UNIVERSITA' DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di laurea in ingegneria dell'automazione

PIANO QUALITA' E LIBRERIE SOFTWARE IN IMPIANTI DI LAMIAZIONE

Quality plan and software library in long roller mills

Laureando:

Francesco Menardi

Matricola:

520029

Relatore:

Augusto Ferrante

Anno accademico 2011/2012

Ansaldo Sistemi Industriali

RESULTS TO THE POWER OF THREE

METALS

INDICE

INTRODUZIONE

La qualità del software è un tema sempre più diffuso nel campo industriale, lo sviluppo di nuove applicazioni del software e l'estensione del suo utilizzo a funzioni essenziali e nevralgiche della nostra società sta comportando un attento riesame delle sue caratteristiche intrinseche per assicurarne l'assenza di errori, la corretta rispondenza all'uso e il buon funzionamento.
La qualità inoltre è fondamentale per la corretta gestione delle risorse e una più efficiente organizzazione del lavoro.
L'elaborato che vi presento vuole spiegare nel dettaglio cosa si intende con il termine "qualità" di un prodotto software e quali requisiti deve soddisfare per potersi definire tale.
In particolare obiettivo principale del mio elaborato è lo studio della qualità applicata alla produzione del software presso "*Ansaldo Sistemi Industriali"*, con la relativa stesura dei documenti e delle procedure necessari per riorganizzare la produzione osservando i principi fondamentali della qualità.
Quest'azienda di fama mondiale ha saputo crearsi negli anni un mercato molto ampio grazie alla varietà dei settori produttivi nei quali opera, risultando tuttora insieme a "*Siemens"* e "*ABB"*, una fra le multinazionali più potenti nel campo dell'automazione industriale.
In particolare il settore in cui ho avuto l'opportunità di lavorare è quello siderurgico, nella produzione cioè di software per macchine specifiche che si prestano alla lavorazione dell'acciaio.
Sempre nell'ambito dell'automazione industriale la mia esperienza come tirocinante in *Ansaldo* ha proseguito con un'ulteriore attività, quella cioè di ampliare e documentare le librerie del software, ovvero tutte quelle funzioni necessarie per la scrittura di un programma e l'avviamento di una macchina generica.
In particolare scopo di quest'attività era quello di standardizzare il più possibile il programma PLC in modo da renderlo più comprensibile favorendo così il *commissioning*, ovvero la messa in servizio degli impianti di laminazione, da parte di qualsiasi tecnico anche diverso da colui che si sia occupato della scrittura del programma stesso.

PRESENTAZIONE DELL' AZIENDA: "ANSALDO"

Presente sul mercato sin dal 1853, Ansaldo si è affermata nel corso degli anni come una delle aziende innovatrici nell'industria italiana.
Ansaldo tutt'oggi ha esteso la sua tecnologia e il suo raggio di competenze ricoprendo molteplici campi nell'economia mondiali, le aziende che oggi portano il nome "Ansaldo" sono:

- **Ansaldo Energia** - Azienda specializzata nella progettazione e costruzione di centrali elettriche chiavi in mano, nella costruzione di turbine a gas, turbine a vapore e alternatori per impieghi civili. Ha sede a Genova.
- **Ansaldo Ricerche** - Società nata nel 1987 dal raggruppamento di diversi enti di ricerca dell'Ansaldo e dalla NIRA (Nucleare Italiana Reattori Avanzati), si occupa di progetti di ricerca per conto delle società Ansaldo ed è impegnata in numerosi progetti di ricerca nazionali ed europei è componente del progetto Internazionale ITER (International Thermonuclear Experimental Reactor) sulla fusione nucleare.
- **Ansaldo Fuel Cells** - Creata nel 2003 da uno spin-off di Ansaldo Ricerche e partecipata da Iritech, si occupa di celle a combustibile.
- **Ansaldo Nucleare** - Creata nel 1989 dalle società Ansaldo Meccanico Nucleare e NIRA, nel 1999 è stata incorporata in Ansaldo Energia e nel 2005 è diventata una società separata controllata al 100% da Ansaldo Energia. Ha sede a Genova ed uffici a Mosca e a Bucarest.
- **AnsaldoBreda** - Nata dalla fusione di Ansaldo Trasporti e Breda Costruzioni Ferroviarie; ha sedi a Napoli, Pistoia, Reggio Calabria e Palermo.
- **Ansaldo STS** - ha sedi a Genova, Napoli, Piossasco e Tito.
- **Ansaldo Sistemi Industriali** - Ex società Finmeccanica ceduta ad un gruppo americano, porta ancora il nome Ansaldo anche se non più parte del gruppo industriale italiano. L'azienda è impegnata nella progettazione e costruzione di sistemi elettrici e di automazione, elettronica di potenza, motori e generatori per numerose applicazioni industriali. Con sede legale a Milano, l'azienda è presente anche a Genova, Montebello Vicentino, Monfalcone; mentre le sedi all'estero sono situate in Cina, Francia, Germania, Romania, Russia, Tailandia, Vietnam, USA.
- **Ansaldo Caldaie** con sede a Gioia del Colle (Ba), produzione di generatori di vapore e centro ricerca combustione.

ANSALDO SISTEMI INDUSTRIALI

Nata da un ramo dell'azienda, Ansaldo Sistemi Industriali (ASI) fornisce sistemi elettrici e di automazione, elettronica di potenza, motori e generatori per numerose applicazioni industriali, soddisfacendo centinaia di clienti in tutto il mondo.
Punto di forza della piattaforma di automazione ideata da Ansaldo Sistemi Industriali è il sistema di controllo integrato in tempo reale (ARTICS). ARTICS è la soluzione ideale di automazione per il controllo di processi e la raccolta in tempo reale dei dati, questa piattaforma integra e semplifica molti passi nell'architettura di sistemi di automazione per aiutare il controllo di apparecchiature meccaniche ed elettriche e per migliorare il processo di produzione.
ARTICS è stato progettato per superare anche i più severi requisiti in tempo reale, sviluppati per il controllo di processi veloci con tempi di ciclo nell'ordine dei millisecondi.
La vasta libreria di algoritmi ed gli avanzati applicativi per strumenti software permettono ad ASI di gestire l'automazione di base e le funzioni di controllo di processo, garantendo una maggiore automazione dei processi e della qualità.
Progettato con flessibilità, il sistema può gestire molteplici interfacce di comunicazione e al tempo stesso integrare diverse tipologie nella stessa unità, in questa maniera ci possono essere sia *slave* (per una maggiore integrazione in rete) che *master* (per I / O remoto connessione) riducendo cosi la complessità e il costo del cablaggio.
Grazie alle competenze di Project Management e all'esperienza nel campo delle applicazioni, l'azienda può vantare il massimo grado di qualità ed efficienza fornibile a livello mondiale.
Alcuni fra i settori di mercato più seguiti da Ansaldo Sistemi Industriali sono:

- **APPLICAZIONI NAVALI**

Ansaldo Sistemi Industriali è una delle aziende leader mondiali nella fornitura di soluzioni di propulsione e di generazione elettrica a bordo nave.
La competenza ingegneristica consente ai tecnici di progettare sistemi che rispondano alle esigenze più specifiche dei progettisti navali in termini di ingombro, riduzione dell'inquinamento e prestazioni.

- **GAS E PETROLIO**

Ansaldo Sistemi Industriali fornisce pacchetti elettrici integrati, motori ed azionamenti a tutto il settore petrolchimico dall'estrazione alla trasformazione, garantendo potenza, affidabilità e sicurezza. Ogni soluzione, progettata in base ai fabbisogni del ciclo di vita degli impianti e alle condizioni ambientali, si adatta perfettamente alle esigenze di ogni cliente grazie anche ad un programma completo di manutenzione predittiva ed assistenza

- **SIDERURGIA**

Ansaldo Sistemi Industriali ha progettato, installato e rinnovato più di 700 impianti in tutto il mondo.
La qualità, l'affidabilità e la robustezza dei sistemi e delle soluzioni di automazione industriale per la produzione di prodotti lunghi, piani e non ferrosi, consente all'azienda di essere uno fra i maggiori fornitori di sistemi di automazione completi.

- **GENERAL INDUSTRY**

Ansaldo Sistemi Industriali fornisce una vasta gamma di azionamenti, sistemi di controllo e di automazione, motori, *drives*, sistemi di distribuzione, strumentazione e supervisione, servizi ausiliari e montaggio.
La capacità di progettazione, di project management ed il servizio di assistenza post vendita, inclusa la manutenzione preventiva, rende questa azienda il partner ideale per mercati industriali quali: cemento, trattamento delle acque, gomma e plastica, movimentazione dei materiali, vetro, ceramica, carta ed ogni tipo di impianto a fune.

- **ENERGIA**

Da oltre 40 anni Ansaldo Sistemi Industriali fornisce in tutto il mondo convertitori a frequenza variabile, generatori, sistemi statici di eccitazione ed avviatori per ogni tipo di centrali elettriche, comprese quelle nucleari. Recentemente ha ampliato il pacchetto prodotti per la trasmissione e distribuzione di energia includendo sistemi HVDC e FACtS e la progettazione di sottostazioni chiavi in mano.
Inoltre, con l'esperienza nel POWER QUALITY ha la possibilità di assistere i clienti in settori emergenti come quello dell'energia rinnovabile: ad esempio progettare sistemi adeguati alle caratteristiche delle reti elettriche esistenti.

- **TRASPORTO**

Ansaldo Sistemi Industriali fornisce un'ampia gamma di sistemi elettrici e di controllo, motori, *drives*, sistemi di distribuzione, strumentazione e supervisione per imbarcazioni *all-electric* e per veicoli elettrici, volti alla riduzione dell'inquinamento legato al trasporto urbano. La passione di Ansaldo Sistemi Industriali per questo settore deriva dall'esperienza nella progettazione di sistemi di automazione e controllo funi in tutto il mondo.

LA QUALITA' DEL SOFTWARE

Il concetto che sta alla base della costruzione di un qualsiasi tipo di software di qualità è riassumibile in una frase: ... *un programma che non funziona è sicuramente errato; ma un programma che funziona non è necessariamente corretto* ... (M. Jackson) .
Infatti della produzione di software normalmente si riscontrano i seguenti fatti:

- Non conformità nei tempi di consegna.
- Non conformità di quanto realizzato con quanto richiesto.
- In generale i costi superano i preventivi, dovuti ad una errata valutazione dello sforzo necessario per revisioni, *testing*, correzioni e assistenza.

Per capire l'origine di questo problema, è necessario considerare le caratteristiche della produzione negli altri settori merceologici, dove in genere esiste un processo produttivo ben strutturato, caratterizzato da:

- Realizzazione del prodotto = progettazione + produzione + controllo
- Progettazione basata su tecniche stabili, ben conosciute e condivise.
- Produzione composta da processi ripetitivi e di tipo continuo
- Controllo basato su processi stabili
- Assenza di rivoluzioni tecnologiche

Questa metodologia produttiva non si applica anche al software.
Nel caso del software, infatti, le particolarità sono, se possibile, ancora più evidenti.
Progettare un sistema di qualità per la produzione del software significa avere molto chiari i concetti e le principali filosofie di conduzione di progetti e di traduzione in attività di progettazione e sviluppo.
La qualità deve essere costruita passo per passo durante tutte le fasi del processo di produzione.
E' necessario quindi analizzare i processi aziendali, rivisitandoli in funzione dei requisiti previsti dalla norma.

Il concetto di "qualità" del software

Tradizionalmente, i parametri (o fattori) rispetto ai quali si può misurare o definire la qualità del software vengono classificati in due famiglie: parametri esterni e parametri interni.
I primi si riferiscono alla qualità del software così com'è percepita dai suoi utenti, e includono:

1. Correttezza.
2. Affidabilità
3. Robustezza
4. Efficienza
5. Usabilità
6. Ecocompatibilitá
7. Scalabitlità

I secondi si riferiscono alla qualità del software così come è percepita dagli sviluppatori, e includono:

8. Verificabilità
9. Manutenibilità
10. Riparabilità
11. Evolvibilità
12. Riusabilità
13. Portabilità

PARAMETRI ESTERNI DI QUALITA’

- **Correttezza**

Un programma o sistema software si dice corretto se si comporta esattamente secondo quanto previsto dalla sua specifica dei requisiti. In termini più informali, un sistema è corretto se fa esattamente quello che è stato progettato per fare. Nell'ingegneria del software "classica" (anni ’60 -‘70), lo sviluppo di software corretto veniva universalmente considerato l'obiettivo *predominante*, a cui praticamente tutti gli altri parametri di qualità erano finalizzati. Molti moderni modelli di processo software, viceversa, escludono la possibilità di sviluppare rispetto a una *specifica* stabilita a priori in modo definitivo e non ambiguo, e quindi, indirettamente, negano alla *correttezza* un valore pratico significativo. In ogni caso, la correttezza è una qualità *assoluta* (un sistema *è* corretto o *non lo è*) e sostanzialmente impossibile da misurare (verificare); non è possibile *stabilire con certezza* che un sistema sia corretto.

Nelle metodologie di sviluppo e progettazione agili, si parla piuttosto di "soddisfazione del cliente", intendendo con questo che i requisiti iniziali potrebbero essere stati modificati, ma con il consenso e a piena soddisfazione del committente.

- **Affidabilità**

Un sistema è tanto più affidabile quanto più raramente, durante l'uso del sistema, si manifestano malfunzionamenti. Una definizione più specifica, ma non universalmente riconosciuta, è basata sul concetto di MTBF (*Mean Time Between Failure*, il tempo medio che intercorre fra due fallimenti successivi). In termini invece più vaghi (ma probabilmente più significativi), si può anche dire che l'affidabilità è la misura in cui l'utente può "fidarsi" del software; questa definizione tiene conto, in particolare, del fatto che malfunzionamenti *gravi* si considerano solitamente più influenti, nella valutazione dell'affidabilità, di errori minori. In questo senso, il modo in cui si intende il termine affidabilità può variare in funzione del tipo di utente (per esempio della sua capacità di adattarsi) e del tipo di applicazione (per esempio in funzione della sua criticità). Poiché l'affidabilità è concettualmente misurabile (una volta specificato un particolare modello, per esempio MTBF o una variante pesata in funzione della gravità dei fallimenti), questo parametro viene spesso considerato la controparte "realistica" della correttezza. L'industria del software moderna, su applicazioni non critiche, tende a considerare economicamente vantaggioso il rilascio di prodotti con un tasso iniziale di malfunzionamenti piuttosto elevato, ma dotati di meccanismi di caricamento periodico di *patch* (per esempio via Internet) attraverso cui gli errori vengono corretti via via che vengono trovati dall'utenza e segnalati al produttore.

- **Robustezza**

In termini generali, la robustezza di un sistema è la misura in cui il sistema si comporta in modo *ragionevole* in situazioni impreviste, non contemplate dalle specifiche. Situazioni di questo tipo in genere riguardano errori ed eccezioni di varia natura (dati di input scorretti, fallimenti di componenti software o hardware esterni al sistema e interagenti con esso, e così via). Anche in questo caso, l'idea intuitiva della robustezza implica certamente considerazioni di valore sugli effetti dannosi che il sistema o l'utente subiscono se il sistema reagisce in modo "irragionevole" a situazioni impreviste.

- **Efficienza**

Un sistema è efficiente se usa memoria, CPU e altre risorse in modo proporzionato ai servizi che svolge, ovvero senza sprechi. Il termine prestazioni ha un significato correlato più specifico; le prestazioni, infatti, sono da considerarsi come uno degli elementi che potrebbe essere specificato dai requisiti (si parla in questo caso di requisiti non funzionali). Efficienza e prestazioni sono difficilmente predicibili e quindi non raramente vengono prese in considerazione solo a sistema realizzato; tuttavia, gli interventi *a posteriori* per migliorare il comportamento del sistema rispetto a questi parametri sono spesso estremamente difficili e costosi. Fra i

modelli utilizzati per misurare (o specificare) l'efficienza di un sistema si possono citare quelli basati sulla complessità algoritmica, le misure sul campo, le misure su modelli matematici o modelli di simulazione.

- **Usabilità**

Un sistema è facile da usare se un essere umano lo reputa tale.
È una qualità soggettiva:

- Dipende dal contesto
- Dipende dall'esperienza

L'interfaccia utente interviene molto sull'amichevolezza di un'applicazione, ma anche in questo caso è la formazione e la cultura dell'utente a giudicare tale caratteristica.

Al di fuori di una visione macchino-centrica, va sottolineata l'esistenza di principi condivisi che permettono di valutare il livello di usabilità di un'applicazione, indipendentemente da fattori soggettivi. Valga come riferimento il noto studio di Jakob Nielsen[1], e quanto espresso dallo standard ISO 9241-10 sugli *ergonomic requirement*:

- **Ecompatibilità**

Un sistema è ecocompatibile se tiene in conto nel suo disegno l'impatto del suo esercizio sull'ambiente che lo circonda. L'ecocompatibilitá equivale all'efficienza del sistema ambiente di cui l'essere umano è parte.

- **Scalabilità**

Un sistema è scalabile se può essere adattato a diversi contesti con forti differenze di complessità (per esempio database molto piccoli o molto grandi) senza che questo richieda la riprogettazione dello stesso sistema. Solitamente, si richiede che le prestazioni di un sistema possano essere aumentate "semplicemente" fornendo al sistema stesso maggiori risorse di calcolo (processori più potenti, maggiori quantità di memoria, sistemi di memoria di massa più capienti o più veloci, e così via).

PARAMETRI INTERNI DI QUALITA'

- **Verificabilità**

Un sistema è verificabile se le sue proprietà di correttezza e affidabilità sono facili da verificare.
Per aumentare il grado di verificabilità si fa uso di:

- tecniche di progettazione modulare
- opportuni linguaggi di programmazione
- software monitor

In alcuni casi diventa una qualità esterna (*Software safety critical*).

- **Manutenibilità**

Facilità di apportare modifiche a sistema realizzato. In origine si pensava che la manutenzione corrispondesse solo al bug fixing, ma oggi la situazione è più complessa; la manutenzione riguarda infatti ogni miglioramento del software e andrebbe indicata più precisamente come evoluzione del software. Ormai il 60% dei costi dipende proprio dalla manutenzione. Per analizzare questi costi occorre suddividere in manutenzione:

- Correttiva
- Adattativa
- Perfettiva

La manutenzione correttiva

- Elimina gli errori presenti sin dall'inizio
- Elimina gli errori introdotti da precedenti interventi di manutenzione
- Rappresenta il 20% del totale della manutenzione

La manutenzione adattativa:

- Modifiche a seguito di cambiamenti nell'ambiente
- Cambiamenti nell'Hardware, nel Sistema operativo, etc.
- 20% del totale

La manutenzione perfettiva

- Modifiche per migliorare le qualità del software
- Introduzione di nuove funzionalità
- Miglioramento delle funzionalità esistenti
- È la parte più consistente (circa il 60% del totale)

L'ultima parte è così consistente soprattutto per il *legacy software* che non può essere sostituito a meno di ingenti spese, per cui viene migliorato e adattato. La manutenibilità dipende da due aspetti

- Riparabilità per indicare ciò che consente di eliminare difetti
- Evolvibilità per indicare ciò che consente l'implementazione di nuovi requisiti

- **Riparabilità**

Un sistema è riparabile se la correzione degli errori è poco faticosa. È una proprietà fondamentale di molti prodotti ingegneristici (automobili, computer, ...). Per questi prodotti la tecnica usata è quella di abbassare il prezzo e di favorire la sostituzione piuttosto che la riparazione. Perché ciò possa avvenire si utilizzano delle parti standard che possono essere cambiate con facilità. Nel software la situazione è diversa.

- Non c'è usura (parti meccaniche);

- Non esistono componenti standard;
- Il costo non dipende dai pezzi ma solo dalla mano d'opera necessaria.

La riparabilità si persegue attraverso la modularizzazione. Non conta tanto il numero, ma piuttosto il come sono organizzati tra loro e al loro interno.

- **Evolvibilità**

Il software, a differenza di altri prodotti ingegneristici, è teoricamente facilmente modificabile. Tuttavia:

- Non si fanno mai studi di analisi di fattibilità approfonditi.
- Non si progettano le modifiche

Peggio ancora, i cambiamenti effettuati non sempre sono documentati per cui le specifiche non vengono aggiornate e ciò rende i cambiamenti futuri difficili da compiere. È necessario prevedere sin dall'inizio che il software può evolvere e progettare tenendo in mente questa evoluzione per sfruttare al meglio i costi sostenuti in passato.

- **Riusabilità**

Affine all'evolvibilità. Tuttavia si ha nell'evolvibilità una modifica per realizzare una nuova versione, mentre nella riusabilità si usano parti di sistema per realizzare un prodotto diverso. Inizialmente si pensava al riuso del software. Oggi si riusa tutto:

- Specifiche
- Progetti
- Collaudo

Indica il livello di maturazione della disciplina. Vantaggi:

- Diminuzione dei costi
- Aumento della affidabilità

Come fare per rendere i diversi manufatti riusabili?

- Tecniche di progettazione
- Tecniche di specifica
- Metodologie
- Standardizzazione del processo di sviluppo

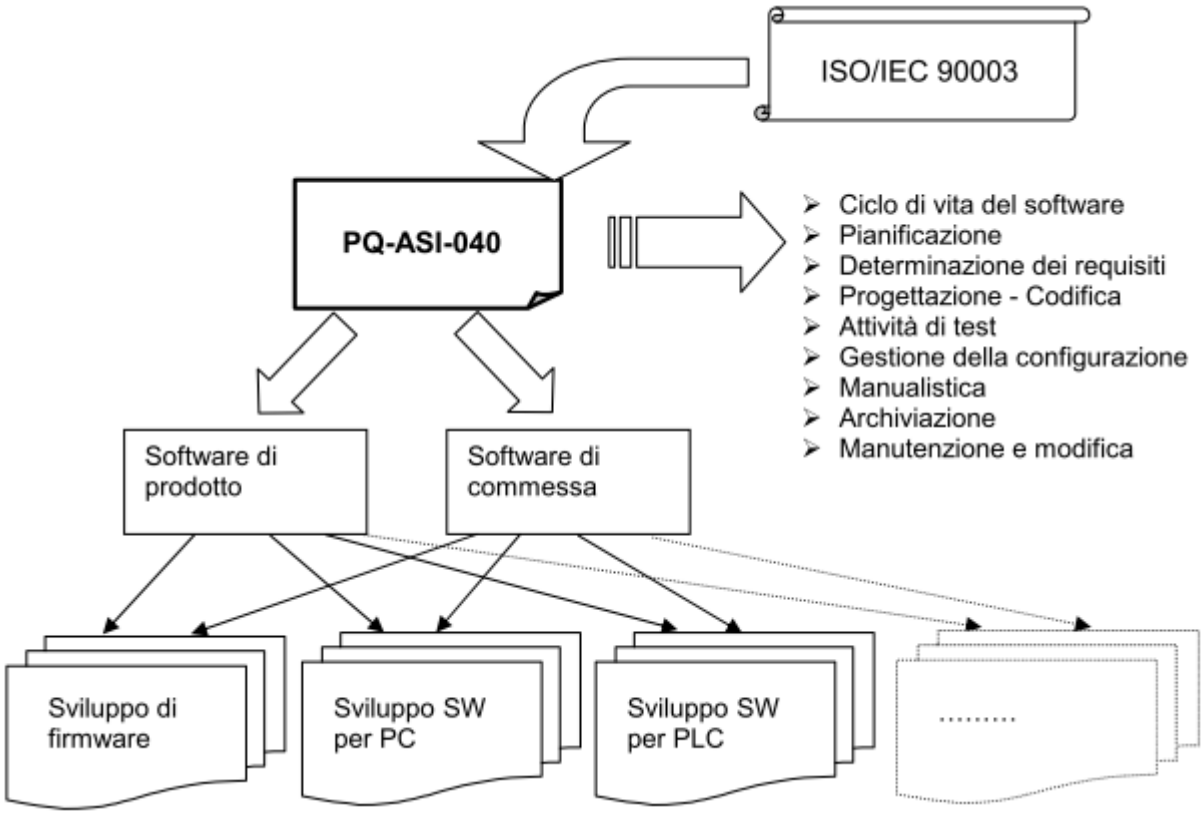
- **Portabilità**

Un sistema è portabile se è in grado di funzionare in ambienti diversi. È diventato un aspetto fondamentale perché consente di avere vantaggi economici, in quanto si possono ammortizzare i costi trasportando l'applicazione in diversi ambienti. Nel caso delle applicazioni web è la chiave di volta. Si usano strumenti e tecniche appositamente pensate per dare luogo ad oggetti portabili.

LA QUALITA' DEL SOFTWARE IN ANSALDO SISTEMI INDUSTRIALI

"Ansaldo Sistemi Industriali" è una realtà molto grande in cui l'organizzazione e i parametri di qualità acquisiscono se possibile significato ancor più importante.
Il mio compito come stagista è stato quello di carpire le condizioni e i requisiti fondamentali che un software di qualità deve avere, è sostanzialmente creare una procedura formale di produzione aziendale del software relativa al settore siderurgico, ovvero alla realizzazione di impianti di laminazione.

Nello schema sottostante è riportata la struttura della documentazione che regola lo sviluppo del software:



I prodotti software si dividono in due categorie principali: software di prodotto e software di commessa. Ciascuno di essi è destinato ad un diverso ambiente hardware e prevede diversi ambienti di sviluppo ed operativi. Un elenco indicativo non esaustivo delle diverse tipologie di prodotti software è il seguente:

- Firmware
- Software per PLC
- Software per PC

Per ciascuna delle tipologie elencate sono o saranno definiti uno o più documenti di dettaglio (Istruzioni Operative). L'elenco dei vari documenti è riportato in Allegato 1.

Processi di sviluppo del software

I processi per lo sviluppo, la gestione operativa e la manutenzione del software sono identificati e descritti nella presente procedura e nelle istruzioni operative collegate.

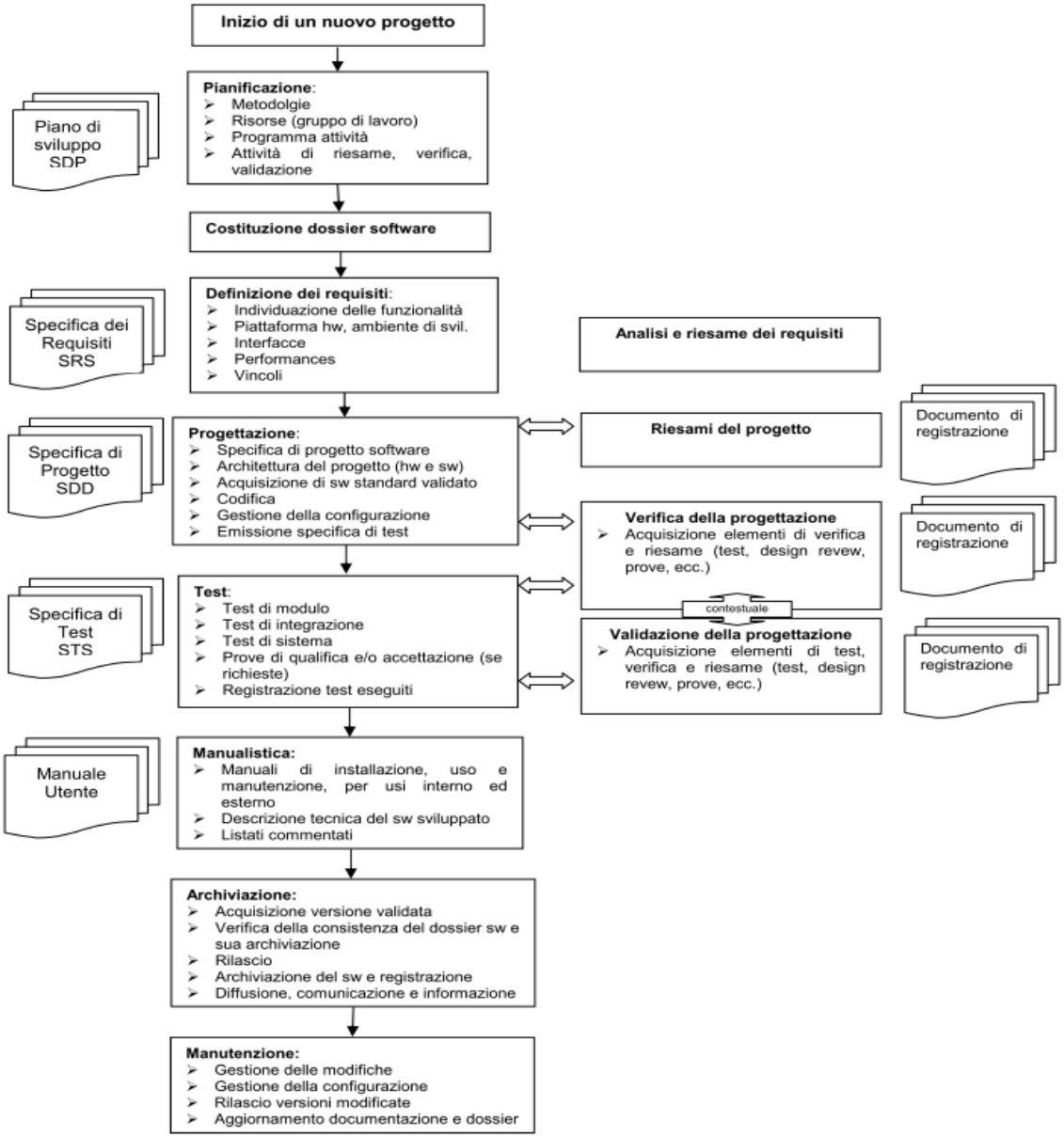
Processi in outsourcing

Alcuni dei processi di sviluppo software si svolgono, totalmente o parzialmente, all'esterno, presso fornitori qualificati.
In ogni caso le attività di sviluppo dovranno essere gestite e controllate in accordo alla presente procedura e documenti collegati, sotto la responsabilità del responsabile di progetto. Le attività di sviluppo svolte dal fornitore dovranno essere periodicamente monitorate rispetto alla pianificazione emessa; in particolare le attività di riesame e verifica devono essere svolte congiuntamente a personale di Ansaldo Sistemi Industriali, mentre la validazione sarà sempre di competenza esclusiva dell'azienda.
Nel caso il software sviluppato all'esterno debba essere integrato con altro software sviluppato internamente, sarà indispensabile fornire informazioni sulla gestione della configurazione e sulla documentazione richiesta.

REALIZZAZIONE DEL SOFTWARE

La produzione di software è da considerarsi un processo articolato composto da più fasi ed attività.
L'ingegneria del software, storicamente, esige l'inquadramento delle attività fondamentali di gestione della vita di un software formulando degli schemi, detti in gergo tecnico "cicli di vita".

Ciclo di vita del software



Il suddetto Ciclo di vita del software in Ansaldo Sistemi Industriali si articola sviluppando principalmente le seguente fasi, che portano alla produzione di vari documenti tecnici chiamati "specifiche"(ad esempio Specifica di Progetto, Specifica dei Requisiti..):

- Pianificazione
- Definizione, analisi e riesame dei requisiti
- Progettazione preliminare e di dettaglio e relative verifiche
- Test del software (di modulo, di integrazione e di sistema)
- Archiviazione e rilascio
- Manutenzione

Pianificazione delle attività

Le attività di sviluppo del software, come tutte le attività di progettazione e produzione, devono essere effettuate in modo controllato al fine di prevenire o minimizzare il verificarsi di problemi. È necessario, pertanto, identificare le metodologie adottate, i gruppi di lavoro e le risorse (in termini di risorse umane, macchine e strumenti), identificare le personalizzazioni da attuare per il progetto nel rispetto degli standard, definire un piano temporale di sviluppo che includa le attività di riesame, verifica e validazione. La pianificazione dello sviluppo del software a livello di progetto/prodotto costituisce, pertanto, l'applicazione del presente Piano di Qualità del Software e del modello del ciclo di vita a determinati progetti, commesse, prodotti o contratti.
La pianificazione della progettazione e dello sviluppo può rientrare nell'ambito di un Piano Qualità di commessa o può far parte del programma delle attività di ingegneria nel caso di sviluppo di software applicativo di commessa. Negli altri casi, quando viene emesso come documento dedicato a se stante, si identifica come Piano di Sviluppo delle attività Software (SDP).
Il documento deve essere disponibile all'interno del dossier software, indipendentemente dalla forma.
La pianificazione (che include la programmazione) delle attività viene aggiornata con l'avanzamento della progettazione e dello sviluppo. La responsabilità della pianificazione dipende dalla struttura organizzativa.
Il documento formale per la pianificazione delle attività di sviluppo del software è disponibile in allegato 1A (pag.36).

Costituzione del dossier software

Il Responsabile di Progetto software, il quale è formalmente incaricato mediante la consegna della "scheda di incarico" visibile in allegato B (pag.43), ha il compito di costituire il dossier software relativo al progetto assegnatogli. Tale dossier è una raccolta ordinata di tutta la documentazione di input ed output relativa allo sviluppo (piani, specifiche, registrazioni, manualistica, ecc.). Deve contenere il software sviluppato o, in alternativa, i riferimenti necessari per poterlo recuperare se diversamente archiviato.
In allegato 1C (pag.45) è illustrata la procedura di costituzione del dossier software da parte del responsabile di progetto.

Determinazione dei requisiti relativi al prodotto

I requisiti devono essere sempre definiti per qualunque tipologia di prodotto software. A seconda dei casi i requisiti possono essere forniti dal cliente, possono essere sviluppati internamente o essere sviluppati congiuntamente. La conseguente formalizzazione, pertanto, può assumere forme diverse che vanno dalla Specifica Tecnica contrattuale, al verbale di riunione, alla documentazione descrittiva dei requisiti di un progetto esistente, a semplice corrispondenza, ecc.
Qualora il software sia sviluppato come parte di un contratto, lo sviluppo dei requisiti spesso avviene in collaborazione con il cliente al fine di prevenire incomprensioni. In taluni casi, tuttavia, i requisiti possono non essere definiti completamente al momento dell'accettazione del contratto ed alcuni possono essere sviluppati in seguito. Le modifiche ai requisiti devono essere tenute sotto controllo in quanto possono portare a varianti contrattuali.
I requisiti possono includere anche caratteristiche quali: funzionalità, affidabilità, usabilità, efficienza, manutenibilità e portabilità. In generale, comunque, i requisiti devono essere espressi in modo chiaro e senza ambiguità. Il documento, in una qualunque delle sue forme, assume l'identificazione di Specifica dei Requisiti del Software (SRS) e, qualora venga sviluppato interamente, può essere utilizzato il documento standard predisposto a tale scopo.
Nel caso i requisiti si trovino in più documenti è necessario che la relativa raccolta sia identificabile come specifica dei requisiti.

Analisi e riesame dei requisiti

I requisiti, comunque siano definiti, devono essere analizzati e riesaminati per assicurare la loro completezza e definizione. Nel caso di progetti complessi il riesame viene fatto in modo interfunzionale nell'ambito della stessa ingegneria e dei reparti interessati.

In fase di riesame è importante tener conto anche di eventuali fattori di rischio quali, ad esempio, aspetti legati a sicurezza e riservatezza, capacità dell'azienda, innovazioni tecnologiche, scarsa precisione nella definizione dei requisiti da parte del cliente, ecc.
La specifica dei requisiti standard può essere anche utilizzata come check-list per effettuare il riesame dei requisiti. La fase di riesame dei requisiti deve essere sempre registrata e tale registrazione può essere attestata dalle revisioni della Specifica dei Requisiti o da verbali di riunione.

Progettazione preliminare e di dettaglio – Codifica

La fase di progettazione può essere suddivisa in una progettazione preliminare e in una progettazione di dettaglio. La progettazione preliminare inizia con un'analisi del sistema da progettare per individuare eventuali sottosistemi ed i requisiti di interfaccia tra gli stessi. In questa fase viene definita l'architettura del progetto (hardware e software).
La progettazione di dettaglio dei sottosistemi prosegue identificando i vari moduli (componenti, funzionalità, ecc.) che li compongono. Ove possibile vengono identificati moduli già esistenti utilizzabili, purché si tratti di software precedentemente validato. Nel caso di applicazioni a commessa si può trattare di funzioni facenti parte di librerie standard, adeguatamente descritte e validate in applicazioni precedenti o con test dedicati. Nel caso di sviluppo prodotto si può trattare di moduli già utilizzati e validati in versioni precedenti e/o in prodotti simili.
Il documento risultante dalla fase di progettazione di dettaglio è la Specifica di Progetto, dalla quale si passa alla fase implementativa di codifica. Il documento assume l'identificazione di Specifica di Progetto del Software (SDD) e, qualora venga sviluppato interamente, può essere utilizzato il documento standard predisposto a tale scopo (SDD001).
Nel caso lo sviluppo del software sia limitato a personalizzazioni per commessa di software standard (librerie) o di prodotto, la specifica di progetto può ridursi alla sola descrizione dell'architettura adottata, così come definita durante la progettazione preliminare.
La fase di codifica e tutte le attività correlate vengono sviluppate con diverse metodologie, linguaggi di programmazione e strumenti e sono tali da non poter essere specificate nel presente documento. Regole particolari, quando esistenti, verranno meglio descritte in documenti di livello inferiore.

Emissione specifica dei test

Contestualmente alla progettazione o immediatamente a valle della stessa è prevista la pianificazione e la definizione delle attività di test. La conseguente specifica dei test definisce i prodotti software oggetto delle prove, gli obiettivi e la tipologia delle prove, l'ambiente e la configurazione di prova, componenti e funzioni da sottoporre a prova, modalità di esecuzione e criteri di accettazione, modalità ed il dettaglio delle relative registrazioni.
Ciò si realizza mediante la stesura di una specifica nella sua versione preliminare che assume, per l'emissione finale, l'identificativo di Specifica di Test (STS).
In allegato 1F (pag. 54) è riportato il documento ufficiale per l'emissione della specifica dei test.

Approvvigionamento di prodotti software

Si tratta di prodotti software disponibili per l'acquisto 6 e l'uso senza la necessità di eseguire attività di sviluppo, compresi il software gratuito e gli strumenti di sviluppo "*open-source*" (COTS).
L'acquisizione e la loro integrazione nei prodotti e sistemi ASI deve considerare gli aspetti legati a:

- gestione e trasferimento delle licenze d'uso
- manutenzione dei prodotti
- servizio di assistenza (help desk)
- servizi di supporto al cliente (preoccupandosi della continuità del servizio sul prodotto acquistato a seguito di rilasci successivi).

Riesame della progettazione

Analogamente a qualsiasi altra attività di progettazione, anche nella progettazione del software, in fasi opportune, devono essere pianificati ed effettuati riesami al fine di valutare la capacità della progettazione e dello sviluppo di ottemperare ai requisiti specificati, di individuare i problemi e proporne le relative soluzioni.
Il livello di formalità e di rigore delle attività di riesame è proporzionale alla complessità del prodotto/progetto, ai requisiti di qualità ed al livello di rischio associato. Il riesame viene eseguito secondo piani stabiliti a priori e definiti in funzione della tipologia di prodotto o progetto. In tali piani dovrebbero essere definiti:

- cosa deve essere riesaminato, quando ed il tipo di riesame;
- quali responsabili di enti e/o funzioni e quali componenti del gruppo di progetto devono essere coinvolti;
- se è previsto un riesame con il cliente.

Registrazioni dei riesami devono essere prodotte nella forma ritenuta più appropriata (verbali di riunione, check-list, ecc.) e conservate. Le registrazioni devono contenere indicazione delle azioni necessarie e del conseguente piano di implementazione; il Resp. di Progetto ha il compito di assicurare la loro realizzazione.

Verifica della progettazione

Lo scopo delle attività di verifica è quello di assicurare che gli elementi in uscita dalle attività di progettazione e sviluppo (documenti e codice) siano conformi con i requisiti in ingresso. La verifica, pertanto, può avvenire sia mediante un esame critico dei documenti associati alle varie fasi della progettazione, sia mediante analisi, prove e dimostrazioni effettuate direttamente sul codice realizzato, inclusa la realizzazione di prototipi e ambienti di simulazione. Tipicamente, la verifica corrisponde alla fase di test del sistema software complessivo delle sue interazioni con il mondo esterno (come descritto al par. 4.5.1).
La verifica deve essere documentata, eventualmente congiuntamente alla fase di validazione, nella forma ritenuta più appropriata.

Attività di test

Le attività di test servono a verificare che il software sviluppato sia congruente con la specifica dei requisiti e di progetto. Nel caso di software di impianto la verifica è esaustiva compatibilmente con la disponibilità di mezzi e apparecchiature presso la sede. Quanto non provato deve essere pianificato e verificato presso il sito di installazione.
Le attività possono riguardare i test di modulo, di integrazione e di sistema ed eventuali prove di regressione. Potrebbero altresì risultare necessarie prove di qualifica (ad es. per le certificazioni) e/o di accettazione (se richieste contrattualmente), secondo quanto previsto dal programma di sviluppo.
Le attività di test vengono pianificate mediante l'emissione di una Specifica di Test.
La fase di codifica e test relativamente ad un modulo software risulta completata quando viene terminata positivamente la fase di debug.
Una volta eseguiti i test è indispensabile registrarli nelle forme più convenienti (check- list, modulistica, verbali, ecc.) ma comunque tali da consentirne il recupero in qualsiasi momento. Le registrazioni devono essere complete di tutte le informazioni che permettono di comprendere gli esiti e le condizioni esecutive dei test effettuati, a complemento della Specifica dei Test.
I test che hanno avuto esito negativo devono ovviamente essere ripetuti, mentre i test che vanno a buon fine vengono registrati nell'apposito documento proposto in allegato 1G (pag. 59).

Validazione della progettazione

Lo scopo delle attività di validazione è quello di dare la ragionevole fiducia che il software realizzato risponderà ai suoi requisiti operativi.
A seconda della tipologia di software sviluppato, la validazione può avvenire in un'unica fase o prevedere diverse fasi/passaggi prima che il prodotto venga definitivamente validato ed eventualmente sottoposto all'accettazione del cliente. Tali validazioni preliminari avvengono creando condizioni quanto più

possibile simili a quelle dell'ambiente finale di applicazione. Al termine di queste fasi avviene il rilascio di una versione del prodotto software.
La validazione viene generalmente effettuata tramite prove, svolte a diversi livelli: test di modulo, di integrazione e di sistema. Le modalità di svolgimento e l'estensione delle prove sono diverse a seconda della tipologia di prodotto e diverse sono le modalità di registrazione delle stesse.
Nel caso di software per prodotto, la validazione è contestuale alla fase di validazione del prodotto nella sua completezza.

Nel caso di software di commessa, al termine della messa in servizio in cantiere vengono effettuate le prove funzionali (prove di impianto e test funzionali), spesso alla presenza del Cliente stesso. Tale fase corrisponde alla validazione definitiva del progetto, a meno che non siano previsti contrattualmente anche dei performance test, nel qual caso la validazione avviene a valle di questi ultimi. Al termine di questa fase avviene il rilascio di una versione del prodotto software.
I risultati delle attività di validazione possono comportare la necessità di apportare modifiche alla progettazione e, di conseguenza, modifiche alla documentazione.
La validazione deve essere documentata, nella forma ritenuta più appropriata, purché registrata come tale.

Gestione della configurazione

L'identificazione e la rintracciabilità dei prodotti software sono comunemente realizzate tramite la gestione della configurazione, i cui principali obiettivi sono:

- fornire la completa visibilità della configurazione attuale e del suo stato;
- rendere noto a tutti coloro che partecipano allo sviluppo del prodotto le versioni disponibili ed il loro stato di rilascio, nonché le modalità di aggiornamento
- rendere noto agli utenti lo stato dei prodotti software consegnati/installati
- assicurare la correlazione degli elementi software con la pertinente documentazione

Le modalità operative per l'identificazione, il controllo di configurazione degli elementi software e la loro registrazione, variano in funzione del prodotto/impianto/applicazione e saranno descritte in istruzioni operative dedicate. È comunque auspicabile che gli identificativi (nome) degli elementi software siano di tipo codificato (ad es. alfanumerico) e che il controllo delle versioni venga diversificato nelle fasi di sviluppo interno rispetto al rilascio all'esterno dell'azienda.
Le modalità di gestione della configurazione dei prodotti software sviluppati devono essere stabilite già in fase di avvio del progetto. Devono essere noti sia i prodotti e/o sottoprodotti software soggetti a gestione della configurazione, sia i momenti di rilascio previsti.
Ogni elemento software soggetto a gestione della configurazione deve essere debitamente ed univocamente identificato (compresi il software riutilizzato, le librerie, il software acquistato e quello fornito dal cliente).

Attività di rilascio

Le attività di rilascio rientrano nella gestione della configurazione e sono diverse a seconda che si tratti di software di prodotto o di commessa. Nello sviluppo di software per prodotti può risultare utile e necessario mettere a disposizione e dare adeguata informativa agli utenti interni e/o esterni del nuovo prodotto software rilasciato o di una sua nuova versione.
Il rilascio di un prodotto software ad utilizzatori interni e/o esterni (clienti) deve essere formalmente documentato. Lo storico delle versioni rilasciate, completo di tutte le informazioni associate ad ogni singola versione di un elemento software (data, responsabile, motivo del rilascio, applicazione, consegna, ecc.), deve essere conservato e documentato nonché accessibile ad utenti interni autorizzati (ad es. il Service). In caso di versioni modificate sull'impianto in fase di messa in servizio, test di validazione o intervento di Service, è responsabilità del tecnico che ha eseguito le modifiche riportare in sede la versione installata (*as-built*) per archiviazione. Il responsabile di progetto ha il compito di recuperare tale versione o reclamarla.

Manualistica

Durante il ciclo di sviluppo, deve essere prevista anche l'emissione dei manuali necessari all'installazione, utilizzo e manutenzione del software. La stesura di tali documenti dovrebbe iniziare già dalle prime fasi di progettazione e recepire le correzioni a seguito delle fasi di test.
La tipologia dei manuali è diversa a seconda del prodotto software a cui sono riferiti e agli utenti a cui sono destinati:

- nel caso di software per prodotti la manualistica costituisce una raccolta di tutta la documentazione necessaria ad un corretto uso e manutenzione del software: procedure di installazione, manuali d'uso (configurazione e parametrizzazione), guida di utilizzo, descrizione tecnica del software sviluppato, procedure per la manutenzione.
- nel caso di software di impianto la manualistica costituisce una raccolta organica di tutta la documentazione tecnica necessaria ad un corretto uso e manutenzione del software del sistema. Essa è costituita in generale da procedure di installazione, manuali d'uso, configurazione e parametrizzazione dei sistemi e delle funzioni applicative, guida all'utilizzo dell'HMI del sistema di automazione, descrizione tecnica del software sviluppato, schemi d'impianto.

Nel caso di funzioni collegate alla "Sicurezza" (ad es. in prodotti certificati) che possono essere gestite dall'utilizzatore del software, anche indirettamente, la manualistica deve descrivere in dettaglio le conseguenze di talune operazioni ad esse collegate e fornire adeguati avvisi, in modo da non comprometterla.

Archiviazione

Completata la fase di test con esito positivo e rilasciata la versione ufficiale, il software e la documentazione pertinente devono essere archiviati in maniera controllata. Le procedure di archiviazione devono assicurare che:

- sia salvaguardata l'integrità del prodotto software: accesso controllato ai prodotti software, protezione da modifiche non autorizzate
- l'agevole rintracciabilità delle versioni rilasciate stabilendo i criteri o il numero di versioni che debbono essere conservate (anche in funzione della attività di service previste)
- la protezione del software da virus
- la protezione dei supporti tenendo in considerazione che sono soggetti a deterioramento e degrado in relazione alla tipologia, alle condizioni ambientali di conservazione, ai luoghi, ecc.
- la predisposizione di copia di salvataggio del software (back-up) per il ripristino da disastro tramite duplicazione di ogni versione rilasciata, preferibilmente su supporti di tipo diverso
- la disponibilità dei mezzi di lettura e recupero del software (se sono state utilizzate tecniche di compressione/decompressione o *tools* di sviluppo particolari), compresi gli ambienti di sviluppo (sistemi operativi, ecc.)

Durante tutto il ciclo di sviluppo la documentazione (cartacea o digitale) viene conservata dal Resp. di Progetto che provvede alla costituzione del dossier conclusivo per l'archiviazione. Tale dossier deve essere completo di tutti i documenti generati nel corso dello sviluppo del software. Le modalità operative per l'archiviazione degli elementi software e della relativa documentazione variano in funzione del prodotto/impianto/applicazione e saranno descritte in istruzioni operative dedicate.

Attività di manutenzione e modifica del software

L'attività di manutenzione riguarda sostanzialmente il software di prodotto ed il software standard (librerie). Tale attività avviene a seguito nella necessità di migliorie, alla correzione di errori riscontrati nell'utilizzo, alla necessità di implementare nuove funzionalità e modificare quelle esistenti, ampliamenti ed estensioni, nuove interfacce. Le attività di manutenzione devono avvenire nel rispetto della gestione della configurazione ed hanno impatto sull'intero ciclo di sviluppo.
Occorre tenere in considerazione l'identificazione dello stato iniziale degli elementi software sottoposti a manutenzione e la necessità di prove di regressione. Nel caso risulti necessario, per ragioni di operatività, eseguire correzioni temporanee, queste devono essere successivamente analizzate e opportunamente

documentate (almeno con delle note da conservare all'interno del dossier di progetto/commessa relative alle modifiche effettuate ed alla loro implementazione).
Le attività di manutenzione si concludono con il rilascio e l’archiviazione di una nuova versione.

Gestione delle modifiche

Le modifiche devono mantenere coerenza tra requisiti, progettazione, codice, specifiche di prova e manuali utente. La gestione delle modifiche, pertanto, ricade nella gestione della configurazione.
Nel caso di modifiche riguardanti interfacce esterne, espansione delle funzioni o miglioramento delle prestazioni la modifica deve ripercorrere tutto l'iter di sviluppo a partire dalla fase di definizione dei requisiti, ripercorrendo via via le fasi di codifica, test, ecc..
Le modifiche implementate possono essere documentate sia in modo formale che informale.
L'aggiornamento dei documenti a mezzo *mark-up* è possibile solo finché non viene compromessa la leggibilità degli stessi.

LE LIBRERIE SOFTWARE (SOFWARE LYBRARIES)

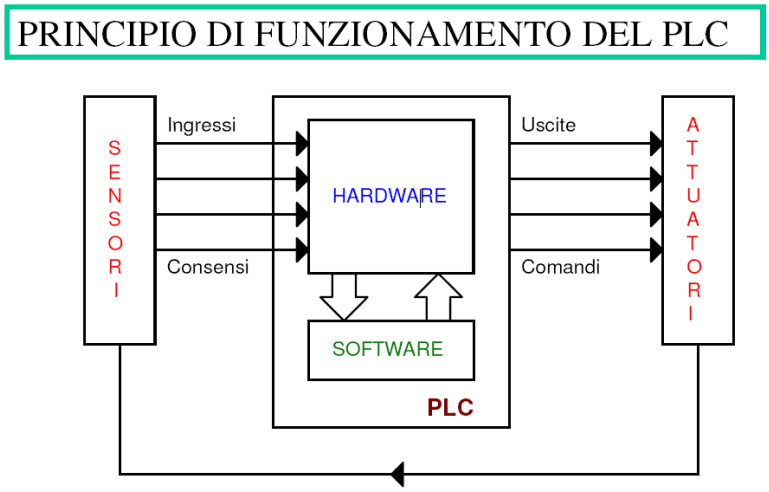
Introduzione: il PLC

L'avvento dei microprocessori ha permesso, da circa la metà degli anni settanta, l'introduzione sul mercato di dispositivi programmabili progettati esclusivamente per realizzare sistemi si automazione.
Con questi dispositivi, denominati controllori logici programmabili (Programmable Logic Controller, PLC) le funzioni di automazione non sono realizzate in hardware (con l'uso cioè di circuiti a relè) bensì da un programma eseguito da PLC che consente all'elettronica di quest'ultimo di generare segnali in risposta alle informazioni avute in ingresso ed elaborate dal programma stesso.
L'avvento dei PLC ha portato grandi miglioramenti nei sistemi di automazione industriale, infatti le funzioni che essi svolgono sono molteplici e potenti.
Essi consentono di realizzare operazioni logico-combinatorie tipiche dei relè, ma sopratutto sono in grado di fornire prestazioni molto simili a quelle dei moderni sistemi a microprocessore, come ad esempio: temporizzatori, conteggi, tecniche di indirizzamento, blocchi decisionali, cicli iterativi,ecc.
Un controllore programmabile è basato su una struttura hardware il cui nucleo centrale, la CPU, esegue ciclicamente un programma memorizzato su un supporto fisico adeguato. Spesso il sistema è completato da dispositivi di interfaccia uomo-macchina i quali permettono un'interazione agevole con l'impianto sotto controllo del PLC (ad esempio il sistema a supervisore HMI).

Nel suo funzionamento, il controllore programmabile acquisisce segnali provenienti dall'impianto da controllare, li elabora ricavandone ulteriori segnali con i quali agisce sull'impianto stesso. Tali funzioni vengono svolte per mezzo di una adeguata sezione di ingresso/uscita (segnali si analogici che digitali) costituita da schede che andranno a comporre il *rack* PLC che costituiscono il vero e proprio interfacciamento del controllore all'impianto.
Tra le unità periferiche vanno menzionate quelle che permettono di collegare il PLC ad altri *rack* che non fanno parte della struttura che sono:

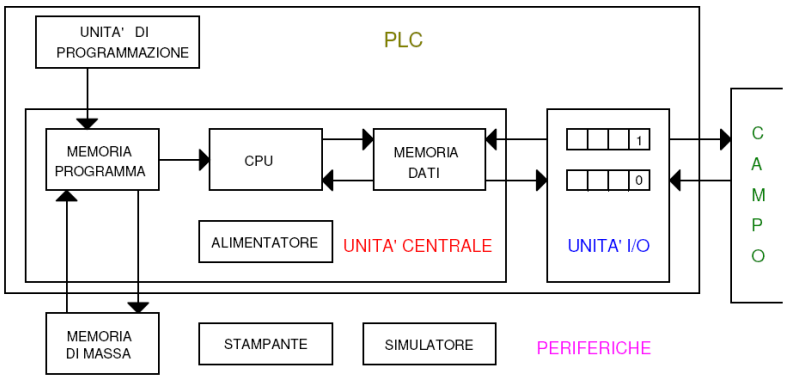
- Schede per interfaccia seriale.
- Schede per interfaccia verso le più diffuse reti di comunicazione industriali come ad esempio il PROFIBUS.

Nella figura sottostante è mostrato il principio di funzionamento del PLC:



Dal punto di vista hardware, un PLC è composto da un'unità centrale e una serie di unità periferiche. Tali unità sono normalmente costituite da moduli che vengono messi in collegamento tra loro da un bus di comunicazione. L'installazione dei moduli avviene su guide profilate denominati "*rack*".
Compito della CPU è quello si eseguire ciclicamente il programma utente residente all'interno di una *memory unit*. Nella figura sottostante è mostrata la struttura hardware del PLC:

HARDWARE DI UN PLC



Per realizzare il programma che un PLC deve eseguire ogni produttore mette a disposizione un sistema di sviluppo: si tratta di un ambiente software che permette la scrittura e il test dei programmi.
Il sistema di sviluppo normalmente fa uso di un Personal Computer come hardware, tuttavia esistono anche dei piccoli terminali per la programmazione dei PLC che nonostante permettano di eseguire solo un set limitato di operazioni, sono piuttosto utili in fase di collaudo sull'impianto date le loro dimensioni molto ridotte e la semplicità d'uso.
Nell'implementazione di un programma per un PLC si possono distinguere due fasi:

- La fase "off-line" di realizzazione nella quale sistema di sviluppo e PLC non sono collegati.
- La fase "on-line" in cui il programma realizzato viene trasferito nel PLC e provato. Il collegamenti tra sistema di sviluppo e PLC permette di usufruire di tutti i tools a disposizione per il test del programmi ad esempio il forzamento di variabili,la lettura del loro stato, l'impostazione di break-point nel programma ecc...

Quasi tutti i produttori adottano un linguaggio di programmazione definito "a contatti" (ladder diagram) in cui viene usata una rappresentazione grafica dove i contatti identificano le variabili, mentre i collegamenti tra i contatti specificano le operazioni da compiere sulle variabili.
Questo linguaggio deriva dai primi impieghi dei PLC che in pratica sostituivano delle logiche cablate realizzate con dei relè e risultava essere di comprensione immediata, le sue potenzialità sono pero piuttosto limitate.
Un altro linguaggio molto diffuso è il cosiddetto schema funzionale (control system flow-chart) dove le operazioni sulle variabili sono definita da blocchi ai quali le variabili fanno capo. Lo schema funzionale può fornire ottime prestazioni, infatti le funzioni svolte dai blocchi possono essere di alto livello, ad esempio regolatori PID, registi a scorrimento, operazioni aritmetiche in virgola mobile, logiche combinatorie molto complesse ecc...
Il terzo linguaggio di programmazione che viene utilizzato è la lista di istruzioni (statement list), si tratta di un linguaggio molo potente che ha alcune istruzione proprie dell'*assembler* del microprocessore del PLC e altre più sofisticate che vengono tradotte dal sistema operativo del PLC in fase di generazione del codice macchina. Per questi motivi la lista di istruzioni è attualmente il linguaggio di programmazione più usato in e Europa anche se risulta meno comprensibile per chi non ha direttamente scritto il programma e necessita di un'adeguata documentazione.

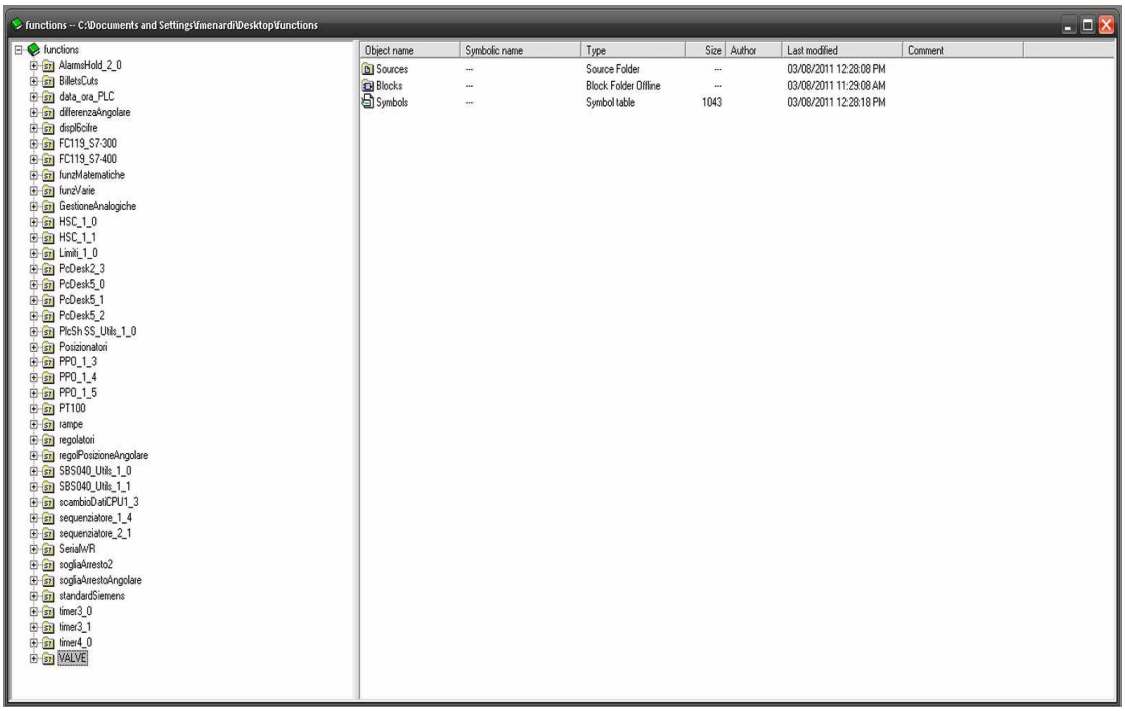
Le Librerie del Software

Nella maggior parte dei casi la gestione di impianti siderurgici sviluppata da Ansaldo Sistemi Industriali prevede l'utilizzo dell'ambiente software fornito da Siemens denominato Step 7 ("S7").
La seconda parte del mio stage prevedeva lo studio del funzionamento di alcune funzioni base del software, usato per la gestione di impianti nel settore siderurgico, e la documentazione di esse con l'aggiunta eventualmente di leggeri perfezionamenti nelle funzionalità.
Innanzi tutto va spiegato cos'è una libreria software.

Durante la scrittura di un programma risulta molto utile utilizzare “funzioni pre-scritte” che svolgono determinate funzioni, in modo tale da rendere più semplice e veloce il lavoro del programmatore.
Ovviamente queste “funzioni pre-scritte” hanno bisogno di:

- una adeguata documentazione al fine di renderne più chiaro possibile l’uso.
- un accurata fase di collaudo in cui vengono provate tutte le possibili condizioni di utilizzo.

Nella maggior parte dei casi la gestione di impianti siderurgici sviluppata da Ansaldo Sistemi Industriali prevede l’utilizzo dell’ambiente software fornito da Siemens denominato Step 7.
Ovviamente il risultato della documentazione che ho sviluppato è tuttora resa disponibile a tutti i programmatori di Ansaldo Sistemi Industriali grazie a una cartella sul *server aziendale* il cui contenuto è mostrato in figura.



Ogni sottocartella è denominata con il nome della funzione che descrive con accanto la versione della stessa in modo tale da tenere una traccia storica dell’evoluzione della funzione in esame. Per ottemperare a questo inoltre è stato redatto un documento ufficiale chiamato *Lybrary_history*, una copia dello stesso è mostrato nella figura sottostante.

	FUNCTION	**VERSION**
	AlarmManag2_0	
	AlarmsHold_FC109	2,0
	AngularDifference	
	AngularDifference_FC25	1,0
	AngularPositionRegulator	
	AngularPositionRegulator_FC27	1,0
	AusiliariDrive_old	
	PPOreadDB_FC12	1,3
	PPOwriteDB_FC13	1,3
	GetSVTAlarms_FC40	1,0
	LinearRamp_FC41	1,3
	WindingsBearingsTemp_FC42	1,0

BilletsCuts		
	BilletsDiagManage_FC185	1,2
BitStandard		
	bit standard_FC119	1,3
DataExchangeCPU1_3		
	S7-400 <> S7-400_FB101	0,1
	S7-400 <> S7-300_FB102	1,3
	FC259	1,1
Date_time_PLC		
	Time_Synchro_FC14	1,0
Displ6cifre		
	6DIGIT s DISPLAY-AUX_FC38	1,0
	6DIGIT s DISPLAY-VIS_FC39	1,1
HSC_1_1		
	HSCmanagement_FC28	2,1
	HscMng_FB28	2,2
Limits_1_0		
	IntegerLimit_FC20	1,4
	DoubleIntegerLimit_FC21	2,3
	RealLimit_FC22	2,0
Mathfunction		
	Y_MX_Q_FC114	1,0
	Gray to binary 16bit_FC117	1,0
	Gray to binary 32bit_FC118	1,0
	INTGR_FB30	1,0
PCDesk_5_2		
	SE_TimerFb _FB12	2,0
	MainDeskMng_FB90	2,3
Positionator		
	P-Reg_FC125	2,1
PT100		
	PT100_FB14	2,1
	FB15	2,1
SearchFaultsPanel		
	RD_panel_FC107	2,5
	panel_CK_FB107	2,4
Sequencer_2_1		
	SEQ_Aux_FC105	3,0
	SEQ_Managem_FB105	3,0
	SEQ_Deco_FB106	3,0
SeriaLedPanel		
	WR_panel_FC108	4,1
	panel_CK_FB107	4,1
	BCD panel write_FB108	4,1
Timer3_1		
	SD_Timer_FC100	5,0
	SF_Timer_FC101	5,0
	SE_Timer_FC102	5,0
	SP_Timer_FC103	5,0
	SD_TimerFb_FB10	5,0
	SF_TimerFb_FB11	5,0
	SE_TimerFb_FB12	5,0
	SP_TimerFb_FB13	5,0

	VariousFunctions		
		BoolsTolnt_FC8	2,1
		IntToBools_FC9	2,1
		BoolsToWord_FC10	2,1
		WordToBools_FC11	2,1
		FC15	2,1
		toS5TIME_FC37	2,1

Tutte le descrizioni delle varie funzioni sono riportate in allegato 2, il software vero e proprio è ovviamente riservato al solo personale aziendale, mi è stato impossibile pertanto riportarne qui qualche esempio.

CONCLUSIONI

Questa esperienza mi ha portato ad approfondire molto il tema della qualità nel campo industriale riuscendo a produrre l'iter e i documenti necessari per uno sano ed efficiente sviluppo del software in Ansaldo Sistemi Industriali.
Attualmente i documenti da me prodotti e che vi ho presentato in allegato rappresentano la procedura ufficiale con la quale viene sviluppato il software, tuttavia il mio lavoro è rimasto circoscritto al settore siderurgico, in cui il software rimane un prodotto per la commessa, senza espandersi verso il settore del "software di prodotto" in cui lo stesso software è il prodotto finale.
Interessante sarebbe stato carpire le esigenze di quest'ultimo settore, sviluppare la relativa documentazione creando cosi una procedura per la qualità del software generale dell'intera azienda.
La doppia tematica del mio percorso come tirocinante in Ansaldo Sistemi Industriali mi ha permesso di ampliare notevolmente le mie conoscenze attraverso i diversi ambiti presenti nell'azienda, passando dalla gestione delle attività alla creazione del'iter necessario per la creazione del software fino ad arrivare alla creazione/documentazione dell'intera libreria software relativa al settore siderurgico.
La mia esperienza in Ansaldo Sistemi Industriali non è tuttavia terminata con il tirocinio, attualmente mi occupo dell'ingegneria ed il *commissiong* di impianti siderurgici in tutto il mondo, riuscendo cosi a mettere in pratica e a vedere realizzato ciò per cui ho lavorato.

BIBLIOGRAFIA e RIFERIMENTI

-"*Qualità del software*", Ercole Colonnese, 2005
-"Piano qualità del software", Asiansaldo 18/05/2010

ALLEGATO 1

Nota Applicativa

GUIDA APPLICATIVA PER LA QUALITÀ DEL SOFTWARE

Numero	Rev.	Data
XXX 000	**00**	**xx/xx/10**

	Redatto	Controllato	Approvato
NOME	F.Menardi	A.Rossi	x
FIRMA			
NOME	x	x	x
FIRMA			

GRADO DI RISERVATEZZA PER ASI		**Per uso interno**		
STATO DI REVISIONE DEL DOCUMENTO		**Prima emissione**		
NUMERO DI PAGINE TOTALI			COPIA NUMERATA:	**NO**

INDICE

Pagina

Documenti richiamati

PQ ASI 040	Piano di Qualità del software
IQ ASI 512	Archiviazione dei dossier di commessa software

Modulistica richiamata

I moduli non sono fisicamente annessi al presente documento. Il loro utilizzo è subordinato alla verifica, presso gli appositi archivi aziendali, della versione/edizione più aggiornata.

Allegati

ALLEG. A	Pian_attività_sviluppo_sw
ALLEG. B	Scheda d'incarico CDQ211
ALLEG. C	Procedura_dossier_sw
ALLEG. D	Indice_dossier_sw
ALLEG. E	Report_di_riesame
ALLEG. F	Specifica_STS_sw
ALLEG. G	Test_report

1. SCOPO DEL DOCUMENTO

Il presente documento è una guida alllo sviluppo del software secondo quanto previsto dal Sistema di Qualità Aziendale.

1.1 Applicabilità

Le indicazioni riportate dal seguente documento si applicano a tutte le commesse software poste in essere dall' unità operativa, in particolare questo documento si applica al reparto impianti siderurgici (GAI).

1.2 Formato della documentazione

Si ricorda che, secondo quanto stabilito dalla Istruzione della Qualità IQ-ASI-512 - Archiviazione dei dossier di commessa software, tutta la documentazione del dossier di commessa software deve essere archiviata in formato elettronico.

1.3 Documenti di riferimento

I documenti di riferimento della presente guida sono:

- PQ-ASI-040: Piano Qualità del Software
- IQ-ASI-512: Archiviazione dei dossier di commessa software

2. DESCRIZIONE DELLE ATTIVITA' DI SVILUPPO

2.1 Pianificazione delle attività

La definizione delle attività da eseguire per lo sviluppo di una commessa dà luogo alla stesura del:

"**Piano di Sviluppo delle attività Software (SDP)"**

Tale documento, nel quale vengono indicate le principali fasi previste dello sviluppo, deve essere aggiornato qualora vi siano significative variazioni alla pianificazione.
Il modello adottato per il seguente documento è riportato in allegato A.

2.1.1 Pianificazione temporale

La pianificazione temporale viene eseguita con gli strumenti software dedicati previsti dall'azienda, in particolare per i software di commessa nel reparto GAI viene utilizzato il software: Microsoft Project.

2.2 Definizione del Responsabile di Progetto e del Gruppo di Lavoro

L'assegnazionedell'incarico di responsabile di progetto e del gruppo di lavoro avviene con l'invio di un'email seguita in allegato dal modulo "CDQ211" previsto per tale utilizzo.
Il modello adottato per la seguente procedura è riportato in allegato B.

2.3 Costituzione del dossier di commessa software

A seguito dell'apertura della commessa si deve predisporre il dossier di commessa software, in cui andrà inserita tutta la documentazione e il software (sia prodotto sia utilizzato) relativi alla commessa.
Il dossier di commessa, in virtù dell'archiviazione elettronica in vigore sarà costituito da una cartella/directory posta su un server (aziendale o di reparto). Tale cartella deve essere accessibile a tutto il personale assegnato alla commessa.
La sua struttura deve essere conforme a quanto previsto dalla IQ-ASI-512 al par 2.2.
Fino alla sua archiviazione è compito del Responsabile di Progetto (o di una persona da lui delegata) il tenere aggiornato e in ordine il dossier di commessa.

Le modalità si costituzione del dossier software sono descritte dal documento riportato in allegato C.

2.3.1 Indice del dossier di commessa software

Il modello per l'indice del dossier di commessa è dato dal modulo CDQxxx.
In esso devono essere riportati obbligatoriamente almeno tutti i documenti previsti dal Piano Qualità e i documenti ritenuti rilevanti per la commessa.
Per ogni documento riportato devono essere indicati:

- Sigla identificativa
- Titolo
- Rev
- Data di Emissione

E' compito del Responsabile di Progetto (o di una persona da lui delegata) mantenerlo allineato con il contenuto del dossier.
Le modalità si costituzione del dossier software sono descritte dal documento riportato in allegato D.

2.4 Determinazione dei requisiti

2.4.1 Riesame dei Requisiti

Tutte le attività di riesame congiuntamente all'attività di riesame delle attività dei test sono documentate durante la fase di validazione del software.
Il modello adottato per il seguente documento è riportato nell'allegato E.

2.5 Progetto

2.5.1 Riesame del Progetto (Verifica della Progettazione)

Tutte le attività di riesame congiuntamente all'attività di riesame delle attività dei test sono documentate durante la fase di validazione del software.
Il modello adottato per il seguente documento è riportato nell'allegato E.

2.6 Specifica dei Test

Contestualmente alla progettazione o immediatamente a valle della stessa è prevista la pianificazione delle attività di test.
La conseguente "Specifica dei test" definisce le funzionalità software oggetto delle prove, gli obiettivi e la tipologia delle prove, l'ambiente e la configurazione di prova, le modalità di esecuzione e i criteri di accettazione.
Il modello adottato per il seguente documento è riportato nell'allegato F.

2.8 Test di modulo, di integrazione e funzionali

Nel caso di software di impianto la verifica è esaustiva compatibilmente con la disponibilità di mezzi e apparecchiature presso la sede. Quanto non provato deve essere pianificato e verificato presso il sito di installazione.
Le registrazioni dei test devono essere complete di tutte le informazioni che permettono di comprendere gli esiti e le condizioni esecutive dei test effettuati, a complemento della "Specifica dei test".
Il modello adottato per il seguente documento è riportato nell'allegato G.

2.9 Validazione

Tutte le attività di riesame congiuntamente all'attività di riesame delle attività dei test sono documentate durante la fase di validazione del software.
Il modello adottato per il seguente documento è riportato nell'allegato E.

2.10 Archiviazione

Il capo commessa deve presentare il materiale per l'archiviazione obbligatoriamente alla conclusione della messa in servizio della commessa.
E' ammessa un'archiviazione preliminare qualora si preveda che trascorra un notevole lasso di tempo tra la conclusione dello sviluppo e l'inizio della messa in servizio.
Nell'intervallo tra la conclusione dello sviluppo e l'inizio della messa in servizio tutto il materiale di commessa deve comunque essere archiviato, sia pure senza una registrazione formale, presso il reparto di

competenza. La sua conservazione è in tal caso affidata alle normali strategie previste dall'attività di sviluppo.
Tutta la documentazione e il software di una commessa vanno archiviati, all'interno della directory di reparto, in una directory il cui nome deve essere del tipo:

NumeroCommessa-Commento (ad esempio: 7025214-Aza_Colina).

In caso di versioni modificate sull'impianto in fase di messa in servizio, test e validazione o intervento in Service previsto dal contratto, è responsabilità del tecnico che ha eseguito le modifiche riportate in sede la versione installlata per procedere con l'archiviazione. Il responsabile di progetto ha il compito di recuperare tale versione o reclamarla.

2.11 Gestione delle modifiche

Le modifiche al software devono mantenere coerenza tra requisiti, progettazione, codice, specifiche di prova e manuali utente.
Nel caso di modiche riguardanti interfacce esterne, espansione delle funzioni o miglioramento delle prestazioni la modifica deve ripercorrere tutto l'iter di sviluppo a partire dalla fase di definizione dei requisiti, ripercorrenso via via le fasi di codifica, test, ecc....

ALLEGATO 1A

PIANO DI SVILUPPO DELLE ATTIVITA' SOFTWARE DI COMMESSA (SDP)

Numero	*Rev.*	*Data*
XXX 000	**00**	**xx/xx/10**

	Redatto	*Controllato*	*Approvato*
NOME	*F.Menardi*	*A.Rossi*	*x*
FIRMA			
NOME	*x*	*x*	*x*
FIRMA			

GRADO DI RISERVATEZZA PER ASI		**Per uso interno**			
STATO DI REVISIONE DEL DOCUMENTO		**Prima emissione**			
NUMERO DI PAGINE TOTALI	7	FILE		*COPIA NUMERATA:*	**NO**

Paragrafo	INDICE	Pagina

Documenti richiamati	
PQ-ASI-040	*Piano di Qualità del software*

Modulistica richiamata

I moduli non sono fisicamente annessi al presente documento. Il loro utilizzo è subordinato alla verifica, presso gli appositi archivi aziendali, della versione/edizione più aggiornata.

Allegati

INTRODUZIONE

- **Obiettivo del documento**

Il presente documento fornisce la pianificazione delle attività durante la creazione del software, i costi, le responsabilità e i tempi previsti al conseguimento di tale obiettivo.

- **Informazioni generali**

Numero di commessa	Committente	Capo Commessa
.....	.....	.....
Destinazione dell'impianto	**Tipologia dell'impianto**	
.....	.....	

GRUPPO DI LAVORO		
Nome e Cognome	**Piano di attività**	**Note**
Responsabile di progetto:	.....	

BUDGET DELLE ATTIVITA'

Ore previste	Data prevista per la consegna	Costi previsti per materiali
.....	.....	.....

DESCRIZIONE E PIANIFICAZIONE TEMPORALE DELLE ATTIVITA'

Attività richiesta		Attività	Documenti di riferimento		Breve descrizione dell'attività, dove necessaria.	Referente	Data	Note
Si	No		Input	Output				
		Programmazione temporale delle attività.		\\Viwn00fp\GAI\Ufficio\Pianificazione\Gai.mpp	Documento excel di piano temporale delle attività			
		Assegnazione del team leader.		CDQ211	Responsabile di progetto			
		Predisposizione del dossier di sviluppo software.		Indice dei documenti	Raccolta di tutta la documentazione relativa al progetto sw.(piani,spec,manuali,registrazioni,e il sw sviluppato o il suo riferimento). Vedi "Procedura dossier_sw"			
		Determinazione dei requisiti.	ASAHW	Specifica dei requisiti. (SRS)	Acquisizione/scrittura della Specifica di funzionale dell'impianto (requisiti del sw e raccolta ASAHW)			
		Assegnazione degli incarichi.		CDQ211	Gruppo di lavoro,e responsabile di progetto			
		Riesame/Verifica di correttezza e di completezza dei requisiti.		Report di riesame	Formalizzazione di avvenuta verifica di correttezza e di completezza dei requisiti. PQ-ASI-040 Vedi par:4.3.1			

		Progetto sw.		ASASW Specifica di progetto. (SDD)				
		Riesame di congruenza delle architetture hw e sw rispetto ai requisiti desiderati.	ASAHW ASASW	Report di riesame				
		Elaborazione della specifica dei test funzionali.	Specifica dei requisiti. (SRS)	Specifica dei test. (STS)				
		Elaborazione lista dei riferimenti da usare.	ASASW Specifica di progetto. (SDD)	Lista dei riferimenti				
		Definizione dei dati di supervisione.	ASASW Specifica di progetto. (SDD)	Dati del supervisore				
		Elaborazione flow chart e diagramma a blocchi	ASASW Specifica di progetto. (SDD)	Allegato FLOW_CHART				
		Codifica e programmazione.	Specifica di progetto. (SDD)	codice				
		Riesame durante la fase di codifica (walkthrough)	codice	Report di riesame CDQ069 CDQ081				

		Stesura del manuale di uso HMI (utilizzo e manutenzione del software).		Riferimento del Manuale	proc. di installazione, guida d'utilizzo, guida all'utilizzo dell'HMI del sistema d'automazione, descrizione tecnica del sw, schemi d'impianto. PQ-ASI-040 Vedi par: 4.7			
		Test di modulo e integrazione del progetto sw.	Specifica dei test. (STS)	"Test report"				
		Validazione del software	Specifica dei requisiti (SRS), specifica dei test (STS).	"Validation report"	Tale validazione è da svolgersi dopo la messa in servizio in cantiere e dopo aver svolto tutti i test funzionali (alla presenza del cliente stesso se richiesto). PQ-ASI-040 Vedi par. 4.5.2			
		Prima archiviazione (PROG).		Software Archiviato	Per i software di commessa è ammessa una archiviazione preliminare qualora si preveda che trascorra un notevole lasso di tempo tra la conclusione dello sviluppo e l'inizio della messa in servizio. IQ-ASI-512 vedi par: 2.1			
		Archiviazioni successive (MIS-REV1 ecc).		Software Archiviato	Il capo commessa deve presentare il materiale per l'archiviazione obbligatoriamente alla conclusione della messa in servizio. IQ-ASI-512 Vedi par: 2.1			

ALLEGATO 1B

Ansaldo Sistemi Industriali S.p.A.

SCHEDA INCARICO
(PER COMMESSA SW)

COMMESSA	IMPIANTO	COMMITTENTE

DATA DI INCARICO	REPARTO	FIRMA DEL RESP. DI REPARTO

COMPOSIZIONE INIZIALE DEL GRUPPO DI LAVORO

IL TEAM LEADER RESPONSABILE DELLO SVILUPPO SOFTWARE É: Se l'incarico di Team Leader viene modificato in corso di sviluppo la presente scheda deve essere riemessa	
CHE SI AVVARRÀ DELLA COLLABORAZIONE DEI SEGUENTI PROGETTISTI SOFTWARE:	
COINVOLGENDO IL SEGUENTE PERSONALE ESTERNO AL REPARTO: Indicare anche il reparto	
IL CAPO COMMESSA HARDWARE É:	

AMPLIAMENTI SUCCESSIVI DEL GRUPPO DI LAVORO

NOME	REPARTO	DATA	FIRMA DEL RESP. DI REPARTO

ALLEGATO 1C

PROCEDURA DI COSTITUZIONE DEL DOSSIER SOFTWARE

Numero	Rev.	Data
X	00	X/11/10

	Redatto	Controllato	Approvato
NOME	F. Menardi	A. Rossi	x
FIRMA			
NOME	X	X	
FIRMA			
NOME	X	X	
FIRMA			

GRADO DI RISERVATEZZA (per ASI)		Per uso interno				
STATO DI REVISIONE DEL DOCUMENTO		Prima emissione				
NUMERO DI PAGINE TOTALI	4	FILE:	x	COPIA NUMERATA	NO	----

INDICE

Pagina

Documenti richiamati

PQ-ASI-040 *Piano qualità del software*

Modulistica richiamata

I moduli non sono fisicamente annessi al presente documento. Il loro utilizzo è subordinato alla verifica, presso gli appositi archivi aziendali, della versione/edizione più aggiornata.

Allegati

1 OGGETTO DELLA PROCEDURA

La presente "Procedura di costituzione del dossier software" descrive i criteri e le modalità operative e gestionali inerenti lo sviluppo del prodotto software e dei suoi elementi costitutivi.
Per quanto applicabile questo documento recepisce le linee guida per l'applicazione della ISO 9001 allo sviluppo del software contenute nella norma ISO/IEC 90003: 2005.

1.1 Applicabilità

Quanto previsto dal presente documento è applicabile esclusivamente nell'ambito della società Ansaldo Sistemi Industriali (di seguito ASI) nella sezione GAI.

1.2 Documenti di riferimento

La presente istruzione fa riferimento ai documenti:

- PQ-ASI-040 "Piano qualità del software".
- IQ-ASI-512-r00 "Archiviazione dei dossier di commessa software".

2 MODALITA' DI COSTITUZIONE DEL DOSSIER SOFTWARE

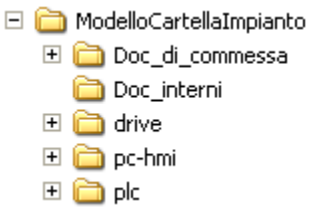
L'archiviazione dei dossier di commessa software deve avvenire esclusivamente in formato elettronico sul server aziendale a ciò dedicato.
La collocazione del server, la gestione dello spazio su disco, degli accessi, dei permessi e il backup delle informazioni in esso contenute sono attività fornite dall'ente IT.
L'archiviazione dei dossier è invece compito esclusivo del personale a ciò designato.

2.1 Directory di archiviazione

L' attività di archiviazione è gestita a livello di reparto, pertanto per ogni reparto che si occupa di sviluppo software è a disposizione sul server aziendale una directory (directory di reparto).

2.1.1 Struttura interna

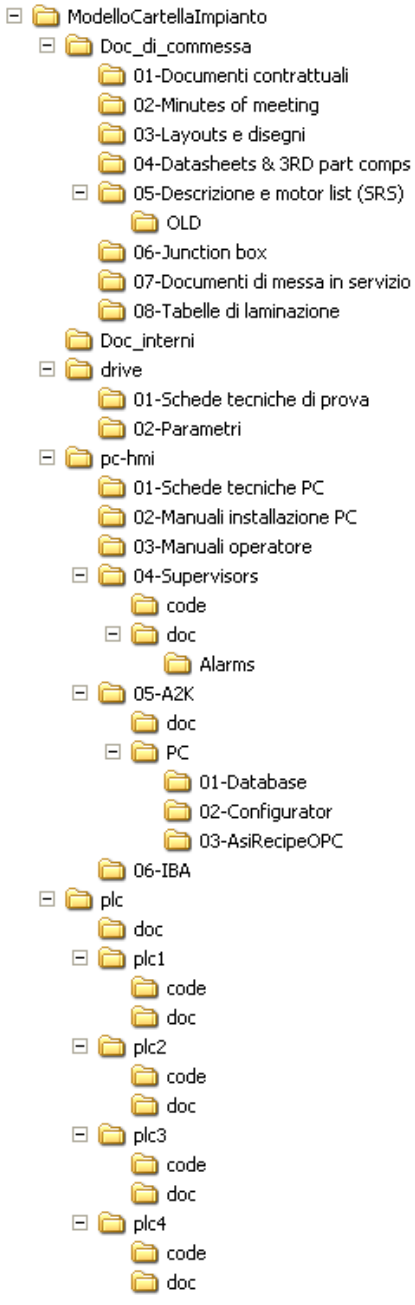
Tutta la documentazione e il software di una commessa vanno archiviati, all'interno della directory di reparto, in una directory il cui nome deve essere del tipo: *NumeroCommessa-NomeProgetto* (ad esempio: 7025214-Aza_Colina).
La struttura interna deve essere la seguente:



- **Doc_di_commessa**: contiene tutta la documentazione della commessa, compresa la scecifica dei requisiti (SRS).
- **Doc_interni**: Documenti interni riguardanti il progetto.
- **Drive**: contiene i backup dei parametri dei drive della commessa.
- **PC-HMI**: contiene tutto il software per PC, progetti per supervisori.
- **PLC**: contiene tutti i progetti PLC(documenti e codice).

E' ammessa la creazione di ulteriori/alternative directory qualora nella commessa siano impiegati prodotti che si estendono su apparecchiature di tipo diverso (es. A2K).

3 DESCRIZIONE DELLA STRUTTURA DI UN DOSSIER SOFTWARE



ALLEGATO 1D

sNumero Documento	Data di emissione	Numero di Pagine	Emissione e controllo
ED – SW AV.71.xxxxxxx	dd/mm/yyyy	1	Tizio Caio

ELENCO DOCUMENTI DOSSIER SOFTWARE

Numero	Rev.	Data di emissione	Titolo completo	Tipo	Note

ALLEGATO 1E

C S Q CERTIFIED QUALITY SYSTEM	Ansaldo Sistemi Industriali S.p.A. **ATTIVITA' DI RIESAME, VERIFICA E TEST**	**G A I** Gruppo Automazione Impianti siderurgia

commessa	Committente	Cliente finale
.....	.....	.....
Tipologia impianto		
.....		
Destinazione impianto	Capo commessa	Team leader
.....	.....	.....

ATTIVITA' DI VERIFICA RIESAME E TEST	
☐	Riesame/Verifica di correttezza e di completezza dei requisiti.
☐	Riesame di congruenza delle architetture hw e sw rispetto ai requisiti desiderati.
☐	Riesame durante la fase di codifica (walkthrough).
☐	Test e prove eseguite in sala prove (hw, interfacciamenti, input/output).

Preso atto dell'attività di riesame e verifica svolte, si dichiara completata la prima fase di validazione del software di commessa.

Team leader	**Date**	**Release**
.....	*xx / xx / xx*	.....

I test eseguiti sul software sono stati completati con esito positivo: Non sussistono ulteriori elementi di non conformità funzionale o prestazionale.
Il progetto software è quindi idoneo in relazione all'uso per il quale è stato sviluppato.

Team leader	**Date**	**Release**
.....	*xx / xx / xx*	.....

ALLEGATO 1F

SPECIFICA DEI TEST DEL SOFTWARE (STS)

Numero	Rev.	Data
XXX 000	00	25/01/11

	Redatto	Controllato	Approvato
NOME	F.Menardi	A.Rossi	x
FIRMA			
NOME	X	X	X
FIRMA			

GRADO DI RISERVATEZZA PER ASI		**Per uso interno**			
STATO DI REVISIONE DEL DOCUMENTO		**Prima emissione**			
NUMERO DI PAGINE TOTALI			*COPIA NUMERATA:*	**NO**	

INDICE

Pagina

Documenti richiamati

PQ ASI 040 *Piano di Qualità del software*

Modulistica richiamata

I moduli non sono fisicamente annessi al presente documento. Il loro utilizzo è subordinato alla verifica, presso gli appositi archivi aziendali, della versione/edizione più aggiornata.

Allegati

- **SCOPO E INTRODUZIONE**

Il presente documento fornisce la pianificazione delle attività di test, in particolare definisce i prodotti software oggetto delle prove, gli obiettivi e la tipologia delle prove, l'ambiente e la configurazione di prova, componenti e funzioni da sottoporre a prova, modalità di esecuzione e criteri di accettazione.

- **DOCUMENTI DI RIFERIMENTO**

Documenti	Descrizione dei documenti

- **DESCRIZIONE DEI TEST (da documentare)**

Item	Componenti e funzionalità da testare	Descrizione dei componenti/funzionalità da sottoporre a test	Condizioni si svolgimento del test	Requisiti per il superamento del test
	HMI Pagine video	Verifica tag HMI		
	HMI Gestione ricette	Scambio dati fra CPU1-CPU2 PLC1		
	PLC Funzione cascata	Simulazione cascata e fotocellule		
		Test aumenta diminuisci velocità cascata		
		Scambio dati PLC2-PLC3		
		Scambio dati PLC4-PLC5		

ALLEGATO 1G

C S Q CERTIFIED QUALITY SYSTEM	Ansaldo Sistemi Industriali S.p.A. **TEST REPORT**	**G A I** Gruppo Automazione Impianti siderurgia

commessa	Committente	Cliente finale
.....	.....	.....
Tipologia impianto		
.....		
Destinazione impianto	Capo commessa	Team leader
.....	.....	.....

- **FUNZIONALITA' SOTTOPOSTE A TEST**

Componenti / Funzionalità da sottoporre a test	
1	HMI Pagine video
2	HMI Gestione ricette
3	PLC Funzione cascata
4	

- **REGISTRAZIONE DEI TEST**

ITEM	**FUNCTIONS DESCRIPTION**	**RESULT**			**REMARKS**
		OK	*NO*	*N.A.*	
	Verifica tag HMI	x			
	Scambio dati fra CPU1-CPU2 PLC1	x			
	Simulazione cascata e fotocellule	x			
	Test aumenta diminuisci velocità cascata	x			
	Scambio dati PLC2-PLC3	x			
	Scambio dati PLC4-PLC5	x			
	Scambio dati PLC2-PLC6	x			

N.A. = Not applicable.
OK= The function test was successful.
NO= The function test was negative.

I test eseguiti sul software sono stati completati con esito positivo: Non sussistono ulteriori elementi di non conformità funzionale o prestazionale.
Il progetto software è quindi idoneo in relazione all'uso per il quale è stato sviluppato.

Team leader	**Date**	**Release**
.....	*xx / xx / xx*	.....

ALLEGATO 2

Ansaldo Sistemi Industriali S.p.A.

TECHNICAL DOCUMENTATION LIBRARIES

SIEMENS S7 V5.4

Numero	*Rev.*	*Data*
XXX 000	**00**	**xx/xx/10**

	Redatto	*Controllato*	*Approvato*
NOME	*F. Menardi*	*A. Rossi*	*x*
FIRMA			
NOME	*X*	*X*	*X*
FIRMA			

GRADO DI RISERVATEZZA PER ASI		**Per uso interno**			
STATO DI REVISIONE DEL DOCUMENTO		**Prima emissione**			
NUMERO DI PAGINE TOTALI		FILE		*COPIA NUMERATA:*	**NO**

INDICE

			(display)		

BIT_ STANDARD					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FC119	Bit standard		Creation of standard bit		16

MATH_FUNCTION					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB30	INTGR		Integrator		30
FC114	Y_MX_Q		Function of the line y=mx+q		31
FC117	Gray t binary 16bit		Conversion from gray code to bynary code, 16 bit		32
FC118	Gray to binary 32bit		Conversion from gray code to bynary code, 32 bit		33

VARIOUS_FUNCTIONS					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FC8	BoolsToInt		Copy of 16bit into an INT with a chance to SWAP of two byte.		57
FC9	IntToBools		Copy of INT into 16 bit with a chance to SWAP of two byte.		60
FC10	BoolsToWord		Copy of 16bit into a WORD with a chance to SWAP of two byte.		58
FC11	WordToBools		Copy of WORD into 16 bit with a chance to SWAP of two byte.		62
FC15			Generetion rising edge anf falling edge		59
FC37	toS5TIME		Conversion of “timebase + moliplicator” in a format S5TIME		61

HSC_1_1					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB28	HscMng		HSC management and possibly simulation		24
FC28	HSCmanagement		HSC management and possibly simulation		26

LIMITS_1_0					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FC20	IntegerLimit		INT limitatin		27
FC21	DoubleIntegerLimit		DINT limitation		28
FC22	RealLimit		REAL limitation		29

SEARCH_FOULTS_PANEL					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB107	Panel_CK		Panel lookforfalse		39
FC107	RD_panel		Reading CercaGuasti panel with tabs 5.543.2…		42

SERIAL_LED_PANEL					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB107	Panel_CK		Panel lookforfalse		39
FB108	BCD panel write		Serial led panel 5.343.1D.		47
FC108	WR_panel		Writing LedSeriale panel with tabs 5.543.2…		48

PC_DESK_5_2					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB12	SE_TimerFb	F. Samuele	Extended pulse timer		34
FB90	MainDeskMng	A.Rossi	PC desk		35

POSITIONATOR					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FC125	P-REG		Generic position controller		36

PT100					
	SYMBOLIC ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB14	PT100		PT100 reading, filtering, absolute pre-alarm end alarm		37
FB15	PT100DerivativeAlarm		Alarm temperature derivatives		38

EXCHANGE_DATE_CPU1_3					
	ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB101	S7-400<>S7-400 COM		Exchange data between S7-400 e S7-400		17
FB102	S7-400<>S7-300 COM		Exchange data between S7-400 e S7-300		18
FC259			DB creation (DP/DP interface coupler 6ES7 158)		19

SEQUANCER 1_4					
	ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB105	SEQ_Managem		Sequencer management		43
FB106	SEQ_Deco		Decimal flag for sequencers		46
FC105	SEQ_Aux		Copy the “step-timeout-time”in step for sequencers.		45

SEQUANCER 2_1					
	ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB105	SEQ_Managem		Sequencer management		43
FB106	SEQ_Deco		Decimal flag for sequencers		46
FC105	SEQ_Aux		Copy the “step-timeout-time”in step for sequencers.		45

TIMER_3_1					
	ID.	**Autore**	**SHORT DESCRIPTION**	**Chap.**	**Page.**
FB10	SD_TimerFb		Timer delayed excitation		53
FB11	SF_TimerFb		Timer delayed disexcitation		51
FB12	SE_TimerFb		Extended pulse timer		34
FB13	SP_TimerFb		Timer pulse		55
FC100	SD_Timer		Delayed timer		54
FC101	SF_Timer		Off-delayed timer		52
FC102	SE_Timer		Extended pulse timer.		49
FC103	SP_Timer		Pulse timer		56

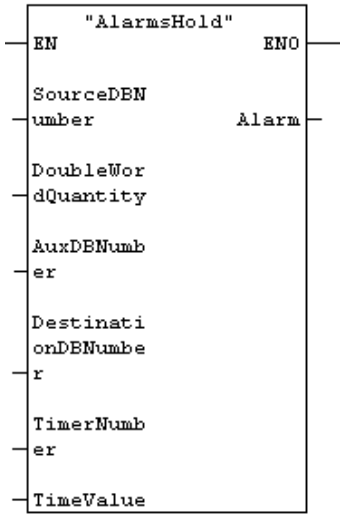
SHORT DESCRIPTION: Alarms temporization.

IN

Name	Type	Description
SourceDBNumber	Int	
DoubleWordQuantity	Int	
AuxDBNumber	Int	
DestinationDBNumber	Int	
TimerNumber	Timer	
TimeValue	S5time	

OUT

Name	Type	Description
Alarms	Bool	



PARAMETERS DESCRIPTION:

SourceDBNumber: number of DB that contains all the alarms, integer value.
DoubleWordQuantity: double word number of alarms that you want to delay when alarms are not excited, integer value.
AuxDBNumber: ausiliary DB number necessary for the operation of FC109, integer value.
DestinationDBNumber: cumulative DB number of the alarms accessed by the supervisor, integer value.
TimerNumber: timer number, integer value.
TimeValue: timer value, S5TIME value.

FUNCTIONING:

This function is designed to delay the alarms dis-excitation to ensure the possibility of the supervisor to read.

DEPENDENCIES:

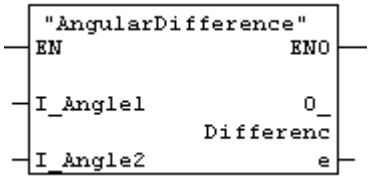
NOTES:

In this DB an alarm must have lasted at least a scan OB1 (not less, like 10ms cycle scan time).
If the "DB source" does not exist, or has less size than, that declared by "DoubleWordQuantity", the FC ends and the output "Alarm" is activated.
"DB source" and the "DB destination" must have a double word size greater than, or equal to,"DoubleWordQuantity", otherwise the alarm at the FC109 output will be active.
If the DB is not present or have a wrong size the function will create or recreate the DB and it will be size with the correct value.
If the DB does not exist nor has less size than that declared by the "DoubleWordQuantity" parameter, the FC end and the "Alarm" will be active. In this DB the alarms will be delayed with dis-excitation a time ranging from 1 to 2 times the value of "TimeValue".
The alarms dis-excitation will be made with a delay of 1 to 2 times the values of this value.

SHORT DESCRIPTION: Calculating difference between two angles.

IN		
Name	**Type**	**Description**
I_Angle1	Int	Angle number 1 (DEGREES)
I_Angle2	Int	Angle number 2 (DEGREES)

OUT		
Name	**Type**	**Description**
O_Difference	Int	Difference result (DEGREES)

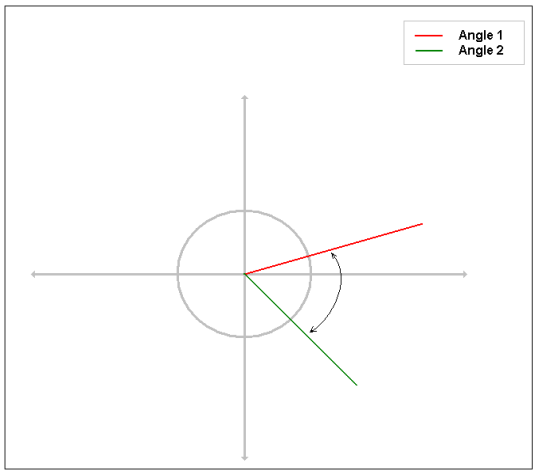


PARAMETERS DESCRIPTION:

I_Angle1: Value of the first angle, from 0 to 359 degrees, integer value.
I_Angle2: Value of the second angle, from 0 to 359 degrees, integer value.
O_Difference: Result of difference operation, degrees, integer value.

FUNCTIONING:

This function is designed to calculate the shortest angular difference between two angles.



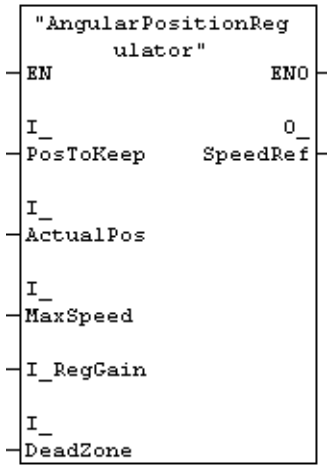
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Angular position regulator.

IN		
Name	**Type**	**Description**
I_PosToKeep	Int	Angular position that must be kept, INT, deg/10
I_ActualPos	Int	Actual angular position, INT, deg/10
I_MaxSpeed	Int	Max motor speed, INT, rpm or % or ....
I_RegGain	Int	Regulator gain, INT, %/10
I_DeadZone	Int	Regulator dead zone, INT, deg/10

OUT		
Name	**Type**	**Description**
O_SpeedRef	Int	Speed reference, INT, normalized at max speed unit



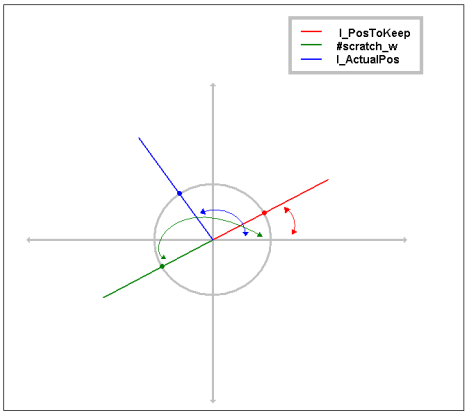
PARAMETERS DESCRIPTION:

I_PosToKeep: Angular position that must be kept, integer value. [Deg/10]
I_ActualPos: Actual angular position, integer value. [Deg/10]
I_MaxSpeed: Max motor speed, integer value. [Rpm or % or ....]
I_RegGain: Regulator gain, integer value. [%/10]
I_DeadZone: Regulator dead zone, integer value. [Deg/10]

O_SpeedRef: Speed reference normalized at max speed unit, integer value.

FUNCTIONING:

This function is projected to regulate the angular position of the motor. It calculates the reference speed of the motor with the references of actual angular position, the maximum motor speed, the regulator dead zone and the regulator gain.

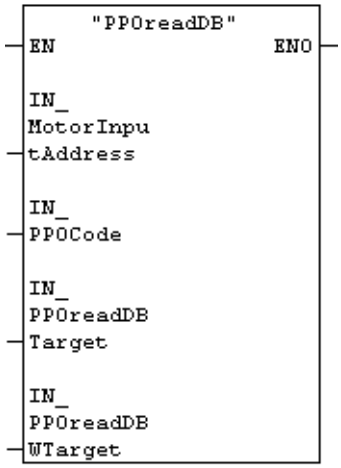


DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: PPO read and transfers to DB area.

IN		
Name	**Type**	**Description**
IN_MotorInputAddress	Word	Base periphery byte address
IN_PPOCode	Int	PPO code number between 1 and 5
IN_PPOreadDBTarget	Int	DB target number
IN_PPOreadDBWTarget	Int	Target area first DBW address



PARAMETERS DESCRIPTION:

IN_MotorInputAddress: Base periphery byte address, word value.
IN_PPOCode: PPO code number, it must be an integer between 1 and 5.
IN_PPOreadDBTarget: DB target number, integer value.
IN_PPOreadDBWTarget: Target area first DBW address, integer value.

FUNCTIONING:

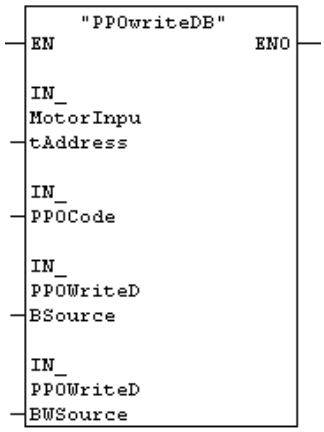
This function is designed to get data from the device.

DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: PPO writes from DB area.

IN		
Name	**Type**	**Description**
IN_MotorInputAddress	Word	Base periphery byte address
IN_PPOCode	Int	PPO code number between 1 and 5
IN_PPOWriteDBSource	Int	DB source number
IN_PPOWriteDBWSource	Int	Source area first DBW address



PARAMETERS DESCRIPTION:

IN_MotorInputAddress: Base periphery byte address, word value.
IN_PPOCode: PPO code number, it must be an integer between 1 and 5.
IN_PPOreadDBTarget: DB source number, integer value.
IN_PPOreadDBWTarget: Source area first DBW address, integer value.

FUNCTIONING:

This FC is designed to put data to the device.

DEPENDENCIES:

NOTES:

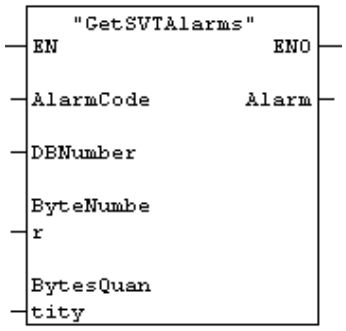
SHORT DESCRIPTION: Decoding bit alarm code on DB... (Inverter and VeCon INTEL).

IN

Name	Type	Description
AlarmCode	Word	MB byte of PZD3 for Vecon, PZD3 for Intel
DBNumber	Int	Destination Data Block
ByteNumber	Int	Destination Byte Number
BytesQuantity	Int	Bytes quantity for alarms bit representation

OUT

Name	Type	Description
Alarm	Bool	Wrong parameters assignment



PARAMETERS DESCRIPTION:

AlarmCode: MB byte of PZD3 for Vecon, for PZD3 Intel, word value.
DBNumber: Destination of data block, integer value.
ByteNumber: Destination of byte number, integer value.
BytesQuantity: Bytes quantity for alarms bit representation, integer value.

Alarm: Wrong parameters assignment, boolean value.

FUNCTIONING:

This function is designed to retrieves the SVT alarms.

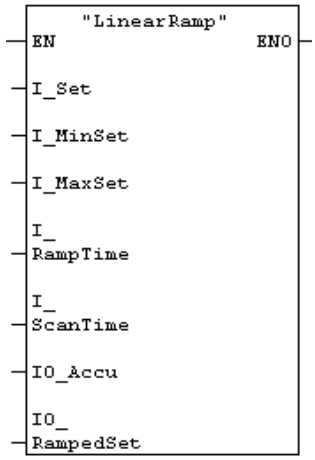
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Linear Ramp.

IN		
Name	**Type**	**Description**
I_Set	Int	Target setpoint
I_MinSet	Int	Minimum setpoint
I_MaxSet	Int	Maximum setpoint
I_RampTime	Int	Time between min and max setpoint (csec)
I_ScanTime	Real	Scan time of the caller OB (msec)

IN-OUT		
Name	**Type**	**Description**
IO_Accu	Real	Ramping function accumulator
IO_RampedSet	Int	Setpoint ramped



PARAMETERS DESCRIPTION:

I_Set: Destination value, desired size, integer value.
I_MinSet: Initial value V1 for calculation of the slope, it must have the same format of "IN_Set", integer value.
I_MaxSet: Final value V2 for calculation of the slope, it must have the same format of "IN_Set", integer value.
I_RampTime: Time value from V1 to V2, integer value. [hundredths of seconds]
I_ScanTime: Duration of the previous scan, if FC41 call in OB1. Value range called OB-time if FC41 call one of these OB, real value. [ms]
IO_Accu: Memory output level achieved, real value.
IO_RampedSet: Exit ramp, same format "IN_Set", integer value.

FUNCTIONING:

This function is designed to get linear ramp. It's possible to get some possible uses:

- With "IN_MinSet", "IN_MaxSet", "IN_RampTime", is calculated the slope to be followed to obtain the variation (variation equal to the difference between the value output and the input value in the scan where it is changed).
- Putting on "IN_MinSet" and on "IN_MaxSet" the minimum and maximum range of input you get a fixed slope ramp. The jump of reference will be made in time proportional to "IN_RampTime".
- Putting on "IN_MinSet" and on "IN_MaxSet" two values such that "INMaxSet" and "INMinSet" are equal to the jump-Reference wanted, 's will be done in time "IN_RampTime".
- By varying the three above values is therefore possible to change the slope at any time, even ramp "already gone". The new slope from the start of the value reached when the three parameters are changed.

DEPENDENCIES:

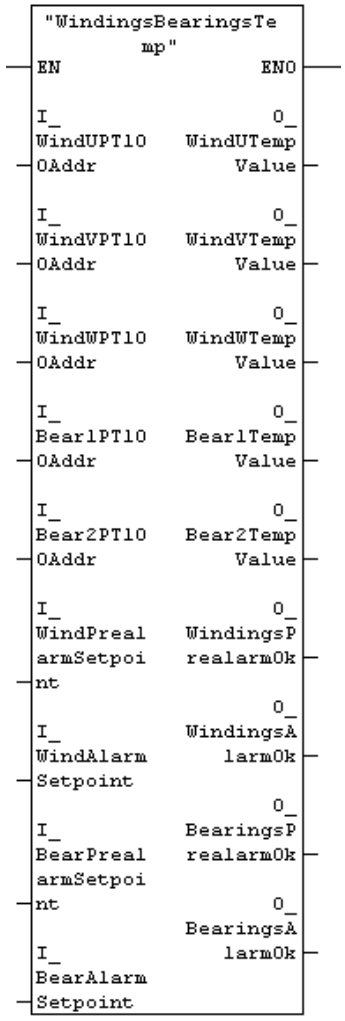
NOTES:

It always must be "IN_MinSet" < "IN_MaxSet".

SHORT DESCRIPTION: PT100 management: windings and engine bearing.

IN		
Name	**Type**	**Description**
I_WindUPT100Addr	Int	Winding phase U PT100 an.input address (PIW number, negative if there is not)
I_WindVPT100Addr	Int	Winding phase V PT100 an.input address (PIW number, negative if there is not)
I_WindWPT100Addr	Int	Winding phase W PT100 an.input address (PIW number, negative if there is not)
I_Bear1PT100Addr	Int	Bearing number 1 PT100 an.input address (PIW number, negative if there is not)
I_Bear2PT100Addr	Int	Bearing number 2 PT100 an.input address (PIW number, negative if there is not)
I_WindPrealarmSetpoint	Int	Windings temperature prealarm setpoint (deg./10)
I_WindAlarmSetpoint	Int	Windings temperature alarm setpoint (deg./10)
I_BearPrealarmSetpoint	Int	Bearings temperature prealarm setpoint (deg./10)
I_BearAlarmSetpoint	Int	Bearings temperature alarm setpoint (deg./10)

OUT		
Name	**Type**	**Description**
O_WindUTempValue	Int	Winding phase U temperature value (deg./10)
O_WindVTempValue	Int	Winding phase V temperature value (deg./10)
O_WindWTempValue	Int	Winding phase W temperature value (deg./10)
O_Bear1TempValue	Int	Bearing number 1 temperature value (deg./10)
O_Bear2TempValue	Int	Bearing number 2 temperature value (deg./10)
O_WindingsPrealarmOk	Bool	Windings temperature prealarm ok (temp. < setpoint)
O_WindingsAlarmOk	Bool	Windings temperature alarm ok (temp. < setpoint)
O_BearingsPrealarmOk	Bool	Bearings temperature prealarm ok (temp. < setpoint)
O_BearingsAlarmOk	Bool	Bearings temperature alarm ok (temp. < setpoint)



PARAMETERS DESCRIPTION:

I_WindUPT100Addr: Winding phase U PT100 an input addr, PIW number, negative if there isn't, integer value.
I_WindVPT100Addr: Winding phase V PT100 an input addr, PIW number, negative if there isn't, integer value.
I_WindWPT100Addr: Winding phase W PT100 an input addr, PIW number, negative if there isn't, integer value.
I_Bear1PT100Addr: Bearing number 1 PT100 an input addr, PIW number, negative if there isn't, integer value.
I_Bear2PT100Addr: Bearing number 2 PT100 an input addr, PIW number, negative if there isn't, integer value.
I_WindPrealarmSetpoint: Windings temperature pre-alarm set-point, integer value. [Deg/10]
I_WindAlarmSetpoint: Windings temperature alarm set-point, in deg/10, integer value. [Deg/10]
I_BearPrealarmSetpoint: Bearings temperature pre-alarm set-point, in deg/10, integer value. [Deg/10]
I_BearAlarmSetpoint: Bearings temperature alarm set-point, in deg/10, integer value. [Deg/10]

O_WindUTempValue: Winding phase U temperature value, integer value. [Deg/10]
O_WindVTempValue: Winding phase V temperature value, integer value. [Deg/10]
O_WindWTempValue: Winding phase W temperature value, integer value. [Deg/10]
O_Bear1TempValue: Bearing number 1 temperature value, integer value. [Deg/10]
O_Bear2TempValue: Bearing number 2 temperature value, integer value. [Deg/10]
O_WindingsPrealarmOk: Windings temperature pre-alarm ok, temp. < set-point, boolean value.
O_WindingsAlarmOk: Windings temperature alarm ok, temp. < set point, boolean value.

O_BearingsPrealarmOk: Bearings temperature pre-alarm ok, temp. < set point, boolean value.
O_BearingsAlarmOk: Bearings temperature alarm ok, temp. < set point, boolean value.

FUNCTIONING:

This function is designed to check temperature in motor windings and bearings.

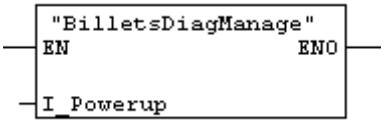
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Registration of cuts in hot shear.

IN

Name	Type	Description
I_Powerup	Bool	



PARAMETERS DESCRIPTION:

I_Powerup: Enable the registration of cuts in hot shear, boolean value.

FUNCTIONING:

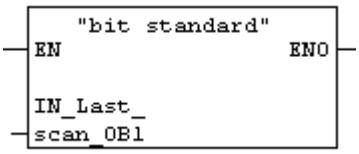
This function is designed to get the diagnostic for bar cuts in billet.

DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Management of useful "standard" bits and values.

IN		
Name	**Type**	**Description**
IN_Last_scan_OB1	Int	last scan time of ob1 (ms).



PARAMETERS DESCRIPTION:

IN_Last_scan_OB1: scan time of last OB1, integer value. [ms]

FUNCTIONING:

The purpose of this function is to manage following useful bits and value (used for other functions programming):

- HEART BIT, one scan true, one scan false (M100.1)
- ALWAYS ON (M100.2)
- ALWAYS OFF (M100.3)
- 100 ms oscillator (M100.4)
- 200 ms oscillator (M100.5)
- 500 ms oscillator (M100.6)
- 1 s oscillator (M 100.7)
- OB1 scan time, integer, ms (MW102)
- OB1 scan time, real, ms (MD104)

DEPENDENCIES:

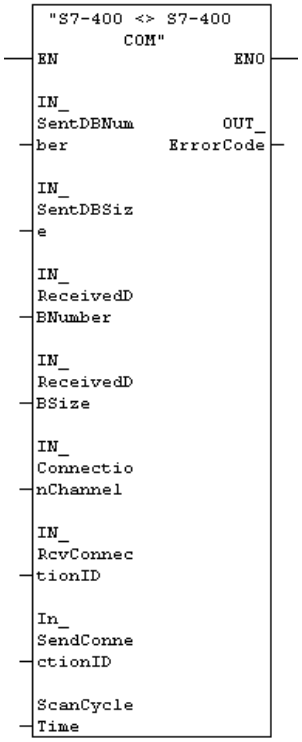
NOTES:

This function must be called at the beginning of OB1.
There are two versions of this function: one for S7-300 system and one for S7-400 system. The second one calls the SFC78 that is not supported by S7-300.

SHORT DESCRIPTION: Exchange data between CPU: S7-400 with S7-400.

IN		
Name	**Type**	**Description**
IN_SentDBNumber	Int	The data block that will be sent to the remote station
IN_SentDBSize	Int	The size of the sent data block
IN_ReceivedDBNumber	Int	The data block that will be received from the remote station
IN_ReceivedDBSize	Int	The size of the received data block
IN_ConnectionChannel	Int	The CONNECTION CHANNEL used for data exchange (set in NETPRO)
IN_RcvConnectionID	Word	To make a connection UNIQUE in the network configuration (RECEIVE)
In_SendConnectionID	Word	To make a connection UNIQUE in the network configuration (SEND)
ScanCycleTime	Int	Previous scan cycle time

OUT		
Name	**Type**	**Description**
OUT_ErrorCode	Int	Difference from zero means error



PARAMETERS DESCRIPTION:

IN_SentDBNumber: Data block that will be sent to the remote station, integer value.
IN_SentDBSize: Size of the sent data block, integer value.
IN_ReceivedDBNumber: Data block that will be received from the remote station, integer value.
IN_ReceivedDBSize: Size of the received data block, integer value.
IN_ConnectionChannel: Connection channel used for data exchange (set in NETPRO), integer value.
IN_RcvConnectionID: To make a connection unique in the network configuration (RECEIVE), word value.
In_SendConnectionID: To make a connection unique in the network configuration (SEND), word value.
ScanCycleTime: Previous scan cycle time, integer value.

OUT_ErrorCode: Difference from zero means error, integer value.

FUNCTIONING:

This function is designed to perform the exchange of data block between CPU S7-400.

DEPENDENCIES:

NOTES:

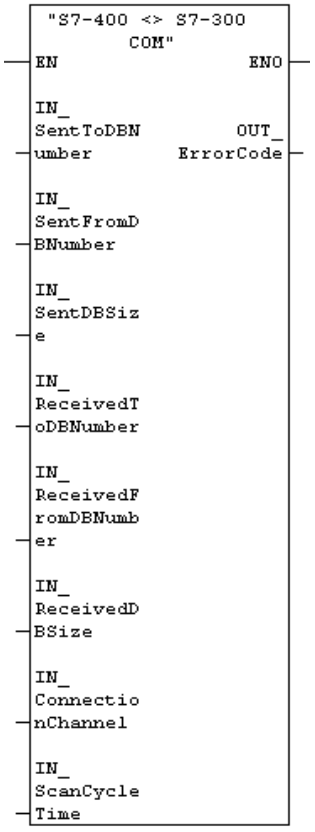
SHORT DESCRIPTION: Exchange data between CPU: S7-400 with S7-300.

IN

Name	Type	Description
IN_SentDBNumber	Int	Data destination on remote cpu
IN_SentFromDBNumber	Int	Data source on local cpu
IN_SentDBSize	Int	The size of the sent data (MAX 240 Bytes)
IN_ReceivedToDBNumber	Int	Data destination on local cpu
IN_ReceivedFromDBSize	Int	Data source on remote cpu
IN_ReceivedDBSize	int	The size of the received data (MAX 240 Bytes)
IN_ConnectionChannel	Int	The CONNECTION CHANNEL used for data exchange (set in NETPRO)
IN_ScanCycleTime	Int	

OUT

Name	Type	Description
OUT_ErrorCode	Int	



PARAMETERS DESCRIPTION:

IN_SentDBNumber: Data destination on remote CPU, integer value.
IN_SentFromDBNumber: Data source on local CPU, integer value.
IN_SentDBSize: The size of the sent data (MAX 240 Bytes), integer value.
IN_ReceivedToDBNumber: Data destination on local CPU, integer value.
IN_ReceivedFromDBSize: Data source on remote CPU, integer value
IN_ReceivedDBSize: The size of the received data (MAX 240 Bytes), integer value
IN_ConnectionChannel: Connection channel used for data exchange (set in NETPRO), integer value.
IN_ScanCycleTime: Previous scan cycle time, integer value.

OUT_ErrorCode: Difference from zero means error, integer value.

FUNCTIONING:

This function is designed to perform the exchange of data block between CPU S7-400 and CPU S7-300.

DEPENDENCIES:

NOTES:

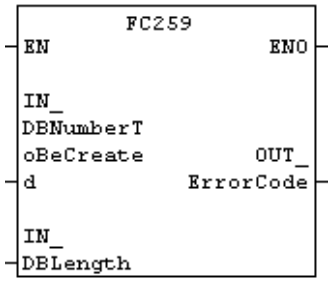
SHORT DESCRIPTION: DB creation (DP/DP interface coupler 6ES7 158).

IN

Name	Type	Description
IN_DBNumberToBeCreated	Int	
IN_DBLength	Int	

OUT

Name	Type	Description
OUT_ErrorCode	Int	



PARAMETERS DESCRIPTION:

IN_DBNumberToBeCreated: Number of DB to create, integer value.
IN_DBLength: Length f the DB to create, integer value.

OUT_ErrorCode: This out notifies that an error has occurred, integer value.

FUNCTIONING:

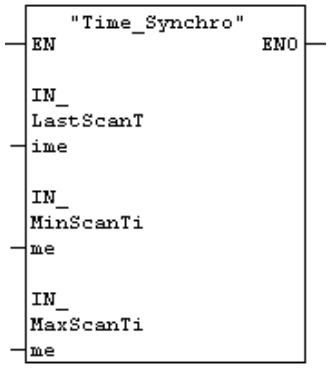
This function is designed to create and re-create DB.

DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Date and time management in PLC.

IN		
Name	**Type**	**Description**
IN_LastScanTime	Int	coming from OB1
IN_MinScanTime	Int	coming from OB1
IN_MaxScanTime	Int	coming from OB1



PARAMETERS DESCRIPTION:

IN_LastScanTime: Last scan time, value coming from OB1, integer value.
IN_MinScanTime: Minimum scan time, value coming from OB1, integer value.
IN_MaxScanTime: Maximum scan time, value coming from OB1, integer value.

FUNCTIONING:

This function is designed to date and time management in PLC. Using this block we can synchronize PLC date and time with the one of PC supervisor data and time, and visualize actual, minimum and maximum cycle time.
This block is used together with DB14 which has defined structure to exchange data between PLC and PC.
In DB14 there is also 1 word (DBW70) with alarms .It is possible to use it (DBW70) in the "alarm DB".
Below it is an example of call and parameter assigment.

```
CALL FC14
   IN_LastScanTime:=      #OB1_PREV_CYCLE
   IN_MinScanTime:=       #OB1_MIN_CYCLE
   IN_MaxScanTime:=       #OB1_MAX_CYCLE
```

DEPENDENCIES:

NOTES:

FC14 should be called in OB1.

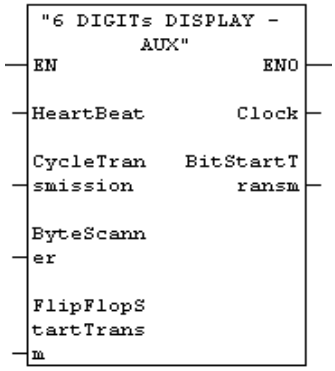
SHORT DESCRIPTION: 6-digit display with cards 5.295-6-7 (general).

IN-OUT

Name	Type	Description
HeartBeat	Bool	HEART BEAT
CycleTransmission	Int	
ByteScanner	Word	Byte scanner (7-0)Mem
FlipFlopStartTransm	Bool	

OUT

Name	Type	Description
Clock	Bool	CLOCK
BitStartTransm	Bool	



PARAMETERS DESCRIPTION:

HeartBeat: Heart Beat, boolean value.
CycleTransmission: Transmission cycle, it is 15 scans of OB11, integer value.
ByteScanner: This word points the date to transmit from bit 7 to bit 0, word value.
FlipFlopStartTransm: bit of transmission state that is high one time and low the other time, boolean value.

Clock: clock time, boolean value.
BitStartTransm: bit of start transmission that is generated at the beginning of every transmission start, it is projected to syncronize FC39 events, boolean value.

FUNCTIONING:

This function and FC39 are responsible for managing the display 5.295.0D.
FC39 is called N times in function of display number, instead this function is called only one time in OB11, and before any FC39 call.
FC38 is designed to generate the utility bits for the FC39.

DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: 6-digit display with cards 5.295-6-7 (display).

IN

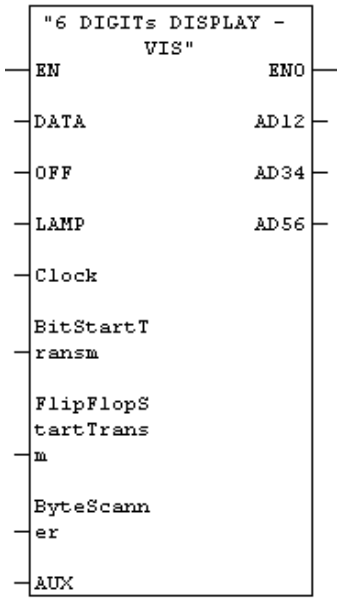
Name	Type	Description
DATA	DWord	DATA
OFF	Bool	OFF
LAMP	Bool	LAMP
Clock	Bool	
BitStartTransm	Bool	
FlipFlopStartTransm	Bool	
ByteScanner	Word	

OUT

Name	Type	Description
AD12	Bool	AD12
AD34	Bool	AD34
AD56	Bool	AD56

IN-OUT

Name	Type	Description
AUX	DWord	AUX



PARAMETERS DESCRIPTION:

DATA: Data, double word value.
OFF: If this input is 1 the display is off, boolean value
LAMP: If this input is 1 the display "lampeggia", boolean value.
Clock: It is created in FC38, boolean value.
BitStartTransm: It is created in FC38, boolean value.
FlipFlopStartTransm: It is created in FC38, boolean value.
ByteScanner: It is created in FC38, word value.

AD12: It is the output of the 1-2 display nibbles, this input is used to send serial data to 8-input of 5.295.0D.
AD34: It is the output of the 3-4 display nibbles, this input is used to send serial data to 7-input of 5.295.0D.
AD56: It is the output of the 5-6 display nibbles, this input is used to send serial data to 6-input of 5.295.0D.

AUX: This input is for internal use, double word.

FUNCTIONING:

This function and FC38 are responsible for managing the display 5.295.0D.
FC39 is called N times in function of display number, instead this function is called only one time in OB11, and before any FC39 call.
FC38 is designed to generate the utility bits for the FC39.

CALL FC 39
DATA:= Double word where you write the visualized date.
OFF:= If you not use, you must insert F100.3
LAMP:= If you not use, you must insert F100.3
Clock:= Bit equal to FC38, where clock is generated.

BitStartTransm:= Bit equal to FC38.
FlipFlopStartTransm:= Bit equal to FC38.
ByteScanner:= Word equal to FC38
AD12:= you can use the Q
AD34:= you can use the Q
AD56:= you can use the Q
AUX:= Double word for internal memory (don't use scractch)

DEPENDENCIES:

NOTES:

You mustn't shift temporary date "Disp1_2Aux", "Disp3_4Aux", "Disp5_6Aux" from temp 0.0,1.0,2.0 because they are used like puntators.

HSC SIMULATION WITH ANY SIMULATION ("HscMng") (FB28)

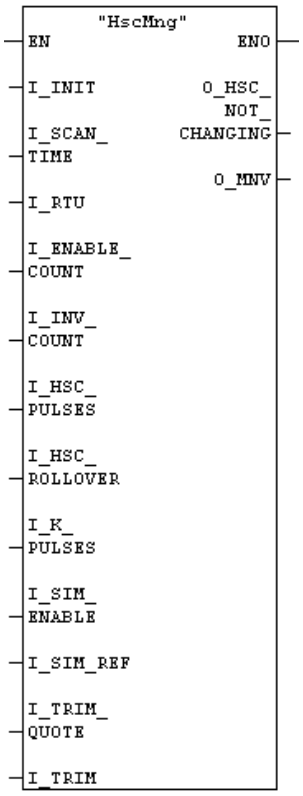
SHORT DESCRIPTION: HSC management with any simulation.

IN

Name	Type	Description
I_INIT	Bool	Init flag
I_SCAN_TIME	Real	Scan time, REAL
I_RTU	Real	Rate Time Unit [s] (e.g. h ==> Rtu=3600.0), REAL
I_ENABLE_COUNT	Bool	Enable count
I_INV_COUNT	Bool	Invert count
I_HSC_PULSES	Dint	Value from hardware HSC, DINT
I_HSC_ROLLOVER	Dint	HSC rollover value (if 0 rollover not managed), DINT
I_K_PULSES	Real	Pulses quantity related to unit of O_MNV, REAL
I_SIM_ENABLE	Bool	Simulation enable (using I_SIM_REF parameter)
I_SIM_REF	Real	Simulation reference, REAL
I_TRIM_QUOTE	Real	Trim quote (O_MNV preset value), REAL
I_TRIM	Bool	Trim preset command

OUT

Name	Type	Description
O_HSC_NOT_CHANGING	Bool	HSC not changing.
O_MNV	Real	Manipulated value [enginnering unit], REAL



PARAMETERS DESCRIPTION:

I_INIT: Must be TRUE while PLC is running 1st scan after start up, boolean value.
I_SCAN_TIME: Elapsed time since previous call of the function. See I_SIM_REF description, real value.
I_RTU: Rate time unit for scan time. See I_SIM_REF description, real value.
I_ENABLE_COUNT: Enables O_MNV increasing/decreasing using I_HSC_PULSES span or using I_SIM_REF integration when I_SIM_ENABLE = true, boolean value.
I_INV_COUNT: Enables inversion of counting direction, boolean value.
I_HSC_PULSES: Counting value from external HSC, double integer value.
I_HSC_ROLLOVER: Rollover value of external HSC (e.g. rollover = 8192 if HSC value is from 0 to 8191), double integer value.
I_K_PULSES: Pulses quantity related to unit of "O_MNV". If 12 pulses correspond to 1 mm and you want that "O_MNV" works in mm, "I_K_PULSES" = 12.0. If 12 pulses correspond to 1 mm and you want that "O_MNV" works in m, "I_K_PULSES" = 12000.0. In case of simulation ("I_SIM_ENABLE" = true) this parameter determs the "O_MNV" increasing/decreasing step precision.
E.g.: "I_K_PULSES" = 1000.0, "I_SIM_REF" = m/s --> "O_MNV" = m with precision of 1 mm.
I_SIM_ENABLE: Enables O_MNV increasing/decreasing using speed (I_SIM_REF) integration (if "I_ENABLE_COUNT" = true).
I_SIM_REF: Speed value for integration: measurement unit must be congruent with "I_SCAN_TIME" and "I_RTU" plus unit of "O_MNV".
E. g.: "I_SIM_REF" = mm/s if "O_MNV" = mm, "I_SCAN_TIME" = ms, "I_RTU" = 1000.0.
I_TRIM_QUOTE: Loading value for "O_MNV" presetting. The unit must be the same of "O_MNV".
I_TRIM: Command for "O_MNV" presetting.

O_HSC_NOT_CHANGING: True when "O_MNV" is not increasing or decreasing. Must be time filtered outside the function.
O_MNV: Manipulated value. Units depends on previous parameters.

FUNCTIONING:

This function is designed to perform the HSC management and simulation.

DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: HSC management and possibly simulation.

IN

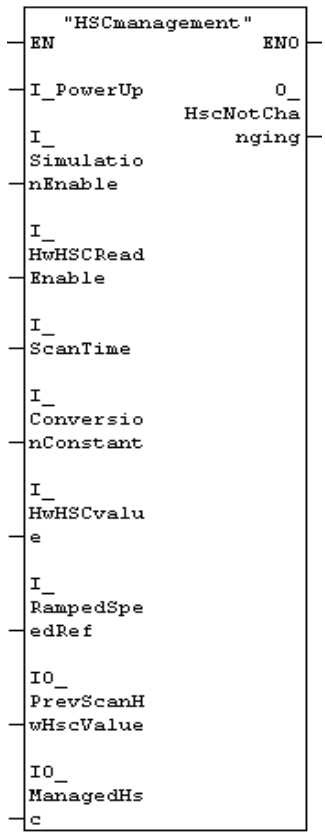
Name	Type	Description
I_PowerUp	Bool	Power up flag
I_SimulationEnable	Bool	Simulation enable (using ramped speed ref.)
I_HwHSCReadEnable	Bool	Enable to read I_HwHSCValue (if false Delta is kept to 0)
I_ScanTime	Real	Scan time, ms, REAL
I_ConversionConstant	Real	Constant for conversion from pulses to wanted unit,REAL
I_HwHSCvalue	Dint	Value from hardware HSC, DINT
I_RampedSpeedRef	Real	Ramped speed ref, for simulation, [Wanted unit (see conv costant)]/s, REAL

OUT

Name	Type	Description
O_HscNotChanging	Bool	HSC not changing (actual value = previous value)

IN-OUT

Name	Type	Description
IO_PrevScanHwHscValue	Dint	Previous scan hardware HSC value, DINT
IO_ManagedHsc	Real	Managed HSC, Wanted unit (see conv costant),REAL



PARAMETERS DESCRIPTION:

I_PowerUp: Power up flag, boolean value.
I_SimulationEnable: This flag enable the simulation using ramped speed reference, boolean value.
I_HwHSCReadEnable: This flag enable the reading of "I_HwHSCValue", in particular if it is false Delta is kept to 0, boolean value.
I_ScanTime: Scan time, real value. [ms]
I_ConversionConstant: Constant for conversion from pulses to wanted unit, real value.
I_HwHSCvalue: Value from hardware HSC, double integer value.
I_RampedSpeedRef: Ramped speed ref, for simulation, real value. [Wanted unit (see conv costant)]/s
O_HscNotChanging: This flag is enable when HSC not changing (actual value = previous value), boolean value.
IO_PrevScanHwHscValue: Previous scan hardware HSC value, double integer value.
IO_ManagedHsc: Managed HSC, real value. [Wanted unit (see conv costant)]

FUNCTIONING:

This function is designed to perform the HSC management.

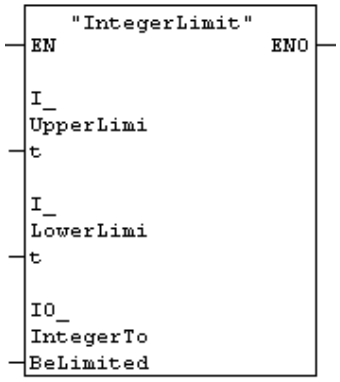
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: INT limitation.

IN		
Name	**Type**	**Description**
I_UpperLimit	Int	Upper limit
I_LowerLimit	Int	Lower limit

IN-OUT		
Name	**Type**	**Description**
IO_IntegerToBeLimited	Int	Integer to be limited



PARAMETERS DESCRIPTION:

I_UpperLimit: Upper limit, integer value.
I_LowerLimit: Lower limit, integer value.
IO_IntegerToBeLimited: Input value to be limited, integer value.

FUNCTIONING:

This function is designed to limit an input into a range introduced by "I_UpperLimit" and "I_LowerLimit" values.

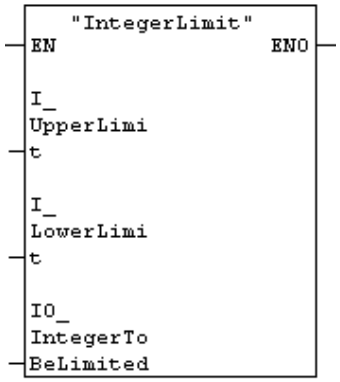
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Double integer limitation.

IN		
Name	**Type**	**Description**
I_UpperLimit	Int	Upper limit
I_LowerLimit	Int	Lower limit

IN-OUT		
Name	**Type**	**Description**
IO_DoubleIntToBeLimited	Dint	Integer to be limited



PARAMETERS DESCRIPTION:

I_UpperLimit: Upper limit, integer value.
I_LowerLimit: Lower limit, integer value.
IO_DoubleIntToBeLimited: Input value to be limited, double integer value.

FUNCTIONING:

This function is designed to limit an input into a range introduced by "I_UpperLimit" and "I_LowerLimit" values.

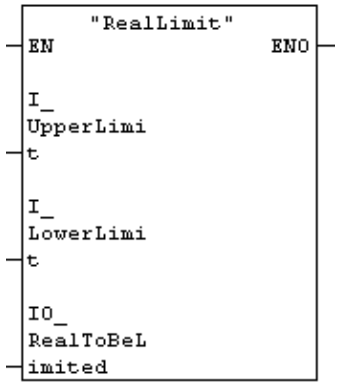
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Real value limitation.

IN		
Name	***Type***	***Description***
I_UpperLimit	Real	Upper limit
I_LowerLimit	Real	Lower limit

IN-OUT		
Name	**Type**	**Description**
IO_RealToBeLimited	Real	Real to be limited



PARAMETERS DESCRIPTION:

I_UpperLimit: Upper limit, real value.
I_LowerLimit: Lower limit, real value.
IO_RealToBeLimited: Input value to be limited, real value.

FUNCTIONING:

This function is designed to limit an input into a range introduced by "I_UpperLimit" and "I_LowerLimit" values.

DEPENDENCIES:

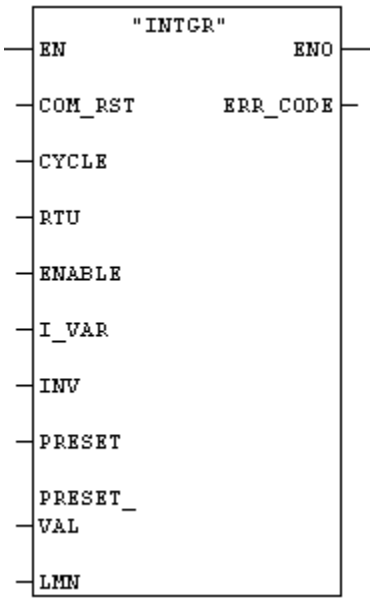
NOTES:

SHORT DESCRIPTION: Mathematical function that performs integration function.

IN		
Name	**Type**	**Description**
Com_rst	Bool	Complete restart, init some internal values
cycle	Real	Sampling time. It maches the calling OB scan time [s]
rtu	Real	Rate time unit [s] (e.g.h➔ rtu=3600.0)
enable	Bool	Enable the integrator
i_var	Real	Variable to integrate
inv	Bool	Invert the sign of the variable to integrate
preset	Bool	Preset to PRESET_VAL
Preset_val	Real	Preset value

OUT		
Name	**Type**	**Description**
Err_code	Bool	Error code

IN-OUT		
Name	**Type**	**Description**
Lmn	Real	Manipolated value (integrated value)



PARAMETERS DESCRIPTION:

Com_rst: Complete restart, init some internal values, boolean value.
Cycle: Sampling time. It maches the calling OB scan time, real value. [s]
Rtu: Rate time unit, real value. [s] (e.g.h ➔ rtu=3600.0).
Enable: Enable the integrator, boolean value.
I_var: Variable to integrate, real value.
Inv: Invert the sign of the variable to integrate, boolean value.
Preset: Preset to PRESET_VAL, boolean value.
Preset_val: Preset value, real value.

Err_code: Error code, boolean value.

Lmn: Manipolated value (integrated value), real value.

FUNCTIONING:

This function is designed to integrate "I_var", the integrated value is available at the "Lmn" output. To get this function work it is necessary to enable boolean input "Enable".

DEPENDENCIES:

NOTES:

FUNCTION OF THE STRAIGHT LINE Y=MX+Q ("Y_MX_Q") (FC114)

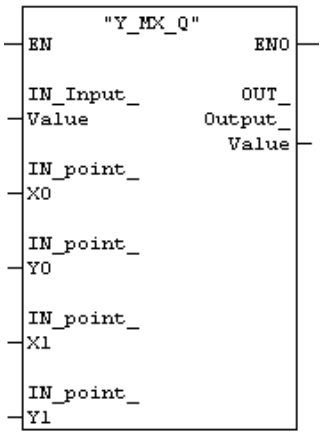
SHORT DESCRIPTION: Function of the straight line "y=mx+q".

IN

Name	Type	Description
IN_Input_Value	Int	Value of input straight function
IN_point_X0	Real	Value of axial coordinate of first point on the straight line
IN_point_Y0	Real	Value of vertical coordinate of first point on the straight line
IN_point_X1	Real	Value of axial coordinate of second point on the straight line
IN_point_Y1	Real	Value of vertical coordinate of second point on the straight line

OUT

Name	Type	Description
OUT_Output_Value	Int	Value of output straight function



PARAMETERS DESCRIPTION:

IN_Input_Value: value of input straight line, integer value.
IN_point_X0: values of axial coordinates of the two points, real value.
IN_point_Y0: values of axial coordinates of the two points, real value.
IN_point_X1: values of axial coordinates of the two points, real value.
IN_point_Y1: values of axial coordinates of the two points, real value.

OUT_Output_Value: output straight function, integer value.

FUNCTIONING:

This FC function is design to calculate the equation of the straight line from the Cartesian coordinates of two points. In particular this function calculates M coefficient that expresses the slope of the straight line.
After this, the function calculates Q coefficient that expresses the intersection of the straight line with the y-axis.
In this way we obtain the expression Y = MX+Q.
The value of the output is an integer that expressed the sum of the equation of the straight line Y=MX+Q, in particular:

"OUT_Output_Value" = M·["IN_Input_Value"] + Q

DEPENDENCIES:

NOTES:

CONVERSION FROM GRAY CODE TO BINARY CODE ("Gray to binary 16bit") (FC117)

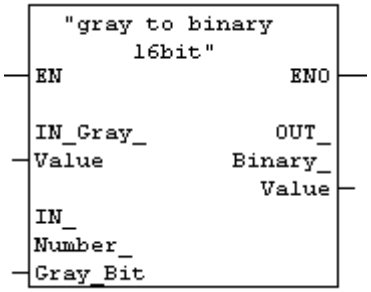
SHORT DESCRIPTION: Conversion from gray code to binary code, 16 bit.

IN

Name	Type	Description
IN_Gray_Value	Word	Input gray value to be converted
IN_Number_Gray_Bit	Int	Number of gray bit to converter (1 to 16)

OUT

Name	Type	Description
OUT_Binary_Value	Word	Output binary value converted by gray input



PARAMETERS DESCRIPTION:

IN_Gray_Value: value of gray code to convert, word value.
IN_Number_Gray_Bit: number of gray bit to converter, from 1 to 16 integer value.
OUT_Binary_Value: binary result value, word value.

FUNCTIONING:

This function will convert data from gray code to binary code, is accepted only value to maximum 16 bit.

DEPENDENCIES:

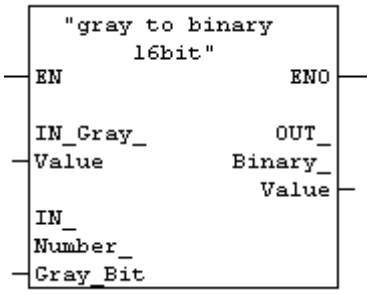
NOTES:

CONVERSION FROM GRAY CODE TO BINARY CODE ("Gray to binary 32bit") (FC118)

SHORT DESCRIPTION: Conversion from gray code to binary code, 32 bit.

IN		
Name	**Type**	**Description**
IN_Gray_Value	Word	Input gray value to be converted
IN_Number_Gray_Bit	Int	Number of gray bit to converter (1 to 32)

OUT		
Name	**Type**	**Description**
OUT_Binary_Value	Word	Output binary value converted by gray input



PARAMETERS DESCRIPTION:

IN_Gray_Value: value of gray code to convert, word value.
IN_Number_Gray_Bit: number of gray bit to converter, from 1 to 32 integer value.
OUT_Binary_Value: binary result value, word value.

FUNCTIONING:

This function will convert data from gray code to binary code, is accepted only value to maximum 32 bit.

DEPENDENCIES:

NOTES:

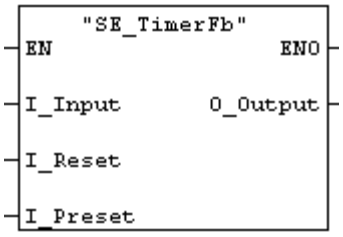
SHORT DESCRIPTION: Extended pulse timer.

IN

Name	Type	Description
I_Input	Bool	
I_Reset	Bool	
I_Preset	DInt	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT

Name	Type	Description
O_Output	Bool	



PARAMETERS DESCRIPTION:

I_Input: Input signal, boolean value.
I_Reset: Enable or disable the timer reset, boolean value.
I_Preset: Preset value, range is from 0 to 2147483647 (more than 5965 hours), double integer value. [s/100]
O_Output: Timer output, boolean value.

FUNCTIONING:

If "I_Input" is setting by a rising edge the output will be set for a period equal to the value entered in "I_Preset" and then return false.
If you enter another positive edge in "I_Input" while "O_Output" is still high, the output will remain high since the last rising edge for a period equal to that shown in "I_Preset".

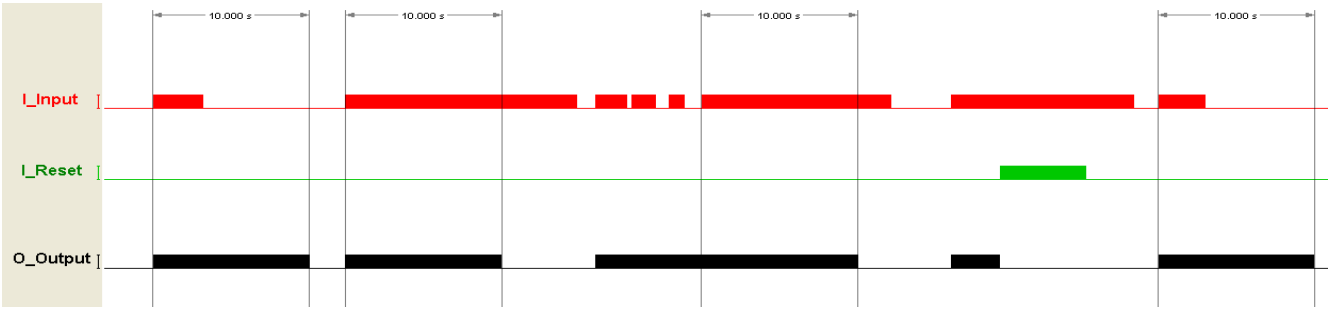


Figura 1 : Simulation.

DEPENDENCIES:

To get this function working, FC99 must be called in 10 ms interupt cycle. MD108 is defined as DINT and used as time counter in cents of s.

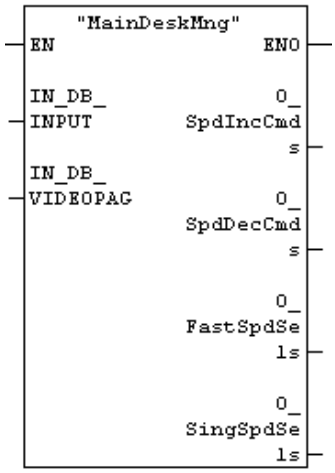
NOTES:

To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

SHORT DESCRIPTION: Pc desk.

IN		
Name	**Type**	**Description**
IN_DB_INPUT	Pointer	
IN_DB_VIDEOPAG	Pointer	

OUT		
Name	**Type**	**Description**
O_SpdIncCmds	DWord	Speed increase commands (devices from 1 to 32)
O_SpdDecCmds	DWord	Speed decrease commands (devices from 1 to 32)
O_FastSpdSels	DWord	Fast speed selections (devices from 1 to 32)
O_SingSpdSels	DWord	Single speed selections (devices from 1 to 32)



PARAMETERS DESCRIPTION:

IN_DB_INPUT: Pointer value for DB input.
IN_DB_VIDEOPAG: Pointer value for DB Video page.

O_SpdIncCmds: Speed increase commands (devices from 1 to 32), double word value.
O_SpdDecCmds: Speed decrease commands (devices from 1 to 32), double word value.
O_FastSpdSels: Fast speed selections (devices from 1 to 32), double word value.
O_SingSpdSels: Single speed selections (devices from 1 to 32), double word value.

FUNCTIONING:

This function is designed to perform the main desk management.

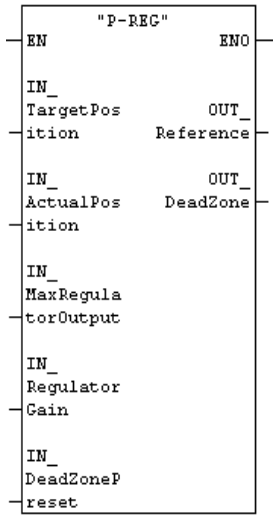
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Generic position controller.

IN		
Name	**Type**	**Description**
IN_TargetPosition	Dint	Target position
IN_ActualPosition	Dint	Actual position
IN_MaxRegulatorOutput	Int	Max value of output
IN_RegulatorGain	Real	constant for gain
IN_DeadZonePreset	Dint	Regulator dead zone (consirering POS-APOS)

OUT		
Name	**Type**	**Description**
OUT_Reference	Int	Output of regulator limited to MAXV: (POS-APOS)*GAIN
OUT_DeadZone	Bool	Signalization when we are in the dead zone



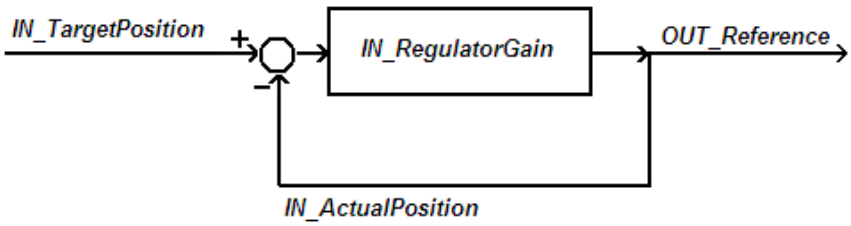
PARAMETERS DESCRIPTION:

IN_TargetPosition: Target position, double integer value.
IN_ActualPosition: Actual position, double integer value.
IN_MaxRegulatorOutput: Maximum value of output, integer value.
IN_RegulatorGain: regulator gain, constant real value.
IN_DeadZonePreset: regulator dead zone, double integer value.

OUT_Reference: output of regulator limited to MAXV: (POS-APOS)*GAIN, integer value.
OUT_DeadZone: Signalization when we are in the dead zone, boolean value.

FUNCTIONING:

This function is designed to regulate the position. The control is based on the retroaction control system; in particular "OUT_Reference" is the output of regulator and is limited to "MAXV" value introduced by "IN_MaxRegulatorPreset".



DEPENDENCIES:

NOTES:

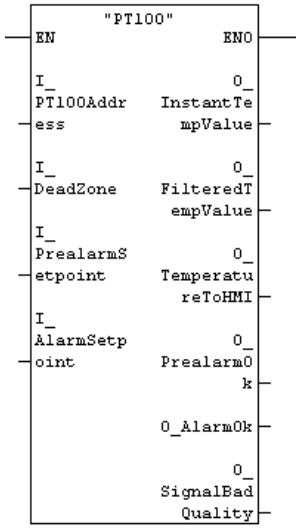
SHORT DESCRIPTION: PT100 reading, filtering, absolute pre-alarm end alarm.

IN

Name	Type	Description
I_PT100Address	Int	PT100 analog input address (PIW number)
I_DeadZone	Int	Dead zone for filter [deg/10].
I_PrealarmSetpoint	Int	Temperature prealarm setpoint [deg/10]
I_AlarmSetpoint	Int	Temperature alarm setpoint [deg/10]

OUT

Name	Type	Description
O_InstanceTempValue	Int	PT100 instantaneus temperature value [deg/10]
O_FilteredTempValue	Int	PT100 filtered temperature value [deg/10]
O_TemperatureToHMI	Int	Temperature value to HMI [deg/10]
O_PrealarmOk	Bool	Temperature prealarm ok (temp. < setpoint)
O_AlarmOk	Bool	Temperature alarm ok (temp. < setpoint)
O_SignalBadQuality	Bool	PT100 signal bad quality



PARAMETERS DESCRIPTION:

I_PT100Address: PT100 analogical input address (PIW number), integer value.
I_DeadZone: Dead zone for filter, integer value. [deg/10]
I_PrealarmSetpoint: Temperature prealarm setpoint, integer value. [deg/10]
I_AlarmSetpoint: Temperature alarm setpoint, integer value. [deg/10]

O_InstanceTempValue: PT100 instantaneus temperature value, integer value. [deg/10]
O_FilteredTempValue: PT100 filtered temperature value, integer value. [deg/10]
O_TemperatureToHMI: Temperature value to HMI, integer value. [deg/10]
O_PrealarmOk: Temperature prealarm ok (temp. < setpoint), boolean value.
O_AlarmOk: Temperature alarm ok (temp. < setpoint), boolean value.
O_SignalBadQuality: PT100 signal bad quality, boolean value.

FUNCTIONING:

This function is designed to manage PT100.

DEPENDENCIES:

NOTES:

In this DB an alarm must have lasted at least a scan OB1 (not less, like 10ms cycle scan time).
If the "DB source" does not exist, or has less size than, that declared by "DoubleWordQuantity", the FC ends and the output "Alarm" is activated.
"DB source" and the "DB destination" must have a double word size greater than, or equal to,"DoubleWordQuantity", otherwise the alarm at the FC109 output will be active.
If the DB is not present or have a wrong size the function will create or recreate the DB and it will be size with the correct value.
If the DB does not exist nor has less size than that declared by the "DoubleWordQuantity" parameter, the FC end and the "Alarm" will be active. In this DB the alarms will be delayed with dis-excitation a time ranging from 1 to 2 times the value of "TimeValue".
The alarms dis-excitation will be made with a delay of 1 to 2 times the values of this value.

DERIVATIVES TEMPERATURE ALARM ("PT100DerivativeAlarm") (FB15)

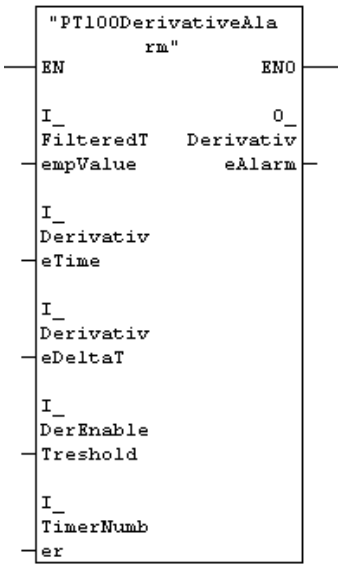
SHORT DESCRIPTION: Derivative temperature alarm.

IN

Name	Type	Description
I_FilteredTempValue	Int	PT100 filtered temperature value [deg/10]
I_DerivativeTime	Int	Temperature variation check time [s/100]
I_DerivativeDeltaT	Int	Maximum temperature variation allowed [deg/10]
I_DerEnableTreshold	Int	Temperature threshold that enables the alarm [deg/10]
I_TimerNumber	Timer	

OUT

Name	Type	Description
O_DerivativeAlarm	Bool	Derivative alarm (1=alarm)



PARAMETERS DESCRIPTION:

I_FilteredTempValue: PT100 filtered temperature value, integer value. [Deg/10]
I_DerivativeTime: Temperature variation checks time, integer value. [S/100]
I_DerivativeDeltaT: Maximum temperature variation allowed; integer value. [Deg/10]
I_DerEnableTreshold: Temperature threshold that enables the alarm, integer value. [Deg/10]
I_TimerNumber: Number of the timer, timer value.

O_DerivativeAlarm: Derivative alarm (1=alarm), boolean value.

FUNCTIONING:

This function is designed to manage the PT100 derivative alarm handling.

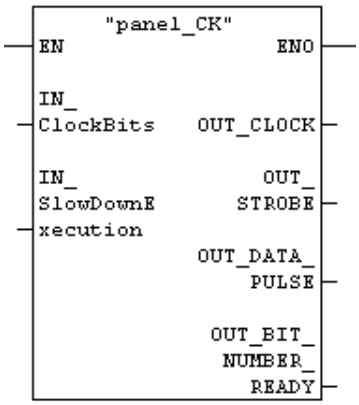
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Search-faults panel.

IN		
Name	**Type**	**Description**
IN_ClockBits	Int	
IN_SlowDownExecution	Int	

OUT		
Name	**Type**	**Description**
OUT_CLOCK	Bool	
OUT_STROBE	Bool	
OUT_DATA_PULSE	Bool	
OUT_BIT_NUMBER_READY	Byte	



PARAMETERS DESCRIPTION:

IN_ClockBits: Input that specify panel bits, integer value.
IN_SlowDownExecution: Input that specify of how many times the execution must to slow down, integer value.

OUT_CLOCK: Clock to be sent to the panel, boolean value.
OUT_STROBE: Strobe to be sent to the panel, boolean value.
OUT_DATA_PULSE: Input that specify if we are reading the bit, boolean value.
OUT_BIT_NUMBER_READY: Bit code read, byte value.

FUNCTIONING:

This function is designed to generate a multi-purpose clock and strobe. FB107 requires an instance data block in order to store counters. The instance FB will be created the first time you call the FB107 in your source program. For example:

```
CALL FC     3                          //    Slow down scan by 2ms
     Msec: =W#16#2

CALL FB 107, DB107
     IN_ClockBits:=B#16#10              //    panel bits: 10h=16=2 panels
     IN_SlowDownExecution:= W#16#3      //    Slow down tree times
     OUT_CLOCK:=Q0.2                    //    Clock to be sent to the panel
     OUT_STROBE:=Q0.1                   //    Strobe to be sent to the panel
     OUT_DATA_PULSE:=M10.2              //    FB50->FC50: read the bit
     OUT_BIT_NUMBER_READY:=MB12         //    FB50->FC50: bit code read

L    QB     0
T    PQB    0                          //    MUST: update clock output here

L    PIB    4                          //    MUST: read input directly
T    MB     80


CALL FC    107
     IN_Panel_FeedBack:=M80.1           //    the state of the data input
     IN_Read_Data_Pulse:=M10.2          //    FB50->FC50: read the bit
     IN_Bit_Number_Ready:=MB12          //    FB50->FC50: bit code read
     OUT_PANEL_IMAGE:=MB15              //    Data read is written from here
```

In this example the CLOCK output is Q0.2, the STROBE output is Q0.1 and the DATA input is I4.1.
This example reads 16 bits from TWO cascaded alarm panels 5.625.1 and stores the read data in MB15 and MB16.
This is the hardware configuration:
- The DATA OUTPUT of the first panel is connected to the DATA INPUT of the second panel.
- The DATA OUTPUT of the second panel is connected to the PLC INPUT for reading.
- The CLOCK and STROBE output generated by the plc are connected to the CLOCK and STROBE inputs of the panels (parallel connection).

If you only have one panel, you just connect the "DATA OUTPUT" of your panel to the "PLC INPUT".

DEPENDECIES:

NOTES:

- HARDWARE SETTINGS:
  Panels' jumpers must be set to ACCEPT EXTERNAL STROBE. In particular, on the 5.625.1 the jumper SJ13 will be set to "EXT". SJ11 and SJ12 can remain open.
- NOTES ON THE ADDRESS OF THE READ DATA:
  The OUT_PANEL_IMAGE of the FCxx will specify the STARTING ADDRESS of the first byte being read from the panel. Since OUT_PANEL_IMAGE is declared as a pointer you can write any address to it:
  For example: M15.0, MB15, MW15, MD15 will produce the same result: The first panel data will be in MB15, the second on MB16 and so on.

  For example: DBX 15.0, DBB15, DBW15, DBD15 will produce the same result: The first panel data will be in DBB15, the second on DBB16 and so on.

- YOU CAN ASSIGN ANYTHING TO THE "OUT_PANEL_IMAGE". What counts is only its address. It is also possible to specify a bit offset, for example M15.1.
- IMPORTANT: the FC107 will overwrite bit addresses starting from the "OUT_PANEL_IMAGE" to the bit with the offset "IN_Bit_Number_Ready".
  This means that if you are using an 8 bit clock the bits overwritten will belong to the firt byte of OUT_PANEL_IMAGE, if your clock is 16 bit long the bits overwritten will belong to the first two bites starting from OUT_PANEL_IMAGE.
- TROUBLESHOOTING, FAQs:

  Q: CANNOT READ DATA FROM PANELS.
  A: Check connections to the panel and jumpers: panel must be set for external strobe. For example on board 5.625.1, jumper SJ13 must be set to "EXT".

  Q: SOME DATA IS READ BUT THE BIT ORDER SEEMS WRONG.
  A: Please be sure that: clock FB is called before data reading, clock output periphery is updated just after the FB call, input are read directly from periphery before the call to the reading block (FCxx). Please refer to the above programming example.

  Q: I HAVE TWO+ PANELS IN CASCADE BUT ONLY ONE WORKS.
  A: Please ensure that you are generating the appropriate clock. The clock setting IN_ClockBits must be:
  W#16#8 for one panel,
  W#16#10 for two panels,
  W#16#18 for three panels,
  W#16#20 for four panels (maximum).

  Q: I HAVE TWO+ PANEL INCASCADE BUT I WANT TO READ DATA ONLY FROM ONE.
  A: Please disconnect the STROBE input from the panel you don't want to read data from.

  Q: SOME DATA IS READ FROM THE PANEL BUT BITS ARE "UNSTABLE"
  A: Your plc scan time might be shorter that the minimum time necessary for a digital input to read a pulse. Remember that with the input modula421 the scan time must at least 5ms. You can use the parameter "SlowDownExecution" to slow down the clock creation and data read. For instance a value of 2 on this field is the same to call the function every 3 scans (2 are skipped).

Q: THE STROBE SIGNAL IS NOT CONNECTED TO THE PANEL: WHAT CAN I DO?
A: Connect it! (Highly recommended). If this is not possible it is necessary to program all the block calls in a 10 or 20ms time interrupt. Be sure to set up the panel for a 10 or 20 ms clock and modify the following internal variables in FB107

#WaitStrobe=1;
#WaitClock=4;
#StrobeLength=4 (for 8 bit clock).

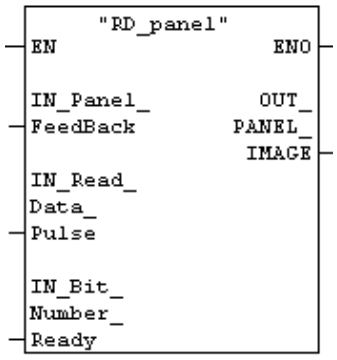
As you can see this FB has not been programmed for this kind of behaviour.

READING “CERCAGUASTI” PANEL (“RD_panel”) (FC107)

SHORT DESCRIPTION: Reading “CercaGuasti” panel with tabs 5.543.2…

IN		
Name	**Type**	**Description**
IN_Panel_FeedBack	Bool	
IN_Read_Data_Pulse	Bool	
IN_Bit_Number_Ready	Byte	

OUT		
Name	**Type**	**Description**
OUT_PANEL_IMAGE	Pointer	



PARAMETERS DESCRIPTION:

IN_Panel_FeedBack: Input that allows direct bit addressing on the CPU memory, boolean value.
IN_Read_Data_Pulse: Input that specify we are reading data, boolean value.
IN_Bit_Number_Ready: Offset value, byte value.
OUT_PANEL_IMAGE: Pointer to the data that contains the Pointer to the address we want, pointer value.

FUNCTIONING:

FC107 requires only some data generated by FB107 (the bit reading command and the bit code) and the signal from a digital input containing the data from the panel.
This FC will assign the bit at the address "OUT_PANEL_IMAGE +IN_Bit_Number_Ready" with the status of "IN_Panel_FeedBack". This action will be performed when the “IN_Read_Data_Pulse” is ON.
What counts in "OUT_PANEL_IMAGE" is the address only.
You can write any value there (M15.0, MB15, MW15, MD15, DB10.DBW5) and get the same result.

DEPENDENCIES:

NOTES:

Incorrect assignment to the input "IN_Number_Ready" could lead to plc stops or other undesired results.
This function will modify just one bit at the specified address.

SHORT DESCRIPTION: Management of the sequencer.

IN

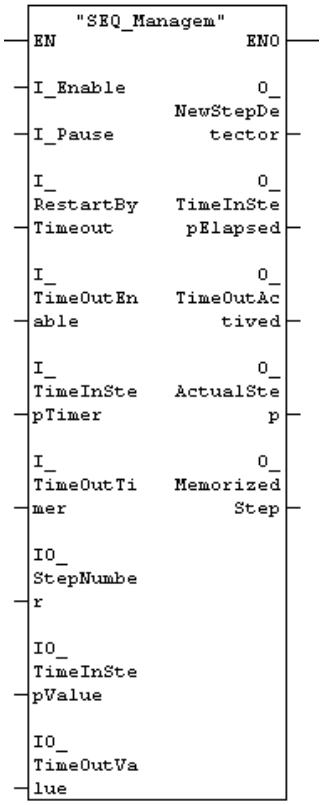
Name	Type	Description
I_Enable	Bool	Signal that enables sequencer to start from step 0
I_Pause	Bool	Signal that stops the sequencer in pause at step 2
I_RestartByTimeout	Bool	Signal that restarts the sequencer from step 1 at memorized step
I_TimeOutEnable	Bool	Input to enable 'timeout in step' function
I_TimeInStepTimer	Timer	Number of timer used for 'time in step' function
I_TimeOutTimer	Timer	Number of timer used for 'timeout' function

OUT

Name	Type	Description
O_NewStepDetector	Bool	Signal TRUE when step is changed
O_TimeInStepElapsed	Bool	Signal TRUE when time in step is elapsed
O_TimeOutActived	Bool	Signal TRUE when timeout time is elapsed
O_ActualStep	Int	Actual step (for debug only)
O_MemorizedStep	Int	Last step memorized (for debug only)

IN-OUT

Name	Type	Description
IO_StepNumber	Int	Step number
IO_TimeInStepValue	Word	Time value for 'time in step' function
IO_TimeOutValue	Word	Time value for 'timeout' function



PARAMETERS DESCRIPTION:

I_Enable: Signal that enables sequencer to start from step 0, if this input is zero, the sequencer is forced at step 0 and no other inputs are evaluated, Boolean value.
I_Pause: Signal that stops the sequencer in pause at step 2, if this input is on and the sequencer is at step different from step 0 or step 1, sequencer will save the last step and will transit to step 2 (PAUSE). If while in step 2 this input bit turns off, sequencer will restart from last memorized step. If while in step 2 I_Enable goes 0, sequencer goes to step 0 & the hold function is lost, Boolean value.
I_RestartByTimeout: Signal that restarts the sequencer from step 1 at memorized step, if sequencer is at step 1 (TIMEOUT) it can restart from to last memorized step with one Rise Edge on this input bit. Otherwise the only solution to exit by timeout is to reset sequencer to step 0 with "I_Enable" == 0, Boolean value.
I_TimeOutEnable: Input to enable "timeout in step" function, Boolean value.
I_TimeInStepTimer: Number of timer used for "time in step" function, timer value.
I_TimeOutTimer: Number of timer used for "timeout" function, timer value.

O_NewStepDetector: Signal TRUE when step is changed, Boolean value.
O_TimeInStepElapsed: Signal TRUE when time in step is elapsed, Boolean value.
O_TimeOutActived: Signal TRUE when timeout time is elapsed, Boolean value.
O_ActualStep: Actual step (for debug only), integer value.
O_MemorizedStep: Last step memorized (for debug only), integer value.

IO_StepNumber: Step number, integer value.
IO_TimeInStepValue: Time value for "time in step" function, word value.
IO_TimeOutValue: Time value for "timeout" function, word value.

FUNCTIONING:

This function is designed to manage all functions of the sequencer, following the commands by input bits.
The first three steps are reserved for special functions and the user should not force transition to these steps writing the corrispondent number in the IO_StepNumber like for other steps. He should use the relative enabling bits.

Step 0 = step restart
Step 1 = timeout step
Step 2 = pause step

TIMEOUT FUNCTIONING:
if sequencer stays in step different by step 0, 1 and 2 with the input "I_TimeOutEnable"== 1 for more than time set in "I_TimeOutValue", the output of timer will turn on and this will be signalled at external by "O_TimeOutActived" (this output is the direct output of the timer therefore is only one pulse before to transit in step 1).
So at next scan the function automatically will transit to step 1 and will save the last step (for possibile restart by last step).

TIME IN STEP FUNCTIONING:
if sequencer stays in step with the input "I_TimeInStepEnable"== 1 for more than time set in "I_TimeInStepValue", the output of timer will turn on and this will be signalled at external by "O_TimeInStepElapsed" (this output is the direct output of the timer).

User can read and use "O_TimeInStepElapsed" in the sequencer or everywhere he needs to know when the sequencer has been stopped at one step for more than the set time.

DEPENDECIES:

NOTES:

COPY THE “STEP-TIMEOUT-TIME” IN STEP FOR SEQUENCE (“SEQ_Aux”) (FC105)

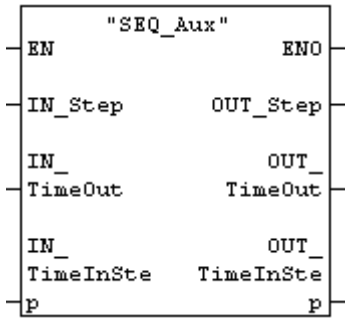
SHORT DESCRIPTION: Copy the “step-timeout-time”in step for sequencers.

IN

Name	Type	Description
IN_Step	Int	number of step to copy
IN_TimeOut	S5time	timeout value to copy
IN_TimeInStep	S5time	time in step value to copy

OUT

Name	Type	Description
OUT_Step	Int	number of step destination to copy
OUT_TimeOut	Word	timeout value destination to copy
OUT_TimeInStep	Word	time in step value destination to copy



PARAMETERS DESCRIPTION:

IN_Step: Number of step copy, integer value.
IN_TimeOut: Timeout value to copy, double integer value. [s*100]
IN_TimeInStep: Time in step value to copy, double integer. [s*100]

OUT_Step: Number of step destination to copy, integer value.
OUT_TimeOut: Timeout value destination to copy, double integer value.
OUT_TimeInStep: Time in step value destination t copy, double integer value.

FUNCTIONING:

This function is designed to transfer step + timeout + time_in_step for sequencer.

DEPENDECIES:

NOTES:

This function is only used for better visualization in ladder style.

DECIMAL FLAG FOR SEQUENCER ("SEQ_Deco") (FB106)

SHORT DESCRIPTION: Decimal flag for sequencer.

IN		
Name	**Type**	**Description**
Step_Number	Int	Step number input choiced

PARAMETERS DESCRIPTION:

Step_Number: Step number input choiced, integer value.

FUNCTIONING:

This function is designed to performs decodify for sequencer, this is the S7 version of old type function DEC-> FLGY. This function only decodify by integer number to single bit step. The bit step are comprhensive in the istance area of this function.

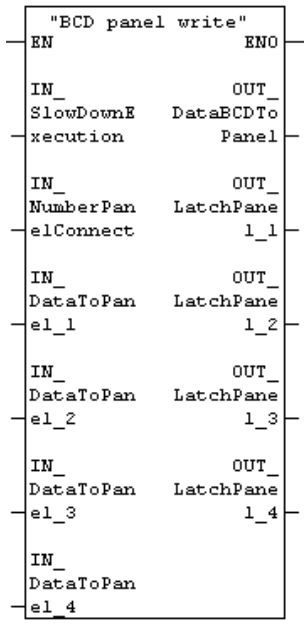
DEPENDECIES:

NOTES:

SHORT DESCRIPTION: Serial-led panel.

IN		
Name	**Type**	**Description**
IN_SlowDownExecution	Int	number cycle to execute this function
IN_NumberPanelConnect	Int	number panel from 1 to 4 that is connected
IN_DataToPanel_1	DWord	data to panel 1 by to converter
IN_DataToPanel_2	DWord	data to panel 2 by to converter
IN_DataToPanel_3	DWord	data to panel 3 by to converter
IN_DataToPanel_4	DWord	data to panel 4 by to converter

OUT		
Name	**Type**	**Description**
OUT_DataBCDToPanel	DWord	data output to panel
OUT_LatchPanel_1	Bool	latch command for first panel
OUT_LatchPanel_2	Bool	latch command for second panel
OUT_LatchPanel_3	Bool	latch command for third panel
OUT_LatchPanel_4	Bool	latch command for fourth panel



PARAMETERS DESCRIPTION:

IN_SlowDownExecution: Number cycle to execute this function, integer value.
IN_NumberPanelConnect: Number panel from 1 to 4 that is connected, integer value.
IN_DataToPanel_1: Data to panel 1 by to converter, double word value.
IN_DataToPanel_2: Data to panel 2 by to converter, double word value.
IN_DataToPanel_3: Data to panel 3 by to converter, double word value.
IN_DataToPanel_4: Data to panel 4 by to converter, double word value.

OUT_DataBCDToPanel: Data output to panel, double word value.
OUT_LatchPanel_1: Latch command for first panel, boolean value.
OUT_LatchPanel_2: Latch command for second panel, boolean value.
OUT_LatchPanel_3: Latch command for third panel, boolean value.
OUT_LatchPanel_4: Latch command for fourth panel, boolean value.

FUNCTIONING:

This function is designed to perform the management of BCD panel.

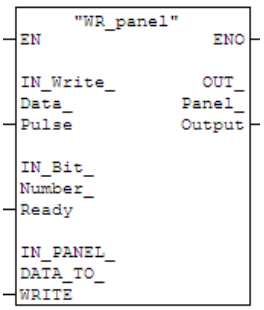
DEPENDENCIES:

NOTES:

SHORT DESCRIPTION: Writing Serial-led panel.

IN		
Name	**Type**	**Description**
IN_Write_Data_Pulse	Bool	data bit ready to transfer
IN_Bit_Number_Ready	Byte	number bit ready for transfer
IN_PANEL_DATA_TO_WRITE	Pointer	pointer at data source that we must to transfer to panel

OUT		
Name	**Type**	**Description**
OUT_Panel_Output	Bool	bit that will be sent to panel like serial data input



PARAMETERS DESCRIPTION:

IN_Write_Data_Pulse: Data bit ready to transfer, boolean value.
IN_Bit_Number_Ready: Number of bit ready for transfer, byte value.
IN_PANEL_DATA_TO_WRITE: Pointer of data source that we must to transfer to panel, pointer value.

OUT_Panel_Output: Bit that will be sent to panel like serial data input, boolean value.

FUNCTIONING:

This function is designed to assign the bit readed at the address "IN_PANEL_DATA_TO_WRITE + IN_bit_Number_Ready" to the output "OUT_Panel_Output". This action is performed when the “IN_Write_Data_Pulse” is ON. What counts in "IN_PANEL_DATA_TO_WRITE" is the address only. You can write any value there (M15.0, MB15, MW15, MD15, DB10.DBW5) and get the same result.

DEPENDECIES:

NOTES:

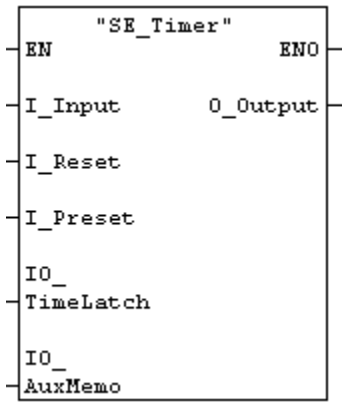
For a complete usage overview, please see documentation for FB107.
Incorrect assignment to the input “IN_Bit_Number_Ready” could lead to plc stops or other undesidered results.

SHORT DESCRIPTION: Extended pulse timer.

IN		
Name	**Type**	**Description**
I_Input	Bool	
I_Reset	Bool	
I_Preset	Dint	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT		
Name	**Type**	**Description**
O_Output	Bool	

IN-OUT		
Name	**Type**	**Description**
IO_TimeLatch	Dint	
IO_AuxMemo	Byte	



PARAMETERS DESCRIPTION:

I_Input: Input signal, boolean value.
I_Reset: Enable or disable the timer reset, boolean value.
I_Preset: Preset value, range is from 0 to 2147483647 (more than 5965 hours). [s/100]

O_Output: Timer output, boolean value.

IO_TimeLatch: Auxiliary variable for internal use, double integer value.
IO_AuxMemo: Auxiliary variable for internal use, byte value.

FUNCTIONING:

If "I_Input" is setting by a rising edge the output will be set for a period equal to the value entered in "I_Preset" and then return false. If you enter another positive edge in "I_Input" while "O_Output" is still high, the output will remain high since the last rising edge for a period equal to that shown in "I_Preset".

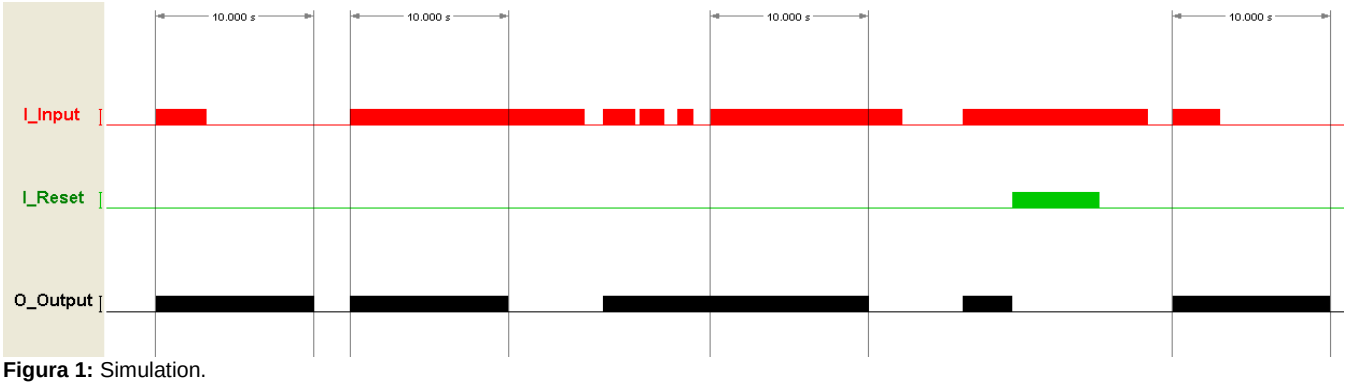


Figura 1: Simulation.

DEPENDENCIES:

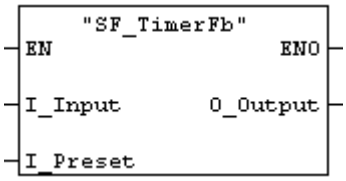
To get this function working, FC99 must be called in 10 ms interupt cycle. MD108 is defined as DINT and used as time counter in cents of s.

NOTES:
To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

SHORT DESCRIPTION: Off delay timer.

IN		
Name	**Type**	**Description**
I_Input	Bool	
I_Preset	DInt	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT		
Name	**Type**	**Description**
IO_Output	Bool	



PARAMETERS DESCRIPTION :

I_Input: input signal, boolean value.
I_Preset: Preset value, range is from 0 to 2147483647 (more than 5965 hours). [s/100]

O_Output: timer output, boolean value.

FUNCTIONING:

O_Output goes false after the I_Preset delay if I_Input is false for a time longer than I_Preset. As soon as I_Input goes true also O_Output goes true. If while counting I_Input goes true and then false again, counting restart from the beginning.

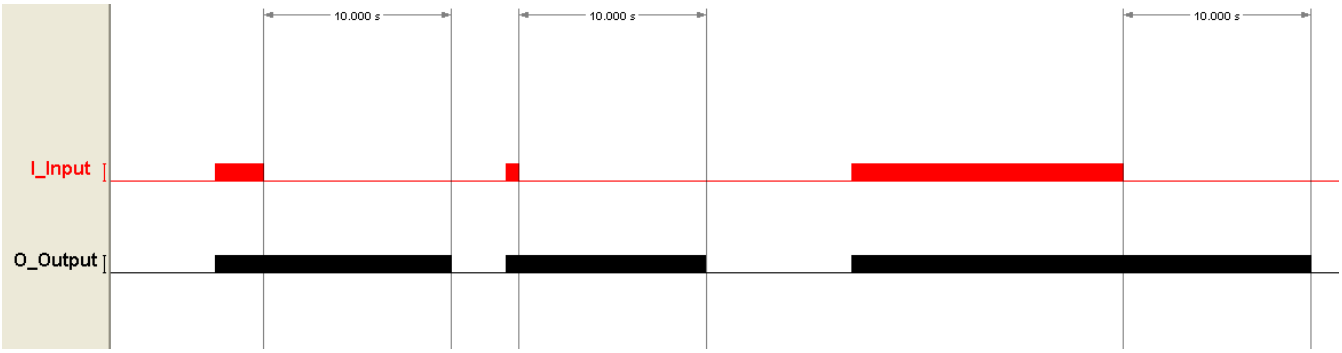


Figure 1: Simulation.

DEPENDENCIES:

To get this function working, FC99 must be called in 10 ms interupt cycle. MD108 is defined as DINT and used as time counter in cents of s.

NOTES:

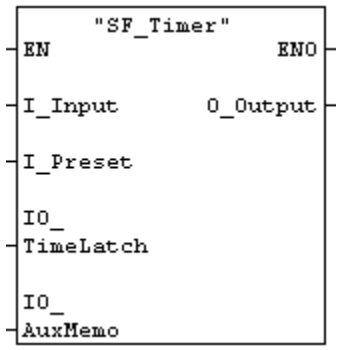
To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

SHORT DESCRIPTION: Off delay timer.

IN		
Name	**Type**	**Description**
I_Input	Bool	
I_Preset	Dint	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT		
Name	**Type**	**Description**
O_Output	Bool	

IN-OUT		
Name	**Type**	**Description**
IO_TimeLatch	Dint	
IO_AuxMemo	Byte	



PARAMETERS DESCRIPTION:

I_Input: Input signal, boolean value.
I_Preset: Preset value, range is from 0 to 2147483647 (more than 5965 hours). [s/100]

O_Output: Timer output, boolean value.

IO_TimeLatch: Auxiliary variable for internal use, double integer value.
IO_AuxMemo: Auxiliary variable for internal use, byte value.

FUNCTIONING:

O_Output goes false after the I_Preset delay if I_Input is false for a time longer than I_Preset. As soon as I_Input goes true also O_Output goes true. If while counting I_Input goes true and then false again, counting restart from the beginning.

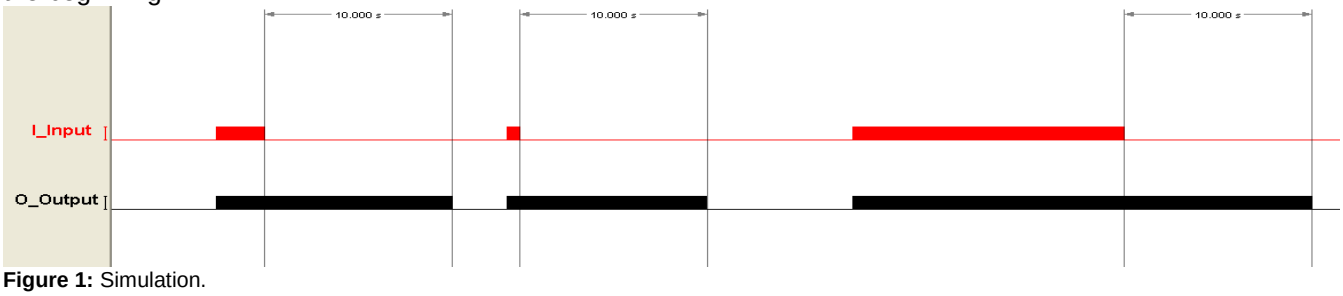


Figure 1: Simulation.

DEPENDENCIES:

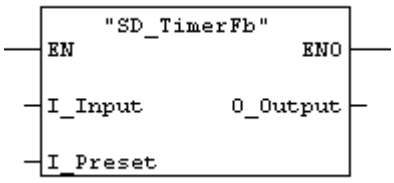
To get this function working, FC99 must be called in 10 ms interupt cycle. MD108 is defined as DINT and used as time counter in cents of s.

NOTES:
To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

SHORT DESCRIPTION: On delay timer.

IN		
Name	**Type**	**Description**
I_Input	Bool	
I_Preset	DInt	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT		
Name	**Type**	**Description**
O_Output	Bool	



PARAMETERS DESCRIPTION :

I_Input: Input signal, boolean value.
I_Preset: Preset value, range is from 0 to 2147483647 (more than 5965 hours). [s/100]

O_Output: Timer output, boolean value.

FUNCTIONING:

O_Output goes true after the I_Preset delay if I_Input is true for a time longer than I_Preset. As soon as I_Input goes false also O_Output goes false. If while counting I_Input goes false and then true again, counting restart from the beginning.

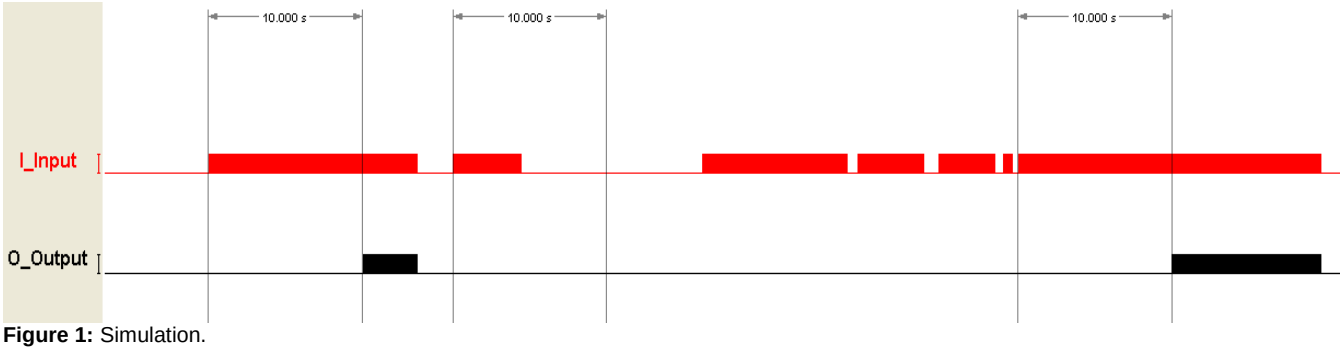


Figure 1: Simulation.

DEPENDENCIES:

To get this function working, FC99 must be called in 10 ms interupt cycle. MD108 is defined as DINT and used as time counter in cents of s.

NOTES:

To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

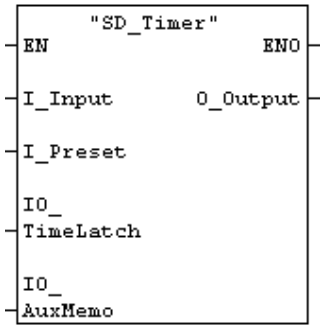
ON-DELAY TIMER ("SD_Timer") (FC100)

SHORT DESCRIPTION: On delay timer.

IN		
Name	**Type**	**Description**
I_Input	Bool	
I_Preset	Dint	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT		
Name	**Type**	**Description**
O_Output	Bool	

IN-OUT		
Name	**Type**	**Description**
IO_TimeLatch	Dint	
IO_AuxMemo	Byte	



PARAMETERS DESCRIPTION :

I_Input: Input signal, boolean value.
I_Preset: Preset value, range is from 0 to 2147483647 (more than 5965 hours). [s/100]

O_Output: Timer output, boolean value.

IO_TimeLatch: Auxiliary variable for internal use, double integer value.
IO_AuxMemo: Auxiliary variable for internal use, byte value.

FUNCTIONING:

O_Output goes true after the I_Preset delay if I_Input is true for a time longer than I_Preset. As soon as I_Input goes false also O_Output goes false. If while counting I_Input goes false and then true again, counting restart from the beginning.

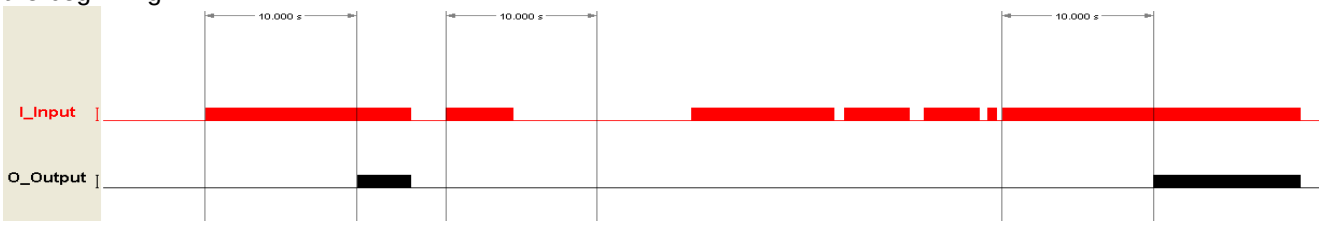


Figure 1: Simulation.

DEPENDENCIES:

To get this function working, FC99 must be called in 10 ms interupt cycle. MD108 is defined as DINT and used as time counter in cents of s.

NOTES:

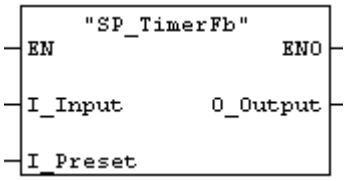
To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

PULSE TIMER ("SP_TimerFb") (FB13)

SHORT DESCRIPTION: Pulse timer.

IN		
Name	**Type**	**Description**
I_Input	Bool	
I_Preset	DInt	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT		
Name	**Type**	**Description**
O_Output	Bool	



PARAMETERS DESCRIPTION:

I_Input: input signal, boolean value.
I_Preset: preset value, range from 0 to 2147483647 (more than 5965 hours). [s/100]

O_Output: Timer output, boolean value.

FUNCTIONING:

If "I_Input" is setting by a rising edge timer starts. "O_Output" follows the input until the timer does not exceed the value of "I_Preset". When it exceeds, output goes to 0 instantly.

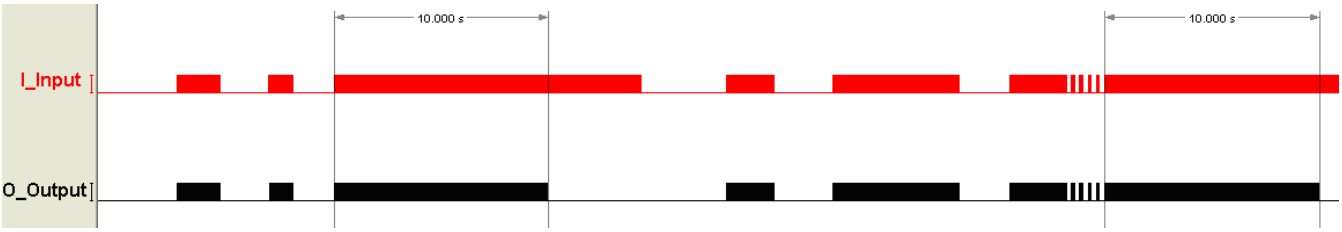


Figure 1: Simulation.

DEPENDECIES:

NOTES:

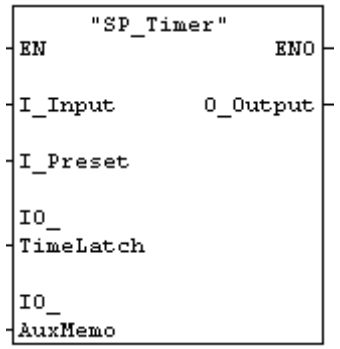
To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

SHORT DESCRIPTION: Pulse timer.

IN		
Name	**Type**	**Description**
I_Input	Bool	
I_Preset	Dint	s/100, range 0 to 2147483647 (more than 5965 hours)

OUT		
Name	**Type**	**Description**
O_Output	Bool	

IN-OUT		
Name	**Type**	**Description**
IO_TimeLatch	Dint	
IO_AuxMemo	Byte	



PARAMETERS DESCRIPTION:

I_Input: Input signal, boolean value.
I_Preset: preset value, range from 0 to 2147483647 (more than 5965 hours). [s/100]

O_Output: Timer output, boolean value.

IO_TimeLatch: Time value of rising edge at the "I_Input", double integer value.
IO_AuxMemo: Auxiliary variable for internal use, byte value.

FUNCTIONING:

If "I_Input" is setting by a rising edge timer starts. "O_Output" follows the input until the timer does not exceed the value of "I_Preset". When it exceeds, output goes to 0 instantly.

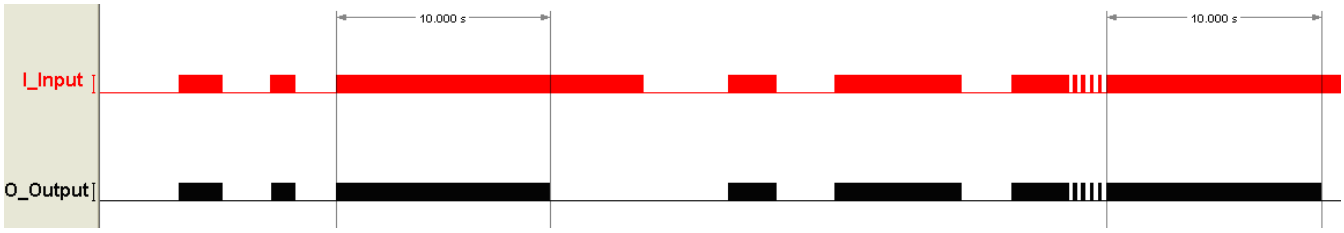


Figure 1: Simulation.

DEPENDECIES:

NOTES:

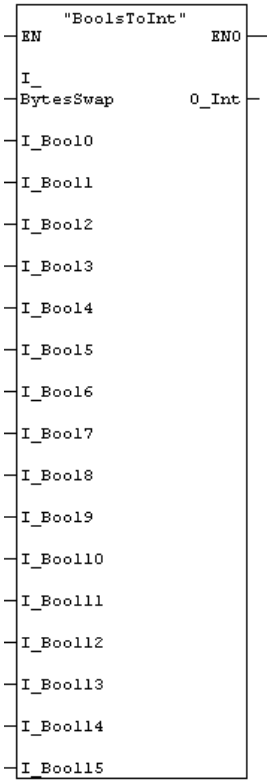
To get this function working, FC99 must be call in 10 ms cycle (OB35). MD108 is defined as DINT and used as time counter in s/100.

COPY 16 BOOLS INTO AN INT ("BoolsToInt") (FC8)

SHORT DESCRIPTION: Copy of 16 bools into an INT with bytes swapping possibility.

IN		
Name	**Type**	**Description**
I_BytesSwap	Bool	
I_Bool0	Bool	
I_Bool1	Bool	
I_Bool2	Bool	
I_Bool3	Bool	
I_Boo4	Bool	
I_Bool5	Bool	
I_Bool6	Bool	
I_Bool7	Bool	
I_Bool8	Bool	
I_Bool9	Bool	
I_Bool10	Bool	
I_Bool11	Bool	
I_Bool12	Bool	
I_Bool13	Bool	
I_Bool14	Bool	
I_Bool15	Bool	

OUT		
Name	**Type**	**Description**
O_Int	Int	int



PARAMETERS DESCRIPTION:

I_BytesSwap: swap the least significant byte with most significant byte, boolean value.
I_Bool0: state for bit 0 of destination integer if I_BytesSwap = false; state for bit 8 of destination integer if I_BytesSwap = true, boolean value.
I_Bool1: state for bit 1 of destination integer if I_BytesSwap = false; state for bit 9 of destination integer if I_BytesSwap = true, boolean value.
…
I_Bool8: state for bit 8 of destination integer if I_BytesSwap = false; state for bit 0 of destination integer if I_BytesSwap = true, boolean value.
…
I_Bool15: state for bit 15 of destination integer if I_BytesSwap = false; state for bit 7 of destination integer if I_BytesSwap = true, boolean value.

O_Int: integer where to copy the states of 16 bools, integer value.

FUNCTIONING:

This function fills bits state of an integer variable.

DEPENDENCIES:

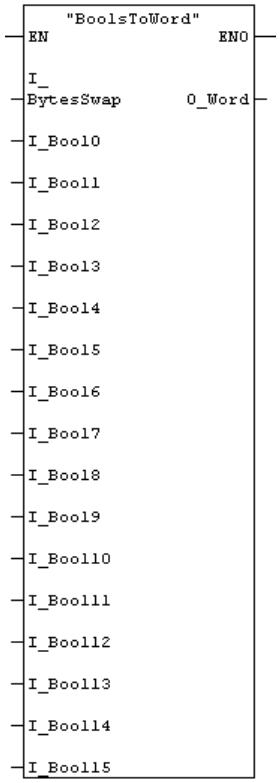
NOTES:

COPY 16 BOOLS INTO A WORD ("BoolsToWord") (FC10)

SHORT DESCRIPTION: Copy of 16 bools into a WORD with bytes swapping possibility.

IN		
Name	**Type**	**Description**
I_BytesSwap	Bool	
I_Bool0	Bool	
I_Bool1	Bool	
I_Bool2	Bool	
I_Bool3	Bool	
I_Boo4	Bool	
I_Bool5	Bool	
I_Bool6	Bool	
I_Bool7	Bool	
I_Bool8	Bool	
I_Bool9	Bool	
I_Bool10	Bool	
I_Bool11	Bool	
I_Bool12	Bool	
I_Bool13	Bool	
I_Bool14	Bool	
I_Bool15	Bool	

OUT		
Name	**Type**	**Description**
O_Word	Word	Word



PARAMETERS DESCRIPTION:

I_BytesSwap: swap the least significant byte with most significant byte, boolean value.
I_Bool0: state for bit 0 of destination integer if I_BytesSwap = false; state for bit 8 of destination integer if I_BytesSwap = true, boolean value.
I_Bool1: state for bit 1 of destination integer if I_BytesSwap = false; state for bit 9 of destination integer if I_BytesSwap = true, boolean value.
…
I_Bool8: state for bit 8 of destination integer if I_BytesSwap = false; state for bit 0 of destination integer if I_BytesSwap = true, boolean value.
…
I_Bool15: state for bit 15 of destination integer if I_BytesSwap = false; state for bit 7 of destination integer if I_BytesSwap = true, boolean value.

O_Word: word where to copy the states of 16 bools, word value.

FUNCTIONING :

This function fills bits state of a word variable.

DEPENDENCIES:

NOTES:

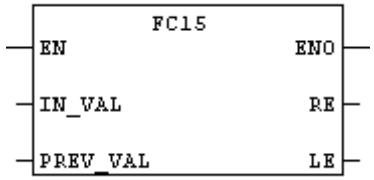
DETECTION OF RISING EDGE AND LOWERING EDGE INSIDE A WORD (FC15)

SHORT DESCRIPTION: Detection of rising edge and lowering edge inside a word.

IN		
Name	**Type**	**Description**
IN_VAL	Word	Scanned input word

OUT		
Name	**Type**	**Description**
RE	Bool	Cumulative rising edge
LE	Bool	Cumulative lowering edge

IN-OUT		
Name	**Type**	**Description**
PREV_VAL	Word	Saved status of the input word



PARAMETERS DESCRIPTION:

IN_VAL: the scanned input word, word value.

PREV_VAL: auxiliary variable containing the saved status of the input word; used internally, word value.

RE: cumulative rising edge, true for one scan when at least one bit rising edge detected inside input word, boolean value.
LE: cumulative lowering edge, true for one scan when at least one bit lowering edge detected inside input word, boolean value.

FUNCTIONING:

This function generates two cumulative “one shot” each time one bit, in the IN_VAL, changes state.

DEPENDENCIES:

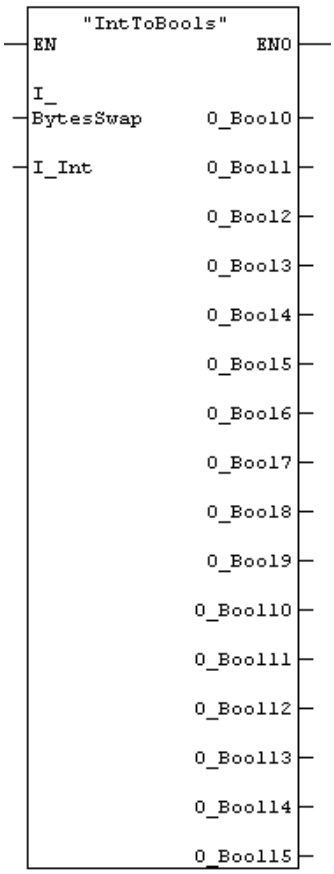
NOTES:

COPY INT INTO 16 BOOLS ("IntToBools") (FC9)

SHORT DESCRIPTION: Copy of INT into 16 bools with bytes swapping possibility.

IN		
Name	**Type**	**Description**
I_BytesSwap	Bool	
I_Int	Int	int

OUT		
Name	**Type**	**Description**
O_BytesSwap	Bool	
O_Bool0	Bool	
O_Bool1	Bool	
O_Bool2	Bool	
O_Bool3	Bool	
O_Boo4	Bool	
O_Bool5	Bool	
O_Bool6	Bool	
O_Bool7	Bool	
O_Bool8	Bool	
O_Bool9	Bool	
O_Bool10	Bool	
O_Bool11	Bool	
O_Bool12	Bool	
O_Bool13	Bool	
O_Bool14	Bool	
O_Bool15	Bool	



PARAMETERS DESCRIPTION:

I_BytesSwap: swap the least significant byte with most significant byte, boolean value.
I_Int: integer from where to get the states of 16 bools, integer value.
O_Bool0: state of bit 0 if I_BytesSwap = false; state of bit 8 if I_BytesSwap = true, boolean value.
O_Bool1: state of bit 1 if I_BytesSwap = false; state of bit 9 if I_BytesSwap = true, boolean value.
…
O_Bool8: state of bit 8 if I_BytesSwap = false; state of bit 0 if I_BytesSwap = true, boolean value.
…
O_Bool15: state of bit 15 if I_BytesSwap = false; state of bit 7 if I_BytesSwap = true, boolean value.

FUNCTIONING:

This function gets singular bit state from integer variable.

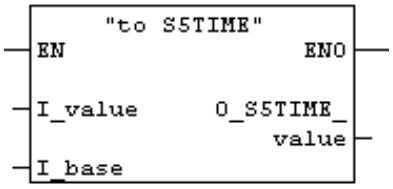
DEPENDECIES:

NOTES:

SHORT DESCRIPTION: Conversion of “timebase + multiplier” to S5TIME format.

IN		
Name	**Type**	**Description**
I_value	Int	Time
I_base	Int	time base (0=0.01s, 1=0.1s, 2=1s, 3=10s)

OUT		
Name	**Type**	**Description**
O_S5TIME_value	S5time	S5TIME format



PARAMETERS DESCRIPTION:

I_Value: time value, from 0 to 999, integer value.
I_base: time base selection (0 = 0.01s, 1 = 0.1s, 2 = 1s, 3= 10s), integer value.
O_S5TIME_value: S5TIME format value.

FUNCTIONING:

This function is used to convert to S5TIME format a time value expressed as time & multiplier.

DEPENDENCIES:

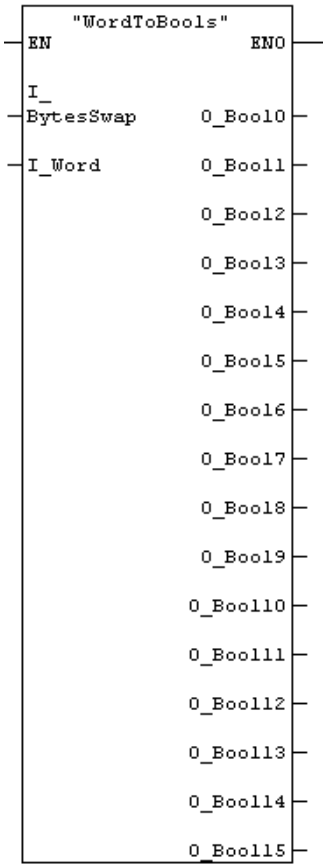
NOTES:

COPY WORD INTO 16 BOOLS ("WordToBools") (FC11)

SHORT DESCRIPTION: Copy of WORD into 16 bools with bytes swapping possibility.

IN		
Name	**Type**	**Description**
I_BytesSwap	Bool	
I_Word	Word	Word

OUT		
Name	**Type**	**Description**
O_Bool0	Bool	
O_Bool1	Bool	
O_Bool2	Bool	
O_Bool3	Bool	
O_Boo4	Bool	
O_Bool5	Bool	
O_Bool6	Bool	
O_Bool7	Bool	
O_Bool8	Bool	
O_Bool9	Bool	
O_Bool10	Bool	
O_Bool11	Bool	
O_Bool12	Bool	
O_Bool13	Bool	
O_Bool14	Bool	
O_Bool15	Bool	



PARAMETERS DESCRIPTION:

I_BytesSwap: swap the least significant byte with most significant byte, boolean value.
I_Word: word from where to get the states of 16 bools, word value.
O_Bool0: state of bit 0 if I_BytesSwap = false; state of bit 8 if I_BytesSwap = true, boolean value.
O_Bool1: state of bit 1 if I_BytesSwap = false; state of bit 9 if I_BytesSwap = true, boolean value.
…
O_Bool8: state of bit 8 if I_BytesSwap = false; state of bit 0 if I_BytesSwap = true, boolean value.
…
O_Bool15: state of bit 15 if I_BytesSwap = false; state of bit 7 if I_BytesSwap = true, boolean value.

FUNCTIONING:

This function gets singular bit state from word variable.

DEPENDECIES:

NOTES: