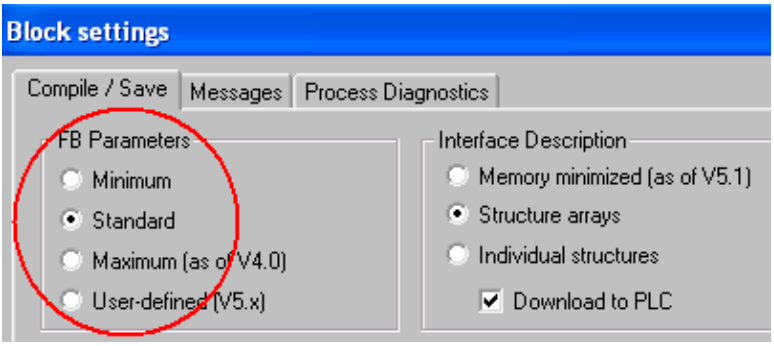
```
CALL FB    3 , DB3
 OFF_SQ :=
 INIT_SQ :=
 ACK_EF  := DB1.DBX0.0
 S_PREV  :=
 S_NEXT  :=
 SW_AUTO :=
 SW_TAP  :=
 SW_MAN  :=
 S_SEL   :=
 S_ON    :=
 S_OFF   :=
 T_PUSH  :=
 S_NO    :=
 S_MORE  :=
 S_ACTIVE:=
 ERR_FLT :=
 AUTO_ON :=
 TAP_ON  :=
 MAN_ON  :=
```

در مورد سایر ورودی و خروجی های فوق در بخش بعد توضیحاتی خواهد آمد.

### ۵-۱٤ پارمترهای FB در S7_Graph

وقتی FB که توسط Graph نوشته شده در بلاک دیگر ی همراه با DB صدا زده می شود پارامترهای ورودی و خروجی آن ظاهر میشوند . همانطور که گفته شد الزامی به اختصاص مقادیر به این پارامترها وجود ندارد و از آنها میتوان فقط در صورت لزوم استفاده نمود.

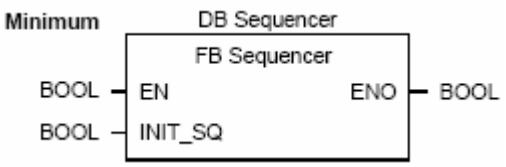
نکته مهمی که در ارتباط با این پارامترها وجود دارد متغیر بودن تعداد آنهاست به عبارت دیگر ممکن است در شرایطی تعدا د پارامترها از آنچه در بالا نشان داده شده کمتر یا بیشتر باشد. این موضوع بستگی به تنظیمی دارد که برای FB در منوی Option>Block Setting از برنامه S7-Graph مانند شکل زیر انجام میشود.



همانطور که در شکل قبل مشاهده میشود جهار گزینه برای انتخاب وجود دارد و بطور پیش فرض گزینه Standard فعال است . گزینه Minimum کمترین امکانات را در اختیار کاربر میگذارد و حافظه کمتری از CPU را اشغال میکند ولی گزینه آخر یعنی User -Defined بیشترین امکانات را فراهم کرده و حافظه بیشتری را نیز به خود اختصاص می دهد. استفاده از مد دستی یا اتوماتیک یا نیمه اتوماتیک از جمله این موارد است. در ادامه شرح دقیقتری از گزینه های چهارگانه فوق آمده است.

**Minimum**

در این حالت حداقل امکانات در دسترس است و فقط جایی استفاده میشود که نیازی به فانکشن های مانیتورینگ یا فانکشنهای کنترل بیشتر وجود ندارد و در عین حال می خواهیم از حافظه بصورت بهینه استفاده گردد . با انتخاب این گزینه و ذخیره سازی FB وقتی در بلاک ماقبل آنرا صدا میزنیم مانند شکل زیر فقط یک ورودی برای آن مشاهده می نماییم:

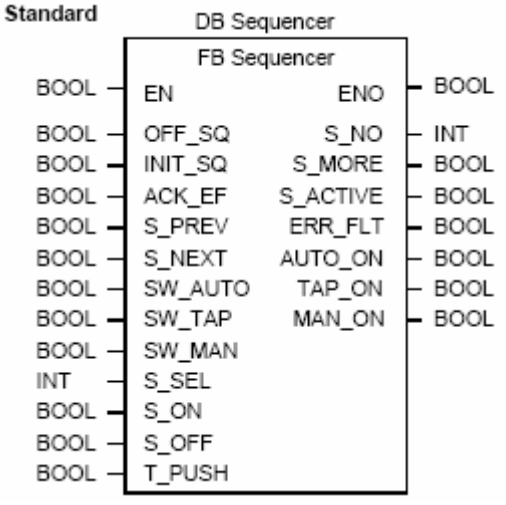



ورودی INIT_SQ برای چهار گزینه فوق الذکر وجود دارد وهمانطور که از نامش پیداست برای Initialize کردن Sequencer بکار می رود و از جنس Bool است. هرگاه این ورودی یک شود Step فعلی در توالی رها شده و به Step اولیه بر میگردد.در برخی موارد لازم است در صورتی که بین اجرای توالی CPU متوقف شود پس از راه اندازی مجدد از Step اولیه شروع نماید . برای اینکار FB را در OB راه اندازی مانند OB100 همراه با یک DB صدا زده و پارامتر INIT_SQ را در آنجا یک میکنیم این امر موجب میشود که همواره پس از راه اندازی ، توالی از مرحله اول شروع نماید. در صفحات آینده مثالی از کاربرد این ورودی در سیستم کنترل آورده شده است.

عدم وجود ورودی و خروجی های دیگر در حالت Minimum امکانات آنرا محدود ساخته است بعنوان مثال بدلیل عدم وجود ورودی ACK_EF امکان Acknowledge کردن خطاها از جمله خطای Supervision وجود ندارد.

**Standard**

در این حالت امکانات استاندارد در دسترس است میتوان مدهای کاری مختلف مانند دستی و اتوماتیک را انتخاب کرد ، امکان Acknowledge کردن خطا ها وجود دارد و .... در اینحالت وقتی از بلاک ماقبل FB را صدا می زنیم ورودی ها و خروجی ها مانند شکل زیر خواهند بود. همانطور که ملاحظه میشود تعداد ورودی و خروجی در اینحالت بسیار بیشتر از حالت Minimum است و امکانات FB نیز به همین دلیل بیشتر است. بجز ورودی INIT_SQ که قبلاً شرح داده شد سایر ورودی ها در جدول بعد معرفی گردیده اند.




لازم است ذکر شود که در اینحالت سه مد کاری برای توالی قابل تنظیم است:

- **مد اتوماتیک** : در این مد گذر از یک مرحله به مرحله بعد بر اساس شرایط Transition بطور اتوماتیک انجام می شود.
- **مد دستی** : در این مد گذر از یک مرحله به مرحله بعد فقط بصورت دستی و توسط ورودی های S_ON و S_OFF از FB انجام میشود و به شرایط Transition بستگی ندارد.
- **مد Inching** : در این مد که نیمه اتومات است برای گذر سیگنال نه تنها بایستی Condition در Transition برآورده شده باشد بلکه لازم است ورودی T_pushنیز فعال شده باشد به همین دلیل به آن Transition And Push یا TAP نیز می گویند.

مد های فوق توسط ورودی های SW_AUTO و SW_MAN و SW_TAP قابل انتخاب است.

بعداً خواهیم دید که برای گزینه های دیگر مد نیمه اتومات دیگری نیز وجود دارد که با ورودی SW_TOP فعال میشود در این مد Condition یا T_Push هر کدام فعال شوند گذر اتفاق می افتد.

| Parameter | Data Type | مفهوم |
|---|---|---|
| OFF_SQ | BOOL | با یک شدن این ورودی توالی یعنی تمام Stepها غیر فعال می شوند. |
| ACK_EF | BOOL | با یک شدن این ورودی خطای پیش آمده Acknowledge میشود و توالی اجباراً به Step بعد هدایت میگردد |
| REG_EF | BOOL | اگر فعال شود تمام خطا ها و Disturbance ها ثبت میگردد |
| ACK_S | BOOL | مرحله ای که شماره آن در خروجی S_NO ظاهر شده با فعال شدن این ورودی Acknowledge می شود. |
| REG_S | BOOL | مرحله ای که شماره آن در S_NO ظاهر شده با فعال شدن این ورودی Register می شود. |
| S_PREV | BOOL | با یک شدن این ورودی توالی از Step جاری به مرحله قبل بر می گردد |
| S_NEXT | BOOL | با یک شدن این ورودی توالی از Step جاری به مرحله بعد میرود |
| SW_AUTO | BOOL | فعال کردن حالت اتومات |
| SW_TAP | BOOL | فعال کردن حالت نیمه اتومات Inching بصورت Transition and Push |
| SW_MAN | BOOL | فعال کردن مد دستی |
| S_SEL | INT | انتخاب شماره Step برای نمایش وضعیت آن روی خروجی S_NO |
| S_ON | BOOL | برای فعال کردن Step در مد دستی |
| S_OFF | BOOL | برای غیرفعال کردن Step در مد دستی |
| T_PREV | BOOL | برای نمایش Transition قبلی روی خروجی T_NO |
| T_NEXT | BOOL | برای نمایش Transition بعدی روی خروجی T_NO |
| T_PUSH | BOOL | در حالت نیمه اتومات برای گذر از یک مرحله بکار میرود . |

خواننده محترم میتواند برای فهم بهتر عملکرد این ورودی ها تمرین های زیر را شخصاً انجام دهد:

**تمرین ۱:** در یکی از مثال هایی که تاکنون ارائه شده در هنگام صدا زدن FB ورودی SW_TAP رابه M0.0 و T_PUSH را به M0.1 نسبت دهید . بلاک ها را ذخیره و به PLC یا سیمولاتور دانلود کنید سپس برنامه Graph را مانیتور نمایید . مشاهده خواهید کرد که تا زمانیکه M0.0 صفر است پردازش برنامه عادی است و مد کاری اتوماتیک است ولی با یک شدن M0.0 در صورتی که M0.1 یک نباشد حتی با وجود برآورده شدن Condition گذر از Step اتفاق نخواهد افتاد مگر اینکه M0.1 یک شود.

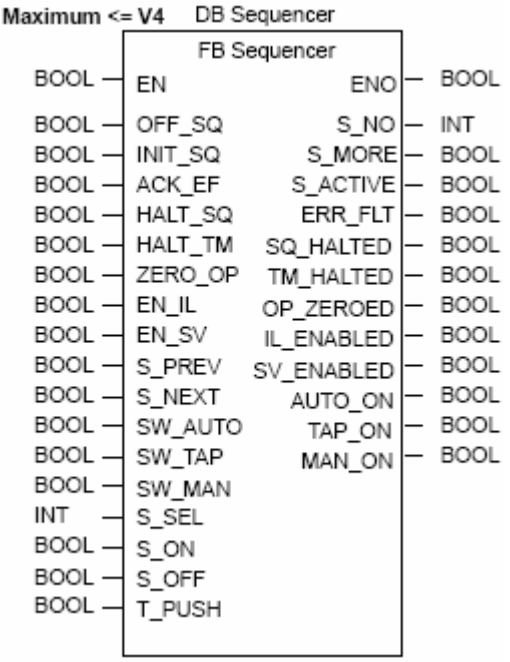
**تمرین ۲:** به ورودی های SW_AUTO و SW_MAN و SW_TAP سه فلگ نسبت دهید مثلاً M0.0 M0.1 و M0.2 سپس پس از دانلود بلاک بترتیب آنها را صفر و یک کنید مشاهده خواهید کرد که با لبه مثبت هر کدام وضعیت مد کاری تغییر کرده و همانطور می ماند تا اینکه با ورودی دیگر این وضعیت تغییر داده شود. در حالت Online مد کاری را میتوان در پایین پنجره Graph با رنگ سبز مشاهده نمود بصورت زیر:

| | | |
|---|---|---|
| MAN | TAP | AUTO |

مد AUTO همان وضعیت پیش فرضی است که تاکنون همه مثالها را با آن تست کردیم . مد TAP را نیز در تمرین ۱ بررسی نمودیم. در مد MAN نیز مشاهده میشود که با بر آورده شدن Condition در Transitoin گذر از Step اتفاق نمی افتد.

**Maximum**

در این حالت امکانات بیشتری نسبت به حالت استاندارد در دسترس است شرح ورودی هایی که این فانکشن علاوه بر فانکشن استاندارد دارد در جدول بعد آمده است.

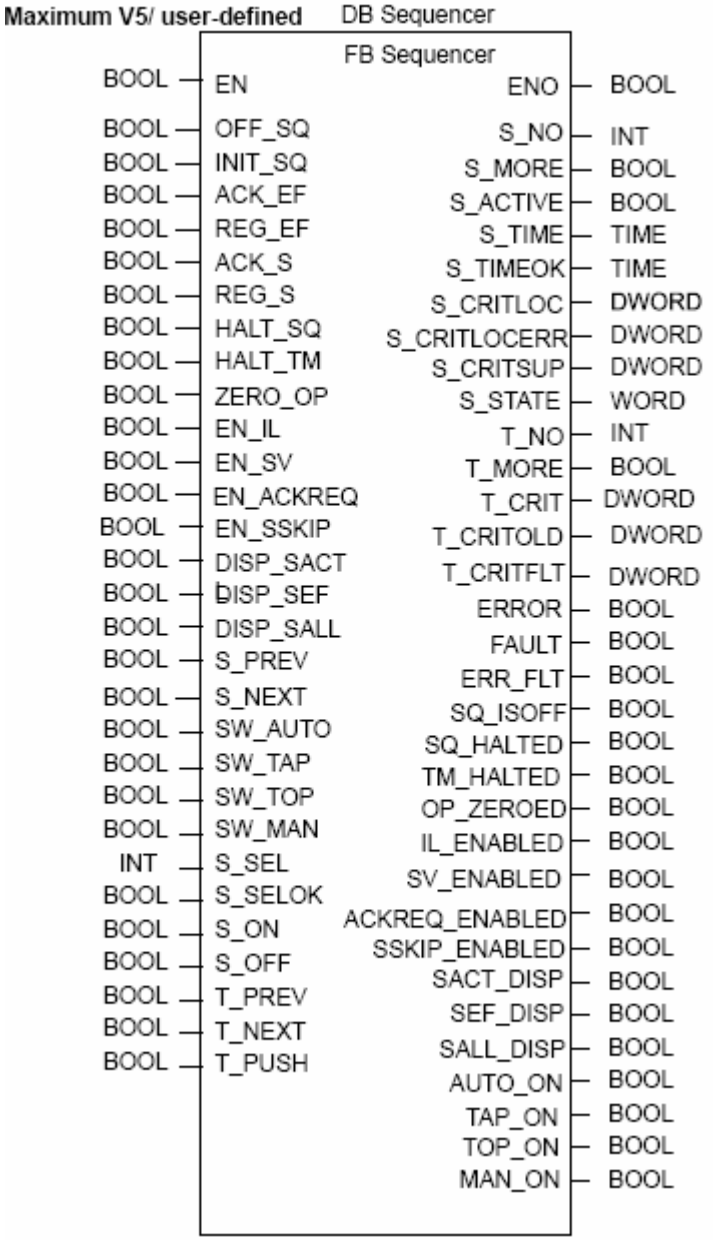


| Parameter | Data Type | مفهوم |
|---|---|---|
| HALT_SQ | BOOL | با یک شدن این ورودی توالی به حالت تعلیق در می آید. |
| HALT_TM | BOOL | با یک شدن این ورودی تمام دستورات مبتنی بر زمان مانند L و D و نیز زمان سنج Step ها به حالت تعلیق در می آید. |
| ZERO_OP | BOOL | اگر فعال شود تمام آدرس هایی که در دستورات N و D و L بکار رفته اند صفر میشوند و دستور CALL نیز اجرا نمیشود. |
| EN_IL | BOOL | با این ورودی می توان اینترلاک را فعال یا غیر فعال کرد. |
| EN_SV | BOOL | با این ورودی می توان Supervision را فعال یا غیر فعال کرد. |

**Maximum**

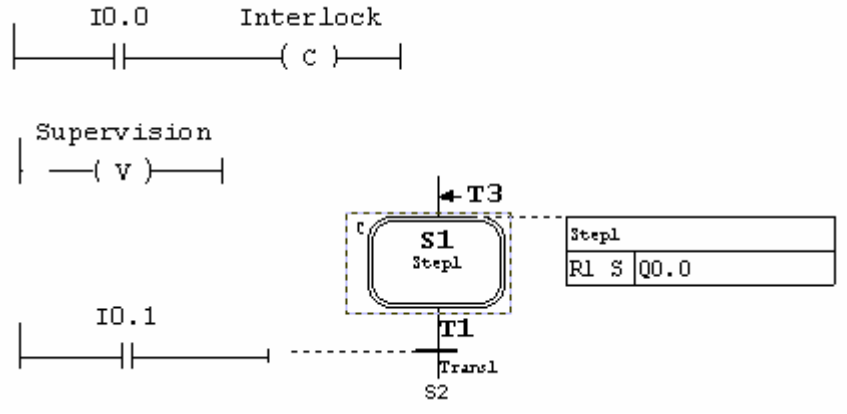
در این حالت امکانات بیشتری نسبت به حالت Maximum در دسترس است شرح ورودی هایی که این فانکشن علاوه بر فانکشن استاندارد دارد در جدول زیر و شکل فانکشن در صفحه بعد آمده است.

| Parameter | Data Type | مفهوم |
|---|---|---|
| REG_EF | BOOL | با یک شدن این ورودی تمام Error ها و Disturbance ها ثبت میشوند. |
| ACK_S | BOOL | با یک شدن این ورودی شماره Step که Acknowledge شده روی S_NO نمایش داده می شود. |
| REG_S | BOOL | با یک شدن این ورودی شماره Step که Register شده روی S_NO نمایش داده می شود. |
| EN_ACKREQ | BOOL | برای فعال کردن نیاز به Acknowledge بکار میرود. |
| DISP_SAKT | BOOL | فقط Step فعال نشان داده می شود. |
| DISP_SEF | | فقط step که دچار Error یا Disturbance شده نشان داده میشود. |
| DISP_SALL | | تمام Step ها نشان داده میشوند. |
| SW_TOP | | مد اتومات/دستی را فعال میکند در اینحالت اگر Conditon برآورده شود یا T_PUSH فعال شود گذر اتفاق می افتد. |
| S_SELOK | | مقدار انتخاب شده برای S_SEL در S_NO نمایش داده میشود. |
| T_PREV | | Transition فعال شده قبلی در T_NO نشان داده میشود. |
| T_NEXT | | Transition فعال شده بعدی در T_NO نشان داده میشود. |
| EN_SSKIP | | پرش از Step را فعال می کند. ( به بحث Skip Step رجوع شود) |


Maximum V5/ user-defined
DB Sequencer
FB Sequencer
BOOL EN
BOOL OFF_SQ
BOOL INIT_SQ
BOOL ACK_EF
BOOL REG_EF
BOOL ACK_S
BOOL REG_S
BOOL HALT_SQ
BOOL HALT_TM
BOOL ZERO_OP
BOOL EN_IL
BOOL EN_SV
BOOL EN_ACKREQ
BOOL EN_SSKIP
BOOL DISP_SACT
BOOL DISP_SEF
BOOL DISP_SALL
BOOL S_PREV
BOOL S_NEXT
BOOL SW_AUTO
BOOL SW_TAP
BOOL SW_TOP
BOOL SW_MAN
INT S_SEL
BOOL S_SELOK
BOOL S_ON
BOOL S_OFF
BOOL T_PREV
BOOL T_NEXT
BOOL T_PUSH
ENO BOOL
S_NO INT
S_MORE BOOL
S_ACTIVE BOOL
S_TIME TIME
S_TIMEOK TIME
S_CRITLOC DWORD
S_CRITLOCERR DWORD
S_CRITSUP DWORD
S_STATE WORD
T_NO INT
T_MORE BOOL
T_CRIT DWORD
T_CRITOLD DWORD
T_CRITFLT DWORD
ERROR BOOL
FAULT BOOL
ERR_FLT BOOL
SQ_ISOFF BOOL
SQ_HALTED BOOL
TM_HALTED BOOL
OP_ZEROED BOOL
IL_ENABLED BOOL
SV_ENABLED BOOL
ACKREQ_ENABLED BOOL
SSKIP_ENABLED BOOL
SACT_DISP BOOL
SEF_DISP BOOL
SALL_DISP BOOL
AUTO_ON BOOL
TAP_ON BOOL
TOP_ON BOOL
MAN_ON BOOL

**تمرین ۳:** در برنامه تمرین ۱ توسط منوی Option> Block Setting گزینه User Defned را انتخاب نمایید. سپس FB را ذخیره و از همانجا به PLC دانلود کنید. پس از آن در بلاک ماقبل مثلاً در OB1 مجدداً FB را همراه با DB صدا بزنید خواهید دید که پارامترهای بیشتری نسبت به حالت قبل در زیر FB ظاهر میشود. یکی از این پارامترها SW_TOP است که اگر آنرا به یک بیت مثلاً M1.0 نسبت دهید با لبه مثبت سیگنال در این ورودی توالی به حالت اتومات/دستی در می آید این وضعیت را با کلمه TOP در پایین پنجره در حالت Online نیز میتوانید مشاهده نمایید. در این مد علاوه بر اینکه Condition موجود در Transition ها موجب گذر از یک مرحله به مرحله بعد هستند میتوان توسط کلیدی که به ورودی T_PUSH اختصاص داده شده نیز باعث گذر از یک Step به Step دیگر شد . پس برخلاف مد TAP که بایستی هم Condition و هم T_PUSH فعال باشد تا گذر اتفاق بیفتد در اینجا یکی از این دو میتواند موجب عبور به مرحله بعد شودبه همین دلیل به آن Transition Or Push یا TOP می گویند.

**تمرین ٤:** در حالتی که گزینه User Defined از منوی Option> Block Setting فعال است امکان استفاده از دستور registration که فقط برای این حالت قابل استفاده است را چک کنید . برای این کار در هنگام صدا زدن FB ورودی REG_EF را به فلگ M0.0 نسبت دهید سپس برنامه ای در یکی از Step ها مانند شکل زیر بنویسید. در این برنامه در شرایط نرمال که اینترلاک I0.0 فعال است خروجی Q0.0 خاموش است وقتی اینترلاک غیر فعال باشد و خطای disturbance رخ دهد در اینحالت در مد online رنگ Step قرمز میگردد و باز خروجی Q0.0 خاموش است ولی در همین شرایط اگر M0.0 را فعال کنیم خروجی Q0.0 که وابسته به event است روشن می گردد.



**تست مد های کاری با استفاده از Debug**

تمام مواردی که در ارتباط با مدهای کاری و پارامترهای مختلف ذکر شد توسط امکان Debug در برنامه Graph نیز قابل تست است. برای فعال کردن آن کافیست پس از دانلود در حالی که PLC در مد Run است با استفاده از منوی Debug > Control Sequencer پنجره ای مانند شکل صفحه بعد را فعال کنیم . توسط این پنجره میتوان تمام پارامترهای ورودی FB را بدون اینکه نیاز باشد در داخل برنامه مقدار دهی شوند تست کرد از جمله :

- تست تمام مدهای کاری AUTO , MAN , TAP , TOP
- فعال یا غیر فعال نمودن توالی
- تست Interlock و Supervision

برای روشن شدن بیشتر چند حالت را با استفاده از این پنجره بررسی می نماییم.

۱ – مدهای کاری را با استفاده از گزینه های بالای این پنجره تغییر دهید و تاثیر آنرا روی Graph در حالت Online ببینید.

۲- Initial کردن توالی را با استفاده از کلید Initialize بالای پنجره چک کنید.

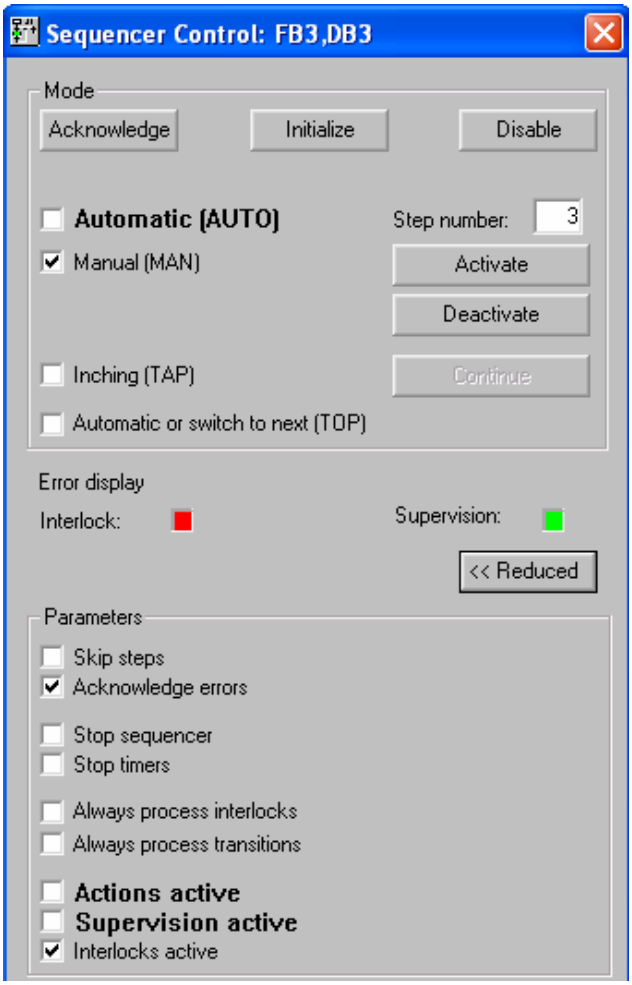
۳- غیر فعال کردن کل توالی را با استفاده از کلید Disable بالای پنجره چک کنید.

۴- در یک Step از اینترلاک استفاده کنید. مد کاری را AUTO انتخاب نمایید . با استفاده از گزینه پایین پنجره میتوانید اینترلاک را فعال یا غیر فعال کنید.

۵- حالت ۴ را برای Supervision چک کنید با گزینه Supervision Active میتوانید آن را فعال یا غیر فعال کنید.

۶- در حالی که در مد AUTO گزینه Supervision فعال است از کلید Acknowlege بالای پنجره استفاده کنید خواهید دید که در مد Online رنگ قرمز تبدیل به سبز شده و گذر اتفاق می افتد.

۷- مد را به حالت Manual تغییر دهید . در اینحالت ابتدا در جلوی Step Number نام Step فعلی که فعال است را بنویسید سپس آنرا با کلید Deactivate غیر فعال کنید خواهید دید که در حالت Online این Step غیرفعال میشود. پس از آن نام Step دلخواه دیگری را بنویسید و آنرا Active کنید مشاهده خواهید کرد که Step مورد نظر که میتواند چندین Step بعد باشد فعال میگردد. در اینحالت دقت کنید که Condition موجود در Transition ها هیچ تاثیری روی گذر از Step ها ندارند و فقط بصورت دستی میتوان هر Step دلخواهی را فعال یا غیر فعال کرد.




۸- توالی را Initial کرده و مد را به حالت TOP تغییر دهید مشاهده خواهید کرد که کلید Continue فعال میشود . اگر این کلید فشار داده شود **یا** شرایط Condition فعال شود گذر از Step اتفاق خواهد افتاد.

۹-توالی را Initial کرده و مد را به حالت TAP تغییر دهید مشاهده خواهید کرد که اگر کلید Continue فشار داده شود **و** شرایط Condition فعال شود گذر از Step اتفاق خواهد افتاد.

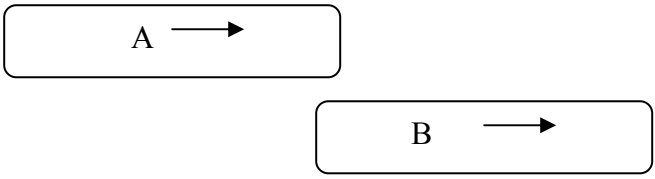
### ۱۴-۶ ارائه چند مثال با S7-Graph

در این قسمت چند برنامه که با S7-Graph نوشته شده مورد بحث قرار میدهیم خواننده محترم توجه دارد که برنامه نویسی همیشه روش منحصر به فرد ندارد و یک برنامه را می توان به روش های مختلفی نوشت.

### مثال ۱: کنترل ترتیبی دو نوار نقاله

در فرآیندی دو نوار نقاله مانند شکل وجود دارند که موادی را به بخش دیگری از فرآیند هدایت می کنند . روشن شدن و خاموش شدن این دو نوار به ترتیب و بطور کلی منطق زیر لازم است بر سیستم حاکم باشد:

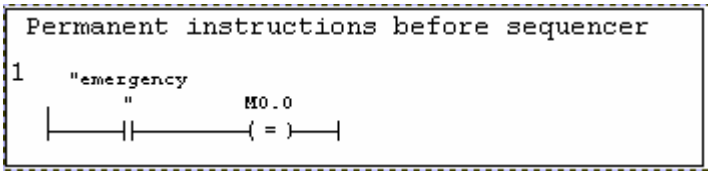
- با زدن شستی Start ابتدا نوار B و با ۵ ثانیه تاخیر نوار A روشن شود.
- با زدن شستی Stop ابتدا نوار A و با ۷ ثانیه تاخیر نوار B خاموش شود.
- با زدن شستی Emergrncy هر دو نوار بلافاصله خاموش شوند.



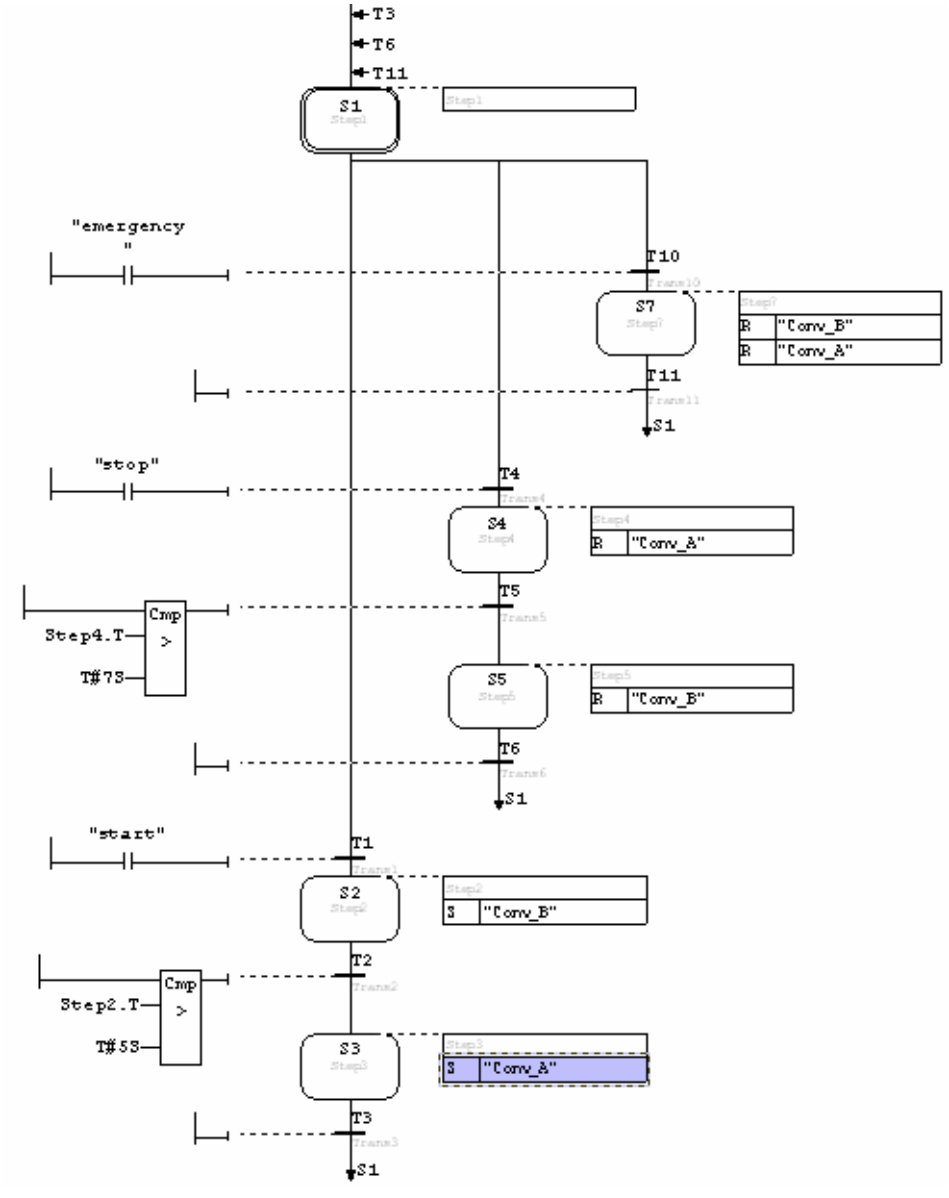

بایستی توجه داشت که اگر کلید Start زده شده و برنامه نوار B را روشن کرده و در حال زمان سنجی برای روشن کردن نوار A می باشد و در این شرایط کلید Emergency زده شد بایستی بلافاصله هردو نوار خاموش شوند و همینطور اگر کلید Stop زده شده و برنامه نوار A را خاموش کرده و در حال زمان سنجی برای خاموش کردن نوار B میباشد و سیگنال Emergency برسد باز در اینجا لازم است نوار B نیز بلافاصله خاموش گردد. برنامه در S7-Graph با توجه به نکات زیر نوشته شده است:

- برای هر کدام از سه حالت روشن خاموش و قطع اضطراری یک شاخه موازی منظور شده است.
- شاخه های موازی همگی به Step1 که Initial تعریف شده متصل هستند. در این step برنامه وضعیت سه کلید را می بیند و متناسب باآن دستورات شاخه مورد نظر را اجرا می کند.

- در شاخه مربوط به روشن شدن ، ابتدا نوار B با دستور S روشن شده و روشن میماند سپس با استفاده از یک مقایسه گر زمان وقتی ۵ ثانیه در Step 2 سپری شد برنامه بطور اتوماتیک وارد Step 3 میگردد. و پس از Set کردن خروجی مربوط به نوار A به step اولیه بازگشت می کند.
- در شاخه مربوط به خاموش شدن ، ابتدا نوار A با دستور R خاموش شده سپس با استفاده از یک مقایسه گر زمان وقتی ۷ ثانیه در Step 4 سپری شد برنامه بطور اتوماتیک وارد Step 6 میگردد. و پس از Reset کردن خروجی مربوط به نوار B به step اولیه بازگشت می کند.
- در شاخه قطع اضطراری هر دو نوار در Step 7 خاموش شده و برنامه به Step1 بازگشت می نماید.
- تا اینجا برنامه در برخی شرایط درست کار میکند ولی در مواردی قطع اضطراری عمل نمی نماید . بعنوان مثال وقتی برنامه در Step2 قرار دارد و هنوز تایمر به ۵ ثانیه نرسیده اگر Emergency برسد عمل نمی کند زیرا فعال شدن Emergency فقط وقتی دیده میشود که برنامه در Step1 قرار داشته باشد. برای رفع این مشکل از تدبیر خاصی استفاده شده است بدینصورت که متغیر حافظه M0.0 به ورودی Initialize مربوط به FB در هنگام فراخوانی در OB1 اختصاص داده شده است. این ورودی در برنامه Gragh بصورتی بکار رفته که با فعال شدن شستی اضطراری M0.0 یک میگردد و منجر به Initialize شدن توالی یعنی بازگشت به Step1 میگردد بدیهی است با بازگشت به Step1 بلافاصله برنامه وارد مسیر قطع اضطراری میگردد و هر دو نوار را خاموش می کند.اما از آنجا که این کنترل لازمست در تمامی Step ها بتواند اعمال شود برنامه آن مطابق شکل زیر در بخش Permanent Instruction نوشته شده است.



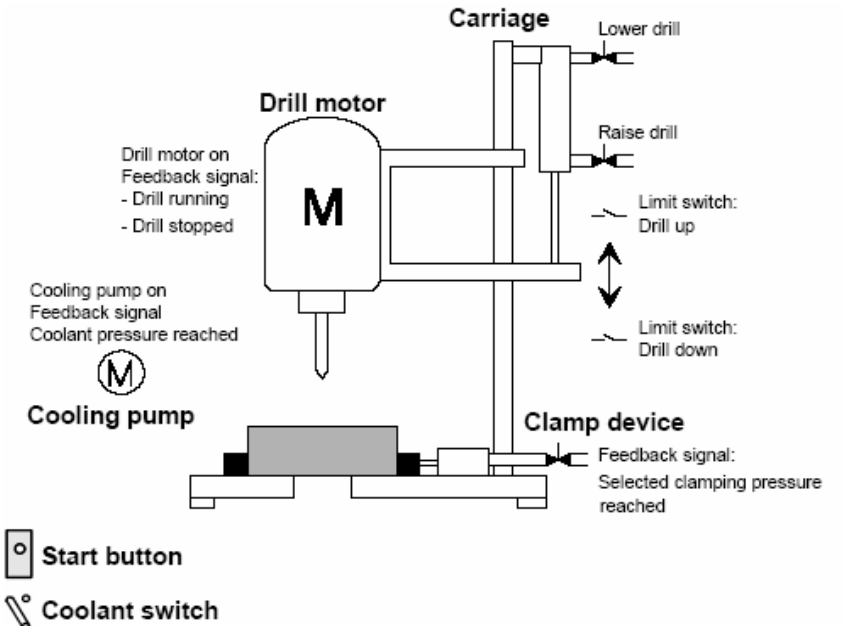

شکل توالی در صفحه بعد ترسیم شده است.


T3
T6
T11
S1
Step1
"emergency"
T10
S7
Step7
R "Conv_B"
R "Conv_A"
T11
S1
"stop"
T4
S4
Step4
R "Conv_A"
T5
Cmp
>
Step4.T
T#7S
S5
Step5
R "Conv_B"
T6
S1
"start"
T1
S2
Step2
S "Conv_B"
T2
Cmp
>
Step2.T
T#5S
S3
Step3
S "Conv_A"
T3
S1

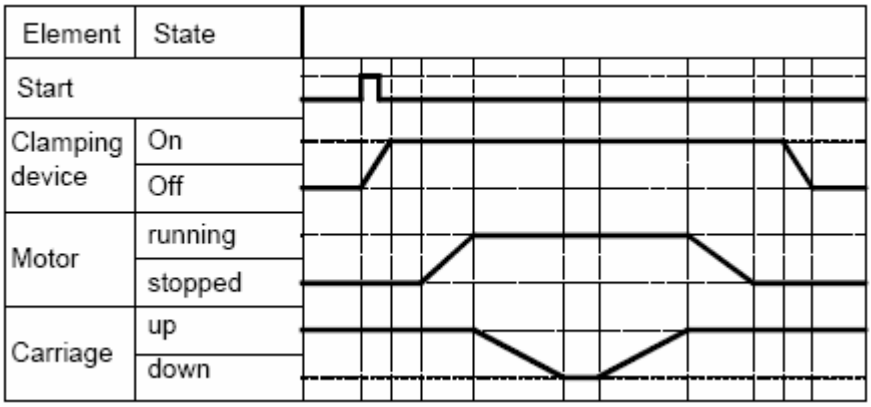
**مثال ۲: کنترل یک دریل اتوماتیک**

یک دریل برقی برای مته کاری روی قطعه کار استفاده میشود بجز گذاشتن قطعه اولیه و برداشتن قطعه سوراخ کاری شده که بصورت دستی است بقیه عملیات بصورت اتوماتیک انجام میشود این مراحل عبارتند از :

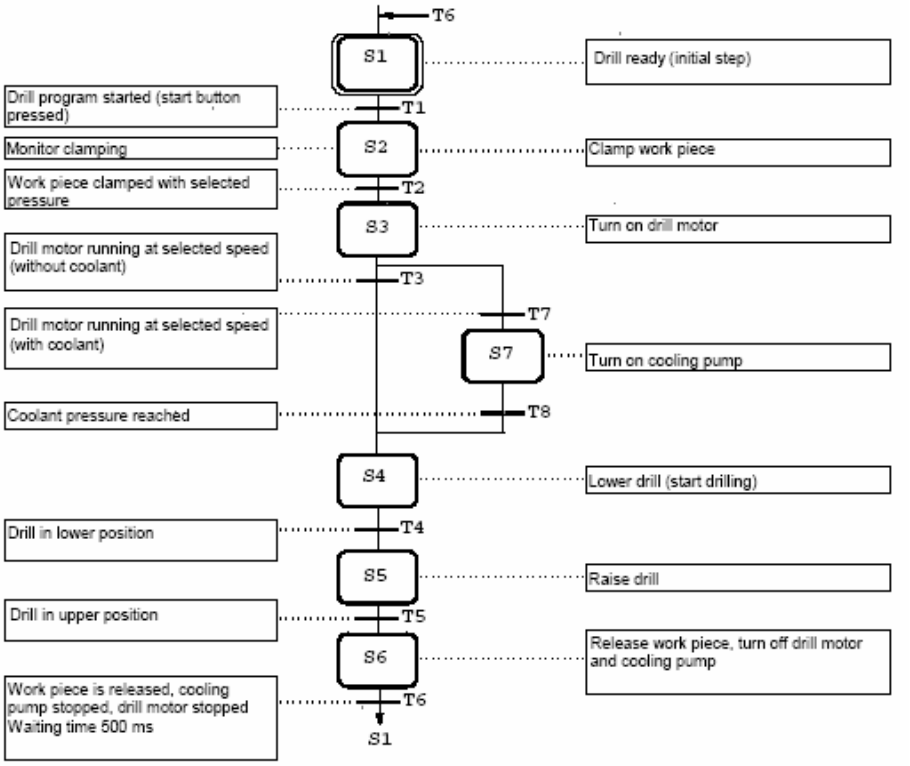
۱. قطعه کار بصورت دستی زیر دریل قرار می گیرد.
۲. بسته به جنس قطعه، سوئیچ Coolant توسط اپراتور انتخاب میشود یعنی در مواردی که لازم باشد در محل مته کاری خنک کاری انجام شود این سوئیچ انتخاب میگردد.
۳. ماشین با کلید Start توسط اپراتور روشن می شود.
۴. قطعه تحت فشار کلمپ میشود و فیدبک فشار به کنترلر داده میشود.
۵. پمپ سیستم خنک کاری در صورتی که کلید coolant انتخاب شده باشد روشن میشود.
۶. دریل توسط کارتریج پایین می آید تا به وضعیتی که توسط لیمیت سوئیچ مشخص شده برسد.
۷. نیم ثانیه در موقعیت نهایی مکث می کند.
۸. دریل توسط کارتریج به بالا بر میگردد تا به لیمیت سوئیچ حد بالا برسد.
۹. موتور دریل و پمپ خنک کاری خاموش میشود.
۱۰. قطعه کار آزاد شده و بصورت دستی برداشته می شود.

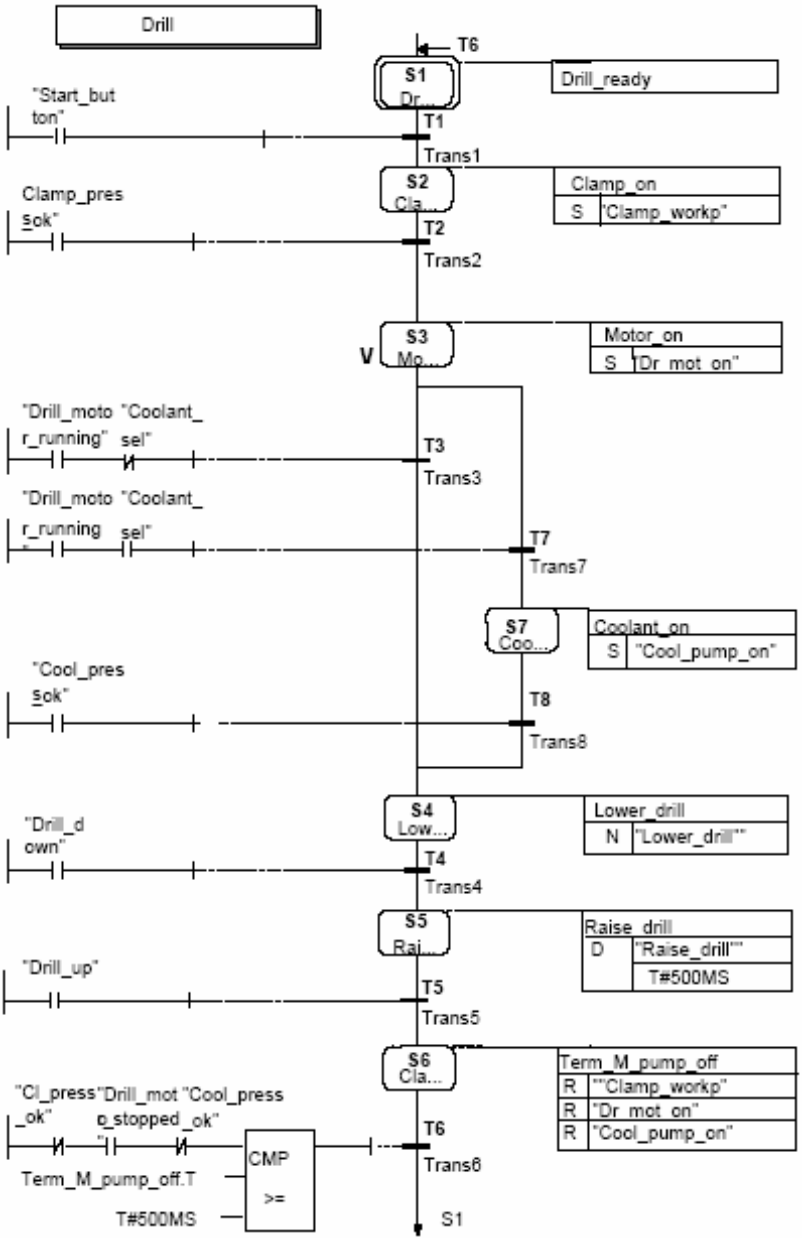


با توجه به مراحل کاری فوق الذکر ترتیب عملیات در شکل زیر نشان داده شده است :

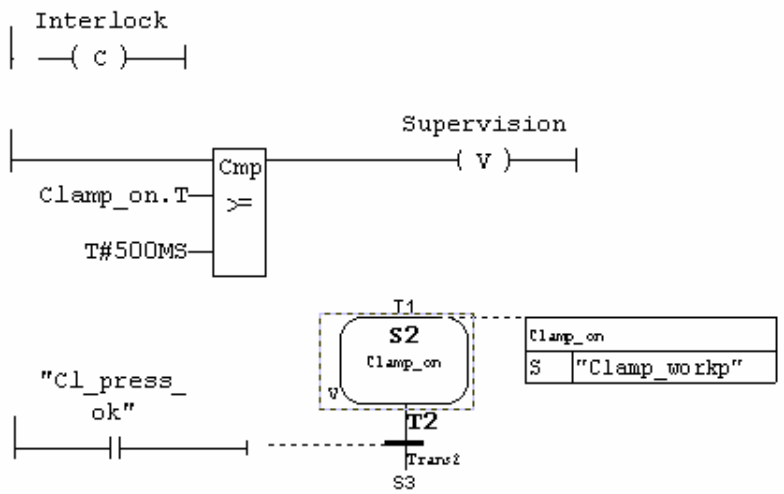



شکل کلی برنامه Graph بصورت زیراست برنامه اصلی در صفحه بعد آمده است .




Drill
T6
S1
Dr...
Drill_ready
"Start_but
ton"
T1
Trans1
S2
Cla...
Clamp_on
S "Clamp_workp"
Clamp_pres
sok"
T2
Trans2
S3
Mo...
V
Motor_on
S "Dr_mot_on"
"Drill_moto "Coolant_
r_running" sel"
T3
Trans3
"Drill_moto "Coolant_
r_running sel"
T7
Trans7
S7
Coo...
Coolant_on
S "Cool_pump_on"
"Cool_pres
sok"
T8
Trans8
S4
Low...
Lower_drill
N "Lower_drill""
"Drill_d
own"
T4
Trans4
S5
Rai...
Raise_drill
D "Raise_drill""
T#500MS
"Drill_up"
T5
Trans5
S6
Cla...
Term_M_pump_off
R ""Clamp_workp"
R "Dr_mot_on"
R "Cool_pump_on"
"Cl_press"Drill_mot "Cool_press
_ok" o_stopped_ok"
CMP
>=
Term_M_pump_off.T
T#500MS
T6
Trans6
S1

همانطور که در توالی دیده میشود Step2 تحت Supervision قرار گرفته اگر آنرا بصورت Single ببینیم مانند شکل زیر خواهد بود. همانطور که ملاحظه میشود اگر در طول مدت ۵۰۰ میلی ثانیه بعد از روشن شدن دریل فیدبک فشار کلمپ نرسد سیستم با خطای Supervision متوقف می گردد

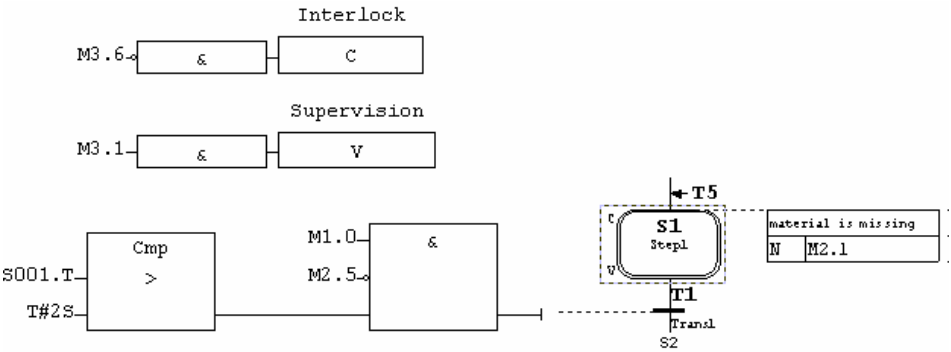



سمبلهایی که برای آدرسهای اصلی در این برنامه استفاده شده بصورت زیر است . در ضمن M0.0 در OB1 به ورودی INIT_SQ داده شده تا توسط آن توالی در صورت لزوم به Step1 برگردد.

| | Symbol | Address | Data type | Comment |
|---|---|---|---|---|
| 1 | Cl_press_ok | I 0.4 | BOOL | Work piece clamping pressure reached |
| 2 | Clamp_workp | Q 0.4 | BOOL | Clamp work piece at required pressure |
| 3 | Cool_press_ok | I 0.6 | BOOL | Cooling pump running |
| 4 | Cool_pump_on | Q 0.1 | BOOL | Supply coolant |
| 5 | Coolant_sel | I 0.5 | BOOL | Cooling pump selector switch on |
| 6 | Cyclic_prg | OB 1 | OB 1 | Cyclic program |
| 7 | Dr_mot_on | Q 0.0 | BOOL | Turn on drill |
| 8 | Dr_mot_running | I 0.0 | BOOL | Drill motor running at selected speed |
| 9 | Dr_mot_stopped | I 0.1 | BOOL | Drill motor stopped |
| 10 | Drill_down | I 0.2 | BOOL | Limit switch "drill in lower position" |
| 11 | Drill_up | I 0.3 | BOOL | Limit switch "drill in upper position" |
| 12 | IDB_Seq_drill | DB 1 | FB 1 | Instance DB of the drill sequencer |
| 13 | INIT_SQ | M 0.0 | BOOL | Sequencer parameter: INIT_SQ |
| 14 | Lower_drill | Q 0.2 | BOOL | Lower drill |
| 15 | Raise_drill | Q 0.3 | BOOL | Raise drill |
| 16 | Seq_drill | FB 1 | FB 1 | Drill sequencer |
| 17 | Start_button | I 0.7 | BOOL | Start button of the drill |

**مثال ۳: ارسال پیام به CPU برای وقوع خطا در S7- Graph**

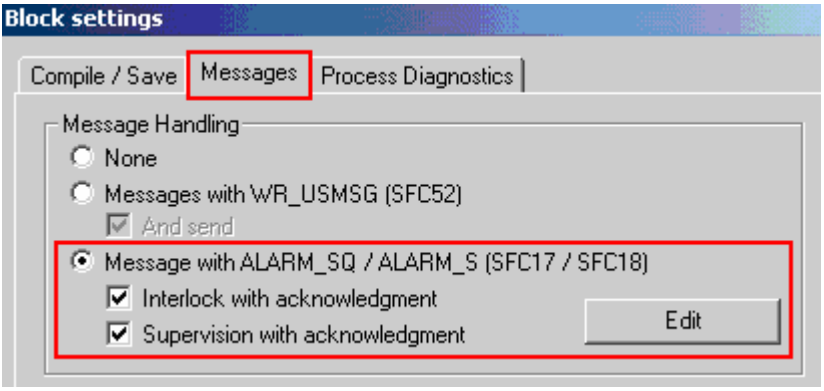
در این مثال با نحوه تنظیم و استفاده از امکانات ارسال پیام آشنا میشویم اگرچه این موضوع منحصر به S7-Graph نیست و در سایر محیط های برنامه نویسی نیز ممکن است ولی در اینجا نکات مربوط به Graph مورد بحث قرار میگیرد.برنامه همانطور که در صفحه بعد نشان داده شده متشکل از پنج Step است. در Step1 هم Supervision و هم Interlock فعال است که در شکل زیر آمده است. در سایر Step ها فقط Supervision وجود دارد.



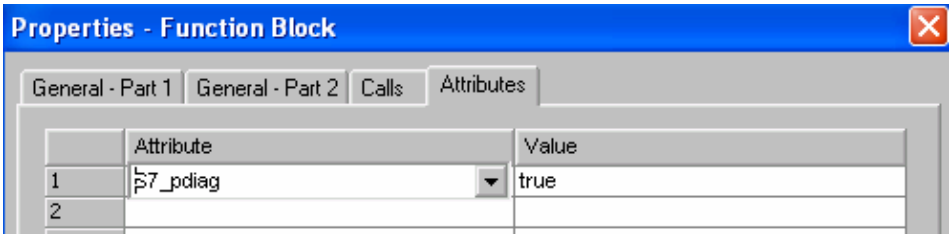

Supervision در Step2 با M3.2 و در Step3 با M3.3 و همینطور سایر Step ها به همین ترتیب فعال شده است. یعنی در Step5 با M3.5 فعال است.

هدف آنست که در هنگام بروز خطای اینترلاک یا supervision پیامی به CPU ارسال گردد. برای اینکار لازمست مراحل زیر را انجام دهیم:

۱- ابتدا در برنامه Graph از منوی Option > Block Setting مانند شکل بعد Message را فعال میکنیم . این تنظیم باعث میشود که شماره Step و پیام آن توسط SFC17/SFC18 در حافظه پیام CPU ثبت گردد.



۲- با استفاده از منوی File>Save تغییرات را ذخیره میکنیم در این وضعیت اگر به آیکون FB در پوشه Blocks از Simatic Manager نگاه کنیم مثلث زرد رنگی را در کنار آن و همینطور در کنار DB مربوط به آن مشاهده خواهیم کرد. با کلیک راست روی FB و مشاهده Properties قسمت Attribute را بصورت شکل زیر خواهیم دید:

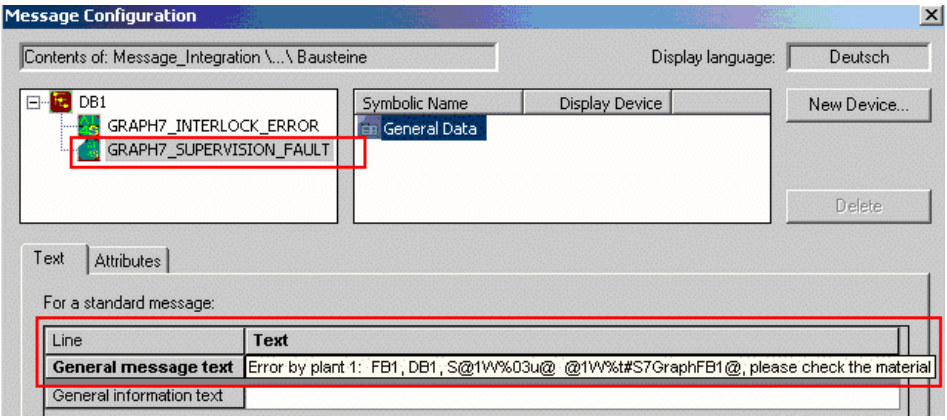



۳- مجدداً در پنجره شکل صفحه قبل روی Edit کلیک میکنیم مشاهده مینماییم که پنجره پیام مانند شکل بعد باز میشود و در جلوی اینترلاک و Supervision عبارات زیر ظاهر شده است.

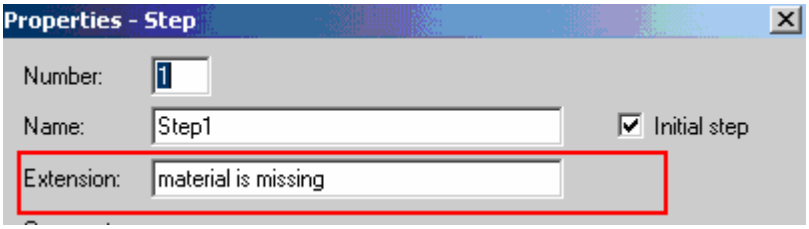
S7GRAPH Interlock Error: FB2, DB2, S@1W%03u@ @1W%t#S7GraphFB2@
S7GRAPH Supervision Error FB2, DB2, S@1W%03u@ @1W%t#S7GraphFB2@

این عبارات توسط خود سیستم ظاهر میشود و برای نوشتن شماره Step و پیام مربوط به آن بکار میرود. در عین حال میتوان به ابتدا و انتهای آن کلمات دیگری را افزود یا کلماتی را حذف کرد مانند:

Fault in Plant 1: FB2, DB2, S@1W%03u@ @1W%t#S7GraphFB2@ check the material



۴- روی Step1 ها راست کلیک کرده و با انتخاب Object Properties در بخش Extension پیام دلخواه را مانند پنجره زیر بنویسید. این کار را برای سایر step ها تکرار کنید و در هر کدام پیام دلخواه را بنویسید.

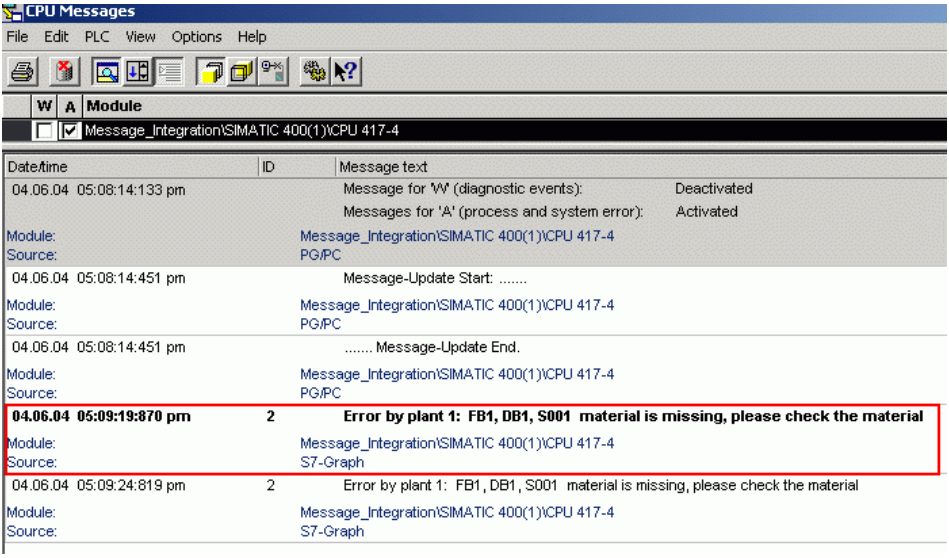



۵- با کلیک کردن روی OK و سپس ذخیره سازی برنامه در Graph اکنون اگر به Simatic Manager برگشته و از منوی Options>Text Libraries > System Text Library استفاده کنیم پنجره زیر را خواهیم دید که پیام های موجود در آن بطور اتوماتیک از Graph تولید شده اند. میتوان در همین جا پیام ها را تغییر داد و ذخیره کرد.

۶- OB1 و سایر بلاک ها را به PLC دانلود کنید سپس در حالت On Line از منوی PLC> CPU Messages در Simatic Manager استفاده کنید تا پیام هایی که در حافظه CPU ذخیره میشوند را مانند شکل صفحه بعد ملاحظه نمایید.در این پنجره گزینه A را فعال کنید تا آلارم ها نمایش داده شوند.

در این شرایط اگر بعنوان مثال وقتی برنامه در Step 1 است ورودی M3.1 مربوط به Supervision فعال شود پیغام زیر را در لیست مزبور خواهیم دید :

Fault in Plant 1: FB1,DB1,S001 No material available, check the material




بعد از برطرف شدن فالت یعنی صفر شدن M3.1 گذر اتفاق نمی افتد تا خطا Acknowledge شود برای این کار ورودیACK_EF مربوط به FB را در هنگام فراخوانی به M10.2 نسبت میدهیم تا با یک کردن آن گذر به Step بعدی اتفاق بیفتد. با M10.0 نیز توالی را Initial کرده و با M10.1 آنرا غیر فعال میکنیم.

```
CALL  FB    1 , DB1
 OFF_SQ   :=M10.0
 INIT_SQ  :=M10.1
 ACK_EF   :=M10.2
```

## ۱۴-۷ لیست دستورات در S7-Graph

| Event | Instruction | Address | شرح |
|---|---|---|---|
| | | | **دستورات استاندارد** |
| | N | Q,I,M,N | در مدتی که Step فعال است آدرس ذکر شده یک است |
| | S | Q,I,M,N | وقتی Step فعال شود آدرس ذکر شده یک میشود و یک می ماند. |
| | R | Q,I,M,N | وقتی Step فعال شود آدرس ذکر شده صفر میشود و صفر می ماند. |
| | D | Q,I,M,N<br>T#<const> | n ثانیه بعدا ز فعال شدن Step آدرس ذکر شده یک میشود و با خروج از Step صفر میشود. |
| | L | Q,I,M,N<br>T#<const> | بعدا ز فعال شدن Step آدرس ذکر شده یک میشود و پس از n ثانیه صفر میشود. |
| | CALL | FC,FB<br>SFC<br>SFB | با فعال شدن Step فراخوانی بلاک اتفاق می افتد. |
| | N C | Q,I,M,N | در مدتی که Step فعال است بشرط برآورده شدن اینترلاک آدرس ذکر شده یک است |
| | S C | Q,I,M,N | وقتی Step فعال شود بشرط برآورده شدن اینترلاک آدرس ذکر شده یک میشود و یک می ماند. |
| | R C | Q,I,M,N | وقتی Step فعال شود بشرط برآورده شدن اینترلاک آدرس ذکر شده صفر میشود و صفر می ماند. |
| | D C | Q,I,M,N<br>T#<const> | n ثانیه بعدا ز فعال شدن Step آدرس ذکر شده بشرط برآورده شدن اینترلاک یک میشود و با خروج از Step صفر میشود. |
| | L C | Q,I,M,N<br>T#<const> | بعدا ز فعال شدن Step بشرط برآورده شدن اینترلاک آدرس ذکر شده یک میشود و پس از n ثانیه صفر میشود. |
| | CALL C | FC,FB<br>SFC<br>SFB | با فعال شدن Step بشرط برآورده شدن اینترلاک فراخوانی بلاک اتفاق می افتد. |
| | | | **دستورات مبتنی بر Event** |
| S1 | N | Q,I,M,N | با ورود به Step آدرس ذکر شده یک میشود |
| S1 | S | Q,I,M,N | با ورود به Step آدرس ذکر شده یک میشود و یک می ماند. |
| S1 | R | Q,I,M,N | با ورود به Step آدرس ذکر شده صفر میشود و صفر می ماند. |
| S1 | CALL | FC,FB<br>SFC<br>SFB | با ورود به Step فراخوانی بلاک اتفاق می افتد. |
| S1 | ON | S | با ورود به Step جاری Step ذکر شده.فعال میشود |
| S1 | OFF | S | با ورود به Step جاری Step ذکر شده.غیر فعال میشود |
| S1 | OFF | S_ALL | با ورود به Step جاری تمام Step ها غیر.فعال میشوند |

| | | | |
|---|---|---|---|
| S1 | N C | Q,I,M,N | با ورود به Step بشرط برآورده شدن اینترلاک آدرس ذکر شده یک میشود |
| S1 | S C | Q,I,M,N | با ورود به Step بشرط برآورده شدن اینترلاک آدرس ذکر شده یک میشود و یک می ماند. |
| S1 | R C | Q,I,M,N | با ورود به Step بشرط برآورده شدن اینترلاک آدرس ذکر شده صفر میشود و صفر می ماند. |
| S1 | CALL C | FC,FB<br>SFC<br>SFB | با ورود به Step بشرط برآورده شدن اینترلاک فراخوانی بلاک اتفاق می افتد. |
| S1 | ON C | S | با ورود به Step جاری Step ذکر شده بشرط برآورده شدن اینترلاک.فعال میشود |
| S1 | OFF C | S | با ورود به Step جاری Step ذکر شده. بشرط برآورده شدن اینترلاک غیر فعال میشود |
| S1 | OFF C | S_ALL | با ورود به Step جاری تمام Step ها بشرط برآورده شدن اینترلاک غیر.فعال میشوند |
| S0 | N | Q,I,M,N | با خروج از Step آدرس ذکر شده یک میشود |
| S0 | S | Q,I,M,N | با خروج از Step آدرس ذکر شده یک میشود و یک می ماند. |
| S0 | R | Q,I,M,N | با خروج از Step آدرس ذکر شده صفر میشود و صفر می ماند. |
| S0 | CALL | FC,FB<br>SFC<br>SFB | با خروج از Step فراخوانی بلاک اتفاق می افتد. |
| S0 | ON | S | با خروج از Step جاری Step ذکر شده.فعال میشود |
| S0 | OFF | S | با خروج از Step جاری Step ذکر شده.غیر فعال میشود |
| V1 | N | Q,I,M,N | با وقوع خطای Supervision آدرس ذکر شده یک میشود |
| V1 | S | Q,I,M,N | با وقوع خطای Supervision آدرس ذکر شده یک میشود و یک می ماند. |
| V1 | R | Q,I,M,N | با وقوع خطای Supervision آدرس ذکر شده صفر میشود و صفر می ماند. |
| V1 | CALL | FC,FB<br>SFC<br>SFB | با وقوع خطای Supervision فراخوانی بلاک اتفاق می افتد. |
| V1 | ON | S | با وقوع خطای Supervision مرحله ذکر شده.فعال میشود |
| V1 | OFF | S | با وقوع خطای Supervision مرحله ذکر شده غیر.فعال میشود |
| V1 | OFF | S_ALL | با وقوع خطای Supervision تمام Step ها غیر.فعال میشوند |
| V1 | N C | Q,I,M,N | با وقوع خطای Supervision بشرط برآورده شدن اینترلاک آدرس ذکر شده یک میشود |
| V1 | S C | Q,I,M,N | با وقوع خطای Supervision بشرط برآورده شدن اینترلاک آدرس ذکر شده یک میشود و یک می ماند. |
| V1 | R C | Q,I,M,N | با وقوع خطای Supervision بشرط برآورده شدن اینترلاک آدرس ذکر شده صفر |

| | | | |
|---|---|---|---|
| | | | میشود و صفر می ماند. |
| V1 | CALL C | FC,FB<br>SFC<br>SFB | با وقوع خطای Supervision بشرط برآورده شدن اینترلاک فراخوانی بلاک اتفاق می افتد. |
| V1 | ON C | S | با وقوع خطای Supervision بشرط برآورده شدن اینترلاک مرحله ذکر شده.فعال میشود |
| V1 | OFF C | S | با وقوع خطای Supervision بشرط برآورده شدن اینترلاک مرحله ذکر شده غیر.فعال میشود |
| V1 | OFF C | S_ALL | با وقوع خطای Supervision بشرط برآورده شدن اینترلاک تمام Step ها غیر.فعال میشوند |
| V0 | N | Q,I,M,N | با رفع خطای Supervision آدرس ذکر شده یک میشود |
| V0 | S | Q,I,M,N | با رفع خطای Supervision آدرس ذکر شده یک میشود و یک می ماند. |
| V0 | R | Q,I,M,N | با رفع خطای Supervision آدرس ذکر شده صفر میشود و صفر می ماند. |
| V0 | CALL | FC,FB<br>SFC<br>SFB | با رفع خطای Supervision فراخوانی بلاک اتفاق می افتد. |
| V0 | ON | S | با رفع خطای Supervision مرحله ذکر شده.فعال میشود |
| V0 | OFF | S | با رفع خطای Supervision مرحله ذکر شده غیر.فعال میشود |
| L0 | N | Q,I,M,N | به محض وقوع شرایط اینترلاک آدرس ذکر شده یک میشود |
| L0 | S | Q,I,M,N | به محض وقوع شرایط اینترلاک آدرس ذکر شده یک میشود و یک می ماند. |
| L0 | R | Q,I,M,N | به محض وقوع شرایط اینترلاک آدرس ذکر شده صفر میشود و صفر می ماند. |
| L0 | CALL | FC,FB<br>SFC<br>SFB | به محض وقوع شرایط اینترلاک فراخوانی بلاک اتفاق می افتد. |
| L0 | ON | S | به محض وقوع شرایط اینترلاک مرحله ذکر شده.فعال میشود |
| L0 | OFF | S | به محض وقوع شرایط اینترلاک مرحله ذکر شده غیر.فعال میشود |
| A1 | N | Q,I,M,N | به محض Acknowledge شدن پیام آدرس ذکر شده یک میشود |
| A1 | S | Q,I,M,N | به محض Acknowledge شدن پیام آدرس ذکر شده یک میشود و یک می ماند. |
| A1 | R | Q,I,M,N | به محض Acknowledge شدن پیام آدرس ذکر شده صفر میشود و صفر می ماند. |
| A1 | CALL | FC,FB<br>SFC<br>SFB | به محض Acknowledge شدن پیام فراخوانی بلاک اتفاق می افتد. |
| A1 | ON | S | به محض Acknowledge شدن پیام مرحله ذکر شده.فعال میشود |
| A1 | OFF | S | به محض Acknowledge شدن پیام مرحله ذکر شده غیر.فعال میشود |
| A1 | N C | Q,I,M,N | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک آدرس ذکر شده یک میشود |

| | | | |
|---|---|---|---|
| A1 | S C | Q,I,M,N | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک آدرس ذکر شده یک میشود و یک می ماند. |
| A1 | R C | Q,I,M,N | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک آدرس ذکر شده صفر میشود و صفر می ماند. |
| A1 | CALL C | FC<br>FB<br>SFC<br>SFB | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک فراخوانی بلاک اتفاق می افتد. |
| A1 | ON C | S | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک مرحله ذکر شده.فعال میشود |
| A1 | OFF C | S | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک مرحله ذکر شده غیر.فعال میشود |
| R1 | N | Q,I,M,N | به محض Register شدن آدرس ذکر شده یک میشود |
| R1 | S | Q,I,M,N | به محض Register شدن آدرس ذکر شده یک میشود و یک می ماند. |
| R1 | R | Q,I,M,N | به محض Register شدن آدرس ذکر شده صفر میشود و صفر می ماند. |
| R1 | CALL | FC,FB<br>SFC<br>SFB | به محض Register شدن فراخوانی بلاک اتفاق می افتد. |
| R1 | ON | S | به محض Register شدن مرحله ذکر شده.فعال میشود |
| R1 | OFF | S | به محض Register شدن مرحله ذکر شده غیر.فعال میشود |
| R1 | N C | Q,I,M,N | به محض Register شدن و برآورده شدن شرایط اینترلاک آدرس ذکر شده یک میشود |
| R1 | S C | Q,I,M,N | به محض Register شدن و برآورده شدن شرایط اینترلاک آدرس ذکر شده یک میشود و یک می ماند. |
| R1 | R C | Q,I,M,N | به محض Register شدن و برآورده شدن شرایط اینترلاک آدرس ذکر شده صفر میشود و صفر می ماند. |
| R1 | CALL C | FC,FB<br>SFC<br>SFB | به محض Register شدن و برآورده شدن شرایط اینترلاک فراخوانی بلاک اتفاق می افتد. |
| R1 | ON C | S | به محض Register شدن و برآورده شدن شرایط اینترلاک مرحله ذکر شده.فعال میشود |
| R1 | OFF C | S | به محض Register شدن و برآورده شدن شرایط اینترلاک مرحله ذکر شده غیر.فعال میشود |
| | | | **دستورات کانترها** |
| S1 | CS | C<br>< counter value> | به محض فعال شدن Step کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |

| | | | |
|---|---|---|---|
| S1 | CU | C | به محض فعال شدن Step کانتر یک شماره افزایش می یابد |
| S1 | CD | C | به محض فعال شدن Step کانتر یک شماره کاهش می یابد |
| S1 | CR | C | به محض فعال شدن Step کانتر ری ست میشود. |
| S1 | CS C | C <initial counter value> | به محض فعال شدن Step و برآورده شدن شرایط اینترلاک کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| S1 | CU C | C | به محض فعال شدن Step و برآورده شدن شرایط اینترلاک کانتر یک شماره افزایش می یابد |
| S1 | CD C | C | به محض فعال شدن Step و برآورده شدن شرایط اینترلاک کانتر یک شماره کاهش می یابد |
| S1 | CR C | C | به محض فعال شدن Step و برآورده شدن شرایط اینترلاک کانتر ری ست میشود. |
| S0 | CS | C < counter value> | به محض غیرفعال شدن Step کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| S0 | CU | C | به محض غیرفعال شدن Step کانتر یک شماره افزایش می یابد |
| S0 | CD | C | به محض غیرفعال شدن Step کانتر یک شماره کاهش می یابد |
| S0 | CR | C | به محض غیرفعال شدن Step کانتر ری ست میشود. |
| L1 | CS | C < counter value> | به محض قطع شدن اینترلاک کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| L1 | CU | C | به محض قطع شدن اینترلاک کانتر یک شماره افزایش می یابد |
| L1 | CD | C | به محض قطع شدن اینترلاک کانتر یک شماره کاهش می یابد |
| L1 | CR | C | به محض قطع شدن اینترلاک کانتر ری ست میشود. |
| L0 | CS | C < counter value> | به محض وقوع اینترلاک کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| L0 | CU | C | به محض وقوع اینترلاک کانتر یک شماره افزایش می یابد |
| L0 | CD | C | به محض وقوع اینترلاک کانتر یک شماره کاهش می یابد |
| L0 | CR | C | به محض وقوع اینترلاک کانتر ری ست میشود. |
| V1 | CS | C < counter value> | به محض وقوع خطای Supervision کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| V1 | CU | C | به محض وقوع خطای Supervision کانتر یک شماره افزایش می یابد |
| V1 | CD | C | به محض وقوع خطای Supervision کانتر یک شماره کاهش می یابد |
| V1 | CR | C | به محض وقوع خطای Supervision کانتر ری ست میشود. |
| V1 | CS C | C < counter value> | به محض وقوع خطای Supervision و برآورده شدن شرایط اینترلاک کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |

| | | | |
|---|---|---|---|
| V1 | CU C | C | به محض وقوع خطای Supervision و برآورده شدن شرایط اینترلاک کانتر یک شماره افزایش می یابد |
| V1 | CD C | C | به محض وقوع خطای Supervision و برآورده شدن شرایط اینترلاک کانتر یک شماره کاهش می یابد |
| V1 | CR C | C | به محض وقوع خطای Supervision و برآورده شدن شرایط اینترلاک کانتر ری ست میشود. |
| V0 | CS | C < counter value> | به محض رفع خطای Supervision کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| V0 | CU | C | به محض رفع خطای Supervision کانتر یک شماره افزایش می یابد |
| V0 | CD | C | به محض رفع خطای Supervision کانتر یک شماره کاهش می یابد |
| V0 | CR | C | به محض رفع خطای Supervision کانتر ری ست میشود. |
| A1 | CS | C < counter value> | به محض وقوع خطای Supervision کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| A1 | CU | C | به محض وقوع خطای Supervision کانتر یک شماره افزایش می یابد |
| A1 | CD | C | به محض Acknowledge شدن پیام کانتر یک شماره کاهش می یابد |
| A1 | CR | C | به محض Acknowledge شدن پیام کانتر ری ست میشود. |
| A1 | CS C | C < counter value> | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| A1 | CU C | C | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک کانتر یک شماره افزایش می یابد |
| A1 | CD C | C | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک کانتر یک شماره کاهش می یابد |
| A1 | CR C | C | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک کانتر ری ست میشود. |
| R1 | CS | C < counter value> | به محض REGISTER شدن کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| R1 | CU | C | به محض REGISTER شدن کانتر یک شماره افزایش می یابد |
| R1 | CD | C | به محض REGISTER شدن کانتر یک شماره کاهش می یابد |
| R1 | CR | C | به محض REGISTER شدن کانتر ری ست میشود. |
| R1 | CS C | C <initial counter value> | به محض REGISTER شدن و برآورده شدن شرایط اینترلاک کانتر ذکر شده با مقدار اولیه داده شده ست میشود. |
| R1 | CU C | C | به محض REGISTER شدن و برآورده شدن شرایط اینترلاک کانتر یک شماره |

| | | | |
|---|---|---|---|
| | | | افزایش می یابد |
| R1 | CD C | C | به محض REGISTER شدن و برآورده شدن شرایط اینترلاک کانتر یک شماره کاهش می یابد |
| R1 | CR C | C | به محض REGISTER شدن و برآورده شدن شرایط اینترلاک کانتر ری ست میشود. |
| | | | **دستورات تایمرها** |
| S1 | TL | T <time> | به محض فعال شدن Step تایمر با زمان مشخص شده بکار می افتد |
| S1 | TD | T <time> | به محض فعال شدن Step تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| S1 | TR | T | به محض فعال شدن Step تایمر ری ست می شود. |
| S1 | TL C | T <time> | به محض فعال شدن Step و برآورده شدن اینترلاک تایمر با زمان مشخص شده بکار می افتد |
| S1 | TD C | T <time> | به محض فعال شدن Step و برآورده شدن اینترلاک تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| S1 | TR C | T | به محض فعال شدن Step و برآورده شدن اینترلاک تایمر ری ست می شود. |
| S0 | TL | T <time> | به محض غیرفعال شدن Step تایمر با زمان مشخص شده بکار می افتد |
| S0 | TD | T <time> | به محض غیر فعال شدن Step تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| S0 | TR | T | به محض غیر فعال شدن Step تایمر ری ست می شود. |
| L1 | TL | T <time> | به محض قطع شدن اینترلاک تایمر با زمان مشخص شده بکار می افتد |
| L1 | TD | T <time> | به محض قطع شدن اینترلاک تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| L1 | TR | T | به محض قطع شدن اینترلاک تایمر ری ست می شود. |
| L0 | TL | T <time> | به محض وقوع اینترلاک تایمر با زمان مشخص شده بکار می افتد |
| L0 | TD | T <time> | به محض وقوع اینترلاک تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| L0 | TR | T | به محض وقوع اینترلاک تایمر ری ست می شود. |
| V1 | TL | T <time> | به محض وقوع خطای Supervision تایمر با زمان مشخص شده بکار می افتد |
| V1 | TD | T <time> | به محض وقوع خطای Supervision تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| V1 | TR | T | به محض وقوع خطای Supervision تایمر ری ست می شود. |

| | | | |
|---|---|---|---|
| V1 | TL C | T <time> | به محض وقوع خطای Supervision و برآورده شدن شرایط اینترلاک تایمر با زمان مشخص شده بکار می افتد |
| V1 | TD C | T <time> | به محض وقوع خطای Supervision و برآورده شدن شرایط اینترلاک تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| V1 | TR C | T | به محض وقوع خطای Supervision و برآورده شدن شرایط اینترلاک تایمر ری ست می شود. |
| V0 | TL | T <time> | به محض رفع خطای Supervision تایمر با زمان مشخص شده بکار می افتد |
| V0 | TD | T <time> | به محض رفع خطای Supervision تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| V0 | TR | T | به محض رفع خطای Supervision تایمر ری ست می شود. |
| A1 | TL | T <time> | به محض Acknowledge شدن پیام تایمر با زمان مشخص شده بکار می افتد |
| A1 | TD | T <time> | به محض Acknowledge شدن پیام تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| A1 | TR | T | به محض Acknowledge شدن پیام تایمر ری ست می شود. |
| A1 | TL C | T <time> | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک تایمر با زمان مشخص شده بکار می افتد |
| A1 | TD C | T <time> | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| A1 | TR C | T | به محض Acknowledge شدن پیام و برآورده شدن شرایط اینترلاک تایمر ری ست می شود. |
| R1 | TL | T <time> | به محض REGISTER شدن تایمر با زمان مشخص شده بکار می افتد |
| R1 | TD | T <time> | به محض REGISTER شدن تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| R1 | TR | T | به محض REGISTER شدن تایمر ری ست می شود. |
| R1 | TL C | T <time> | به محض REGISTER شدن و برآورده شدن شرایط اینترلاک تایمر با زمان مشخص شده بکار می افتد |
| R1 | TD C | T <time> | به محض REGISTER شدن و برآورده شدن شرایط اینترلاک تایمر به اندازه زمان مشخص شده صبر میکند و پس از آن یک میشود |
| R1 | TR C | T | به محض REGISTER شدن و برآورده شدن شرایط اینترلاک تایمر ری ست می شود. |

# فصل پانزدهم - افزونگی نرم افزاری
# Software Redundancy

**مشتمل بر :**


**۱۵-۱ مقدمه**

**۱۵-۲ عملکرد کلی در افزونگی نرم افزاری**

**۱۵-۳ اصول کاربرد افزونگی نرم افزاری**

**۱۵-٤ بلاک های ویژه**

**۱۵-۵ نکات مهم در افزونگی نرم افزاری**

**۱۵-٦ جابجایی بین سیستم و اجزای آن**

**۱۵-۷ شبکه های قابل استفاده جهت اتصال دو سیستم اصلی و پشتیبان**

**۱۵-۸ تغییر پیکربندی یا برنامه کاربردی در مد RUN**

**۱۵-۹ مدول های مناسب جهت استفاده در یک سیستم مبتنی بر افزونگی نرم افزاری**

**۱۵-۱۰ ارتباط با سایر سیستم ها**

**۱۵-۱۱ جایگاه وضعیت Standby در افزونگی نرم افزاری**

**۱۵-۱۲ استفاده از OB های مربوط به مدیریت خطا**

**۱۵-۱۳ مثالی از پیاده سازی افزونگی نرم افزاری با استفاده از S7-300**

### ۱۵-۱ مقدمه

نقص در عملکرد یک سیستم اتوماسیون که به دلایل مختلف (مثلا اشکال در CPU) ممکن است بروز کند همیشه پیامدهای نامطلوب و هزینه های ناشی از خسارت به همراه دارد. بنابراین همواره در انتخاب سیستم اتوماسیون جهت کنترل یک فرآیند میزان **"دسترسی"** آن سیستم به عنوان یک معیار مهم مد نظر قرار می گیرد.

در برخی از سیستم های اتوماسیون گستردگی فرآیند تحت کنترل و ماهیت آن در حدی نیست که لازم باشد از سیستم های کنترلی که به کمک افزونگی سخت افزاری سطح بالایی از دسترسی را تامین می کنند و در فصل ۱۶ تشریح شده اند استفاده نمود. معمولا در چنین فرآیندهایی یک مکانیزم نرم افزاری جهت تغییر کنترل به هنگام وقوع خطا از سیستم اصلی به سیستم پشتیبان به کار میرود.

### موارد کاربرد

افزونگی نرم افزاری(Software Redundancy) در اتوماسیون فرآیندهایی قابل استفاده است که ماهیت فرآیند به گونه ای باشد که اگر زمان تغییر کنترل از سیستم اصلی به سیستم رزرو در حد چند ثانیه باشد و در این مدت کنترل از دست برود مشکلی برای فرآیند رخ ندهد. به عنوان مثال فرآیندهای زیر از این دسته اند :

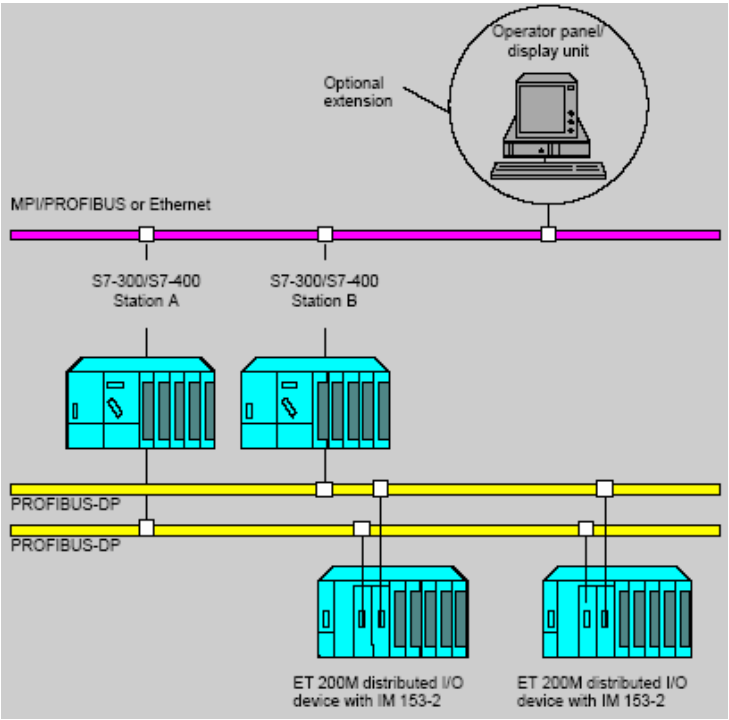
- سیستم تصفیه آب آشامیدنی
- کنترل ترافیک
- کنترل دما در کوره ها
- کنترل دما در سردخانه ها

### خطاهای قابل کنترل توسط افزونگی نرم افزاری

- اشکال سخت افزاری یا نرم افزاری در عملکرد CPU
- اشکال در عملکرد اجزای مرتبط با CPU (منبع تغذیه , DP Master و باس که از طریق آن CPU و سایر اجزای PLC به هم متصل می شوند)
- اشکال در واحد PROFIBUS یک واسط slave که در بخش پشتیبان از یک سیستم افزونه به کار رفته است (مثل IM 153-3 )
- قطع کابل باس در لینک یا واسط DP slave پشتیبان در یک سیستم افزونه

## نیازمندی های سخت افزاری

- دو سیستم S7-300 یا S7-400 (یا یک S7-300 و یک S7-400) که از طریق یک باس باهم تبادل داده می کنند
- در هر سیستم باید یک CPU و نیز واسط اتصال به سیستم DP Master وجود داشته باشد
- ارتباط هر یک از این سیستم ها با تجهیزات از طریق DP Master ای که برای هر سیستم وجود دارد صورت میگیرد
- واحد های ورودی / خروجی توزیع شده ET 200M همراه با واسط های پشتیبان و افزونه DP Slave به **دو** سیستم DP Master متصل میشوند

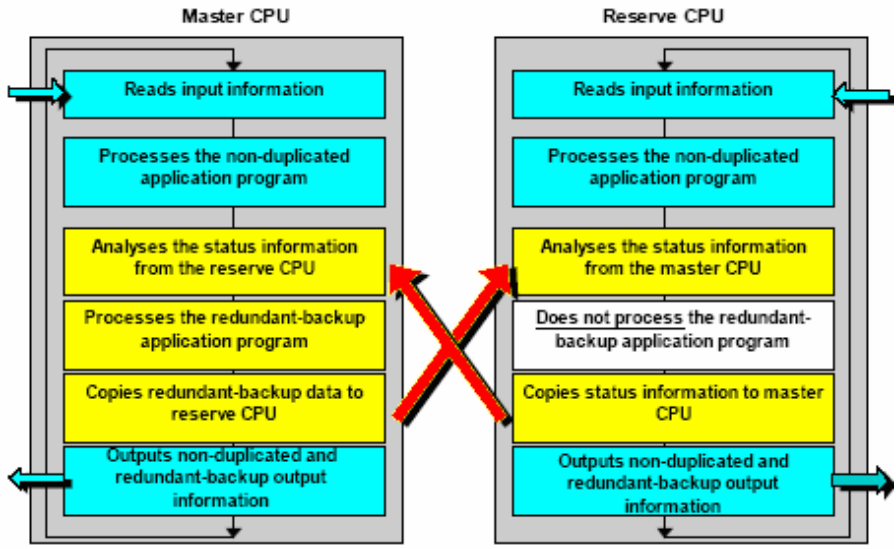



## نیازمندی های نرم افزاری

- نرم افزار STEP7 نسخه 5.2 یا بالاتر
- استفاده های از بلاک های نرم افزاری ویژه موجود در بسته نرم افزاری Redundant-Backup Software

## ۱۵-۲ عملکرد کلی در افزونگی نرم افزاری

شکل زیر روند پیاده سازی افزونگی نرم افزاری را نشان می دهد. در این روش بخش مهم برنامه PLC که اجرای دائم آن حیاتی است در هر دو CPU قرار داده میشود. البته برنامه در حالت عادی فقط در CPU اصلی اجرا میشود و CPU رزرو در حالت آماده قرار گرفته و دائما داده های جاری حاصل از اجرای برنامه را ازCPU اصلی دریافت می کند تا چنانچه CPU اصلی دچار مشکل شود اجرای برنامه را ادامه دهد و کنترل فرآیند را به دست گیرد. این حالت همان ویژگی Warm Standby می باشد و فرق آن با Hot Standby این است که در حالت دوم (که در سیستم های سری H به کار میرود) هر دو CPU همزمان در حال اجرای برنامه می باشند.




## ۱۵-۳ اصول کاربرد افزونگی نرم افزاری

### سخت افزار

- واحدهای ET200M مورد استفاده در هر دو سیستم که برای پیاده سازی افزونگی استفاده شده اند و دارای واسط DP Slave می باشند باید به طور کاملا یکسان ترکیب بندی شوند. این کار از طریق گزینه Edit > Insert Redundant Copy در برنامه HWconfig قابل انجام است .
- به هنگام ترکیب بندی سخت افزار دو سیستم باید از نواحی همجوار در حافظه استفاده کرد یعنی مثلا همه ورودی ها کنار هم آدرس دهی شوند .

- فقط یک سیستم PROFIBUS-DP می تواند به عنوان Master باشد
- نرخ انتقال داده روی واسط های DP Slave ای که برای پیاده سازی افزونگی استفاده شده اند 187.5 K تا 12 M است.

**نرم افزار**

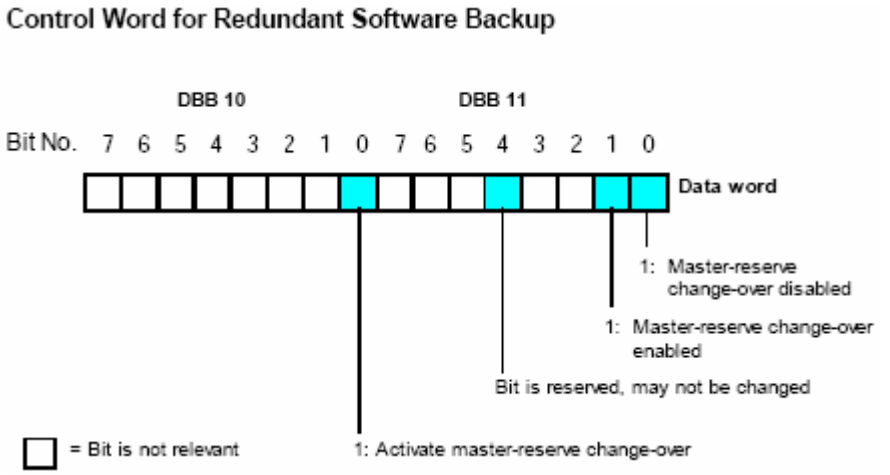
- اگر برنامه سیستم کنترل به نحوی است که فقط باید **بخشی** از آن به هنگام توقف سیستم اصلی در سیستم رزرو اجرا گردد باید این بخش از برنامه را به صورت کاملا مستقل و جدا از بخش های دیگر برای دو سیستم برنامه نویسی کرد. برای این کار میتوان از OB های مختلف برای بخش های مختلف برنامه استفاده نمود. (مثلا OB1 و OB35)
- FB101 , FB103 , FB104 , FB105, FC100, FC101, FC102 بلاک های خاص افزونگی نرم افزاری هستند.
- در ابتدا و انتهای برنامه FB 101 فراخوانی می شود : بار اول پارامتر CALL_POSITION برابر True و بار دوم برابر False است.
- چنانچه ارتباط دو سیستم اصلی و رزرو از طریق یکی از پروتکل های ارتباطی S7 (مثلا MPI یا Profibus ) است و در عین حال از این واسط برای کاربردهای ارتباطی دیگر نیز استفاده می شود باید در این کاربردها پارامتر R_ID را برابر عددی بزرگتر از ۲ قرار داد زیرا مقادیر ۱و۲ توسط نرم افزار افزونه استفاده می شود.
- اگر در برنامه از بلاک FB103 برای ارتباطات استفاده شود نرم افزار افزونه از بلاک های ارتباطی SFC65 و SFC66 با R_ID > 8000 0000**H** استفاده خواهد کرد.
- اگر در برنامه از بلاک FB104 برای ارتباطات استفاده شود نرم افزار افزونه از بلاک های ارتباطی FC6 و FC5 با R_ID > 8000 0000**H** استفاده خواهد کرد.
- اگر در برنامه از بلاک FB105 برای ارتباطات استفاده شود باید در هنگام تنظیم پارامترهای مربوط به ارتباطات پارامتر Send operating status messages برابر Yes قرار داده شود تا قطع ارتباط به سرعت قابل تشخیص باشد.
- زمان سنج ها و شمارنده های S7 را نمی توان در بخش افزونه برنامه استفاده کرد و به جای آنها باید از زمان سنج ها و شمارنده های IEC استفاده نمود.
- همه SFC ها و SFB های مورد استفاده در نرم افزار افزونه باید در پروژه S7 قرار داده شوند.

- اگر پیکربندی بلاک Startup تغییر کند باید بلاک های زیر حذف شوند تا پارامترهای جدید مطابق با تغییرات انجام شده تنظیم شوند.

| | |
|---|---|
| DB_WORK_NO | Working DB |
| DB_SEND_NO | Send DB |
| DB_RCV_NO | Receive DB |
| DB_A_B_NO | DB مربوط به تبادل داده بین بخش غیر افزونه سیستمA و نرم افزار افزونه |
| DB_B_A_NO | DB مربوط به تبادل داده بین بخش غیر افزونه سیستمB و نرم افزار افزونه |

- هیچ متغیری نباید در ۲۰ بایت اول از متغیرهای محلی OB86 قرار داده شود زیرا این ناحیه مورد استفاده نرم افزار افزونه است.
- چنانچه پارامترهای خروجی که جزو PIQ(Process Image Output table) نیستند در بلاک FC100 تعریف شوند خطای "دسترسی به واحدهای ورودی / خروجی" رخ خواهد داد.

عملکرد دو سیستم اصلی و رزرو ممکن است پس از آنکه انتخاب یکی از این دو با استفاده از بیت کنترل صورت گرفت به درستی انجام نگردد و در صورت بروز مشکل باید مجددا توسط همان بیت، کنترل به سیستم سالم بازگردانده شود




- در زمان سوئیچ بین دو سیستم اصلی و رزرو موقتا هر دو سیستم اصلی یا رزرو خواهند بود.
- اگر یک CPU در حالت Stop باشد و دیگری در حالت Run , این احتمال وجود دارد که واسط فعال DP Slave ای که در سیستم افزونه وجود دارد و دوباره راه اندازی و تنظیم شده

است به CPU ای که Stop است اختصاص داده شود که این مسئله سیستم را دچار اشکال میکند . برای جلوگیری از بروز این مشکل باید یک CPU خاموش ( Power Off) باشد.

## ۱۵-٤ بلاک های ویژه

پس از نصب بسته نرم افزاری مربوط به پشتیبانی از افزونگی نرم افزاری ، یک کتابخانه به نام SWR_LIB در STEP 7 ایجاد می شود که از طریق فرمان **File > Open > Libraries** در محیط STEP 7 قابل دسترس خواهد بود. این کتابخانه شامل ۵ دسته بلاک می باشد که دو تا از آنها برای استفاده در سیستم های مبتنی بر S7-300 و دو دسته دیگر مربوط به S7-400 است. انتخاب این بلاک ها در هر سیستم ، مطابق آنچه در جدول زیر آمده است به نوع ارتباط و شبکه ای که دو سیستم افزونه از طریق آن به هم متصل هستند بستگی دارد.

**دسته بلاک های مربوط به S7-300**

| نام دسته | نوع شبکه | نوع ارتباط | نحوه اتصال به شبکه |
|---|---|---|---|
| XSEND_300 | MPI | ثابت و غیر قابل تغییر | از طریق واسط MPI موجود در CPU |
| AG_SEND_300 | PROFIBUS | FDL | از طریق CP342-5 |
| | Industrial Ethernet | ISO | از طریق CP345-1 |

**دسته بلاک های مربوط به S7-400**

| نام دسته | نوع شبکه | نوع ارتباط | نحوه اتصال به شبکه |
|---|---|---|---|
| XSEND_400 | MPI | ثابت و غیر قابل تغییر | از طریق واسط MPI موجود در CPU |
| AG_SEND_400 | PROFIBUS | FDL | از طریق CP443-5 |
| | Industrial Ethernet | ISO | از طریق CP443-1 |
| BSEND_400 | MPI | S7 | از طریق واسط MPI موجود در CPU |
| | PROFIBUS | | از طریق CP443-5 |
| | Industrial Ethernet | | از طریق CP443-1 |

هر دسته شامل چهار بلاک است که این بلاک ها در ترکیب با یک دیگر به کار می روند ولی بلاک های دسته های مختلف قابل ترکیب نیستند و این کار منجر به بروز خطا در عملکرد سیستم می گردد.

### محتویات دسته XSEND_300 و XSEND_400

| نام بلاک | ملاحظات |
|---|---|
| FC 100 'SWR_START' | این بلاک می بایست در OB100 فراخوانی شود |
| FB 101 'SWR_ZYK' | این بلاک باید توسط برنامه ای که به طور چرخشی و یا تحت کنترل زمان سنج ها اجرا میشود فراخوانی گردد و این فراخوانی باید قبل و بعد از اجرای برنامه افزونه صورت گیرد. |
| FC 102 'SWR_DIAG' | این بلاک می بایست در OB86 فراخوانی شود |
| FB 103 'SWR_SFCCOM' | این بلاک انتقال داده را پشتیبانی می کند و غیر مستقیم توسط FB 101 فراخوانی میشود بنابراین کافی است روی هر دو CPU بارگذاری شود. |

### محتویات دسته AGSEND_300 و AGSEND_400

| نام بلاک | ملاحظات |
|---|---|
| FC 100 'SWR_START' | این بلاک می بایست در OB100 فراخوانی شود |
| FB 101 'SWR_ZYK' | این بلاک باید توسط برنامه ای که به طور چرخشی و یا تحت کنترل زمان سنج ها اجرا میشود فراخوانی گردد و این فراخوانی باید قبل و بعد از اجرای برنامه افزونه صورت گیرد. |
| FC 102 'SWR_DIAG' | این بلاک می بایست در OB86 فراخوانی شود |
| FB 104 'SWR_AG_COM' | این بلاک انتقال داده را پشتیبانی می کند و غیر مستقیم توسط FB 101 فراخوانی میشود بنابراین کافی است روی هر دو CPU بارگذاری شود.<br>این بلاک به نوبه خود به طور غیرمستقیم FC5 و FC6 را فراخوانی میکند که هر دو متعلق به NCM S7 هستند و باید روی هر دو CPU بارگذاری شوند. |

### محتویات دسته BSEND_400

| نام بلاک | ملاحظات |
|---|---|
| FC 100 'SWR_START' | این بلاک می بایست در OB100 فراخوانی شود |
| FB 101 'SWR_ZYK' | این بلاک باید توسط برنامه ای که به طور چرخشی و یا تحت کنترل زمان سنج ها اجرا میشود فراخوانی گردد و این فراخوانی باید قبل و بعد از اجرای برنامه افزونه صورت گیرد. |
| FC 102 'SWR_DIAG' | این بلاک می بایست در OB86 فراخوانی شود |
| FB 105 'SWR_SFBCOM' | این بلاک انتقال داده را پشتیبانی می کند و غیر مستقیم توسط FB 101 فراخوانی میشود بنابراین کافی است روی هر دو CPU بارگذاری شود. |

## شرح تک تک بلاک ها

**FC 100**

- برای راه اندازی هر دو سیستم افزونه استفاده میشود.
- نواحی مربوط به بخش های زیر که در نرم افزار افزونه استفاده میشوند را تعیین می نماید به نحوی که پیوستگی در هر بخش رعایت شود.

  - Outputs
  - Bit memory Address
  - Data blocks
  - Instance DB for the IEC counters/timers

- جزئیات و تنظیمات مربوط به ارتباطات و نیز مدول های محلی (Local) را مشخص میکند.
- سه DB را که بلاک های پشتیبان نرم افزار افزونه جهت ذخیره سازی داده های Internal خود به آنها نیاز دارند را تعیین می کند.
- بایستی توسط OB100 فراخوانی شود.
- وقفه پذیر است.

### نکات مهم

- در صورت عدم استفاده از یک ناحیه خاص باید پارامتر مربوط به آن برابر **صفر** مقداردهی گردد. به عنوان مثال اگر از هیچکدام از زمان سنج ها و شمارنده های IEC استفاده نشود باید به ترتیب زیر عمل شود :

  IEC_NO = 0
  IEC_LEN = 0.

- چنانچه از بلاک های DB_A_B_NO یا DB_B_A _NO استفاده نمی شود باید یک DB (با هر شماره ای غیر از صفر) معین شده و طول آن معادل **صفر** قرار داده شود. مثلا اگر DB_A_B_NO در برنامه مورد استفاده قرار نگیرد :

  DB_A_B_NO =DB 255
  DB_A_B_NO_LEN = W#16#0

- بلاک های DB_SEND_NO و DB_RCV_NO نیز همانند بلاک های DB_A_B_NO و DB_B_A _NO باید در هر دو سیستم افزونه شماره های یکسانی داشته باشند.

## پارامترها

| Parameter | Decl. | Data Type | Description | Example |
|---|---|---|---|---|
| AG_KENNUNG | IN | CHAR | Station ID ‘A’ for station A. ‘B’ for s tation B. | ‘A’ |
| DB_WORK_NO | IN | Block-DB | Working DB for redundant software backup. internal data only | DB1 |
| DB_SEND_NO | IN | Block-DB | DB in which data to be sent to communication peer is collected. Contains internal data only. | DB2 |
| DB_RCV_NO | IN | Block-DB | DB in which the CPU collects the data received from communication peer, Contains internal data only. | DB3 |
| MPI_ADR | IN | INT | MPI address of ***communication peer*** | 4 |
| LADDR | IN | INT | Logical base address of communication processor (in HWconfig). | 260 |
| VERB_ID | IN | INT | Connection ID Number of connection for redundant -backup link (specified in connection configuration) | 1 |
| DP_MASTER_SYS_ ID | IN | INT | DP master system ID . ID of DP master system to which the ET 200M slaves are connected (in hwconfig) | 1 |
| DB_COM_NO | IN | Block-DB | Instance DB of FB 101 ‘SWR_ZYK’. | DB5 |
| DP-KOMMUN | IN | INT | Identification number of DP master<br>1 for DP master is a CPU with integral DP interface<br>2 for DP master is a CP. | 1 |
| ADR_MODUS | INT | INT | Increment size for the matrix in which the CPU allocates the I/O addresses (address matrix is CPU-dependent).<br>1, if base addresses 0, 1, 2, 3 ... 4, if base addresses 0, 4, 8, 12 ... | 1 |
| PAA_FIRST | IN | INT | Number of first output byte used by an ET 200M with redundant-backup IM 153. | 0 |
| PAA_LAST | IN | INT | Number of last output byte used by an ET 200M with redundant -backup IM 153. Output bytes in the range PAA_FIRST to PAA_LAST must form a contiguous range and may only be used by the ET 200Ms with redundant-backup IM 153s | 4 |
| MB_NO | IN | INT | Number of first memory byte used by redundant-backup appl program | 20 |
| MB_LEN | IN | INT | Total number of memory bytes used by redundant-backup application program. Memory bytes must be allocated contiguously | 30 |
| IEC_NO | IN | INT | Number of first instance DB for IEC counters/timers used by redundant-backup application program. | 111 |
| IEC_LEN | IN | INT | Total number of instance DBs for IEC counters/timers used by redundant-backup application program. Instance DBs must be allocated contiguously | 7 |
| DB_NO | IN | INT | Number of first data block used by redundant- backup application program. | 8 |
| DB_NO_LEN | IN | INT | Total number of data blocks used by redundant-backup Appl. program Data blocks must be allocated contiguously. | 2 |
| SLAVE_NO | IN | INT | Lowest PROFIBUS address used for an ET 200M DP slave with redundant-backup IM153 | 3 |
| SLAVE_LEN | IN | INT | Total number of ET 200M DP slaves used. PROFIBUS addresses must be allocated contiguously | 1 |
| SLAVE_DISTANCE | IN | INT | Identifier for IM 153-2 PROFIBUS address Setting<br>1, if both interfaces have same PROFIBUS address<br>2, if interfaces have PROFIBUS addresses n and n+1 | 1 |
| DB_A_B_NO | IN | Block-DB | Send DB for non-duplicated data sent from station A to B | DB11 |
| DB_A_B_NO_LEN | IN | WORD | Number of data bytes used in DB_A_B_NO. | W#16#6 4 |
| DB_B_A_NO | IN | Block-DB | Send DB for non-duplicated data sent from station B to A | DB12 |
| DB_B_A_NO_LEN | IN | WORD | Number of data bytes used in DB_B_A_NO. | W#16#6 4 |
| RETURN_VAL | OUT | WORD | Block return value | MW2 |
| EXT_INFO | OUT | WORD | Return value of a lower-level block | MW4 |

### مقادیر RETURN_VAL و EXT_INFO

| Error Code | Explanation |
|---|---|
| W#16#0 | No error. |
| W#16#8001 | Illegal value for parameter Teil-AG-Kennung. |
| W#16#8002 | DB_WORK_NO could not be generated. Cause identifiable from return value of SFC 22. Return value stored in EXT_INFO. |
| W#16#8003 | DB_SEND_NO could not be generated. Cause identifiable from return value of SFC 22. Return value stored in EXT_INFO. |
| W#16#8004 | DB_RCV_NO could not be generated. Cause identifiable from return value of SFC 22. Return value stored in EXT_INFO. |
| W#16#8005 | DB_A_B_NO could not be generated. Cause identifiable from return value of SFC 22. Return value stored in EXT_INFO. |
| W#16#8006 | DB_B_A_NO could not be generated. Cause identifiable from return value of SFC 22. Return value stored in EXT_INFO. |
| W#16#8007 | Illegal value for parameter DP_MASTER_SYS_ID or SLAVE_NO or SLAVE_LEN or SLAVE_DISTANCE. Specified value does not match hardware configuration. |
| W#16#8008 | Illegal value for parameter DP-KOMMUN if EXT_INFO=W#16#8888 or diagnosis could not be performed. Cause identifiable from return value of SFC 51. Return value stored in EXT_INFO. |
| W#16#8009 | Changeover lock for slaves could not be disabled. Cause identifiable from return value of SFC 58. Return value stored in EXT_INFO. |
| W#16#800A | Status of DP slave interface could not be determined, Cause identifiable from return value of SFC 59. Return value stored in EXT_INFO. |
| W#16#800B | Error determining the PAA area used. Cause identifiable from return value of SFC 50. Return values stored in EXT_INFO. |
| W#16#800C | Illegal value for parameter ADR_MODUS. |
| W#16#800D | Illegal value for parameter SLAVE_DISTANCE. |
| W#16#800E | DB_WORK_NO can not be read. Reload blocks. |
| W#16#800F | Illegal value for parameter DP_KOMMUN (no interfaces specified). |
| W#16#80F1 | Error determining the addresses of the PAA. Cause identifiable from return value of SFC 50. Return value stored in EXT_INFO.<br>Details specified for PAA_FIRST and PAA_LAST do not match hardware configuration. |
| W#16#8027 | Internal error. |

**FB 101**

- به منظور راه اندازی انتقال اطلاعات از سیستم اصلی به سیستم پشتیبان مورد استفاده قرار می گیرد.
- باید **قبل و بعد** از برنامه افزونه فراخوانی شود.
- به هنگام فراخوانی عملیات انتقال داده به صورت اتوماتیک انجام می شود و به نوبه خود بلاک های لازم جهت این کار فراخوانی می شوند.
- وقفه پذیر است.
- وقتی این بلاک فراخوانی می شود باید یک Instance DB تعیین شود که شماره آن بایستی به هنگام تعریف پارامترهای FC100 در پارامتر DB_COM_NO مشخص شده باشد.

## پارامترها

| Parameter | Decl. | Data Type | Description | Example |
|---|---|---|---|---|
| DB_WORK_NO | IN | Block-DB | Working DB; details must be identical to those specified in parameter DB_WORK_NO of FC 100 ‘SWR_START’. | DB1 |
| CALL_POSITION | IN | BOOL | This parameter indicates at what point in the application program FB 101 ‘SWR_ZYK‘ is invoked.<br>TRUE if it is invoked before the redundant-backup application program<br>FALSE if it is invoked after the redundant-backup application program | TRUE |
| RETURN_VAL | OUT | WORD | Block return value | MW6 |
| EXT_INFO | OUT | WORD | Return value of a lower-level block | MW8 |

## مقادیر RETURN_VAL و EXT_INFO

| Error Code | Explanation |
|---|---|
| W#16#0 | No error |
| W#16#8008 | Illegal value for parameter DP-KOMMUN if EXT_INFO=W#16#8888 or diagnosis could not be performed. Cause identifiable from return value of SFC 51. |
| W#16#800A | Status of DP slave interface could not be determined. Cause identifiable from return value of SFC 59. Return value stored in EXT_INFO. |
| W#16#800F | Illegal value for parameter DP_KOMMUN (no interfaces specified). |
| W#16#8010 | Changeover of DP slave could not be performed. Cause identifiable from return value of SFC 58. Return value stored in EXT_INFO. |
| W#16#8011 | Connection can not be established. Teil-AG-Kennung invalid. |
| W#16#8012 | No job present in communication FB (FB 103 ‘SWR_SFBCOM’), (instance DB defective or internal error). |
| W#16#8013 | Error encountered when sending (FB 103 ‘SWR_SFBCOM’, FB 104 ‘SWR_AG_COM’, FB 105 ‘SWR_SFCCOM’). Cause identifiable from return value of SFC 65 ‘X_SEND’/FC 5 ‘AG_SEND’/SFB 12 ‘BSEND’. Return value stored in EXT_INFO. |
| W#16#8014 | Error encountered when receiving (FB 103 ‘SWR_SFBCOM’, FB 104 ‘SWR_AG_COM’, FB 105 ‘SWR_SFCCOM’). Cause identifiable from return value of SFC 66 ‘X_RCV’/FC 5 ‘AG_RCV’/SFB 13 ‘BRCV’. Return value stored in EXT_INFO. |
| W#16#8015 | Redundant-backup link failure. Check hardware. |
| W#16#8016 | Communication peer status can not be read (FB 103 ‘SWR_SFBCOM’). Cause identifiable from return value of SFB 23 ‘USTATUS’. Return value stored in EXT_INFO. |
| W#16#8017 | All DP slaves have failed. |
| W#16#8018 | Not possible to write to Send DB (FB 104 ‘SWR_AG_COM’, FB 105 SWR_SFCCOM’). Cause identifiable from return value of SFC 20. Return value stored in EXT_INFO. |
| W#16#8019 | Not possible to read Receive DB (FB 104 ‘SWR_AG_COM’, FB 105 SWR_SFCCOM’). |
| W#16#8020 | Internal error. |

**FC 102**

- توسط OB86 فراخوانی می شود.
- شماره این بلاک نباید تغییر داده شود.
- این بلاک تضمین می کند که به محض وقوع اشکال در یک DP Slave تغییر اتوماتیک از سیستم اصلی به سیستم پشتیبان انجام می شود.
- وقفه پذیر است.

## پارامترها

| Parameter | Decl. | Data Type | Description | Example |
|---|---|---|---|---|
| DB_WORK | IN | INT | Number of working DB for redundant software backup. Must be identical with that specified for parameter DB_WORK_NO of FC 100 'SWR_START'. DB contains internal data only. | 1 |
| OB 86_EV_CLASS | IN | INT | Startup information from diagnostic OB, OB 86. Copy the variable from the declaration table of OB 86. | #OB86_EV_CLASS |
| OB 86_FLT_ID | IN | INT | Startup information from diagnostic OB, OB 86. Copy the variable from the declaration table of OB 86. | #OB86_FLT_ID |
| RETURN_VAL | OUT | WORD | Block return value | MW14 |

### مقادیر RETURN_VAL و EXT_INFO

| Error Code | Explanation |
|---|---|
| W#16#0 | No error. |
| W#16#80F2 | Illegal value for one of the parameters of FC 102 'SWR_DIAG'. |
| W#16#80F3 | More DP slaves present than specified in FC 100 'SWR_START'. Check parameter SLAVE_NO or SLAVE_LEN. |

**FB 103 ،FB 104 و FB 105**

- شماره این بلاک ها نباید تغییر پیدا کند.
- به طور غیر مستقیم توسط بلاک FC 101 فراخوانی شده و عملیات انتقال داده از سیستم اصلی به سیستم پشتیبان را سازمان دهی می کنند.
- بلاک های لازم باید حتما درCPU هر دو سیستم افزونه بارگذاری شده باشند.
- چنانچه در پروژه از FB 104 استفاده شود باید FC5 و FC6 نیز بدون آنکه شماره آنها تغییر کند در پروژه موجود باشند.

**DB_RCV_NO ، DB_SEND_NO و DB_WORK_NO**

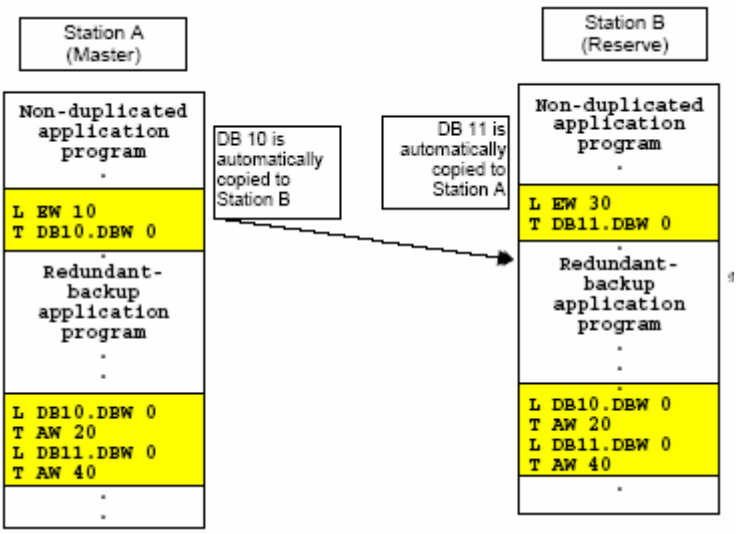
- منحصرا جهت ذخیره سازی داده های داخلی مورد استفاده قرار می گیرند.
- فقط یک بار آنهم در زمان Startup توسط FC100 و با طول تعیین شده در این FC ایجاد می شوند
- در صورت تغییر پارامترهای FC100 طبیعتا تغییراتی در DB ها نیز باید انجام شود. به همین دلیل پس از تغییر FC100 باید DB های قبلی حذف شوند تا مجددا در زمان Startup، DB های جدید با طول لازم و مشخص شده ایجاد گردند. در غیر این صورت (حذف نکردن DB ها) خطا در عملکرد سیستم رخ خواهد داد.

**DB_B_A و DB_A_B**

- طول این دو بلاک توسط پارامترهای FC100 قابل تنظیم است.
- وظیفه این بلاک ها این است که دو سیستم افزونه بتوانند داده های غیر افزونه خود را نیز مبادله کنند. (مثلا وضعیت یک مدول ورودی که فقط در Rack مرکزی سیستم A واقع شده است) بدین ترتیب دسترسی هر دو سیستم به داده های یکسان تضمین می گردد و بخش افزونه برنامه می تواند با بخش های غیر افزونه تبادل اطلاعات کند.
  - مثال :
    1. در Rack مرکزی سیستم A یک مدول غیر افزونه با آدرس EW 10 وجود دارد.
    2. در Rack مرکزی سیستم B یک مدول غیر افزونه با آدرس EW 30 وجود دارد.
    3. وضعیت هر کدام از این مدول ها در هر سیستم باید به سیستم دیگر منتقل شود تا در آنجا توسط برنامه افزونه با استفاده از آدرس های AW20 و AW40 نمایش داده شود.
  - مراحل انجام :
    1. در هنگام تعریف پارامترهای FC100 ، DB ها معین می شوند. مثلا :

DB_A_B = DB 10
DB_B_A = DB 11.

    2. هر دو سیستم مطابق شکل بعد برای انجام این کار برنامه ریزی می شوند.




**DB_COM_NO**

- این DB ، یک Instance DB مربوط به FB 101 می باشد و توسط STEP 7 ایجاد می شود. (البته به شرطی که تمام SFC و SFB های استفاده شده در نرم افزار افزونه در پروژه وجود داشته باشند. )
- علاوه بر داده های داخلی مورد استفاده در برقراری و انجام ارتباطات ، این بلاک شامل word های **وضعیت** و **کنترل** نیز می باشد.

**ساختار بلاک**

| DBW | Meaning | Content |
|---|---|---|
| 0...6 | Internal data | Input and output parameters of FB 101 'SWR_ZYK' |
| 8 | Status word | Status Word for Redundant Software Backup |
| 10 | Control word | Control Word for Redundant Software Backup |
| 12 upwards | Internal data | Irrelevant |

## ۱۵-۵ نکات مهم در افزونگی نرم افزاری

- دسترس پذیری سیستم

سیستم شامل دو CPU می باشد : CPU اصلی برنامه کاربردی را اجرا می کند و اطلاعات لازم را به CPU پشتیبان منتقل می نماید تا این CPU بتواند در هنگام وقوع خطا کنترل بخش افزونه برنامه کاربردی را به دست گرفته و آن را اجرا کند. این واحد پشتیبان هنگامی که در حالت " انتظار" قرار دارد فقط بخش local و غیر افزونه برنامه کاربردی را اجرا می کند و وقتی که CPU اصلی دچار مشکل می شود اجرای برنامه کاربردی را ادامه می دهد.

- مدت زمان انتقال داده به روز شده از واحد اصلی به واحد پشتیبان

این مسئله به CPU ، پروتکل شبکه یا پروتکل ارتباطی به کار رفته و اندازه برنامه کاربردی بستگی دارد.

- مدت زمان تغییر از سیستم اصلی به سیستم پشتیبان

این مورد به عاملی که باعث این تغییر شده (یعنی نوع خطا) ، زمان لازم برای انتقال داده و نیز تعداد واحدهای DP Slave که در کل سیستم وجود دارند بستگی دارد.

- برنامه کاربردی

برنامه های کاربردی که روی هر دو CPU اجرا می شوند می توانند به طور کامل یا فقط بخش هایی از آنها با همدیگر یکسان باشند.

- زبان های برنامه نویسی : LAD , FBD , STL , CFC , SCL
- کاربرد FB های استاندارد

تمام FB ها قابل استفاده هستند. اما در بلاک هایی که از زمان سنج ها یا شمارنده های S7 استفاده می شود فقط زمان سنج ها یا شمارنده های نوع IEC مجاز هستند.

- کاربرد کنترل های نرم افزاری استاندارد : هیچ محدودیتی وجود ندارد.
- پردازش "آلارم" در برنامه کاربردی

هیچ محدودیتی وجود ندارد. هرچند آلارم ها ممکن است در زمان تغییر از سیستم اصلی به سیستم پشتیبان از بین بروند.

- تعداد واحدهای ET 200M DP slave قابل استفاده

به CPU بستگی دارد.

به عنوان مثال حداکثر ۶۴ مدول ET 200M DP slave برای یک CPU مدل 414-2DP

- مدول های دیجیتال و آنالوگ

  تمام مدول های آنالوگ و دیجیتال قابل استفاده روی واحدهای ET 200M

- مدول های ویژه

  مدول شمارنده FM 350

- حداکثر میزان داده افزونه قابل انتقال

  ۸ کیلو بایت برای S7-300

  ۶۴ کیلو بایت برای S7-400

- خطاهای دوم و سوم

  فقط خطاهای اول قابل مدیریت و سامان دهی هستند. به عنوان مثال اگر هنگام پردازش یک خطا ، خطاهای دوم و سومی هم رخ دهد ممکن است برنامه افزونه اجرا نشود.

## ۱۵-۶ جابجایی بین سیستم و اجزای آن

### تغییر از سیستم اصلی به سیستم پشتیبان

این تغییر در صورت وقوع یکی از شرایط زیر انجام می شود :

- درخواست از طرف کاربر (با تنظیم بیت مربوطه در Control Word)
- اشکال در سیستم اصلی (قطع تغذیه یا حالت Stop)
- اشکال درDP Master سیستم اصلی
- اشکال در واسط DP Slave افزونه

و زمان آن از فرمول زیر محاسبه می گردد :

*مدت زمان تشخیص خطا + مدت زمان انتقال داده + مدت زمان switch به واسط های DP Slave*

داده هایی که منتقل می شود شامل PIQ ، نواحی آدرس پذیر به صورت بیتی ، DB های تعیین شده در FC 100 و نیز داده های داخلی می باشد و زمان انتقال آن به حجم اطلاعات ، نرخ انتقال داده ، شبکه و نوع اتصال و قابلیت CPU از لحاظ قدرت ارتباطی بستگی دارد. حجم این اطلاعات به طور تخمینی معادل سه برابر تعداد بایت های خروجی در نظر گرفته می شود و از آنجا که انتقال همه اطلاعات از یک سیستم به سیستم دیگر در یک سیکل امکان پذیر نیست بنابراین داده به بسته های کوچکتر تقسیم شده و در چند سیکل منتقل میگردد.

### مثال ۱ – انتقال اطلاعات بین دو CPU مدل 315-2DP

انتقال اطلاعات در صورت استفاده از FB 104 به صورت بلاک های ۲۴۰ بایتی و چنانچه از FB 103 استفاده شود با بلاک های ۷۶ بایتی انجام می شود بنابراین حداکثر یک بلاک در هر بار فراخوانی نرم افزار افزونه منتقل میشود

| Transmission time for PROFIBUS (AG_SEND) 187.5 kBaud to 1.5 MBaud | Transmission time for Industrial Ethernet (AG_SEND) 10 MBaud | Transmission time for MPI connection (XSEND) 187.5 kBaud |
|---|---|---|
| 60 ms per 240-byte block | 48 ms per 240-byte block | 152 ms per 76-byte block |

زمان های مندرج در جدول فوق به شرطی معتبر است که حداکثر دو واحد از سیستم افزونه در یک شبکه به هم وصل شده باشند و چنانچه بیش از دو Node در شبکه وجود داشته باشد انتقال داده ممکن است بیشتر از زمان های فوق طول بکشد.

همچنین برنامه کاربردی افزونه نیز در OB1 نوشته می شود که حداکثر زمان اجرای آن 10 ms می باشد.

### مثال ۲ – انتقال اطلاعات بین دو CPU مدل 414-2DP

| Number of bytes to be transferred | Transmission time for PROFIBUS/Industrial Ethernet, 187.5 kBaud to 12 Mbaud | Transmission time for MPI connection at 187.5 kBaud |
|---|---|---|
| 1 kByte | 250 ms | 340 ms |
| 4 kByte | 1 s | 1.36 s |
| 16 kByte | 4 s | 5.44 s |
| 64 kByte | 16 s | 21.76 s |

مانند مثال قبل در جدول فوق فرض بر این است که حداکثر دو واحد از سیستم افزونه در یک شبکه به هم وصل شده باشند و نیز انتقال اطلاعات از طریق بلاک های BSEND / BRCV انجام شود. مدت زمان انتقال اطلاعات به ظرفیت ارتباطی CPU (یعنی K bus) نیز بستگی دارد و بنابراین ممکن است بیشتر (مثلا برای CPU 412) یا کمتر (مثلا برای CPU 416 ) باشد.

### مدت زمان جایگزینی DP Slave های ET200M

هنگامی که تغییر از سیستم اصلی به سیستم پشتیبان رخ می دهد DP Salve های ET200M به طور خودکار از DP Master سیستم اصلی به DP Master سیستم پشتیبان متصل می شوند. برای S7-300 حداکثر چهار DP Slave و برای S7-400 حداکثر هشت DP Slave در هر بار فراخوانی انتقال می یابد بنابراین چنانچه تعداد DP Slave ها بیشتر از این باشد در قالب گروه های۴ یا ۸ تایی طی چندین نوبت فراخوانی منتقل می شوند.

*نکته : پریود فراخوانی جهت انتقال DP Slave ها بین دو بار فراخوانی OB1 (یا OB هایی که اجرای آنها به طور زمانی کنترل می شود) می بایست از زمان لازم جهت انتقال ۴ یا ۸ DP Slave بیشتر باشد مگر آنکه تعداد DP Slave های مورد استفاده در سیستم کمتر از ۴ یا ۸ واحد باشد.*

### مثال ۱ – CPU مدل 315-2DP با DP Master مجتمع

| Number of DP Slaves | CPU consisting of S7-300 with integrated DP Master... 12 MBaud | 1,5 MBaud | 500 kBaud | 187,5 kBaud |
|---|---|---|---|---|
| 1 | 6 ms | 6 ms | 7 ms | 12 ms |
| 2 | 12 ms | 12 ms | 14 ms | 24 ms |
| 4 | 25 ms | 25 ms | 30 ms | 50 ms |
| 8 | 2 x 25 ms | 2 x 25 ms | 2 x 30 ms | 2 x 50 ms |
| 16 | 4 x 25 ms | 4 x 25 ms | 4 x 30 ms | 4 x 50 ms |
| 32 | 8 x 25 ms | 8 x 25 ms | 8 x 30 ms | 8 x 50 ms |
| 64 | 16 x 25 ms | 16 x 25 ms | 16 x 30 ms | 16 x 50 ms |

### مثال ۲– CPU های رده S7-400 با DP Master مجتمع یا همراه با مدول های CP به عنوان DP Master

| Number of DP Slaves | CPU consisting of S7-400 with integrated DP Master 12 Mbaud | 1,5 MBaud | 500 kBaud | 187,5 kBaud | CP as DP Master (CP 443-5) 187,5 kBaud to 12 MBaud |
|---|---|---|---|---|---|
| 1 | 5 ms | 9 ms | 13 ms | 20 ms | 55 ms |
| 2 | 10 ms | 18 ms | 26 ms | 40 ms | 100 ms |
| 4 | 20 ms | 36 ms | 39 ms | 80 ms | 200 ms |
| 8 | 40 ms | 64 ms | 78 ms | 160 ms | 400 ms |
| 16 | 2 x 40 ms | 2 x 64 ms | 2 x 78 ms | 2 x 160 ms | 2 x 400 ms |
| 32 | 4 x 40 ms | 4 x 64 ms | 4 x 78 ms | 4 x 160 ms | 4 x 400 ms |
| 64 | 8 x 40 ms | 8 x 64 ms | 8 x 78 ms | 8 x 160 ms | 8 x 400 ms |

## مدت زمان تشخیص خطا

| خطاهای سیستم اصلی | | |
|---|---|---|
| **علت خطا** | **زمان تشخیص** | **واکنش سیستم** |
| CPU سیستم اصلی در حالت Stop قرار دارد یا تغذیه این سیستم قطع شده است | تقریبا یک ثانیه | ۱. واسطه های DP به Master جدید منتقل می شوند.<br>۲. انتقال از سیستم اصلی به سیستم پشتیبان صورت می گیرد.<br>۳. Status Word به نحوی تنظیم می شود که خطای " نقص در اتصال افزونه " را نشان بدهد. |
| نقص در کل یا بخشی از DP Master سیستم اصلی | در حد چند ms | ۱. واسطه های DP به Master جدید منتقل می شوند.<br>۲. انتقال از سیستم اصلی به سیستم پشتیبان صورت می گیرد.<br>۳. Status Word به نحوی تنظیم می شود که خطای " هیچ DP Slave ایی وجود ندارد " را نشان بدهد. |
| **خطاهای سیستم پشتیبان** | | |
| CPU سیستم پشتیبان در حالت Stop قرار دارد یا تغذیه این سیستم قطع شده است | تقریبا یک ثانیه | • هیچ واکنشی از طرف سیستم اصلی صورت نمی گیرد و این سیستم مطابق قبل به کار خود ادامه می دهد.<br>• Status Word به نحوی تنظیم می شود که خطای " نقص در اتصال افزونه " را نشان بدهد. |
| نقص در کل یا بخشی از DP Master سیستم پشتیبان | در حد چند ms | • هیچ واکنشی از طرف سیستم اصلی صورت نمی گیرد و این سیستم مطابق قبل به کار خود ادامه می دهد.<br>• Status Word در سیستم پشتیبان به نحوی تنظیم می شود که خطای " هیچ DP Slave ایی وجود ندارد " را نشان بدهد. |
| **خطاهای اتصال افزونه** | | |
| بروز اشکال در اتصال (link) افزونه | تقریبا یک ثانیه | • هر دو سیستم "سیستم اصلی" می شوند و واحدهای DP Salve مطابق قبل با Master قبلی متصل باقی می مانند. |
| **خطاهای مدول های محلی** | | |
| بروز اشکال در واسط DPی ET200M (IM 153-3) که به سیستم اصلی متصل است | در حد چند ms | ۱. واسط DPی ET200M به سیستم پشتیبان سوئیچ می کند.<br>۲. تمام DP Slave های دیگر با سیستم پشتیبان ارتباط برقرار می کنند.<br>۳. انتقال از سیستم اصلی به سیستم پشتیبان انجام می شود. |
| بروز اشکال در واسط DPی ET200M (IM 153-3) که به سیستم پشتیبان متصل است | در حد چند ms | • هیچ واکنشی از طرف سیستم اصلی صورت نمی گیرد و این سیستم مطابق قبل به کار خود ادامه می دهد.<br>• Status Word در سیستم پشتیبان به نحوی تنظیم می شود که خطای " هیچ |

| | | |
|---|---|---|
| | | DP Slave ایی وجود ندارد " را نشان بدهد. |
| بروز اشکال در تغذیه ET200M (IM 153-3) | در حد چند ms | ۱. تمام DP Salve های در دسترس به سیستم پشتیبان سوئیچ می کنند.<br>۲. انتقال از سیستم اصلی به سیستم پشتیبان انجام می شود. |

## ۱۵-۷ شبکه های قابل استفاده جهت اتصال دو سیستم اصلی و پشتیبان

دو سیستم اصلی و پشتیبان را می توان از طریق یکی از شبکه های MPI ، PROFIBUS یا Industrial Ethernet به هم متصل نمود . البته شبکه MPI فقط وقتی استفاده می شود که حجم داده ای که قرار است روی شبکه منتقل شود نسبتا کم (تا سقف یک کیلو بایت) باشد.

همچنین برای برقراری این اتصال می بایست بلاک ها ی مربوط به نرم افزار افزونه از Library خاص و مناسب شبکه پیکربندی شده در هر دو سیستم کپی شود. جداول زیر گزینه های مختلف جهت اتصال دو سیستم را نشان می دهد.

### اتصال دو سیستم S7-300

| Stations Networked Via… | Network Connection Interface | Transmission Speed | Connection Required | Library for Blocks Required |
|---|---|---|---|---|
| MPI | CPU | 187.5 KBaud | Permanently configured connection | XSEND_300 |
| PROFIBUS | CP 342-5 | max. 1.5 MBaud | FDL connection | AG_SEND_300 |
| Industrial Ethernet | CP 345-1 | 10 MBaud | ISO connection | AG_SEND_300 |

### اتصال دو سیستم S7-400

| Stations Networked Via… | Network Connection Interface | Transmission Speed | Connection Required | Library for Blocks Required |
|---|---|---|---|---|
| MPI | CPU | 187.5 KBaud | Permanently configured Connection– S7 connection | XSEND_400<br>BSEND_400 |
| PROFIBUS | CP 443-5 | max. 12 MBaud | FDL connection<br>S7 connection | AG_SEND_400<br>BSEND_400 |
| Industrial Ethernet | CP 443-1 | 10 MBaud | ISO connection<br>S7 connection | AG_SEND_400<br>BSEND_400 |

### ۱۵-۸ تغییر پیکربندی یا برنامه کاربردی در مد RUN

برای ایجاد چنین تغییراتی وقتی که سیستم در حال کار است ، ابتدا باید واحد پشتیبان را غیر فعال کرد که این کار با تنظیم بیت "De-activate Redundant Backup" که یکی از بیت های Control Word می باشد انجام می گردد. سپس باید تغییر مورد نظر ابتدا در واحد پشتیبان و بعد در سیستم اصلی اعمال شود. حال میتوان با تنظیم بیت "Activate Redundant Backup " مجددا افزونگی را برقرار کرده و کنترل فرآیند را تحت این ویژگی ادامه داد.

هنگامی که سیستم در حال اجرا است نمی توان اندازه نواحی داده در واحد پشتیبان را تغییر داد لیکن محتویات یک ناحیه از داده به شرط آن که اندازه آن ثابت بماند قابل تغییر است. بنابراین بهتر است به هنگام تعریف و تنظیم نواحی داده این نکته را در نظر داشت و از ابتدا حداکثر میزان مورد انتظار را برای نواحی داده اختصاص داد.

### مراحل لازم جهت افزودن یک مدول ET 200M DP Slave به واحد پشتیبان

۱- غیر فعال کردن واحد پشتیبان با تنظیم بیت مربوطه در Control Word

۲- قرار دادن CPU واحد پشتیبان در حالت STOP

۳- ترکیب بندی مدول اضافه شده و سپس انتقال پیکربندی سخت افزار به CPU

۴- تنظیم پارامترهای مرتبط با فراخوانی FC 100 (PAA_FIRST, PAA_LAST, SLAVE_NO, SLAVE_LEN).

۵- حذف بلاکهای داده DB_WORK_NO, DB_SEND, DB_RCV, DB_A_B_NO, DB_B_A_NO.

۶- قرار دادن CPU واحد پشتیبان در حالت RUN

۷- قرار دادن CPU واحد اصلی در حالت STOP

۸- ترکیب بندی مدول اضافه شده و سپس انتقال پیکربندی سخت افزار به CPU

۹- تنظیم پارامترهای مرتبط با فراخوانی FC 100 (PAA_FIRST, PAA_LAST, SLAVE_NO, SLAVE_LEN).

۱۰- حذف بلاکهای داده DB_WORK_NO, DB_SEND, DB_RCV, DB_A_B_NO, DB_B_A_NO.

۱۱- قرار دادن CPU واحد اصلی در حالت RUN

### مراحل لازم جهت جایگزینی CPU یا ارتقا Firmware آن

۱- قرار دادن CPU در حالت STOP

۲- جایگزین کردن CPU جدید و انتقال کلیه اطلاعات شامل پیکربندی سخت افزار ، ترکیب بندی اتصال دو سیستم اصلی و پشتیبان و نیز بلاک های برنامه به این CPU

۳- قرار دادن CPU در حالت RUN

### جدا کردن و اتصال مجدد مدول های جانبی

این کار مشابه آنچه در S7 استاندارد و بدون افزونگی صورت می گیرد در یک سیستم افزونه نیز قابل انجام است فقط باید اطمینان داشت که به هنگام جدا کردن یا اتصال مجدد مدول جانبی تغییر از واحد اصلی به پشتیبان رخ ندهد بدین منظور می توان امکان این تغییر را غیر فعال کرد ( Disable master-reserve change-over).

### ویژگی های خاص برنامه نویسی به زبان CFC

چنانچه برنامه کاربردی به زبان CFC نوشته شده باشد در این صورت FC100 می بایست از داخل یک FB فراخوانی شود. همچنین می بایست تا حد لزوم برنامه را به زبان STL نوشت و فقط FB فوق می تواند در لیست Startup قرار داده شده و به زبان .CFC باشد.

یک FC به نام FC99 در شاخه Blocks یک برنامه نمونه برای سیستم S7-400 وجود دارد که در آن کارهای لازم جهت استفاده از ویژگی فوق برنامه نویسی شده و قابل کاربرد در برنامه های دیگر نیز می باشد.

### ۱۵-۹ مدول های مناسب جهت استفاده در یک سیستم مبتنی بر افزونگی نرم افزاری

مدول های زیر برای کاربرد در یک سیستم مبتنی بر افزونگی نرم افزاری مناسب هستند. البته تعداد این مدول ها به طور دائم در حال تغییر است و برای دسترسی به یک لیست به روز شده می توان به آدرس زیر مراجعه کرد

http://www.siemens.com/automation/service&support

## CPU

| Description | Order Number |
|---|---|
| CPU 315-2DP | 6ES7 315-2AFxx-0AB0<br>6ES7 315-2AG10-0AB0 |
| CPU 316-2DP | 6ES7 316-2AGxx-0AB0 |
| CPU 318-2DP | 6ES7 316-2AJxx-0AB0 |
| CPU 412-1 | 6ES7 412-1XFxx-0AB0<br>6ES7 412-1FK03-0AB0 |
| CPU 412-2 | 6ES7 412-2XGxx-0AB0 |
| CPU 413-1 | 6ES7 413-1XGxx-0AB0 |
| CPU 413-2DP | 6ES7 413-2XGxx-0AB0 |
| CPU 414-1 | 6ES7 414-1XGxx-0AB0 |
| CPU 414-2DP | 6ES7 414-2XGxx-0AB0<br>6ES7 414-2XJxx-0AB0 |
| CPU 414-3DP | 6ES7 414-3XJxx-0AB0 |
| CPU 416-1 | 6ES7 416-1XJxx-0AB0 |
| CPU 416-2DP | 6ES7 416-2XKxx-0AB0<br>6ES7 416-2XLxx-0AB0 |
| CPU 416-3DP | 6ES7 416-3XLxx-0AB0 |
| CPU 417-4 | 6ES7 417-4XLxx-0AB0 |

## Communication Modules with DP Master Function

| Description | Order Number |
|---|---|
| Communication module CP 443-5 Extended<br>(for connection to PROFIBUS network) | 6EK7 443-5DXxx-0XE0 |
| DP master interface IM 467 or IM 467-FO<br>(for use in V1.1 only) | 6ES7 4675GJxx-0AB0<br>6ES7 4675FJxx-0AB0 |

## Communication Modules for linking the stations

| Description | Order Number |
|---|---|
| Communication module CP 342-5 | 6ES7 342-5DA00-0XE0<br>6GK7 342-5DA02-0XE0 |
| Communication module CP 343-1 | 6GK7 343-1BA00-0XE0<br>6GK7 343-1EX11-0XE0 |
| Communication module CP 443-5 Extended<br>(for connection to PROFIBUS network) | 6EK7 443-5DXxx-0XE0 |
| Communication module CP 443-1 ISO<br>(for connection to Industrial Ethernet) | 6EK7 443-1BXxx-0XE0 |

## Modules for ET 200M Distributed I/O Device

| Description | Order Number |
|---|---|
| 2x DP slave interface IM 153-2 | 6ES7 153-2AA02-0XB0 => version 2<br>6ES7 153-2AB01-0XB0 => version 2 |
| All digital and analogue modules for ET 200M | (See catalogue ST70) |
| Counter module FM 350 | 6ES7 350-1AH0x-0AE0 |
| CP 341 (20 mA TTY, RS232, RS422/485) | 6ES7 341-1xH01-0AE0 |

### ۱۵-۱۰ ارتباط با سایر سیستم ها

یک سیستم مبتنی بر افزونگی نرم افزاری می تواند با سایر سیستم ها نیز ارتباط برقرار کند. البته از آنجا که در قسمت ورودی / خروجی های توزیع شده ET 200M نمی توان از مدول ارتباطی استفاده کرد بنابراین این ارتباط باید به کمک مدول های CP که روی CPU به کار گرفته شده اند، انجام شود. همچنین جهت بالا بردن میزان دسترس پذیری سیستم می بایست یک CP روی CPU واحد اصلی و یک CP روی CPU واحد پشتیبان نصب شود.

### ارتباط با یک سیستم S7-300/S7-400

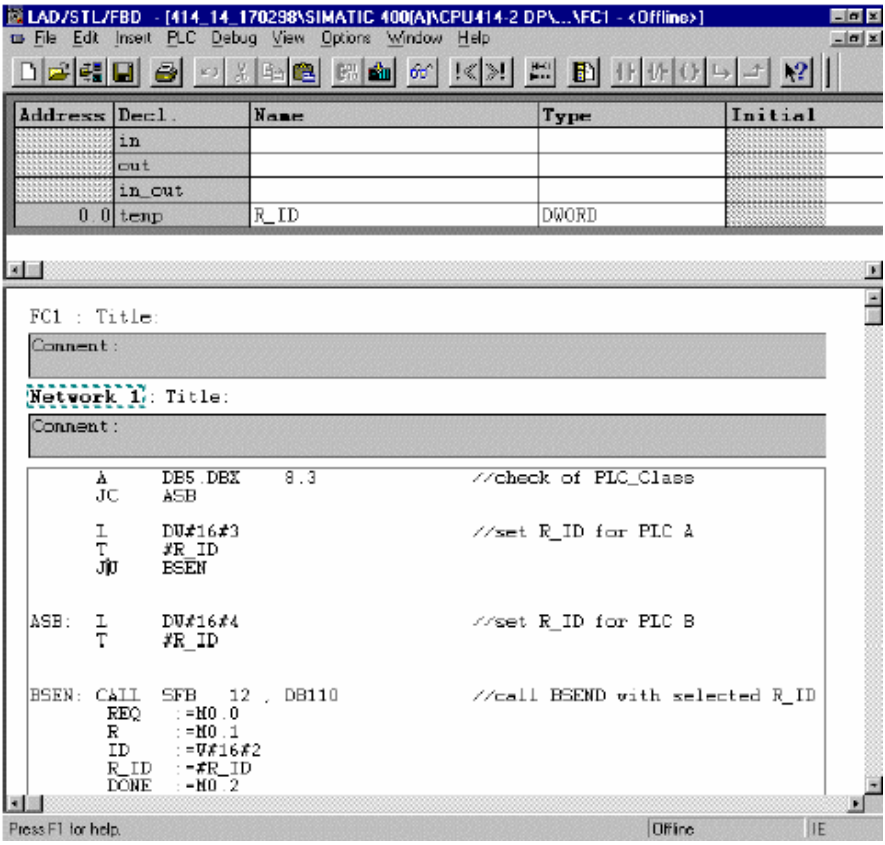
ابتدا باید دو ارتباط مجزا از دو واحد اصلی و پشتیبان به سیستم s7-300/s7-400 ترکیب بندی شود. سپس برای آن که اشکالی در برقراری ارتباط رخ ندهد می بایست بخش های ارتباطی نرم افزار در سیستم پشتیبان هم اجرا شوند. ساختار زیر که در OB1 یا OB35 پیاده سازی می شود برای استفاده در برنامه کاربردی افزونه پیشنهاد می شود.

| | |
|---|---|
| FB101 در ابتدای OB1 یا OB35 فراخوانی می شود<br>پارامتر CALL_POSITION برابر TRUE قرار داده می شود همچنین اطلاعات "وضعیت" و "کنترل" را می توان در instance DB تعیین شده پردازش کرد | CALL FB 101, DB5<br>DB_WORK_NO :=DB1<br>CALL_POSITION :=TRUE<br>RETURN_VAL :=MW6<br>EXT_INFO :=MW8 |
| اطلاعات "وضعیت" و نیز خود برنامه به نحوی تحلیل می شود تا CPU وقتی که به عنوان CPU سیستم پشتیبان تلقی می شود از اجرای برنامه کاربردی افزونه صرف نظر کند.. | A DB5.DBX 9.1<br>JC M001 |
| برنامه کاربردی افزونه در این قسمت نوشته می شود. | Instructions for redundant-backup application program |
| برنامه ارتباطی در این قسمت نوشته می شود | M001: CALL FC 1<br>Instructions for communication program |
| FB101 در انتهای OB1 یا OB35 فراخوانی می شود<br>پارامتر CALL_POSITION برابر FALSE قرار داده می شود با این کار به سیستم اعلام می شود که پردازش برنامه کاربردی افزونه به اتمام رسیده است. | CALL FB 101, DB5<br>DB_WORK_NO :=DB1<br>CALL_POSITION :=FALSE<br>RETURN_VAL :=MW10<br>EXT_INFO :=MW12 |

**ملاحظات**

- بلاک های ارتباطی باید در بلاک FC1 فراخوانی شوند و حتما باید عدد تعیین شده برای پارامتر R_ID در دو سیستم اصلی و پشتیبان متفاوت باشد.
- "کلمه وضعیت" باید جزو ناحیه داده ایی که منتقل می گردد باشد تا سیستم مقصد بتواند تحلیل کند که در حال حاضر کدام ارتباط (اصلی یا پشتیبان) فعال است. تحلیل بیشتر داده های دریافتی فقط در سیستم اصلی انجام می شود.
- در صورتیکه برنامه کاربردی به زبان CFC نوشته شده باشد ابتدا باید FC1 به یکی از زبان های LAD ، FBD یا STL برنامه نویسی شود. همچنین آن برنامه کاربردی نبایست شامل هیچ "متغیر فرآیند" یا "شماره پیغام" ایی باشد.

**یک نمونه برنامه FC1**



### ارتباط با یک سیستم افزونه دیگر

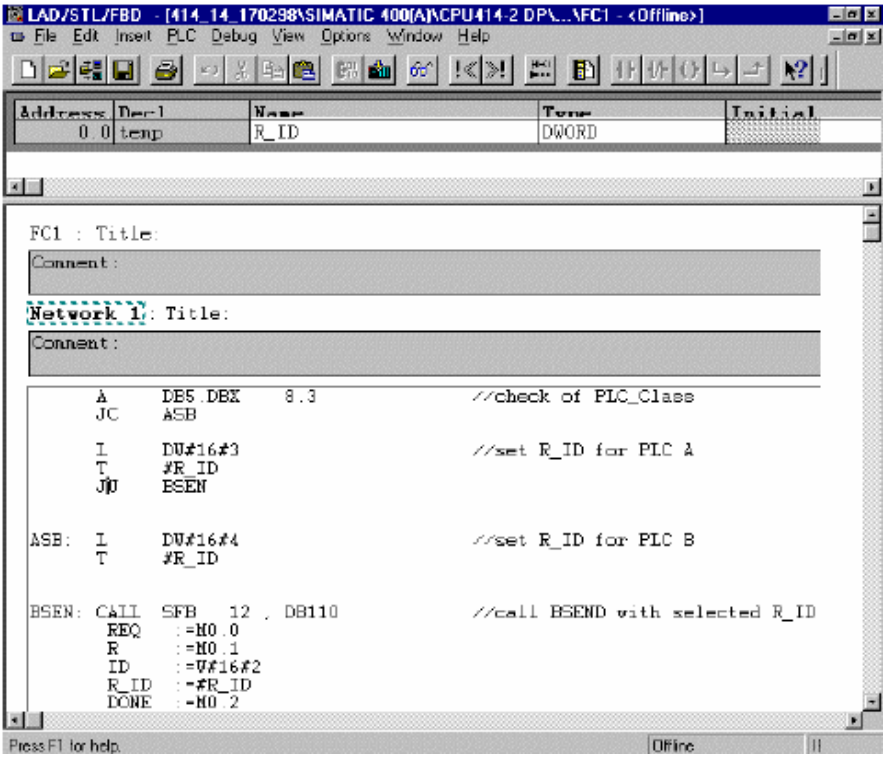
برای آنکه افزونگی در هر دو سیستم به طور مستقل از هم برقرار باشد چهار ارتباط بایستی تنظیم شود :
دو ارتباط از سیستم اصلی به سیستم افزونه و دو ارتباط از سیستم پشتیبان به سیستم افزونه
سپس برای آن که اشکالی در برقراری ارتباط رخ ندهد می بایست بخش های ارتباطی نرم افزار در سیستم پشتیبان هم اجرا شوند. ساختار زیر که در OB1 یا OB35 پیاده سازی می شود برای استفاده در برنامه کاربردی افزونه پیشنهاد می شود.

| | |
|---|---|
| FB101 در ابتدای OB1 یا OB35 فراخوانی می شود<br>پارامتر CALL_POSITION برابر TRUE قرار داده می شود<br>همچنین اطلاعات "وضعیت" و "کنترل" را می توان در instance DB تعیین شده پردازش کرد | CALL FB 101, DB5<br>DB_WORK_NO :=DB1<br>CALL_POSITION :=TRUE<br>RETURN_VAL :=MW6<br>EXT_INFO :=MW8 |
| اطلاعات "وضعیت" و نیز خود برنامه به نحوی تحلیل می شود تا CPU وقتی که به عنوان CPU سیستم پشتیبان تلقی می شود از اجرای برنامه کاربردی افزونه صرفنظر کند. | U DB5.DBX 9.1<br>SPB M001 |
| برنامه کاربردی افزونه در این قسمت نوشته می شود. | Instructions for redundant-backup application program |
| برنامه ارتباطی در این قسمت نوشته می شود | M001: CALL FC 1<br>CALL FC 2<br>Instructions for communication program |
| FB101 در انتهای OB1 یا OB35 فراخوانی می شود<br>پارامتر CALL_POSITION برابر FALSE قرار داده می شود<br>با این کار به سیستم اعلام می شود که پردازش برنامه کاربردی افزونه به اتمام رسیده است. | CALL FB 101, DB5<br>DB_WORK_NO :=DB1<br>CALL_POSITION :=FALSE<br>RETURN_VAL :=MW10<br>EXT_INFO :=MW12 |

**ملاحظات**

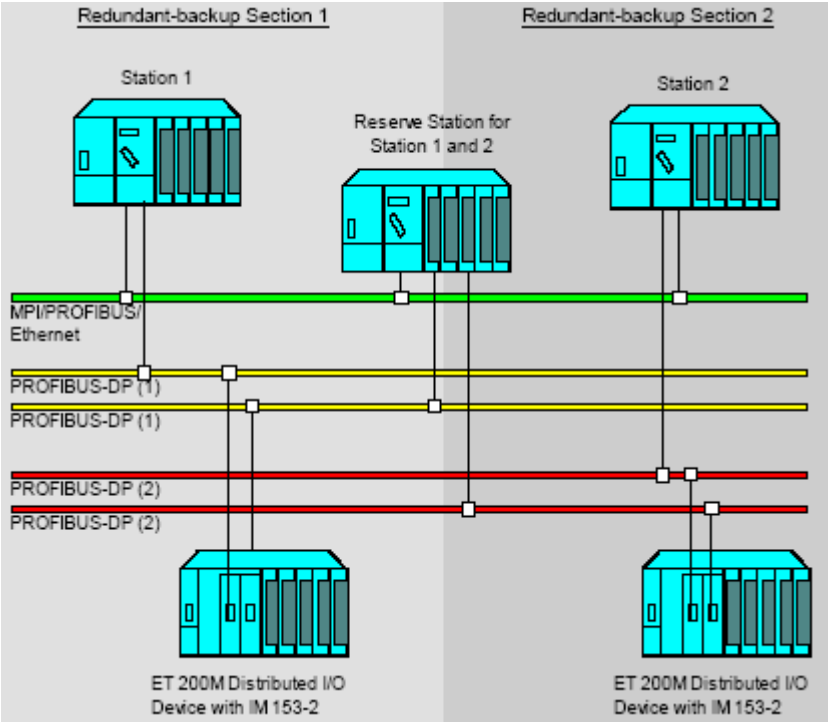
- بلاک های ارتباطی مربوط به واحد اصلی هر سیستم در FC1 و بلاک های ارتباطی مربوط به واحد پشتیبان آن سیستم در FC2 فراخوانی می شوند و حتما باید عدد تعیین شده برای پارامتر R_ID در دو واحد اصلی و پشتیبان متفاوت باشد.
- "کلمه وضعیت" باید جزو ناحیه داده ایی که منتقل می گردد باشد تا واحد مقصد بتواند تحلیل کند که در حال حاضر کدام ارتباط (اصلی یا پشتیبان) فعال است. تحلیل بیشتر داده های دریافتی فقط در واحد اصلی انجام می شود.
- در صورتیکه برنامه کاربردی به زبان CFC نوشته شده باشد ابتدا باید FC1 به یکی از زبان های LAD ، FBD یا STL برنامه نویسی شود. همچنین آن برنامه کاربردی نبایست شامل هیچ "متغیر فرآیند" یا "شماره پیغام" ایی باشد.

**یک نمونه از برنامه FC1/FC2**



## ۱۵-۱۱ جایگاه وضعیت Standby در افزونگی نرم افزاری

با در نظر گرفتن شکل زیر می توان حالت Standby را در سیستم های مبتنی بر افزونگی نرم افزاری بدین صورت تشریح کرد که هر زمان که گروهی ا ز واحد ها (مثلا در شکل زیر Station 1 یا Station 2) دچار مشکل شوند سیستم پشتیبان(Station R ) وارد عمل شده و وظایف واحد معیوب را به عهده می گیرد.




**ملاحظات**

- حتما بایستی یک اتصال افزونه بین Station 1 و Station R و یک اتصال افزونه دیگر بین Station 2 و Station R برقرار باشد.
- برنامه هایی که روی Station 1 و Station 2 اجرا می شوند باید روی Station R هم بارگذاری شوند.
- واحد پشتیبان ( Station R ) باید بتواند به ورودی / خروجی های توزیع شده ET200M مربوط به هر دو واحد (Station 1 و Station 2) دسترسی داشته باشد.

### ۱۵-۱۲ استفاده از OB های مربوط به مدیریت خطا

به منظور آن که سیستم در هنگام وقوع خطا به حالت STOP رفته و هیچ واکنشی از خود نشان ندهد باید با استفاده از OB ها واکنش های سیستم را در "کلاس های اولویت" قرار داده و آنها را اولویت بندی کرد. همچنین برای آن که سیستم به هنگام نقص در عملکرد یک DP slave به حالت STOP رفته و هیچ کاری نکند OB های زیر که مربوط به مدیریت خطا هستند نیز می بایست علاوه بر OB 86 در CPU بارگذاری شوند :

- OB 80 : به هنگام تغییر از سیستم اصلی به سیستم پشتیبان امکان اینکه Cycle time از میزان تعیین شده بیشتر شود وجود دارد.
- OB 82 : آلارم تشخیص خطا از یک مدول (مثل IM 153-2) که روی واسط DP salve سیستم پشتیبان قرار دارد
- OB 83 : آلارم نشان دهنده جدا کردن یا اتصال مجدد مدول هایی که روی واسط DP salve قرار دارند
- OB 85 : خطای اجرای برنامه که هنگامی رخ می دهد که واسط DP slave دچار مشکل شود
- OB 87 : خطای در ارتباط
- OB 87 : خطای دسترسی به مدول های جانبی (مثلا هنگامی که IM 153-2 یا مدول دیگری روی یکی از سیستم ها دچار مشکل شود)

با استفاده از OB های فوق برنامه کاربردی نسبت به خطایی که در سیستم اتفاق می افتد واکنش مناسب نشان می دهد بدون آنکه نرم افزار افزونه این OB ها را تحلیل کرده و یا واکنشی را آغاز نماید.

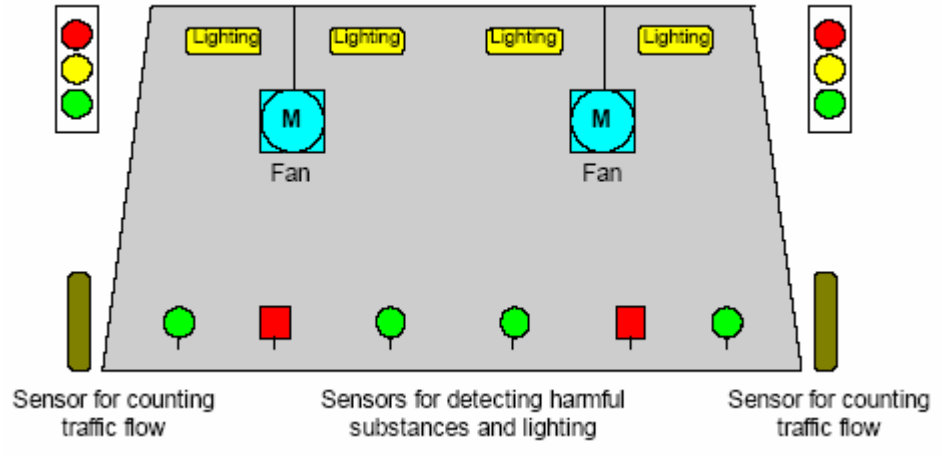
### ۱۵-۱۳ مثالی از پیاده سازی افزونگی نرم افزاری با استفاده از S7-300

یک تونل در مسیر رفت و آمد خودروها قرار دارد که دو هواکش کار تهویه هوای آن را انجام می دهند. هر کدام از این هواکش ها دارای دو سرعت هستند که با توجه به میزان تجمع ذرات مضر در تونل سرعت مناسب آنها تعیین می شود. میزان ذرات مضر نیز توسط دو سیگنال آنالوگ دریافت می شود. این دو هواکش مهمترین بخش این فرآیند هستند و بنابراین آن بخش از برنامه که این دو هواکش را کنترل می کند باید در هر دو سیستم A و B بارگذاری شود.

به منظور داشتن یک آمار صحیح ، میزان تردد خودروها در داخل تونل در هر روز ثبت می شود که این تعداد توسط دو سنسور که در دو انتهای تونل نصب شده اند استخراج می گردد با توجه به آنکه این کمیت در کنترل تونل از اهمیت چندانی برخوردار نیست بنابراین برنامه مربوط به این بخش فقط در سیستم A بارگذاری می شود.

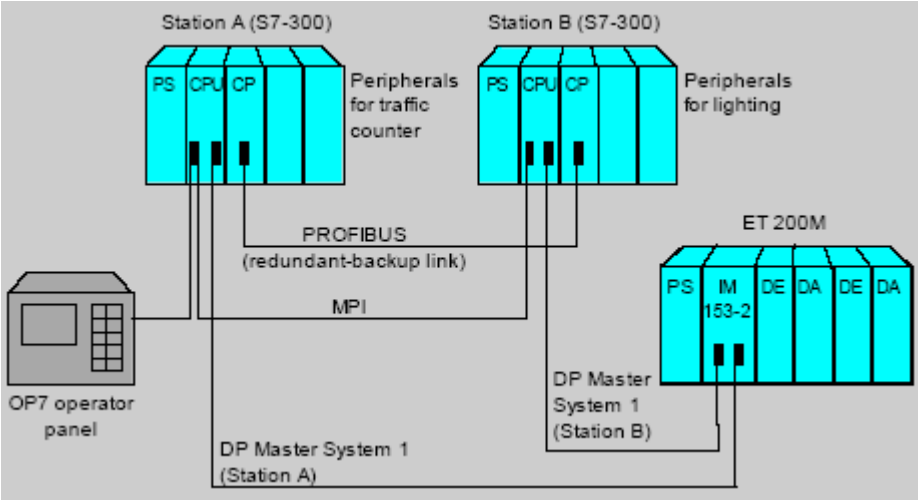
روشنایی تونل نیز توسط چهار سیگنال دیجیتال نظارت می شود و چنانچه روشنایی هر یک از چهار بخش تونل دچار اشکال شود سیگنال مربوط به آن بخش فعال می شود. این بخش از برنامه کاربردی نیز نیاز نیازی به افزونگی ندارد و می تواند در سیستم B بارگذاری شود.

شکل زیر شمایی از سیستم تحت کنترل را نشان می دهد.



## سخت افزار سیستم

شکل زیر سخت افزار پیشنهادی برای کنترل این فرآیند را نشان می دهد که شامل دو سیستم S7-300 هر کدام دارای CPU مدل 315-2DP و نیز مدول ET 200M DP salve می باشد. واسط IM 153-2 DP در ET 200M با دو ارتباط مجزا هم به CPU سیستم A و هم به CPU سیستم B متصل است. همچنین دو سیستم A و B توسط یک CP 342-5 که به شبکه PROFIBUS متصل است با هم ارتباط دارند




## نحوه پیکربندی سخت افزار

- یک پروژه در محیط Step 7 شامل دو سیستم A و B ایجاد کرده و سپس سیستم A را باز کنید
- یک Rack انتخاب کنید و آن را باز کنید
- منبع تغذیه ، CPU 315-2DP و سایر مدول های جانبی را اضافه کنید
- سیستم B را باز کرده و مراحل فوق را برای آن تکرار کنید
- IM 153-3 را به DP master اضافه کنید
- مدول های ET 200M را اضافه کنید
- چنانچه می خواهید بیش از یک ET 200m DP salve داشته باشید دو مرحله قبل را تکرار کنید
- تمام شاخه DP را در DP master دوم کپی کنید

**ملاحظات**

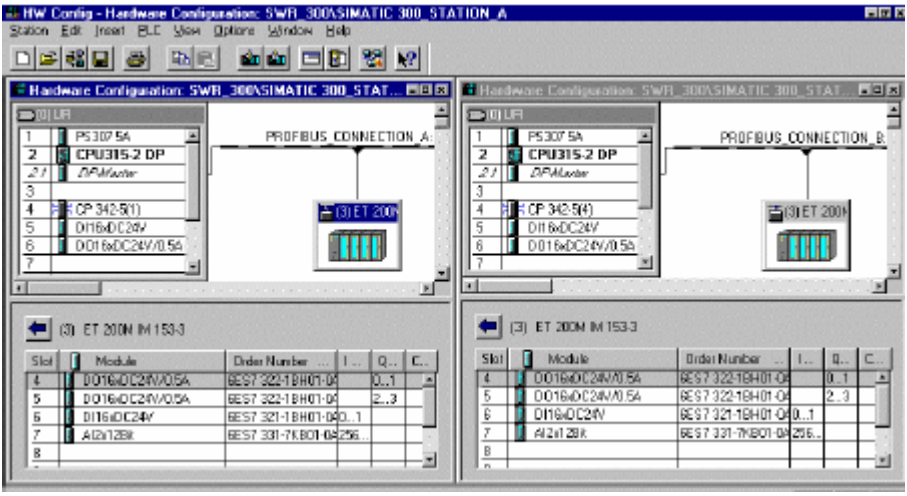
- پیکربندی مدول های محلی برای هر دو سیستم باید دقیقا یکسان باشد
- برای آنکه یکپارچگی سیستم حفظ شود همیشه باید کل DP master سیستم اول در DP master سیستم دوم کپی شود حتی در صورتیکه تغییرات جزئی در آن ایجاد شده باشد. برای این کار می توان به طریق زیر عمل کرد :

**Edit > Insert Redundant Copy**

با استفاده از روش فوق آدرس مدول ها ی جانبی نصب شده روی DP slave های هر دو سیستم نیز یکسان خواهند شد.

## نمونه ایی از پیکربندی سخت افزار برای دو سیستم A و B

شکل زیر نمونه ای از پیکربندی یکسان سخت افزار در هر دو DP master را نشان می دهد.




## پیکربندی شبکه ها

در سیستم های مبتنی بر افزونگی نرم افزاری دو شبکه متمایز وجود دارد :

- شبکه ای که دو سیستم اصلی و پشتیبان را به هم متصل می کند و انتقال اطلاعات بین دو سیستم از طریق آن صورت می گیرد. این شبکه می تواند از نوع MPI ، PROFIBUS یا Industrial Ethernet باشد که در این مثال از شبکه PROFIBUS استفاده شده است. به این ترتیب :

- یک شبکه PROFIBUS ایجاد کنید
- مدول CP از سیستم A را به این شبکه متصل کرده و یک آدرس برای این node انتخاب کنید (مثلا ۳)
- مدول CP از سیستم B را به این شبکه متصل کرده و یک آدرس برای این node انتخاب کنید (مثلا ۴)

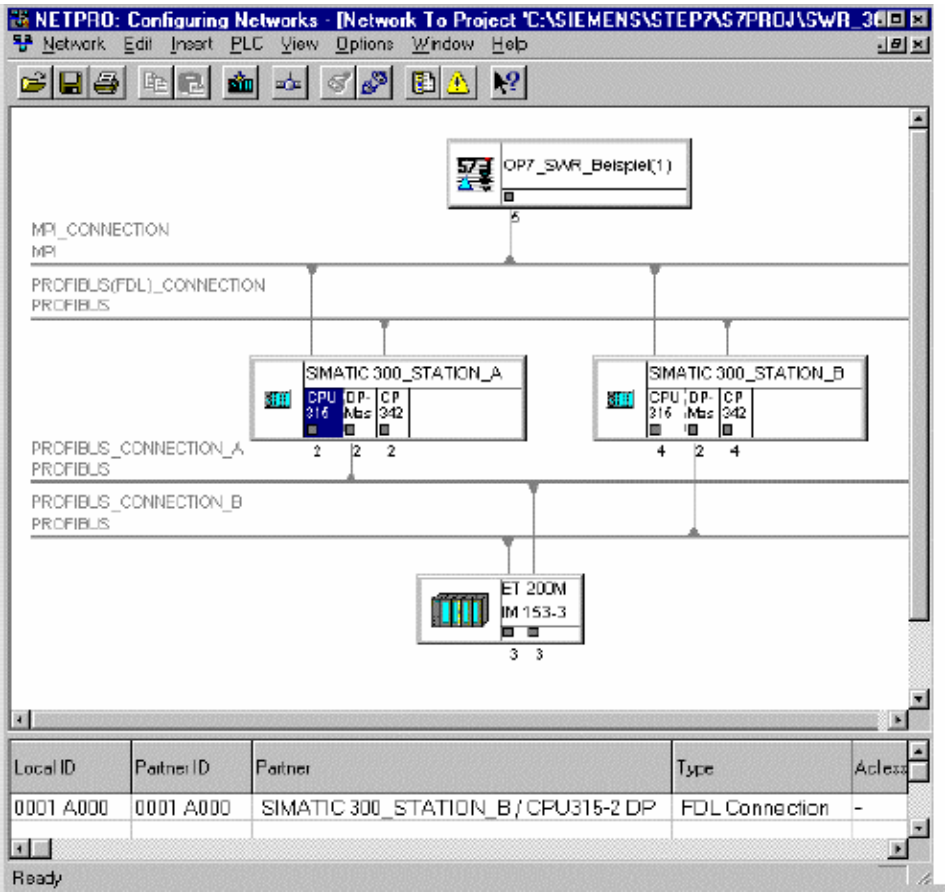
- شبکه PROFIBUS-DP که DP master ها را به مدول های جانبی ET 200M متصل می کند و از طریق آن سیستم های اصلی و پشتیبان با مدول های محلی خود ارتباط برقرار می کنند. مدول های جانبی ET 200M دو واسط DP دارند که یکی به DP master سیستم A و دیگری به DP master سیستم B متصل می شود ، به این ترتیب :
  - دو شبکه PROFIBUS-DP ایجاد کنید.
  - اتصال DP مربوط به CPU سیستم A را انتخاب کرده و آن را به شبکه اول وصل کنید.
  - اتصال DP مربوط به CPU سیستم B را انتخاب کرده و آن را به شبکه دوم وصل کنید.
  - از قسمت Hardware Catalogue یک IM 153-2 انتخاب کنید (از زیر شاخه PROFIBUS-DP مربوط به ET 200M)

## پیکربندی ارتباط ها

در این مثال یک شبکه PROFIBUS با ارتباط نوع FDL برای انتقال داده ها بین دو سیستم در نظر گرفته شده است که به منظور ایجاد ارتباطات منطقی لازم باید به ترتیب زیر عمل کرد. :

- از محیط SIMATIC Manager به محیط شبکه (Network view) سوئیچ کنید
- برای آنکه DP slave ها هم نشان داده شوند گزینه مربوط به آن را در قسمت View > DP Slaves انتخاب کنید

- در محیط Network view روی Connection table دو بار کلیک کنید تا پنجره مربوط به تعریف نوع ارتباط ها باز شود.
- هر دو سیستم را انتخاب کرده و سپس ارتباط نوع FDL را انتخاب نمایید.




## ایجاد برنامه کاربردی

برنامه کاربردی این مثال شامل یک برنامه افزونه و دو برنامه متفاوت می باشد. برنامه افزونه که روی هر دو سیستم به طور کاملا یکسان کپی شده به عنوان بخشی از OB 35 اجرا می شود و دو برنامه دیگر هر کدام روی یک سیستم و به عنوان بخشی از OB 1 فراخوانی می شوند.

## ساختار برنامه کاربردی

**OB 100 Startup Program**

```
CALL FC 100
AG_KENNUNG          :='A'
DB_WORK_NO          :=DB1
DB_SEND_NO          :=DB2
DB_RCV_NO           :=DB3
MPI_ADR             :=4
          etc.
```

**شرح :**

OB 100 بلاک FC 100 را فراخوانی می کند که این بلاک نیز به نوبه خود به سیستم اطلاع می دهد که چه آدرس هایی می بایست برای ارتباط مورد استفاده قرار گیرند و چه فضاهایی از داده ها قرار است در این ارتباط بین دو سیستم منتقل شوند. این داده ها شامل مقدار ورودی ها , حافظه بیتی ، DB ها و Instance DB های مربوط به شمارنده / زمان سنج های IEC می باشد.

**OB 35 Timer-controlled Program**

```
CALL FB 101, DB5
DB_WORK_NO    :=DB1
CALL_POSITION :=TRUE
RETURN_VAL    :=MW115
EXT_INFO      :=MW117

A     DB5.DBX     9.1
            JC    M001

Instructions for redundant-backup application program
(program section exists on station A and station B)

M001: CALL FB 101, DB5
DB_WORK_NO      :=DB1
CALL_POSITION   :=FALSE
RETURN_VAL      :=MW119
EXT_INFO        :=MW121
```

**شرح :**

FC 101 باید در ابتدای OB 35 فراخوانی شود و پارامتر CALL_POSITION را معادل TRUEقرار دهد. و پس از آن Instance DB تعیین شده می تواند برای پردازش دادهای کنترلی یا وضعیت مورد استفاده قرار گیرد.

سپس اطلاعات وضعیت تحلیل می شود و چنانچه CPU متعلق به سیستم پشتیبان باشد از اجرای برنامه افزونه صرف نظر می کند. پس از این بخش برنامه افزونه قرار داده می شود و در نهایت FC 101 در انتهای OB 35 فراخوانی شده و پارامتر CALL_POSITION را معادل TRUEقرار دهد

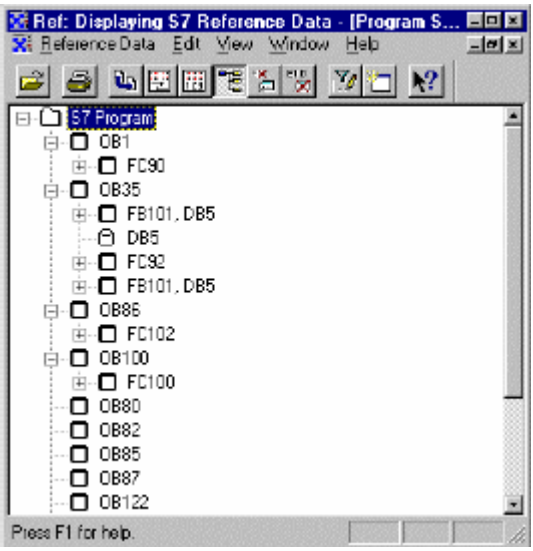
**OB 86 Diagnostic Program**

```
CALL FC 102
DB_WORK              :=W#16#1
OB86_EV_CLASS        :=#OB86_EV_CLASS
OB86_FLT_ID          :=#OB86_FLT_ID
RETURN_VAL           :=MW130
```

**شرح :**

OB 86 باید FC 102 را فراخوانی کرده و پارامترهای مربوط به Start up را مقدار دهی نماید. این فراخوانی از آن جهت صورت می گیرد تا سیستم بتواند به طور اتوماتیک به نقص در عملکرد DP slave ها واکنش نشان داده و تغییر از سیستم اصلی به پشتیبان انجام داده شود.

**ساختار بلاکی برنامه**




**ملاحظات مربوط به برنامه کاربردی**

برنامه کاربردی باید به نحوی تنظیم شود که بخش های افزونه و معمولی مجزا از هم باشند. همچنین فقط از زمان سنج ها و شمارنده های نوع IEC می توان در بخش افزونه برنامه استفاده کرد زیرا داده مربوط به زمان سنج ها و شمارنده های نوع S7 را نمی توان بین دو سیستم منتقل کرد.

# فصل شانزدهم - افزونگی سخت افزاری
## Hardware Redundancy

**مشتمل بر :**

**١٦-١ سیستم H در یک نگاه**

**١٦-٢ I/O Redundancy چیست ؟**

**١٦-٣ اجزای سخت افزاری سیستم H**

**١٦-٤ روش نصب یک سیستم نمونه**

**١٦-٥ نحوه پیکر بندی در Step7**

**١٦-٦ نحوه عملکرد سیستم های S7-400H**

**١٦-٧ تغییر در سخت افزار سیستم های H در حین کار**

**١٦-٨ MTBF در سیستم های H**

### ۱۶-۱ سیستم H در یک نگاه

سیستم های H یا Hot Standby همانطور که از نامشان پیداست سیستم هایی هستند که با قابلیت Redundancy بصورت پشتیبان یکدیگر میتوانند یک پروسه را کنترل نمایند.این سیستم ها بصورت دوقلو یا جفت بکار میروند و فرآیند تحت کنترل یکی از آنهاست که Master نامیده میشود. سیستم دیگر که Stand by نامیده میشود و بحالت آماده بکار است در آن لحظه نقشی در کنترل فرآیند ندارد ولی در صورت بروز اشکال در Master سیستم پشتیبان در زمان بسیار کوتاهی وارد مدار شده و کار کنترل فرآیند را ادامه میدهد.وقتی یک سیستم در حال کار و دیگری آماده بکار است مد کاری مجموعه را Redundant میگویند و وقتی یک سیستم بدلیل خطا متوقف میشود و دیگری در مدار می آید مد کاری مجموعه را Single می نامند.

ساختاری که به اینصورت ایجاد میشود Hardware Redundancy خوانده میشود و نباید آنرا با Software Redundancy که قبلاً توضیح داده شد اشتباه گرفت. در واقع بین این دو سیستم از نظر ساختار و از نظر عملکرد تفاوتهای اساسی زیر وجود دارد:

| Hardware Redundancy | Software Redundancy |
|---|---|
| فقط با CPU های خاصی از خانواده S7-400H قابل پیاده سازی است | با CPU های خانواده S7-300 و S7-400 قابل پیاده سازی است. |
| عمل سنکرون سازی حافظه بین دو CPU بطریق سخت افزاری و ازطریق ماژولهای خاصی به نام SYNC که با فیبر نوری به هم متصلند انجام میشود. | سنکرون سازی حافظه دو CPU با فانکشنهای خاص برنامه نویسی و از طریق کابل شبکه انجام میشود. |
| قابلیت اطمینان بالاتر دارد و زمان جابجایی از سیستم اصلی به سیستم پشتیبان خیلی کوتاهست. | قابلیت اطمینان آن به اندازه سیستم H نیست و زمان جابجایی آن نیز به کوتاهی سیستم های H نمی باشد. |
| هزینه آن زیاد است | هزینه بالایی ندارد |

انتخاب بین دو روش فوق بر اساس نیاز فرآیند صورت میگیرد ممکن است در پروسه ای خسارات ناشی از توقف آنقدر زیاد باشد که سرمایه گذاری اولیه برای داشتن یک سیستم H در مقابل آن ناچیز محسوب شود.
سیستم های H در S7 فقط توسط S7-400H قابل پیاده سازی هستند و دو مدل CPU نیز برای آنها ارائه شده است CPU 414-4 H و CPU 417-4 H

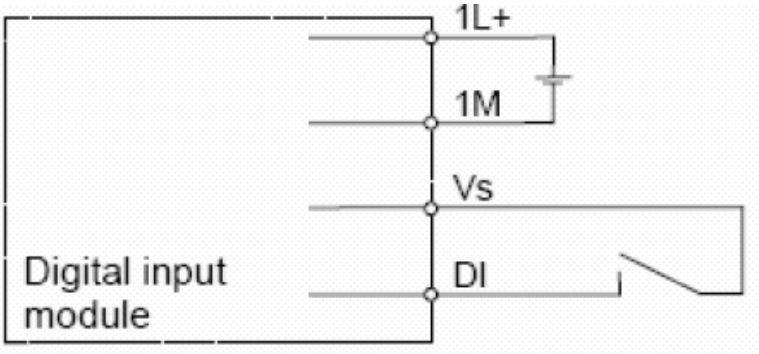
### ۲-۱۶ I/O Redundancy چیست ؟

در فرآیند های حساس ممکن است بالا بودن قابلیت اطمینان فقط در سطح سخت افزار CPU کافی نباشد و بالا بودن میزان دسترسی در سطح Field و کارتهای I/O نیز ضرورت پیدا کند . از این نظر در کنار Hardware Redundancy بحث دیگری به نام I/O Redundancy مطرح میگردد.

در مواردی که اهمیت I/O بالاست میتوان با اتخاذ تمهیدات خاص قابلیت اطمینان سیستم را بالا برد به موارد زیرکه برای یک ورودی دیجیتال بحث شده توجه کنید:

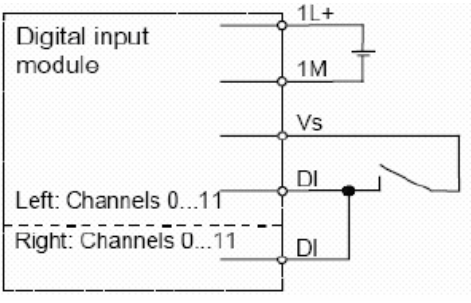
۱- اتصال ساده بدون Redundancy

در این روش که شکل نمونه آن در زیر نشان داده شده است کمترین قابلیت اطمینان وجود دارد و اگر کابل ارتباطی یا سوئیچ مشکل پیدا کند همینطور اگرکانال مربوطه در کارت DI یا کارت DI دچار اشکال شود سیگنال از دست میرود:



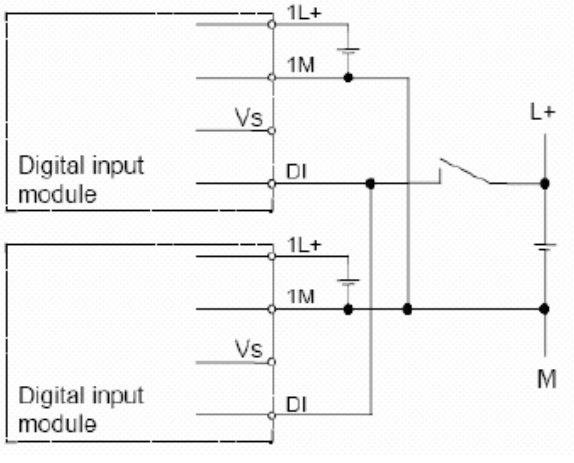
۲- اتصال با کابل دوبل و کارت واحد

در این روش سنسور توسط دو کابل ارتباطی مجزا به کانالهای مختلفی از یک کارت متصل شده است بنابراین اگر یکی از کابلهای ارتباطی یا یکی از کانال های مربوطه در کارت DI مشکل پیدا کند سیگنال باز هم قابل دسترس است ولی اگر سوئیچ یا کارت DI مشکل پیدا کند سیگنال از دست میرود.



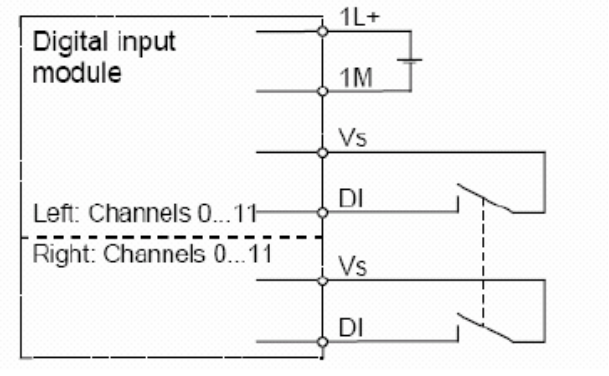

۳- اتصال با کابل دوبل و کارت دوبل

در این روش سنسور توسط دو کابل ارتباطی مجزا به کانالهای مختلفی از دو کارت مجزا متصل شده است بنابراین فقط اگر سوئیچ مشکل پیدا کند سیگنال از دست میرود ولی در بقیه موارد مانند اشکال در کابل ارتباطی یا کانال یا کارت سیگنال باز هم قابل دسترس خواهد بود .



۴- اتصال با کابل دوبل و کارت واحد و سنسور دوبل

در این روش سنسور بصورت دوبل است یعنی یک سنسور که همزمان دو خروجی دارد یا دو سنسور هر کدام با یک خروجی .در هر کدام از حالات فوق نهایتا دو خروجی داریم که میتوان آنها را به کانالهای مجزای کارت DI متصل کرد در اینجا قابلیت اطمینان بالاتر است ولی اگر کارت DI مشکل پیدا کند سیگنال از دست خواهد رفت.

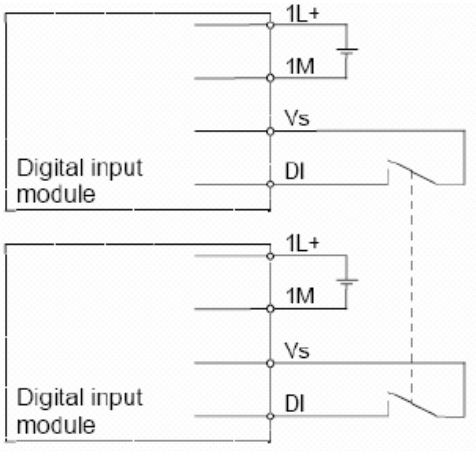



۵- اتصال با کابل دوبل و کارت دوبل و سنسور دوبل

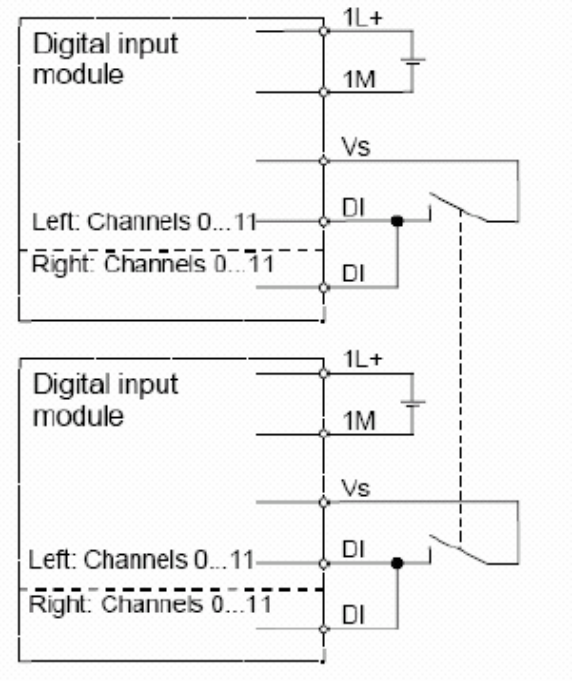
در این روش همه اجزا Redundant هستند و قابلیت اطمینان بالاست

با توجه به شکلهای زیر میتوان دید که حالت **ج** از **ب** مطمئن تر و **ب** از **الف** مطمئن تر است

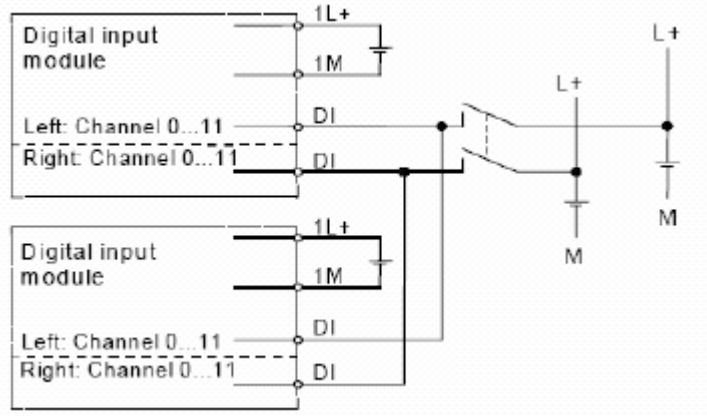
**الف )**



ب)


Digital input module
1L+
1M
Vs
DI
Left: Channels 0...11
Right: Channels 0...11
DI
Digital input module
1L+
1M
Vs
DI
Left: Channels 0...11
Right: Channels 0...11
DI

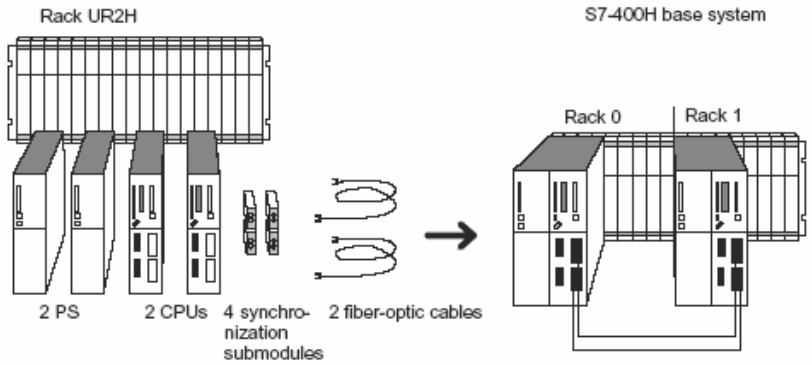
ج)


Digital input module
1L+
1M
Left: Channel 0...11
Right: Channel 0...11
DI
DI
L+
L+
M
M
Digital input module
1L+
1M
Left: Channel 0...11
Right: Channel 0...11
DI
DI

### ۱۶-۳ اجزای سخت افزاری سیستم H

سیستم H میتواند دارای اجزای زیادی باشد از CPU و کارتهای Central گرفته تا شبکه و کارت های I/O ولی حداقل سخت افزار برای سیستم H که به آن سیستم پایه S7-400H می گوییم شامل موارد زیر است:

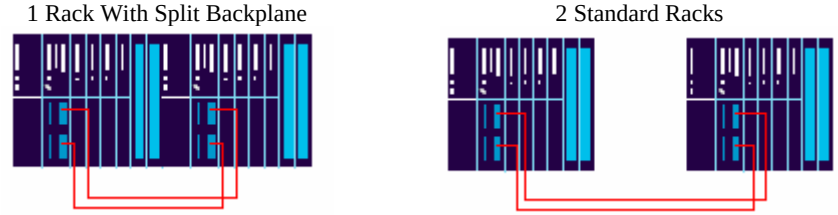
۱- دو CPU
۲- دو منبع تغذیه
۳- چهار مدول سنکرون سازی(۲ تا برای هر CPU )
۴- دو کابل فیبر نوری
۵- رک




پس از بسته شدن این اجزا CPU که در رک صفر قرار می گیرد را بنام CPU0 و CPU که در رک یک قرار می گیرد را CPU1 می نامیم.

### Rack برای S7-400H

انتخاب رک برای سیستم S7-400H میتواند به دو روش انجام شود. روش اول استفاده از رک خاص UR2H روش دوم استفاده از دو رک معمولی UR1 یا UR2 .

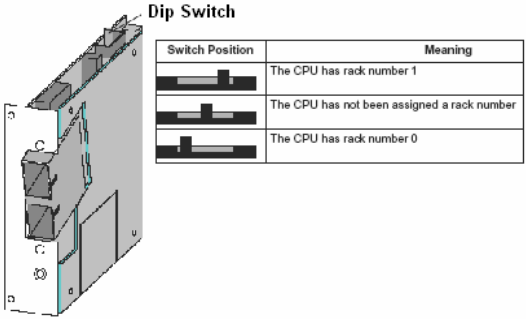



### منبع تغذیه

می توان از منابع تغذیه 10A و 20A استاندارد استفاده کرد یک PS در هر Rack برای اطمینان زیادتر در هر Rack دو منبع تغذیه از نوع Redundant گذاشت.اگر از رک های معمولی UR1 و UR استفاده شود لازم است رک مزبور قابلیت استفاده از منابع تغذیه Redundant را داشته باشد.

## مدولهای سنکرون سازی

این مدولها برای اتصال CPU ها به یکدیگر از طریق فیبر نوری به کار می روند روی هر CPU دو مدول سنکرون سازی (SYNC) قرار می گیرد. روی مدول های SYNC مانند شکل زیر یک DIP Switch وجود دارد که دارای وضعیت 0 و 1 است برای رک صفر آنرا روی صفر و برای رک دیگر آنرا روی یک تنظیم میکنیم با روشن شدن سیستم ابتدا آنکه روی صفر تنظیم شده بعنوان Master کار خواهد کرد.

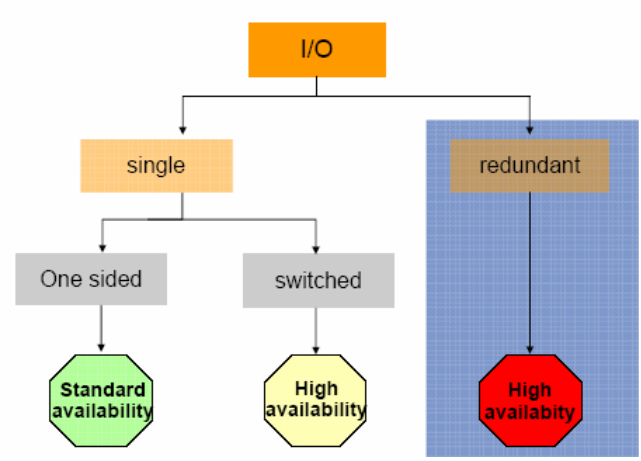



## کابل فیبر نوری

بین هر دو مدول SYNC یک زوج فیبر متصل میشود یعنی کلاً دو زوج فیبر نوری بین دو CPU بعنوان ارتباط Redundant قرار می گیرد.

## مدولهای ورودی وخروجی (I/O)

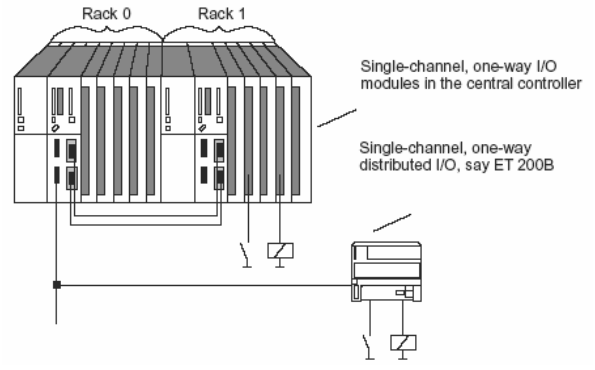
مدولهای ورودی وخروجی که می توانند در کنار خود CPU یا روی رک اضافی یا روی شبکه قرار بگیرند را میتوان مطابق شکل بعد دسته بندی کرد.



بر این اساس I/O ها بصورت زیر خواهند بود:

**۱- تک کاناله - تک مسیره معمولی (Single Channel ,One Way I/O )**

این کارت ها روی یکی از Subsystem ها (یعنی مثلاً در رک ۰ یا رک ۱ ) قرار می گیرند و فقط توسط یکی از دو سیستم آدرس دهی می شوند و I/O های متصل به این کارتها فقط می توانند توسط یکی از دو CPU استفاده شوند. این I/O ها را می توان در رک های اضافی یا روی شبکه DP بکار برد.این روش فقط برای I/O هایی که در دسترس بودن آنها معمولی (نه با اولویت زیاد) باشد می تواند بکار رود.



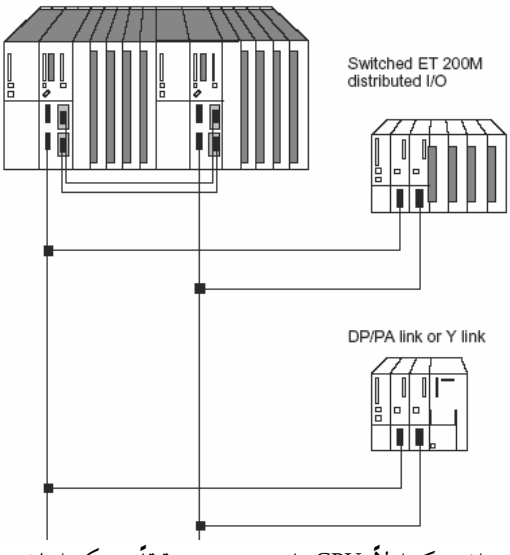

وقتی I/O های فوق فقط توسط یکی از سیستم ها خوانده شوند بطور اتوماتیک توسط لینک ارتباطی(فیبر نوری) به CPU دیگر نیز انتقال داده می شود ، و در برنامه ای که در هر دوی CPU ها در حال اجراست از آن استفاده می شود.بنابراین مهم نیست که این I/O به Master CPU متصل باشد یا Standby CPU ، بشرط اینکه مدکاری Redundant باشد.در مد Single کارتهای I/O فوق فقط برای همان CPU قابل دسترسی است و CPU دیگر با آنها ارتباط ندارد.

**۲- تک کاناله - دو مسیره Single-Channel,Switched I/O**

این I/O ها در مد کاری Redundant برای هر دو سیستم قابل دسترسی و قابل آدرس دهی هستند و در مد کاری Single نیز Master می تواند به این I/O ها دسترسی داشته باشد بنابراین قابلیت اطمینان این روش بیشتر از نوع قبلی است.این I/O ها را به دوطریق میتوان متصل نمود:

روش اول : با استفاده از یک نوع ET200M که دارای اتصال Redundant است (IM153-2 و IM153-2FO ).
هر کدام از IM های فوق به یکی از سیستم ها از طریق شبکه متصل می شوند. (شکل بعد)

روش دوم : از طریق یک وسیله به نام Y-Link که بین دو باس شبکه DP قرار میگیرد و از سمت دیگر به وسیله یا I/O مورد نظر متصل میشود . Y-Link همان کوپلر DP/PA میباشد . این روش برای اتصال I/O ها و وسایل معمولی مناسب است.




در روشهای فوق باید توجه داشت که اولاً CPU های دو سیستم دقیقاً در یک اسلات قرار گرفته باشند ثانیاً DP Master های دو سیستم یکسان باشند ، مثلاً هر دو از پورت DP روی CPU باشند ، نه اینکه یکی از پورت DP و دیگری از کارت CP گرفته شده باشد.

در این روش در مد Redundant هر کدام از سیستم ها می توانند به I/O ها دسترسی داشته باشند. اطلاعات بطور اتوماتیک از طریق لینک سنکرون سازی تبادل و مقایسه می شود به هر حال مقدار واحدی در هر زمان برای یک ورودی یا خروجی توسط هر دو سیستم استفاده می گردد.

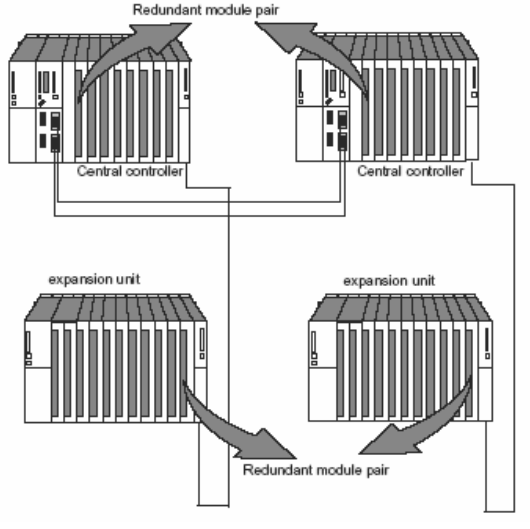
در استفاده از ET200M توجه شود که S7-400H فقط یکی از IM ها را در هر لحظه استفاده می کند .IM فعال را می توان از روشن بودن چراغ ACT روی آن شناسایی کرد . بنابراین وقتی دو IM153-2 داریم یکی بعنوان Active Channel و دیگری به نام Passive Channel شناخته می شود. اگر چه در سیکل DP با هر دو ارتباط برقرار می شود ولی فقط اطلاعات کانال Active در اختیار برنامه قرار می گیرد .

در شرایط بروز اشکال روی I/O :

- اگر کانال Active دچار اشکال شود سیستم بطور اتوماتیک روی کانال دیگر سوئیچ شده وآن کانال بصورت Active در می آید ، در اینحالت I/O Redundancy از بین رفته و OB70 که مربوط به خطای Redundancy است صدا زده می شود . این OB کد خطای W#16#73A3 را بر می گرداند.
- وقتی خطای فوق بر طرف شدI/O Redundancy مجدداً برقرار می شود و OB70 کد W#16#72A3 را بر می گرداند . در این شرایط کانال Active بکار خود ادامه می دهد و کانالی که مشکل آن رفع شده بصورت Passive در می آید.
- اگر در حالی که کانال قبلی مشکل پیدا کرده کانال Active فعلی نیز دچار مشکل شود ، در این صورت کل Station مشکل پیدا می کند و OB86 صدا زده شده و کد W#16#39C4 را برمی گرداند.
- اگر DP-Master (یعنی CPU یا کارت CP ) دچار مشکلی شود که تمام Slave های آن را دچار مشکل کند مثلاً بیرون آمدن کانکتور اتصال یا اتصال کوتاه در کابل ، در اینصورت OB86 کد W#16#39C3 را برمی گرداند.

### ۳- دو کانالهٔ Redundant با استفاده از رک اضافی

در S7-400H می توان ۲۰ رک اضافی داشت. که رک ها با شمارهٔ زوج به CPU0 و رک ها با شمارهٔ فرد به CPU1 اختصاص داده می شوند.روی رک های اضافی میتوان از کارت های I/O با قابلیت Redundant استفاده کرد .این کارت ها I/O ها را بصورت Redundant می پذیرند و برای هر دو سیستم قابل دسترسی هستند.

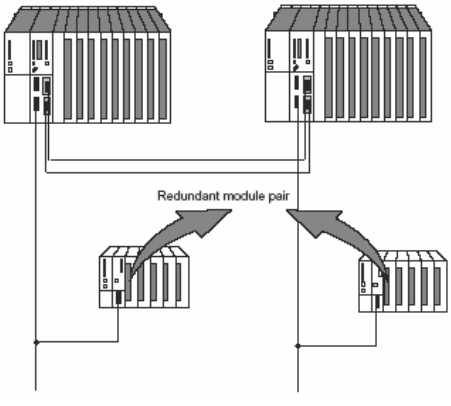


وقتی از کارتهای SM که Redundant هستند استفاده کنیم ، می توان این پیکربندی را ایجاد نمود . به اینصورت که از یک جفت کارت یکی در رک اضافی مربوط به CPU0 و دیگری در رک اضافی CPU1 قرار می گیرد.باید توجه داشت که اگر کارتی که در رک زوج قرار گرفته در اسلات شماره X باشد ، کارت متناظر آن در رک فرد نیز باید در همان شماره اسلات قرار گیرد.

لیست کارتهای SM با قابلیت Redundant در ادامه آورده شده و توضیحات لازم در مورد آنها ارائه شده است.

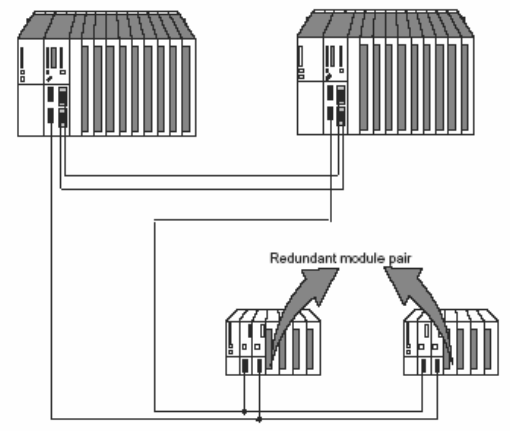
### ٤- دو کانالهٔ Redundant با استفاده از ET200M معمولی

در این روش ET200M از نوع سوئیچ شونده که دارای دو IM است نمی باشد ، بعبارت دیگر دو ET200M معمولی مجزا داریم ولی کارتهایی که روی آنهاست بصورت جفتی و Redundant می باشند. در اینحالت باید دقت کرد که آدرس کارتهای متناظر در دو طرف یکسان باشند.

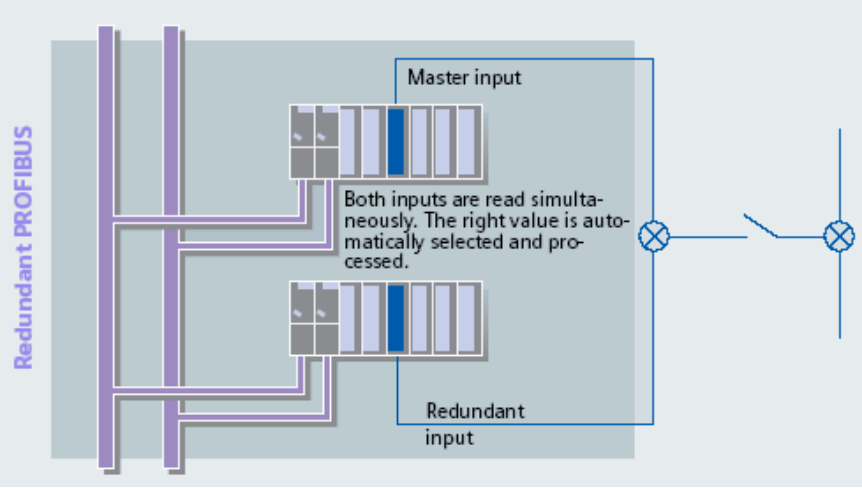



### ٥- دو کانالهٔ Redundant با استفاده از ET200M سوئیچ شونده

در این روش علاوه بر ET200M سوئیچ شونده یعنی IM153-2 کارتهای Redundant جفتی نیز استفاده شده است که قابلیت اطمینان آن بالاتر از نوع قبلی است( شکل بعد ). در اینجا نیز لازم است آدرس ها در کارتهای متناظر یکسان باشند.



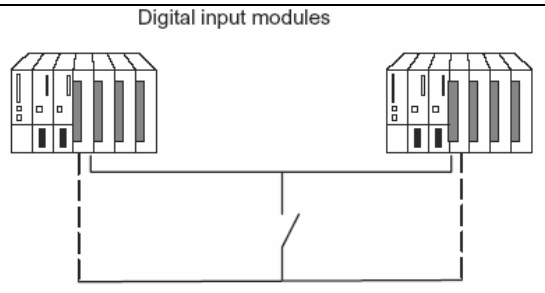
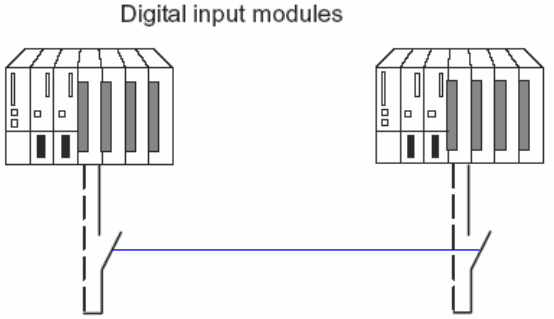

شکل زیر شماتیک اتصال یک سیگنال را به دو کارت I/O با قابلیت Redundancy نمایش میدهد.




لیست کارتهای SM که میتوانند بصورت Redundant روی رک های اضافی یا ET200M استفاده شوند در جدول بعد آمده است.

| Modules | Order Number | Remark |
|---|---|---|
| **Local: Redundant dual-channel DI** | | |
| DI 16 x 24 V DC Alarm | 6ES7 421-7BH00-0AB0 | |
| DI 32 x 24 V DC | 6ES7 421-1BL0x-0AA0 | |
| DI 32 x 120 V AC | 6ES7 421-1EL00-0AA0 | |
| **Distributed: Redundant dual-channel DI** | | |
| DI 24 x 24 V DC | 6ES7 326-1BK00-0AB0 | F module in standard operating mode |
| DI 8 x NAMUR [EEx ib] | 6ES7 326-1RF00-0AB0 | F module in standard operating mode |
| DI 16 x 24 V DC, Alarm | 6ES7 321-7BH00-0AB0 | |
| DI 16 x 24 V DC | 6ES7 321-1BH02-0AA0 | |
| DI 32 x 24 V DC | 6ES7 321-7BL00-0AA0 | |
| DI 32 x 24 V DC | 6ES7 321-7BH01-0AB0 | |
| DI 8 x 230 V AC | 6ES7 321-1FF01-0AA0 | |
| DI 16 x Namur | 6ES7 321-7TH00-0AB0 | |
| DI 4 x Namur | 6ES7 321-7RD00-0AB0 | |
| **Local: Redundant dual-channel AI** | | |
| AI 6x16Bit | 6ES7 431-7QH00-0AB0 | |
| **Distributed: Redundant dual-channel AI** | | |
| AI 6 x 13 bits | 6ES7 336-1HE00-0AB0 | F module in standard operating mode |
| AI 8 x 12 bits | 6ES7 331-7KF02-0AB0 | |
| AI 8 x 16 bits | 6ES7 331-7NF00-0AB0 | |
| AI 4 x 15 bits | 6ES7 331-7RD00-0AB0 | |
| **Local: Redundant dual-channel DO** | | |
| DO 32 x 24V DC / 0.5A | 6ES7 422-7BL00-0AB0 | |
| DO 16 x 120 / 230V AC / 2A | 6ES7 422-1FH00-0AA0 | |
| **Distributed: Redundant dual-channel DO** | | |
| DO 10 x 24 V DC / 2 A | 6ES7 326-2BF00-0AB0 | F module in standard operating mode |
| DO 32 x 24 V DC / 0.5 A | 6ES7 322-1BL00-0AA0 | |
| DO 8 x 24 V DC / 2 A | 6ES7 322-1BF01-0AA0 | |
| DO 8 x 24 V DC / 0.5 A | 6ES7 322-8BF00-0AB0 | |
| DO 8 x 230 V AC / 2 A | 6ES7 322-1FF01-0AA0 | |
| DO 16 x 24 V DC / 0.5 A | 6ES7 322-8BH00-0AB0 | |
| DO 16 x 24 V / 10 nA (Ex) | 6ES7 322-5SD00-0AB0 | |
| **Distributed: Redundant dual-channel AO** | | |
| AO 4 x 12 bits | 6ES7 332-5HD01-0AB0 | |
| AO 8 x 12 Bit | 6ES7 332-5HF00-0AB0 | |
| AO 4 x 15 Bit | 6ES7 332-5RD00-0AB0 | |
| AO 8 x 12bit | 6ES7 332-5HF00-0AB0 | |

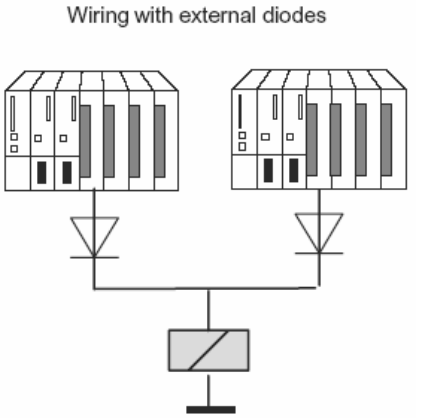
## ۱- کارتهای ورودی دیجیتال DI

| | |
|---|---|
| الف) بدون سنسور Redundant |  |
| ب) با سنسور Redundant |  |

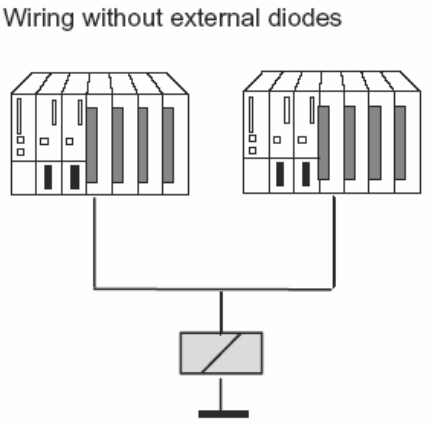
## ۲- کارتهای خروجی دیجیتال DO

مانند شکل زیر به دو دسته تقسیم می شوند:

الف) با دیود خارجی




ب) بدون دیود خارجی



جدول زیر نشان می دهد که کدامیک از کارتهای DO را می توان با یا بدون دیود بست:

| Modules | with diodes | without diodes |
|---|---|---|
| 6ES7 422-7BL00-0AB0 | X | – |
| 6ES7 422-1FH00-0AA0 | – | X |
| 6ES7 326-2BF00-0AB0 | X | X |
| 6ES7 322-1BL00-0AA0 | X | – |
| 6ES7 322-1BF01-0AA0 | X | – |
| 6ES7 322-8BF00-0AB0 | X | X |
| 6ES7 322-1FF01-0AA0 | – | X |
| 6ES7 322-8BH00-0AB0 | X | – |
| 6ES7 322-5SD00-0AB0 | X | – |

دیود های مناسب برای این کار انواع IN4003……IN4007 یا سایر انواعی که مشخصات زیر را داشته باشند :

U-r>= 200V
I-F>= 1A

## ۳- کارتهای ورودی آنالوگ AI

برای کارتهای AI که بصورت Redundant به کار می روند سه پارامتر وجود دارد که باید مشخص شود:

۱- پنجره تولرانس Tolerance Window

یعنی درصدی از مقدار سیگنال که اگر سیگنال AI در آن بازه باشد معتبر است.وقتی دو ورودی AI هر دو در یک پنجره تولرانس قرار گیرند هر دو توسط سیستم یکسان فرض می شوند.

۲- زمان شناسایی تناقص Discrepancy Time

ماکزیمم زمانی که بعد از سپری شدن آن ، خطا آشکار می گردد.

**تذکرات مهم :**

- اگر یک سنسور با دو خروجی استفاده شود ، زمان پیش فرض سیستم کافی است.
- اگر دو سنسور جدا استفاده شوند این زمان را افزایش دهید (بویژه برای سنسورهای دما)

۳- مقدار اعمالی به برنامه Applied value

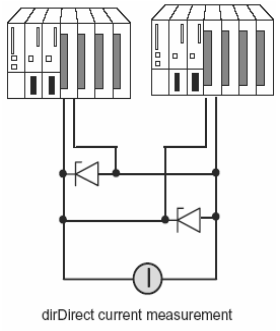
پس از مقایسه دو ورودی AI ، مقدار نهایی که در اختیار برنامه قرار می گیرد را Applied-value گویند.

شکل های بعد نمونه هایی از ورودی های آنالوگ Redundant را نشان می دهد.

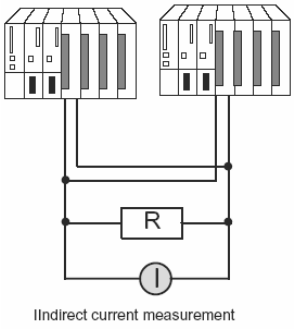
الف)ورودی از جنس ولتاژ :این ورودی بصورت موازی به هر دو کارت متصل می شود(شکل ۱)

ب) ورودی جریان بصورت غیر مستقیم : می توان جریان را به ولتاژ تبدیل و آن را در کارت AI با ورودی ولتاژ استفاده کرد شبیه حالت الف (شکل ۲ ). این برای ترانسدیوسرهای ۲ سیمه و ۴ سیمه امکان پذیر است مقاومت شنت فوق می تواند ۵۰ یا ۲۵۰ اهم باشد (جدول صفحه بعد)

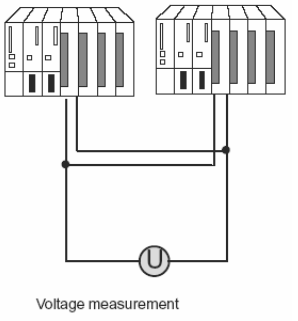
| Resistance | 50 Ohm | 250 Ohm | |
|---|---|---|---|
| Current measuring range | +/–20mA | +/–20mA | 4...20mA |
| Input range to be configured | +/–1V | +/–5 V | 1...5V |
| Measuring range cube positioning | “A” | “B” | |
| Resolution | 12bit+sign | 12bit+sign | 12bit |




شکل ۳




شکل ۲

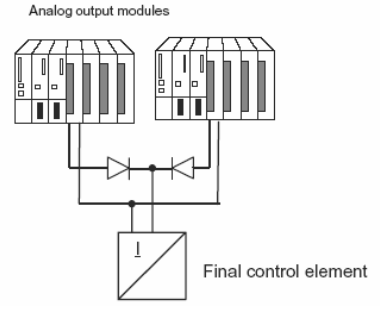



شکل ۱

ترانسدیوسرهای جریان ۲ سیمه با تغذیه بیرونی را می توان مطابق شکل ۳ متصل نمود.

## کارتهای خروجی آنالوگ Redundant

شکل روبرو کارتهای AO از نوع Redundant را نشان می دهد .




## مثالی از کاربرد Redundant I/O در برنامه

در مثال زیر دو کارت DI از نوع Redundant با فرضیات زیر استفاده شده اند.

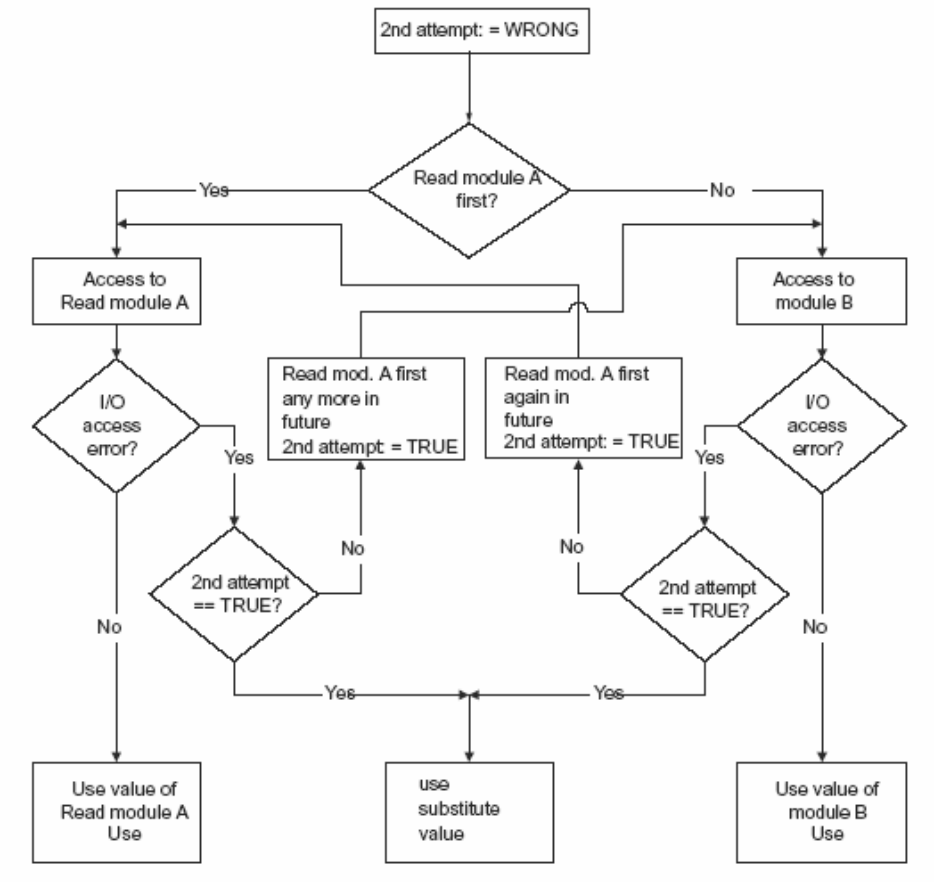
- مدول A در Rack0 با آدرس بیس 8
- مدول B در Rack1 با آدرس بیس 12

روش کار بدین صورت است که:

- توسط Master CPU در برنامه OB1 یکی از دو جدول A یا B مستقیماً خوانده می شود اگر خطایی وجود نداشت عمل پردازش با توجه به مقدار خوانده شده انجام می شود.

- اگر خطای خواندن در مدول A پیش آمد OB1 در بار( attempt) دوم مطابق فلوچارت صفحهٔ بعد به مدول B مراجعه می کند و مقدار را می خواند و مبنای پردازش خود قرار می دهد.
- اگر مدول B نیز مشکل داشت در اینصورت برنامه با مقدار Substitue Value کار خواهد کرد.

فلوچارت اجرای برنامه در OB1 بصورت زیر میباشد:




برنامه OB1 و OB122 در صفحه بعد آمده است.

Table 7-6 OB 1

| STL | | Explanation |
|---|---|---|
| | NOP 0; | |
| | SET; | |
| | R VERSUCH2; | //Initialization |
| | A BGA; | //Read module A first? |
| | JCN WBGB; | //If No, continue with module B |
| WBGA: | SET; | |
| | R PZF_BIT; | //Delete PZF bit |
| | L PED 8; | //Read CPU 0 |
| | U PZF_BIT; | //Was PZF detected in OB 122? |
| | SPBN PZOK; | //If no, process access OK |
| | U VERSUCH2; | //Was this access the second attempt? |
| | SPB WBG0; | //If yes, use substitute value |
| | SET; | |
| | R BGA; | //Do not read module A first any more<br>//in future |
| | S VERSUCH2; | |
| WBGB: | SET; | |
| | R PZF_BIT; | //Delete PZF bit |
| | L PED 12; | //Read CPU 1 |
| | U PZF_BIT; | //Was PZF detected in OB 122? |
| | SPBN PZOK; | //If no, process access OK |
| | U VERSUCH2; | //Was this access the second attempt? |
| | SPB WBG0; | //If yes, use substitute value |
| | SET; | |
| | S BGA; | //Read module A first again in future |
| | S VERSUCH2; | |
| | JU WBGA; | |
| WBG0: | L ERSATZ; | //Substitute value |
| PZOK: | | //The value to be used is in Accumulator1 |

Table 7-7 OB 122

| STL | Explanation |
|---|---|
| | // Does module A cause PZF? |
| L OB122_MEM_ADDR; | //Logical base address affected |
| L W#16#8; | |
| == I; | //Module A? |
| SPBN M01; | //If no, continue with M01 |
| | //PZF upon access to module A |
| SET; | |
| = PZF_BIT; | //Set PZF bit |
| SPA CONT; | |
| | // Does module B cause PZF? |
| M01: NOP 0; | |
| L OB122_MEM_ADDR; | //Logical base address affected |
| L W#16#C; | |
| == I; | //Module B? |
| SPBN CONT; | //If no, continue with CONT |
| | //PZF upon access to module B |
| SET; | |
| = PZF_BIT; | //Set PZF bit |
| CONT: NOP 0; | |

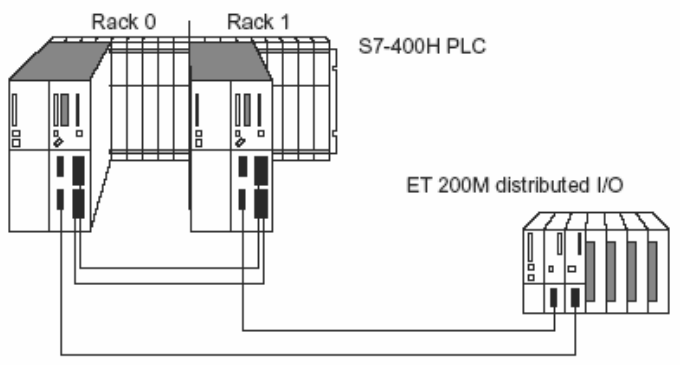
## ۱۶-٤ روش نصب یک سیستم نمونه

سیستم S7-400H را همراه با ET200M سوئیچ شونده در نظر بگیرید .در این پیکر بندی از اجزا زیر استفاده می شود.

- رک UR2H
- دو منبع تغذیه PS407 10A
- دو CPU از خانواده H ( CPU 417-4H یا CPU 414-4H )
- چهار مدول SYNC
- دو کابل نوری
- دو عدد IM153-2
- یک عدد کارت DI از نوع SM 321 DI 16 × DC24V

- یک عدد کارت DO از نوع SM 322 DO 16 × DC24V
- سایر وسایل جنبی مورد نیاز

این اجزا را مطابق شکل زیر متصل می کنیم :




برای نصب این سیستم مراحل زیر بترتیب لازم است انجام شود:

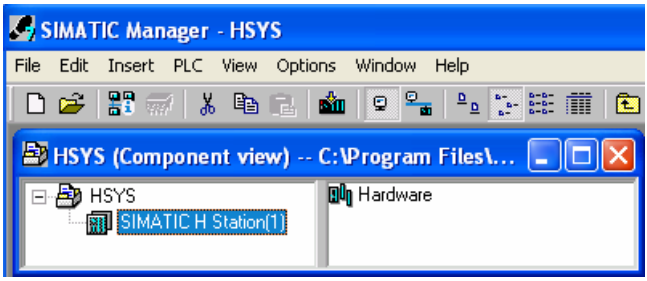
۱. اجزای فوق را مطابق روشی کلی نصب اجزای S7-400 بهم متصل می کنیم.
۲. توسط سوئیچ های روی مدولهای SYNC شماره رک را در هر طرف ست می کنیم. این تنظیم پس از روشن شدن PLC یا ری ست شدن آن توسط CPU استفاده می شود. اگر بطور غلط تنظیم شده باشد ، ارتباط On Line قطع شده و CPU در حالات خاصی RUN نخواهد شد.
۳. مدولهای SYNC را در محل خود قرار داده و توسط پیچ های تعبیه شده آنها را محکم کنید.
۴. فیبر های نوری را به SYNC ها اتصال دهید و SYNC های بالایی دو PLC را بهم و SYNC های پایینی را نیز با فیبر جداگانه بهم وصل کنید. قبل از روشن کردن CPU کابلهای نوری باید متصل شده باشند ، در غیر این صورت هر دو CPU بعنوان Master کار خواهند کرد.
۵. کارتهای ET200M را نیز روی ریل قرار دهید بدیهی است دو IM باید در کنار هم قرار گیرند.
۶. PG را به CPU0 متصل کنید این CPU در سیستم بعنوان Master خواهد بود.
۷. پس از روشن شدن منبع تغذیه حدود ۸ ثانیه وقت صرف چک کردن RAM می شود. در طول این مدت CPU نمی تواند توسط پورت MPI شناسایی شود وچراغ STOP چشمک می زند.اگر باطری Backup روی CPU موجود باشد. مد تست فوق در روشن شدن های بعدی ظاهر نخواهد شد.
۸. سوئیچ هر دو CPU را روی STOP قرار دهید.

## ۱۶-۵ نحوه پیکر بندی در Step7

برای پیکربندی سیستم H در Step7 قدمهای زیر را بردارید:

۱. Simatic Manager را اجرا کرده و یک پروژه جدید با نام دلخواه ایجاد کنید.

۲. ازمنوی Insert یا با کلیک راست یک H Station وارد کنید.

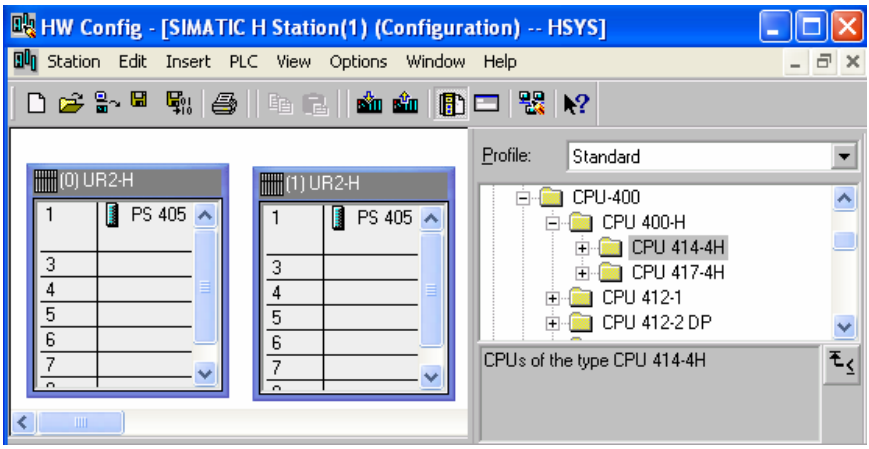



۳. روی Hardware Click کرده تا برنامه Hwconfig باز شود.

۴. از کاتالوگ برنامه دو رک UR2H از زیر مجموعه Rack-400 وارد کنید. استفاده از رک UR2H الزامی نیست میتوان دو رک معمولی بکار برد.

۵. از کاتالوگ منبع تغذیه دلخواهی را انتخاب کرده و در رک اول و مجدداً در رک دوم قرار دهید.توجه شود که میتوان برای داشتن قابلیت اطمینان بیشتر در هر رک دو منبع تغذیه از نوع Redundant قرار داد.

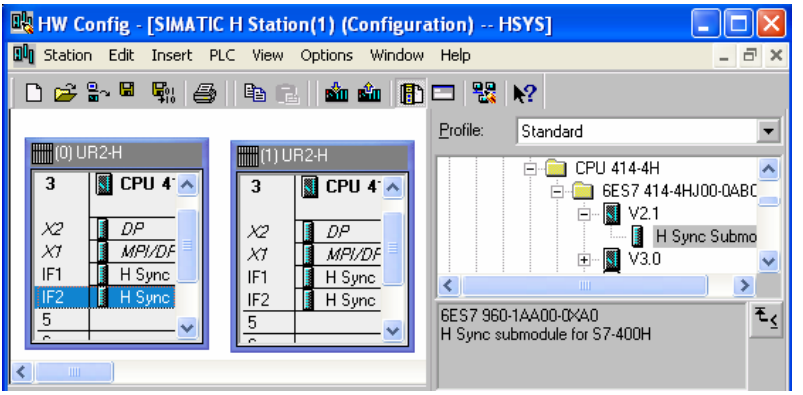
۶. در کاتالوگ از زیر مجموعه CPU-400 قسمت CPU400H را باز کنیدمشاهده میکنید که فقط دو مدل CPU مانند شکل وجود دارد که هر کدام حاوی CPU با نسخه های مختلف هستند.



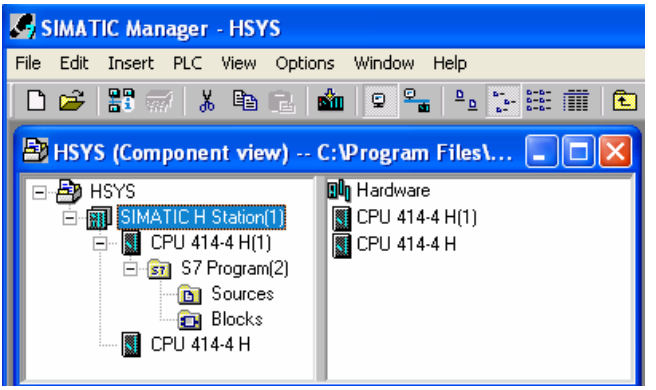
۷. یکی از CPU ها را انتخاب کرده و در رک اول قرار دهید مجدداً همین CPU را در رک دوم وارد کنید . توجه شود که در سیستم های H هر دو CPU بایستی کاملاً مشابه باشند حتی Version و Order number آنها بایستی یکی باشد.

۸. اگر در این مرحله Save & Compile یا Consistency Check را اجرا کنیم خطایی که مربوط به عدم وجود SYNC Module است گزارش می شود.

۹. در کاتالوگ از زیر مجموعه CPU که وارد کرده ایم مدول SYNC را انتخاب کرده و در هر رک دو عدد از آنها را در زیر CPU وارد میکنیم.



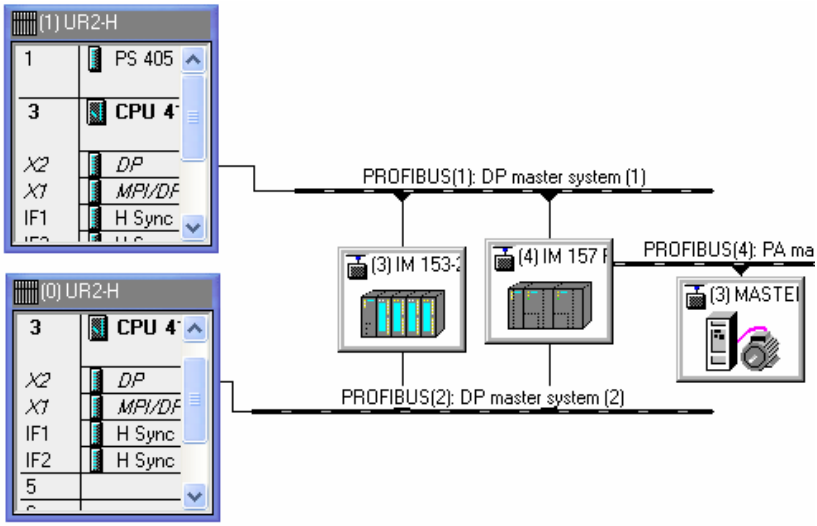

تا اینجا پیکر بندی سیستم پایه S7-400H انجام شده است. پیکر بندی را ذخیره میکنیم.اکنون اگر به Simatic Manager بازگردیم CPU ها را در زیر مجموعه سخت افزار مشاهده میکنیم اما جالب اینست که فقط یکی از آنها دارای زیر شاخه S7 Program است . از همین جا بایستی دریافت که اطلاعات و برنامه فقط به یک CPU دانلود میشود و CPU دیگر آنها را از طریق لینک نوری دریافت میکند.



### پیکر بندی با ET200M سوئیچ شونده و Y-Link

برای پیکربندی ET200M فوق الذکر در برنامه HWconfig ابتدا دو شبکه DP از پورت CPU ها یا از طریق دو کارت CP مشابه فعال میکنیم. سپس از کاتالوگ Profibus-DP از زیر مجموعه ET200M مدول IM153-2 با قابلیت Redundancy را انتخاب میکنیم و روی یکی از خطهای شبکه قرار میدهیم میبینیم که بطور اتوماتیک به هر دو شبکه متصل میشود. پس از این کار کارتهای دلخواه را در آن وارد می نماییم.

برای استفاده از Y-Link ار کاتالوگ Profibus-DP از زیر مجموعه DP/PA Link مدول IM157 با قابلیت Redundancy را انتخاب کرده وآنرا روی خط شبکه قرار می دهیم در پنجره ای که ظاهر میشود بسته به نیاز خروجی DP یا PA را انتخاب میکنیم مشاهده خواهیم کرد که مدول فوق بطور اتوماتیک به هر دو باس شبکه متصل شده و یک خروجی DP یا PA نیز فعال میشود. مدولهایی که مستقیماً قابلیت Redundancy ندارند را روی این باس سوم قرار میدهیم تا برای هر دو CPU قابل دسترس باشند.




### نکات پیکر بندی سیستم های H از طریق Step7

برای پیکر بندی S7-400H ، نرم افزار Step7 ممکن است کافی نباشد .Step7 V5.2 نیاز به نرم افزار جداگانه بنام H-System دارد که باید روی Step7 نصب شود. اگر نسخه Step7 V5.3 Proffessional به بالا استفاده شود نیاز به نرم افزار جداگانه نمی باشد.

پیکربندی S7-400H بجز در موارد زیر مشابه سیستم های معمولی می باشد:

۱- OB70 و OB72 مربوط به وقفه های Redundancy می باشند.این OB ها و سایر OB های وقفه اگر به CPU دانلود نشده باشند ، در صورت وقوع خطا سیستم H متوقف خواهد شد.

۲- در پیکر بندی سخت افزار باید CPU در اسلات های مشابه Rack0 و Rack1 قرار گیرد.

۳- DP-Master هایی که توسط IM یا CP ایجاد می شوند باید در اسلات های مشابه در دو رک قرار گیرند.

۴- مدولهایی که بصورت Redundant قرار می گیرند مانند دو CPU یا دو IM153-2 باید دقیقاً از نظر کد سفارش و Version و سایر مشخصات یکسان باشند.

۵- سیستم H ماکزیمم می تواند ۲۰ رک اضافی داشته باشد. رک های زوج فقط می توانند به CPU0 اختصاص یابند و رک های فرد فقط می توانند به CPU1 اختصاص یابند.

۶- مدولهایی که به باس شبکه (C-BUS ) متصل می شوند می توانند فقط در رک های ۰ تا ۶ قرار گیرند.

۷- وقتی از کارت CP برای ارتباط Fault Tolerant در رک اضافی استفاده می کنید به شماره رک توجه داشته باشید . این شماره باید ترتیبی باشد و با عدد زوج شروع شود مثلاً رک های ۲ و ۳ درست و رک های ۳ و ۴ غلط است.

۸- اگر از ۹ عدد DP-Master یا بیشتر استفاده شود شماره رک به این DP-Master ها اختصاص داده می شود . در اینحالت تعداد رک اضافی کاهش می یابد .

۹- ساده ترین راه برای پیکر بندی سخت افزار بصورت Redundant اینست که یک رک ابتدا با تمام اجزایش پیکربندی شود سپس آن را کپی کنید تا همان اجزا با رک جدید تکرار شوند سپس در صورت لزوم نسبت به تغییر برخی پارامترهای مورد نیاز اقدام نمایید.

۱۰- پس از اتمام پیکر بندی توسط Step7 عمل دانلود صرفاً به CPU0 انجام میشود.

## تنظیم پارامترهای CPU

- تنظیم پارامترها فقط برای CPU0 امکان پذیر است . بدیهی است هر مقداری که برای پارامترهای CPU0 داده شود بطور اتوماتیک به CPU1 ارسال می گردد.
- در CPU1 فقط می توان دو مورد زیر را تغییر داد :
  ۱- آدرس CPU روی MPI
  ۲- آدرس CPU روی شبکه DP که متصل به پورت CPU شده است.
- سیکل اسکن را معمولاً بزرگ انتخاب می کنیم(مثلاً $_{MS}$ 6000 )
- PII را معمولاً کوچک انتخاب می کنیم ( کمی بیش از تعداد ورودی های واقعی )
- PIQ را نیز معمولاً کوچک انتخاب می کنیم (کمی بیش از خروجی های واقعی )

### ۶-۱۶ نحوه عملکرد سیستم های S7-400H

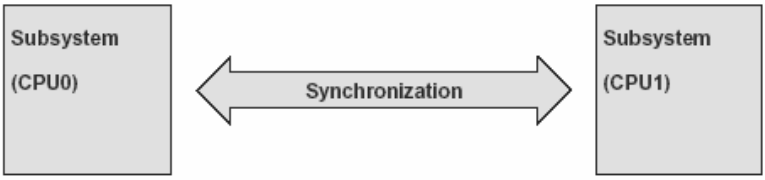
همانطور که ذکر شد S7-400H از دو Subsystem تشکیل شده است. یک CPU بعنوان Master و دیگر رزرو (Standby) است. برنامه همزمان توسط دو CPU بطور مشابه اجرا می شود. یک فرق بین عملکرد Master و Standby اینست که در صورت بروز خطا روی لینک Redundant آنکه رزرو است Stop شده ولی Master در مد RUN باقی می ماند.برای اختصاص دادن اینکه کدام Master باشد وکدام Standby باید توجه داشت که وقتی سیستم برای اولین بار روشن می شود ، اولین CPU یعنی CPU0 بعنوان Master و دیگری بعنوان رزرو تلقی می شود. در شرایط زیر جای Master و Standby عوض می شود :

۱- توقف یا خطای Master در مد Redundant

۲- CPU رزرو حداقل ۳ ثانیه قبل از Master شروع به کارکند.

۳- خطایی در مد Troubleshooting یافت نشود.(که بعداً توضیح داده خواهد شد. )

عمل سنکرون سازی دو CPU بصورت اتوماتیک توسط سیستم عامل CPU ها انجام می شود و لازم نیست کار خاصی توسط کاربر انجام شود.بنابراین کاربر مانند یک CPU400 معمولی کار برنامه نویسی را انجام می دهد.




عمل سنکرون سازی اصطلاحاً بصورت Event Driven است.این نحوه کار قبلاً در سیستم های H سریS5 یعنی S5-115H و S5-155H به کار می رفته و بدین معنا ست که عمل سنکرون سازی دیتا بین Master و Standby به محض وقوع هر رخدادی که ممکن است منجر به تغییر مد کاری Subsystem ها شود انجام می گیرد. در این شرایط دیتاها و موارد زیر بین دو CPU سنکرون می شوند :

- تایمرها
- دیتاهای مربوط به ارتباط با شبکه (Communication)
- وقفه ها
- دسترسی سیستم به I/O ها

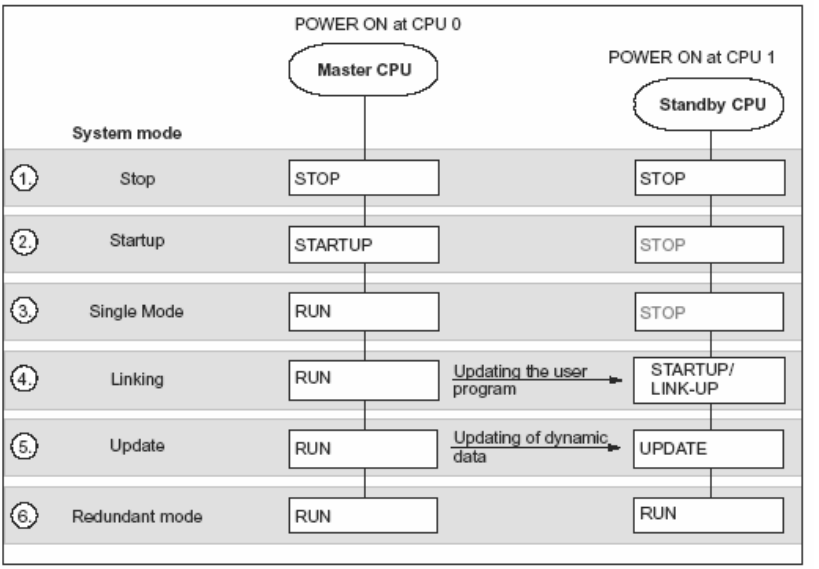
سوئیچ شدن از سیستم Master به Standby با سنکرون شدن های فوق از توقف پروسه وسیستم جلوگیری می نماید.

## مد کاری سیستم H

وقتی صحبت از مد کاری سیستم H می شود منظور مد کاری هر دو CPU است که باید با یک کلمه توصیف شود. جدول زیر مد کاری سیستم را بر اساس مد های کاری دو CPU نشان می دهد.

| System Modes of the S7-400H | Operating Modes of the two CPUs | |
|---|---|---|
| | Master | Standby |
| Stop | STOP | STOP, deenergized, DEFECTIVE |
| Startup | STARTUP | STOP, deenergized, DEFECTIVE, no synchronization |
| Single Mode | RUN | STOP, TROUBLESHOOTING, deenergized, DEFECTIVE, no synchronization |
| Linking | RUN | STARTUP, LINK-UP |
| Update | RUN | UPDATE |
| Redundant mode | RUN | RUN |
| Hold | HOLD | STOP, deenergized, DEFECTIVE |

وقتی یک سیستم Redundant از Stop به RUN می رود مد کاری سیستم از مراحل مختلفی مانند شکل زیر عبور می نماید.باید توجه داشت که تغذیه CPU0 قبل از CPU1 بر قرار می گردد.



مراحلی که در شکل فوق نمایش داده شد عبارتند از:

۱- وقتی منبع تغذیه وصل می شود هر دو CPU در مد STOP هستند.

۲- CPU0 راه اندازی شده و OB راه اندازی خود را اجرا می کند.

۳- اگر نتیجه راه اندازی رضایت بخش بود و مشکلی وجود نداشت CPU0 به مد RUN رفته وبصورت Single mode کار می کند.

۴- اگر CPU1 درخواست اتصال(LINK-UP) کند در این صورت دو CPU برنامه های خود را با هم مقایسه می کنند اگر اختلافی بین آنها باشد CPU0 برنامه CPU1 را با خود تطبیق می دهد.

۵- بعد از اینکه عمل LINK-UP بصورت رضایت بخش انجام شد در این وضعیت Master دیتاهای Dynamic ورودی ها و خروجی ها و تایمرها و دیتا بلاک ها و متعییر های حافظه هستند. بعد از این عمل محتویات حافظه دو CPU مانند هم خواهد بود.

۶- در این مرحله هر دو CPU در حالت RUN هستند و برنامه کاربری را به طور همزمان اجرا می کنند و در مد Redundant قرار می گیرند. این وقتی امکان پذیر است که دو CPU کاملاً مشابه وFirmware آنها یکسان باشد.

## دو مثال از عملکرد سیستم

### مثال ۱ : قطع منبع تغذیه یکی از سیستم ها

سیستم بصورت Redundant در حال کار است و :

۱- CPU0 بدلیل قطع شدن منبع تغذیه از کار می افتد در نتیجه:

- روی CPU1 چراغهای IF M2F و IF M1F و REDF روشن می شوند
- و CPU1 بلافاصله در مد کاری (Single Mode ) قرار گرفته و اجرای برنامه را ادامه می دهد.

۲- اشکال منبع تغذیه CPU0 رفع می شود وتغذیه بر می گردد در اینجاست که :

- CPU0 بطور اتوماتیک عمل LINK-UP و UPDATE را انجام می دهد.
- CPU0 به مد RUN رفته و شروع به کار می کند **ولی بعنوان Standby** .
- سیستم مجدداً بصورت Redundant کار خود را ادامه می دهد .

**مثال ۲ : قطع شدن کابل فیبر نوری**

در حالی که سیستم بصورت Redundant مشغول کار است :

۱. یکی از کابلهای FO قطع می شود ، در نتیجه چراغ REDF روی هر دو CPU و چراغ IFM1F یا IFM2F (بسته به اینکه کدام فیبر قطع شده است.) روی هر دو CPU روشن می شود.در اینحالت CPU اصلی (Master) CPU0 به حالت Single mode رفته و اجرای برنامه را ادامه می دهد.

۲. مجدداً کابل نوری وصل می شود.در اینحالت باید CPU1 که Standby بوده و اکنون در مد STOP است. توسط Step7 مجدداً Restart شود . پس از اینکار CPU1 بطور اتوماتیک عمل LINK-UP و UPDATE را انجام داده و مجدداً سیستم بحالت Redundant برمی گردد.

## نکات مربوط به مدهای کاری سیستم

**نکات مد STOP**

- وقتی CPU ها در مد STOP هستند و بخواهید پیکر بندی را به آنها دانلود کنید باید این کار را روی CPU اصلی (Master ) انجام دهید.
- ری ست کردن حافظه در مد STOP فقط حافظه همان CPU که عمل ری ست روی آن انجام می شود را پاک می کند ، بنابراین لازم است که آنها را یکی پس از دیگری ری ست کنید.

**نکات مد Startup**

- برای HCPU فقط امکان راه اندازی Cold و Warm وجود دارد و Hot Restart در آن امکان پذیر نیست.
- در مد راه اندازی CPU رزرو نقشی ایفا نمی کند .پیکر بندی به Master دانلود می شود . Master آن را با پیکر بندی واقعی چک می کند و اگر اختلافی بود رفتارش در این شرایط شبیه CPU های ۴۰۰ معمولی خواهد بود.

**نکات مدLINK-UP و UPDATE**

قبل از اینکه سیستم Fault-Tolerant مد Redundant را قبول کند ، Master CPU محتویات حافظه Standby CPU را چک وUpdate می کند.

در این مد Master CPU در مد RUN و CPU دیگر در مد LINK-UP یا UPDATE است.

جزئیات بیشتر در مورد LINK-UP بعداً خواهد آمد.

## نکات مربوط به مد RUN

وقتی سیستم در مد Redundant قرار گرفت هر دو CPU بصورت RUN هستند و هر دو برنامه را بصورت سنکرون چک می کنند .

در مد Redundant امکان تست برنامه توسط Breakpoint نیست ، در این مد اشکالات زیر منجر به خروج سیستم از حالت Redundant میگردد.

۱- اشکال روی یک CPU
۲- اشکال در لینک ارتباطی (مدول SYNC یا فیبر نوری)
۳- اشکال در مقایسه RAM

در مد Redundant نه تنها باید دو CPU کاملاً شبیه هم باشند بلکه باید مدولهای Redundant نیز یکسان باشند. بعنوان مثال اگر از IM153-2 استفاده شود باید هر دو دارای یک شماره سفارش ، یک Version و یک Firmware باشند.

## نکات مربوط به مد Hold

مد Hold مد خاصی است که فقط برای تست استفاده می شود.این مد فقط می تواند از مد Startup یا مد RUN مربوط به Single mode قابل دسترس باشد.وقتی سیستم H در مد Hold است امکان LINKUP و UPDATE نیست . در این حالت Standby CPU در مد STOP می ماند.بنابراین وقتی سیستم در مد Redundant است امکان استفاده از Hold وجود ندارد.

## حالت کاریTrouble shooting

اگر هر کدام از Event های زیر به وقوع پیوندد ، مد Trouble shooting فعال می شود :

۱- اگر وقتی سیستم در مد Redundant است OB121 فقط روی یکی از CPU ها صدا زده شود در این صورت این CPU به مد Trouble shooting می رود و CPU دیگر به عنوان Master کار را بصورت Single mode ادامه می دهد.
۲- اگر در مد Redundant خطای Checksum روی فقط یک CPU رخ دهد در اینصورت آن CPU به مد Troubleshooting رفته ودیگری به عنوان Master ولی بصورت Single Mode کار را ادامه می دهد.
۳- اگر در مد Redundant هنگام مقایسه RAM/PIQ خطایی وجود داشته باشد Stand By CPU به Troubleshooting رفته و Master CPU کار را بصورت Single Mode ادامه می دهد.

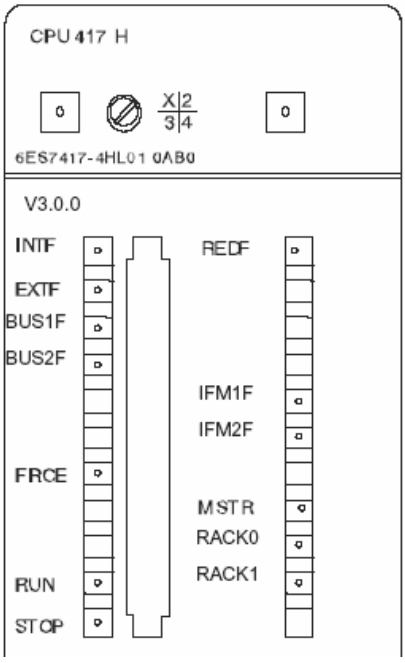
هدف از مد Troubleshooting آشکار سازی و Localize کردن CPU مشکل دار است در طول زمان جستجو برای اشکال Stand By CPU در حال Self-Test داخلی و Master CPU در مد RUN می ماند. اگر خطا آشکار شود CPU به مد Defective می رود و اگر اشکال وجود نداشته باشد CPU مجدداً LINK-UP شده و مد Redundant برقرار می گردد و جابجایی دو CPU برای ایجاد اطمینان است تا وقتی اشکال بعدی پیش آمد مطمئن باشیم که سخت افزار Master CPU قبلاً تست شده است ، در وضعیت جدید Standby CPU مشغول Self Test است وMaster CPU مشغول RUN ، بنابراین همیشه از وضعیت تست Master قبلی اطمینان حاصل می شود.

**تذکر:**

- در مدTrouble shooting امکان ارتباط Communication وجود ندارد.

## توضیح LED های روی CPU

شکل روبرو LED های روی یک CPU را نشان میدهد LED های سمت چپ شبیه CPU های 400 معمولی است و LED های سمت راست مخصوص نوع H است. همه موارد در جدول بعد توضیح داده شده اند.



| مفهوم | رنگ | LED |
|---|---|---|
| اشکال داخلی | قرمز | INTF |
| اشکال بیرونی | قرمز | EXTF |
| فعال بودن Force | زرد | FRCE |
| مد RUN ( جدول بعد را ببینید) | سبز | RUN |
| مد STOP ( جدول بعد را ببینید) | زرد | STOP |
| اشکال باس شبکه در MPI/PROFIBUS DP interface 1 | قرمز | BUS1F |
| اشکال باس شبکه در PROFIBUS DP interface 2 | قرمز | BUS2F |
| CPU بعنوان Master در حال کار است | زرد | MSTR |
| اگر با فرکانس 0.5 HZ چشمک بزند مد کاری Linking است<br>اگر با فرکانس 2 HZ چشمک بزند مد کاری Update است<br>اگر خاموش باشد Redundancy بدون مشکل است<br>اگر بصورت ثابت روشن باشد Redundancy مشکل دارد | قرمز | REDF |
| CPU در رک صفر قرار دارد | زرد | RACK0 |
| CPU در رک یک قرار دارد | زرد | RACK1 |
| اشکال در مدول SYNC اول | قرمز | IFM1F |
| اشکال در مدول SYNC دوم | قرمز | IFM2F |

در مورد مد RUN و مد STOP توضیحات در جدول زیر آمده است:

| مفهوم | LED | |
|---|---|---|
| | STOP | RUN |
| CPU در مد RUN است. | D | H |
| CPU در مد STOP است | H | D |
| CPU در وضعیت DEFECT است<br>چراغهای INTF و EXTF و FRCE نیز چشمک میزنند. . | B 2 Hz | B 2 Hz |
| CPU در مد HOLD است | H | B 0.5 Hz |
| سیستم در حال Restart شدن یا Warm Restart است | H | B 2 Hz |
| Self Test در حال اجراست | B 2 Hz | B 2 Hz |
| درخواست Memory Reset | B 0.5 Hz | x |
| در حال اجرای Memory Reset | B 2 Hz | x |

خاموش=D روشن =H چشمک زن با فرکانس ذکر شده =B نامشخص=X

**تأثیرات Link-up و Update روی عملکرد CPU**

این دوحالت توسط چراغ REDF روی CPU مشخص می شود.

- در حالت Linkup این دو چراغ با فرکانس 0.5 Hz چشمک می زند.
- در حالت Update این دوچراغ با فرکانس 2H چشمک می زند.

تأثیرات دو مد فوق روی عملکرد CPU در جدول زیر آمده است :

| عملکرد | Linking | Update |
|---|---|---|
| اجرای برنامه کاربری | اجرا می شود | به تعویق می افتد و پس ار انجام Update اجرا می شود. |
| حذف ، ایجاد ، فشرده سازی و Load کردن بلاک ها | ممکن نیست | ممکن نیست |
| | و اگر کارهای فوق در حال اجرا باشد امکان Linkup و Update نخواهد بود. | |
| اجرای فانکشن های ارتباطی و ارتباط با PG | امکان پذیر است | محدود شده به تعویق می افتد و پس از انجام Update اجرا می شود. |
| اجرای Self-Test توسط CPU | اجرا نمی شود | اجرا نمی شود |
| فانکشنهای Test و Monitor (On/Off) | امکان پذیر نیست | ممکن نیست |
| | و اگر کارهای فوق در حال اجرا باشد امکان Linkup و Update نخواهد بود. | |
| اتصال به Master CPU | اتصالات قبلی باقی می ماند امکان اضافه کردن اتصال جدید نیست . | اتصالات قبلی باقی می ماند امکان اضافه کردن اتصال جدید نیست |
| اتصال به Standby CPU | اتصالات قبلی شکسته می شوند .امکان اضافه کردن اتصال جدید نیست. | اتصالات قبلی شکسته می شوند .امکان اضافه کردن اتصال جدید نیست. |

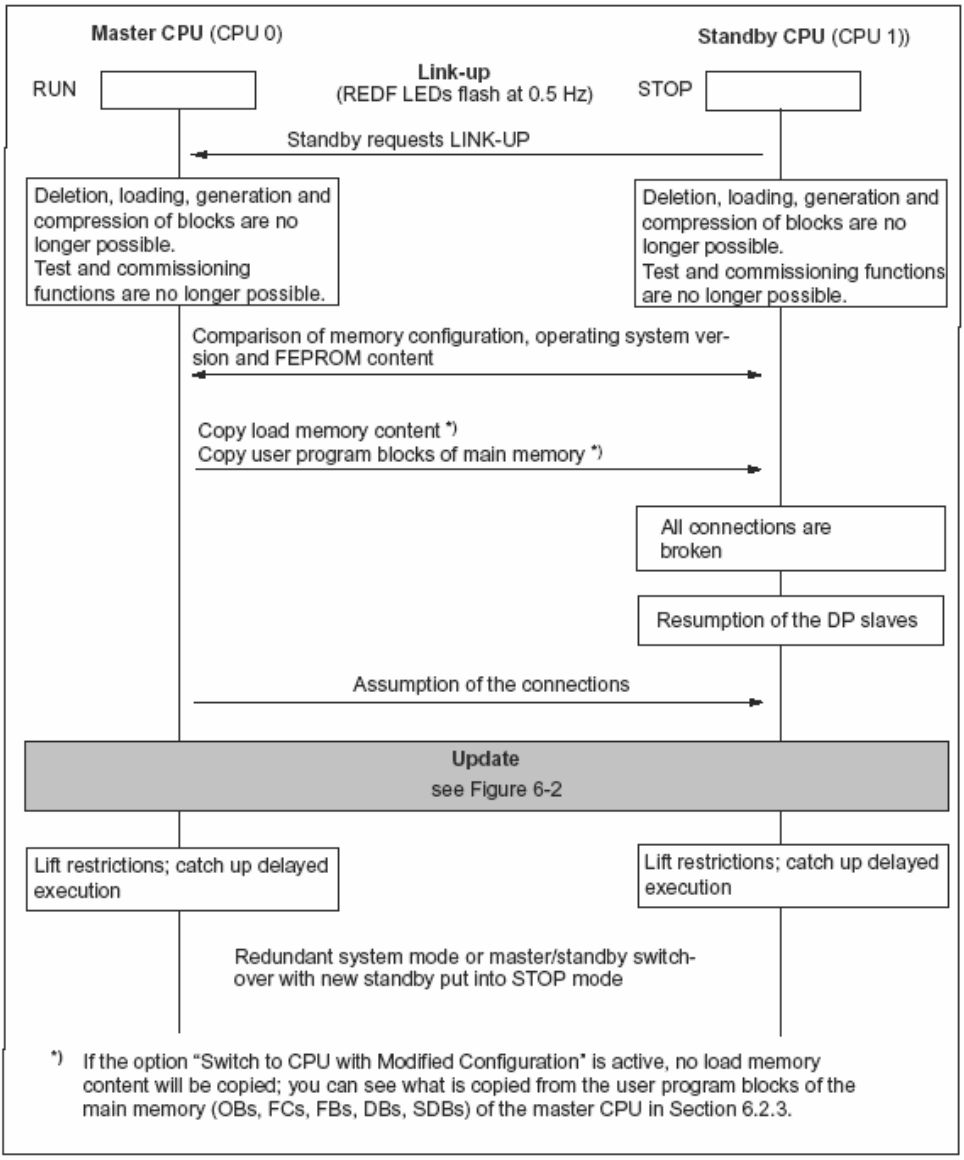



Figure 6-1 Functional sequence of link-up and update

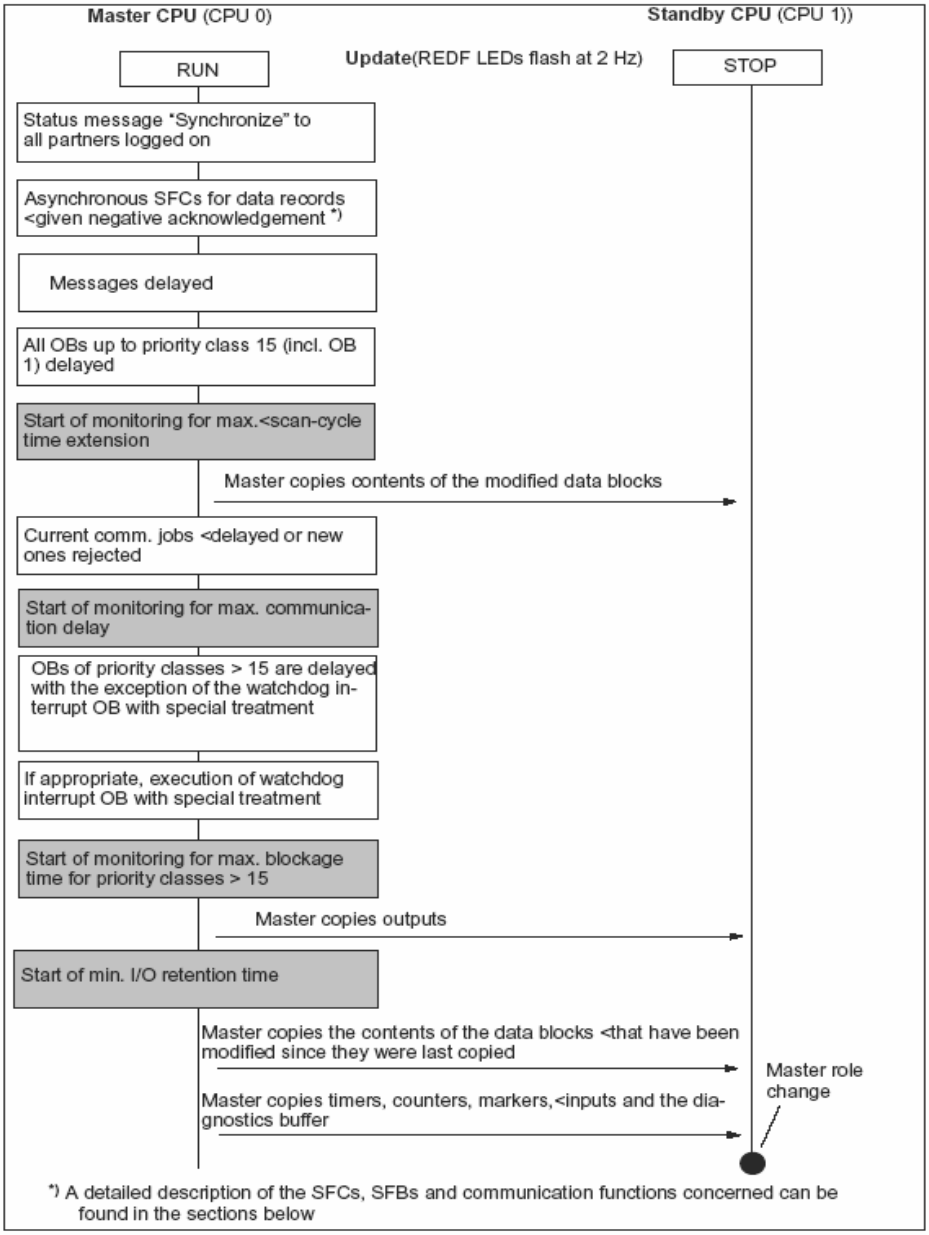
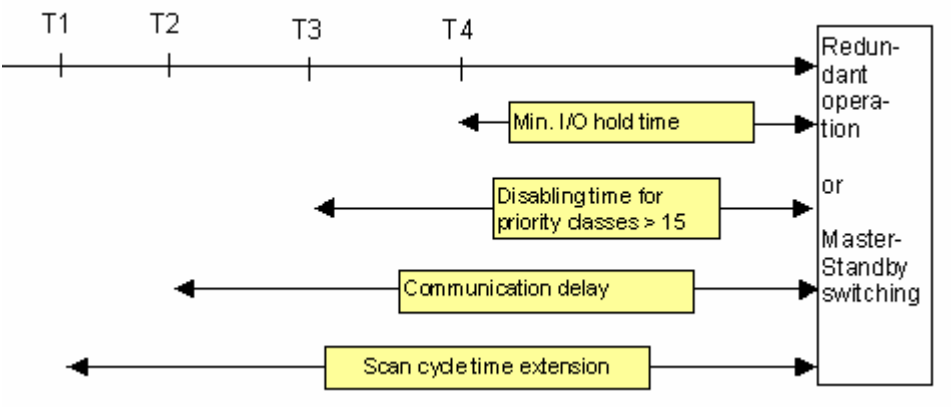



Figure 6-2 Process for update

## حداقل زمان برای سیگنالهای ورودی در طول Update

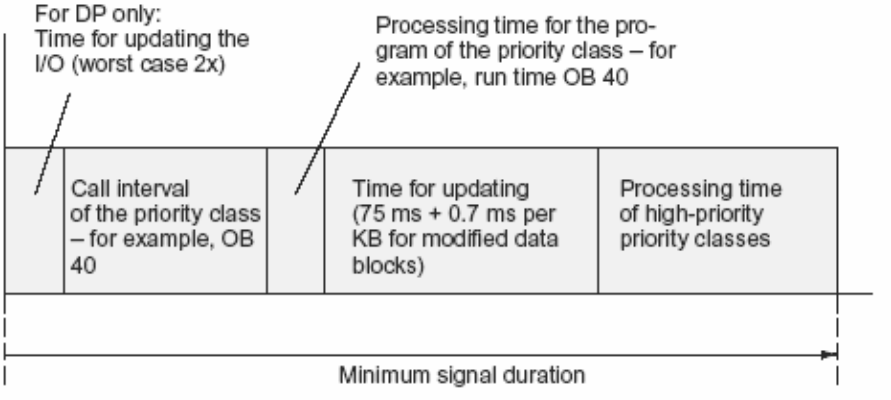
در حین Update ، سیکل اسکن برای مدت مشخصی متوقف می شود.




بنابراین CPU باید تغییراتی را در زمان گرفتن سیگنالهای ورودی اعمال کند ، این حداقل زمان طبق فرمول زیر محاسبه می شود :

Min. signal duration > 2 xx time required for I/O update (for DP only)
+ call interval of the priority class
+ processing time for the program of the priority class
+ time for the update
+ processing time for programs of high-priority classes

در شکل زیر مثالی برای این حداقل زمان برای Priority Class>15 مانند OB 40 نشان داده شده است



**پروسه Link-Up برای رسیدن به مد Redundant**

برای جلوگیری از هر گونه اختلاف بین دو CPU موارد زیر بین دو CPU مقایسه می شود:

۱- یکسان بودن ساختار حافظه

۲- یکسان بودن Version سیستم عامل

۳- یکسان بودن محتویات Load Memory (FEPROM Card)

۴- یکسان بودن محتویات Load Memory (RAM Card) و Integrated SRAM

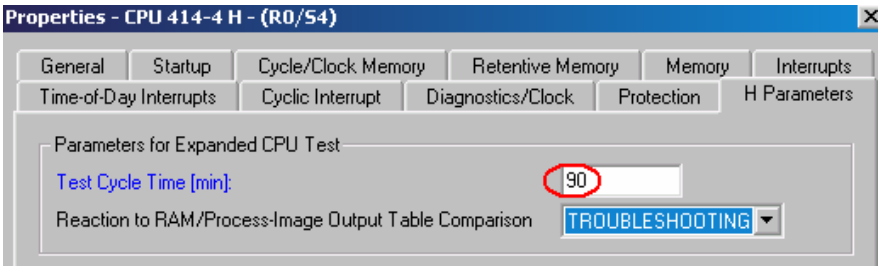
۵- اگر هر کدام از موارد ۱و۲و۳ یکسان نبود Standby CPU به STOP رفته و پیغام خطا ظاهر می شود.

۶- اگر مورد ۴ یکسان نبود Master CPU برنامه کاربری از RAM خودش به RAM مربوطه به Standby CPU کپی می کند . بدیهی است برای مورد ۳ امکان این کار نیست و باید قبل از Link up انجام شده باشد.

**Self Test چیست؟**

وقتی CPU بار اول برقدار می شود یا وقتی در مد Troubleshooting است به اجرای کامل Self Test می پردازد که بین ۹۰ تا ۲۲۰ ثانیه بسته به پیکر بندی سیستم طول می کشد.

در مد RUN سیستم عامل عملیات Self Test را به قسمت های کوچک ( Test Slices ) تقسیم می کند وآنها را تدریجاً اجرا می نماید. حدوداً ۹۰ دقیقه طول می کشد تا در مد RUN این تست کامل شود. زمان فوق در هنگام پیکر بندی سخت افزار توسط Step7 قابل تغییر است . در پارامترهای CPU مانند شکل زیر




اگر Self Test در حین کار اشکالی را تشخیص بدهد رفتار سیستم بصورت زیر خواهد بود:

**الف ) خطای سخت افزاری بدون فراخوانی یکطرفه OB121**

در اینحالت :

- CPU مشکل دار به مد Defective می رود.

- سیستم به حالت Single mode در می آید.
- علت اشکال در Diagnostic Buffer ثبت می شود.

### ب) خطای سخت افزاری با فراخوانی یکطرفهٔ OB121

در اینحالت :

- CPU مشکل دارد به مدTrouble shooting می رود.
- سیستم به حالت Single mode در می آید.
- علت اشکال در Diagnostic Buffer ثبت می شود.
- اگر OB121 دوباره صدا زده شود (تا هفت روز) در اینصورت CPU مشکل دار به مد Defective خواهد رفت و سیستم بصورت Single mode در می آید.

### ج) اشکال مقایسه RAM/PIQ

در اینحالت سیستم از حالت Redundant خارج شده و Standby CPU به مد Trouble shooting می رود .(که در پیکر بندی پیش فرض است) و علت اشکال در Diagnostic Buffer ثبت می شود .

### د) خطاهای Checksum

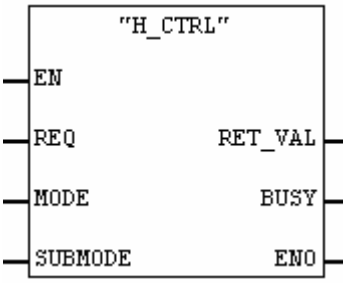
در اینحالت رفتار سیستم مطابق جدول زیر خواهد بود.

| | |
|---|---|
| خطا بعد از وصل تغذیه | CPU مشکل دار به مدDefective رفته و سیستم بصورت Single mode به کار ادامه می دهد. |
| در طولSelf Test سیکلی (در حالت STOP یا Single mode ) | خطا اصلاح می شود .CPU مشکل داردر مد فعلی یعنی STOP یا Single mode باقی می ماند. |
| در طول Self Test سیکلی (در مد Redundant ) | - خطا اصلاح می شود.<br>- CPU مشکل دار به مد Trouble shooting می رود .<br>- سیستم به صورت Single mode در می آید. |
| در مد Trouble shooting | - CPU مشکل دار به مد Defective می رود.<br>- سیستم به صورت Single mode در می آید. |

- در تمام حالات فوق علت اشکال در Diagnostic Buffer ثبت می شود.
- اگر اشکالChecksum مجدداً تا ۷ روز اتفاق بیفتد ، CPU مشکل دار به مد Defective و کل سیستم به مد Single خواهد رفت.

## فعال کردن و غیر فعال کردن مدهای کاری توسط SFC

SFC90 با نام سمبلیک H-CTRL برای کنترل برخی مدهای کاری و نیز تأثیری روی Self Test بکار می رود:




ورودیREQ : یک بیت است که براساس لبهٔ آن ، فانکشن فعال می شود .

ورودی Mode : یک بایت است که توسط آن می توان تغییراتی در برخی مدهای کاری ایجاد کرد.

| Mode | عملکرد |
|---|---|
| 1 | غیرفعال کردنUPDATE |
| 2 | فعال کردنUPDATE |
| 3 | غیرفعال کردنUP– LINK |
| 4 | فعال کردنUP– LINK |

وارد کردن Mode به تنهایی کافی نیست و لازم است همراه با آن Submode نیز که بصورت Word است داده شود طبق جدول زیر :

| Mode | توضیح | Sub mode |
|---|---|---|
| **20** | غیر فعال کردن تست مشخص شده با کد Sub mode | یکی از اعدادمندرج در جدول بعد |
| **21** | فعال کردن تست مشخص شده با کد Sub mode | |
| **22** | اجرای فوری تست مشخص شده با کد Sub mode | |

| Sub mode | Description |
|---|---|
| W#16#0 | SP7-ASIC-Test |
| W#16#1 | Code Memory Test |
| W#16#2 | Data Memory Test |
| W#16#3 | Operating System Code Checksum Test |
| W#16#4 | Code Block Checksum Test |
| W#16#5 | Comparison Of Numbers,Times,Markers and Data block in Redundant Operation |

**تذکر**: در سیستم Redundant اگر تست برای بیش از ۲۴ ساعت غیرفعال باشد ، CPU به مد STOP خواهد رفت یعنی در این سیستم ها حتماً باید سیستم عامل حداقل در طول ۲۴ ساعت یک تست کامل را انجام داده باشد.

### خروجی های SFC90

SFC90 دو خروجی دارد یکی Busy که بصورت بیت است و اگر ۱ باشد نشان دهنده اینست که کار فعال سازی یا غیر فعال سازی هنوز تکمیل نشده است و خروجی Retval یک عدد صحیح است که کد خطای احتمالی مربوط به اجرای فانکشن را نشان می دهد.

## ۱۶-۷ تغییر در سخت افزار سیستم های H در حین کار

اضافه کردن برخی کارتهای سخت افزاری به سیستم های H یا برداشتن آنها از روی سیستم H در حین کار امکان پذیر است اگر چه برای لحظاتی سیستم را از مد Redundant خارج کرده ودر مد Single قرار می دهد.
بطور کلی امکان تغییرات سخت افزاری زیر وجود دارد:

۱- اضافه کردن کارتها به رک های اضافی یا برداشتن آنها از روی رک های مزبور
۲- اضافه کردن یا برداشتن کارتها از روی Remote I/O های شبکه
۳- استفاده از کانالهای خالی کارتها
۴- تغییر برخی پارامترهای CPU
۵- تغییر پارامترهای کارتهای نصب شده

برخی موارد نیز امکان تغییر آنها در حین کار نیست از جمله :

۱- برخی از پارامترهای خاص CPU
۲- سرعت انتقال دیتا روی شبکه DP Master
۳- اتصالات S7 و S7H

### تذکر :

کارتهای IM460 و IM461 مربوط به اتصال رک اضافی و کارت CP443 مربوط به شبکه DP را نمی توان در حین کار وارد یا خارج کرد و اتصال آنها را برقرار نمود این کار وقتی سیستم خاموش وبدون برق است امکان پذیر می باشد.

**الف ) اضافه کردن اجزا به سیستم های H در حین کار**

سخت افزار مناسب (مانند کارتهای I/O مناسب) را می توان در حین کار به رک Central یا روی شبکه DP اضافه کرد. قدم های زیر بترتیب باید برداشته شوند .

**قدم اول** : اضافه کردن سخت افزار جدید به سیستم .در حالیکه سیستم در مد Redundant مشغول کار است می توان سخت افزار جدید مانند موارد زیر را به آنها اضافه کرد:

۱- وارد کردن کارتهای Central به Rack

۲- وارد کردن کارت به DP-station های مدولار

۳- وارد کردن DP-Station جدید روی شبکه DP-Master

هیچ خللی در عملکرد سیستم پیش نخواهد آمد وسیستم کماکان در مد Redundant به کار خود ادامه می دهد . ولی تا اینجا سخت افزار جدید آدرس دهی نشده و در برنامه استفاده نشده است.

**قدم دوم** : پیکر بندی را در مد Offline در برنامه Step7 اصلاح کنید و سخت افزار جدید را به ان معرفی نمایید، ولی به PLC دانلود نکنید.

**قدم سوم** : اطمینان از وجود وعملکرد OB های وقفه

با بررسی OB های 4x , 82 , 85., 86 و 122 مطمئن شوید که سیستم در صورت وقوع وقفه بنحو درست پاسخ خواهد داد . اگر OB ها موجود نیستند آنها را ایجاد ، برنامه نویسی وبه PLC دانلود کنید.

**قدم چهارم** : Stop کردن Standby CPU

توسط برنامه Simatic manager از طریق ورودی PLC> Operation Mode در اینحالت با انتخاب Stand by CPU آنرا Stop کنید.

- Master CPU کماکان در بصورت RUN باقی می ماند ولی کل سیستم از مد Redundant خارج شده در مد Stop قرار می گیرد.
- I/O های یک مسیره متصل به Standby CPU دیگر در اختیار Master CPU نخواهد بود.
- OB72 که برای از بین رفتن CPU-Redundancy طراحی شده صدا زده می شود . ولی OB70 که مربوط به I/O-Redundancy است نمی تواند Call شود.

**قدم پنجم** : دانلود کردن پیکر بندی جدید به CPU–Stand by

سیستم هنوز بصورت Single mode است در اینحالت سخت افزار جدید را به CPU–Standby دانلود می کنیم.

**توجه** : برنامه وپیکر بندی ارتباطات نباید در مد Single به CPU دانلود شود.

**قدم ششم :** در مدار آوردن Standby-CPU وایجاد نقش Master برای آن.

با استفاده از منوی PLC>Operating Mode برنامه Stand by-CPU را RUN می کنیم و With Modified Configuration را انتخاب می نماییم.

در اینحالت :

- Stand by CPU در مد Link-Up سپس Update قرار می گیردو به عنوان Master در می آید.
- Master قبلی به مد Stop سوئیچ می شود.
- سیستم بصورت Single Mode با پیکر بندی جدید شروع به کار می کند.
- I/O های تک مسیره مربوط به Master-CPU قبلی توسط Master جدید قابل آدرس دهی نخواهند بود.
- I/O های تکمسیره مربوط به Master جدید در طول زمان فوق ری ست شده و در اختیار Master-CPU قرار می گیرند.

**قدم هفتم :** رفتن به مد Redundant

از منوی PLC>Operating برنامه ، Stand by-CPU فعلی که لحظاتی قبل Master بوده و هم اکنون STOP است را راه اندازی Warm می کنیم . این CPU در مد Link-Up سپس Update قرار می گیرد ، پس از آن کل سیستم به حالت Redundant در می آید .

**قدم هشتم :** تغییر و دانلود کردن برنامه

تغییرات مورد نظر را در DB , FB , FC , OB اعمال وبه CPU دانلود کنید (Master)

**تذکر مهم :** اگر ساختار FB تغییر کند (IN وOUT و...) یا ارتباطش با DB مربوط عوض شود ، کل سیستم از Redundant به مد Stop سوئیچ می شود (هر دو CPUمتوقف می شوند )

## ب) برداشتن یا حذف کردن اجزا در سیستم های H

**قدم ۱:** حذف کارت مورد نظر در پیکر بندی بصورت Offline .

**قدم ۲:** اعمال تغییرات در برنامه (آنچه مربوط به کارت حذف شده است ) بصورت Offline و سپس ذخیره سازی و دانلود به Master-CPU .

**قدم۳:** Stop کردن Standby-CPU

**قدم٤:** دانلود کردن سخت افزار جدید به Standby-CPU

**قدم۵**: راه اندازی Standby CPU توسط منوی PLC>Operating .این کار منجر به Stop شدن Master-CPU می شود وStand By CPU قبلی به صورت Master در می آید.

**قدم٦**: راه اندازی CPU دیگر ( Master قبلی) توسط منوی PLC>Operating Mode راه اندازی CPU دیگر را انجام می دهیم ، پس از عبور از مراحل Link-Up و Update سیستم به حالت Redundant در می آید.

**قدم۷**: باز کردن کارتها و کابلها و سنسورهای اضافی مطابق آنچه در پیکر بندی معرفی شده است.

## پارامتر های خاص H CPU

پارامترهای زیر در H CPU قابل تغییر توسط Hwconfig نیستند وتوسط ری ست کردن حافظه نیز پاک نمی شوند.

۱- شماره رک مربوطه به H CPU که می تواند ۰ یا ۱ باشد.

این شماره توسط سوئیچ روی SYNC تنظیم می گردد. برای تغییر آن نیز باید به طریق سخت افزاری عمل شود. پس از تغییر CPU را برداشته و دوباره در محل قرار دهید و سپس آن را بصورت دستی ری ست کنید تا شماره به حافظه CPU بار شود.

۲- مد کاری H CPU که Single است یا Redundant .

تغییر مد کاری فوق به روش سخت افزاری به شرح زیر است :

**الف) تغییر از Redundant به Single:**

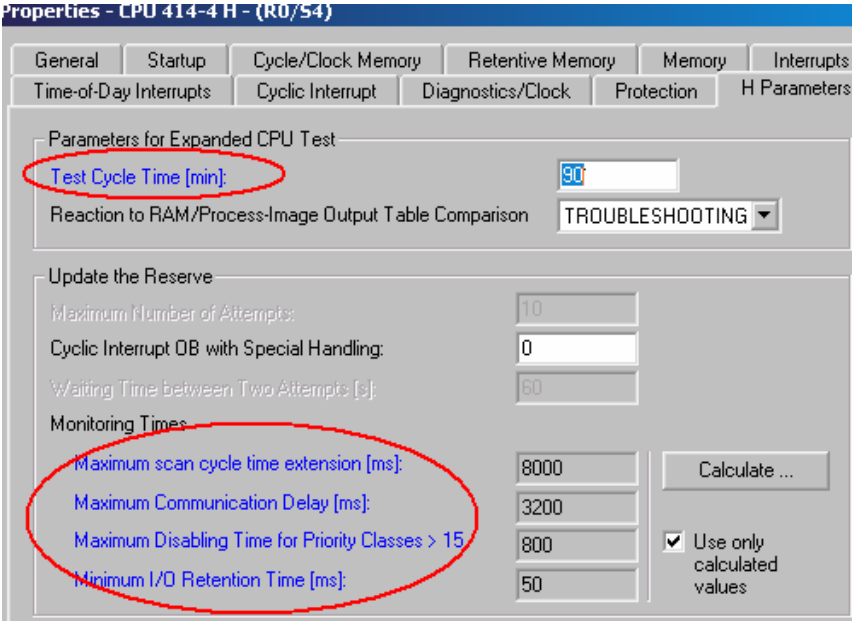
- مدول ارتباطی را بردارید.
- CPU را برداشته ، دوباره در محل خود قرار دهید.
- پروژه ای که بصورت Single Mode پیکربندی شده را به CPU دانلود کنید.

**ب) تغییر از Single به Redundant :**

- در CPU مورد نظر مدول ارتباطی را وصل کنید.
- CPU را برداشته دوباره در محل خود قرار دهید.
- پروژه ای که بصورت Redundant پیکر بندی شده را به CPU دانلود کنید.

## تغییر پارامترهای CPU در حین کار

همانطور که قبلاً نیز ذکر شد در سیستم های H-CPU فقط برخی از پارامترها در حین کار قابل تغییر هستند. این پارامترها با رنگ آبی در صفحه ظاهر می شوند مانند شکل زیر :




اگر بجز پارامترهای قابل تغییر رنگ (آبی رنگ) ، پارامتر دیگری عوض شود این پارامتر بطور اتوماتیک در حین RUN به PLC داده نخواهد شد و event با کد W#16#5966 در بافر Diagnostic ثبت خواهد شد برای رفع آن باید این پارامتر را به وضعیت درست قبلی برگردانید.

لیست پارامترهای قابل تغییر H-CPU در جدول زیر آمده است.

| Tab | Modifiable Parameter |
|---|---|
| Startup | Monitoring time for signaling readiness by modules |
| | Monitoring time for transferring parameters to modules |
| Scan cycle/clock memory | Scan cycle monitoring time |
| | Cycle load due to communications |
| | Size of the process image of inputs |
| | Size of the process image of outputs |
| Memory | Local data (for the different priority classes) |
| | Communication resources: maximum number of communication jobs (you are only allowed to increase this parameter compared to its previously configured |

| | |
|---|---|
| | value.) |
| Time-of-day interrupts (for every time-of-day interruptevery time-of-day interrupt OB)OB) | “Active” check box |
| | “Execution” list box |
| | Starting date |
| | Time |
| Watchdog interrupt (for every watchdog interruptevery watchdog interrupt OB) | Execution |
| | Phase offset |
| Diagnostics/clock | Correction factor |
| Security | Protection level and password |
| Fault-tolerant parameters | Test scan cycle time |
| | maximum scan-cycle time extension |
| | Maximum communication delay |
| | Maximum retention time for priority classes > 15 |
| | minimum I/O retention time |

با توجه به توضیحات فوق برای تغییر پارامترهای مجاز در حین کار قدمهای زیر بترتیب باید برداشته شوند:

## نحوه تغییر پارامترهای مجاز

### قدم اول :

سیستم بصورت Redundant در حال کار است.پارامترهای CPU را بصورت Offline در Hwconfig تغییر می دهیم و کامپایل می کنیم ولی به PLC دانلود نمی کنیم .

### قدم دوم : Stop کردن Standby-CPU

با منوی PLC>Operating Mode در نرم افزار Standby-CPU را Stop می کنیم . در نتیجه سیستم به حالت Single Mode در می آید.

### قدم سوم: دانلود کردن پارامترها به Standby-CPU

**تذکر**: در اینحالت فقط سخت افزار دانلود شود ، برنامه واتصالات نباید دانلود شود.

### قدم چهارم: روشن کردن Stand by-CPU

توسط نرم افزار Standby-CPU را روشن می کنیم . این CPU پس از پشت سر گذاشتن مد Linkup و Update بعنوان Master شروع به کار می کند و Master قبلی Stop می شود .

### قدم پنجم: روشن کردن CPU دیگر(که STOP شده )

پس از Warm Restart دومین CPU سیستم به حالت Redundant در می آید و این CPU بصورت Standby خواهد بود.

## تعویض اجزاء حافظه H-CPU در حین کار

در سیستم های Redundant باید شرایط زیر از نظر حافظه برقرار باشد:

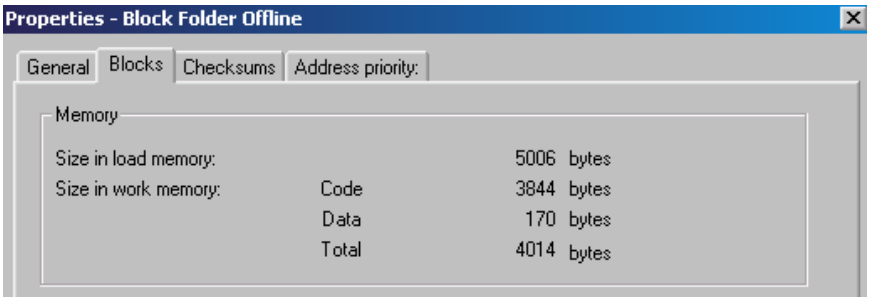
- حافظه اصلی هر دو CPU دارای اندازه یکسان باشد.
- حافظه Load Memory هر دو CPU چه از نوع RAM و چه از نوع FLASH باید هم اندازه باشد.

با وجود این می توان حافظه سیستم H که در حال کار است را افزایش داد یا کارت حافظه را با نوع دیگر عوض کرد ، مثلاً به جای RAM از کارت FLASH استفاده نمود.

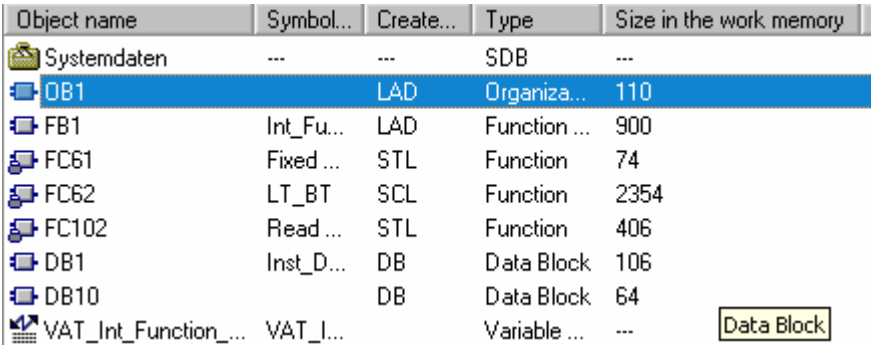
## نحوهٔ افزایش حافظه H-CPU

میزان حافظه ای که بلاک ها در حافظه سیستم اشغال می کنند ممکن است زیاد باشد ، در این شرایط افزایش حافظه سیستم ضرورت پیدا می کند.

قبل از هر چیز میزان حافظه مورد نیاز برای بلاک ها را می توان با کلیک راست روی پوشه Blocks و انتخاب Object properties مانند شکل زیر مشاهده کرد.




برای مشاهده اینکه هر بلاک چه حجم حافظه ای را اشغال می کند می توان اطلاعات دقیقتر را با استفاده از منوی View>Details مشاهده کرد ، شکل زیر.



| Object name | Symbol... | Create... | Type | Size in the work memory |
|---|---|---|---|---|
| Systemdaten | --- | --- | SDB | --- |
| OB1 | | LAD | Organiza... | 110 |
| FB1 | Int_Fu... | LAD | Function ... | 900 |
| FC61 | Fixed ... | STL | Function | 74 |
| FC62 | LT_BT | SCL | Function | 2354 |
| FC102 | Read ... | STL | Function | 406 |
| DB1 | Inst_D... | DB | Data Block | 106 |
| DB10 | | DB | Data Block | 64 |
| VAT_Int_Function_... | VAT_I... | | Variable ... | --- |

**نکته ۱ :**

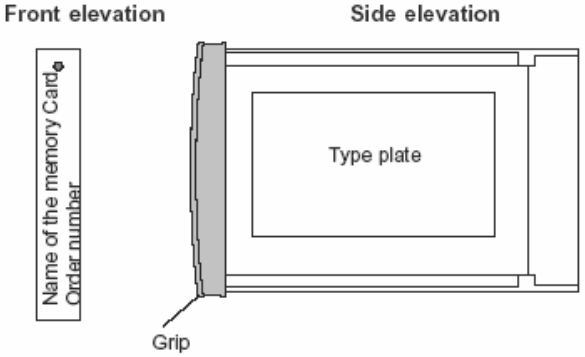
VAT و VDT از آنجا که به CPU دانلود نمی شوند به آنها سایز اختصاص داده نشده است.

**نکته ۲:**

دیتا بلاک هایی که در پوشه Offline موجود نیستند و توسط SFB در حافظه سیستم ایجاد می شود را فقط در مد Online می توان مشاهده کرد.

## افزایش Load Memory با استفاده از کارت حافظه Memory Card

کارت حافظه یا MC برای افزایش حافظه CPU استفاده می شود و در اسلات روی CPU قرار می گیرد . Load Memory سیستم در این حالت مجموع حافظه سیستم و حافظه کارت خواهد بود.




روی کارت حافظه می توان بلاک های OB و FB و FC و DB و دیتاهای سیستم را ذخیره کرد.

دو نوع کارت حافظه وجود دارد:

۱) RAM
۲) Flash یا (FEPROM)

## کارت RAM

از این کارت می توان برای ذخیره سازی دیتا و برنامه استفاده کرد . کارت را داخل اسلات قرار داده سپس در حالی که CPU در مد Stop یا RUN/P است از طریق برنامه Step7 بلاک ها را در آن ذخیره می کنیم.

توجه شود که:

۱)اگر کارت RAM از اسلات CPU بیرون آورده شود اطلاعات روی آن همگی پاک می شود.

۲) اگر تغذیه CPU قطع شود و تغذیه Back up نیز وجود نداشته باشد کارت RAM پاک می شود.

## کارت Flash

از این کارت نیز می توان شبیه کارت RAM برای ذخیره سازی دیتا و برنامه استفاده کرد ، ولی فرق آن با RAM این است که اگر کارت بیرون آورده شود یا تغذیه قطع شود ، پاک نمی شود و به اصطلاح Fail Proof می باشد.

کارت Flash را می توان روی PG برنامه ریزی کرد. وهمین طور می توان روی خود CPU آن را Program نمود.

برای Load کردن برنامه به کارت Flash به یکی از دو روش زیر می توان اقدام نمود :

۱- سوئیچ CPU را در مد Stop قرار داده ، کارت Flash را وارد می کنیم . سپس از طریق Step 7 برنامه را به CPU دانلود می نمائیم .

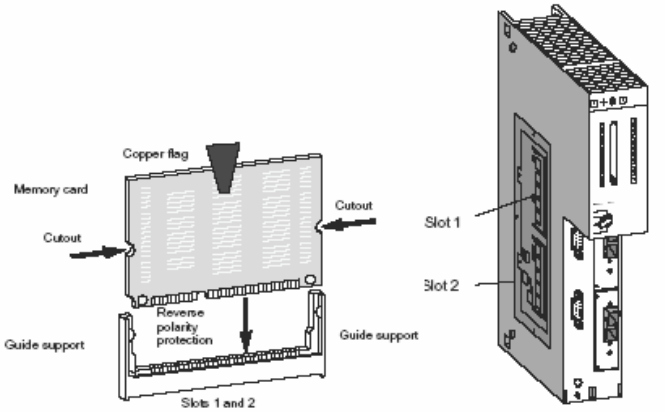
۲- کارت Flash را در حالت Offline روی اسلات مربوطه در PG قرار داده و برنامه را به آن Load می کنیم سپس کارت را در اسلات CPU قرار می دهیم.

**تذکر**: برای تعویض کارت های حافظه بهتر است همیشه ابتدا CPU را STOP کرده سپس کارت را بیرون بیاوریم.

## افزایش حافظه Work Memory در CPU 417-4H

نکات قابل توجه :

- فقط در CPU 417-4H امکان افزایش Work Memory وجود دارد.
- این کار با قرار دادن کارتهای خاص در کنار CPU در اسلات ۱ یا ۲ انجام میشود.



- اگر فقط یک کارت حافظه نیاز باشد باید آن را در اسلات ۱ قرار دهیم .
- کارت دوم در صورتی می تواند اضافه شود که قبلاً در اسلات ۱ ، کارت ۴ مگا بایتی قرار داده باشیم ، بنابراین ترکیب استفاده از کارت ها مطابق جدول زیر خواهد بود:

| Combination | Slot 1 | Slot 2 |
|---|---|---|
| 1 | 2 Mbytes | – |
| 2 | 4 Mbytes | – |
| 3 | 4 Mbytes | 2 Mbytes |
| 4 | 4 Mbytes | 4 Mbytes |

**تذکر** : در موقع قرار دادن کارت های فوق باید به تخلیه الکترو استاتیکی یا ESD که ممکن است منجر به آسیب رساندن به کارت یا CPU شود ، توجه داشته باشید (عدم تماس دست با قسمتهای حساس)

## نحوه افزایش حافظه

افزایش حافظه CPU معمولاً با کارت RAM معنا می دهد ، اگر چه با کارت FLASH نیز می توان Load Memory را افزایش داد . ولی در این حالت این وظیفه کاربر است که اطلاعات برنامه وسخت افزار را روی کارت FLASH ذخیره نماید.

وقتی سیستم در مد Redundant مشغول کار است قدمهای زیر برای افزایش حافظه باید برداشته شود.

### قدم۱: Stop کردن Standby-CPU

سیستم به حالت Single Mode می رود.

### قدم ۲ : افزایش دادن حافظه اصلی

- خاموش کردن منبع تغذیه Standby-CPU
- بیرون کشیدن CPU فوق از RACK
- نصب کارت حافظه جدید یا تعویض آن
- گذاشتن CPU در رک
- اطمینان از وصل بودن مدولهای SYNC
- اطمینان از اینکه سوئیچ روی CPU در مد RUN یا RUN-P است .
- وصل تغذیه CPU

**قدم ۳ : افزایش حافظهٔ Load Memory**

در اینحالت به جای مراحل قدم دوم صرفاً کافی است که کارت قبلی را از Standby CPU بیرون آورده کارت جدید با حجم بیشتر ولی از همان نوع را وارد می کنیم. در اینحالت CPU با چشمک زدن چراغ STOP درخواست Reset می کند .

**قدم ٤ :** ری ست کردن Standby-CPU توسط نرم افزار.

**قدم ٥ : روشن کردن Standby-CPU توسط نرم افزار**

- Standby-CPU مراحل Linkup و Update را پشت سر گذاشته و به عنوان Master وارد عمل می شود.
- CPU دیگر STOP می شود.
- سیستم بصورت Single Mode در می آید.

**قدم ٦ :** قطع کردن تغذیه CPU دوم

**قدم ۷ :** افزایش حافظهٔ اصلی یا Load Memory این CPU به همان روشی که قدمهای ۲ تا ۴ ذکر شد.

**قدم ۸ :** روشن کردن CPU دوم :

این CPU مراحل Linkup و Update را انجام داده ولی Master نمی شود . با حالت Standby سیستم بصورت Redundant در می آید.

**نحوهٔ تعویض نوع حافظه Load Memory**

همانطور که می دانید دو نوع کارت حافظه داریم :

RAM که بیشتر برای دوران تست وراه اندازی بکار می رود.

FLASH که برای ذخیره سازی دائمی برنامهٔ نهایی بکار می رود.

با فرض اینکه سیستم در مد Redundant است گامهای زیر باید برداشته شود.

**گام ۱** : Stop کردن Standby-CPU از طریق نرم افزار

**گام ۲** : بیرون آوردن کارت فعلی و قرار دادن کارت جدید ، در اینحالت CPU درخواست ری ست می کند.

**گام ۳**: ری ست کردن Standby-CPU از طریق نرم افزار.

**گام ٤** : دانلود کردن برنامه و پیکر بندی سخت افزار به Standby-CPU

**گام ٥** : روشن کردن Standby-CPU از طریق نرم افزار.

در اینحالت این CPU مراحل Linkup و Update را انجام داده و Master می شود. CPU دیگر STOP می شود.سیستم به حالت Single Mode در می آید.

**گام ٦** : تعویض کارت حافظهٔ CPU دوم شبیه گام ۲

**گام ۷ :** دانلود کردن برنامه و پیکربندی سخت افزار به CPU دوم .

**گام ۸ :** روشن کردن CPU دوم از طریق نرم افزار .

در اینحالت این CPU مراحل Linkup وUpdate را انجام داده ولی Master نمی شود ، سیستم به مد Redundant در می آید.

**تذکر** :

- برای تعویض کارت RAM تمام مراحل فوق لازم است انجام شود.
- برای تعویض کارت FLASH می توان برنامه و سخت افزار را بیرون از PLC به آنLoad کرد ، بنابراین گامهای ۴ تا ۷ را می توان حذف کرد.

## نحوهٔ نوشتن روی کارت FLASH در سیستم های H :

در سیستم های H می توان روی کارت FLASH در مد RUN نوشت بدون اینکه نیاز به STOP کردن باشد ، برای این کار پیش نیاز آنست که دیتاهای Online (سخت افزار و برنامهٔ CPU ) دقیقاً با دیتاهای Offline روی PG/PC یکی باشند. مراحل :

۱) Standby-CPU را Stop کرده و کارت FLASH را در CPU قرار دهید .

۲) از طریق S7 حافظهٔ Standby-CPU را ری ست کنید.

۳) پیکر بندی سخت افزار را به Standby-CPU دانلود کنید .

۴) از طریق منوی زیر برنامه را به کارت FLASH دانلود کنید .(دقت کنید شمارهٔ CPU را در پنجره درست انتخاب کنید )Simatic Manager PLC>Download User Program To Memory Card

۵) Standby-CPU را از طریق منوی Operating Mode روشن کنید .این CPU به حالت Master در می آید و دیگری Stop می شود .

۶) کارت Flash مشابه را در CPU دیگر قرار داده و توسط نرم افزار آن CPU را ری ست کنید.

۷) قدم ۴ را برای این CPU و این کارت نیز تکرار کنید.

۸) این CPU را Warm Restart کنید ، بصورت Standby شروع به کار کرده و کل سیستم به حالت Redundant بر می گردد.

### ۸-۱۶ MTBF در سیستم های H

MTBF که مخفف Mean Time Between Failure یا زمان میانگین بین خطاهاست ، شاخصی است که بر اساس آن قابلیت اطمینان سیستم تعیین می گردد در مدولها و کارتهای S7 زیمنس مدعی است که :

- MTBF برای CPU ها = ۱۵ سال
- MTBF برای کارتهای I/O = ۵۰ سال

می باشد . بدیهی است وقتی اجزاء مختلفی در کنار هم قرار می گیرند مقدار MTBF سیستم از نو باید محاسبه شود . وقتی Standby-CPU را Stop کرده و کارت FLASH را در CPU قرار دهید از دو سیستم به عنوان سیستم Redundant استفاده می شود MTBF سیستم جدید از فرمول زیر محاسبه خواهد شد .

$$MTBF_{1v2} = \frac{MTBF_{1v1}^{2}}{2MDT + 2(1 - dc)\cdot MTBF_{1v1}}$$

که در آن :

$MTBF_{1V2}$ : MTBF سیستم جدید

$MTBF_{1V2}$ : MTBF سیستم قبلی

MDT : مخفف Mean Down Time یا زمان توقف میانگین است که از جمع دو آیتم زیر بدست می آید.

زمان شناسایی عیب +زمان تعمیر ورفع عیب = MDT

در S7-400H :زمان شناسایی عیب حدود نصف زمان سیکل تست ( که بطور پیش فرض ۹۰ دقیقه است ) می باشد یعنی ۴۵ دقیقه.زمان تعمیر معمولاً ۴ ساعت است.

dc : مخفف Diagnostic Coverage است و حدوداً ۹۵٪ است.

**: Availibity**

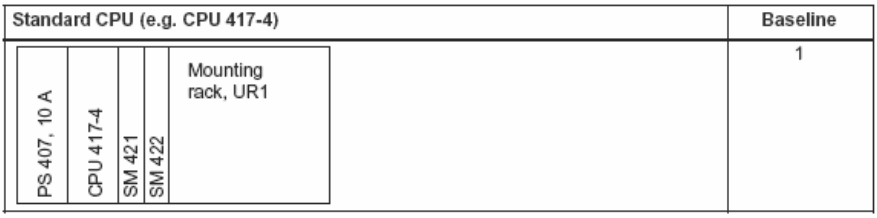
یا میزان دسترسی که تجهیز اماده بکار یا در حال کار است از رابطهٔ زیر بدست می آید و پارامتر دیگری برای ارزیابی سیستم است .

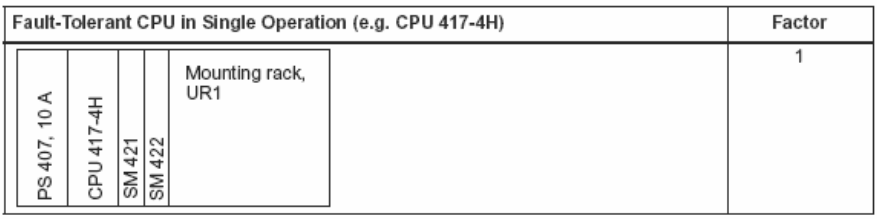
$$V = \frac{MTBF_{1v2}}{MTBF_{1v2} + MDT} 100\%$$

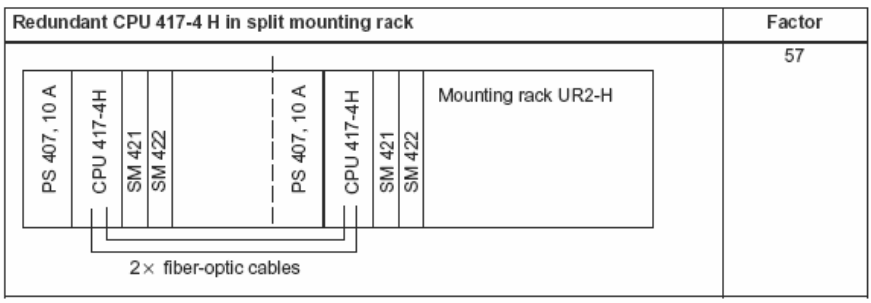
با توجه به نکات فوق الذکر بدون پرداختن به محاسبات ریاضی صرفاً میزان دسترسی سیستم های S7-400H با پیکر بندی های مختلف را با یک پیکر بندی پایه مقایسه می کنیم.

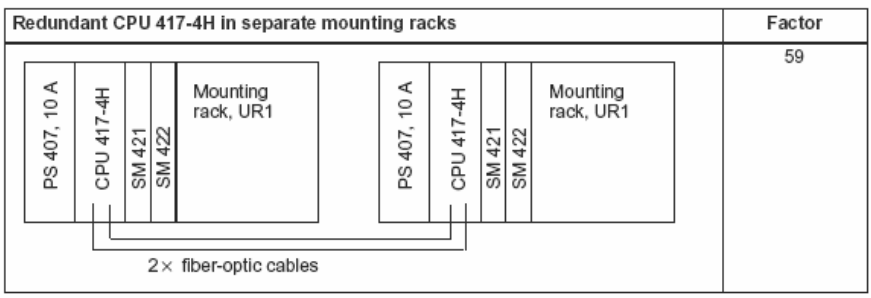
## مقایسه سیستم H با سیستم های معمول

در شکل زیر پیکر بندی یک سیستم S7-417 بعنوان پایه با ضریب ۱ در نظر گرفته شده و سایر پارامترها نسبت به آن سنجیده شده اند.



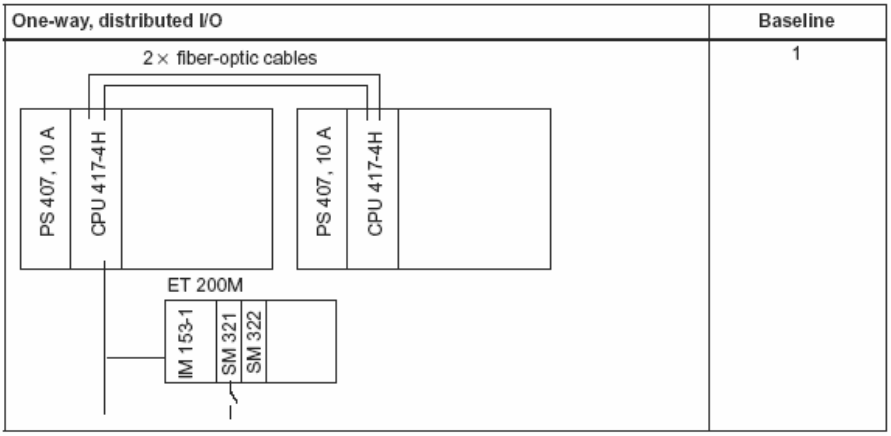



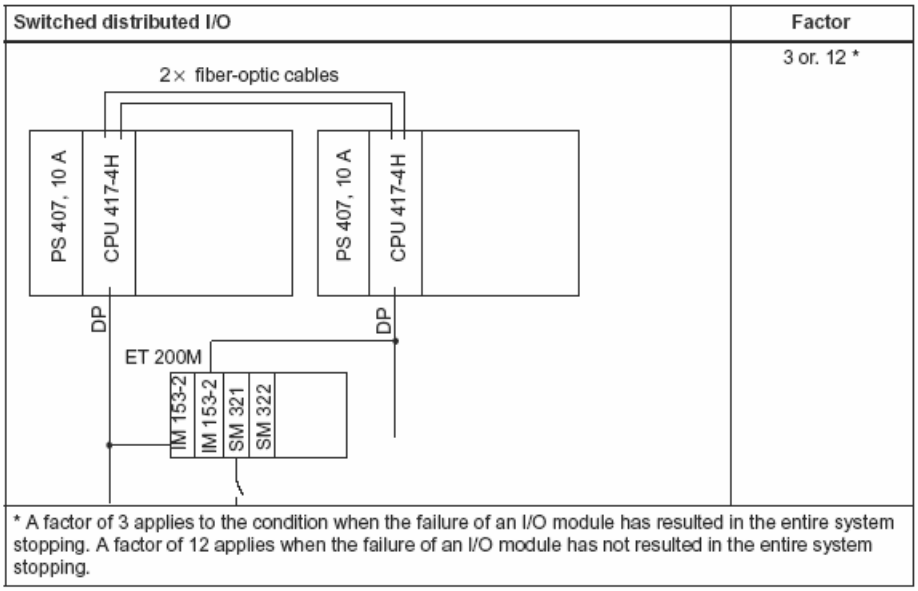



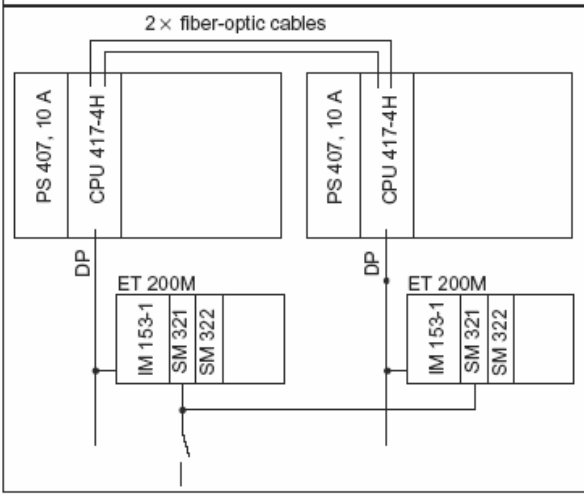



**مقایسه سیستم های H با I/O Redundancy**

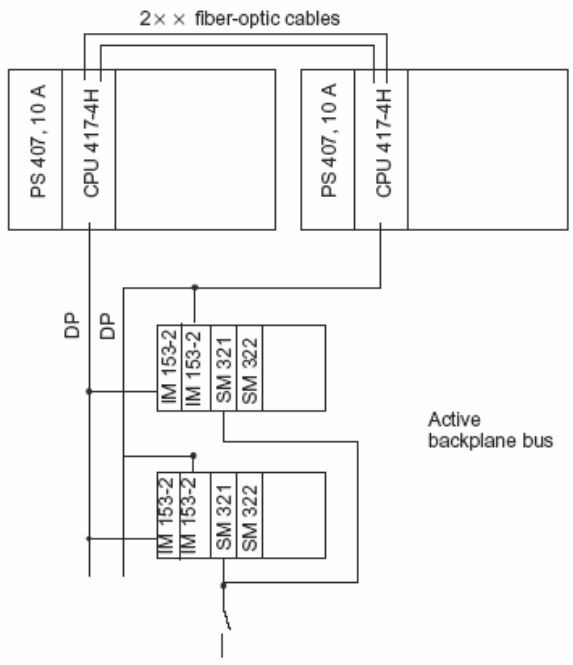
شکل زیر با I/O تک مسیره بعنوان پایه در نظر گرفته و سایر ساختارها نسبت به آن سنجیده می شود.



| Single–channel, one–way I/O | MTBF factor |
|---|---|
|  | 65 |



| Single–channel switch I/O | MTBF factor |
|---|---|
|  | 70 |

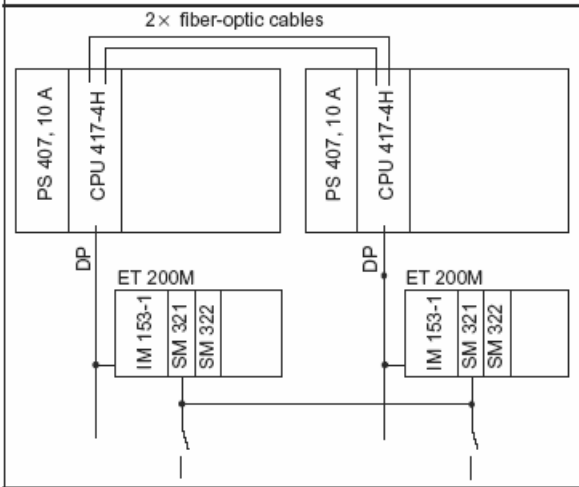
| Redundant I/O | MTBF factor |
|---|---|
|  | see table below |

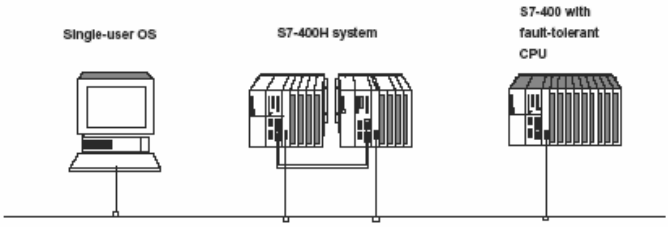


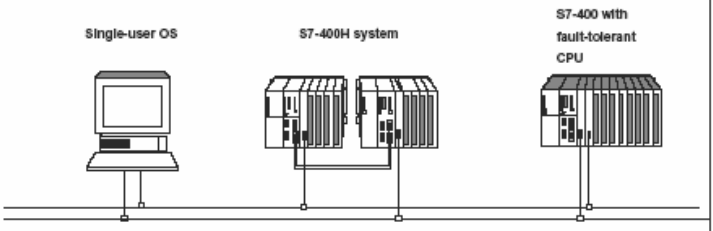
Table A-1 MTBF factor for redundant I/O

| Modules | MLFB | MTBF factor |
|---|---|---|
| **Digital input module, distributed** | | |
| DI 24xDC24V | 6ES7 326-1BK00-0AB0 | 500 |
| DI 8xNAMUR [EEx ib] | 6ES7 326-1RF00-0AB0 | 500 |
| DI16xDC24V, interrupt | 6ES7 321-7BH00-0AB0 | 20 |
| **Analog input module, distributed** | | |
| AI 6x13Bit | 6ES7 336-1HE00-0AB0 | 500 |
| AI8x12Bit | 6ES7 331-7KF02-0AB0 | 25 |
| **Digital output module, distributed** | | |
| DO 10xDC24V/2A | 6ES7 326-2BF00-0AB0 | 500 |
| DO8xDC24V/2A | 6ES7 322-1BF01-0AA0 | 3 |
| DO32xDC24V/0.5A | 6ES7 322-1BL00-0AA0 | 3 |

## مقایسه سیستم های H مجهز به شبکه Redundant

این مقایسه در شکل زیر نشان داده شده است .

| Standard communication | Baseline |
|---|---|
|   | 1 |

| Fault-tolerant communication | Factor |
|---|---|
|   | 83 |

# فصل هفدهم - سیستم های ایمن و مقاوم در برابر خطا
# Fail Safe & Fault Tolerant Systems

**مشتمل بر :**


**۱۷-۱ مقدمه**

**۱۷-۲ کاربرد سیستم های F و FH**

**۱۷-۳ انواع ترکیب بندی**

**۱۷-٤ اجزای یک سیستم S7-400F**

**۱۷-۵ کار با سیستم های F**

**۱۷-٦ پیکربندی سیستم F**

**۱۷-۷ راه اندازی و نظارت بر خطاها در سیستم F**

**۱۷-۸ پیکر بندی سیستم FH**

**۱۷-۹ راه اندازی و نظارت بر خطاها در سیستم FH**

**۱۷-۱۰ مکانیزم های ایمنی**

**۱۷-۱۱ واکنش در برابر خطا**

**۱۷-۱۲ مدهای کاری یک سیستم S7-400F/FH**

**۱۷-۱۳ نظارت منطقی یا زمانی بر اجرای برنامه**

**۱۷-۱٤ ارتباطات ایمنی**

**۱۷-۱۵ نکات پیکربندی سخت افزار و تنظیم پارامترها**

### ۱۷-۱ مقدمه

همگام با افزایش سطح اتوماسیون در صنایع و فرآیندها، قابلیت های مورد انتظار از سیستم ها کنترل نیز افزایش یافته است .ضرورت کاربرد نوعی از سیستم های کنترل که در برابر خطاهاو وضعیت های ناخواسته مقاوم بوده و کنترل فرآیند را ادامه داده و یا آن را به نحو مناسب و پیش بینی شده ای متوقف می کنند ازآنجا نشات می گیرد که بروز این خطاها، خطرات و هزینه های وسیعی را در پی دارد و گاه در برخی فرآیند ها (مانند فرآیندهای شیمیایی که در یک پالایشگاه باید به طور متوالی و بدون توقف انجام شود) به هیچ عنوان قابل قبول نیست. بنابراین توقع اصلی و مهم چنین فرآیند هایی دسترس پذیری بالای سیستم کنترل(High Availability ) به هنگام وقوع خطا می باشد و در این راستا هزینه بالای این سیستم ها در مقایسه با خسارت های توقف فرآیند به مراتب کمتر و کاملا توجیه پذیر است.

دسترس پذیری بالای سیستم کنترل همیشه مترادف با "افزونگی" می باشد که سطح آن از افزونگی فقط CPU تا افزونگی تمام اجزای سیستم کنترل متغیر است و انتخاب سطح مناسب آن به حساسیت و نیاز فرآیند بستگی دارد.

در حالت کلی اجزای افزونه – مثلا دو CPU افزونه- وظیفه یکسانی را موازی با یکدیگر انجام می دهند یعنی هر دو در حالت Hot Standby قرار دارند تا هرقت که یکی از آنها دچار مشکل شد دیگری کنترل را ادمه داده و هیچ توقفی در فرآیند رخ ندهد. در این حالت پس از جایگزینی مدول معیوب دوباره افزونگی از سر گرفته می شود.

ایمنی افراد، ماشین آلات و محیط زیست در برابر خطراتی که به دلیل کاربرد اشتباه و یا خرابی تجهیزات ممکن است رخ دهد موضوع کاملا متفاوتی است. چنین خطراتی مانند نشت مواد سمی، اختلال در کارکرد دستگاهها و ازبین رفتن مواد اولیه فرآیند باید پیش بینی و توسط سیستم های کنترل ایمن در برابر خطا از بروز آنها جلوگیری شود. در حالت کلی گسترش سطح اتوماسیون و افزودن مکانیزم ها و توابعی که مرور و نظارت دائم نقاط حساس فرآیند را انجام می دهند گامی در جهت برقراری ایمنی بیشتر در فرآیند می باشد.

در مجموع این مسئله که چه میزان **دسترس پذیری(Availability)** و چه میزان **ایمنی (Fail Safe)** ازیک سیستم کنترل مورد انتظار است به نوع کاربرد و فرآیند بستگی دارد.

### ۲-۱۷ کاربرد سیستم های سری F و FH

این سیستم ها برای کنترل فرآیندهایی به کار می روند که در آنها **ایمنی** از اهمیت بسیار بالایی برخوردار است. در چنین فرآیندهایی به هنگام وقوع خطا یا توقف عملکرد ، سیستم کنترل باید فرآیند را به یک حالت ایمن از پیش تعریف شده ببرد تا خسارتی متوجه افراد، محیط زیست، دستگاهها و فرآیند نگردد.

### سطح ایمنی

سیستم های سری F و FH استانداردهای ایمنی را مطابق زیر تحت پوشش قرار می دهند :

- سطوح AK1 تاAK6 ازاستاندارد DIN V 19250/ DIN V VDE 0801
- سطوح SIL1 تا SIL3 از استاندارد IEC 61508
- رده های ۱ تا ۴ از استاندارد EN 954-1

### پایه و اساس توابع ایمنی

ایمنی در برابر خطا اصولا به کمک توابع موجود در نرم افزار حاصل می شود. در واقع این توابع اجرا می شوند تا در موقع بروز شرایط خطرناک ، فرآیند در یک حالت امن و مطمئن قرار گیرد .

تابع ایمنی مورد نیاز فرآیند از دو راه اجرا می شود :

- تابع ایمنی که توسط کاربر تهیه شده است
- تابع واکنش در برابر خطا

در واقع هنگامی که یک سیستم F به هنگام بروز خطا نتواند تابع ایمنی اصلی خود را که توسط کاربر تهیه شده اجرا کند، تابع وکنش در برابر خطا اجرا می شود و مثلا خروجی های مربوطه به حالت Off می روند و در صورت نیاز CPU در حالت Stop قرار می گیرد.

**مثال :** سیستم کنترل سری F می بایست در یک فرآیند خاص هنگامی که فشار از حدی بیشتر شود یک شیر را باز کند. این در واقع تابع ایمنی است که توسط کاربر نوشته شده و بنابراین هنگامی که این حالت پیش می آید و سیستم کنترل هم دچار عیب نشده است فقط شیر مورد نظر باز می شود. اما اگر نقصی در عملکرد CPU رخ دهد تمام خروجی ها Off می شوند بنابراین این شیر باز شده و بقیه محرک ها (Actuator) که در فرآیند هستند به یک حالت امن و مطمئن می روند. در واقع حالت دوم واکنش خود سیستم کنترل – مستقل از برنامه کاربر- به اختلال در عملکرد است.

توابع ایمنی اصولا در اجزای زیر تعبیه می شوند :

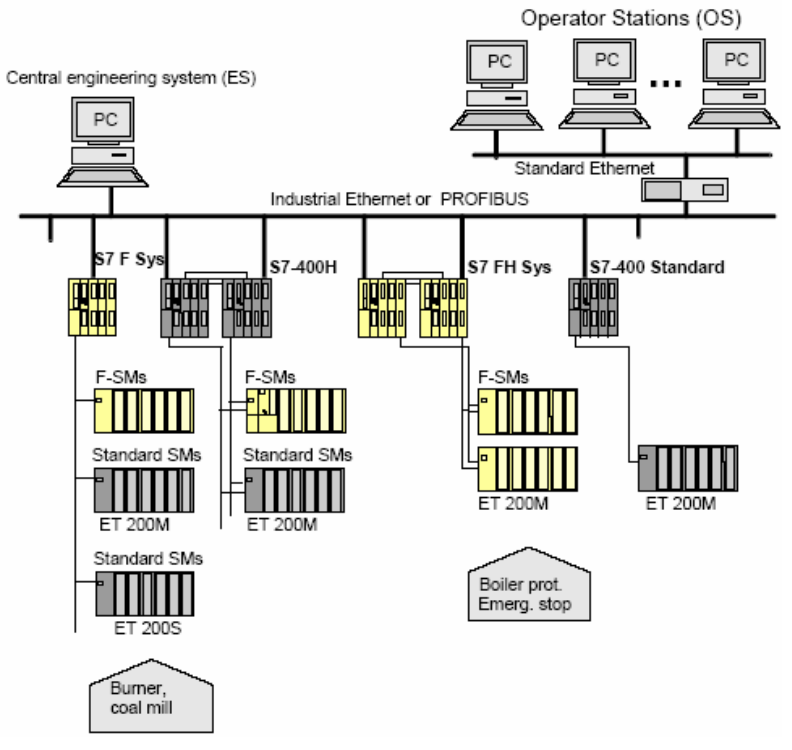
- برنامه کاربر که با لحاظ کردن شرایط ایمنی تهیه شده و در CPU اجرا می شود
- واحدهای ورودی / خروجی ایمن در بربر خطا

## ایمنی و دسترس پذیری

به منظور افزایش دسترس پذیری سیستم اتوماسیون و جلوگیری از زیان های ناشی از توقف عملکرد، می توان سیستم های ایمن در برابر خطا را به صورت مقاوم در برابر خطا نیز ترکیب بندی نمود که البته این میزان دسترس پذیری به کمک افزونگی اجزای سیستم کنترل حاصل می شود.

در نتیجه یک سیستم کنترل ایمن و مقاوم در برابر خطا مانند S7-400FH اجازه می دهد که فرآیند بدون هیچ مشکل و زیانی برای افراد و محیط به کار خود ادامه دهد.

شکل زیر کاربرد S7-400F و S7-400FH در اتوماسیون فرآیند را نشان می دهد.



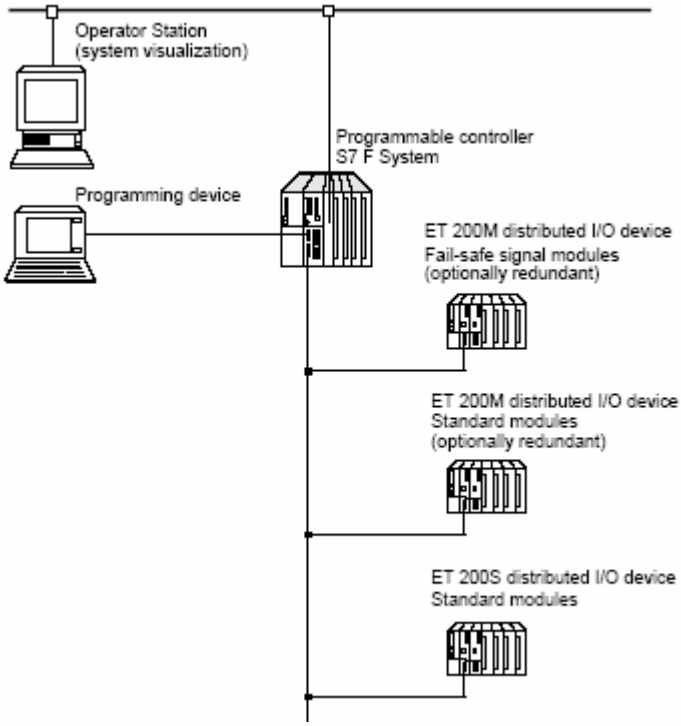
## ۱۷-۳ انواع ترکیب بندی

### ۱- ترکیب بندی سیستم S7-400F

این سیستم یک سیستم اتوماسیون ایمن در برابر خطا است که حداقل از اجزای زیر تشکیل شده است :

- یک CPU با قابلیت ایمنی در برابر خطا (یا به اختصار F) مانند CPU 417-4H که بتواند برنامه کاربر از نوع F را اجرا کند
- مدول های ورودی/خروجی نوع F در یک مجموعه ورودی/خروجی های توزیع شده ET 200M که البته افزونگی هم در صورت نیاز می تواند پیاده سازی شود

شکل زیر اجزای نرم افزاری و سخت افزاری یک سیستم F را نشان می دهد. این ترکیب بندی را می توان با استفاده از مدول های استاندارد S7-300 و S7-400 گسترش داد.

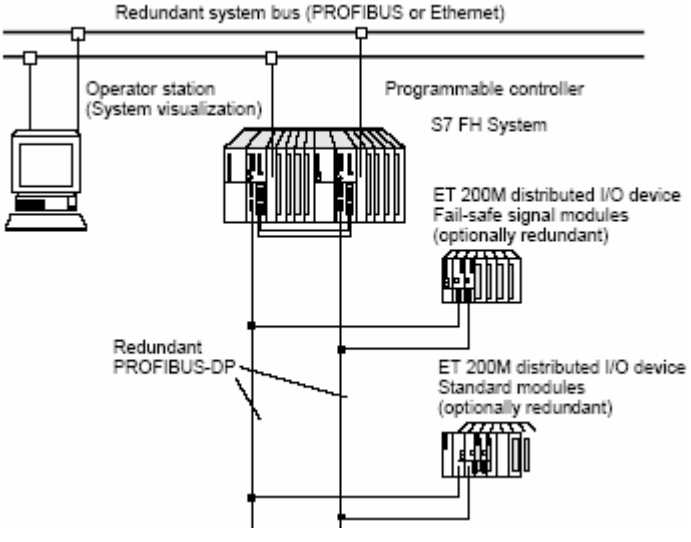


### ۲- ترکیب بندی سیستم S7-400FH

این سیستم یک سیستم اتوماسیون ایمن و مقاوم در برابر خطا است که حداقل از اجزای زیر تشکیل شده است :

- یک سیستم مقاوم در برابر خطا مدل S7-400H (هر دو واحد اصلی و پشتیبان)که یک برنامه کاربر نوع F را اجرا می کند
- مدول های ورودی/خروجی نوع F در یک مجموعه ورودی/خروجی های توزیع شده ET 200M به عنوان ورودی/خروجی های سوئیچ شده که البته افزونگی هم در صورت نیاز می تواند پیاده سازی شود

شکل زیر مثالی از ترکیب بندی S7-400FH با CPU افزونه را نشان می دهد که در آن مدول های ورودی/خروجی توزیع شده و سوئیچ شده که بین CPU های افزونه به اشتراک گذاشته شده اند از طریق یک Bus افزونه به سیستم متصل شده اند.




### ترکیب اجزای استاندارد، مقاوم(H) و ایمن(F) در برابر خطا

این سه نوع سیستم می توانند مطابق زیر با هم ترکیب شوند:

- سیستم های استاندارد، نوع H، نوع F و نوع FH می توانند در یک سیستم کنترل و در کنار هم به کار روند.

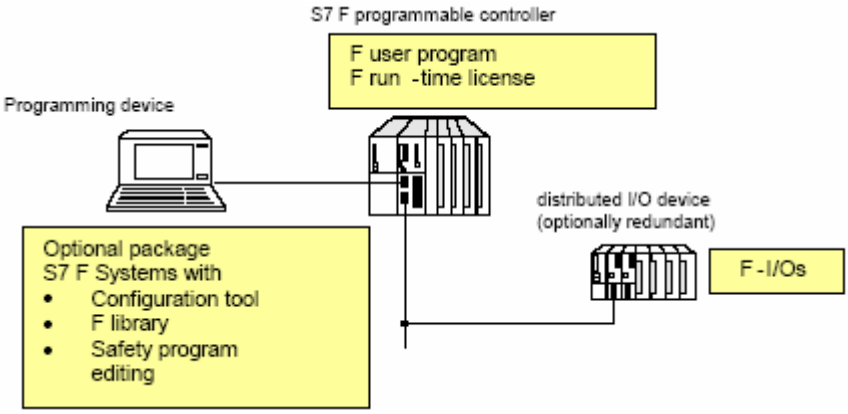
- مدول های استاندارد و نوع F می توانند در یک سیستم اتوماسیون در کنار هم به کار روند. اما در حالتی که سیستم اتوماسیون از نوع ایمن در برابر خطا است این مدول ها باید در ورودی/خروجی های ET 200M جدا از هم به کار برده شوند.
- یک برنامه نوع F می تواند همراه با برنامه های عادی و استاندارد روی سیستم های نوع F یا سیستم های نوع FH اجرا گردد.

در مجموع می توان گفت ترکیب این سه نوع سیستم مزایای زیر را در پی دارد :

- می توان یک سیستم اتوماسیون مرکب پیاده سازی کرد که در آن بتوان از قابلیت های CPUهای استاندارد و نیز واحدهای نوع F مستقل از واحدهای استاندارد مانند FMها و CPها به طور همزمان بهره گرفت. نهایتا کل سیستم با اجزای نرم افزاری استاندارد قابل ترکیب بندی و برنامه ریزی می باشد.
- این موضوع که می توان بخش های عادی و نیز ایمن در برابر خطا را در یک CPU اجرا کرد هزینه های جانبی مانند تحویل و نگهداری را کم می کند.

## ۱۷-٤ اجزای یک سیستم S7-400F

شکل زیر سخت افزار و نرم افزار لازم جهت ترکیب بندی و سپس کارکرد یک سیستم S7-400F را نشان می دهد.



### ارتباط اجزا

سیستم S7-400F متشکل از اجزای نرم افزاری و سخت افزاری می باشد که می بایست با هم ترکیب شوند تا بتوان یک سیستم ایمن در برابر خطا از نوع S7-400F را پیکر بندی نمود.

سیم بندی مدول های ورودی/خروجی نوع F

این مدول ها باید طوری به سنسورها و محرک ها متصل شوند که سطح ایمنی مورد نیاز تامین شود.

پیکربندی سخت افزار

نمودار مداری ورودی/خروجی ها بایستی با تنظیمات پارامترها مطابقت داشته باشد. در واقع پیکربندی سخت افزار که در محیط نرم افزار صورت می گیرد باید متناظر با پیکربندی واقعی سخت افزار باشد.

ایجاد برنامه از نوع F

این کار در محیط CFC و با استفاده از بلاک های موجود در کتابخانه Fail Safe Blocks صورت می گیرد. همچنین برای اتصال به مدول های ورودی/خروجی نوع F بایستی از بلاک های Driver نوع F استفاده و پارامترهای مربوط به آن را تنظیم کرد. البته برخی از این پارامترها خود به خود پس از ترکیب بندی این مدول ها انجام می شود.

وقتی که برنامه قابل اجرای نوع F تهیه می شود تست های مربوط به ایمنی به طور اتوماتیک انجام شده و توابع خطایابی دیگری هم به برنامه اضافه می شود.

### اجزای سخت افزاری

یک سیستم نوع F از اجزای سخت افزاری که نیازمندی های ایمنی مشخصی را برطرف می کنند تشکیل می شود. از قبیل :

- یک CPU مانند CPU 417-4H با تاییدیه برای ویژگی F
- مدول های ورودی/خروجی نوع F

البته یک سیستم نوع F را می توان با یا اجزای استاندارد S7-300 و S7-400 گسترش داد

### CPUهای دارای ویژگی F

برای سیستم های S7-400F و S7-400FH ، CPU با تاییدیه F به تنهایی یا در قالب یک سیستم اصلی/پشتیبان مقاوم در برابر خطا مورد استفاده قرار می گیرد. این تاییدیه اجازه می دهد CPU به عنوان یک CPU نوع F مورد استفاده قرار گیرد.

چنین CPUایی مجاز به استفاده در یک سیستم S7-400F/FH خواهد بود و فقط وقتی از ویژگی ایمن در برابر خطا در این CPU استفاده می شود که یک برنامه نوع F روی آن اجرا شود در غیر این صورت مانند یک CPU استاندارد عمل می کند.

البته ادغام برنامه عادی و برنامه نوع F نیز امکان پذیر است که در این حالت باید CPU برای کارکرد به صورت نوع F تنظیم شود. همچنین در این حالت داده مورد استفاده برنامه نوع F در برابر تغییرات احتمالی توسط برنامه عادی حفاظت می شود.

بخش های نوع F برنامه باید با استفاده از رمز عبور در برابر دستکاری غیر مجاز حفاظت شوند. همچنین برای افزایش میزان خطایابی باید از حداکثر قابلیت CPU در انجام روال های خودآزمون بهره گرفت.

## مدول های ورودی/خروجی نوع F

- مدول های ورودی دیجیتال شامل :
  - SM 326; DI 8 x NAMUR
  - SM 326; DI 24 x DC 24V
- مدول خروجی دیجیتال SM 326; DO 10 x DC 24V/2A
- مدول های ورودی آنالوگ SM 336; AI 6 5 13Bit (با قابلیت وقفه تشخیص خطا)

این مدول ها فقط در بخش ET 200M نصب و استفاده می شوند و البته می توانند در کاربردهایی که نیاز به ملاحظات ایمنی ندارند هم همراه با CPU های استاندارد به کار برده شوند.

همچنین می توان از این مدول ها به صورت تک کاناله و یا با قابلیت افزونگی استفاده کرد که در نتیجه در یک سیستم ET 200M و یا در چند سیستم از این نوع نصب و به کار برده می شوند.

## اجزای نرم افزاری

نرم افزار یک سیستم S7-400F شامل این اجزای می باشد :

- بسته نرم افزاری جداگانه S7 F Systems که مخصوص پیکربندی و برنامه ریزی سیستم های F می باشد.
- برنامه کاربر نوع F که روی CPU اجرا می شود.

امکانات بسته نرم افزاری S7 F Systems

- ترکیب بندی مدول های ورودی/خروجی نوع F به کمک بخش HWCONFIG
- دسترسی به کتابخانه "Fail Safe Blocks" برای برنامه نویسی برنامه های نوع F
- پشتیبانی از پردازش برنامه های نوع F و همچنین لحاظ کردن توابع تشخیص خطا در این برنامه ها

برنامه کاربر نوع F

این برنامه در محیط CFC و با استفاده از توابع موجود در کتابخانه "Fail Safe Blocks" تهیه می شود. این کتابخانه شامل توابعی جهت برنامه ریزی توابع ایمنی و نیز توابع تشخیص خطا و واکنش در برابر خطا می باشد. در واقع این توابع تضمین می کنند که در صورت بروز خطا، تشخیص آن به موقع انجام شده و واکنش لازم نسبت به آن صورت می گیرد و نهایتا سیستم F در یک حالت ایمن به کار خود ادامه می دهد.

برنامه کاربر می تواند شامل هر دو نوع F و عادی باشد که البته بخش های نوع F باید در روال های جداگانه CFC برنامه نویسی شوند و تبادل داده بین بخش های عادی و بخش های نوع F از طریق بلاک های خاص این کار یعنی Conversion Blocks صورت می گیرد.

در هنگام کامپایل کردن، دسته خاصی از توابع تشخیص خطا و واکنش در برابر خطا به طور اتوماتیک به برنامه F اضافه می شوند.

### نصب نرم افزار S7 F Systems

برای نصب این بسته نرم افزاری، باید قبلا حداقل نرم افزارهای زیر(یا نسخه های جدیدتر آنها) نصب شده باشند:

- STEP 7 as of V5.0 + Service Pack 3
- CFC V5.2
- PCS 7 driver blocks as of V5.0 with Service pack 3
- S7 H Systems as of V5.1 (optional for S7-400FH)

در این صورت با اجرای برنامه Setup.exe از CD حاوی S7 F Systems این نرم افزار نصب می شود. پس از نصب این نرم افزار بخش های لازم به محیط Simatic Manager اضافه می شود.

## ۱۷-۵ کار با سیستم های F

در این قسمت ملاحظاتی که اضافه بر نیازمندی های یک سیستم معمولی، هنگام طراحی یک سیستم ایمن باید در نظرگرفته شود تشریح می گردد.

برای طراحی چنین سیستمی ابتدا می بایست از سطح ایمنی مورد نیاز سیستم یعنی SIL مطلع بود و سپس باید با توجه به این سطح، توابع ایمنی و اجزای مناسب مانند PLCها، سنسورها و ... را انتخاب نمود و با تفکیک کردن بخش های F و بخش های معمولی به طراحی ، پیکربندی و برنامه ریزی سیستم پرداخت.

**روند پایه واصلی برای کار با سیستم های F**

| **پیکربندی سخت افزار S7-400F/FH** |
| --- |
| تنظیم آدرسهای مدول های F از طریق DIP switch<br>سیم بندی مدول ها مطابق با نقشه |

| **پیکربندی سیستم** |
| --- |
| تنظیم پارامترهای CPU برای اجرای برنامهF |
| تنظیم پارامترهای مدول های F مطابق با سطح ایمنی و نقشه |

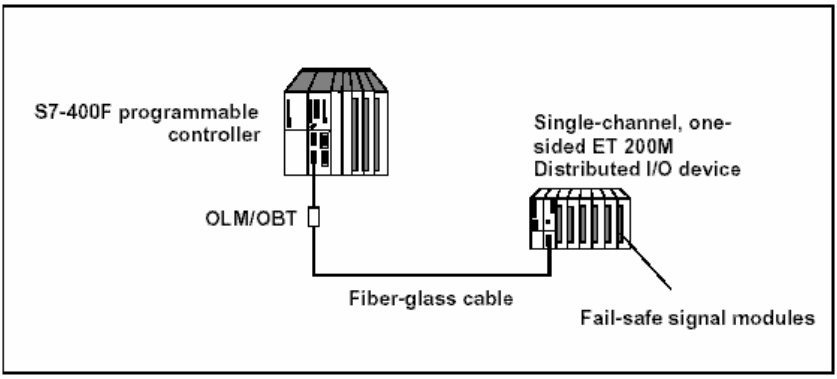
| **ایجاد برنامهF** |
| --- |
| اضافه کردن FBهای نوع F و تنظیم پارامترهای آن |
| تهیه کد اجرایی برنامه و بارگذاری آن روی CPU سیستم S7-400F/FH |

| **راه اندازی سیستم** |
| --- |
| تایید بخش های ایمن (F) توسط یک منخصص قبل از شروع به کار سیستم |

| **نگهداری سیستم** |
| --- |
| جایگزینی اجزای سخت افزاری |
| تغییر برنامه F |
| به روز کردن سیستم عامل |

## ۱۷-٦ پیکر بندی سیستم F

شکل زیر مثالی از ترکیب بندی سخت افزار را نشان می دهد.




برای داشتن چنین سیستمی این اجزا مورد نیازاست :

- یک PLC شامل :

- رک (UR2-H)
- منبع تغذیه (PS 407 10A)
- CPU 417-4H

- یک سیستم ET 200M دارای Bus فعال شامل :

- منبع تغذیه (PS307 5A)
- IM 153-2 FO with 1 OLM 12M (optical link module)
- مدول ورودی دیجیتال نوع F (SM 326F DI 24xDC24V)
- مدول خروجی دیجیتال نوع F (SM 326F DO10xDC24V/2A)

- سایر ملزومات :

- کابلها و کانکتورهای PROFIBUS
- کابل پلاستیکی فیبر نوری (2.2 mm)

DIP Switchهای این اجزا به صورت زیر تنظیم می شوند :

- OLM 12M Termination ON CH1
  - S1=1,S2=0 (1.5 Mbps)
  - S3=1 (Monitor OFF CH3), S4=S5=0 (LINE Mode CH3)
  - S6=1 (Monitor OFF CH2), S7=S8=0 (LINE Mode CH2)
- IM153-2 FO PROFIBUS address 3
- SM 326F DI 24Module address 8
- SM 326F DO10 Module address 24

## پیکربندی یک سیستم F توسط نرم افزار

برای ایجاد یک پروژه و پیکربندی سخت افزار بایستی به طریق زیر عمل کرد:

۱. در محیط Simatic Manager یک پروژه جدید به نام Fproject ایجاد کنید.

File > New

۲. یک سیستم S7-400 اضافه کنید.

Insert > Station > SMATIC 400 Station

۳. HWCONFIG این سیستم را باز کنید.

۴. اجزای مورد نیاز را از بخش Hardware Catalog انتخاب کرده و به سیستم اضافه کنید:

a. رک UR2 را اضافه کنید.

b. منبع تغذیه PS 407 10A را در slot اول قرار دهید.

c. CPU 417-4H V2.1 را در slot سوم قرار داده ویک زیر شبکه ایجاد کنید. این زیر شبکه بعدا به ET 200M متصل می شود.

Properties > PROFIBUS Interface DP Master > **New**.

d. CPU را انتخاب ودر منوی Properties قسمت Protection یک رمز عبور وارد نموده و گزینه CPU Contains Safety Program را نیز علامت بزنید.

e. مدول IM 153-2 FO را در قسمت PROFIBUS DP Master System قرار داده و در در بخش پارامترها از منوی Properties آدرس ۳ را وارد کنید.

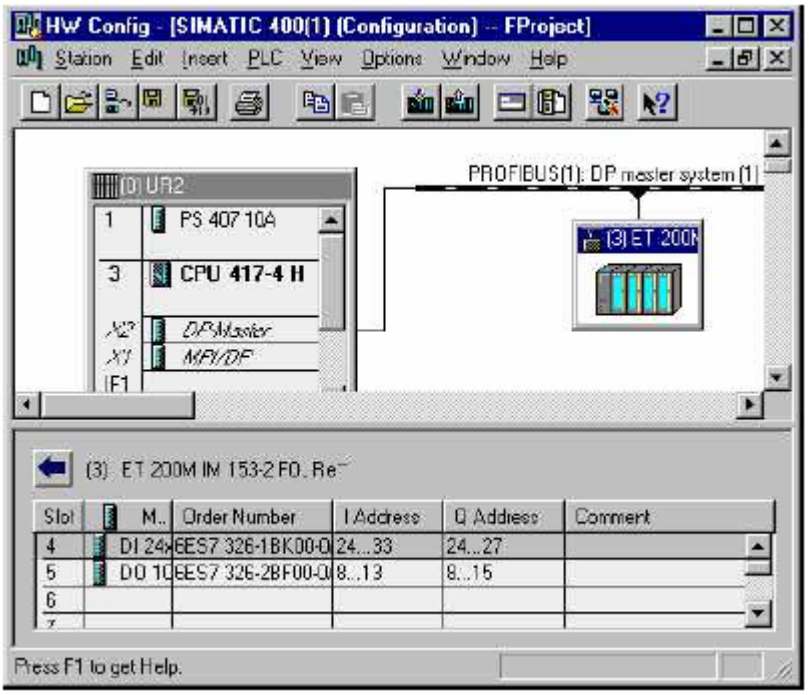
f. مدول ورودی SM 326F DI24xDC24V را از زیرمجموعه IM 153-2 FO انتخاب کرده و در slot چهارم ET 200M قرار دهید. سپس این مدول را انتخاب کرده و نام های سمبلیک متناظر با تک تک کانالها که بعدا در برنامه به کار گرفته خواهند شد را تعریف کنید. همچنین در بخش Inputs از منوی Properties، این گزینه ها را انتخاب کنید

"Enable Diagnostic Interrupt"
"Safety Mode" with "1oo1 Evaluation"

g. مدول خروجی SM 326F DO10xDC24V/2A را از زیرمجموعه IM 153-2 FO انتخاب کرده و در slot پنجم ET 200M قرار دهید. سپس نام های سمبلیک متناظر با تک تک کانالهارا تعریف کنید. همچنین در بخش Outputs از منوی Properties، گزینه زیر را انتخاب کرده و Group Diagnosis را یرای تمام کانال ها غیر فعال کنید.

"Safety Mode in Accordance with SIL2 / AK4"

۵. در پایان این ترکیب بندی را به کمک فرمان Station > Save and Compile ذخیره نمایید و سپس از طریق PLC > Download to Module این ترکیب بندی به CPU ارسال کنید




### ایجاد برنامه F

یک برنامه F شامل دو نمودار است :

a. نمودار F-Cyc جهت نظارت بر Cycle time

b. نمودار F Blocks مربوط ارتباطات سایر قسمت های برنامه

### ایجاد نمودارهای CFC

۱. در محیط Simatic Manager پس از انتخاب CPU 417-4H و به کمک منوی Insert > S7 Software > CFC دو پوشه نمودار یکی به نام F-Cyc و دیگری به نام F blocks ایجاد کنید.

۲. نمودار F-Cyc را باز کرده و از قسمت F-User Blocks در کتابخانه FailSafe Blocks یک FB به نام F-CYC-CO F را به نمودار اضافه کنید.

۳. ST MAX_CYC را انتخاب و قسمت FailSafe Block I/O را باز کنید. سپس مقدار پارامتر DATA را معادل 1s قرار دهید. همچنین می توانید نام این بلاک را به دلخواه تغییر دهید تا درک نمودارهای طولانی آسان تر باشد.

### ویرایش Run Sequence

FBهای نوع F را باید در گروه بلاک های Run-Time قرار داد. به این ترتیب :

با استفاده از فرمان Edit > Run Sequence به این محیط سوئیچ کرده و یک Cyc Group از نوع Run-Time در OB35 قرار دهید. سپس بلاک F-Cyc را به این گروه اضافه کنید.

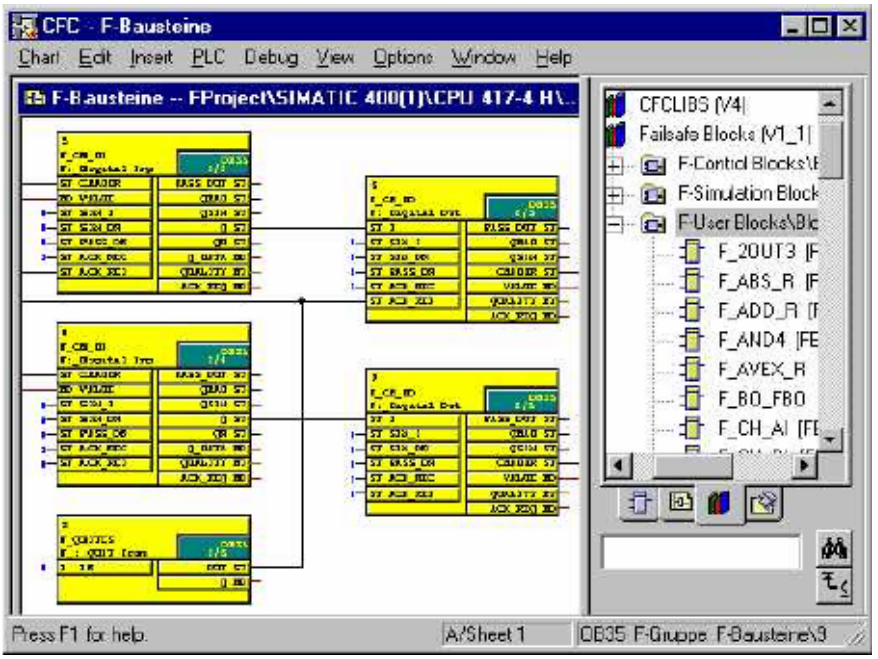
یگ گروه دیگر از نوع Run-Time به نام F Group ایجاد کرده و پس از انتخاب آن، آن را به صورت Predecessor for Installation تنظیم کنید.

### افزودن بلاک های F

۱. در محیط SIMATIC Manager نمودار F Blocks را بازکنید و دو درایور F_CH_DI را اضافه کنید که این دو جهت خواندن مقدار کانال های صفر و یک مدول ورودی دیجیتال F به کار می روند.
۲. پارامتر VALUE I/O را به اسامی سمبلیک کانال های صفر و یک ارتباط دهید. مثلا E24.0 برای کانال صفر و E24.1 برای کانال یک
۳. پارامتر ACK_NEC I/O را برابر ۱ قرار دهید. بنابراین در هنگام خطا تایید کاربر (از طریق پارامتر ACK_REI) جهت از سرگیری کنترل لازم خواهد بود.
۴. دو درایور F_CH_DO را اضافه کنید که این دو جهت نوشتن مقدار کانال های صفر و یک مدول خروجی دیجیتال F به کار می روند.
۵. پارامتر VALUE I/O را به اسامی سمبلیک کانال های صفر و یک ارتباط دهید. مثلا AB.0 برای کانال صفر و AB.1 برای کانال یک
۶. پارامتر ACK_NEC I/O را برابر ۱ قرار دهید.
۷. خروجی های Q هر دو درایور F_CH_DI را به ورودی های I متناظر در درایورهای F_CH_DO وصل کنید.

۸. یک بلاک F_QUITES را اضافه کرده و OUT I/O آن را به ورودی های ACK_REI هر چهار درایور F_CH_DI و F_CH_DO متصل نمایید.

۹. توسط منوی Run-Time group مجددا بررسی کنید که آیا تمام Block Fها در دو گروه Run-Time هستند یا خیر




## کامپایل کردن بلاک ها

ازطریق فرمان **Chart > Compile > Charts as Program** برنامه را کامپایل کنید و گزینه Generate Module Drivers را نیز فعال نمایید. همچنین توجه کنید که به هنگام ویرایش برنامه F رمز عبور پرسیده می شود.

پس از کامپایل کردن نمودارها، بلاک های کنترلی زیر نیز اضافه می شوند :

- در نمودار F_Cyc : F_TEST ، F_TESTM ، F_TESTC
- در نمودار F Blocks : F_PLK ، F_PLKO
- در یک نمودار جداگانه : F_M_DI24 and F_M_DO10 F1

همچنین OBهای لازم جهت تشخیص خطابه طور اتوماتیک به بلاک های برنامه اضافه می شوند.
پس از کامپایل کردن، می توان نمودارهای CFC را با دستور PLC > Download to Module برای CPU ارسال کرد.

**تذکر**: برنامه نویسی به روش CFC در این کتاب توضیح داده نشده است.

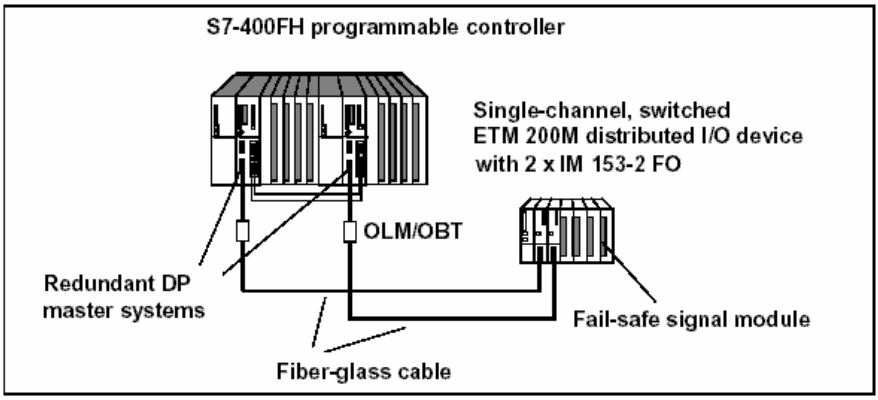
## ۷-۱۷ راه اندازی و نظارت بر خطاها در یک سیستم F

کلید انتخاب وضعیت CPU را در حالت RUN-P قرار داده و یک Warm Restart انجام دهید. پس از این کار خواهید دید که هر گاه ولتاژی به یکی از ورودی ها اعمال شود خروجی متناظر آن فعال می شود.
برای نظارت بر خطا ها بترتیب زیر عمل کنید:

۱. کانکتور جلوی مدول .SM 326F DI24xDC24V را از جای خود خارج کنید. با این کار در این مدول خطا اعلام شده، چراغ SF روشن و چراغ SAFE خاموش می شود. چراغ EXTF روی CPU هم روشن می شود ولی CPU در وضعیت RUN باقی می ماند.
۲. به کمک فرمان PLC > Operating Mode > Diagnostic Buffer مشاهده خواهید کرد که یک مدول با آدرس ۸ غیرفعال اعلام شده است اما به دلیل وجود OB82 وقفه این خطا منجر به توقف CPU نشده است.اطلاعات بیشتر در ارتباط با مدول معیوب از طریق PLC > Diagnose قابل دسترسی است.
۳. به نمودار F Blocks رفته و آن را در وضعیت Test قرار دهید. خواهید دید که ورودی/خروجی های QBAD از بلاک های F_CH_DI فعال هستند که این نشان دهنده خطا است.
۴. مجددا کانکتور را سر جای خود قرار دهید. بعد از حدود یک دقیقه چراغ SAFE روشن، SF خاموش و EXTF نیز خاموش می شود. همچنین در "Diagnostic Buffer" این مدول سالم و بی عیب اعلام می شود ولی در نمودار و در وضعیت Test همچنان خطا اعلام می شود و اگر یکی از ورودی های این مدول را فعال کنید خروجی متناظر آن در مدول خروجی فعال نخواهد شد. بنابراین این مدول باید راه اندازی شود و این کار نیاز به تایید کاربر ازطریق فعال کردن ACK_REI دارد. به این منظور می توان از یک FB به نام F_QUITES استفاده کرد به این ترتیب که یک بار به پایه IN آن مقدار ۶ و سپس در کمتر از یک دقیقه مقدار ۹ اعمال کنید. با این کار سیگنال خروجی ۱ در یک سیکل ایجاد و ACK_REI فعال می گردد. پس از این خطای بلاک ها رفع و خروجی ها متناظر با ورودی ها فعال و غیر فعال می شوند.

## ۸-۱۷ پیکربندی سیستم FH

شکل زیر مثالی از ترکیب بندی سخت افزار را نشان می دهد.




برای داشتن چنین سیستمی این اجزا مورد نیازاست :

- یک PLC شامل :

- رک (UR2-H)
- دو منبع تغذیه (PS 407 10A)
- دو CPU 417-4H
- چهار مدول همزمان سازی (synchronization)
- دو کابل فیبر نوری

- یک سیستم ET 200M دارای Bus فعال شامل :

- منبع تغذیه (PS307 5A)
- دو IM 153-2 FO with 2 OLM 12M (optical link module)
- مدول ورودی دیجیتال نوع F (SM 326F DI 24xDC24V)
- مدول خروجی دیجیتال نوع F (SM 326F DO10xDC24V/2A)

- سایر ملزومات :

- کابلها و کانکنورهای PROFIBUS
- کابل پلاستیکی فیبر نوری (2.2 mm)

DIP Switchهای این اجزا به صورت زیر تنظیم می شوند :

- OLM 12M Termination ON CH1
  - S1=1,S2=0 (1.5 Mbps)
  - S3=1 (Monitor OFF CH3), S4=S5=0 (LINE Mode CH3)
  - S6=1 (Monitor OFF CH2), S7=S8=0 (LINE Mode CH2)
- IM153-2 FO PROFIBUS address 3
- SM 326F DI 24Module address 8
- SM 326F DO10 Module address 24

رک ها را شماره گذاری کرده (صفر و یک) و محرک ها (actuator) ها را به کانال های خروجی وصل کنید و قابلیت تشخیص خطا برای کانال های بی استفاده را غیر فعال نمایید.

## پیکربندی یک سیستم FH توسط نرم افزار

برای ایجاد یک پروژه و پیکربندی سخت افزار بایستی به طریق زیر عمل کرد:

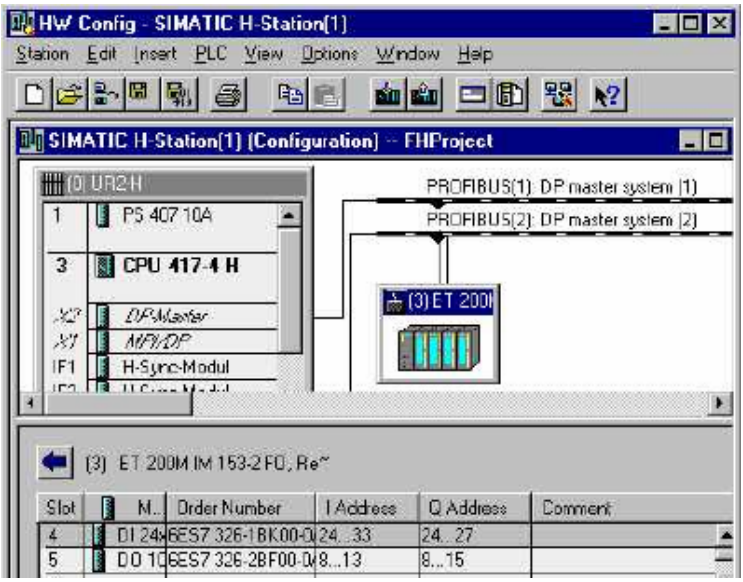
۱. یک پروژه جدید به نام FHproject ایجاد کنید.

۲. یک سیستم SIMATIC Hاضافه کنید.

۳. HWCONFIG این سیستم را باز کنید.

۴. رک UR2 را اضافه کنید.

۵. منبع تغذیه PS 407 10A را در slot اول قرار دهید.



۶. CPU 417-4H V2.1 را در slot سوم قرار داده ویک زیر شبکه ایجاد کنید. دو مدول همزمان سازی (H Sync) در مکان های IF1 و IF2 قرار دهید.

۷. CPU را انتخاب ودر منوی Properties قسمت Protection یک رمز عبور وارد نموده و گزینه CPU Contains Safety Program را نیز علامت بزنید.

۸. کل رک را کپی کنید و CPU دوم را به دومین زیر شبکه PROFIBUS متصل کنید.

۹. مدول IM 153-2 FO را به یکی از دو زیر شبکه PROFIBUS اضافه کرده و آدرس آن را برابر ۳ قراردهید. از این پس واحد ET 200M به طور اتوماتیک به هر دو زیر شبکه تصل می شود.

۱۰. مدول ورودی SM 326F DI24xDC24V را در slot چهارم ET 200M قرار دهید. سپس نام های سمبلیک متناظر با تک تک کانالها را تعریف کنید. همچنین در بخش Inputs از منوی Properties، این گزینه ها را انتخاب کنید

"Enable Diagnostic Interrupt"
"Safety Mode" with "1oo1 Evaluation"

۱۱. مدول خروجی SM 326F DO10xDC24V/2A را در slot پنجم ET 200M قرار دهید. سپس نام های سمبلیک متناظر با تک تک کانالهارا تعریف کنید. همچنین در بخش Outputs از منوی Properties، گزینه زیر را انتخاب کنید.

"Safety Mode in Accordance with SIL2 / AK4"
“Enable Diagnostic Interrupt"

۱۲. در پایان این ترکیب بندی را به کمک فرمان Station > Save and Compile ذخیره نماییدو سپس آن را CPU رک صفر (CPU0) ارسال کنید. همچنین به این نکته توجه کنید که در محیط SIMATIC Manager تمام بلاک ها در این CPU ذخیره می شوند.

### ایجاد برنامه FH

همانند آنچه که در مورد سیستم F گفته شد یک برنامه F تهیه کرده و پس از کامپایل کردن نمودارها، برنامه را روی CPU0 بارگذاری نمایید.

## ۱۷-۹ راه اندازی و نظارت بر خطاها در سیستم FH

کلیدانتخاب وضعیت CPU0 را در حالت RUN-P قرار داده و یک Warm Restart انجام دهید. سپس همین کار را برای CPU1 انجام دهید. در نتیجه CPU0 به عنوان سیستم اصلی راه اندازی می شود. CPU1 هم پس از برقراری link و به روز شدن اطلاعات و برنامه ها به عنوان سیستم پشتیبان کار می کند.

همچنین اولین مدول IM 153-2 FO که به CPU0 متصل است فعال شده و چراغ ACT روشن می شود.
برای نظارت بر خطاها بصورت زیر عمل کنید:

قطع شبکه PROFIBUS

۱. کابل PROFIBUS را از CPU0 جدا کنید. درنتیجه چراغ BUS2F چشمک زن و چراغ REDF روشن می شود. در این حالت IM 153-2 FO دوم فعال شده و IM 153-2 FO اول خطای Bus را اعلام می کند.
۲. Diagnostic buffer مربوط به CPU0 را بررسی کنید. خواهید دید که علیرغم آنکه افزونگی مختل شده است، ورودی/خروجی ها همچنان بدون خطا به کار خود ادامه می دهند.
۳. مجددا کابل PROFIBUS را سر جای خود قرار دهید. چراغ های نشان دهنده خطا خاموش می شوند ولی IM 153-2 FO دوم همچنان فعال باقی می ماند.

خطا در SM 326F DO10xDC24V/2A

۱. کانال شماره صفر این مدول خروجی را از محرک یا بار مقاومتی متصل به آن جدا کنید.
۲. به کانال صفر مدول ورودی ولتاژ اعمال کنید. حال باید کانال خروجی فعال شود ولی مدول خروجی خطا اعلام کرده و چراغ این کانال خاموش و چراغ SF روشن می شود.
۳. Diagnostic buffer را بررسی کنید. خواهید دید که در قسمت مربوط به این مدول، قطع سیم متصل به کانال صفر گزارش شده است.
۴. به نمودار F Blocks رفته و آن را در وضعیت Test قرار دهید. خواهید دید که ورودی/خروجی های QBAD از بلاک های F_CH_DI فعال هستند که این نشان دهنده خطا است.
۵. قطعی را که ایجاد کرده بودید برطرف و مجددا کانال را متصل کنید.
۶. به محض آن که ACK_REQ فعال شد مدول خروجی را مشابه آنچه برای یک سیستم F گفته شداز طریق F_QUITES راه اندازی کنید. در نتیجه دیگر خطا گزارش نشده و چراغ SF خاموش می شود.

## ۱۰-۱۷ مکانیزم های ایمنی

مکانیزم های ایمنی در CPU به دو بخش تقسیم می شوند :

- حفاظت در برابر دسترسی غیر مجاز به سیستم F، که این کار منجر به پیشگیری از خطا می شود.
- اجرای خود-آزمون ها (Self Test) که به تشخیص خطا کمک می کند.

توابع مربوط به ایمنی که عملیات تشخیص خطا و واکنش به آن را انجام می دهند عمدتا در برنامه F و در مدول های ورودی/خروجی نوع F گنجانده می شوند. این توابع به کمک بلاک های مناسب از نوع F پیاده سازی و توسط سخت افزار و نیز سیستم عامل CPU پشتیانی می شوند.همچنین این توابع در مدهای ایمنی فعال می شوند.

مد ایمنی مدول های ورودی/خروجی نوع F

به هنگام پیکربندی این مدول ها در HWCONFIG می توان از پارامتر "Safety Mode" برای تنظیم این مد در این مدول ها استفاده کرد. بدیهی است در صورت عدم انتخاب این مد، مدول مربوطه در حالت عادی و غیر ایمن کار خواهد کرد.

مد ایمنی برنامه F

معمولا برنامه F در مد ایمنی روی CPU اجرا می شود و بنابراین تمام مکانیزم های تشخیص و واکنش در برابر خطا فعال هستند. همچنین وقتی برنامه F در مد ایمنی است نمی توان آن را در حین اجرا تغییر داد. اما به کمک فرمان **Options > Customize Safety Program** می توان مد ایمنی را برای برنامه F فعال و غیرفعال کرد تا تغییر برنامه در حالت RUN میسر شود.

## خود-آزمون ها و تست فرمان ها

خود-آزمون ها Self Tests

این تست ها به منظور تشخیص خطا توسط سیستم S7-400F/FH اجرا می شوند و زمان تناوب اجرای آنها هنگام پیکربندی قابل تنظیم است.این زمان به طور پیش فرض ۹۰ دقیقه است و حداکثر مقدار آن ۱۲ ساعت است.

همچنین خود-آزمون های مربوط به ایمنی نبایستی تغییر داده شوند، در غیر این صورت CPU حداکثر ظرف ۲۴ ساعت متوقف خواهد شد.

اجرای این تست ها و نتیجه آنها در برنامه F و توسط بلاک F_TESTC بررسی می شود. این بلاک به طور اتوماتیک در هنگام کامپایل شدن به برنامه F اضافه می شود.

تست فرمان ها

برخی از فرمان ها و دستورات در کوتاه ترین سیکل اجرای برنامه F تست می شوند که این کار در بلاک F_TEST انجام می شود. این بلاک نیز به طور اتوماتیک در هنگام کامپایل شدن به برنامه F اضافه می شود.

## ۱۱-۱۷ واکنش در برابر خطا

اساس مفهوم ایمنی آن است که باید برای تمام متغیرها یک وضعیت و مقدار ایمن و خنثی وجود داشته باشد که به عنوان مثال برای مدول های دیجیتال این مقدار معادل صفر می باشد.

واکنش به خطا در CPU وسیستم عامل

چنانچه CPU خطایی را سخت افزاری (از طریق زمان سنجی) و یا نرم افزاری (انجام خود-آزمون ها)تشخیص دهد طبق پیش فرض به حالت STOP می رود.

واکنش به خطا در برنامه F

این وضعیت موجب انتقال به یکی از دو حالت ایمن می شود:

- توقف CPU : خروج از این حالت تنها با راه اندازی محدد (warm یا cold) امکان پذیر است.
- بلوکه شدن تمام خروجی های مربوطه به نحوی که در مقابل اختلال مصون باشند. در نتیجه خطاهای ورودی/خروجی یا ارتباطی که روی این خروجی ها تاثیر می گذارند بلوکه شده و CPU متوقف نخواهد شد. سپس با تایید کاربر (از طریق یک ورودی درایور F )خروجی ها از حالت بلوکه خارج می شوند.

در واکنش به خطا، توابع تشخیص و گزارش از نوع عادی و غیر ایمن نیز می توانند اجرا شوند. نکته دیگر این که چنانچه سیستم اصلی متوقف شود تغییر از سیستم اصلی به سیستم پشتیبان در S7-400FH رخ خواهد داد.

## ۱۲-۱۷ مدهای کاری یک سیستم S7-400F/FH

مدهای کاری این سیستم ها از لحاظ نحوه راه اندازی و عملکرد آنها در مد HOLD با سیستم های معمولی تفاوت دارد.

نحوه راه اندازی

نحوه راه اندازی توسط برنامه F به این ترتیب تعیین می شود که پس از هر بار وقفه در اجرای برنامه (به علت قطع تغذیه یا توقف CPU) از سرگیری اجرای برنامه تنها با مقادیر اولیه (Initial Value) بلاک های F امکان پذیر خواهد بود. نکته دیگر این که Warm Restart فقط در مورد بخش های عادی برنامه صورت می گیرد و برای بخش های F انجام شدنی نیست. پس از Cold Restart نیز اجرای برنامه F با مقادیر اولیه بلاک های نوع F از سرگرفته می شود.

به هنگام کامپایل کردن برنامه F، بلاک های دیگر و فراخوانی هایی که دیگر نباید تغییر داده شوند، به طور اتوماتیک در OB100 قرار می گیرند.

حفاظت در هنگام راه اندازی

از سرگیری اجرای برنامه F با مقادیر اولیه ممکن است به دلایل دیگری مانند بروز یک خطای داخلی نیز صورت پذیرد. چنانچه ماهیت فرآیند طوری باشد که این اتفاق نبایستی رخ دهد باید واکنش مناسب به آن در برنامه F پیش بینی شود. بلاک F_START به منظور تنظیم ازسرگیری اجرای برنامه F با مقادیر اولیه در اختیار کاربر قرار دارد.

حفاظت در هنگام Hot Restart

هنگامی که یک مشکل در ایمنی سیستم رخ دهد، برنامه F تمام خروجی های مربوطه را بلوکه می کند و خروج از این حالت با راه اندازی مجدد (Warm یا Cold) انجام شده و البته با این کار اطلاعات مربوط به خطا که روی CPU ذخیره شده است پاک می شود.

چنانچه پس از واکنش S7-400F به یک خطای داخلی، Hot Restart فرآیند مجاز نباشد باید روال های لازم جهت فعال کردن دستی خروجی ها پس از ازسرگیری اجرای برنامه F با مقادیر اولیه، برنامه ریزی شود.

مد HOLD

این مد در سیستم های S7-400F/FH پشتیبانی نمی شود و چنانچه اجرای برنامه با فراخوانی این مد متوقف شود برای لغو آن تنها می توان سیستم را مجددا راه اندازی کرد.(Warm یا Cold)

## ۱۳-۱۷ نظارت منطقی یا زمانی بر اجرای برنامه

خطاها می توانند منجر به ایجاد اشکال در اجرای برنامه شوند. بنابراین نظارت بر اجرای برنامه و نیز گردش اطلاعات این موضوع را آشکار خواهد کرد.

نظارت منطقی بر اجرای برنامه و گردش اطلاعات

برای انجام چنین نظارتی بلاک های نوع F به هنگام کامپایل کردن به برنامه اضافه می شوند. در هر دسته ای از بلاک های F، یک بلاک F_PLK (که قبل از خروجی ها فراخوانی می شود) و یک بلاک F_PLK_O (که بعد از خروجی ها فراخوانی می شود)اضافه می شود.

هنگامی که یک خطای مهم تشخیص داده شود، بررسی منطقی اجرای برنامه باعث درخواست توقف CPU با فراخوانی بلاک SFC46 می شود که پس از آن اجرای برنامه بایستی مجددا از سرگرفته شود (Warm Restart یا Cold Restart).

نظارت زمانی بر اجرای برنامه

این کار به دو روش انجام می شود :

- نظارت زنده و دائم به هنگام انجام ارتباطاتی که به نحوی به ایمنی مربوط می شوند
- نظارت بر زمان سیکل F

نظارت زنده

برنامه F متناوبا از طریق پرونکل های ایمن خاصی با مدول های F و دیگر برنامه های F سایر CPUها در خال ارتباط است. در این ارتباط طرف های گیرنده به هنگام بروز مشکل تابع واکنش به خطا را به صورت زیر اجرا می کنند:

- مدول های خروجی نوع F خروجی ها را غیرفعال می کنند
- بلاک های F_RCVBO و F_RCVR که در برنامه F سایر CPUها وجود دارند مقادیر جایگزین را به خروجی می فرستند.

پس از برطرف شدن مشکل یا باید تایید کاربر روی درایور F یا یکی از بلاک های F_RCVBO و F_RCVR اعلام شود یا این که Cold Restart یا Warm Restart انجام گردد.

نظارت بر زمان سیکل F

حداکثر زمان سیکل F با یکی از پارامترهای ورودی بلاک F_CYC_CO در نمودار برنامه F تعیین می شود که البته این بلاک باید در هر سیکل F حضور داشته باشد. چنانچه زمان سیکل از این مقدار بیشتر شود CPU متوقف شده و تمام خروجی ها به حالت ایمن می روند.

## مقادیر زمانی F

مقادیر زمانی که در برنامه F و توسط بلاک های F_TP، F_TON و F_TOFF تولید می شود توسط مکانیزم های ایمنی CPU نظارت می شوند. بدین منظور دو شمارنده زمانی مستقل از یکدیگر با هم مقایسه می شوند. تا وقتی که اختلاف این دو شمارنده در یک بازه زمانی ۵۰ ثانیه کمتر از ۱۰ میلی ثانیه یاشد، مشکلی وجود نخواهد داشت و زمان "صحیح" در نظرگرفته می شود. در غیر این صورت خطای سخت افزاری اعلام خواهد شد.

حداکثر خطای زمانی قابل قبول را می توان بر مبنای جدول بعد محاسبه کرد :

| User Times From | To | Max. Inaccuracy |
|---|---|---|
| 10 ms | 50 s | ± 5 ms |
| > 50 s | 100 s | ± 10 ms |
| ... | ... | ... |
| > n* 50 s | (n+1)*50 s | ± (n+1)*5 ms |

البته خطای واقعی کمتر از این مقادیر خواهد بود. همچنین بایستی خطای زمانی ناشی از پردازش وقفه های دوره ای را نیز در نظر گرفت.

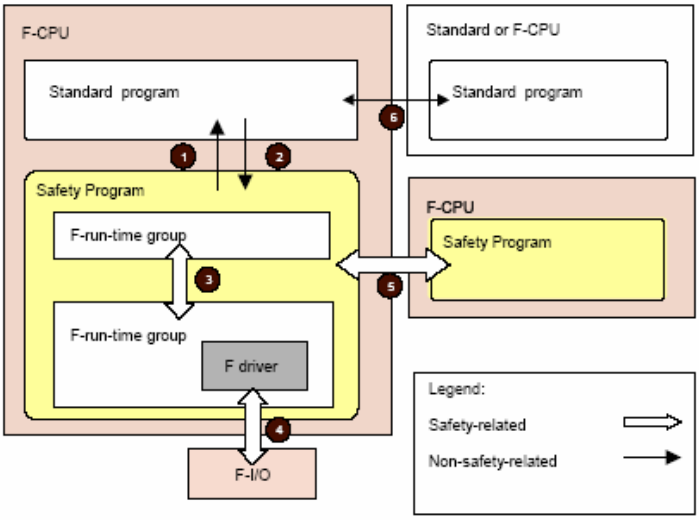
## حفاظت به کمک رمز عبور

با استفاده از رمز عبور می توان سیستم S7-400F/FH را برابر دسترسی های غیر مجاز مانند بارگذاری برنامه روی CPU محافظت کرد. در این سیستم ها علاوه بر یک رمز عبور برای CPU، رمز عبور دیگری برای برنامه F (به نام رمز عبور F)مورد نیاز است. جدول زیر مقایسه این دو رمز عبور را نشان می دهد.

| | **CPU Password** | **Password for Safety Program** |
|---|---|---|
| Input | In HWCONFIG, during configuration of the CPU, "Protection" tab in the "Properties" dialog box | In SIMATIC Manager, **Options** > **Customize Safety Program** |
| Requested at | Downloading of the whole program from CFC or SIMATIC Manager<br>Downloading of F program changes from CFC<br>Downloading and deletion of F blocks from SIMATIC Manager<br>Downloading to the EPROM memory card on the CPU from SIMATIC Manager<br>Memory reset from CFC or SIMATIC Manager<br>Modification of F constants in CFC test mode | Compilation of changes to the F program<br>Switching safety mode on and off<br>Downloading of changes to the data of the F program when safety mode is inactive<br>Modification of F constants in CFC test mode |
| Validity | Legitimization is valid without restrictions, until explicitly withdrawn via the corresponding SIMATIC Manager function or until all Step 7 applications have been terminated. | An hour after the password has been entered or until the access rights are explicitly canceled |

## ۱۷-۱٤ ارتباطات ایمنی

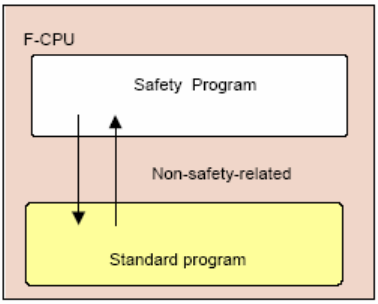
شکل زیر گزینه های ارتباطی موجود در سیستم F را نشان می دهد.




| Number | Communication Between | And | Safety-Related |
|---|---|---|---|
| 1 | F program in F CPU | Standard program | No |
| 2 | Standard program | F program | No |
| 3 | F run-time group | F run-time group | Yes |
| 4 | F program in F CPU | F SM | Yes |
| 5 | F program in F CPU | F program in F CPU | Yes |
| 6 | Standard program in standard or F CPU | Standard program in standard or F CPU | No |

ارتباط بین برنامه F و برنامه معمولی

این دو برنامه قالب های مختلفی از داده را استفاده می نمایند بنابراین جهت تبادل داده بین این دو برنامه، لازم است تا از بلاک های خاصی که کار تبدیل فرمت داده را انجام می دهند کمک گرفته شود.



| From | To | Block | Safety-Related |
|---|---|---|---|
| F program | Standard program | F_F*data type_data type* | No |
| Standard program | F program | F_*data type*_F*data type* | No |

پارامترهایی که به برنامه F فرستاده می شوند داده هایی با فرمت F هستند بنابراین اگر برنامه معمولی بخواهد دادهای برنامه F را پردازش کند (مثلا آن ها را نظارت کند) باید بلاکی جهت تبدیل داده (F_Fdatatype_datatype) در نمودار CFC برنامه معمولی اضافه شود. این بلاک ها را می توان در کتابخانه User Blocks در بخش Failsafe Blocks یافت.

حال اگر قرار باشد که داده های برنامه معمولی در برنامه F پردازش شوند بایستی به کمک بلاک های تبدیل داده (F_datatype_ Fdatatype) در برنامه F از دادهای با فرمت عادی دادهایی با قرمت F درست کرد و اگر لازم باشد می بایست صحت این تبدیل و معقول بودن نتیجه آن را نیز با برنامه نویسی و استفاده از بلاک های F بررسی کرد.

ارتباط بین گروه های Run Time نوع F

گروه های Run Time گروه هایی هستند که شامل بلاک های F می باشند. تبادل داده بین این گروه ها در یک برنامه ازطریق بلاک های F_S_datatype و F_R_datatype صورت می گیرد تا این ارتباط در یک وضعیت ایمن انجام شود. توسط این بلاک ها تعداد مشخصی ازپارامترها که همگی از یک فرمت هستند منتقل می شود. برای آنکه ارتباط بین این نوع گروه ها در OB های وقفه ای مختلف میسر شود می بایست وقفه دوره ایی (Cyclic Interrupt) را که سیکل کوتاه تری دارد به عنوان اولویت بالاتر تنظیم نمود.

بلاک F_S_datatype در گروه Run-Time نوع F که فرستنده اطلاعات می باشد قرار می گیرد و پارامترهای ورودی آن به پارامترهای ارسالی دیگر بلاک های نوع F ارتباط داده می شوند. همچنین بلاک F_R_datatype در گروه Run-Time نوع F که گیرنده اطلاعات می باشد قرار می گیرد و پارامترهای خروجی آن به ورودی های دیگر بلاک های نوع F ارتباط داده می شوند. نهایتا اتصال این دو بلاک ازطریق ارتباطات لازم در برنامه CFC انجام می شود.

ارتباط بین CPU و مدول های ورودی/خروجی F

ارتباط ایمن از طریق ProfiSafe

ProfiSage نسخه ایمن PROFIBUS DP/PA می باشد که از طریق آن برنامه F با مدول های ورودی/خروجی F ارتباط برقرار می کند. این پروتکل در برنامه F (در بلاک های F) و در مدول های F (در برنامه موجود روی سخت افزار آنها) پیاده سازی می شود.

این ارتباط ایمن توسط انتقال داده به صورت دوره ایی انجام می شود و پارامتر مهم آن "زمان نظارت" (monitoring time) است که به هنگام پیکربندی مدول نوع F تعیین می شود و به طور اتوماتیک به عنوان یک پارامتر ورودی به بلاک های F ارسال می شود.

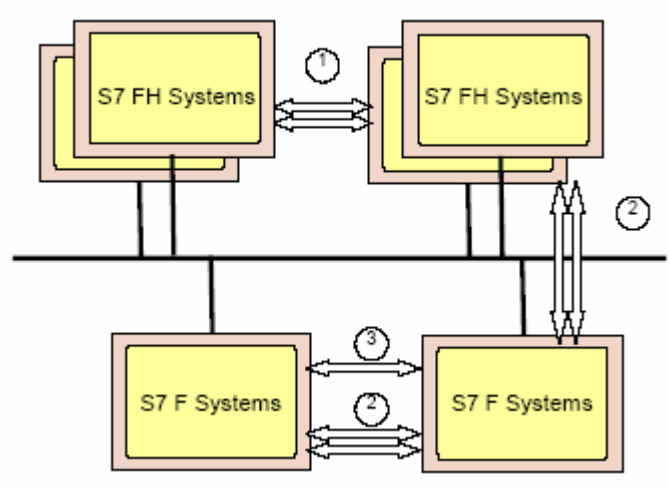
ارتباط عادی

برای برقراری چنین ارتباطی می توان از روش های معمول مانند دسترسی مستقیم و یا دسترسی به Process Image یا Process Recordsاستفاده کرد. به عنوان مثال ارسال اطلاعات مربوط به تشخیص وضعیت که از توع F نیستند توسط انتقال غیر دوره ایی مقادیر ثبت شده از مدول F انجام می گردد.

ارتباط ایمن بین CPUها

گزینه های موجود برای این ارتباط

ارتباط ایمن بین CPUها ازطریق یک ارتباط S7 پیکر بندی شده به طور استاندارد یا به طور مقاوم در برابر خطا انجام می شود.



| Number | Communication From... | To | Connection Type | Safety-Related |
|---|---|---|---|---|
| 1 | S7-400FH | S7-400FH | S7 connection, fault-tolerant | Yes |
| 2 | S7-400F/FH | S7-400F | S7 connection, fault-tolerant | Yes |
| 3 | S7-400F | S7-400F | S7 connection | Yes |

برای برقراری ارتباط بین برنامه های F موجود روی CPUهای مختلف بایستی از بلاک های F_SENDdatatype و F_RCVdatatype استفاده کرد که در این ارتباط تعداد مشخصی از پارمترهای نوع F_data (که می تواند از نوع F_BOOL یا F_REAL باشد) منقل می شوند.

نکته مهم این است که استفاده از شبکه های عمومی جهت برقراری ارتباط ایمن بین CPUها مجاز نیست.

ارتباط با CPUهای عادی

ارتباط مستقیم بین یک برنامه F و یک CPU امکان پذیر نیست. این کار باید از طریق یک برنامه عادی که روی CPU نوع F اجرا می شود انجام گردد و باید قبل از آن به کمک بلاک های تبدیل داده، داده های نوع F را به داده های معمولی تبدیل کرد. سپس ارتباط برنامه عادی از طریق توابع ارتباطی استاندارد صورت می پذیرد.

## ۱۵-۱۷ نکات پیکربندی سخت افزار و تنظیم پارامترها

پیکربندی یک سیستم ایمن در برابرخطا همانند یک سیستم استاندارد انجام می شود و به ترتیب زیر است :

- ایجاد پروژه و افرودن سیستم ها به آن
- پیکربندی سخت افزار و شبکه
- ارسال داده های سیستمی و مربوط به پیکربندی به PLC

البته تغییر تنظیمات یک سیستم F نیاز به اجازه دسترسی دارد و به هنگام پیکربندی چنین سیستمی باید این موارد را در نظر گرفت:

- واحد ET 200M باید تنها شامل مدول های F باشد
  - استثنائا مدول SM 331; AI 2 x 12Bit که یک مدول معمولی است می تواند در کنار مدول های F به کار گرفته شود
- تنها از IM 153-2 FO می توان در واحد ET 200M و در کنار سایر مدول های ورودی/خروجی F استفاده کرد.
- آدرس مدول های F باید توصط سوئیچ های موجود روی آنها در محدوده ۸ تا ۸۱۹۱ تعیین شده و با آدرس پیکربندی شده مطابقت داشته باشد.

- گزینه مربوط به قابلیت اجرای برنامه F توسط CPU بایستی فعال شده و رمز عبور نیز تعیین گردد.
- چنانچه پیکربندی تغییر کند (مثلا یکی از تنظیمات CPU عوض شود) مجددا باید برنامه F را کامپایل و به CPU ارسال کرد و توجه داشت که همیشه قبل از ارسال برنامه، باید پیکربندی به CPU فرستاده شده باشد.

### نکات مهم در تنظیم پارامترهای CPU

به منظور عیرفعال شدن نظارت زمانی در هنگام تغییر از سیستم اصلی به سیستم پشتیبان، می بایست بلاک های OB3x مورد استفاده در برنامه های F برای اولویت ۱۵ به بالا تنظیم شوند (از طریق منوی Cyclic Interrupts).

OB وقفه دوره ای موجود در برنامه F باید برای حالت "OB های وقفه دوره ای با کاربرد خاص " زیر تنظیم شود. در این صورت این وقفه پیش از آغاز زمان بلوکه شدن و به هنگام به روز کردن سیستم پشتیبان برای اولویت های بالای ۱۵ ،فراخوانی می گردد. برای انجام این کار از منوی CPU Properties صفحه H parameters را انتخاب کنید و سپس در قسمت "Cyclic Interrupt OB with Special Handling" شماره OB وقفه دوره ایی که دارای بالاترین اولویت می باشد و بلاک های برنامه F با توجه به آن در CFC مربوطه تنظیم شده اند را وارد کنید.

### نکات مهم در تنظیم پارامترهای مدول های F

- برای این مدول ها باید سطح ایمنی را تنظیم کرد.
- می توان برای افزایش میزان دسترس پذیری، این مدول ها را به صورت افزونه استفاده کرد که در این صورت مدول های افزونه می توانند در یک رک یا واحد ET 200M یا در رک ها و واحدهای مختلف نصب شوند.
- این مدول ها فقط به طور غیر مستقیم و از طریق درایورهای F قابل دسترسی هستند.
- تنظیم پارامترهای آنها از طریق بلاک های SFC امکان پذیر نیست.
- باید برای تمام کانال ها نام های سمبلیک تعریف و این نام ها در قسمت VALUE I/O هر داریور F وارد شود. با این کار پارامترهای مدول که در HWCONFIG تعیین شده با پارامترهای I/O موجود در درایور های مربوطه نوع F، به طور اتوماتیک تنظیم خواهد شد.
- می توان برای مدول یک نام(حداکثر ۱۲ حرف) انتخاب کرد و چنانچه درایور این مدول به طور اتوماتیک درج شود این نام برای آن نیز استفاده می شود و در نتیجه درک ارتباط مدول و درایورش آسان تر خواهد شد.

- پارامتر "Group Diagnosis" ارسال پیام های نشان دهنده وضعیت (مانند قطع اتصال، اتصال کوتاه و ...) از مدول به CPU را فعال یا غیر فعال می کند. این پارامتر را می توان برای کانال های بی استفاده غیر فعال کرد و باید برای تمام کانال های مورد استفاده فعال شود. در حالت کلی چنانچه این پارامتر غیر فعال شود اگر مدول ورودی باشد به هنگام خطا مقدار صفر بدون همراهی هیچ پیام خطایی به CPU ارسال می شود. اما اگر مدول خروجی باشد:
  - CPU هیچ پیامی دریافت نمی کند و خروجی ها ( بسته به تنظیمات درایورها) غیر فعال نخواهند شد.
  - اگر این گزینه جداگانه و برای تک تک کانال ها غیر فعال شده باشد، کانال های متاثر از خطا خاموش نخواهند شد.
  - اگر این گزینه برای نیمی از کانال ها غیر فعال شده باشد به هنگام خطا فقط نیمی از کانال های این مدول خاموش می شوند.

### پیکربندی مدول های F به طور افزونه

- هر دو مدول باید از یک نوع بوده و به یک صورت تنظیم شده باشند.
- یک "زمان نظارت" باید برای هر دو تعیین شده باشد.
- مد ایمن برای هر دو مدول باید فعال شده باشد.
- در منوی افزونگی گزینه "Redundancy 2x" برای مدول دوم انتخاب شود.
- در صفحه "Find Redundant Module" باید مدولی که مد نظر است انتخاب شود.
- اختلاف زمانی دو مدول افزونه در صورت نیاز قابل تنظیم است.

### پیکربندی شبکه و ارتباطات

تنها تفاوت پیکربندی شبکه و ارتباطات در یک سیستم ایمن با یک سیستم عادی و استاندارد در این است که بلاک های خاص نوع F جهت انجام ارتباطات مورد استفاده قرار می گیرد و چون این بلاک ها فقط در برنامه F قابل استفاده هستند در نتیجه ارتباط ز طریق دو برنامه F موجود روی دو CPU صورت می گیرد.

### وظایف و قابلیت های سیستم برنامه ریز

این وظایف با آنچه که برای یک سیستم عادی و غیر ایمن در اختیار است فرق چندانی ندارد جز این که برای انجام موارد زیر رمز عبور و سطح مجاز دسترسی بررسی می شود:

- بارگذاری کل برنامه از محیط CFC یا محیط SIMATIC Manager
- بارگذاری تغییرات برنامه F از محیط CFC
- بارگذاری و تشخیص بلاک های F از محیط SIMATIC Manager

- بارگذاری روی حافظه EPROM در CPU
- Reset کردن حافظه از محیط CFC یا محیط SIMATIC Manager

**نکات مهم**

- چنانچه در حالت Online مقادیر I/O بلاک های نوع F تغییر داده شود، CPU متوقف خواهد شد.
- پس از درخواست مد HOLD (جهت ایجاد Breakpoint در برنامه)بایستی Cold Restart یا Warm Restart انجام شود.

## تنظیم، تغییر و لغو حق دسترسی

برای ایجاد حق دسترسی به CPU، در محیط SIMATIC Manager برنامه S7 یا خود CPU را انتخاب کنید و از طریق فرمان **PLC > Access Rights > Setup** رمز عبوری را که قبلا هنگام پیکربندی CPU انتخاب کرده بودید را در اینجا نیز وارد کنید.

این حق دسترسی تا وقتی که از طریق فرمان **PLC > Access Rights > Cancel** لغو نشود و یا از برنامه خارج نشوید، معتبر می باشد.

تغییر رمز عبور تنها با تغییر پیکربندی میسر است. برای این کار در سیستم S7-400F، CPU باید متوقف شود ولی در سیستم S7-400FH نیازی به این کار نیست.

برای ایجاد یا تغییر رمز عبور برای برنامه F، در محیط SIMATIC Manager برنامه S7 یا خود CPU را انتخاب کنید و از طریق فرمان **Options > Customize Safety Program** در بخش "Password..."رمز عبور را وارد کنید. چنانچه رمز عبور تعیین نشود برای اولین بار که برنامه کامپایل می شود باید آن را تعریف کنید. در مجموع این رمز عبور در موارد زیر پرسیده می شود :

- کامپایل برنامه F تغییر داده شده
- فعال و غیر فعال کردن مد ایمنی
- بارگذاری داده های تغییر داده شده برنامه F هنگامی که مد ایمنی غیر فعال است
- تغییر مقادیر ثابت نوع F هنگامی که محیط CFC در مد تست است

**نکته**

اعتبار زمانی رمزعبور فقط یک ساعت وپس از این مدت دوباره پرسیده می شود.

برای لغو رمز عبور برای برنامه F، در محیط SIMATIC Manager برنامه S7 یا خود CPU را انتخاب کنید و از طریق فرمان **Options > Customize Safety Program** در بخش "Password..." گزینه "Cancel Access Rights" را انتخاب کنید.

# ضمیمه ۷ - برنامه های کاربردی با CPU های Compact

**مشتمل بر :**

**برنامه ۱** : کاربرد CPU312 IFM همراه با SFB29 برای شمارش در فرآیند بطری پر کنی

**برنامه ۲** : کاربرد CPU312 IFM همراه با SFB29 برای شمارش (تکمیل برنامه ۱ )

**برنامه ۳** : کاربرد CPU312 IFM همراه با SFB29 برای شمارش (تکمیل برنامه ۱ و ۲ )

**برنامه ٤** : کاربرد CPU312 IFM همراه با SFB30 برای اندازه گیری سرعت شافت

**برنامه ٥** :کاربرد CPU314 IFM همراه با SFB39 برای کنترل موقعیت در سیستمی با تغذیه کنتاکتوری

**برنامه ٦** :کاربرد CPU314 IFM همراه با SFB39 برای کنترل برش در سیستم مبتنی بر درایو

**برنامه ۷**: انجام Positioning با خروجی آنالوگ CPU31xC و توسط SFB44

**برنامه ۸** : انجام Positioning با خروجی های دیجیتال CPU31xC و توسط SFB46

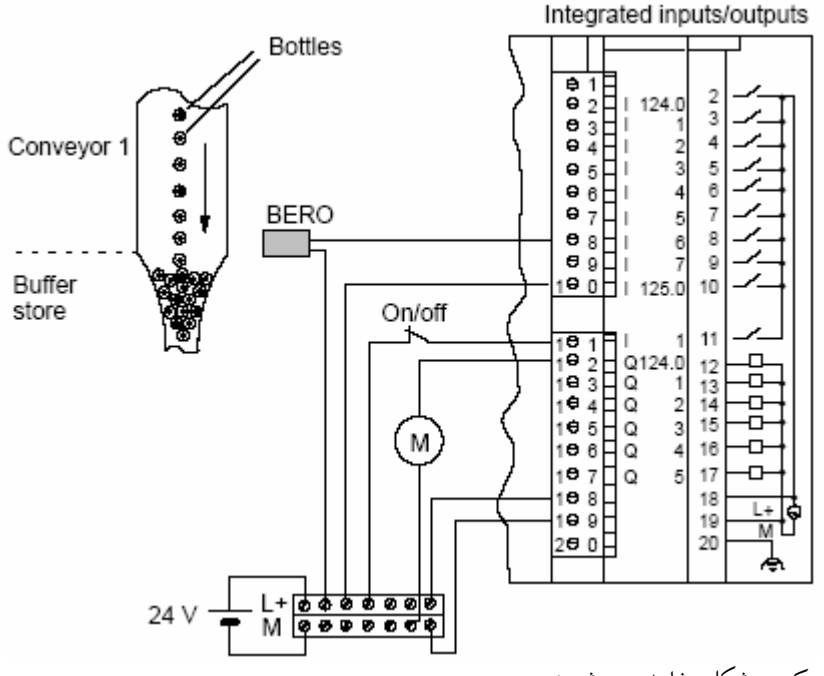
**برنامه ۹** : انجام عملیات شمارش با CPU31xC و توسط SFB47

**برنامه ۱۰** : اندازه گیری سرعت شافت با CPU31xC و توسط SFB48

**برنامه ۱۱**: استفاده از PWM برای کنترل دما با CPU31xC و توسط SFB49

**برنامه ۱ : کاربرد CPU312 IFM همراه با SFB29 برای شمارش**

در فرآیند بطری پرکنی بطری های پر شده مانند شکل زیر از روی یک نوار نقاله به داخل یک محل ذخیره که بافر نامیده می شود هدایت می شوند. بافر برای این منظور تعبیه شده که همواره از وجود تعداد بطری کافی در این محل اطمینان وجود داشته باشد.ظرفیت بافر ۲۵۰ بطری است و اگر پر شود لازم است موتور نوار نقاله ۱ خاموش گردد. تشخیص بطری های عبوری از روی نوار توسط یک سنسور پروکسیمیتی BERO انجام می شود. کلید نرمال بسته ای نیز برای اپراتور تعبیه شده که توسط آن اپراتور میتواند در صورت بروز خطا عمل شمارش را متوقف نماید.




همانطور که در شکل مشاهده می شود:

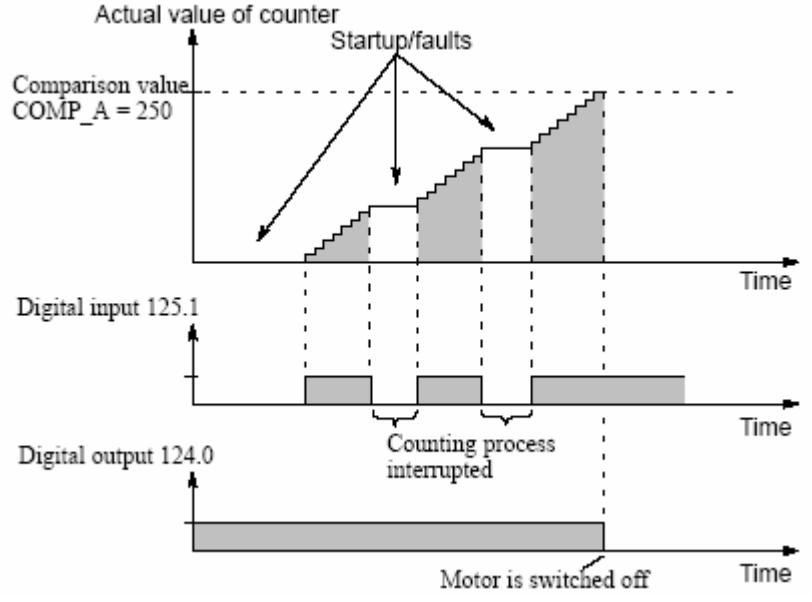
**I 124.6** : متصل به پروکسیمیتی سوئیچ است . که عبور هر بطری را تشخیص می دهد. در CPU312 IFM این آدرس برای شمارش افزایشی کانتر SFB29 بکار می رود.

**I 125.0** : این ورودی مستقیماً به تغذیه ۲۴ ولت متصل شده تا همیشه یک باقی بماند. معنای این کار با توجه به نقش I 125.0 اینست که نیازی به معکوس شدن جهت شمارش نیست بعبارت دیگر ورودی UP(یعنی I 124.6) همیشه برای افزایش شمارنده استفاده می شود نه کاهش آن.

**I125.1** : سوئیچ نرمال بسته که اپراتور با آن میتواند عمل شمارش را متوقف کند به این ورودی متصل است

**Q124.0**: این خروجی که به موتور نوار نقاله ۱ متصل است در برنامه از خروجی مقایسه گر A فرمان می گیرد. تا اگر تعداد بطری به ۲۵۰ رسید نوار قطع شود.

عملکرد مدار در شکل زیر نشان داده شده است همانطور که مشاهده می شود در زمانی که اپراتور کلید نرمال بسته را فعال کرده شمارش متوقف ولی موتور کار می کند.




برنامه به زبان STL در ادامه آورده شده است. در این برنامه ابتدا SFB29 در بلاک راه اندازی یعنی OB100 صدا زده می شودتا شرایط اولیه فراهم شودسپس در OB1 در صورتی که قبلاً SFB29 در هنگام راه اندازی اجرا شده باشد مقادیر اصلی به SFB29 اعمال شده و کار کنترل انجام می شود.در هردو OB فانکشن فوق با DB63 که پیش فرض سیستم است صدا زده شده است.

در OB100 هدف از صدا زدن SFB29 ری ست کردن این فانکشن و آماده سازی آن برای اجرا در OB1 است. همانطور که در برنامه OB100 مشاهده میشود در انتهای برنامه وضعیت بیت BR چک شده و در متغیر M24.0 ریخته شده است. اگر اجرایSFB29 در OB100 موفقیت آمیز باشد BR=1 خواهد شد و در اینصورت OB1 اجازه خواهد داشت که کار اصلی را انجام بدهد ولی در صورت بروز خطا و عدم اجرای SFB29 در OB100 بیت BR=0 خواهد شد و در OB1 فانکشن SFB29 فراخوان نمی شود.

**OB100**

Network 1

```
      CALL          SFB 29, DB 63    Call of SFB 29 with instance DB
 PRES_COUNT:        =
PRES_COMP_A:        =
PRES_COMP_B:        =
   EN_COUNT:        =

      EN_DO:        =

 SET_COUNT:         = FALSE          SET_COUNT = 0, to generate pos. edge in OB 1.
SET_COMP_A:         = FALSE          SET_COMP_A = 0, to generate pos. edge in OB 1.
SET_COMP_B:         =
     COUNT:         =
    COMP_A:         =
    COMP_B:         =
  STATUS_A:         =
  STATUS_B:         =
 A                  BR               Scan BR bit (= ENO at SFB 29) to enable SFB 29 in OB 1
 =                  M 24.0
```

در OB1 ابتدا وضعیت M24.0 چک میشود و اگر یک باشد به معنی اجرای موفقیت آمیز SFB29 در OB100 است و OB1 کار نرمال را با صدا زدن SFB29 شروع میکند در غیر اینصورت به سطر m01 پرش کرده و سیگنال خطا را در M24.1 آشکار می سازد.

با صدا زدن SFB29 مقدار اولیه کانتر صفر و حد بالای شمارش ۲۵۰ داده میشود. در اینحالت کانتر با ورودی Set_Count فعال شده و مقایسه گر A نیز با True شدن ورودی Set_Comp_A فعال میگردد. در اینحالت اپراتور با کلید متصل به I125.1 می تواند در صورت لزوم شمارش را متوقف سازد.ورودی EN_DO نیز فعال شده تا خروجی مستقیماً ازمدول کنار CPU اعمال گردد.

مقدار شمارش شده در خروجی Count و مقدار مبنای مقایسه در خروجی Comp_A به روی متغیر های ۳۲ بیتی ریخته شده که توسط جدول VAT به سهولت قابل مشاهده است.

وقتی مقدار شمارش به حد بالا میرسد توسط مقایسه گر خروجی Status_A روشن است این خروجی روی بیت M26.0 ریخته شده و در انتهای برنامه فقط در شرایطی که M26.0 یک نیست خروجی Q124.0 که به نوار نقاله ۱ متصل است فعال میشود.

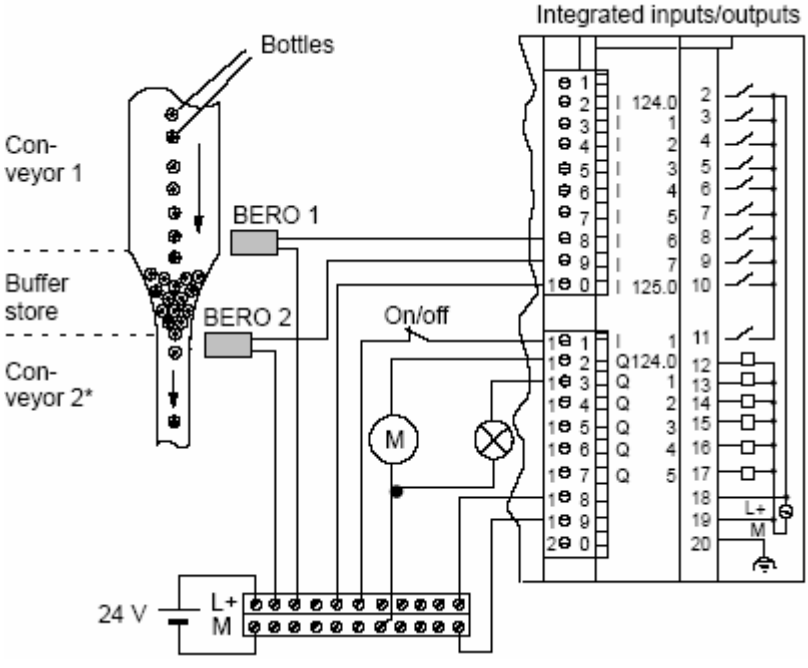
**OB 1**

Network 1

| | | | |
|---|---|---|---|
| | A | M 24.0 | If M 24.0 = 1, i.e. EN = 1 at SFB 29, SFB is executed; |
| | JNB | m01 | If RLO = 0, jump to m01 |
| | CALL | SFB 29, DB 63 | Call SFB 29 with instance DB |
| | PRES_COUNT: | = L#0 | Define start value PRES_COUNT |
| | PRES_COMP_A: | = L#250 | Define comparison value PRES_COMP_A |
| | PRES_COMP_B: | = | |
| | EN_COUNT: | = I 125.1 | The counting process can be interrupted by activating the normally-closed switch |
| | EN_DO: | = TRUE | Digital outputs are enabled for Counter integrated function |
| | SET_COUNT: | = TRUE | Start value PRES_COUNT is passed |
| | SET_COMP_A: | = TRUE | Comparison value PRES_COMP_A is passed |
| | SET_COMP_B: | = | Assignment of output parameters |
| | COUNT: | = MD 14 | |
| | COMP_A: | = MD 18 | |
| | COMP_B: | = | |
| | STATUS_A: | = M 26.0 | |
| | STATUS_B: | = | |
| m01: | A | BR | Query BR bit (= ENO at SFB 29) for |
| | = | M 24.1 | error evaluation |
| | AN | M 26.0 | If status bit A not set, conveyor |
| | S | Q 124.0 | belt 1 runs, Q 124.0 is reset by IF if comparison value COMP_A reached from below. |

**برنامه ۲ : کاربرد CPU312 IFM همراه با SFB29 برای شمارش (تکمیل برنامه ۱ )**

برنامه قبلی را بگونه ای تعمیم می دهیم که اگر تعداد بطری موجود در بافر از ۵۰ عدد کمتر شد یک لامپ سیگنال روشن گردد. در اینحالت بطری های خارج شده از بافر توسط یک سنسور دیگر مانند شکل آشکار می گردند.

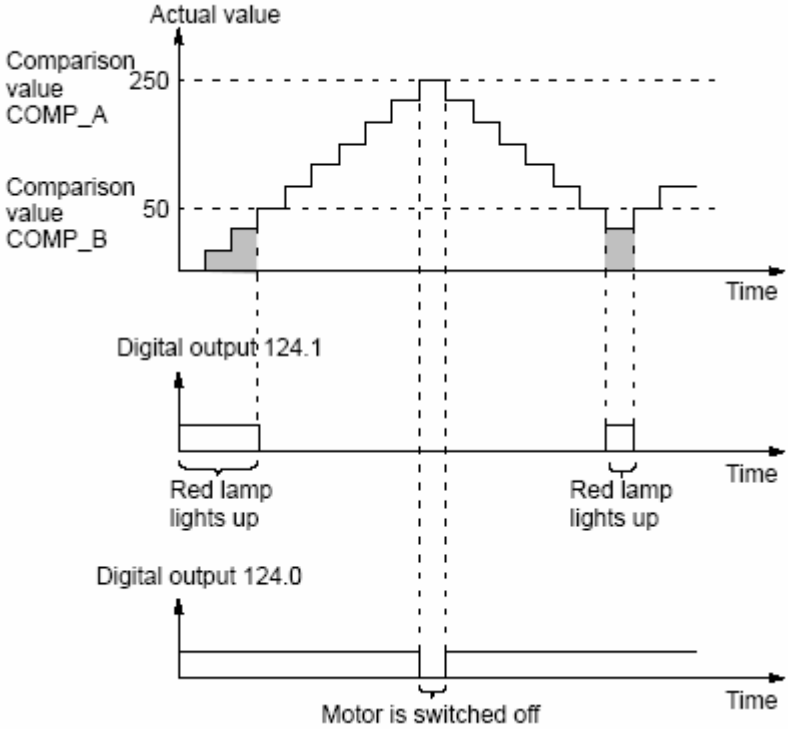



علاوه بر ورودی و خروجی های قبلی در اینجا موارد زیر را داریم:

**I 124.7** : این ورودی که به سنسور خروجی متصل است برای کاهش شمارنده بکار می رود.

**Q 124.1** : این خروجی به لامپ سیگنال متصل است و توسط مقایسه گر B در SFB29 کنترل می گردد.

عملکرد شمارنده برای اینحالت در شکل صفحه بعد ترسیم شده و بدنبال آن برنامه OB100 و OB1 به زبان STL ارائه گردیده است.




روش کار در OB100 مشابه آنچه برای برنامه ۱ ذکر شد می باشد با این تفاوت که در اینجا مقادیر مبنای مقایسه برای مقایسه گر A مقدار ۲۵۰ و برای مقایسه گر B مقدار ۵۰ داده میشود. توجه شود با تخصیص مقادیر فوق به ورودی های SFB29 در OB100 لزومی به تخصیص دوباره در OB1 نیست زیرا این مقادیر در OB100 هنگام فراخوانی فانکشن در دیتا بلاک DB63 ذخیره میشوند و چون OB1 نیز فانکشن را با همین DB صدا میزند مقادیر فوق در OB1 نیز قابل دسترس خواهند بود.

سایر ورودیهای فانکشن مشابه قبل ریست شده و لامپ سیگنال Q124.1 نیز به وضعیت خاموش در می آید.

**OB 100**

Network 1

```
L     L#0
T     MD 0      Define start value PRES_COUNT in MD 0
L     L#250     Define new comparison value PRES_COMP_A
T     MD 4      in MD 4
L     L#50      Define new comparison value PRES_COMP_B
T     MD 8      in MD 8
SET             Enable execution of SFB 29
=     M 26.2
A     M 26.2    If M 26.2 = 1, i.e. EN = 1 at SFB 29, then SFB is executed;
```

```
        JNB          m01            If RLO = 0, jump to m01
       CALL            SFB 29, DB 63
     PRES_COUNT:    = MD 0         Assignment of input parameters
    PRES_COMP_A:    = MD 4
    PRES_COMP_B:    = MD 8
       EN_COUNT:    = FALSE        Counter not yet enabled
          EN_DO:    = FALSE        Digital outputs are not enabled for Counter integrated function
      SET_COUNT:    = FALSE        SET_COUNT = 0, to generate pos. edge in OB 1
     SET_COMP_A:    = FALSE        SET_COMP_A = 0, to generate pos. edge in OB 1
     SET_COMP_B:    = FALSE        SET_COMP_B = 0, to generate pos. edge in OB 1
          COUNT:    = MD 14        Assignment of output parameters
         COMP_A:    = MD 18
         COMP_B:    = MD 22
       STATUS_A:    = M 26.0
       STATUS_B:    = M 26.1
m01: A              BR             Query BR bit (= ENO at SFB 29) for er-
       =            M 26.3         ror evaluation
       AN           M 26.1         Fulfill start condition, i.e. red lamp
       =            Q 124.1        lights up
       AN           M 26.0         Conveyor belt on if comparison value
       =            Q 124.0        COMP_A not yet reached
```

در OB1 روش کار مشابه آنچه در برنامه ۱ ذکر شد میباشد با این تفاوت که در اینجا لازم است هر دو مقایسه گر A و B فعال شوند.

**OB 1**

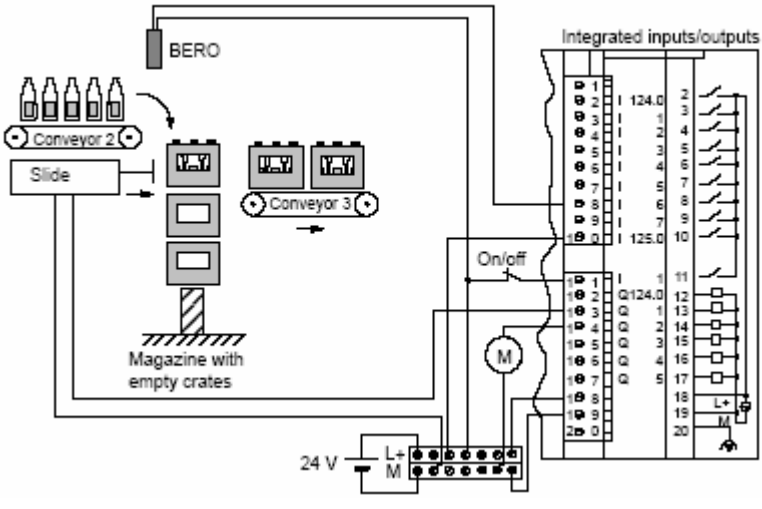
Network 1

```
          A              M 26.3          If M 26.3 = 1, SFB is executed;
        JNB              m01             If RLO = 0, jump to m01
       CALL                SFB 29, DB 63
     PRES_COUNT:        =
    PRES_COMP_A:        =
    PRES_COMP_B:        =
       EN_COUNT:        = I 125.1       The counting process can be interrupted
                                        by activating the normally-closed switch
          EN_DO:        = TRUE          Digital outputs are enabled for Counter
                                        integrated function
      SET_COUNT:        = TRUE          Start value PRES_COUNT is transferred
     SET_COMP_A:        = TRUE          Comparison value PRES_COMP_A is transferred

     SET_COMP_B:        = TRUE          Comparison value PRES_COMP_B is transferred
          COUNT:        = MD 14         Assignment of output parameters
         COMP_A:        = MD 18
         COMP_B:        = MD 22
       STATUS_A:        = M 26.0
       STATUS_B:        =
m01: A                  BR              Query BR bit (= ENO at SFB 29) for er ror evaluation
       =                M 26.3
```

**برنامه ۳ : کاربرد CPU312 IFM همراه با SFB29 برای شمارش (تکمیل برنامه ۱ و ۲ )**

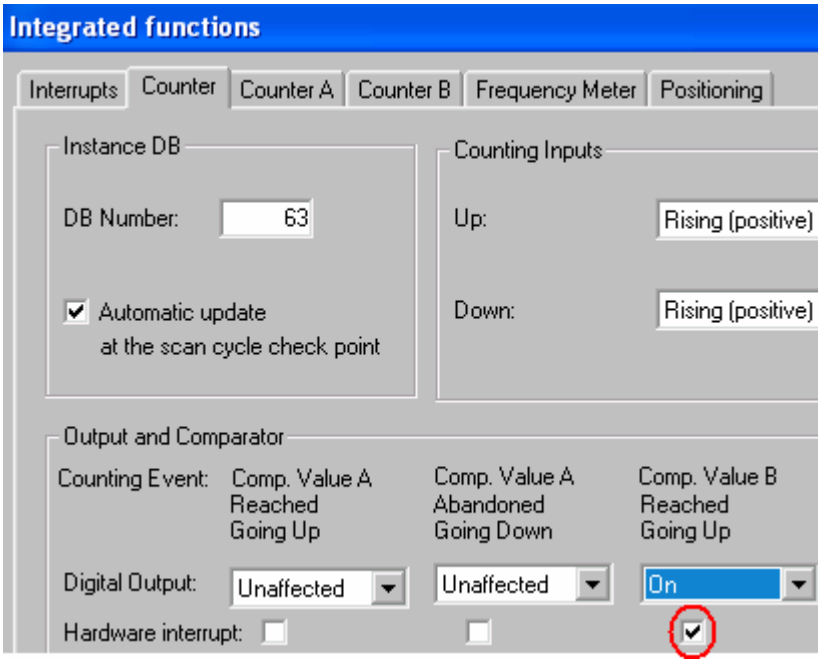
در این مثال که تکمیل کننده دو مثال قبلی است یک CPU 312 IFM دیگر کنترل گذاشتن بطری ها در جعبه ها را بعهده دارد. بطری ها از بافر قبلی به روی نوار نقاله شماره ۲ قرار گرفته و از آنجا به داخل جعبه های خالی که هر کدام ظرفیت ۶ بطری را دارند گذاشته می شوند.پس از اینکه تعدادبطری ها به ۶ عدد رسید نوار ۲ خاموش شده و هل دهنده Slide استارت شده و زمان به اندازه ۵ ثانیه منظور می گردد در طول این ۵ ثانیه هل دهنده جعبه پر را به روی نوار ۳ انتقال داده و پس از سپری شدن ۵ ثانیه هل دهنده به موقعیت اولیه خود برگشته و نوار ۲ استارت می شود. فرآیند به همین نحو برای جعبه های دیگر تکرار می گردد.

اپراتور توسط سوئیچ نرمال بسته میتواند فرآیند شمارش را در موقع بروز خطا یا زمان راه اندازی نوار ۲ متوقف کند.



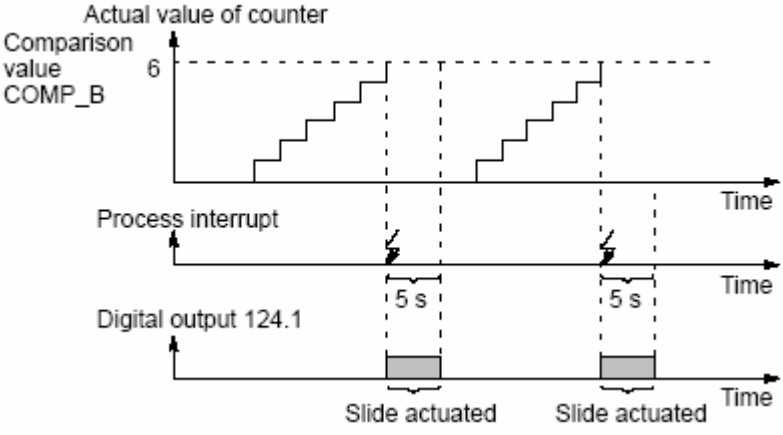

در اینجا I 124.6 به سنسور متصل است و برای شمارش افزایشی بکار می رود. I 125.0 مستقیماً به ۲۴ ولت متصل شده چون نیازی به معکوس شدن عملکرد شمارش نمی باشد. I 125.1 به سوئیچ نرمال بسته متصل است و اپراتور از طریق آن میتواند شمارش را متوقف سازد. Q124.1 که از مقایسه گر فانکشن می آید برای کنترل تعداد بطری داخل جعبه است وقتی تعداد به ۶ برسد این خروجی فعال شده و زمان گیری ۵ ثانیه شروع می شود. خروجی Q124.2 برای کنترل موتور نوار نقاله ۲ بکار می رود. در اینجا برای Set کردن و روشن کردن

مداوم تایمر با زمان ۵ ثانیه از وقفه سخت افزاری استفاده شده است این وقفه در پارامترهای CPU در بخش Integrated Function برای مقایسه گر B فعال شده است .




به محض اینکه شرایط مقایسه یعنی رسیدن تعداد بطری ها به ۶ برآورده شود.این وقفه فعال شده و سیستم عامل OB40 را صدا زده و برنامه آن را اجرا می کند. در برنامه OB40 تایمر با زمان ۵ ثانیه راه اندازی می شود.

عملکرد مقایسه گر در شکل زیر و برنامه STL در صفحه بعد آمده است.



با توضیحاتی که برای دو برنامه قبلی داده شد خواننده محترم با برنامه OB100 آشنا گردیده و نیازی به تکرار آنها نیست . در اینجا مقدار 0 به عنوان مقدار اولیه و مقدار 6 بعنوان مقدار مبنای مقایسه (به مقایسه گر B) داده میشود.

**OB 100**

```
Network 1
        L          L#0          Define start value PRES_COUNT in MD 0
        T          MD 0

        L          L#6          Define new comparison value PRES_COMP_B
        T          MD 8         in MD 8
      SET                       Enable execution of SFB 29
        =          M 26.2

        A          M 26.2       If M 26.2 = 1, i.e. EN = 1 at SFB 29,
                                then SFB is executed;
      JNB          m01          If RLO = 0, jump to m01

     CALL          SFB 29, DB 63
  PRES_COUNT:      = MD 0       Assignment of input parameters
 PRES_COMP_A:      =
 PRES_COMP_B:      = MD 8
    EN_COUNT:      = FALSE      Counter not yet enabled
       EN_DO:      = TRUE       Digital outputs are enabled for Counter integrated function

   SET_COUNT:      = FALSE      SET_COUNT = 0, to generate pos. edge in OB 1
  SET_COMP_A:      =
  SET_COMP_B:      = FALSE      SET_COMP_B = 0, to generate pos. edge in OB 1
       COUNT:      = MD 14      Assignment of output parameters
      COMP_A:      =
      COMP_B:      = MD 22
    STATUS_A:      =
    STATUS_B:      =

m01: A             BR           Query BR bit (= ENO at SFB 29) for
        =          M 26.3       error evaluation
```

در OB1 علاوه بر توضیحات مشابهی که قبلاً داده شد لازم است توجه شود که اولاً فقط مقایسه گر B فعال میگردد ثانیاً ازوضعیت تایمر T0 که توسط OB40 راه اندازی میشود در OB1برای کنترل اسلاید و نوارنقاله ۲ استفاده میگردد. وقتی تایمر خاموش شد یعنی ۵ ثانیه سپری گردید اسلاید توسط Q124.1 از کار می افتد . در طول ۵ ثانیه ای که تایمر کار میکند اسلاید فعال و نوار نقاله ۲ خاموش است.

**OB 1**

```
Network 1
        SET                     Motor for conveyor 2 is switched on
         S          I 124.2

         A          M 26.2      If M 26.2 = 1, i.e. EN = 1 at SFB 29,
                                then SFB is executed
       JNB          m01         If RLO = 0, jump to m01

      CALL          SFB 29, DB 63
  PRES_COUNT:       =
 PRES_COMP_A:       =
 PRES_COMP_B:       =
    EN_COUNT:       = I 125.1   The counting process can be interrupted
                                by activating the normally-closed switch
       EN_DO:       =
   SET_COUNT:       = TRUE      Counter is set at first OB 1 pass

  SET_COMP_A:       =
  SET_COMP_B:       = TRUE      Comparison value PRES_COMP_P is set at first OB 1 pass
       COUNT:       = MD 14     Assignment of output parameters
      COMP_A:       =
      COMP_B:       = MD 22
    STATUS_A:       =
    STATUS_B:       =

m01: A              BR          Query BR bit (= ENO at SFB 29) for error evaluation
         =          M 26.3

        AN          T 0         When the time of 5 s has expired, the
         R          I 124.1     slide is no longer actuated.
         A          T 0         As long as the time of 5 s is running,
         R          I 124.2     the motor for conveyor 2 is switched
        AN          T 0         off and at the same time the slide is
        FR          T 0         triggered by the integrated function (Q 124.1)
```

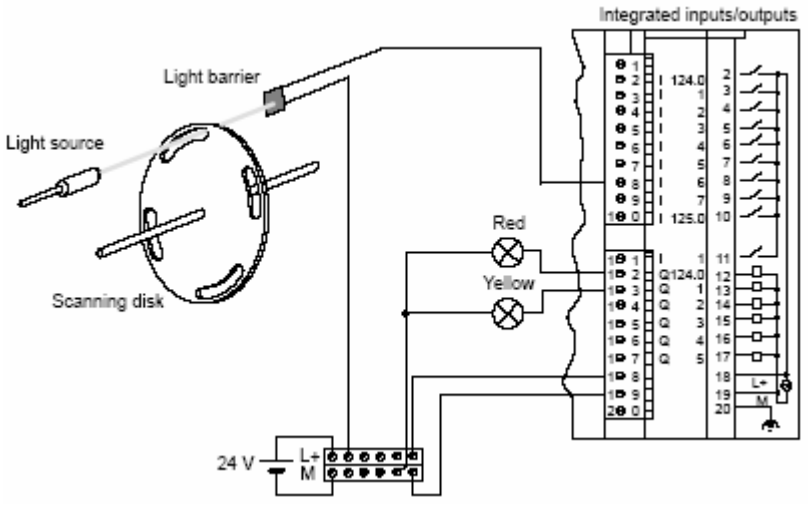
OB40 بلاکی است که در هنگام وقوع وقفه سخت افزاری توسط سیستم عامل صدا زده میشود. همانطور که دیدیم این وقفه در پارامترهای سخت افزار در حالتی که مقدار شمارش به حد بالا برسد فعال گردید. در برنامه OB40 تایمر T0 بصورت Extended Pulse با زمان ۵ ثانیه تنظیم و راه اندازی میشود.

**OB 40**

```
Network 1
        AN          T 0
         L          S5T#5S      Start timer T 0 for 5 s
        SV          T 0
```

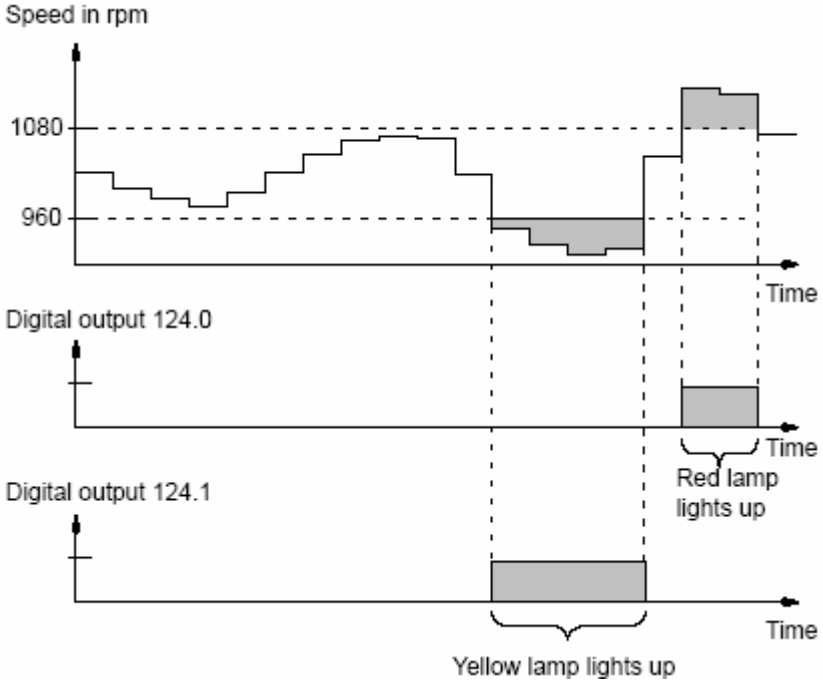
**برنامه ٤ : کاربرد CPU312 IFM همراه با SFB30 برای اندازه گیری سرعت شافت**

شافت یک موتور با سرعت نسبتاً ثابت می چرخد . برای اندازه گیری سرعت از یک Light Barrier استفاده می شود که نور آن از صفحه ای که همراه با شافت میچرخد و دارای شکافهای متقارن است عبور کرده و به سنسور نوری که در سمت دیگر صفحه قرار دارد برخورد می کند. سنسور مزبور به ورودی I 124.6 از یک CPU312 IFM متصل است . سوراخ های صفحه کاملا متقارن است بعلاوه فضای قسمت خالی با فضای قسمت پر کنار آن برابر است مانند اینکه دایره را به ۸ قسمت مساوی تقسیم کرده و یک در میان شکاف هایی ایجاد کرده باشیم. با توجه به تعداد سوراخ ها فرکانس واقعی شافت یک چهارم فرکانس پالس های سنسور نوری است.




سرعت شافت در حالت نرمال بین ۹۶۰ تا ۱۰۸۰ دور در دقیقه rpm قابل قبول است میخواهیم اگر سرعت از 1080 rpm بالاتر رفت لامپ قرمز رنگ که به خروجی Q124.0 متصل است روشن و اگر سرعت از 960rpm پایین تر آمد لامپ زرد رنگ متصل به خروجی Q124.1 روشن گردد.

عملکرد فانکشن SFB30 برای این نیاز کنترلی در صفحه بعد ترسیم شده است . در پارامترهای سخت افزاری زمان نمونه برداری را 10 s انتخاب کرده سپس در برنامه نویسی SFB30 را همراه با DB62 (که پیش فرض سیستم و در عین حال در پارامترهای HWconfig قابل تغییر است ) در OB100 و OB1 صدا میزنیم.




توجه شود که فرکانس مبنای مقایسه از دیدگاه برنامه فرکانس پالسهای سنسور نوری است که ۴ برابر فرکانس شافت است از اینرو لازم است ابتدا سرعتهای rpm مورد نظر را مطابق جدول زیر به فرکانس مبنا تبدیل کنیم.

| Comparison Value | Speed | Frequency at a Configured Sample Time of 10 s | | | Upper/Lower Limit |
|---|---|---|---|---|---|
| Upper limit | 1080 rpm | 1080/ 60 | = | 18 Hz | 18 Hz 4 (light slots) = 72 Hz =72000 mHz |
| Lower limit | 960 rpm | 960/ 60 | = | 16Hz | 16 Hz 4 (light slots) = 64 Hz =64000 mHz |

فرکانس حاصل برای استفاده در SFB30 لازم است بصورت mHZ تبدیل گردد. بلاک های برنامه نویسی OB1 و OB100 در صفحات بعد آورده شده اند . همانطور که در OB100 مشاهده می شود فرکانسهای فوق در داخل متغیر های حافظه ذخیره شده و این متغیرها بعنوان ورودی SFB30 در OB1 استفاده گردیده اند.

در OB100 فانکشن SFB30 را صدا زده و مقادیر بالا و پایین یعنی 64000 و 72000 را به ورودی های آن میدهیم ولی سایر ورودی ها را Reset می نماییم. بعلاوه با بررسی بیت BR شرایط را برای اجرا در OB1 آماده می سازیم.

**OB 100**

```
Network 1
          L          L#64000      Define comparison value PRES_L_LIMIT in
          T          MD 4         MD 4 (monitoring possible with STATUS VAR)
        SET                       Enable SFB 30 execution
          =          M 24.0
          A          M 24.0       If M 24.0 = 1, i.e. EN = 1 at SFB 30, SFB is executed;
        JNB          m01          If RLO = 0, jump to m01
       CALL          SFB 30, DB 62
    PRES_U_LIMIT:    = L#72000    Define comparison value PRES_U_LIMIT
    PRES_L_LIMIT:    = MD 4       Assign to MD 4
     SET_U_LIMIT:    = FALSE      SET_U_LIMIT = 0, to generate pos. edge in OB 1
     SET_L_LIMIT:    = FALSE      SET_L_LIMIT = 0, to generate pos. edge in OB 1
            FREQ:    =
          U_LIMIT:   =
          L_LIMIT:   =
        STATUS_U:    =
        STATUS_L:    =
m01: A               BR           Query BR bit (= ENO at SFB 30) for error evaluation
          =          M 24.1
```

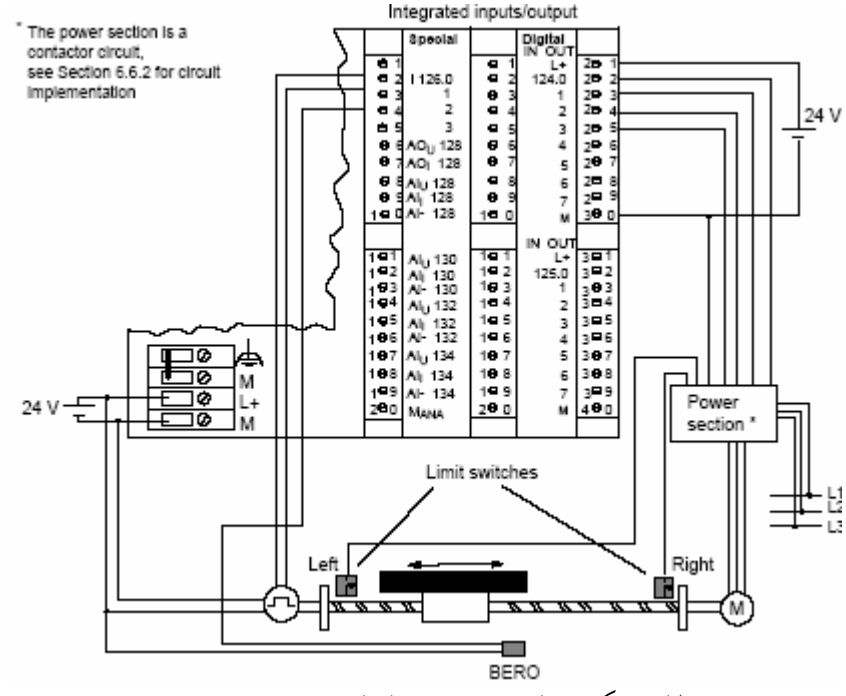
در OB1 ورودی های Set_U_Limit و Set_L_Limit همانطور که می بینیم True شده اند تا حدود بالا و پایین که مقادیر آنها در OB100 داده شده فعال گردند. مقدار فرکانس خروجی روی MD8 ریخته شده سپس با تقسیم بر 4000 و ضرب در 60 مقدار سرعت شافت بر حسب rpm محاسبه و در MD12 ریخته شده است.

**OB 1**

```
Network 1
          A          M 24.1       If M 24.1 = 1, i.e. EN = 1 at SFB 30, SFB is executed
        JNB          m01          If RLO = 0, jump to m01
       CALL          SFB 30, DB 62 Call SFB 30 with instance DB
    PRES_U_LIMIT:    =
    PRES_L_LIMIT:    =
     SET_U_LIMIT:    = TRUE       Set comparison values with pos. edge
     SET_L_LIMIT:    = TRUE       Current measured frequency is stored in
            FREQ:    = MD 8       MD 8
          U_LIMIT:   =
          L_LIMIT:   =
        STATUS_U:    = Q 124.0    If Q 124.0 = 1, red lamp lights up
        STATUS_L:    = Q 124.1    If Q 124.1 = 1, yellow lamp lights up
m01: A               BR           Query BR bit (= ENO at SFB 30) for er ror evaluation
          =          M 24.1
          L          MD 8         End if valid speed value has not been read
          L          L# -1
        ==D
        BEC
          L          MD 8         Convert measured frequency to actual shaft speed
          L          4000
         /D
          L          L#60
         *D
          T          MD 12        Speed is stored in MD 12 in decimal format in 1/min
```

## برنامه ۵ :کاربرد CPU314 IFM همراه با SFB39 برای کنترل موقعیت در سیستمی با تغذیه کنتاکتوری

در این مثال عملیات ماشین کاری روی قطعه کار که بر روی میز متحرک قرار گرفته انجام می شود. میز کار حرکت می کند تا به موقعیت مناسب برسد سپس متوقف شده تا عملیات ماشین کاری روی قطعه انجام شود. این عملیات در ۳ نقطه انجام می گردد.




با روشن شدن سیستم عملیات سنکرون سازی بصورت زیر انجام می شود:

بدون توجه به موقعیت فعلی ، میز کار بصورت Forward حرکت می کند تا به لیمیت سوئیچ سمت چپ برسد. در اینحالت موتور خاموش می شود. سپس موتور بصورت Backward روشن شده و حرکت می کند تا به سوئیچ رفرنس برسد. با آشکار شدن لبه مثبت توسط سوئیچ رفرنس عملیات سنکرون سازی انجام می شود و موتور مجدداً خاموش می شود.پس از مراحل فوق عملیات Positioning توسط برنامه شروع می گردد. در این مرحله میز کار بصورت Forward حرکت می کند تا به ۳ نقطه ای که برای ماشین کاری در نظر گرفته شده برسد در این نقاط موتور خاموش می شود. پس از اتمام عملیات ماشین کاری اپراتور قطعه کار را برداشته و قطعه جدید را روی میز قرار می دهد. مجدداً Positioning جدید توسط برنامه بصورت اتومات انجام می گیرد.

همانطور که قبلاً اشاره شد ورودیهای I126.0 و I126.1برای انکودر و I126.3 برای سوئیچ رفرنس است. Q124.0 سرعت کم و Q124.1 سرعت زیاد را فعال می کند. Q124.2 برای حرکت Backward و Q124.3 برای حرکت Forward بکار می رود.

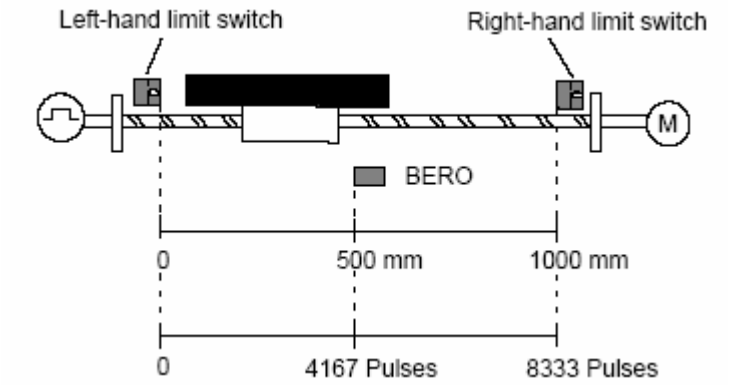
در این مثال انکودر ۲۵۰ پالس در هر دور میفرستد. هر دور انکودر معادل ۱۰ دور موتور است. بنابراین برای هر دور موتور انکودر ۲۵ پالس می فرستد. میز کار با هر دور موتور 3 mm حرکت می کند پس هر پالس انکودر طبق رابطه زیر برابر 0.12 mm خواهد بود .

3mm / 25 pulse = 0.12 mm

مقدار فوق که معادل یک پالس است را اصطلاحاً فاصله Increment می گویند.

در این مثال در هر عملیات Positioning لازم است مجدداً نقطه مبنا بررسی شود. این نقطه در وسط میز قرار گرفته است . طول کل تراورس ۱ متر و مبنا در 0.5 متری واقع شده است. با توجه به این اعداد معادل Increment آنها بصورت زیر خواهد بود:

500 mm / 0.12 mm = 4176 pulse
1000mm/ 0.12mm = 8333 pulse

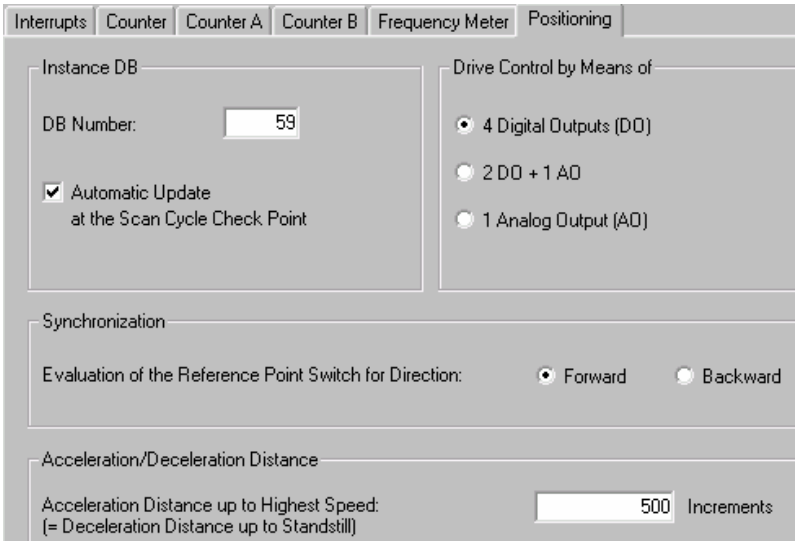



در این مثال میز کار لازم است در سه نقطه برای عملیات ماشین کاری بایستد . این نقاط همراه با معادل Increment آنها در جدول زیر آورده شده اند:

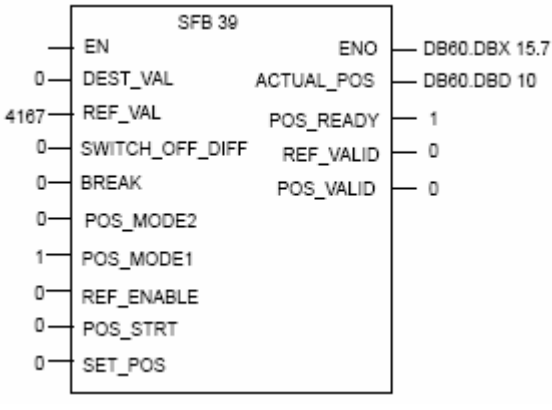
| Destination Position | Conversion for Specifying to SFB 39 |
|---|---|
| 750 mm | 750 mm : 0.12 mm per pulse = **6250**pulses (distance increments) |
| 400 mm | 400 mm : 0.12 mm per pulse = **3333**pulses (distance increments) |
| 100 mm | 100 mm : 0.12 mm per pulse = **833**pulses (distance increments) |

پارامتر دیگری که لازم است مشخص گردد فاصله توقف است . منظور از فاصله توقف فاصله بین سرعت Creep تا نقطه توقف می باشد. این فاصله 60 mm در نظر گرفته شده که معادل ۵۰۰ پالس است.

60mm / 0.12mm = 500 pulse




DB که لازم است همراه با SFB39 بکار رود طبق تنظیم فوق DB59 می باشد که این شماره پیش فرض سیستم است در عین حال قابل تغییر می باشد. مقادیر اولیه ازجمله نقطه رفرنس و سه نقطه ماشین کاری در دیتا بلاک دیگری DB60 ریخته شده که در برنامه از آن استفاده می گردد. این DB همراه با برنامه OB1 در صفحات بعد آمده است.



**DB 60**

| Address | Name | Type | Starting value | Comment |
|---|---|---|---|---|
| 0.0 | | STRUCT | | |
| +0.0 | DEST_VAL | DINT | L#0 | Always contains the currently valid destination position for drive (SW1, SW2 or SW3) |
| +4.0 | REF_VAL | DINT | L#4176 | Reference point for BERO = 500 mm |
| +8.0 | SWITCH_OFF_DIFF | INT | 0 | Switch-off difference (calculated at startup) |
| +10.0 | ACTUAL_POS | DINT | L#0 | Output: Current actual value |
| +14.0 | Control byte | BYTE | B#16#0 | Control bits for positioning |
| +15.0 | Checkback byte | BYTE | B#16#0 | Checkback status bits from position-ing |
| +16.0 | Istw1 | DINT | L#0 | Old actual value |
| +20.0 | Sw1 | DINT | L#6250 | Destination position for 1st machin-ing step (750 mm) |
| +24.0 | Sw2 | DINT | L#3333 | Destination position for 2nd machin-ing step (400 mm) |
| +28.0 | Sw3 | DINT | L#833 | Destination position for 3rd machin-ing step (100 mm) |
| +32.0 | SK1 | WORD | W#16#0 | Auxiliary marker for sequencer |
| +34.0 | SK2 | WORD | W#16#0 | Counter for jump-to list |
| =36.0 | | END_STRUCT | | |

**OB 1**

Network 1

Call positioning

```
CALL SFB 39 , DB59
        DEST_VAL :=DB60.DBD0      Destination position for drive
         REF_VAL :=DB60.DBD4      Reference point for BERO
SWITCH_OFF_DIFF:=DB60.DBW8        Switch-off difference
          BREAK :=                Unassigned means default value applies (0)
      POS_MODE2 :=DB60.DBX14.0    Jog mode forward
      POS_MODE1 :=DB60.DBX14.1    Jog mode backward
     REF_ENABLE :=DB60.DBX14.2    Control signal: Evaluate reference signal
       POS_STRT :=DB60.DBX14.3    Start positioning operation
        SET_POS :=
```

```
                ACTUAL_POS :=DB60.DBD10     Output: Current actual value
               POS_READY :=DB60.DBX15.0     Checkback signal: Pos. op./jog mode terminated
                REF_VALID :=DB60.DBX15.1    Checkback signal: Reference point switch reached
                POS_VALID :=DB60.DBX15.2    Checkbacksignal:Synchronization hastakenplace
A BR                                        Scanning the BR bits (= ENO at SFB 39) for
                = DB60.DBX 15.7             error evaluation
Checking that drive is at standstill
                L S5T#200MS                 Scan for drive at standstill
AN T 1                                      If no change in position within 200 ms,
SD T 1                                      then drive at standstill
JC m1
                L DB60.DBD 16               Save old actual value and
                L DB60.DBD 10               current actual value
                T DB60.DBD 16               for next comparison
==D
                = DB60.DBX 14.4             Memory bit for drive at standstill
m1: NOP 0
                A DB60.DBX 14.5             Memory bit for processing set
JC m13
Switching the system on
                A I 0.0                     Momentary-contact switch “Setup”
               FP DB60.DBX 32.1             Edge evaluation for momentary-contact switch
                AN I 0.1                    Interlock with automatic mode
               AN DB60.DBX 32.2
               S DB60.DBX 32.0              Auxiliary marker for “Setup” sequencer
               AN DB60.DBX 32.0             Jump if not “Setup”
JC m8
                L DB60.DBW 34               Counter for jump-to list
JL m2                                       Call jump-to list
JU m3                                       Traverse to left limit switch
JU m4                                       Switch off axis
JU m5                                       Start forward to right limit switch
JU m6                                       Switch off axis
JU m7                                       Terminate setup
m2: L 0
                T DB60.DBW 34
BEU
m3:     NOP     0
        AN      DB60.DBX    14.1
        S       DB60.DBX    14.1            Jog mode backward
        A       DB60.DBX    14.4            Axis still at standstill
        BEC
        L       1                           Next step
        T       DB60.DBW    34
        BEU
m4:     NOP     0
        AN      DB60.DBX    15.0            If positioning not terminated
        A       DB60.DBX    14.4            and drive at standstill,
        S       DB60.DBX    14.1            then switch axis to Stop
        S       DB60.DBX    14.0
        ON      DB60.DBX    14.1            Wait for axis stop
        ON      DB60.DBX    14.0
        BEC
        L       2                           Next step
```

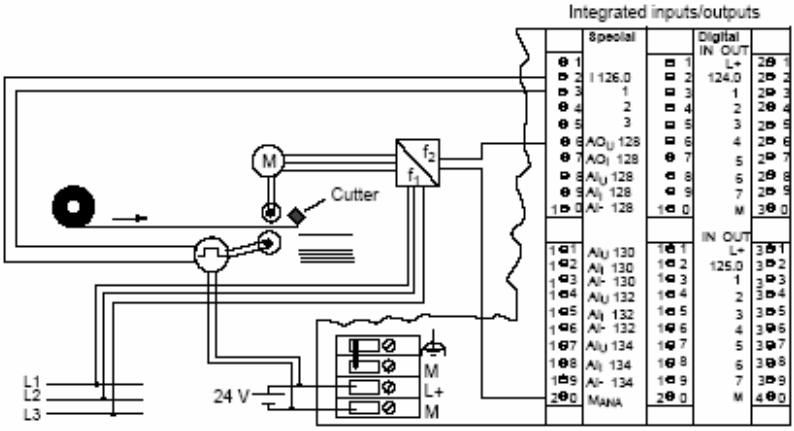
```
        T     DB60.DBW   34
       BEU
m5:    NOP   0
        A     DB60.DBX   14.0     If Stop signal active,
        A     DB60.DBX   14.1
        R     DB60.DBX   14.1     then reset Stop signal
        R     DB60.DBX   14.0
       BEC
       SET
        S     DB60.DBX   14.0     Jog mode forward
        S     DB60.DBX   14.2     Set control signal: REF_ENABLE
        A     DB60.DBX   14.4     Axis still at standstill
       BEC
        L     3                   Next step
        T     DB60.DBW   34
       BEU
m6:    NOP   0
       AN     DB60.DBX   15.0     Positioning not terminated
        A     DB60.DBX   14.4     Drive at standstill
        S     DB60.DBX   14.1     Switch axis to stop
        S     DB60.DBX   14.0
       ON     DB60.DBX   14.0     Wait for axis stop
       ON     DB60.DBX   14.1
       BEC
        L     4                   Next step
        T     DB60.DBW   34
       BEU
m7:    NOP   0                   Terminate setup
       SET
        R     DB60.DBX   14.1     Reset Stop signal
        R     DB60.DBX   14.0     (terminate setup)
        R     DB60.DBX   32.0
        L     0                   Reset counter for jump-to list
        T     DB60.DBW   34
       BEU
m8:    NOP   0
Automatic mode
                A I 0.1           Momentary-contact switch for Automatic
            FP DB60.DBX   32.3    Edge evaluation for momentary-contact switch
                AN I 0.0          Interlock with “Setup”
           AN DB60.DBX   32.0
             S DB60.DBX   32.2    Set memory bit for “Automatic” sequencer
           AN DB60.DBX   32.2    End if not “Automatic”
          BEC
             L DB60.DBW   34      Counter for jump-to list
         JL m9                    Call jump-to list
        JU m10                    Load 1st destination position
        JU m11                    Load 2nd destination position
        JU m12                    Load 3rd destination position
m9:      L 0
             T DB60.DBW   34
          BEU
m10:     NOP 0
             L DB60.DBD   20      Load destination position for 1stmachiningstep
```

```
            T DB60.DBD    0          Save it as destination position for drive
           AN DB60.DBX    14.3       Start positioning operation
            S DB60.DBX    14.3
          BEC
           ON DB60.DBX    15.0       If positioning operation not yet terminated
           ON DB60.DBX    14.4       or drive running
          BEC
          L 1                       Next step
            T DB60.DBW    34
          SET
            R DB60.DBX    14.3       Reset control signal for start positioning
                                     operation
            S DB60.DBX    14.5       Start machining
          BEU
m11:      NOP 0
            L DB60.DBD    24         Loaddestination position for2ndmachining step
            T DB60.DBD    0          Save it as destination position for drive
           AN DB60.DBX    14.3       Start positioning operation
            S DB60.DBX    14.3
          BEC
           ON DB60.DBX    15.0       If positioning operation not yet
           ON DB60.DBX    14.4       terminated or drive running
          BEC
          L 2                       Next step
            T DB60.DBW    34
          SET
            R DB60.DBX    14.3       Reset control signal for start positioning
                                     operation
            S DB60.DBX    14.5       Start machining
          BEU
m12:  NOP     0
        L     DB60.DBD    28         Loaddestination position for3rdmachining step
        T     DB60.DBD    0          Save it as destination position for drive
       AN     DB60.DBX    14.3       Start positioning operation
        S     DB60.DBX    14.3
      BEC
       ON     DB60.DBX    15.0       If positioning operation not yet terminated
       ON     DB60.DBX    14.4       or drive running
      BEC
        L     0                      Next step
        T     DB60.DBW    34
      SET
        R     DB60.DBX    14.3       Reset control signal for start positioning
                                     operation
        S     DB60.DBX    14.5       Start machining
        R     DB60.DBX    32.2       Terminate automatic mode
      BEU
Machining
m13:  NOP     0                      Simulation of machining via waiting time
        A         T 2
        R     DB60.DBX    14.5       Terminate machining
        L     S5T#2S
        A     DB60.DBX    14.5
       SD         T 2
```

**برنامه ٦ :کاربرد CPU314 IFM همراه با SFB39 برای کنترل برش در سیستم مبتنی بر درایو**

در فرآیندی مانند شکل زیر لازمست فویل کلاف به طولهای ۲ متری بریده شود. انکودر Incremental فاصله لبه فویل را آشکار میسازد. فویل برای عملیات برش متوقف شده و عمل برش انجام می شود. وقتی رول جدیدی نصب می شود یا وقتی که عملیات برش با خطا مواجه می شود لازم است Jog Mode انجام پذیرد.

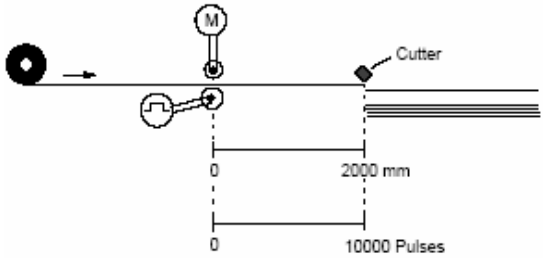
باز شدن رول توسط برنامه انجام می شود و وقتی برش انجام شد نقطه رفرنس مجدداً به فانکشن IFM بعنوان مقدار جدید داده می شود. خروجی آنالوگ که به درایو داده شده از نوع ±10V است این خروجی به آدرس PQW128 و انکودر به ورودی های I126.0 و I126.1 متصل می باشد.




در این مثال انکودر در هر دور ۱۰۰ پالس میفرستد. هر دور انکودر معادل ۵ دور موتور است. بنابر این برای هر دور موتور انکودر ۲۰ پالس ارسال می دارد. از آنجا که با هر دور موتور فویل 4 mm باز می شود بنابراین هر پالس معادل 0.2 mm است که به این فاصله Increment می گوییم. (4 mm / 20 pulse = 0.2 mm)

طول برش 2 متر است که معادل 10,000 پالس خواهد بود. ( 2000 mm / 0.2 mm = 10,000 pulse )

عدد 10,000 بعنوان DEST_VAL به ورودی فانکشن SFB39 داده می ود.



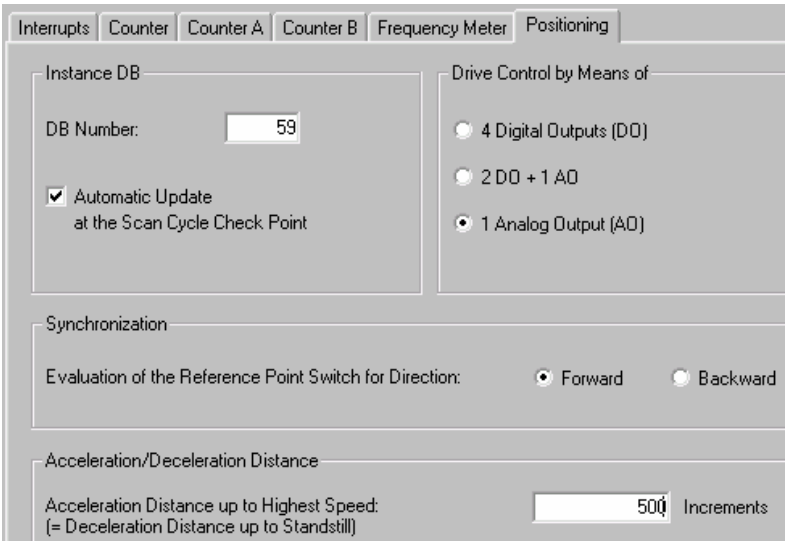
### ماکزیمم سرعت

فرض شده جنس فویل به گونه ای است که میتواند با ماکزیمم سرعت باز شود . بیشترین سرعت با مقدار آنالوگ v= 10 V حاصل می شود بنابراین مقدار Break از رابطه زیر صفر بدست می آید . این مقدار به ورودی فانکشن SFB39 داده می شود.

$$v = \frac{10\,V}{256} \times (256 - BREAK) \text{ or } BREAK = 256 \times (1 - \frac{v}{10\,V})$$

### ماکزیمم شتاب

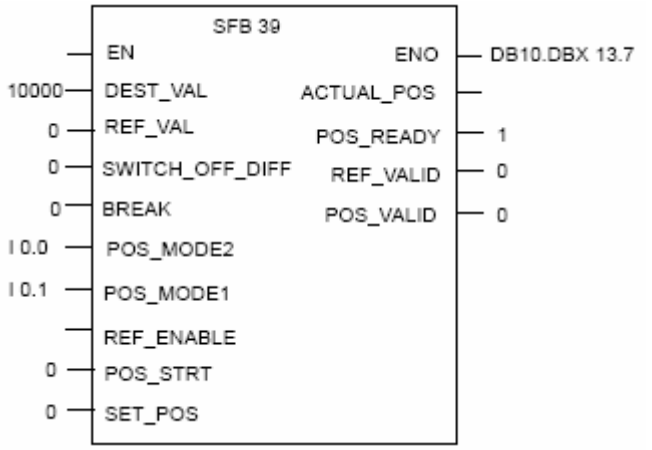
با فرض اینکه ماکزیمم سرعت لازمست در نقطه 0.1 متری اتفاق بیفتد داریم 100 mm / 0.2 mm = 500 pulse مقدار 500 در پارامترهای سخت افزار Hwconfig وارد می شود.




### رسیدن به نقطه نهایی (Switch_Off_Diff)

این فاصله ای است که فویل پس از قطع شدن موتور طی می کند و بصورت یک ورودی به SFB39 داده می شود.برای محاسبه دقیق لازمست ابتدا این مقدار صفر داده شود سپس یک بار روی فویل عمل برش انجام شود. میزان اختلاف آن با طول ۲ متر را اندازه گرفته با تقسیم بر 0.2 mm به پالس تبدیل کنید و عدد نتیجه را به ورودی SFB39 بدهید.

شکل کلی بلاک همراه با برنامه OB1 در صفحه بعد آمده است . مقادیر اولیه در DB10 تعریف شده اند و SFB39 همراه با DB59 صدا زده شده است.




**DB10**

| Address | Name | Type | Starting value | Comment |
|---|---|---|---|---|
| 0.0 | | STRUCT | | |
| +0.0 | DEST_VAL | DINT | L#10000 | Destination position: Length of the foil = 2 m |
| +4.0 | REF_VAL | DINT | L#0 | Reference point = 0 |
| +8.0 | SWITCH_OFF_D IF F | INT | 0 | Switch-off difference (calculated at startup) |
| +10.0 | Break | BYTE | B#16#0 | Maximum velocity = 10 V |
| +11.0 | --- | BYTE | B#16#0 | Unused |
| +12.0 | Control byte | BYTE | B#16#0 | Control bits for positioning |
| +13.0 | Checkback byte | BYTE | B#16#0 | Checkback status bits from positioning |
| =14.0 | | END_STR UCT | | |

**OB1**

Network 1
Call positioning

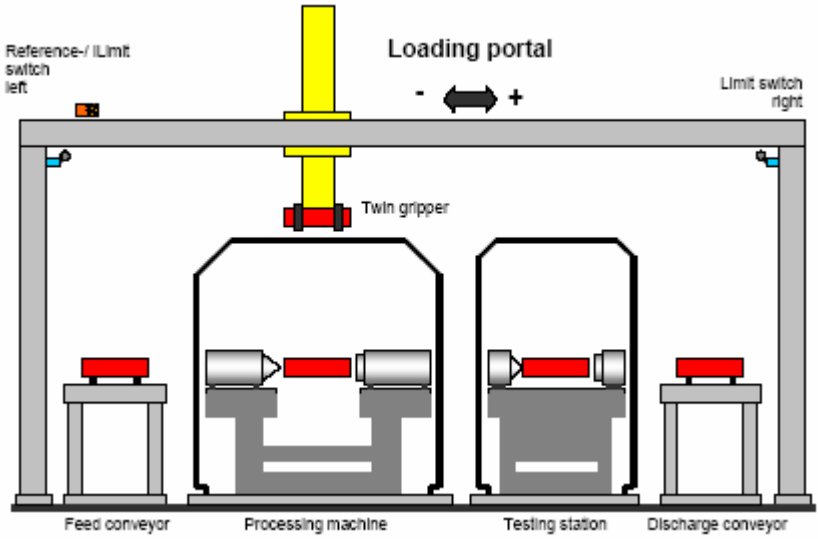
```
CALL SFB 39 , DB59
        DEST_VAL :=DB10.DBD0      Destination position (foil length = 2 m)
         REF_VAL :=DB10.DBD4      Reference point (foil start)
SWITCH_OFF_DIFF:=DB10.DBW8        Switch-off difference
          BREAK :=DB10.DBB10      Maximum velocity
     POS_MODE2 :=DB10.DBX12.0     Jog mode forward
     POS_MODE1 :=DB10.DBX12.1     Jog mode backward
    REF_ENABLE :=
       POS_STRT :=DB10.DBX12.2    Start positioning operation
        SET_POS :=DB10.DBX12.3    Control signal: Accept REF_VAL as new actual
    ACTUAL_POS :=                 value
     POS_READY :=DB10.DBX13.0     Checkback signal: Pos. oper./jog mode running
      REF_VALID :=DB10.DBX13.1    Checkback signal: Ref. point switch reached
```

| | |
|---|---|
| POS_VALID :=DB10.DBX13.2 | Checkback signal: Synchron. has taken place |
| A BR | Scanning the BR bit (= ENO at SFB 39) for |
| = DB10.DBX 13.7 | error evaluation |
| Setting up the foil | |
| A DB10.DBX 12.4 | During job execution: Cut foil? If yes, |
| JC m1 | then jump to execution: Cut foil |
| A I 0.0 | Momentary-contact switch: Jog forward |
| AN I 0.1 | Interlock with jog backward |
| AN I 0.3 | Interlock with automatic |
| = DB10.DBX 12.0 | Start jog forward |
| A I 0.1 | Momentary-contact switch: Jog backward |
| AN I 0.0 | Interlock with jog forward |
| AN I 0.3 | Interlock with automatic |
| = DB10.DBX 12.1 | Start jog backward |
| A I 0.2 | Momentary-contact switch: Cut foil and set reference point |
| FP DB10.DBX 12.7 | Edge evaluation for momentary-contact switch |
| A DB10.DBX 13.0 | Scan POS:READY for positioning terminated |
| S DB10.DBX 12.4 | Set memory marker for job request: Cut foil |
| Automatic mode | |
| AN I 0.3 | Automatic mode switch |
| AN DB10.DBX 12.5 | Auxiliary memory marker for terminating auto- |
| BEC | matic mode |
| AN DB10.DBX 12.2 | Start positioning operation |
| S DB10.DBX 12.2 | |
| S DB10.DBX 12.5 | Set auxiliary memory marker for terminating |
| BEC | automatic mode |
| A DB10.DBX 13.0 | If positioning terminated, then |
| S DB10.DBX 12.4 | set memory marker for cutting the foil |
| R DB10.DBX 12.2 | |
| R DB10.DBX 12.5 | Reset auxiliary memory marker |
| BEU | |
| Cut foil, accept reference point | |
| m1: NOP 0 | |
| A I 0.7 | Checkback signal from cutter, cutting termi-nated |
| A DB10.DBX 12.3 | Reference point has been accepted by IF as new actual value |
| R DB10.DBX 12.3 | Reset signal |
| R DB10.DBX 12.4 | Reset memory marker for cutting job |
| R Q 4.0 | Reset signal for cutter |
| L S5T#500MS | Waiting time till drive standstill |
| A DB10.DBX 12.4 | (e.g.: 500 ms) |
| SD T 1 | |
| A DB10.DBX 13.0 | Positioning terminated, |
| A DB10.DBX 12.4 | Memory marker for cutting job set |
| A T 1 | and time out? |
| S DB10.DBX 12.3 | Then accept reference point as actual value |
| S Q 4.0 | Start cutting |

**برنامه ۷ : انجام Positioning با خروجی آنالوگ CPU31xC و توسط SFB44**

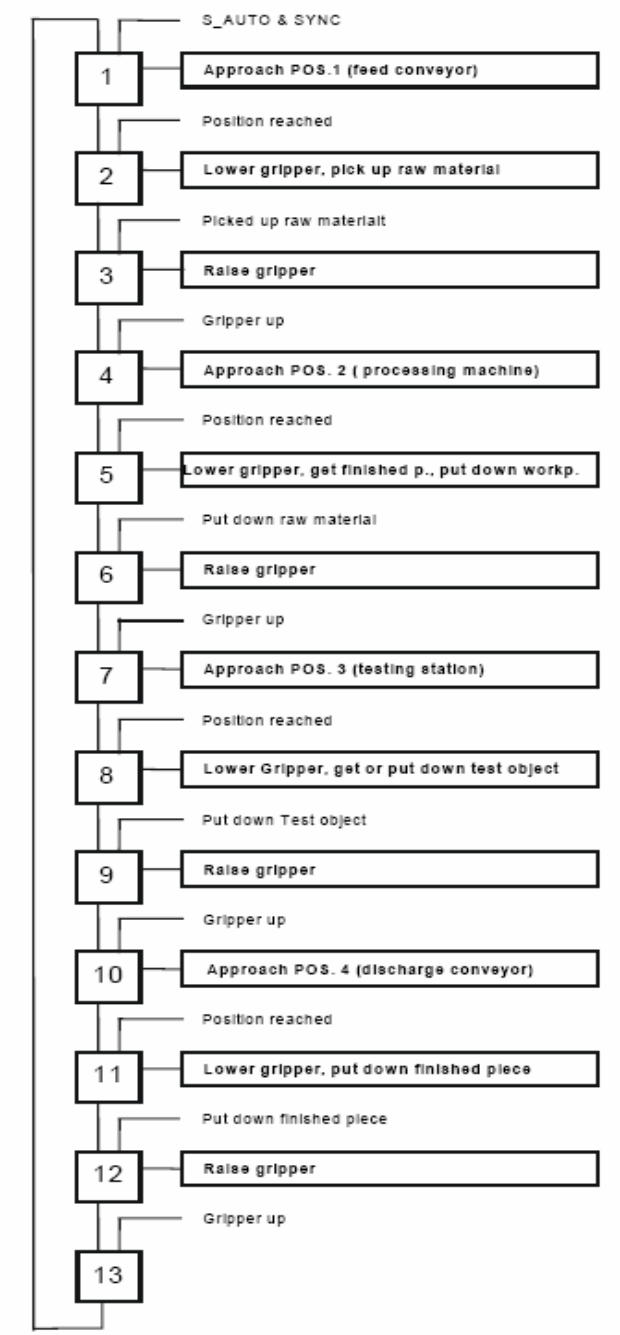
در این مثال Positioning با خروجی آنالوگ و استفاده از SFB44 نشان داده میشود سیگنال های کنترلی برای مد اتوماتیک بصورت ترتیبی ایجاد میشوند امکان مد دستی یعنی Jog Mode و Incremental نیز وجود دارد.

همانطور که در شکل زیر نشان داده شده است یک Loading Portal وظیفه جابجایی قطعه ای که لازم است روی آن عملیات ماشین کاری انجام شود را بعهده دارد. این پرتال قطعه را از روی نوار نقاله ورودی برداشته وروی ماشین دوم قرار می دهد. پس از اتمام ماشین کاری آنرا برداشته و به ایستگاه تست منتقل می کندو پس از تست آنرا روی نوار نقاله خروجی قرار می دهد.




حرکت افقی پرتال توسط موتور سروداریو و حرکت عمودی آن با سیلندر پنوماتیکی انجام می شود.

اگر محور سنکرون نشده باشدلازم است قبل از شروع کار عملیات سنکرون سازی انجام شود . برای پرهیز از طولانی شدن برنامه فرض بر اینست که محورها سنکرون شده اند لذا صرفاً به عملیات اصلی Positioning میپردازیم.توالی عملیات کنترلی در شکل صفحه بعد نشان داده شده است.


S_AUTO & SYNC
1
Approach POS.1 (feed conveyor)
Position reached
2
Lower gripper, pick up raw material
Picked up raw materialt
3
Raise gripper
Gripper up
4
Approach POS. 2 ( processing machine)
Position reached
5
Lower gripper, get finished p., put down workp.
Put down raw material
6
Raise gripper
Gripper up
7
Approach POS. 3 (testing station)
Position reached
8
Lower Gripper, get or put down test object
Put down Test object
9
Raise gripper
Gripper up
10
Approach POS. 4 (discharge conveyor)
Position reached
11
Lower gripper, put down finished piece
Put down finished piece
12
Raise gripper
Gripper up
13

## تبدیل فواصل به پالس

انکودر در هر دور ۲۵۰۰ پالس میفرستد . هر دور انکودر معادل ۴ دور موتور است و با هردور موتور پرتال به اندازه 100 mm جابجا می شود.یعنی هر دور انکودرمعادل 25mm است. پس

25 mm / 2500 pulse = 0.01 mm

یعنی هر پالس انکودر معادل جابجایی 0.01 mm است.

بدین ترتیب موقعیت فواصل ۴ گانه پرتال با معادل پالس آنها در جدول زیر آمده است.

| **Target positions** | **Conversion to pulses (distance increments)** |
|---|---|
| Position 1 Feed conveyor | 100 mm / 0.01 mm/Imp. = **10 000**pulses |
| Position 2 processing machine | 1800 mm / 0.01 mm/pulse = **180 000**pulses |
| Position 3 Testing station | 3000 mm / 0.01 mm/Imp. = **300 000**pulses |
| Position 4 Discharge conveyor | 4500 mm / 0.01 mm/Imp. = **450 000**pulses |

## پارامترهای تنظیمی در HWconfig

این پارامترها مطابق جدول زیر تنظیم می شود:

| **Parameters** | **Input** |
|---|---|
| Type of technology | Positioning with analog output |
| Target range | 100 pulses<br>(1 mm / 0,01 mm/Imp. = 100) |
| Monitoring time | 2000 ms |
| Maximum speed | 10000 pulses/s |
| Creep/reference approach speed | 100 pulses/s |
| Axis type | Linear axis |
| Software limit switch Start | -5000 pulses<br>Operating range limit Start<br>(-50 mm / 0.01 mm/Imp. = -5000 pulses) |
| Software limit switch End | 500,000 pulses<br>Operating range limit End<br>(5000 mm / 0.01 mm/Imp. = 500,000 pulses) |
| Reference point to reference point switch | In plus direction |
| Increments per encoder revolution | 2500 |
| Counting direction | Standard<br>(Track A transition before B = positive actual value) |

**Changeover Difference**

در این مثال فرض شده که فاصله Changeover معادل 50 mm است که معادل پالس آن بصورت زیر است:

50 mm / 0.01 mm = 5000 pulses

**Switch-Off Difference**

این فاصله که اختلاف بین نقطه Off و مقصد است بایستی بطور دقیق و مطابق مراحل زیر انجام شود:

۱. ابتدا مقدار آنرا بزرگتر یا مساوی نصف Target Range قرار دهید. یعنی بزرگتر یا مساوی ۵۰
۲. پرتال را یکبار در مد Absoulute Incremental حرکت دهید تا بایستد.
۳. فاصله نقطه ایستادن را تا نقطه مقصد اندازه گیری کنید وآنرا بعنوان فاصله Switch off استفاده کنید.

در این مثال این فاصله ۶۰ پالس در نظر گرفته شده است.

در این مثال برنامه OB100 برای شرایط و مقادیر اولیه بکار رفته است . در OB1 بلاکFB3 همراه با DB3 صدا زده شده و در FB3 فانکشن SFB44 همراه با DB6 فراخوان شده است.

```
DATA_BLOCK DB 6
TITLE =
AUTHOR : SIMATIC
NAME : DI_ANA
VERSION : 1.0

 SFB 44
BEGIN
  LADDR := W#16#310;
  CHANNEL := 0;
  DRV_EN := FALSE;
  START := FALSE;
  DIR_P := FALSE;
  DIR_M := FALSE;
  STOP := FALSE;
  ERR_A := FALSE;
  MODE_IN := 1;
  TARGET := L#1000;
  SPEED := L#1000;
  WORKING := FALSE;
  POS_RCD := FALSE;
  MSR_DONE := FALSE;
  SYNC := FALSE;
  ACT_POS := L#0;
  MODE_OUT := 0;
  ERR := W#16#0;
  ST_ENBLD := FALSE;
  ERROR := FALSE;
  STATUS := W#16#0;
  ACCEL := L#100;
  DECEL := L#100;
  CHGDIFF_P := L#1000;
```

```
  CUTOFFDIFF_P := L#100;
  CHGDIFF_M := L#1000;
  CUTOFFDIFF_M := L#100;
  PARA := FALSE;
  DIR := FALSE;
  CUTOFF := FALSE;
  CHGOVER := FALSE;
  RAMP_DN := FALSE;
  RAMP_UP := FALSE;
  DIST_TO_GO := L#0;
  LAST_TRG := L#0;
  BEG_VAL := L#0;
  END_VAL := L#0;
  LEN_VAL := L#0;
  JOB_REQ := FALSE;
  JOB_DONE := TRUE;
  JOB_ERR := FALSE;
  JOB_ID := 0;
  JOB_STAT := W#16#0;
  JOB_VAL := L#0;
  JOB_DBNR := 0;
  JOB_DBOFFSET := 0;
  RES[0] := B#16#0;
  RES[1] := B#16#0;
  RES[2] := B#16#0;
  RES[3] := B#16#0;
  RES[4] := B#16#0;
  RES[5] := B#16#0;
  RES[6] := B#16#0;
  RES[7] := B#16#0;
END_DATA_BLOCK
```

---

**FUNCTION_BLOCK FB 3**

```
TITLE =Sample 3: Loading portal
//This sample shows a practical application.
//The program has the following structure:
//  1. Sequential control for automatic processing
//  2. Operating mode section with position assignment / calculation
//  3. Positioning call SFB ANALOG
//  4. Evaluation of IN POSITION (POS_RCD)
AUTHOR : SIMATIC
NAME : PORTAL
VERSION : 1.0


VAR
  DRV_EN : BOOL ;   //SFB parameter: Drive enable
  START : BOOL ;    //SFB parameter: Start general
  DIR_P : BOOL ;    //SFB-Parameter: Start positive run
  DIR_M : BOOL ;    //SFB parameter: Start negative run
  STOP : BOOL ; //SFB parameter: Run Stop
  ERR_A : BOOL ;    //SFB parameter: Global acknowledgment of hardware error
  MODE_IN : INT ;    //SFB parameter: Operating mode
  TARGET : DINT ;   //SFB parameter: Target / Distance
  SPEED : DINT ;    //SFB parameter: Speed
  WORKING : BOOL ;  //SFB parameter: Run in operation
  POS_RCD : BOOL ;  //SFB parameter: In specified position
  MSR_DONE : BOOL ; //SFB parameter: Length measurement done
  SYNC : BOOL ; //SFB parameter: Axis is synchronized
  ACT_POS : DINT ;  //SFB parameter: actual position value
  MODE_OUT : INT ;  //SFB parameter: active / configured operating mode
```

```
  ERR : WORD ;   //SFB parameter: Hardware error
  ST_ENBLD : BOOL ; //SFB parameter: Start enable
  ERROR : BOOL ;    //SFB parameter: Run start/resume error
  STATUS : WORD ;   //SFB parameter: Error ID
  S_DRV_EN : BOOL ; //Control signal:  Drive enable / Controller enable
  S_STOP : BOOL ;   //Control signal:  Stop
  S_ERR_A : BOOL ;  //Control signal:  Global acknowledgment of hardware error
  S_REF : BOOL ;    //Control signal:  Start Reference point approach
  S_DIR_PF : BOOL ; //Control signal:  Jogging into positive direction, rapid speed
  S_DIR_MF : BOOL ; //Control signal:  Jogging into negative direction, rapid speed
  S_DIR_PS : BOOL ; //Control signal:  Jogging into positive direction, slow speed
  S_DIR_MS : BOOL ; //Control signal:  Jogging into positive direction, negative speed
  S_POS1 : BOOL ;   //Control signal:  Approach Pos 1
  S_POS2 : BOOL ;   //Control signal:  Approach Pos 2
  S_POS3 : BOOL ;   //Control signal:  Approach Pos 3
  S_POS4 : BOOL ;   //Control signal:  Approach Pos 4
  S_AUTO : BOOL ;   //Control signal:  Automatic=1 Manual=0
  Pos1 : DINT  := L#10000;  //Setpoint: Position 1 (Feed conveyor)
  Pos2 : DINT  := L#180000; //Setpoint: Position 2 (Processing machine)
  Pos3 : DINT  := L#300000; //Setpoint: Position 3 (Testing station)
  Pos4 : DINT  := L#450000; //Setpoint: Position 4 (Delivery conveyor)
  Speed_FAST : DINT  := L#600;  //Setpoint: RAPID speed (depending on maximum speed)
  Speed_SLOW : DINT  := L#100;  //Setpoint: SLOW speed (depending on maximimum speed)
  A_Pos1 : BOOL ;  //Auto: Approach Pos. 1
  A_Pos2 : BOOL ;   //Auto: Approach Pos. 2
  A_Pos3 : BOOL ;   //Auto: Approach Pos. 3
  A_Pos4 : BOOL ;   //Auto: Approach Pos. 4
  Pos1_RCD : BOOL ; //Position 1 reached
  Pos2_RCD : BOOL ; //Position 2 reached
  Pos3_RCD : BOOL ; //Position 3 reached
  Pos4_RCD : BOOL ; //Position 4 reached
  POS_RCD_FM : BOOL ;   //Edge evaluation position reached
  TimeStart1 : BOOL ;   //Delay time 1 start
  TimeStart2 : BOOL ;   //Delay time 2
  StepNr : BYTE ;   //Step number
END_VAR
BEGIN
NETWORK
TITLE =Default values see OB100
//Switchover/switch-off difference, acceleration, deceleration and speed are set
//in OB100


NETWORK
TITLE =Sequential control for auto mode

//Reset sequential control
     A     #S_AUTO;
     JC    sres;
     R     #A_Pos1;
     R     #A_Pos2;
     R     #A_Pos3;
     R     #A_Pos4;
     L     0;
     T     #StepNr;
     JU    sEnd;
sres: NOP   0;

//Sequential control
     L     #StepNr;
     JL    se;
```

```
    JU    s0; //Step 0
    JU    s1; // Step 1, approach POS.1 (Feed tape)
    JU    s2; // Step 2, lower gripper, fetch raw material
    JU    s3; // Step 3, raise gripper
    JU    s4; // Step 4, approach POS.2 (Processing machine)
    JU    s5; // Step 5, lower gripper, fetch finished part, deliver raw material
    JU    s6; //Step 6, raise gripper
    JU    s7; // Step 7, approach POS.3 (Testing station)
    JU    s8; // Step 8. Lower gripper, fetch / deliver test object
    JU    s9; //Step 9, raise gripper
    JU    s10; // Step 10, approach POS.4 (Delivery conveyor)
    JU    s11; //Step 11, lower gripper, deliver finished part
    JU    s12; //Step 12, raise gripper
    JU    s13; //Step 13, return jump

se:  L     0;
    T    #StepNr;
    JU    sEnd;
s0:  A    #SYNC; //If axis is synchronized ...
    JC    next; //.. then next step
    JU    sEnd; //.. otherwise Stop
//-----------------------------------------------------------------------------
s1:  SET   ; //STEP 1
    S    #A_Pos1; // -> approach POS.1 FEED CONVEYOR
    A    #Pos1_RCD; //If in POS.1 ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s2:  SET   ; //STEP 2
    S    #TimeStart1; // -> Lower gripper, fetch raw material
    R    #A_Pos1;
    A    T     10; //When raw material is fetched ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s3:  SET   ; //STEP 3
    S    #TimeStart2; // -> Raise gripper
    R    #TimeStart1;
    A    T     11; // After gripper is raised ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

//-----------------------------------------------------------------------------
s4:  SET   ; //STEP 4
    S    #A_Pos2; // -> Approach POS.2 PROCESSING MACHINE
    R    #TimeStart2;
    A    #Pos2_RCD; // When in POS.2 ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s5:  SET   ; //STEP 5
    S    #TimeStart1; // -> Lower gripper, fetch finished part, deliver raw material
    R    #A_Pos2;
    A    T     10; // When raw material is delivered ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s6:  SET   ; //STEP 6
    S    #TimeStart2; // -> Raise gripper
    R    #TimeStart1;
    A    T     11; //After gripper is raised
```

```
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

//-----------------------------------------------------------------------------
s7:   SET   ; //STEP 7
    S     #A_Pos3; // -> Approach POS.3 TESTING STATION
    R     #TimeStart2;
    A     #Pos3_RCD; // When in POS.3 ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s8:   SET   ; //STEP 8
    S     #TimeStart1; // -> Lower gripper, fetch / deliver test object
    R     #A_Pos3;
    A     T     10; // When test object is delivered ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s9:   SET   ; //STEP 9
    S     #TimeStart2; // -> Raise gripper
    R     #TimeStart1;
    A     T     11; // After gripper is raised ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

//-----------------------------------------------------------------------------
s10:  SET   ; //STEP 10
    S     #A_Pos4; // -> Approach POS.4 DELIVERY CONVEYOR
    R     #TimeStart2;
    A     #Pos4_RCD; // When in POS.4 ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s11:  SET   ; //STEP 11
    S     #TimeStart1; // -> Lower gripper, deliver finished part
    R     #A_Pos4;
    A     T     10; //Finished part delivered ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s12:  SET   ; //STEP 12
    S     #TimeStart2; // -> Raise gripper
    R     #TimeStart1;
    A     T     11; // After gripper is raised ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

//-----------------------------------------------------------------------------
s13:  SET   ; //STEP 13
    R     #TimeStart2;
    L     0; //Return jump
    T     #StepNr;
    JU    sEnd;

next: NOP   0;
    L     #StepNr; //Load Step number ..
    +     1; // .. and increment by 1 ..
    T     #StepNr; // ..

sEnd: NOP   0; //End of stepping sequence
    A     #TimeStart1;
```

```
    L    S5T#2S; // Simulation "Lower gripper, Fetch / deliver part"
    SD   T    10; //   via delay

    A    #TimeStart2;
    L    S5T#2S; // Simulation "Deliver tool plate"
    SD   T    11; //   via delay

NETWORK
TITLE =Default
//In this network the "Run Start signals" are reset, because the signals in the
//specific networks remain set due to conditional processing
    CLR   ;
    =    #START; // Start, general
    =    #DIR_P; // Run Start, positive direction
    =    #DIR_M; // Run Start, negative direction
    L    0;
    T    #SPEED; //Speed
NETWORK
TITLE =Reference point approach
//Setting requests and control signals for the operating mode "Reference point
//approach".
//
//Definition:    You can synchronize the channel as a result of a
//              repetitive external event.
//
//Prerequisites: 1. The axis must be configured
//               2. You are using an incremental encoder
    A    #S_REF; //START REFERENCE POINT APPROACH
    JCN   ref; // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    L    3;
    T    #MODE_IN; //Operating mode: Start reference point approach
    L    #Speed_FAST;
    T    #SPEED; //Rapid speed
    S    #DIR_M; // Run Start, negative direction
ref:  NOP   0;
NETWORK
TITLE =Jogging
//Setting control signals for the operating modes
//  - "Jogging into direction + with rapid speed"
//  - "Jogging into direction - with rapid speed"
//  - "Jogging into direction + with slow speed"
//  - "Jogging into direction - with slow speed".
//
//Prerequisite: 1. The axis must be configured
//
    A    #S_AUTO;
    JC    tip4;

    A    #S_DIR_PF; //JOGGING FORWARD AT RAPID SPEED
    JCN   tip1; // ~~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_P; // Run Start, positive direction
    L    #Speed_FAST;
    T    #SPEED; //Rapid speed

tip1: A    #S_DIR_MF; //JOGGING BACKWARD AT RAPID SPEED
    JCN   tip2; // ~~~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_M; // Run Start, negative direction
```

```
    L    #Speed_FAST;
    T    #SPEED; //Rapid speed

tip2: A    #S_DIR_PS; //JOGGING FORWARD AT SLOW SPEED
    JCN  tip3; // ~~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_P; // Run Start, positive direction
    L    #Speed_SLOW;
    T    #SPEED; // Slow speed

tip3: A    #S_DIR_MS; // JOGGING BACKWARD AT SLOW SPEED
    JCN  tip4; // ~~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_M; // Run Start, negative direction
    L    #Speed_SLOW;
    T    #SPEED; // Slow speed

tip4: NOP  0;
NETWORK
TITLE =Absolute incremental approach
//Setting control signals for the operating mode "Absolute incremental approach".
//
//Prerequisite: 1. The axis must be configured
//              2. The channel is synchronized, that is, a reference point
//                 approach must be carried out before, or a reference point
//must be set.
//
    AN   #SYNC; //Axis is synchronized
    JC   sma4;

    A    #S_POS1; // APPROACH POSITION 1A (ABSOLUTE)
    AN   #S_AUTO; // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    O    #A_Pos1;
    JCN  sma1;
    L    5;
    T    #MODE_IN; //Operating mode: Absolute incremental approach
    L    #Pos1;
    T    #TARGET; //Target/distance
    S    #START; // Start
    L    #Speed_FAST;
    T    #SPEED; //Rapid speed

sma1: A    #S_POS2; // APPROACH POSITION 2A (ABSOLUTE)
    AN   #S_AUTO; // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    O    #A_Pos2;
    JCN  sma2;
    L    5;
    T    #MODE_IN; //Operating mode: Absolute incremental approach
    L    #Pos2;
    T    #TARGET; //Target/distance
    S    #START; // Start
    L    #Speed_SLOW;
    T    #SPEED; //Rapid speed

sma2: A    #S_POS3; // APPROACH POSITION 3A (ABSOLUTE)
    AN   #S_AUTO; // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    O    #A_Pos3;
    JCN  sma3;
    L    5;
```

```
    T    #MODE_IN; //Operating mode: Absolute incremental approach
    L    #Pos3;
    T    #TARGET; //Target/distance
    S    #START; // Start
    L    #Speed_SLOW;
    T    #SPEED; //Rapid speed

sma3: A    #S_POS4; // APPROACH POSITION 4A (ABSOLUTE)
    AN   #S_AUTO; // ~~~
```

```
        ERR                 := #ERR,
        ST_ENBLD            := #ST_ENBLD,
        ERROR               := #ERROR,
        STATUS              := #STATUS);

    NOP   0;
NETWORK
TITLE = Evaluation of IN POSITION (POS_RCD)

    A     #POS_RCD; //In specified position
    FP    #POS_RCD_FM;
    JCN   rpos; // Edge evaluation POS_RCD

    A     #S_POS1;
    AN    #S_AUTO;
    O     #A_Pos1;
    S     #Pos1_RCD; // Position 1 reached (Feed conveyor)

    A     #S_POS2;
    AN    #S_AUTO;
    O     #A_Pos2;
    S     #Pos2_RCD; // Position 2 reached (Processing machine)

    A     #S_POS3;
    AN    #S_AUTO;
    O     #A_Pos3;
    S     #Pos3_RCD; // Position 3 reached (Testing station)

    A     #S_POS4;
    AN    #S_AUTO;
    O     #A_Pos4;
    S     #Pos4_RCD; // Position 4 reached (Delivery conveyor)


rpos: AN    #POS_RCD; // Reset Position memory bit
    R     #Pos1_RCD;
    R     #Pos2_RCD;
    R     #Pos3_RCD;
    R     #Pos4_RCD;
END_FUNCTION_BLOCK
```

---

```
DATA_BLOCK DB 3
TITLE =
VERSION : 0.0

 FB 3
BEGIN
  DRV_EN := FALSE;
  START := FALSE;
  DIR_P := FALSE;
  DIR_M := FALSE;
  STOP := FALSE;
  ERR_A := FALSE;
  MODE_IN := 0;
  TARGET := L#0;
  SPEED := L#0;
  WORKING := FALSE;
  POS_RCD := FALSE;
  MSR_DONE := FALSE;
  SYNC := FALSE;
  ACT_POS := L#0;
```

```
  MODE_OUT := 0;
  ERR := W#16#0;
  ST_ENBLD := FALSE;
  ERROR := FALSE;
  STATUS := W#16#0;
  S_DRV_EN := FALSE;
  S_STOP := FALSE;
  S_ERR_A := FALSE;
  S_REF := FALSE;
  S_DIR_PF := FALSE;
  S_DIR_MF := FALSE;
  S_DIR_PS := FALSE;
  S_DIR_MS := FALSE;
  S_POS1 := FALSE;
  S_POS2 := FALSE;
  S_POS3 := FALSE;
  S_POS4 := FALSE;
  S_AUTO := FALSE;
  Pos1 := L#10000;
  Pos2 := L#180000;
  Pos3 := L#300000;
  Pos4 := L#450000;
  Speed_FAST := L#600;
  Speed_SLOW := L#100;
  A_Pos1 := FALSE;
  A_Pos2 := FALSE;
  A_Pos3 := FALSE;
  A_Pos4 := FALSE;
  Pos1_RCD := FALSE;
  Pos2_RCD := FALSE;
  Pos3_RCD := FALSE;
  Pos4_RCD := FALSE;
  POS_RCD_FM := FALSE;
  TimeStart1 := FALSE;
  TimeStart2 := FALSE;
  StepNr := B#16#0;
END_DATA_BLOCK
```

**ORGANIZATION_BLOCK OB 1**

```
TITLE =Cycle Execution
//Application sample for S7-300C technology: Positioning with analog output
//
//Functions / function blocks used:
//  FB3      FB  PORTAL         Sample 3: Loading portal
//  SFB44    SFB POS_ANALOG     Integrierter Systemfunktionsbaustein
//
//Data blocks used:
//  DB3      DB  DI_PORTAL      Instance DB to PORTAL
//  DB6      DB  DI_ANALOG      Instance DB to SFB POS_ANALOG
VERSION : 1.0


VAR_TEMP
  OB1_EV_CLASS : BYTE ; //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ;   //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ; //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ; //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;   //Reserved for system
  OB1_RESERVED_2 : BYTE ;   //Reserved for system
  OB1_PREV_CYCLE : INT ;    //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ; //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ; //Maximum cycle time of OB1 (milliseconds)
```

```
  OB1_DATE_TIME : DATE_AND_TIME ;   //Date and time OB1 started
  RETVAL_CTRL : INT ;  //Return value CTRL
END_VAR
BEGIN
NETWORK
TITLE =Call of sample program
//
//

    CALL FB     3 , DB     3 ;//Sample 3: Loading portal
    BE    ;
END_ORGANIZATION_BLOCK
```

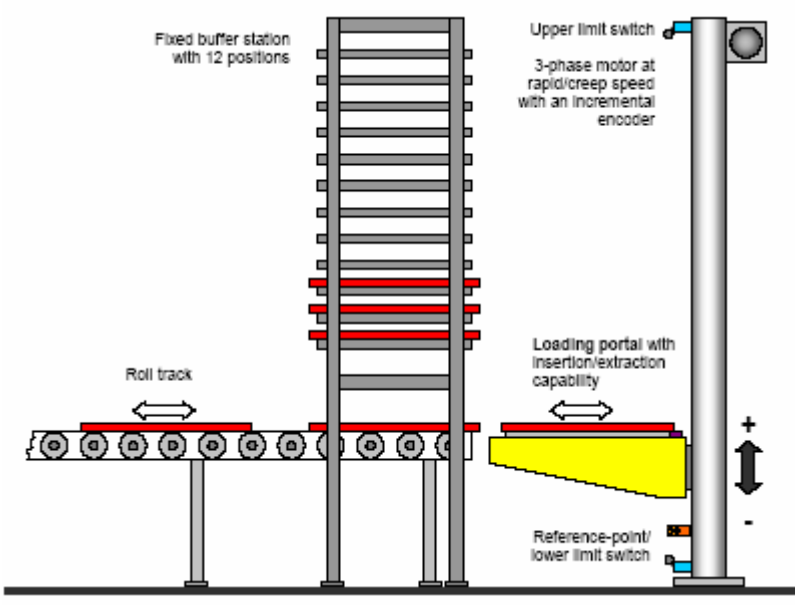
---

**ORGANIZATION_BLOCK OB 100**

```
TITLE =Complete Restart
AUTHOR : SIMATIC
VERSION : 1.0


VAR_TEMP
  OB100_EV_CLASS : BYTE ;   //16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
  OB100_STRTUP : BYTE ; //16#81/82/83/84 Method of startup
  OB100_PRIORITY : BYTE ;   //27 (Priority of 1 is lowest)
  OB100_OB_NUMBR : BYTE ;   //100 (Organization block 100, OB100)
  OB100_RESERVED_1 : BYTE ; //Reserved for system
  OB100_RESERVED_2 : BYTE ; //Reserved for system
  OB100_STOP : WORD ;   //Event that caused CPU to stop (16#4xxx)
  OB100_STRT_INFO : DWORD ; //Information on how system started
  OB100_DATE_TIME : DATE_AND_TIME ; //Date and time OB100 started
END_VAR
BEGIN
NETWORK
TITLE =Reset control signals

    L     0;
    T     DB3.DBW   26; // Reset Control signals Sample 3

NETWORK
TITLE =Default values
//Switchover/switch-off difference, acceleration and deceleration
//
    L     L#5000; // Switchover difference
    T     DB6.DBD   38;
    T     DB6.DBD   46;
    L     L#60; // Switch-off difference
    T     DB6.DBD   42;
    L     L#60;
    T     DB6.DBD   50;
    L     L#10000; //Acceleration
    T     DB6.DBD   30;
    L     L#10000; //Deceleration
    T     DB6.DBD   34;
    L     L#0; // Rapid speed (e.g. 600, depending on creep/maximum speed -> see HW Config)
    T     DB3.DBD   44;
    L     L#0; // Slow speed (e.g.300, depending on creep/maximum speed -> see HW Config)
    T     DB3.DBD   48;

END_ORGANIZATION_BLOCK
```

**برنامه ۸ : انجام Positioning با خروجی های دیجیتال CPU31xC و توسط SFB46**

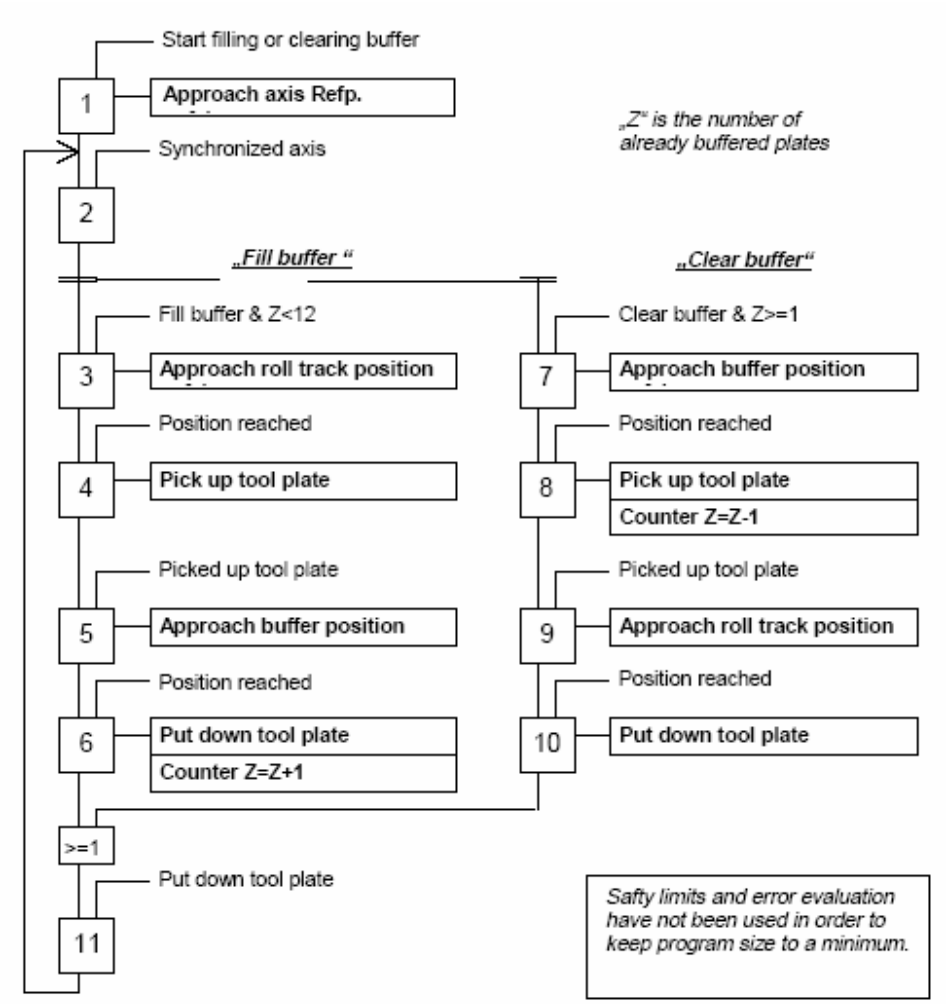
در این مثال مطابق شکل زیر یک پرتال صفحاتی را از روی نوار بر می دارد سپس آنها را به محل ذخیره سازی (بافر) انتقال می دهد. بافر ظرفیت ۱۲ صفحه را دارد. عمل عکس یعنی تخلیه بافر نیز امکان پذیر است در اینحالت پرتال صفحات را از روی بافر برداشته و به نوار منتقل می کند.




پرتال با یک موتور ۳ فاز دو سرعته (Rapid/Creep) کار می کند. انکودر Incremental بعنوان فیدبک موقعیت در نظر گرفته شده است. سوئیچ نقطه رفرنس در زیر پرتال قرار دارد . سوئیچ های حد پایین و حد بالا نیز تعبیه شده اند.

مراحل کنترل ترتیبی سیستم در شکل صفحه بعد ترسیم شده است. پرتال پس از روشن شدن سیستم ابتدا سنکرون شده سپس عملیات پر کردن یا تخلیه بافر را انجام می دهد.برای پر کردن موقعیت اولیه آن روبروی نوار است سپس روبروی یکی از موقعیت های ۱۲ گانه بافر قرار می گیرد و آنرا پر می کند. پر کردن به ترتیب صورت می گیرد و هر بار به یک موقعیت بالاتر می رود. کانتری شمارش این صفحات را انجام می دهد.

عمل تخلیه بافر بر عکس است یعنی از موقعیت بالاتر به سمت موقعیت پایین.


Start filling or clearing buffer
1
Approach axis Refp.
Synchronized axis
2
„Z" is the number of already buffered plates
„Fill buffer "
„Clear buffer"
Fill buffer & Z<12
3
Approach roll track position
Position reached
4
Pick up tool plate
Picked up tool plate
5
Approach buffer position
Position reached
6
Put down tool plate
Counter Z=Z+1
>=1
Put down tool plate
11
Clear buffer & Z>=1
7
Approach buffer position
Position reached
8
Pick up tool plate
Counter Z=Z-1
Picked up tool plate
9
Approach roll track position
Position reached
10
Put down tool plate
Safty limits and error evaluation have not been used in order to keep program size to a minimum.

## تبدیل فاصله به پالس

انکودر ۲۵۰۰ پالس در هر دور میفرستد . هر دور انکودر معادل ۴ دور موتور است پس هر دور موتور معادل 10,000 پالس است . موتور در هر دور پرتال را 100 mm جابجا می کند بنابر این داریم:

100 mm / 10,000 Pulse = 0.01mm

یعنی هر پالس انکودر معادل 0.01 mm جابجایی است. باتوجه به این عدد موقعیت پله های بافر و سایر موقعیت ها طبق جدول زیر خواهد بود:

| **Target positions** | **Conversion in pulses (distance increments)** |
|---|---|
| Buffer height 12 | Buffer height 1 + (12-1) * Buffer comp. height<br>1100 mm + (12-1) * 100mm = 2200 mm<br>2200 mm : 0.01 mm/pul. = **220 000**Pulses |
| Buffer height 2..11 | See calcultion for buffer height 12 |
| Buffer height 1 | 1100 mm : 0.01 mm/pul. = **110 000** Pulses |
| Roll track height | 600 mm : 0.01 mm/pul. = **60 000** Pulses |
| Reference point | 300 mm : 0.01 mm/pul. = **30 000** Pulses |
| Buffer com .height | 100 mm : 0.01 mm / pul= **10 000** Pulses |

## تنظیمات Hwconfig

این تنظیمات طبق جدول زیر است:

| **Parameters** | **Description** |
|---|---|
| Type of technology | Positioning With Digital Outputs |
| Type of control | 1 |
| Target range | 100 pulses (1 mm / 0.01 mm/Imp. = 100 pulses) |
| Monitoring time | 2000 ms |
| Axis type | Linear axis |
| Software limit switch Start | 25000 pulses Operating range limit Start (250 mm / 0.01 mm/Imp. = 25000 pulses) |
| Software limit switch End | 250000 pulses |
| | Operating range limit End |
| | (2250 mm / 0.01 mm/Imp. = 225 000 pulses) |
| Reference point coordinate | 30000 pulses |
| Reference point to reference point switch | In plus direction |
| Increments per encoder revolution | 2500 |
| Counting direction | Standard |

**Changeover Difference**

این فاصله یعنی زمانی که طول می کشد درایو از سرعت rapid به سرعت creep برسد در این مثال 60mm در نظر گرفته شده است. که با تقسیم بر 0.01 mm معادل 6000 پالس است.

**Switch-off Difference**

این مقدار بر اساس تستی که انجام می شود بدست می آید . مراحل تست شبیه آنچه در مثال آنالوگ قبلی گفته شد می باشد یعنی ابتدا مقدار آن را بزرگتر یا مساوی نصف Target Range ( که در اینجا 100پالس تنظیم شده) در نظر گرفته و پرتال را در مد Absolute Incremental حرکت می دهیم تا بایستد سپس مقدار واقعی اختلاف تا مقصد را بدست آورده به عنوان Switch off Difference وارد می کنیم در این مثال مقدار فوق - 2500 بدست آمده است.پس مقادیر زیر را در برنامه وارد می کنیم:

| | | | |
|---|---|---|---|
| Switchover difference + | = | **6000** | pulses |
| Switchover difference - | = | **6000** | pulses |
| Switch-off difference + | = | **250** | pulses |
| Switch-off difference - | = | **250** | pulses |

سایر مقادیری که بعنوان ورودی داده می شوند بصورت زیر هستند:

| **Parameters** | **Setting** | **Meaning** |
|---|---|---|
| **PosCon** | xx xxx xxx | Setpoint Position value roll track height |
| **PosS1** | xx xxx xxx | Setpoint Position value buffer 1 |
| **PosSH** | xx xxx xxx | Setpoint Position value buffer compartment height (relative dimension) |
| **S_DRV_EN** | 1 | Control signal: Drive enable |
| **S_Fill** | 1 | Start auto mode for filling the buffer |
| **S_Empty** | 1 | Start "Clear buffer" auto mode |
| **S_REF** | 1 | Start reference point approach |
| **S_DIR_PF** | 1 | Start jogging forward at rapid speed |
| **S_DIR_MF** | 1 | Start jogging backward at rapid speed |
| **S_DIR_PS** | 1 | Start jogging forward at slow speed |
| **S_DIR_MS** | 1 | Start jogging backward at slow speed |
| **S_PosCon** | 1 | Approach roll track Start Pos (absolute) |
| **S_PosS1** | 1 | Approach buffer 1 Start Pos. (absolute) |
| **S_PosS_P** | 1 | Approach buffer Start Pos. +1 (absolute) |
| **S_PosS_M** | 1 | Approach buffer Start Pos. -1 (absolute) |

با این توضیحات برنامه OB100 و OB1 در صفحات بعد آمده است.در OB1 فانکشن FB1 همراه با DB3 صدا زده شده و در FB3 فانکشن SFB46 همراه با DB6 فراخوان شده است.

```
DATA_BLOCK DB 6
TITLE =
AUTHOR : SIMATIC
NAME : DI_DIGIT
VERSION : 1.0

 SFB 46
BEGIN
  LADDR := W#16#310;
  CHANNEL := 0;
  DRV_EN := FALSE;
  START := FALSE;
  DIR_P := FALSE;
  DIR_M := FALSE;
  STOP := FALSE;
  ERR_A := FALSE;
  MODE_IN := 1;
  TARGET := L#1000;
  SPEED := FALSE;
  WORKING := FALSE;
  POS_RCD := FALSE;
  MSR_DONE := FALSE;
  SYNC := FALSE;
  ACT_POS := L#0;
  MODE_OUT := 0;
  ERR := W#16#0;
  ST_ENBLD := FALSE;
  ERROR := FALSE;
  STATUS := W#16#0;
  CHGDIFF_P := L#1000;
  CUTOFFDIFF_P := L#100;
  CHGDIFF_M := L#1000;
  CUTOFFDIFF_M := L#100;
  PARA := FALSE;
  DIR := FALSE;
  CUTOFF := FALSE;
  CHGOVER := FALSE;
  DIST_TO_GO := L#0;
  LAST_TRG := L#0;
  BEG_VAL := L#0;
  END_VAL := L#0;
  LEN_VAL := L#0;
  JOB_REQ := FALSE;
  JOB_DONE := TRUE;
  JOB_ERR := FALSE;
  JOB_ID := 0;
  JOB_STAT := W#16#0;
  JOB_VAL := L#0;
  JOB_DBNR := 0;
  JOB_DBOFFSET := 0;
  RES[0] := B#16#0;
  RES[1] := B#16#0;
  RES[2] := B#16#0;
  RES[3] := B#16#0;
  RES[4] := B#16#0;
  RES[5] := B#16#0;
  RES[6] := B#16#0;
  RES[7] := B#16#0;
END_DATA_BLOCK
```

```
FUNCTION_BLOCK FB 3
TITLE =Sample 3: BUFFER STATION
//This sample shows a practical application.
//The program has the following structure:
//  1. Sequential control for automatic processing
//  2. Operating mode section with position assignment / calculation
//  3. Positioning call SFB DIGITAL
//  4. Evaluation of IN POSITION (POS_RCD)
AUTHOR : SIMATIC
NAME : STORAGE
VERSION : 1.0

VAR
  DRV_EN : BOOL ;   //SFB parameter: Drive enable
  START : BOOL ;    //SFB parameter: Start general
  DIR_P : BOOL ;    //SFB-Parameter: Start positive run
  DIR_M : BOOL ;    //SFB parameter: Start negative run
  STOP : BOOL ; //SFB parameter: Run Stop
  ERR_A : BOOL ;    //SFB parameter: Global acknowledgment of hardware error
  MODE_IN : INT ;   //SFB parameter: Operating mode
  TARGET : DINT ;   //SFB parameter: Target / Distance
  SPEED : BOOL ;    //SFB parameter: Speed
  Res9 : BYTE ; //SFB parameter: Reserve
  WORKING : BOOL ;  //SFB parameter: Run in operation
  POS_RCD : BOOL ;  //SFB parameter: In specified position
  MSR_DONE : BOOL ; //SFB parameter: Length measurement done
  SYNC : BOOL ; //SFB parameter: Axis is synchronized
  ACT_POS : DINT ;  //SFB parameter: actual position value
  MODE_OUT : INT ;  //SFB parameter: active / configured operating mode
  ERR : WORD ;  //SFB parameter: Hardware error
  ST_ENBLD : BOOL ; //SFB parameter: Start enable
  ERROR : BOOL ;    //SFB parameter: Run start/resume error
  STATUS : WORD ;   //SFB parameter: Error ID
  S_DRV_EN : BOOL ; //Control signal:  Drive enable
  S_STOP : BOOL ;   //Control signal:  Stop
  S_ERR_A : BOOL ;  //Control signal:  Global acknowledgment of hardware error
  S_REF : BOOL ;    //Control signal:  Start Reference point approach
  S_DIR_PH : BOOL ; //Control signal:  Jogging into positive direction, rapid speed
  S_DIR_MH : BOOL ; //Control signal:  Jogging into negative direction, rapid speed
  S_DIR_PS : BOOL ; //Control signal:  Jogging into positive direction, slow speed
  S_DIR_MS : BOOL ; //Control signal:  Jogging into positive direction, negative speed
  S_PosCon : BOOL ; //Control signal:  Approach Pos. Roll track
  S_PosS1 : BOOL ;  //Control signal:  Pos. Magazine compartment 1
  S_PosS_P : BOOL ; //Control signal:  Pos. Magazine compartment + 1 (relative)
  S_PosS_M : BOOL ; //Control signal:  Pos. Magazine compartment - 1 (relative)
  S_Fill : BOOL ;   //Control signal:  Start filling magazine (Automatic)
  S_Empty : BOOL ;  //Control signal:  Start emptying magazine (Automatic)
  PosCon : DINT  := L#60000;    //Position value: Roll track height
  PosS1 : DINT  := L#110000;    //Position value: magazine 1
  PosSx : DINT ;    //Position value: magazine x (calculated)
  PosSH : DINT  := L#10000; //Position value: magazine height (Relative dimension)
  Hand : BOOL ; //Manual=1 Auto=0
  A_REF : BOOL ;    //Auto: Approach Reference point
  A_PosCon : BOOL ; //Auto: Approach Pos. Roll track
  A_PosSx : BOOL ;  //Auto: Approach Pos. Magazine compartment x
  PosCon_RCD : BOOL ;   //In roll track position
  PosS1_RCD : BOOL ;    //Position magazine compartment 1 reached
  PosSx_RCD : BOOL ;    //Position magazine compartment x reached
  POS_RCD_FM : BOOL ;   //Edge evaluation position reached
  TimeStart : BOOL ;    //Start delay time
  StepNr : BYTE ;   //Step number
```

```
  Count : BYTE ;    //Counter: Number of plates in magazine
END_VAR
BEGIN
NETWORK
TITLE =Default values see OB100
//The switchover/switch-off difference is set in OB100.


NETWORK
TITLE =Sequential control for auto mode
//Operating mode
//Manual mode
     AN    #S_Fill; //If no start signal "Fill" ..
     AN    #S_Empty; // .. and no start signal "Empty" .. is active,..
     =     #Hand; // .. then manual mode

// Reset stepping sequence and position memory
     AN    #Hand; //If Auto not selected ..
     JC    sres; // .. jump to sres
     R     #A_PosCon; // .. otherwise,  reset position memory bit
     R     #A_PosSx;
     R     #A_REF;
     L     0; // .. and stepping sequence
     T     #StepNr;
sres: NOP   0;

//Sequential control
     L     #StepNr;
     JL    se;
     JU    s0; //Step 0
     JU    s1; //Step 1, approach reference point
     JU    s2; // Step 2, branch-off FILL or EMPTY
     JU    s3; // Step 3 FILL, approach roll track pos.
     JU    s4; //Step 4 FILL, fetch tool plate
     JU    s5; //Step 5 FILL, approach magazine
     JU    s6; //Step 5 FILL, deliver tool plate
     JU    s7; //Step 7 EMPTY, approach magazine pos.
     JU    s8; //Step 8 EMPTY, fetch tool plate
     JU    s9; //Step 9 EMPTY, approach roll track pos.
     JU    s10; //Step 10 EMPTY, deliver tool plate
     JU    s11; //Step 11, return jump

se:  L     0;
     T     #StepNr;
     JU    sEnd;
s0:  NOP   0; //STEP 0
     A     #S_Fill; // If "Start filling magazine"
     O     #S_Empty; // or "Start emptying magazine" is active ..
     JC    next; //.. then next step
     JU    sEnd; // .. otherwise wait

s1:  AN    #SYNC; //STEP 1
     S     #A_REF; // -> Approach Reference point
     A     #SYNC; //If axis is synchronized ...
     A     #ST_ENBLD; // .. and Start is enabled ..
     JC    next; //.. then next step
     JU    sEnd; // .. otherwise wait

s2:  SET   ; //STEP 2
     R     #A_REF;
     L     #Count; // If the magazine is not full ..
```

```
    L     12;
    <I    ;
    A     #S_Fill; // .. and "Fill magazine" is selected ..
    JC    next; // .. then next step FILL BUFFER

    NOP   0;
    L     #Count; // If at least one tool plate is in the magazine ..
    L     1;
    >=I   ;
    A     #S_Empty; // .. and "Empty magazine" is selected ..
    JCN   sEnd;
    L     7; // .. then jump to step 7 EMPTY MAGAZINE
    T     #StepNr;
    JU    sEnd;
// ---------------------------------------  F I L L I N G  T H E  M A G A Z I N E -------
s3:  SET   ; //STEP 3
    S     #A_PosCon; // -> approach roll track pos.
    A     #PosCon_RCD; // When in position roll track ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s4:  SET   ; //STEP 4
    S     #TimeStart; // -> fetch tool plate from roll track
    R     #A_PosCon;
    A     T     10; // When tool plate is picked up ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s5:  SET   ; //STEP 5
    S     #A_PosSx; // -> Approach Magazine x
    R     #TimeStart;
    A     #PosSx_RCD; // When in magazine position ..
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s6:  SET   ; //STEP 6
    S     #TimeStart; // -> deliver the tool plate to the magazine
    R     #A_PosSx;
    A     T     10; // When tool plate is delivered ..
    JCN   sEnd;
    L     #Count; // .. increment Counter Z=Z+1 ..
    INC   1;
    T     #Count;
    L     11; // .. and return jump
    T     #StepNr;
    JU    sEnd; // .. otherwise wait

// ---------------------------------------  E M P T Y I N G  T H E  M A G A Z I N E -------
s7:  SET   ; //STEP 7
    S     #A_PosSx; // -> Approach Magazine x
    R     #TimeStart;
    A     #PosSx_RCD; // When in magazine position
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s8:  SET   ; //STEP 8
    S     #TimeStart; // -> fetch tool plate from magazine
    R     #A_PosSx;
    A     T     10; // When tool plate is picked up ..
    JCN   sEnd;
    L     #Count; // ..  Decrement Counter Z=Z-1 ..
```

```
    DEC   1;
    T     #Count;
    L     9; // .. and next step
    T     #StepNr;
    JU    sEnd; // .. otherwise wait

s9:   SET   ; //STEP 9
    R     #TimeStart;
    S     #A_PosCon; // -> Approach roll track
    R     T     10;
    A     #PosCon_RCD; // When in roll track position
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s10:  SET   ; //STEP 10
    S     #TimeStart; // -> Deliver the tool plate to the roll track
    R     #A_PosCon;
    A     T     10; // When tool plate is delivered
    JC    next; //.. then next step
    JU    sEnd; // .. otherwise wait

s11:  SET   ; //STEP 11
    R     #TimeStart;
    L     2; //Return jump
    T     #StepNr;
    JU    sEnd;

next: NOP   0;
    L     #StepNr; //Load Step number ..
    +     1; // .. and increment by 1 ..
    T     #StepNr; // ..

sEnd: NOP   0; //End of stepping sequence

    A     #TimeStart;
    L     S5T#3S; // Simulation "Deliver tool plate"
    SD    T     10; //   via delay

NETWORK
TITLE =Operating mode section with position assignment / calculation

NETWORK
TITLE =Default
//In this network the "Run Start signals" are reset, because the signals in the
//specific networks remain set due to conditional processing
    CLR   ;
    =     #START; // Start, general
    =     #DIR_P; // Run Start, positive direction
    =     #DIR_M; // Run Start, negative direction
    =     #SPEED; //Speed
NETWORK
TITLE =Reference point approach
//Setting requests and control signals for the operating mode "Reference point
//approach".
//
//Definition:    You can synchronize the channel as a result of a
//               repetitive external event.
//
//Prerequisites: 1. The axis must be configured
//               2. You are using an incremental encoder
    A     #S_REF; // START REFERENCE POINT APPROACH (Manual mode)
```

```
    A    #Hand;
    O    #A_REF; // START REFERENCE POINT APPROACH (Auto mode)
    JCN  ref;
    L    3;
    T    #MODE_IN; //Operating mode: Start reference point approach
//  S    #SPEED                    // Optional: rapid speed
    S    #DIR_M; // Run Start, negative direction
ref:  NOP  0;
NETWORK
TITLE =Jogging
//Setting control signals for the operating modes
//  - "Jogging into direction + with rapid speed"
//  - "Jogging into direction - with rapid speed"
//  - "Jogging into direction + with creep speed"
//  - "Jogging into direction - with creep speed".
//
//Prerequisite: 1. The axis must be configured
//              2. Auto mode "Fill / empty magazine" is disabled
    AN   #Hand; //MANUAL MODE
    JC   smr2;

    A    #S_DIR_PH; //JOGGING UP FAST
    JCN  tip1; // ~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_P; // Run Start, positive direction
    S    #SPEED; //Rapid speed

tip1: A    #S_DIR_MH; //JOGING DOWN FAST
    JCN  tip2; // ~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_M; // Run Start, negative direction
    S    #SPEED; //Rapid speed

tip2: A    #S_DIR_PS; //JOGGING UP SLOW
    JCN  tip3; // ~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_P; // Run Start, positive direction

tip3: A    #S_DIR_MS; //JOGGING DOWN SLOW
    JCN  tip4; // ~~~~~~~~~~~~~~~~~~~~~~~~
    L    1;
    T    #MODE_IN; //Operating mode: Jogging
    S    #DIR_M; // Run Start, negative direction
tip4: NOP  0;
NETWORK
TITLE =Relative incremental approach
//Setting control signals for the operating mode "Relative incremental approach"
//
//Prerequisite: 1. The axis must be configured
//              2. The channel is synchronized, that is, a reference point
//                 approach must be carried out before, or a reference point
//must be set.
//              3.  Auto mode "Fill / empty magazine" is disabled.
    A    #S_PosS_P; //MAGAZINE COMPARTMENT +1
    A    #SYNC; // ~~~~~~~~~~~~~~~~~
    JCN  smr1;
    L    4;
    T    #MODE_IN; //Operating mode: Relative incremental approach
```

```
    L    #PosSH; // Position value magazine compartment height
    T    #TARGET; //Target/distance
    S    #DIR_P; //Run forward
    S    #SPEED; //Rapid speed

smr1: A    #S_PosS_M; //MAGAZINE COMPARTMENT -1
    A    #SYNC; // ~~~~~~~~~~~~~~~~~~
    JCN  smr2;
    L    4;
    T    #MODE_IN; //Operating mode: Relative incremental approach
    L    #PosSH; // Position value magazine compartment height
    T    #TARGET; //Target/distance
    S    #DIR_M; //Run backward
    S    #SPEED; //Rapid speed
smr2: NOP  0;
NETWORK
TITLE =Absolute incremental approach
//Setting control signals for the operating mode "Absolute incremental approach".
//
//Prerequisite: 1. The axis must be configured
//              2. The channel is synchronized, that is, a reference point
//                 approach must be carried out before, or a reference point
//must be set.
    AN   #SYNC; //Axis is synchronized
    JC   sma3;
    A    #S_PosCon; //APPROCH ROLL TRACK POSITION
    A    #Hand; // ~~~~~~~~~~~~~~~~~~~~~~~~~~
    O    #A_PosCon;
    JCN  sma1;
    L    5;
    T    #MODE_IN; //Operating mode: Absolute incremental approach
    L    #PosCon; // Position value roll track height
    T    #TARGET; //Target/distance
    S    #START; // Start
    S    #SPEED; //Rapid speed

sma1: A    #S_PosS1; //APPROACH MAGAZINE COMPARTMENT 1
    A    #Hand; // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    JCN  sma2;
    L    5;
    T    #MODE_IN; //Operating mode: Absolute incremental approach
    L    #PosS1; // Positions value magazine 1
    T    #TARGET; //Target/distance
    S    #START; // Start
    S    #SPEED; //Rapid speed

// Calculation: FILL magazine = Magazine compartment 1 + ( Counter    * Magazine compartment height)
//            EMPTY magazine = Magazine 1 + ((Counter-1) * Magazine compartment height)

sma2: A    #A_PosSx; //APPROACH MAGAZINE X
    JCN  sma3; // ~~~~~~~~~~~~~~~~~~~~~~~~~~
    L    5;
    T    #MODE_IN; //Operating mode: Absolute incremental approach
    A    #S_Empty; //Fill magazine
    JCN  spf1;
    L    #Count; //Load number of plates in magazine
    DEC  1; // Decrement by 1
    JU   spf2;
spf1: L    #Count; // Empty the magazine
spf2: L    #PosSH; // Load magazine compartment height
    *D   ;
```

```
    L    #PosS1; // Load magazine 1
    +D   ;
    T    #TARGET; // calculated magazine
    S    #START; // Start
    S    #SPEED; //Rapid speed
sma3: NOP   0;
NETWORK
TITLE = Positioning call SFB POS_DIGITAL
//Call of SFB POS_DIGITAL//Transfer control signals and feedback, process the jobs.
    A    #S_DRV_EN;
    =    L     0.0;
    BLD  103;
    A    #START;
    =    L     0.1;
    BLD  103;
    A    #DIR_P;
    =    L     0.2;
    BLD  103;
    A    #DIR_M;
    =    L     0.3;
    BLD  103;
    A    #S_STOP;
    =    L     0.4;
    BLD  103;
    A    #S_ERR_A;
    =    L     0.5;
    BLD  103;
    A    #SPEED;
    =    L     0.6;
    BLD  103;
    CALL SFB   46 , DB     6 (
        LADDR          := W#16#310,
        CHANNEL        := 0,
        DRV_EN         := L     0.0,
        START          := L     0.1,
        DIR_P          := L     0.2,
        DIR_M          := L     0.3,
        STOP           := L     0.4,
        ERR_A          := L     0.5,
        MODE_IN        := #MODE_IN,
        TARGET         := #TARGET,
        SPEED          := L     0.6,
        WORKING        := #WORKING,
        POS_RCD        := #POS_RCD,
        MSR_DONE       := #MSR_DONE,
        SYNC           := #SYNC,
        ACT_POS        := #ACT_POS,
        MODE_OUT       := #MODE_OUT,
        ERR            := #ERR,
        ST_ENBLD       := #ST_ENBLD,
        ERROR          := #ERROR,
        STATUS         := #STATUS);

    NOP   0;
NETWORK
TITLE = Evaluation of IN POSITION (POS_RCD)

    A    #POS_RCD; //In specified position
    FP   #POS_RCD_FM;
    JCN  rpos; // Edge evaluation POS_RCD
```

```
    A    #S_PosCon;
    A    #Hand;
    O    #A_PosCon;
    S    #PosCon_RCD; // Roll track position reached

    A    #S_PosS1;
    A    #Hand;
    S    #PosS1_RCD; // in magazine 1 position

    A    #A_PosSx;
    S    #PosSx_RCD; // in magazine x position

rpos: AN    #POS_RCD; // Reset Position memory bit
    R    #PosCon_RCD;
    R    #PosS1_RCD;
    R    #PosSx_RCD;

END_FUNCTION_BLOCK
```

**DATA_BLOCK DB 3**

```
TITLE =
VERSION : 0.0

 FB 3
BEGIN
  DRV_EN := FALSE;
  START := FALSE;
  DIR_P := FALSE;
  DIR_M := FALSE;
  STOP := FALSE;
  ERR_A := FALSE;
  MODE_IN := 0;
  TARGET := L#0;
  SPEED := FALSE;
  Res9 := B#16#0;
  WORKING := FALSE;
  POS_RCD := FALSE;
  MSR_DONE := FALSE;
  SYNC := FALSE;
  ACT_POS := L#0;
  MODE_OUT := 0;
  ERR := W#16#0;
  ST_ENBLD := FALSE;
  ERROR := FALSE;
  STATUS := W#16#0;
  S_DRV_EN := FALSE;
  S_STOP := FALSE;
  S_ERR_A := FALSE;
  S_REF := FALSE;
  S_DIR_PH := FALSE;
  S_DIR_MH := FALSE;
  S_DIR_PS := FALSE;
  S_DIR_MS := FALSE;
  S_PosCon := FALSE;
  S_PosS1 := FALSE;
  S_PosS_P := FALSE;
  S_PosS_M := FALSE;
  S_Fill := FALSE;
  S_Empty := FALSE;
  PosCon := L#60000;
  PosS1 := L#110000;
  PosSx := L#0;
```

```
  PosSH := L#10000;
  Hand := FALSE;
  A_REF := FALSE;
  A_PosCon := FALSE;
  A_PosSx := FALSE;
  PosCon_RCD := FALSE;
  PosS1_RCD := FALSE;
  PosSx_RCD := FALSE;
  POS_RCD_FM := FALSE;
  TimeStart := FALSE;
  StepNr := B#16#0;
  Count := B#16#0;
END_DATA_BLOCK
```

---

**ORGANIZATION_BLOCK OB 1**

```
TITLE =Cycle Execution
//Application sample for S7-300C Technology: Positioning with Digital Outputs
//
//Functions / function blocks used:
//  FB3      FB  STORAGE        Sample 3: Buffer station
//  SFB46    SFB POS_DIGITAL    Integrated system function block
//
//Data blocks used:
//  DB3      DB  DI_STORAGE     Instance DB to STORAGE
//  DB6      DB  DI_DIGITAL     Instance DB to SFB POS_DIGITAL
VERSION : 1.0


VAR_TEMP
  RETVAL_CTRL : INT ;   //Return value CTRL
END_VAR
BEGIN
NETWORK
TITLE =Call
     CALL FB     3 , DB     3 ;// Sample 3: Magazine
     BE    ;
END_ORGANIZATION_BLOCK
```

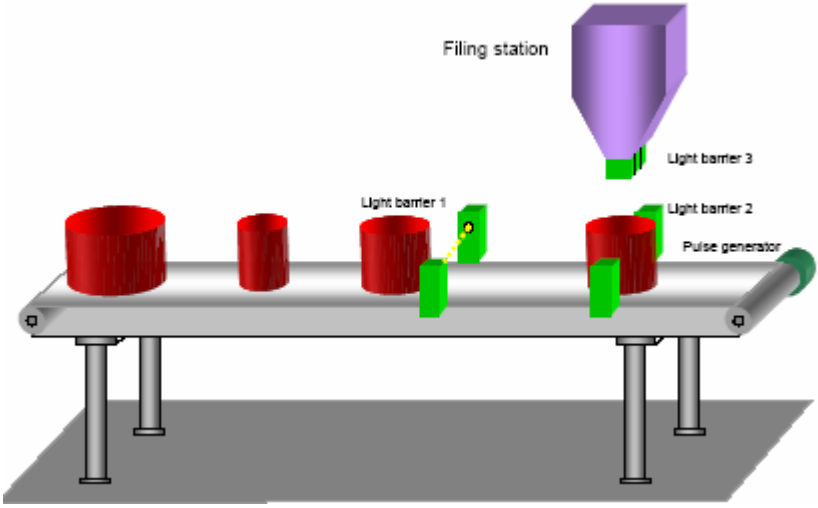
---

**ORGANIZATION_BLOCK OB 100**

```
TITLE =Complete Restart
AUTHOR : SIMATIC
VERSION : 1.0

VAR_TEMP
END_VAR
BEGIN
NETWORK
TITLE =Reset control signals
     L     0;
     T     DB3.DBW   24; // Reset Control signals
NETWORK
TITLE =Default values
//Um- Abschaltdifferenz
     L     L#6000; // Switchover difference
     T     DB6.DBD   28;
     T     DB6.DBD   36;
     L     L#250; // Switch-off difference
     T     DB6.DBD   32;
     T     DB6.DBD   40;
END_ORGANIZATION_BLOCK
```

**برنامه ۹ : انجام عملیات شمارش با CPU31xC و توسط SFB47**

در فرآیندی مطابق شکل زیر ظروف استوانه ای شکل با ابعاد مختلف توسط یک ایستگاه تزریق (Filling) پر می شوند. ظروف روی نوار حرکت می کنند وقتی Light Barrier 1 ظرف را حس کرد اندازه گیری قطر ظرف با شمارش پالسهایی که از یک تولید کننده پالس می رسد شروع می شود اینکار با عبور ظرف از جلوی سنسور فوق خاتمه می یابد. قطر ظرف برای محاسبه حجم آن استفاده می شود و میزان پر شدن با توجه به حجم تنظیم می گردد.




روش کار باینصورت است که وقتی هیچ ظرفی روی نوار نیست یا کار پر سازی تمام شده نوار استارت می شود.کانتر شماره 0 قطر ظرف را محاسبه می کندگیت این کانتر توسط Light Barrier 1 بصورت سخت افزاری کنترل می شود. وقتی ظرف از جلوی سنسور فوق عبور کرد کانتر از کار می افتد و مقدار شمارش شده برای بکار انداختن کانتر ۱ بکار می رود. کانتر ۱ برای ایستگاه Filling با Job شماره ۴ که مربوط به مبنای مقایسه است بکار می رود.

وقتی ظرف در موقعیت حس شده توسط Light Barrier 2 قرار گرفت کار Filling شروع می شود میزان موادی که به ظرف شارژ می شود توسط کانتر ۱ محاسبه می شود.این مواد بصورتی هستند که در هر میلیمتر مکعب تعداد 10,000 ماده وجود دارد.

ظرف استوانه ای شکل است و حجم آن از رابطه $d^2 * \pi/4 * h$ بدست می آید. ارتفاع برای همه ظروف یکسان و 100 mm است .

با این مشخصات اگر قطر یک ظرف 150 mm اندازه گیری شود خواهیم داشت:

$$\begin{aligned} \text{Filling quantity} &= d^2 * \pi/4 * h/10\,000 \\ &= 150\text{ mm} * 150\text{ mm} * (\pi/4) * 100\text{ mm}/10000\text{ parts/mm}^3 \\ &= \underline{176\text{ parts}} \end{aligned}$$

پس این ظرف میتواند ۱۷۶ قطعه را در خود جای دهد.

### تنظیمات HWconfig

این تنظیمات طبق جدول زیر است:

برای کانتر 0

| Parameters | Input |
|---|---|
| Channel | 0 |
| Operating mode | Infinite count |
| Gate function | Abort count |
| Signal evaluation | Pulse/Direction |
| Input function | Use hardware gate |
| Reaction of output DO | No comparison |

برای کانتر 1

| Parameters | Input |
|---|---|
| Channel | 1 |
| Operating mode | Infinite count |
| Gate function | Abort count |
| Signal evaluation | Pulse/Direction |
| Input function | Hardware gate not used |
| Reaction of output DO | The output switches when Counter value >= Comparison value |

### برنامه نویسی

در OB1 بلاک FB12 با DB12 صدا زده شده و در FB12 فانکشن SFB47 یکبار با DB16 (برای کانتر 0) و بار دیگر با DB17 (برای کانتر 1 ) فراخوان شده است.

**DATA_BLOCK DB 16**
TITLE =
AUTHOR : SIMATIC
NAME : DI_COUNT
VERSION : 1.0

SFB 47
BEGIN
LADDR := W#16#300;
CHANNEL := 0;
SW_GATE := FALSE;
CTRL_DO := FALSE;
SET_DO := FALSE;
JOB_REQ := FALSE;
JOB_ID := W#16#0;
JOB_VAL := L#0;
STS_GATE := FALSE;
STS_STRT := FALSE;
STS_LTCH := FALSE;
STS_DO := FALSE;
STS_C_DN := FALSE;
STS_C_UP := FALSE;
COUNTVAL := L#0;
LATCHVAL := L#0;
JOB_DONE := FALSE;
JOB_ERR := FALSE;
JOB_STAT := W#16#0;
RES00 := FALSE;
RES01 := FALSE;
RES02 := FALSE;
STS_CMP := FALSE;
RES04 := FALSE;
STS_OFLW := FALSE;
STS_UFLW := FALSE;
STS_ZP := FALSE;
JOB_OVAL := L#0;
RES10 := FALSE;
RES11 := FALSE;
RES_STS := FALSE;
END_DATA_BLOCK

---

**DATA_BLOCK DB 17**
TITLE =
AUTHOR : SIMATIC
NAME : DI_COUNT
VERSION : 1.0

SFB 47
BEGIN
LADDR := W#16#300;
CHANNEL := 0;
SW_GATE := FALSE;
CTRL_DO := FALSE;
SET_DO := FALSE;
JOB_REQ := FALSE;
JOB_ID := W#16#0;
JOB_VAL := L#0;
STS_GATE := FALSE;
STS_STRT := FALSE;
STS_LTCH := FALSE;
STS_DO := FALSE;

```
  STS_C_DN := FALSE;
  STS_C_UP := FALSE;
  COUNTVAL := L#0;
  LATCHVAL := L#0;
  JOB_DONE := FALSE;
  JOB_ERR := FALSE;
  JOB_STAT := W#16#0;
  RES00 := FALSE;
  RES01 := FALSE;
  RES02 := FALSE;
  STS_CMP := FALSE;
  RES04 := FALSE;
  STS_OFLW := FALSE;
  STS_UFLW := FALSE;
  STS_ZP := FALSE;
  JOB_OVAL := L#0;
  RES10 := FALSE;
  RES11 := FALSE;
  RES_STS := FALSE;
END_DATA_BLOCK
```

---

**FUNCTION_BLOCK FB 12**

```
TITLE =Filling station with automatic detection of container size
//The Task:
//Containers of different sizes are filled in this plant.
//Containers of different sizes are are transported on a feed conveyor belt.
//Light barrier 1
//detects the containers. The container diameter is determined
//per pulse count.
//The volume is calculated and the filling quantity of the filling station is set
//according to the determined container diameter.
//
AUTHOR : SIMATIC
NAME : FILL
VERSION : 1.0


VAR
  S_START : BOOL ;  //control signal: Start
  S_LS1 : BOOL ;    //control signal: Light barrier 1 (Container diameter)
  S_LS2 : BOOL ;    //control signal: Light barrier 2 (Container in filling position)
  TOGGLE : BOOL ;   //Toggle bit for pulse generation
  F_PULSE : BOOL ;  //Pos. Edge of the pulse period has expired
  T_PULSE : S5TIME  := S5T#200MS;   //Setpoint: Pulse period
  DIAM : DINT ;   //Container diameter (diameter)
  FILL_SETP : DINT ;    //Fill setpoint
  FILL_VAL : DINT ; //Fill actual value
  STS_STRT0 : BOOL ;    //Hardware gate 0 Light barrier 1 (Container diameter)
  FN_STS_STRT0 : BOOL ; //Edge memory bit for measuring the container diameter
  CONV_ON : BOOL ;  //Conveyor ON
  C_POS_RCD : BOOL ;    //Conveyor in position
  FILL_STRT : BOOL ;    //Start filling
  FILL_WORK : BOOL ;    //Filling in operation
  FILL_DONE : BOOL ;    //Filling is done
  FN_FILL_W : BOOL ;    //Edge memory bit Filling is done
  JOB_REQ : BOOL ;  //Load fill setpoint to counter
  JOB_DONE : BOOL ; //Counter Job done
  JOB_VAL : DINT ;  //Counter Control value
  JOB_ERR : BOOL ;  //Job error
  JOB_STAT : WORD ; //Job error ID
END_VAR
```

```
BEGIN
NETWORK
TITLE =Pulse generation and simulation of inputs

// Pulse generation
//******************
    A    T    10; //Pulse period expired
    =    DB12.DBX    0.4; // Pos.edge Pulse period expired
    JCN   takt;
    AN    DB12.DBX    0.3; // Toggle Bit
    =    DB12.DBX    0.3;
takt: NOP   0;
    AN    DB12.DBX    0.4; // Pulse period Start
    L    DB12.DBW    2; //pulse width
    SD    T    10; // Timer Pulse period

// Simulation of inputs
//**************************
    A    DB12.DBX    0.3; // If Toggle bit=1
    =    Q    124.2; //then activate pulse input A

    A    DB12.DBX    0.1; // Control signal: Light barrier 1 (Container diameter)
    =    Q    124.4; // Open hardware gate

NETWORK
TITLE =start conveyor

    NOP   0;
    A    #S_START; // Control signal: Start
    A(    ;
    ON    DB12.DBX    0.2; // Control signal: Light barrier 2 (Container in filling position)
    O    #FILL_DONE; //Filling done
    )    ;
    =    #CONV_ON; //Conveyor ON
NETWORK
TITLE =measuring container diameter

    A    #CONV_ON;
    =    L      0.0;
    BLD   103;
    CALL SFB   47 , DB    16 (
        LADDR                := W#16#300,
        CHANNEL                := 0,
        SW_GATE                := L      0.0,
        STS_STRT               := #STS_STRT0,
        COUNTVAL                := #DIAM);

    NOP   0;
NETWORK
TITLE =calculate fill setpoint

//Fill setpoint is calculated when leaving light barrier 1
    A    #STS_STRT0; //Hardware gate: Light barrier 1 (Container diameter)
    FN    #FN_STS_STRT0;
    JCN   ber;
    SET   ;
    =    #JOB_REQ; //Counter 2 Load fill setpoint
    L    #DIAM; //Container diameter
    L    #DIAM;
    *D    ;
    L    314;
```

```
    *D   ;
    L    100;
    /D   ;
    L    4;
    /D   ;
    L    100; // Height = 100 mm
    *D   ;
    L    10000; // parts per 10000 mm3
    /D   ;
    T    #FILL_SETP; //Fill setpoint (comparison value)
    L    1;
    -D   ;
    T    #JOB_VAL; //Counter setpoint = Fill setpoint - 1
    L    0;
    >=D  ;
    JC   ber;
    L    0;
    T    #JOB_VAL;
ber:  NOP   0;


NETWORK
TITLE =start to fill

    NOP   0;
    A    #S_LS2; // Control signal: Light barrier 2 (Container in filling position)
    AN   #FILL_DONE; //Filling done
    =    #FILL_STRT; //Start to fill

NETWORK
TITLE =filling station

    A    #FILL_STRT;
    =    L     0.0;
    BLD  103;
    A    #FILL_STRT;
    =    L     0.1;
    BLD  103;
    A    #JOB_REQ;
    =    L     0.3;
    BLD  103;
    CALL SFB   47 , DB    17 (
        LADDR          := W#16#300,
        CHANNEL          := 1,
        SW_GATE          := L     0.0,
        CTRL_DO          := L     0.1,
        JOB_REQ          := L     0.3,
        JOB_ID         := W#16#4,
        JOB_VAL          := #JOB_VAL,
        STS_DO         := #FILL_WORK,
        COUNTVAL          := #FILL_VAL,
        JOB_DONE          := #JOB_DONE,
        JOB_ERR         := #JOB_ERR,
        JOB_STAT          := #JOB_STAT);
    NOP   0;
NETWORK
TITLE =filling is done

    AN   #FILL_WORK; //If filling is done
    FP   #FN_FILL_W;
    S    #FILL_DONE; //set "Filling is done"
```

```
    AN    #S_LS2; //When the container has left filling position, ...
    R     #FILL_DONE; //reset "Filling is done"

NETWORK
TITLE =reset the jobs
//
//
//
    NOP   0;
    A     #JOB_DONE; //job counter 2 is done
    R     #JOB_REQ;
END_FUNCTION_BLOCK
```

---

**DATA_BLOCK DB 12**

```
AUTHOR : SIMATIC
NAME : DI_FILL
VERSION : 1.0

 FB 12
BEGIN
  S_START := FALSE;
  S_LS1 := FALSE;
  S_LS2 := FALSE;
  TOGGLE := FALSE;
  F_PULSE := FALSE;
  T_PULSE := S5T#200MS;
  DIAM := L#0;
  FILL_SETP := L#0;
  FILL_VAL := L#0;
  STS_STRT0 := FALSE;
  FN_STS_STRT0 := FALSE;
  CONV_ON := FALSE;
  C_POS_RCD := FALSE;
  FILL_STRT := FALSE;
  FILL_WORK := FALSE;
  FILL_DONE := FALSE;
  FN_FILL_W := FALSE;
  JOB_REQ := FALSE;
  JOB_DONE := FALSE;
  JOB_VAL := L#0;
  JOB_ERR := FALSE;
  JOB_STAT := W#16#0;
END_DATA_BLOCK
```

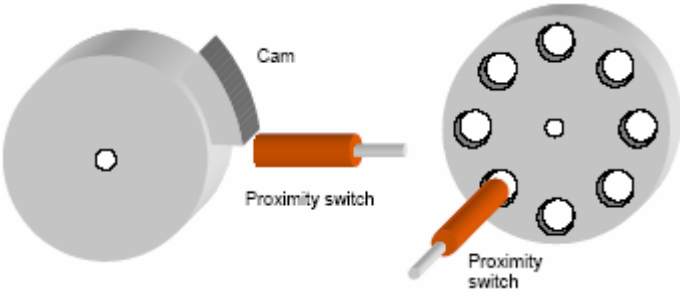
---

**ORGANIZATION_BLOCK OB 1**

```
TITLE =Cycle Execution
VERSION : 1.0


VAR_TEMP
 L_BEARB : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Call of sample "Counting 2, Filling station"

    CALL FB    12 , DB    12 ;
    BE    ;
END_ORGANIZATION_BLOCK
```

**برنامه ۱۰ : اندازه گیری سرعت شافت با CPU31xC و توسط SFB48**

در این مثال با استفاده از SFB48 تغییرات سرعت یک شافت اندازه گیری می شود. پروکسیمیتی سوئیچ به یکی از دو طریق نشان داده شده در شکل زیر متناسب با دور موتور پالس میفرستد:




با رسیدن سیگنال Start اندازه گیری فرکانس شروع می شود. برنامه سرعت مبنا و تولرانس آن را می گیرد سپس حد بالا و حد پایین را محاسبه می کند اگر سرعت واقعی Speed_Val به حد بالا برسد بیت STATUS_U و اگر به حد پایین برسد بیت STATUS_O فعال می گردد.

نحوه محاسبه حد بالا و حد پایین از فرمول های زیر است:

$$\text{lower limit} = \frac{\text{speed}}{60} \times \frac{(100 - \text{tolerance})}{100} \times \text{pulse} \times 1000\ \text{mHz}$$

$$\text{upper limit} = \frac{\text{speed}}{60} \times \frac{(100 + \text{tolerance})}{100} \times \text{pulse} \times 1000\ \text{mHz}$$

**مثال** :

Speed Setpoint = 600 1/ min
Tolerance = 5%
Pulse/rev = 8 (یعنی دیسک دارای ۸ سوراخ است)

در این حالت حدود بالا و پایین بصورت زیر بدست می آید:

| Comparison value | RPM | Frequency | Comparison value upper/lower limit |
|---|---|---|---|
| Lower limit | 600 1/min - 5%<br>= 570 1/min | 570 1/min / 60<br>= 9.5 Hz | 9.5 Hz x 8 (pulses/rev) = 76 Hz<br>Lower limit (in mHz): **76000**mHz |
| Upper limit | 600 1/min - 5%<br>= 630 1/min | 630 1/min / 60<br>= 10.5 Hz | 10.5 Hz x 8 (pulses/rev) = 84 Hz<br>Lower limit (in mHz): **84000**mHz |

### تنظیمات HWconfig

این تنظیمات طبق جدول زیر انجام می شود:

| Parameters | Input |
|---|---|
| Channel | 0 |
| Operating mode | Frequency measurement |
| Integration time | 1000 ms |
| Lower limit | 0 mHz |
| Upper limit | 60,000,000 mHz (CPU 314)<br>30,000,000 mHz (CPU 313)<br>10,000,000 mHz (CPU 312) |
| Output measured value | Direct |
| Signal evaluation | Pulse/Direction |
| Input function | No hardware gate used |
| Output reaction | Out of range |

### برنامه

OB100 برای راه اندازی و شرایط اولیه بکار رفته است. از OB1 فانکشن FB22 همراه با DB22 صدا زده شده و از FB22 فانکشن SFB48 همراه با DB26 فراخوان شده است.

```
DATA_BLOCK DB 26
TITLE =
AUTHOR : SIMATIC
NAME : DI_FREQ
VERSION : 1.0
 SFB 48
BEGIN
  LADDR := W#16#300;
  CHANNEL := 0;
  SW_GATE := FALSE;
  MAN_DO := FALSE;
  SET_DO := FALSE;
  JOB_REQ := FALSE;
  JOB_ID := W#16#0;
  JOB_VAL := L#0;
  STS_GATE := FALSE;
  STS_STRT := FALSE;
  STS_DO := FALSE;
  STS_C_DN := FALSE;
  STS_C_UP := FALSE;
  MEAS_VAL := L#0;
  COUNTVAL := L#0;
  JOB_DONE := FALSE;
  JOB_ERR := FALSE;
  JOB_STAT := W#16#0;
  RES00 := FALSE;
  RES01 := FALSE;
  RES02 := FALSE;
  STS_CMP := FALSE;
  RES04 := FALSE;
  STS_OFLW := FALSE;
  STS_UFLW := FALSE;
  RES07 := FALSE;
```

```
  JOB_OVAL := L#0;
  RES10 := FALSE;
  RES11 := FALSE;
  RES_STS := FALSE;
END_DATA_BLOCK
```

---

**FUNCTION_BLOCK FB 22**

```
TITLE =Speed monitoring
//The Task:
//The rate of revolution of a system's drive shaft is to be moitored. The rate of  revolution
//well as consistency of a variable speed range is monitored via proximity  switch.
//An error message is generated if the speed exceeds the permitted value.
//Sequence
//The Start signal triggers frequency measurement. You can specify the speed
//(SPEED), tolerance (TOLERANCE) and the pulses/rev (PULSE).
//The incoming pulses form the actual speed value (SPEED_VAL) and are
//compared with the calculated limit values. Bit STATUS_U is set if the value drops
//below the lower limit, bit STATUS_O is set if the value exceeds the upper limit..
AUTHOR : SIMATIC
NAME : SPEED
VERSION : 1.0

VAR
  START : BOOL ;    //Control signal: Freuquency Measurement ON
  RES_STS : BOOL ;  //Control signal: Reset Status
  TOGGLE : BOOL ;   //Toggle bit for pulse generation
  F_PULSE : BOOL ;  //Pos. Edge of the pulse period has expired
  T_PULSE : S5TIME  := S5T#250MS;   //Setpoint: Pulse period
  SPEED : INT  := 300;  //Setpoint: speed setpoint in RPM.
  TOLERANCE : INT  := 5;    //Setpoint: Tolerance in %
  PULSE : INT  := 1;    //Setpoint: Pulses/rev
  SPEED_VAL : DINT ;  //Actual value: actual speed
  STS_DO : BOOL ;   //Status: Speed out of range
  STATUS_U : BOOL ; //Status: Dropped below limit
  STATUS_O : BOOL ; //Status: Exceeded upper limit
  JOB_REQ : BOOL ;  //SFB Parameter: Level-controlled job initiation
  JOB_ID : WORD ;   //SFB Parameter: Control job
  JOB_VAL : DINT ;  //SFB Parameter: Control value
  MEAS_VAL : DINT ; //SFB Parameter: Actual measurement value
  COUNTVAL : DINT ; //SFB Parameter: Fetched counter status
  JOB_DONE : BOOL ; //SFB Parameter: Job done or running
  JOB_ERR : BOOL ;  //SFB Parameter: Job error
  JOB_STAT : WORD ; //SFB Parameter: Job error ID
  UFLW_VAL : DINT ; //calculated actual lower limit
  OFLW_VAL : DINT ; //calculated actual upper limit
  UFLW_LOAD : DINT ;    //Low limit of loaded value
  OFLW_LOAD : DINT ;    //Upper limit of loaded value
  START_JOB1 : BOOL ;   //Start JOB1 Load low limit
  START_JOB2 : BOOL ;   //Start JOB2 Load upper limit
END_VAR
BEGIN
NETWORK
TITLE =Pulse generation and simulation of inputs

//Pulse generation
//****************
    A    T    10; //Pulse period expired
    =    DB22.DBX    0.3; // Pos.edge Pulse period expired
    JCN   takt;
    AN    DB22.DBX    0.2; // Toggle Bit
    =    DB22.DBX    0.2;
```

```
takt: NOP   0;
    AN    DB22.DBX    0.3; // Pulse period Start
    L     DB22.DBW    2; // Setpoint Pulse period
    SD    T     10; // Timer Pulse period

// Simulation of inputs
//*************************
    A     DB22.DBX    0.2; // If Toggle bit=1
    =     Q    124.2; // then activate pulse A

NETWORK
TITLE =Calculate lower/upper limit

// calculate low limit
    L     100;
    L     DB22.DBW    6; // Tolerance in %
    -I    ;
    L     1000; // multiply by 1000 because specified in mHz
    *I    ;
    L     100;
    /D    ;
    L     DB22.DBW    4; // Speed setpoint in RPM.
    *D    ;
    L     DB22.DBW    8; // Pulses/rev
    *D    ;
    L     60; // Conversion to Hz
    /D    ;
    T     #UFLW_VAL; // calculated low limit in mHz

// calculate upper limit
    L     DB22.DBW    6; // Tolerance in %
    L     100;
    +I    ;
    L     1000; // multiply by 1000 because specified in mHz
    *I    ;
    L     100;
    /D    ;
    L     DB22.DBW    4; // Speed setpoint in RPM.
    *D    ;
    L     DB22.DBW    8; // Pulses/rev
    *D    ;
    L     60; // Conversion to Hz
    /D    ;
    T     #OFLW_VAL; // calculated upper limit in mHz
NETWORK
TITLE =Load lower/upper limit
//CAUTION !!!
//Note the order of loading the low/upper limit.
//Example:
//  old low limit =  5000 mHz
//  old upper limit  =  6000 mHz
//  new low limit =  8000 mHz
//  new upper limit   =  9000 mHz
//You must first load the 9000 mHz upper limit and then the low limit value.
//Otherwise, you will receive the error message "Low limit > Upper limit".
    A     #START_JOB1; // Transfer Start low limit
    JCN   M001;
    =     #JOB_REQ; // Job initiation
    L     1;
    T     #JOB_ID; // Control job 1=Load low limit
    L     #UFLW_VAL;
```

```
    T    #JOB_VAL; //Control value
    T    #UFLW_LOAD; //last transferred low limit

M001: A    #START_JOB2; //transfer Start upper limit
    JCN  M002;
    =    #JOB_REQ; // Job initiation
    L    2;
    T    #JOB_ID; // Control job 2=Load upper limit
    L    #OFLW_VAL;
    T    #JOB_VAL; //Control value
    T    #OFLW_LOAD; //last transferred upper limit

M002: A(   ; //Change of low limit???
    L    #UFLW_VAL; //actual low limit ...
    L    #UFLW_LOAD; // ... With transferred low limit
    <>D  ; //... Compare
    )    ;
    A(   ; // Low limit < Upper limit ???
    L    #UFLW_VAL; //actual low limit ...
    L    #OFLW_LOAD; //... With transferred upper limit
    <D   ; //... Compare
    )    ;
    AN   #JOB_REQ;
    =    #START_JOB1; // transfer low limit

    A(   ; //Change of upper limit???
    L    #OFLW_VAL; //actual upper limit ...
    L    #OFLW_LOAD; //... With transferred upper limit
    <>D  ; //... Compare
    )    ;
    A(   ; // Upper limit > Low limit ???
    L    #OFLW_VAL; //actual upper limit ...
    L    #UFLW_LOAD; // ... With transferred low limit
    >D   ; //... Compare
    )    ;
    AN   #JOB_REQ;
    AN   #START_JOB1;
    =    #START_JOB2; //transfer upper limit

NETWORK
TITLE =Call of SFB FREQUENC
    A    DB22.DBX   0.0;
    =    L     0.0;
    BLD  103;
    A    DB26.DBX   4.1;
    =    L     0.1;
    BLD  103;
    A    DB26.DBX   4.2;
    =    L     0.2;
    BLD  103;
    A    #JOB_REQ;
    =    L     0.3;
    BLD  103;
    CALL SFB  48 , DB   26 (
        LADDR          := W#16#300,
        CHANNEL        := 0,
        SW_GATE        := L     0.0,
        MAN_DO         := L     0.1,
        SET_DO         := L     0.2,
        JOB_REQ        := L     0.3,
        JOB_ID         := #JOB_ID,
```

```
        JOB_VAL          := #JOB_VAL,
        STS_DO           := #STS_DO,
        MEAS_VAL         := #MEAS_VAL,
        COUNTVAL         := #COUNTVAL,
        JOB_DONE         := #JOB_DONE,
        JOB_ERR          := #JOB_ERR,
        JOB_STAT         := #JOB_STAT);
    NOP   0;
NETWORK
TITLE =reset request bit / error evaluation
//Here you can evaluate the output parameters of the JOB_ERR SFBs
    NOP   0;
    A     #JOB_DONE;
    R     #JOB_REQ;
NETWORK
TITLE =calculate actual speed value

    L     #MEAS_VAL;
    L     60;
    *D    ;
    L     #PULSE;
    /D    ;
    L     1000;
    /D    ;
    T     #SPEED_VAL;
NETWORK
TITLE =status evaluation
    A     DB26.DBX   26.6;
    =     #STATUS_U;
    A     DB26.DBX   26.5;
    =     #STATUS_O;
    A     DB22.DBX    0.1;
    =     DB26.DBX   32.2;
END_FUNCTION_BLOCK
```

---

```
DATA_BLOCK DB 22
AUTHOR : SIMATIC
NAME : DI_SPEED
VERSION : 1.0
 FB 22
BEGIN
  START := FALSE;
  RES_STS := FALSE;
  TOGGLE := FALSE;
  F_PULSE := FALSE;
  T_PULSE := S5T#250MS;
  SPEED := 300;
  TOLERANCE := 5;
  PULSE := 1;
  SPEED_VAL := L#0;
  STS_DO := FALSE;
  STATUS_U := FALSE;
  STATUS_O := FALSE;
  JOB_REQ := FALSE;
  JOB_ID := W#16#0;
  JOB_VAL := L#0;
  MEAS_VAL := L#0;
  COUNTVAL := L#0;
  JOB_DONE := FALSE;
  JOB_ERR := FALSE;
  JOB_STAT := W#16#0;
```

```
  UFLW_VAL := L#0;
  OFLW_VAL := L#0;
  UFLW_LOAD := L#0;
  OFLW_LOAD := L#0;
  START_JOB1 := FALSE;
  START_JOB2 := FALSE;
END_DATA_BLOCK
```

**ORGANIZATION_BLOCK OB 1**

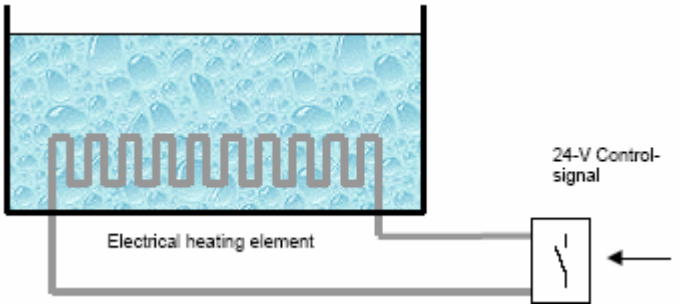
```
TITLE =Cycle Execution
VERSION : 1.0

VAR_TEMP
 L_BEARB : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Call of "Frequency 2 Speed monitoring"
    CALL FB    22 , DB    22 ;
    BE    ;
END_ORGANIZATION_BLOCK
```

**ORGANIZATION_BLOCK OB 100**

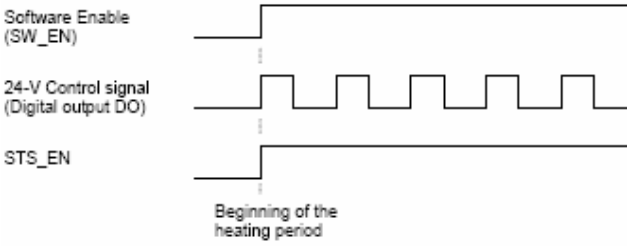
```
TITLE =Complete Restart
AUTHOR : SIMATIC
VERSION : 1.0

VAR_TEMP
END_VAR
BEGIN
NETWORK
TITLE =Sample SPEED: Fetch low limit
//Here, the default low limit of the frequency specified in HW Config is read out.
    CALL SFB   48 , DB    26 (
        CHANNEL                := 0,
        JOB_REQ                := TRUE,
        JOB_ID                 := W#16#81);

    L     DB26.DBD   28;
    T     DB22.DBD   42;
    CALL SFB   48 , DB    26 (
        CHANNEL                := 0,
        JOB_REQ                := FALSE);

NETWORK
TITLE =Sample SPEED: Fetch upper limit
//Here, the default upper limit of the frequency specified in HW Config is read out.

    CALL SFB   48 , DB    26 (
        CHANNEL                := 0,
        JOB_REQ                := TRUE,
        JOB_ID                 := W#16#82);
    L     DB26.DBD   28;
    T     DB22.DBD   46;
    CLR   ;
    =     DB26.DBX    4.1;
    =     DB26.DBX    4.2;
    =     DB26.DBX    4.3;
END_ORGANIZATION_BLOCK
```

**برنامه ۱۱: استفاده از PWM برای کنترل دما با CPU31xC و توسط SFB49**

در فرآیندی مانند شکل زیر مایع توسط المنت های الکتریکی گرم می شود. انرژی مورد نیاز توسط قطع و وصل کردن سوئیچ (مثلاً کنتاکتور) بدست می آید. سیگنال کنترلی ۲۴ ولت در خروجی CPU برای وسیله سوئیچینگ (کنتاکتور) بکار می رود. میزان گرما بستگی به پریود On/Off سیگنال کنترلی دارد. بدیهی است هر چقدر زمان On بودن بیشتر باشد دمای مایع بالاتر خواهد رفت.




باید دقت داشت که حداقل عرض Pulse/Pause در PWM از زمان پاسخ دهی وسیله سوئیچینگ و وسیله گرمایی بزرگتر باشد.




**تنظیمات Hwconfig**

| Parameters | Input |
|---|---|
| Channel | 0 |
| Operating mode | Pulse width modulation |
| Output format | Per mil |
| A Timebase | The resolution of all specified times is set to 1 ms |
| On delay | 0 ms The 24-V control signal is output instantaneously when SW_EN = 1 |
| Period | 10000 In the selected timebase equivalent to 10 s |
| Input function | No hardware gate used |
| Minimum pulse width | 500 ms Minimum pulse width In the selected timebase equivalent to 500 ms. |

**برنامه**

از OB1 فانکشن FB22 همراه با DB22 و در FB22 فانکشن SFB49 همراه با DB63 فراخوان می گردد.

```
DATA_BLOCK DB 36
TITLE =
AUTHOR : SIMATIC
NAME : DI_PULSE
VERSION : 1.0

 SFB 49
BEGIN
   LADDR := W#16#300;
   CHANNEL := 0;
   SW_EN := FALSE;
   MAN_DO := FALSE;
   SET_DO := FALSE;
   OUTP_VAL := 0;
   JOB_REQ := FALSE;
   JOB_ID := W#16#0;
   JOB_VAL := L#0;
   STS_EN := FALSE;
   STS_STRT := FALSE;
   STS_DO := FALSE;
   JOB_DONE := FALSE;
   JOB_ERR := FALSE;
   JOB_STAT := W#16#0;
   JOB_OVAL := L#0;
END_DATA_BLOCK
```

---

```
FUNCTION_BLOCK FB 32
TITLE =Heating a liquid
//An electrical heating element heats a liquid.The energy required by the heating
//element is applied with the help of a switching element (e.g. a contactor).
//The CPU 31xC generates a 24-V control signal at the digital output for the
//switching element. The heating element temperature depends on the length of
//the/
//on/off period of the 24-V control signal.
//Longer on times of the 24-V control signal increase the heating period and,
//accordingly, liquid temperature..
AUTHOR : SIMATIC
NAME : HEATING
VERSION : 1.0


VAR
  START : BOOL ;    //Control signal: Heating ON
  POWER : INT ; //Setpoint: Heating power in %
  PERIOD : DINT  := L#1000; //Setpoint: Period
  STS_DO : BOOL ;   //Status: Heater ON
  JOB_REQ : BOOL ;  //Transfer period
  OUTP_VAL : INT ;  //Actual output value
  PERIOD_LOAD : DINT ;  //loaded period
END_VAR
BEGIN
NETWORK
TITLE =Specify heating power

     L     DB32.DBW    2; //Heating power in %
     L     10; //multiply by 10
     *I    ;
     T     #OUTP_VAL; //Heating power in per mil
NETWORK
TITLE =load period
```

```
    L     #PERIOD_LOAD; //loaded period
    L     DB32.DBD    4; //Setpoint Period
    <>D   ;
    =     #JOB_REQ; // Job initiation
    T     #PERIOD_LOAD; //save transferred period
NETWORK
TITLE =Call of SFB PULSE

    A     DB32.DBX    0.0;
    =     L      0.0;
    BLD   103;
    A     #JOB_REQ;
    =     L      0.3;
    BLD   103;
    CALL SFB   49 , DB    36 (
        CHANNEL              := 0,
        SW_EN                := L      0.0,
        OUTP_VAL             := #OUTP_VAL,
        JOB_REQ              := L      0.3,
        JOB_ID               := W#16#1,
        JOB_VAL              := #PERIOD,
        STS_DO               := #STS_DO);

    NOP   0;
END_FUNCTION_BLOCK
```

---

**DATA_BLOCK DB 32**

```
AUTHOR : SIMATIC
NAME : DI_HEAT
VERSION : 1.0

 FB 32
BEGIN
  START := FALSE;
  POWER := 0;
  PERIOD := L#1000;
  STS_DO := FALSE;
  JOB_REQ := FALSE;
  OUTP_VAL := 0;
  PERIOD_LOAD := L#0;
END_DATA_BLOCK
```

---

**ORGANIZATION_BLOCK OB 1**

```
TITLE =Cycle Execution
VERSION : 1.0


VAR_TEMP
  L_BEARB : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Call of sample "PWM 2, Heating"

    CALL FB    32 , DB    32 ;
    BE    ;
END_ORGANIZATION_BLOCK
```

# ضمیمه ۸ - نحوه تبدیل برنامه S5 به S7

## نحوه تبدیل برنامه Step5 به Step7

برنامه های Step5 و Step7 تا حد زیادی با یکدیگر قابل تطبیق هستند بویژه اگر برنامه بصورت STL باشد. البته با انجام تبدیل همه قسمتهای برنامه Step5 به Step7 تبدیل نمیشوند بعنوان مثال دستوراتی مانند LIR و TIR که برای آدرس دهی مطلق بکار میروند تبدیل نشده یا فانکشنهایی مانند DO FW و DO DW که برای آدرس دهی غیر مستقیم بکار میروند فقط بخشی قابل تبدیل هستند. بهرحال در چنین مواردی کاربر باید پس از انجام تبدیل بصورت دستی اصلاحات لازم را در برنامه انجام دهد.

نکته دیگری که لازم است به آن توجه شود اینست که در صورت انجام بهینه سازی برای جایگزینی کارتها و CPU از خانواده S7 به جای S5 لازمست ملاحظات دیگری مد نظر باشد که در این جا مورد بحث ما نیست.

## پیش نیازهای تبدیل

قبل از تبدیل باید فایلهای زیر که مربوط به برنامه Step5 هستند روی کامپیوتر موجود باشند :

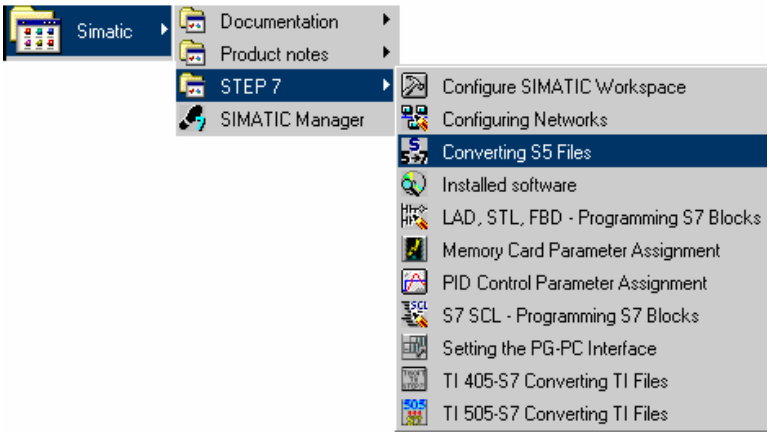
۱. فایل برنامه Step5 بصورت <filename>ST.S5D

۲. فایل برنامه Step5 بصورت <filename>XR.INI

در صورت نیاز به استفاده از اسامی سمبلیک فایل <filename>Z0.SEQ نیز لازم است.

## انجام تبدیل

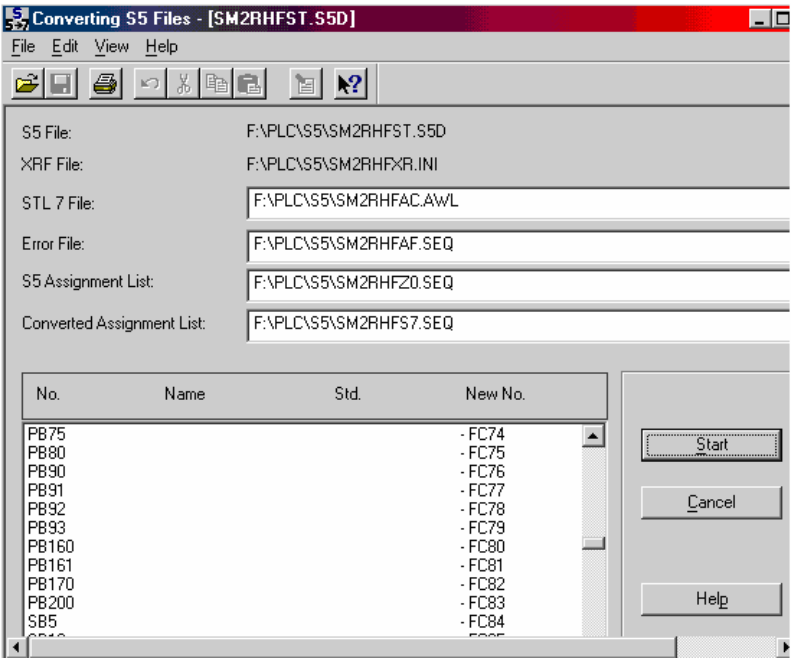
۱. برنامه Converting S5 File راکه در زیر برنامه های STEP7 مطابق شکل زیر موجود است را اجرا میکنیم:



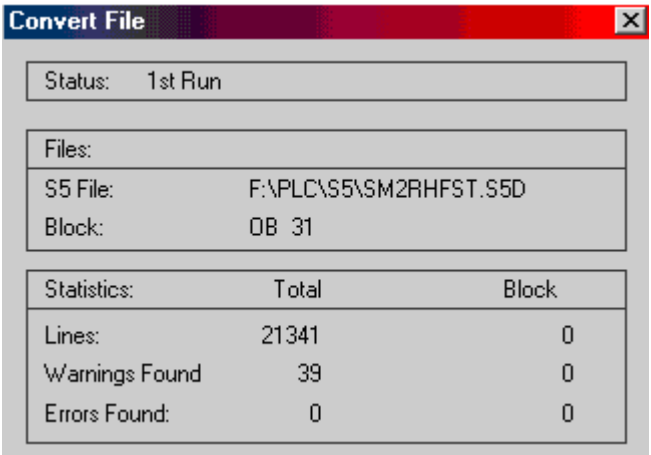
۲. پس از اجرای برنامه فوق با استفاده از منوی **File>Open** فایل <filename>ST.S5D را که قبلاً روی کامپیوتر آماده کرده ایم باز میکنیم.

۳. مانند شکل زیر برنامه علاوه بر فایل فوق فایل <filename>XR.INI را نیز شناسایی کرده و نام و مسیر فایل جدید STEP7 را تعیین میکند. کاربر میتواند در صورت لزوم این نام و آدرس را تغییر دهد.

۴. لیست بلاکهای قبلی همراه با نام جدید Step7 آنها در جدول ظاهر میگردد. همانطور که ملاحظه میشود بلاکهای PB و SB مربوط به Step5 در Step7 به FC تبدیل میشوند.



۵. روی Start کلیک کرده تا تبدیل شروع گردد. درطول تبدیل شکلی شبیه زیر ظاهر میگردد در واقع تبدیل در ۲ مرحله انجام میشود ابتدا برنامه به S5 Source و سپس به STL Source تبدیل میگردد. شکل زیر 1st Run را نشان میدهد.




۶. پس از اتمام تبدیل مرحله اول جدولی که در آن لیست Warning ها و Error ها نمایش داده شده ظاهر میگردد.جدول انتهای این ضمیمه راهنمایی لازم را در مورد Error ها ارائه داده است.

۷. فایل تبدیل شده S7 با پسوند AWL بوده و همراه آن ۲ فایل دیگر نیز با پسوند SEQ نیز بوجود می آید.

۸. پس از اتمام تبدیل در Simat Manager یک پروژه جدید تعریف کرده و در داخل پوشه Blocks پوشه Source را باز میکنیم. سپس از منوی **Insert>External Source** روی فایل با پسوند AWL فوق الذکر از آدرس تعیین شده کلیک میکنیم . فایل S7 به پوشه Source منتقل خواهد شد.

۹. فایل Source قابل انتقال به PLC نیست قبل از آن باید آنرا به بلاک تبدیل کرد برای اینکار ابتدا روی فایل Source کلیک کرده تا توسط برنامه LAD/STL/FBD باز شود. پس از آن روی **Compile** از منوی File کلیک میکنیم. لیست Warning ها و Errorهای مربوط به تبدیل Sourse به بلاک در پنجره ای ظاهر میگردد. لیست Error های این مرحله $2^{nd}$ Run در جدول زیر آمده است.

۱۰. Warning ها مانع تبدیل نیستند ولی با وجود Error برنامه قابل تبدیل به بلاک نیست از اینرو ابتدا باید آنها را برطرف کرد. روی هر Error که کلیک کنیم به سطر مربوطه پرش مینماید. پس از رفع Errorها مجدداً برنامه را Compile میکنیم. با این کار بلاکهای مربوطه در پوشه Blocks ساخته میشوند.

۱۱. پوشه بلاکها را باز کرده و چنانچه اصلاحاتی لازم بوده بصورت دستی انجام شود در آنها اعمال مینماییم.

| Error Message | Source | Meaning | Remedy |
|---|---|---|---|
| Absolute parameter does not match address | 1st run | Wrong address ID | Check the instruction |
| Bit access to T/C is no longer allowed (please check) | 2nd run | S5 program contains bit access to timers and counters | Check the STL program |
| Block not available | 1st run | Called block (FB, FX) missing or block is shown in the block list but it does not exist in the program file | Check the program structure |
| | 2nd run | Block is called that does not exist in the program file | Check whether the cross-reference list was specified, or check the program structure |
| CALL OB is not allowed | 2nd run | Calling OBs is not allowed in S7 | If necessary, use the statement CALL SFC |
| CALL SFC xy generating, please extend parameter list | 2nd run | Parameters for SFC missing | Complete the SFC parameter list |
| Command in block not allowed | 1st run | For example, jump within a program block | Check the instruction |
| Comment too long | 1st run | ErrorinS5file | Check the program file |
| Conversion error | 2nd run | BI without constant | Include a constant with the load instruction |
| Directory not available | 1st run | Program file does not contain any blocks | Check the program file |
| Error in macro file. Macro xy ignored | 2nd run | Macro error | Check the macro instruction |
| Error in parameter | 1st run | Error in the S5 program | Check the program file |
| File not found | general | Selected file does not exist | Check the program file |
| Invalid MC5 code was converted | 1st run | Conversion of an older S5 instruction | None |
| Invalid operator | 1st run | Operator in S5 file not known or cannot be converted | Replace the operator with the appropriate S7 instruction |
| Invalid operator, may be replaced by the instruction \"L P# formal parameter\" | 2nd run | The operator cannot be loaded into S7 in this form | You may have to use the specified instruction |
| Jump label cannot be generated | 2nd run | JUR instruction exceeds block limit | Correct the error in the S5 program |
| Label invalid | 1st run | Jump label contains invalid characters | Check the S5 file |
| Label undefined | 1st run | Jump label not defined in the preheader | Check the S5 file |
| Memory overflow in programming device (space problem) | 1st run | Not enough main memory | Delete files you no longer require in the main memory |
| No access rights | general | File is read-only | Clear the read-only attribute |
| No block name given | 1st run | Block name consists of only blanks | Enter a block name |
| Undefined command | 1st run | Invalid MC5/STL instruction | Correct the S5 program file |
| | 2nd run | Instruction does not exist in S7 | Edit a macro or replace the instruction with the appropriate S7 instruction sequence |
| Undefined formal parameter | 1st run | More parameters in calling block | Check the S5 program file |
| Write error on diskette | general | File is read-only or there is no space on the diskette | Clear the read-only attribute or delete unnecessary data |
| Wrong address g | 1st run | Address does not match instruction | Check the S5 source file |
| | 2nd run | Address does not match instruction | Modify the STL file |
| Wrong comment length | 1st run | ErrorinS5file | Check the program file |
| Wrong nesting depth | 1st run | End of bracketed expression incorrect | Check the nesting levels, correct the programming error |
| Wrong number of parameters | 1st run | Error in the S5 program | Check the program file |
| Wrong parameter type | 1st run | Error in the S5 program | Check the program file |

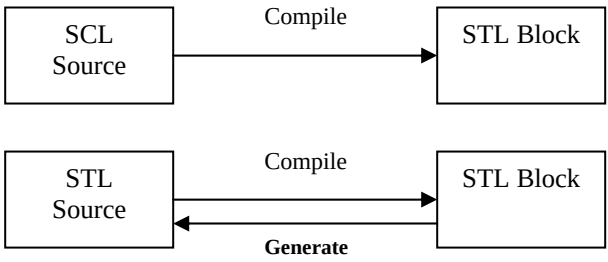
# ضمیمه ۹ – نحوه ایجاد STL Source

**مقدمه**

دربحث S7-SCL اشاره شد که علاوه بر فایل SCL Source میتوان فایل STL Source نیز ایجاد کرد .

STL Source دو تفاوت عمده با SCL Source دارد:

- دستورات در فایل STL Source بصورت STL نوشته میشوند نه بصورت SCL
- از STL Block میتوان STL Source ساخت در حالیکه برای SCL Source این کار امکان پذیر نیست. شکل زیر این موضوع را بهتر مورد مقایسه قرار داده است.




همانطور که مشاهده میشود برای STL Source تبدیل دو طرفه است . ممکن است این سوال به ذهن خواننده برسد که فایل STL Source چه مزیتی دارد و چه لزومی دارد از آن استفاده شود در پاسخ باید اشاره کرد که:

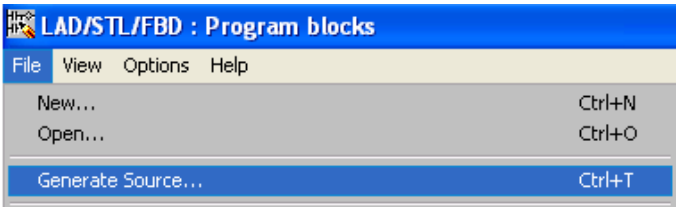
- نوشتن برنامه بصورت Source در محیط هر Editor امکان پذیر است
- لازم نیست بلاک های مختلف در فایلهای مختلف نوشته شوند. میتوان در برنامه Source همه آنها را زیر هم تعریف کرد.
- Source File را حتی اگر Syntax Error داشته باشد میتوان ذخیره کرد ولی بلاکها در صورتی ذخیره میشوند که فاقد اشکال فوق باشند. بدیهی است خطاهای برنامه Source در زمان کامپایل کردن گرفته میشود.
- در فایل Source میتوان از امکاناتی استفاده کرد که در STL Block موجود نیست بعنوان مثال توسط فایل STL Source میتوان هر بلاکی که به زبانهای LAD , FBD , STL نوشته شده باشد را Protect نمود.

## نحوه ایجاد STL Source

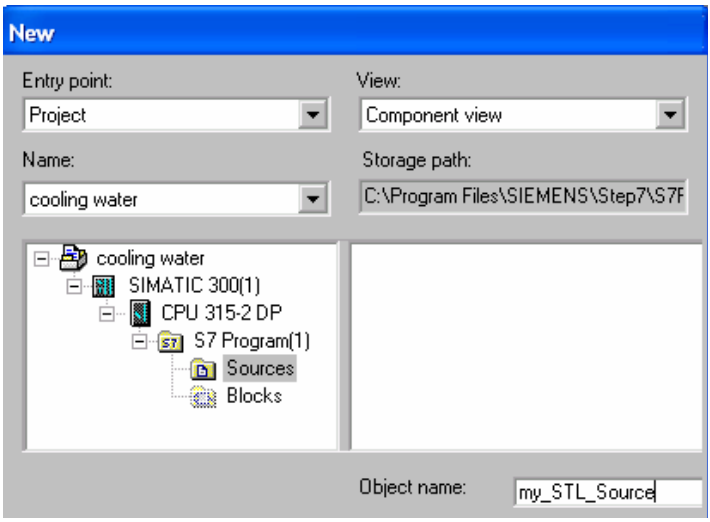
برای ایجاد STL Source قدم های زیر را بترتیب بردارید:

۱. برنامه LAD/STL/FBD را اجرا کنید . هیچ بلاکی در آن باز نباشد.

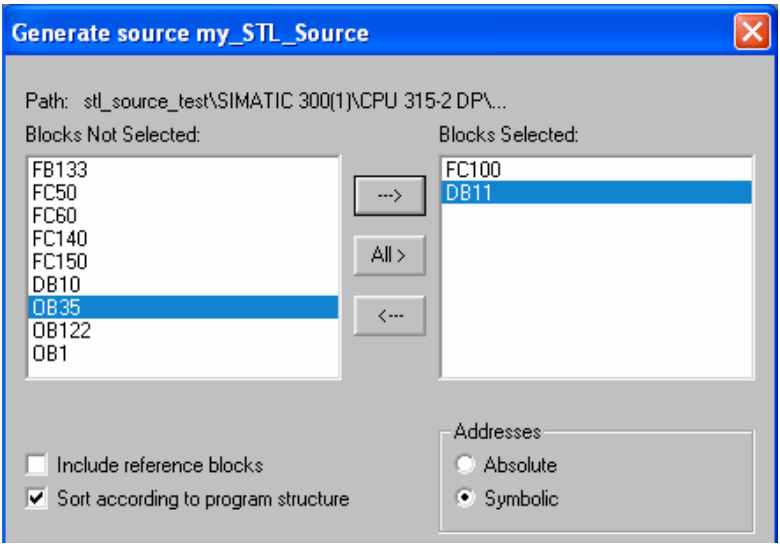
۲. از منوی File > Generate Source مانند شکل زیر استفاده کنید.




۳. پنجره ای مانند شکل زیر باز میشود مسیر را ادامه دهید تا به پوشه Source برسید و نام دلخواهی را برای STL Source وارد کنید



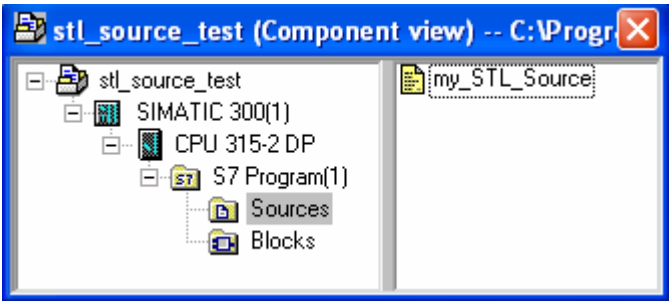

۴. با کلیک روی OK در پایین پنجره فوق ، پنجره دیگری مانند شکل بعد باز می شودکه در آن تمام بلاکهای موجود درپروژه بجز بلاکهای Protect شده لیست شده اند.




۵. میتوان از تمام بلاک ها یا برخی از آنها Source ساخت برای این منظور بلاکهای مورد نظر را توسط فلش --> به سمت راست منتقل میکنیم (شکل قبل را ببینید) یا با انتخاب All همه آنها را به سمت راست انتقال می دهیم.

۶. با انتخاب گزینه های پایین پنجره میتوان فایل Source را با آدرسهای مطلق یا سمبلیک ایجاد نمود.

۷. با کلیک روی OKدر پنجره قبل فایل Source ساخته میشود و میتوان با بازگشتن به Simatic Manager آنرا در پوشه Source مشاهده نمود.




۸. با دابل کلیک روی فایل Source می بینیم که توسط برنامه LAD/STL/FBD باز می شود. و در آن برنامه ای مانند صفحه بعد خواهیم دید. همانطور که مشاهده میشود ساختار این برنامه مشابه SCL بوده با این تفاوت که دستورات بصورت STL هستند.

۹. در این شرایط با کامپایل کردن STL Source در برنامه LAD/STL/FBD از طریق منوی File>Compile یا توسط آیکون کامپایل در Toolbar بالای پنجره میتوان مجدداً بلاک های STL را ایجاد نمود.

۱۰. اگر در برنامه Source در زیر کلمه Version عبارت Know_how_protect را بنویسیم و کامپایل کنیم خواهیم دید که بلاک ایجاد شده دارای قفل است.

۱۱. اگر در برنامه Source در سطر زیر نام دیتا بلاک کلیک کنیم و کلمه Read_only را نوشته کامپایل کنیم این ویژگی برای دیتا بلاک فعال خواهد شد

**تذکر ۱** : میتوان بدون Generate کردن STL Source ساخت و در آن مطابق همین الگو برنامه نوشت. برای این کار کافیست در پوشه Source در Simatic Manager راست کلیک کنیم و STL Source را انتخاب نماییم.

**تذکر ۲** : اگر از ویرایشگر بیرونی استفاده شده باشد لازم است در Simatic Manager فایل را با منوی Insert > External Source به پوشه Source وارد کنیم.

```
FUNCTION_BLOCK   FB1
TITLE =
AUTHOR : IJ
VERSION : 0.1

VAR_OUTPUT
  SSL_error : BOOL ;
END_VAR
VAR
  t_lvw : INT ;
END_VAR
VAR_TEMP
  v_max_l : INT ;
  v_min_l : INT ;
END_VAR
BEGIN
NETWORK
TITLE =
     A     "DO_SpLVw";
     AN    "DI_SpLHinten";
     JNB   _001;
     L     #t_lvw;
     L     "KomDB".allg.last_cycl;
     +I    ;
     T     #t_lvw;
_001: NOP   0;
END_FUNCTION_BLOCK
```

# فهرست منابع و مراجع

فهرست منابع و مراجعی که در تالیف این کتاب استفاده شده است :


- *System and Standard Functions for S7-300 , S7-400* — *Siemens*
- *CPU 31xC Technological Functions* — *Siemens*
- *Sample Programs For Technological Functions* — *Siemens*
- *S7-300 Programmable Controller Integrated Functions CPU312 IFM / 314 IFM* — *Siemens*
- *S7-Graph V5.3 For S7-300 / 400 Programming Sequential Control Systems* — *Siemens*
- *S7-SCL V5.3 For S7-300 / 400* — *Siemens*
- *Software Redundancy For Simatic S7-300 and S7-400* — *Siemens*
- *Automation System S7-400H Fault-tolerant Systems* — *Siemens*
- *Automation System - S7-300 Fail-Safe Signal Modules* — *Siemens*
- *Programmables Controllers S7-400F and S7-400FH Fail-Safe Systems* — *Siemens*
- *From S5 to S7* — *Siemens*
- *www.siemens.com/automation/service&support* — *Siemens*