Modulprogrammerat bansystem med HMI och metadatainsamling

Av:
Tobias Nilsson
Oscar Jalgén

17 augusti 2018

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Division of Industrial Electrical Engineering and Automation

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2018

Nyckelord

TIA Portal
Modul programmering
Programmable Logic Controller - PLC
Metadata
User Defined Datatype - UDT
Human Machine Interface - HMI
Faceplate

Sammanfattning

Examensarbetet handlar om att skapa ett PLC-program som hanterar ett bansystem samt skapa en HMI-panel för visualisering av systemet och processen. Examensarbetet har utförts i Totally Integrated Automation Portal (TIA Portal) som är en plattform från Siemens där ett antal olika mjukvaruprogram integreras. Ett PLC-program har skapats i mjukvaruprogrammet STEP 7 och en HMI-panel har konstruerats i mjukvaruprogrammet WinCC. Allting utförs i en virtuell miljö men programmet är fortfarande kompatibelt med fysisk hårdvara. Om programmet önskas köras med fysisk hårdvara så är det endast att koppla upp programmet mot systemet och sätta upp de nödvändiga kopplingarna.

Processen fungerar som så att en truck lämnar en pall på inmatningsstationen. Från det att pallen har lämnats så sköts pallförflyttning automatiskt, tills den kommer till en utmatningsstation där pallen hämtas med en truck. Däremellan passerar pallen olika typer av sektioner som ska simulera ett verklighetstroget system som kan finnas i en fabrik. Eftersom processen skall vara helt automatisk så används fotoceller som givare för att ha kännedom om det befinner sig en pall på sektionen eller inte. Fotocellerna används också som grund till kommunikationen mellan bansektionerna. Varje sektion kan endast ha en pall åt gången och varje pall har ett ID som identifikator. Systemet byggdes med en inmatningsstation och två utmatningsstationer samt en T-sektion. Med pallens ID avgör T-sektionen vilken av utmatningsstationerna den ska förflyttas till.

För ban- och motorstyrning har moduler konstruerats. En modul är ett fristående programblock som kan återanvändas genom att det anropas flera gånger, de olika modulerna och dess anrop kan interagera med varandra. Larmhantering, manuellt läge och metadatainsamling har implementerats inuti modulerna.

En stor del av examensarbetets tid lades på att samla in och presentera metadata, samt reflektioner över hur den kan användas. Metadata anses i detta fall vara statistik över systemets olika komponenter, information kring processen och pallarna. Grundtanken när examensarbetet strukturerades var att använda ett överordnat databassystem för lagring av metadata men på grund av tidsbrist användes istället de verktyg som var tillgängliga inuti

TIA portalen. Det skapades register över pallinformation och metadata samt HMI-skärmar för att presentera dem.

Resultatet blev att det skapades en simulering i realtid av processen i ett HMI samt att historisk data kunde granskas. Fotocellerna och de andra funktionerna styrs genom HMI:et då programmet inte var uppkopplat mot ett fysiskt system. HMI-skärmar skapades för en lättöverskådlig presentation av metadata och pallregister

Keywords

TIA Portal
Modular programming
Programmable Logic Controller - PLC
Metadata
User Defined Datatype - UDT
Human Machine Interface - HMI
Faceplate

Abstract

The thesis work is about creating a PLC program to handle a track system and to produce an HMI panel to visualize the system and the process. The thesis work has been carried out in the Totally Integrated Automation Portal (TIA Portal) which is a framework by Siemens where a number of different software programs are integrated. A PLC program has been created with the software program STEP 7 and an HMI panel has been produced with the software program WinCC. Everything has been done in a virtual environment but the program is still compatible to be used with physical hardware. If the program is to be run with physical hardware the only thing necessary is the connection to the real system and the components.

The process works in such a way that a truck drops off a pallet at the input station. Once the pallet has been dropped off its movement occurs automatically until it arrives at the output station where it is to be picked up by a truck. In-between the pallet passes through different sections that simulates a physical system that may occur in a factory. Because the process should be completely automatic the program uses the photocells as sensors to be aware if there is a pallet on the section or not. The photocells are also used in the communication between the sections. Every section can only handle one pallet at a time and every pallet has an ID as its identifier. The system was built with one input station, two output stations and one T-section. The T-section uses the pallet ID to determine which output station it should be sent to.

Modules has been created for the drive of a motor and the track sections. A module is an independent program block that is reusable since it can be called multiple times. The various modules and their calls can interact with eachother. Alarm management, manual drive and the collection of metadata has been implemented within the modules.

A big part of the thesis focused on how to collect and present the metadata and what use it could be of. Metadata in this thesis work is considered to be statistics about the systems different components, information concerning the process and the pallets. The main idea when the thesis work was structured was to have a superordinate database system for storage of metadata but due to time constraints it was decided to use the tools that

is available within the TIA portal. Registers for pallet information and metadata was created and HMI screens was produced to present them.

The result of the process was a simulation in real and off-line time displayed in a HMI. The photocells and the other functions were managed from the HMI screen since there is no physical hardware. The metadata and the pallet register screens created an easily foreseeable picture over statistics and status for the system and it was made easy to distinguish the important information.

Förord

Vi vill först och främst tacka Pöyry som gav oss möjligheten att utföra examensarbetet. Vi vill rikta ett stort tack till våran handledare på Pöyry, Magnus Fransson, som inte bara gav oss vägledning utan också en inblick i hur arbetslivet fungerar. Vi vill även tacka de övriga medarbetarna på Pöyry som gav oss hjälpfulla synpunkter.

Slutligen vill vi tacka vår handledare Mats Lilja och examinator Johan Björnstedt.

Innehållsförteckning

1 Inledning

I det här kapitlet ges en kortfattad beskrivning av företaget på vars uppdrag examensarbetet utförts, följt av en överblick av arbetets syfte, mål- och problemformulering samt avgränsningar och en motivering av examensarbetet.

1.1 Bakgrund

Pöyry är ett globalt konsult- och ingenjörsföretag som levererar ingenjörstjänster och strategisk rådgivning för verkställning av projekt. Pöyrys fokusområden består av transmission och distribution av högspänning, produktion av vatten- och värmekraft, kemi och bioraffinering, transport, vatten, gruv och metall samt skogsindustri. I Sverige ligger Pöyrys fokus framförallt på gruv och metallindustrin samt massa- och pappersindustrin. På Pöyrys kontor i Lund, där examensarbetet utfördes, är den största sysselsättningen konsultarbete.

Det är inom diverse industrier idag vanligt förekommande med någon form av löpande band för transport eller montagelinje. Examensarbetet kretsar kring ett bansystem för transport av europapallar. Huvudfokus ligger på hur man så smidigt och effektivt som möjligt konstruerar ett program för att sköta processen.

Det som är nytt med examensarbetet är att koppla samman ett relativt standardproblem som kan uppkomma i industrisamhället med den nya industrivärlden, Industri 4.0. Industri 4.0 är den generella termen på den nya industriella revolutionen som ska ge ett nytt perspektiv och tillvägagångssätt utav automatisering ute på fabriker. Utvecklingen och efterfrågan av Industri 4.0 har uppkommit från västvärlden för att kunna konkurrera med den låga arbetskostnaden som finns i andra delar av världen.

1.2 Syfte

Syftet med examensarbetet är att bygga kapslade program, så kallade moduler som styr bansektioner i ett bansystem. En modul skall anropas flera gånger och varje anrop styra en separat bansektion. Det innebär att en modul återanvänds för andra likadana sektioner. En modul omfattar motor, fotocell, larmhantering samt förbindelse till HMI. Även signalgränssnitt

mellan modulanrop implementeras.

Bansystemet, dess process och historik skall visualiseras i en HMI-panel. Det omfattar även pallinformation och larm. När det utförts undersöks det vilken metadata som kan samlas från processen, vad den kan användas till och hur den kan presenteras. Slutligen försöktes arbetet kopplas samman mot Industri 4.0.

Det förväntade resultatet var en lätthanterlig HMI-panel som processen styrs genom. HMI-panelen förväntades även tydligt visualisera vad som sker i anläggningen. Det omfattar allt från pallinformation till information kring motorer. Tänkbara fel som kan uppkomma förväntades att bli presenterade i HMI:et.

1.3 Målformulering

Följande mål fastställdes ihop med handledaren på Pöyry, Magnus Fransson. Det valdes att presentera dem i form av en numerisk lista för att i slutändan enkelt kunna avgöra om ett specifikt mål har uppnåtts eller inte.

1. Utföra kodningen i moduler.
2. Testa skalbarheten genom att på minimal tid expandera anläggningen. Det innebär att om moduler är gjorda korrekt ska man kunna kopiera dem och på så sätt expandera anläggningen på minimal tid.
3. Visualisera och styra bansystemet i realtid genom ett HMI-gränssnitt.
4. Larmhantering, där eventuella fel och driftstörningar ska visas i HMI:et. Integrera larmhantering i programmodul.
5. Samla in den relevanta metadatan för bansystemet. Reflektera över användningsområden för insamlad metadata.
6. Resonera kring hur man kan presentera insamlad metadata på ett tilltalande sätt, exempelvis med tabeller och diagram.
7. På ett effektivt sätt kunna lagra samt hämta intressant metadata.

1.4 Problemformulering

Sju stycken frågor som examensarbetet skulle besvara formulerades. Precis som målen har dessa frågor formulerats ihop med handledaren Magnus Fransson. Det har försökts att göra dem så kortfattade och väldefinierade som möjligt.

- Hur ska man styra ett bansystem på ett säkert och effektivt sätt för en tänkt fabrik utefter förbestämda förutsättningar?
- På vilket sätt väljer man att utföra kodningen så man på ett enkelt och smidigt sätt kan utöka anläggningen?
- Hur kan man följa processen, både i realtid och via historik?
- Hur ska man kunna upptäcka fel som håller på att ske?
- Vilken metadata kan man samla från processen?
- Vad kan denna metadata användas till?
- Hur kan man presentera metadatan på ett systematiskt och lättöverskådligt sätt?

1.5 Motivering av examensarbetet

Examensarbete valdes p.g.a. ett par anledningar. Främst för att det är högst relevant för utbildningen men även för att handledaren Magnus Fransson har varit väldigt tillmötesgående och entusiastisk angående examensarbetet. Pöyrys intresse av att erbjuda examensarbete bygger på att de vill knyta kontakter med studenter som ska ut på arbetsmarknaden. Sekundärt för att det här examensarbetet kan man relatera till ett problem som kan tänkas uppkomma i verkligheten, vilket kändes som en viktigt faktor.

1.6 Avgränsningar

Programmet är tänkt att bli en generell lösning för ett bansystem som transporterar europapallar. Programmet ska alltså fungera för en mängd olika typer av motorer och givare. Val av reell hårdvara för motor och givare kommer alltså inte att vara en del av examensarbetet.

Programmet kommer att utvecklas i Siemens TIA Portal och kommer inte vara kompatibelt med andra mjukvaruprogram som t.ex. Mitsubishi, Rockwell etc.

I Siemens Tia Portal finns möjligheten att skapa en kommunikation ner till äldre och andra versioner utav Siemens mjukvaruprogram. Detta kommer dock inte vara en del av projektet. Området som omfattar metadatan kan bli väldigt djupgående. Hur pass djupt det kommer att kunna gå inom ämnet blir i mån av tid efter det att programmet för bansystemet och tillhörande HMI är färdigt.

2 Teknisk Bakgrund

Totally Integrated Automation Portal (TIA Portal) är en plattform från Siemens där en mängd olika mjukvaruprogram integreras. Examensarbetet har genomförts i TIA Portalen V14 SP1 med STEP 7 Professional samt WinCC Professional. Delkapitlen i detta avsnitt förklarar begrepp och funktioner som använts inom STEP 7 och WinCC samt en teoretisk bakgrund till arbetsuppgiftens process.

2.1 SIMATIC STEP 7

SIMATIC STEP 7 är ett mjukvaruprogram som används för att skapa PLC-program. I det här delkapitlet beskrivs funktioner, block, datatyper samt de programspråk som använts i STEP 7.

2.1.1 Funktionsblock

Funktionsblock (FB) är ett block med kod som kräver ett instansdatablock för att lagra sina blockparametrar och statisk lokal data. På så vis är funktionsblockets parametrar och variabler tillgängliga efter blocket har exekverat, därav brukar de kallas för block med minne. Det finns även temporära variabler som inte lagras utan enbart är tillgängliga under ett cykelvarv. [7]

2.1.2 Datablock

Ett datablock (DB) är en minnesarea där data lagras. Det finns flera olika typer av datablock och i examensarbetet har två av dem använts, globala datablock samt instansdatablock.

Globala datablock kan skapas oberoende av andra block i programmet och data i blocket kan specificeras av användaren. Alla block i programmet har åtkomst till globala datablock.

Ett instansdatablock hör ihop med ett funktionsblock eller ett systemblock. Om instansdatablocket hänger ihop med ett funktionsblock så definieras datastrukturen av funktionsblockets blockgränssnitt. Hänger det ihop med ett systemblock så är datastrukturen fördefinierad. Data kan inte ändras i datablocket utan enbart genom det tillhörande funktions- eller systemblocket.

Det finns tre typer av instansdatablock varav två förekommer i examensarbetet. Ett *Single Instance* är ett separat datablock som genereras vid anrop av ett funktionsblock, datablocket är sedan förknippat med detta anrop. När ett funktionsblock anropar ytterligare ett funktionsblock måste inte ett separat datablock skapas. Istället kan ett *Multi Instance* datablock genereras. Det anropade funktionsblockets data sparas då som en lokal instans inuti instansdatablocket hos det funktionsblock som utför anropet. [2]

2.1.3 Programblock

Read System Time (RD_SYS_T) är en färdig instruktion i TIA Portalen där CPU:ns klocka avläses för aktuell datum och tid. Det returneras sedan som en variabel av datatypen DATE_AND_TIME. [2]

TONR Timer är en IEC timer funktion som kräver ett instansdatablock där timerns data lagras. En TONR timer ackumulerar tid upp till ett visst tidsvärde som bestäms av ingångsparametern PT. Tid börjar ackumuleras då en positiv signalförändring sker hos insignalen IN och fortsätter då insignalen är aktiv. Ackumulerad tid skrivs alltid till en utgång ET. När den ackumulerade tiden når det förinställda tidsvärdet på PT blir utgången Q aktiv. *TONR timern* har även en ingång R för att återställa timern. [2]

Program_Alarm är ett färdigt block i TIA Portalen för att generera larm. Blocket anropas inuti funktionsblock som en lokal instans och har en insignal som övervakas för signalförändring. När en signalförändring detekteras genereras larm. Varje larm som genereras tilldelas en tidsstämpel som hämtas från CPU:ns klocka. Användaren har möjlighet att ange upp till tio parametrar som ska förknippas med larmet och skickas med när det genereras. Exempelvis när ett larm som anger att en pall saknas genereras så skickas pallens id med som en parameter. [2] När signalförändring detekteras generas larm.

Blocket **Get_AlarmState** används ihop med *Program_Alarm*. Instansdatablocket som tillhör *Program_Alarm* anges som ingångsparameter till *Get_AlarmState* blocket. Utgången *AlarmState* anger sedan statusen på det angivna *Program_Alarm* i form av en byte. [2]

2.1.4 Datatyper

TIME är ett nummer av 32-bitar med teckenbit och representerar tid i millisekunder. Intervallet för datatypen är - 24d 20h 31m 23s 648ms till + 24d 20h 31m 23s 647ms. För att få tidsvärdet som ett heltal kan värdet hos en variabel av typen *TIME* flyttas till en *DINT*. [7]

LTIME är ett nummer av 64-bitar med teckenbit och representerar tid i nanosekunder. Intervallet för datatypen är - 106751d 23h 47m 16s 854ms 775μs 808ns till + 106751d 23h 47m 16s 854ms 775μs 807ns. För att få tidsvärdet som ett heltal kan värdet hos en variabel av typen *LTIME* konverteras till en *LINT*. [7]

DATE_AND_TIME är en 8-byte variabel som representerar en specifik tidpunkt. Representation av en tidpunkt består av *år-månad-dag-timme:minut:sekund*. [7]

User-Defined Data Type (UDT) är en komplex och justerbar datatyp vars struktur kan bestå av variabler med olika datatyper. Strukturen definieras av användaren och UDT:en kan sedan användas i hela programmet. UDT används då en struktur av variabler upprepas i olika logiska block eller som mall då flera olika datablock av samma struktur ska skapas. En UDT kan innehålla en annan UDT, det vill säga en struktur inbäddad i en annan, detta är möjligt upp till åtta nivåer[2][7].

2.1.5 Programspråk

Function Block Diagram (FBD) är ett grafiskt programspråk för PLC programmering som introducerades i standarden *IEC 61131-3*. En programinstruktion som är skapad med ett flertal rader kod erbjuds som ett block med in- och utgångspinnar. Användaren anger signaler och data på ingångspinnar som sedan bearbetas inuti blocket till ett eller flera utgångsvärden [2].

Structured Control Language (SCL) är ett textbaserat högnivåspråk som är baserat på PASCAL och korresponderar med Strukturerad Text (ST) definierat i standarden *IEC 61131-3*. Vanliga högnivå kommandon som t.ex. for och while loopar kombineras med element typiska för PLC-programmering

som t.ex. timer, räknare och blockanrop [2].

2.1.6 Watchtable

Watchtable är en tabell där användaren kan ange variabler från data- och funktionsblock. När programmet körs kan variablernas tillstånd eller värde övervakas i tabellen. I tabellen finns även möjligheten att ändra variablers tillstånd. En watchtable kan då användas för simulering genom att påverka ingångars tillstånd och övervaka utgångars respons [2].

2.2 SIMATIC WinCC

SIMATIC WinCC är ett mjukvaruprogram som används för att skapa HMI-skärmar där processer visualiseras och styras. I det här delkapitlet beskrivs objekt och tillämpningar som använts i WinCC.

2.2.1 I/O fält

Ett I/O fält är ett grafiskt objekt som knyts till en HMI-tagg som i sin tur är knuten till en PLC-tagg. Det finns tre typer av I/O fält, input där ett värde anges, input/output där ett värde anges och visas samt typen output där enbart HMI-taggens värde visas [3].

2.2.2 Symboliskt I/O fält

Symboliskt I/O fält är ett grafiskt objekt som knyts till en HMI-tagg. Fältet kan även knytas till en textlista där olika texter är sammankopplade med unika värden. Genom att knyta en HMI-tagg och en textlista till ett symboliskt I/O fält kan man skapa en dropdown meny. Klickar man på fältet kan man välja bland texterna ur textlistan som då sätter HMI-taggens värde till de värdet som är sammankopplat med texten. Symboliska I/O fält finns i typerna input, input/output och output precis som för ett vanligt I/O fält [3].

2.2.3 Faceplate

En faceplate är en konfigurerad grupp av grafiska objekt som sparas undan i projektbiblioteket och kan återanvändas i andra projekt. När man valt objekten som ska utgöra faceplaten går man in i en faceplate editor. Inuti

editorn importerar man en UDT vars variabler knyts samman med objekten. När en faceplate sedan används på en skärm anger man en HMI-tagg som gränssnitt, HMI-taggen består av samma struktur som UDT:en som man importerat inuti editorn [4].

2.2.4 Alarm View

Alarm View är ett fönster som visar rådande larm och deras status i HMI:et. Användaren anger vilka larmklasser som ska synliggöras i fönstret och ifall historiken ska sparas undan i ett register. I fönstret finns en knapp för att kvittera aktiva larm [2].

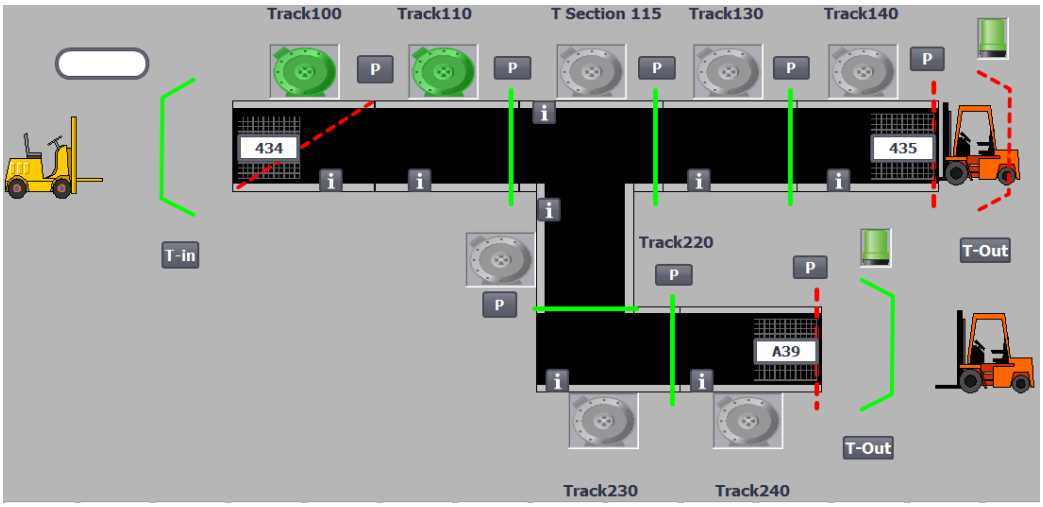
2.3 Process

Uppgiften bygger på en process där pallar ska transporteras längst en montagelinje som består av bansektioner. In- och utmatning av pallar hanteras av truckar och i golvet vid in- och utmatningssektionerna finns en signalslinga som indikerar om en truck befinner sig i arbetszonen. Om en signalslinga är påverkad får den närliggande sektionen inte köra under några omständigheter. Varje bansektion består av ett transportband som drivs av valsar som i sin tur drivs av en motor. Vid varje sektion finns det även en fotocell som är placerad i slutet av sektionen, undantaget är inmatningssektionen där fotocellen är riktad diagonalt över sektionen. En pall transporteras till nästa sektion när denna är redo att ta emot, det vill säga när när den är tom. Pallar får aldrig stöta emot varandra. Programmet ska alltid hålla reda på ifall det finns en pall på varje enskild sektion och reagera smart om en fotocell blir påverkad. Detta innebär att om en fotocell blir påverkad av t.ex. en operatör och ingen pall finns på banan ska motorn inte starta.

Programmet kommer att innehålla ett antal larm och för att återstarta ett objekt efter att ett larm varit aktivt ska en knapp *Reset Tripped Object* (RTO) användas. När ett larm utlösts anses larmobjektet vara trippat, objektet ska då aldrig kunna återstartas genom att enbart kvittera larmet. Efter att larm kvitterats måste objektets trippade status återställas med RTO knappen för att återigen kunna starta.

Uppgiften bygger på att skapa moduler som hanterar banstyrning, dessa moduler ska anropas flera gånger och varje anrop ska styra en separat sektion. Antalet bansektioner och därmed längden på hela banan blir då teoretiskt sett obegränsad.

Hela examensarbetet realiseras och simuleras i en virtuell miljö som avbildar en likvärdig fysisk anläggning. Se figur 1 för en helhetsbild över processen.



Figur 1: *Helhets bild över processen*

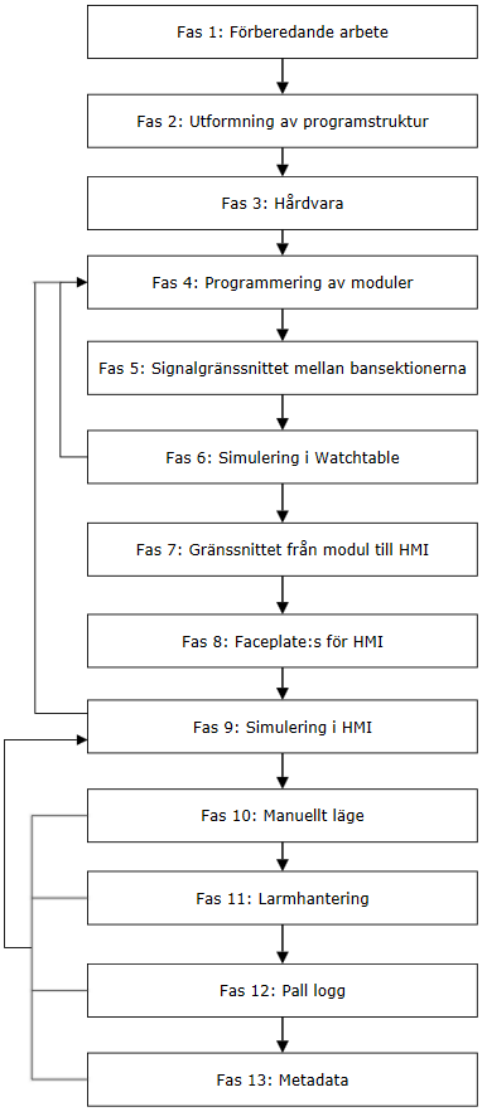
3 Metod

I det här kapitlet redovisas examensarbetets tillvägagångssätt och implementation indelat i 13 faser. Examensarbetet har genomförts på Pöyrys kontor vilket har möjliggjort daglig kommunikation med handledaren i mån av tid.

3.1 Faser

Mjukvaruprogrammet delades in i 13 olika faser som beskrivs i punktlistan nedan. För att se en illustration över faserna i ett arbetsflöde se figur 2

- Fas 1: Förberedande arbete - Uppsättning av den digitala arbetsmiljön
- Fas 2: Utformning av programstruktur - Hur programstrukturen utformades.
- Fas 3: Hårdvara - Val av virtuell hårdvara.
- Fas 4: Programmering av moduler - Hur modulerna programmeras.
- Fas 5: Signalgränssnittet mellan bansektionerna - Kommunikationen mellan bansektioner.
- Fas 6: Simulering i watchtable - Simulering i ett inbyggt simuleringsverktyg.
- Fas 7: Gränssnitt från modul till HMI - Kopplingen mellan PLC:n och HMI.
- Fas 8: Faceplates för HMI - Hur faceplates skapades samt kopplades samman med HMI-taggar.
- Fas 9: Simulering i HMI - Simulering i ett externt simuleringsverktyg.
- Fas 10: Manuellt läge - Visning och funktionalitet när programmet är i manuellt läge.
- Fas 11: Larmhantering - Hantering och presentation av larm.
- Fas 12: Pallregister - Konstruktion och presentation av pallregister.
- Fas 13: Metadata - Insamling och presentation över metadata.



Figur 2: *Fasernas arbetsflöde*

3.2 Utvecklingsprocess

Arbetet har inte utgått efter en specifik utvecklingsmodell men eftersom de till övervägande del involverar mjukvaruprogrammering så påminner arbetsprocessen mycket om Spiralmodellen. Det som avviker mest från

standardmodellen enligt Barry Boehm [6] är att det inte har funnits något behov av riskbedömning så därför finns inte det avsnittet med.

Likheten mellan arbetsprocessen och spiralmodellen är hur en prototyp har framtagits för att sedan vidareutvecklas till nästa prototyp, tills slutligen det färdiga programmet är framtaget. Framtagandet av den första prototypen utförs i fas 1-6 och den andra prototypen utförs i fas 7-9. Fas 11: Larmhantering avviker sig från spiralmodellen på så vis att om spiralmodellen skulle följas strikt så skall fasen egentligen vara med i den första prototypen. Fas 10-13 implementerades som den slutgiltiga prototypen innan en sammanställning gjordes och prototypen utvecklades till ett färdigt program.

3.3 Källkritik

Källorna [1] - [5] är utgivna av Siemens själva och eftersom det är de som har gett ut programmeringsverktyget ansågs det vara trovärdiga källor. [6] är en vetenskaplig artikel som är sponsrad av U.S. Department of Defense och kan därför anses som en seriös källa. [7] är en inofficiell manual som är allmänt känd inom branschen och som är mer djupgående i sina beskrivningar än Siemens egna manual och därför anses den som en trovärdig källa. [8] är utgivna av WMware själva och eftersom det är de som har gett ut programmet så ansågs det vara en trovärdig källa.

4 Genomförande

I det här kapitlet redovisas en ingående beskrivning av genomförandet av samtliga faser som introducerats i delkapitlet 3.1.

4.1 Fas 1: Förberedande arbete

I detta delkapitel beskrivs den inledande fasen som bestod av att ta fram den digitala arbetsmiljön som examensarbetet genomfördes i.

Datorer tillhandahölls av Pöyry med personliga konton för inloggning i deras system. Programvaran *VMware Workstation Player* för den virtuella maskinen överfördes från en extern hårddisk och i datorns BIOS aktiverades virtualisering [8]. Hälften av den fysiska datorns RAM-minne tilldelades den virtuella maskinen. Ett LAN-nätverk sattes upp mellan de två virtuella maskinerna och en delad mapp med gemensam access skapades för att kunna spara och öppna program.

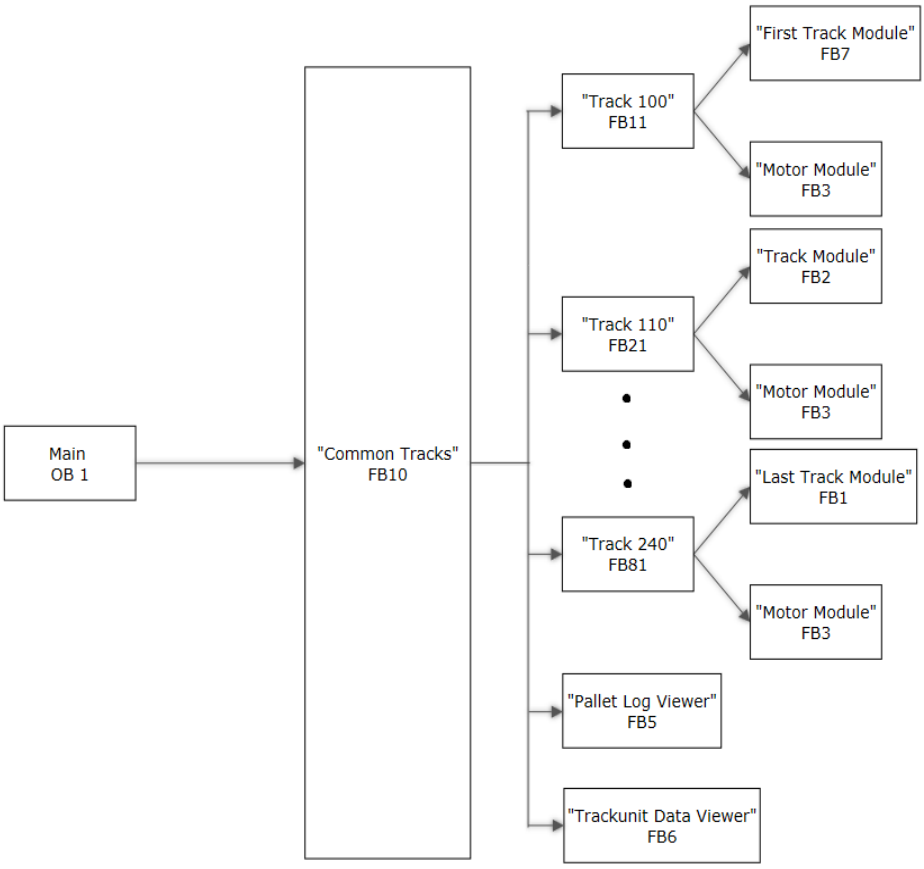
På den ena datorn konfigurerades en Multiuser server av en administratör och den andra användaren lades till som medarbetare. Därefter skapades ett Multiuser projekt på Multiuser servern som både två kunde arbeta i parallellt [5].

4.2 Fas 2: Programstruktur

I detta delavsnitt presenteras programstrukturen som fastställdes i samråd med handledaren Magnus Fransson. Programmets struktur upprättades enligt ett modulariserat synsätt och har implementerats vid programmering i de följande faserna.

Modulerna för ban- och motorstyrning skulle realiseras i individuella funktionsblock. Därefter skulle ett funktionsblock för varje bansektion skapas, i dessa funktionsblock som utgör bansektioner skulle sedan motorstyrningsmodul samt en av banstyrningsmodulerna anropas. Ett funktionsblock *Common Tracks* som utgör hela bansystemet skulle sedan bildas och inuti i det skulle varje funktionsblock som utgör en bansektion anropas. För att programmet ska vara distinkt och lätt att följa skulle *Multi Instance* datablock använts då ett funktionsblock anropar ytterligare ett funktionsblock. Alla funktionsblock

skulle namnges på ett förnuftigt sätt och olika block med likadana funktioner skulle tilldelas samma dekadnummer. För en bild över programstrukturen se figur 3.



Figur 3: *Programstruktur*

4.3 Fas 3: Virtuell hårdvara

I det här delkapitlet förklaras den virtuella hårdvara som valts i TIA Portalen. Som CPU valdes en 1515-2 PN soft PLC ur S7-1500 serien och som HMI valdes en TP 1200 Comfort panel. Det industriella kommunikationsprotokol-

let PROFINET användes för att koppla samman PLC:n och HMI:et. PLC:n tilldelades en IP-adress och sedan adresseras resterande enheter automatiskt av PROFINET i stigande ordning, varje enhet har samma subnätmask. För varje bansektion infogades en ET200SP distribuerad I/O nod i nätverket, varje nod består av en digital ingångs- och utgångsmodul samt en server modul. PLC:n har enbart två ethernet uttag så för att sammankoppla PLC:n med I/O noderna och HMI:et infogades två stycken SCALANCE X208 ethernet switchar.

4.4 Fas 4: Programmering av Moduler

I det här delavsnittet beskrivs framtagningen av modulerna för ban- och motorstyrning. All programkod för logik i modulerna har skrivits i *FBD* och testats med hjälp av simulering i watchtable och HMI.

Den första banstyrningsmodulen som konstruerades var *Track Module* som utgör en mellanliggande bansektion, d.v.s. en bansektion som tar emot pallar från en föregående sektion och levererar till en efterföljande. *Track Module* konstruerades först då den är mest förekommande och används vid programmering av de andra banstyrningsmodulerna. *Track Module* är ett funktionsblock och inledningsvis skapades ingångarna *nödstopp*, *RTO*, *fotocell (P2)*, *föregående sektions fotocell (P1)*, *driftsläge*, *PallID-in* och *motor feedback*. Initialt skapades utsignal till motorstyrningsmodulen för att signalera start/stopp av motor samt *PallID-out*. Alla variabler för intern logik i modulen lades i en UDT *TrackModuleTags* för att hålla modulens blockgränssnitt kompakt och organiserat.

Motorstyrningsmodulen avser styrningen av en motor och inte själva motorn. För modulen bildades ett funktionsblock *Motor Module* med insignalerna *nödstopp*, *motorskyddsbrytare*, *säkring*, *RTO*, *larm återställning*, och *startsignal från banstyrningsmodul*. Som utsignaler skapades initialt *startsignal till motor* och en *feedback* signal till tillhörande bansektion. Statiska variabler som användes för intern logik i modulen lades i en UDT *MotorModuleTags*.

För den första bansektionen som utgör inmatningsstation bildades ett funktionsblock *First Track Module*. Modulen *Track Module* kunde användas för att bilda *First Track Module* genom att kopieras och modifieras.

Insignalerna var likadana som i *Track Module*, men istället för tillståndet på föregående sektions fotocell bildades *T-in* för signalslingan i golvet som anger om en truck är i arbetszonen. *First Track Module* hade inledningsvis samma utsignaler som *Track Module*. Statiska variabler för intern logik lades i en UDT *FirstTrackModuleTags*

För funktionsblocket *Last Track Module* som utgör banstyrningsmodulen för en utmatningsstation kunde *Track Module* återigen användas. *Last Track Module* hade samma insignaler som *Track Module* samt *T-out* för signalslingan i golvet. Utsignalerna var desamma som för *Track Module* samt en utsignal som tänder en lampa för att signalera när en pall finns att hämta. Statiska variabler för intern logik lades i en UDT *LastTrackModuleTags*.

Inledningsvis skapades modulerna samt grundläggande funktioner för att hantera de blockparametrar som nämnts ovan. Mycket av det arbete som sedan genomfördes i efterföljande faser är tillämpningar som placerats inuti modulerna. Genomförandet av tillämpningarna beskrivs i egna faser för tydlighets skull, men de är starkt sammankopplade med denna fas. Nedan följer en kort genomgång av blockparametrar samt statisk lokal data som lagts till i modulerna i samband med nästkommande faser.

När signalgränssnittet utvecklats i *Fas 5: Signalgränsnitt mellan bansektioner* infogades de i banstyrningsmodulerna. UDT:erna *InputsPrevious* och *InputsNext* som insignaler samt *OutputsPrevious* och *OutputsNext* som utsignaler. *FBD* kod skrevs och integrerades med den befintliga koden inuti modulerna för att hantera handskakning mellan bansektioner.

I samband med *Fas 7: Gränssnitt från Modul till HMI* bildades ett nätverk med SCL kod i varje modul för kommunikation med HMI. Som statisk lokal data infogades en UDT *TrackModuleHMI* i banstyrningsmodulerna och UDT:en *MotorModuleHMI* i motorstyrningsmodulen.

Vid genomförandet av *Fas 11: Larmhantering* skapades *Program_Alarm* inuti banstyrningsmodulerna. Larmen gäller då en pall fastnat eller trillat av banan. En utsignal *PalletLostMotor* bildades och används för att stanna motorn då ett larm inträffat. I motorstyrningsmodulen bildades *Program_Alarm* för motorskyddsbrytare, säkring samt ett för underhåll av motor. Kod skrevs i *FBD* och integrerades med befintlig kod.

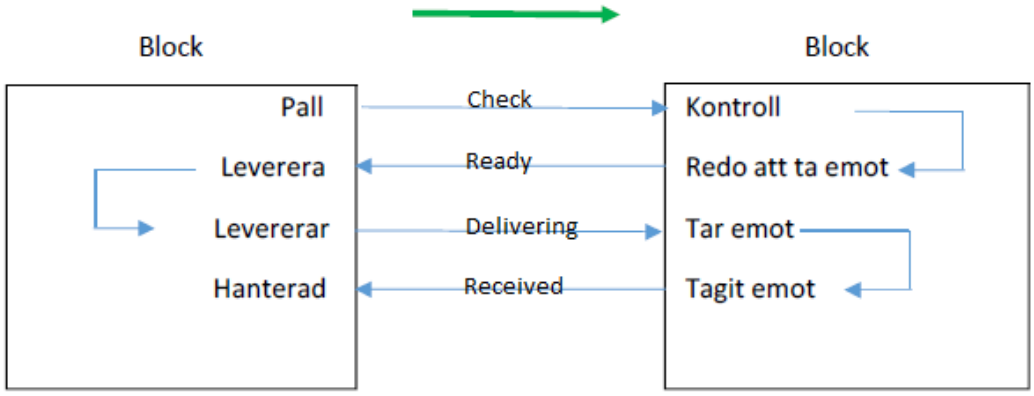
I samband med *Fas 12: Pallregister* bildades en utgång *PalletLog* ur varje banstyrningsmodul. *FBD* kod för att registrera pallinlägg i registret integrerades med befintlig kod.

Vid implementationen av *Fas 13: Metadata* skapades en utgång *DataLog* i banstyrningsmodulerna. *FBD* kod skrevs och integrerades i modulerna för att registrera metadata. I motorstyrningsmodulen bildades en utgång *MotorData* för att registrera metadata. *FBD* kod för att registrera metadata integrerades med befintlig kod.

När det kom till att undersöka programmets skalbarhet introducerades en ny banstyrningsmodul *T Track Module*. Modulen omfattar en T-sektion som kan ta emot från en sektion och sedan skicka vidare till en av två olika sektioner. För att konstruera denna modul kopierades *Track Module* och sedan modifierades den för att kunna leverera pallar till två olika sektioner. Modifikationen bestod av att ytterligare en in- och utgång bildades för att hantera handskakning med tre olika bansektioner. *FBD* kod för att hantera leverans till en av två olika sektioner integrerades med befintlig kod. Även kod för att avgöra vilken sektion inkommande pall ska levereras till skapades. Villkoret som avgör vilken sektion leverans sker till är det första tecknet i ID't hos den inkommande pallen. Alla statiska variabler för intern logik lades i en UDT *T-TrackModuleTags*. Hur den fysiska T-sektionen ska hantera leverans till två olika sektioner har inte behandlats i modulen. Vid implementation av en fysisk anläggning kommer en utgång för att aktivera en kicker integreras i koden.

4.5 Fas 5: Signalgränssnitt mellan bansektioner

I det här delkapitlet redovisas signalgränssnittet för handskakning mellan bansektioner. En handskakning för leverans av en pall mellan två sektioner följer tillvägagångssättet illustrerat i figur 4.



Figur 4: *Handskakning mellan bansektioner vid palleverans*

Fyra stycken UDT:er som innehöll de nödvändiga signalerna för handskakning mellan bansektioner bildades, se figur 5 och 6.

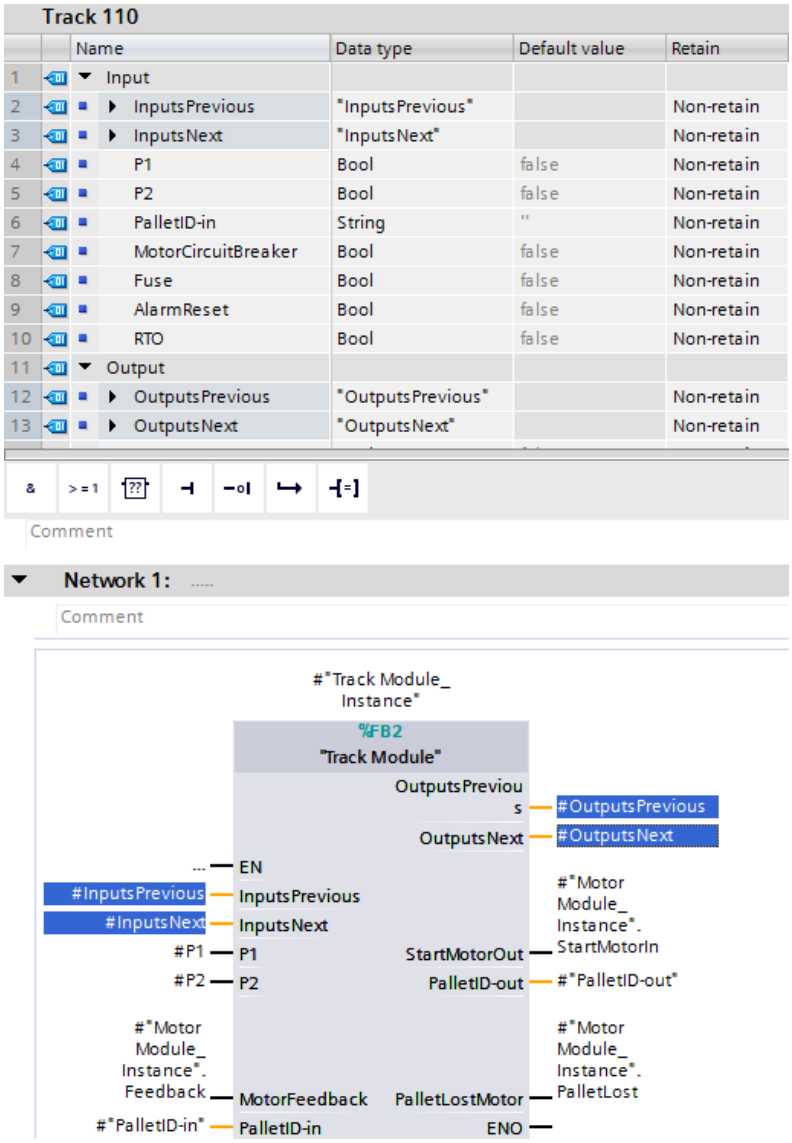
InputsPrevious	"InputsPrevious"	Handshake signals from the previous trackunit
Check	Bool	Check enabled by the previous trackunit
Delivering	Bool	Receiving a pallet from the previous trackunit
InputsNext	"InputsNext"	Handshake signals from the next trackunit
Ready	Bool	Ready to deliver a pallet to the next trackunit
Received	Bool	Pallet has been delivered to the next trackunit

Figur 5: *User-Defined Data Types InputsPrevious och InputsNext.*

OutputsPrevious	"OutputsPrevious"	Handshake signals to the previous trackunit
Ready	Bool	Ready to receive a pallet from the previous trackunit
Received	Bool	Pallet has been received from the previous trackunit
OutputsNext	"OutputsNext"	Handshake signals to the next trackunit
Check	Bool	Trackunit has a pallet, enable check in next trackunit
Delivering	Bool	Delivering a pallet to the next trackunit

Figur 6: *User-Defined Data Types OutputsPrevious och OutputsNext.*

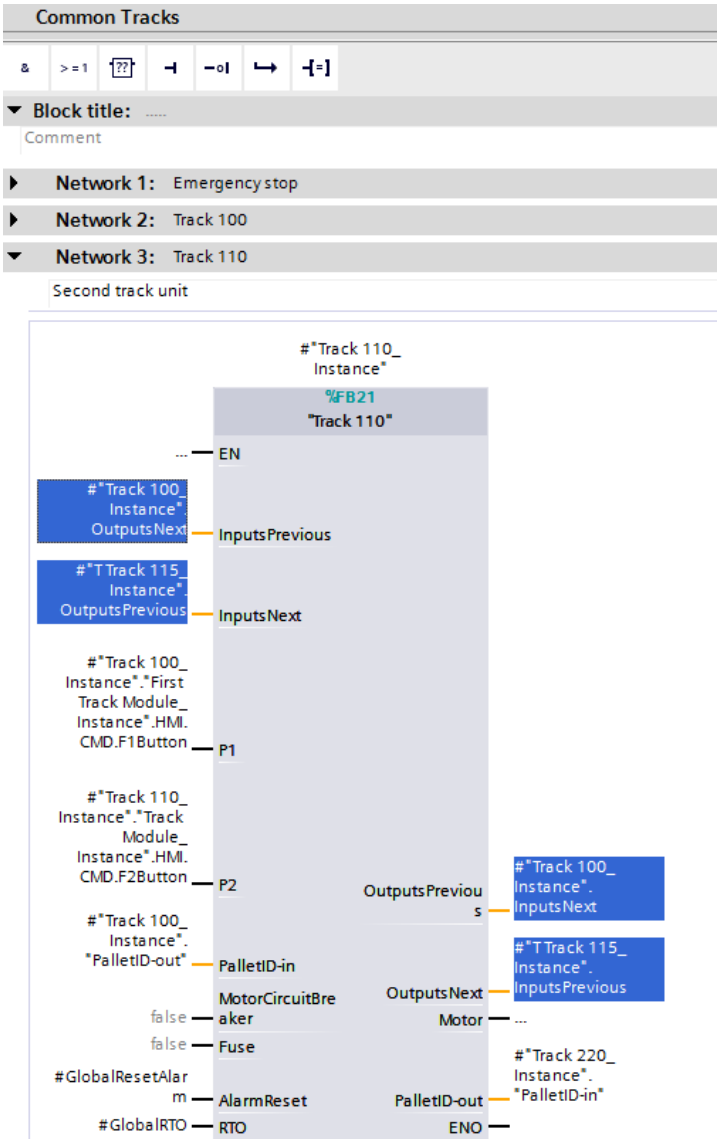
UDT:erna infogades som in- och utgångar hos banstyrningsmodulerna och signalerna integrerades i modulernas interna *FBD* kod. Därefter infogades UDT:erna som in- och utgångar på funktionsblocket som utgör en bansektion. De knöts sedan till korresponderande in- och utgångspinnar på anropet av banstyrningsmodulen, se figur 7.



Figur 7: *Track 110, signalgränssnittets in- och utgångar knyts till in- och utgångspinnar på anropet av modulen.*

Sedan togs ett steg ut i programstrukturen till funktionsblocket *Common Tracks* som utgör hela bansystemet med anrop av alla bansektions block. Där anknöts signalgränssnittet för handskakning mellan anropen av de olika bansektionerna. Det innebär att *OutputsPrevious* knöts till *InputsNext* och

OutputsNext knöts till *InputsPrevious* för intilliggande sektioner, se figur 8.



Figur 8: *Common Tracks, signalgränssnittets in- och utgångar knutet till föregående/efterföljande sektions in- och utgångspinnar.*

4.6 Fas 6: Simulering i watchtable

I det här avsnittet presenteras genomförandet av den första simuleringsfasen. För en kort teknisk beskrivning av watchtable se avsnitt 2.1.6.

Det skapades en watchtable och variablerna som skulle övervakas och justeras lades in. Variablerna lades in sektionsvis för att få fram en helhetsbild över systemet. Simuleringen utfördes så att givarna, fotocellerna och truckslingorna, påverkades och medförde att olika funktioner startades och stoppades. För att påverka och övervaka watchtable:ns inlagda variabler krävdes att funktionen monitor var aktiv. Monitor funktionen användes också i funktionsblocken och då gavs tillstånden för de logiska operationerna. Se figur 9 för watchtable tabellen med de inlagda variablerna.

	Name	Displa..	Monitor..	Tag comment
2	// Track 100 - First			
3	"Common Tracks_DB"."Track 100_Instance"."First Track Module_Instance".FirstTrackModuleTags.HasPallet	Bool		Pallet on the trackunit
4	"Common Tracks_DB"."Track 100_Instance".P1	Bool		Diagonal photocell on the trackunit
5	"Common Tracks_DB"."Track 100_Instance".T-in"	Bool		Sensor forklift in the workzone
6	"Common Tracks_DB"."Track 100_Instance".InputsNext.Ready	Bool		Ready to deliver a pallet to the next trackunit
7	"Common Tracks_DB"."Track 100_Instance".InputsNext.Received	Bool		Pallet has been delivered to the next trackunit
8	"Common Tracks_DB"."Track 100_Instance".OutputsNext.Check	Bool		Trackunit has a pallet, enable check in next trackunit
9	"Common Tracks_DB"."Track 100_Instance".OutputsNext.Delivering	Bool		Delivering a pallet to the next trackunit
10	"Common Tracks_DB"."Track 100_Instance".MotorCircuitBreaker	Bool		Motor Protective Circuit Breaker
11	"Common Tracks_DB"."Track 100_Instance".Fuse	Bool		Motor fuse
12	"Common Tracks_DB"."Track 100_Instance".AlarmReset	Bool		Button to reset Fuse/CB alarm
13	"Common Tracks_DB"."Track 100_Instance".RTO	Bool		Button to reset tripped objects after an alarm has been acknowledged
14	"Common Tracks_DB"."Track 100_Instance".Motor	Bool		Motor that runs the track
15	"Common Tracks_DB"."Track 100_Instance"."PalletID-in"	String		Pallet ID, read from the HMI
16	"Common Tracks_DB"."Track 100_Instance"."PalletID-out"	String		Pallet ID to send along with the pallet to the next trackunit
17	"Alarms".AlarmsCommon	Hex		Bit 0 = E-stop
18	// Track 110			
19	"Common Tracks_DB"."Track 110_Instance"."Track Module_Instance".TrackModuleTags.HasPallet	Bool		Pallet on the trackunit
20	"Common Tracks_DB"."Track 110_Instance".P1	Bool		Photocell on the previous trackunit
21	"Common Tracks_DB"."Track 110_Instance".P2	Bool		Photocell on the trackunit
22	"Common Tracks_DB"."Track 110_Instance".InputsPrevious.Check	Bool		Check enabled by the previous trackunit
23	"Common Tracks_DB"."Track 110_Instance".InputsPrevious.Delivering	Bool		Receiving a pallet from the previous trackunit

Figur 9: *Watchtable med de inlagda variablerna.*

Watchtable:n användes som en första simulator för att det visades tydligt vad resultatet blev av en process och det behövdes inga kopplingar till andra system. Watchtable var dock begränsande på så vis att ju mer avancerat programmet blev desto fler variabler krävdes. Variablerna för ett flertal sektioner kunde då inte visas samtidigt på grund av att de inte fick plats på datorskärmen.

Efter att en watchtable simulerades återgicks det sedan tillbaka till fas 4 enligt figur 2. Det implementerades nya och effektivare lösningar men samtidigt kompletterades fel som uppkom. Denna process upprepades ett flertal gånger innan ett fullt fungerande och effektivt program hade

framtagits.

4.7 Fas 7: Gränssnitt från Modul till HMI

I det här delavsnittet beskrivs hur gränssnittet mellan modulerna och HMI:et sattes upp. Programmeringsspråket ändras från FBD till SCL för att det uppleveds som smidigare för kontinuerlig uppdateringen av variablerna.

Gränssnittet från modul till HMI programmerades i tillhörande funktionsblock. Det skapades en UDT för varje typ av modul som innehöll de nödvändiga variablerna för kommunikationen mellan modulen och HMI:et. I modulerna bildades en statisk tagg *HMI* med tillhörande UDT som datatyp. Därefter skapades HMI-taggar med samma UDT som datatyp och kopplades ihop med de statiska PLC-taggarna i modulerna..

Därpå gjordes funktionerna för den kontinuerliga uppdateringen från modul till HMI:et på så vis att funktionsblockets variabler skrevs över till UDT:ens variabler.

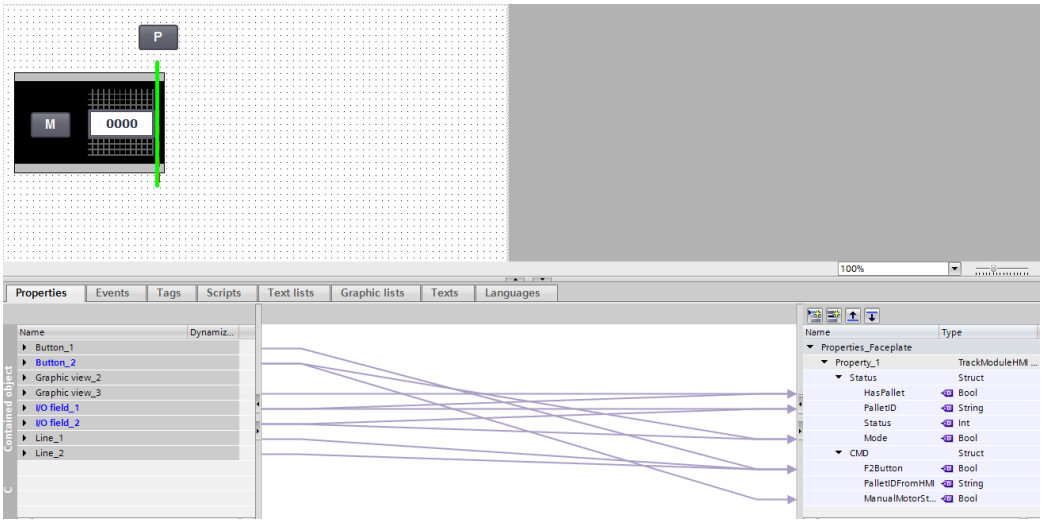
Alla skapade UDT:er sparades undan i projektbiblioteket för vidare användning och för att bibehålla modulstrukturen.

4.8 Fas 8: Faceplates

I det här delkapitlet förklaras hur faceplates bildades och hur de kopplades till PLC-programmet.

Det skapades en faceplate för varje typ av sektion samt en för motorn. När en faceplate användes mer en gång så kopierades och återanvändes den, på så vis bibehölls modulstrukturen. En faceplate skapades genom att de önskade grafiska objekten som skulle innehållas samlades inom ett område på HMI-skärmen. Därefter markerades objekten och alternativet skapa faceplate valdes. Den namngavs och sparades under projektbiblioteket. Därpå öppnades redigeringsverktyget och under egenskaper valdes de önskade UDT:erna. HMI-taggarna från UDT:erna kopplades sedan samman med de grafiska objekten genom processvärde eller visning vid olika tillstånd. Se

figur 10 för en faceplate för *Track Module*.



Figur 10: *Faceplate för en mellanliggande sektion med kopplingar till HMI-taggar.*

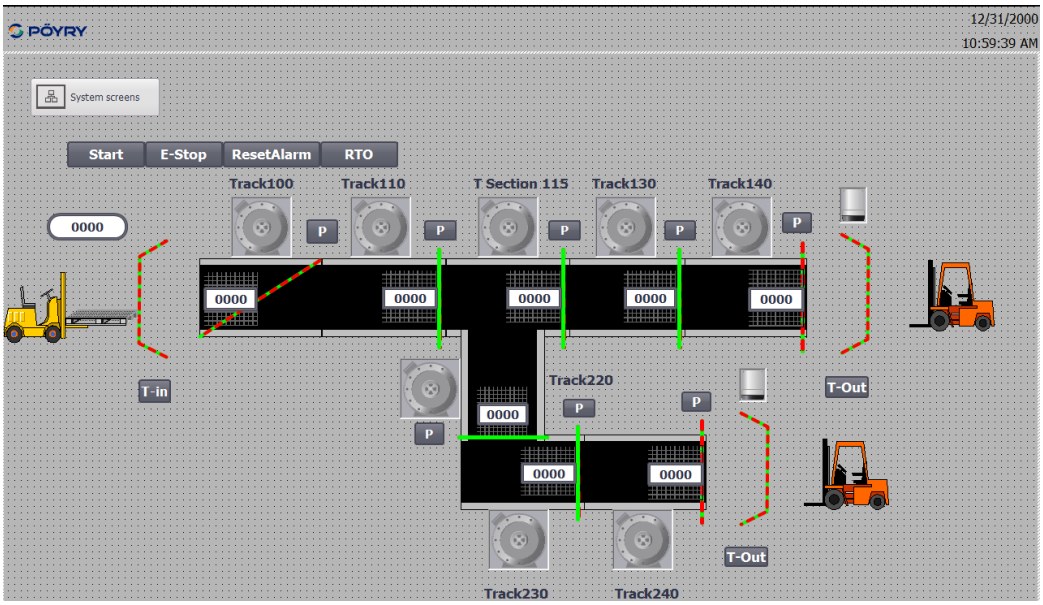
Om ett grafiskt element har två eller flera olika grafiska tillstånd skapades en grafisk lista där olika värden av den sammankopplade taggen gav olika tillstånd för elementet. Taggarna under egenskaper skapades som interface för faceplaten och från interfacet gjordes kopplingar mot ekvivalenta HMI-taggar.

4.9 Fas 9: Simulering i HMI

I det här avsnittet presenteras den andra simuleringsfasen, simulering i HMI.

När faceplates och gränssnittet från PLC till HMI var gjorda påbörjades simulering i HMI. Från HMI:et simulerades processen i realtid fast eftersom systemet inte var uppkopplat mot ett fysiskt system styrdes processen manuellt. Processen styrdes genom knapptryckningar på olika funktioner som simulerade hur processen hade körts i ett fysiskt system. Den stora fördelen med HMI:et till skillnad från simulering i watchtable är att det

presenterades som en grafisk simulation. På så vis skapades en överblick av systemet och det gavs en mer verklighetstrogen bild över processen. Se figur 11 för simulering i HMI:et.



Figur 11: *Simulering i HMI*

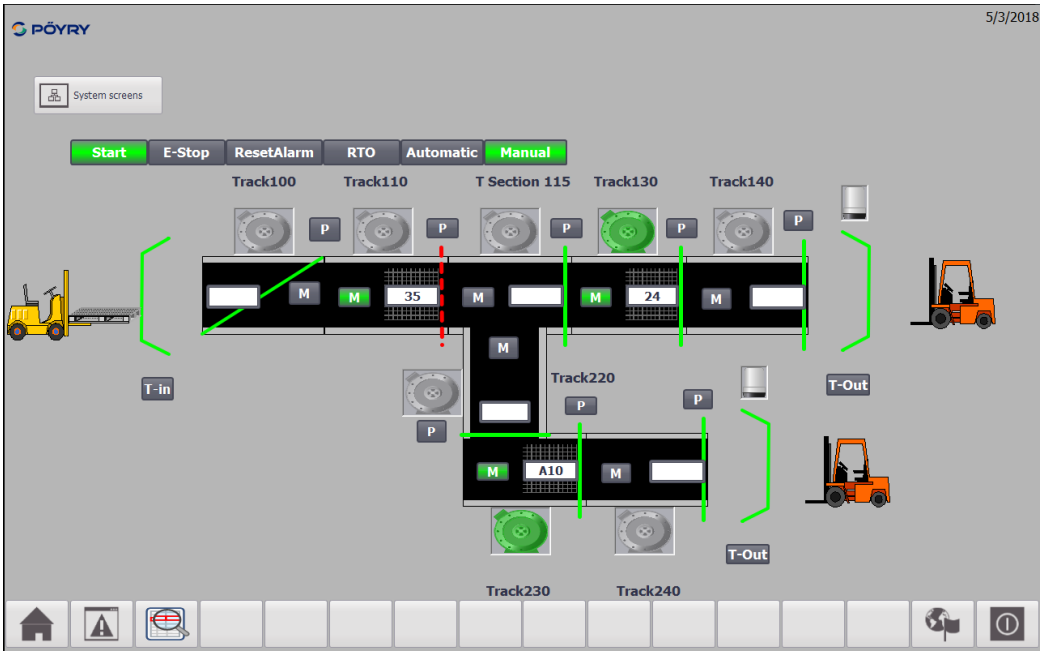
På motsvarande sätt som vid simulering med watchtable var detta en fortlöpande process där det gicks tillbaka till fas 4 och kontinuerligt implementerades nya funktioner enligt arbetstillvägagångssättet i figur 2.

4.10 Fas 10: Manuellt läge

I det här kapitlet redovisas hur det manuella läget implementerades i systemet.

För att programmet ska avspegla en verklig fabrik valdes att implementera ett manuellt läge. I en verklig fabrik används manuellt läge till flera funktioner än vad som används av det här programmet. Programmet kan hantera dessa funktioner men presenteras då inte eftersom programmet enbart är tänkt som en simulering av hårdvara. För att aktivera manuellt läge trycker

man på *Manual*-knappen i HMI:t. När manuellt läge är tillslaget styrs allting av en tänkt fysisk operatör. Systemet styrs av att operatören bestämmer vilka motorer som körs. Motorerna startades och stoppades genom att trycka på *M*-knappen som endast är synlig i manuellt läge. Det valdes dock att programmet skulle ha stöd av semiautomatiska säkerhetsfunktioner som förhindrade att motorerna kördes när en säkerhetsrisk uppstår, trots att programmet befinner sig i manuellt läge. De semiautomatiska säkerhetsfunktionerna togs med eftersom programmet ska kunna köras i en verklig fabrik och då krävs en säker arbetsmiljö. Figur 12 visar manuellt läge simulerat i HMI:et



Figur 12: *Manuellt läge där motorstyrning och semiautomatiska funktionerna visas*

All kod lades direkt i modulerna och är principiellt samma oavsett typ av modul. Kodmässigt gjordes så att om manuellt läge är till skickas inte längre några signaler mellan sektionerna, d.v.s. signalgränssnittet släcks. Detta gjorde att modulernas enda uppgift var att registrera om det fanns en pall på sektionen eller inte samt förhindra motorn från att köras om det uppstår en

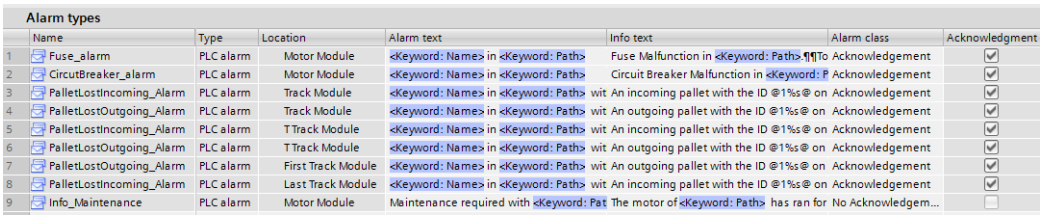
säkerhetsrisk. I och med att signalgränssnittet släcks är det möjligt att en pall kan skapas på vilken sektion som helst i systemet. Pallarna skapades genom att i HMI:et lägga till ett nytt pall-ID på önskad sektion och togs bort genom att ta bort pall-ID:et. På så vis fungerade programmet att det skapades en ny pall på sektionen och skulle den förflyttas fick den tas bort och i nästa sektion skapas som en ny pall, allt genom manuell inmatning. Skapades en pall i manuellt läge och det återgår tillbaks till automatiskt läge så återupptogs den automatiska processen. På så vis detta simulerades fortfarande pallförflyttningen.

4.11 Fas 11: Larmhantering

I det här avsnittet förklaras hur larmhanteringen för systemet fungerar.

Larmhanteringen utfördes på två olika sätt, från önskemål av handledaren. Första sättet kan ses som det "gamla sättet" och användes enbart för nödstopp. Det "gamla sättet" programmerades så att det skapades ett larm i *HMI Alarms* som kopplades till en tagg i PLC. Eftersom nödstoppet är centralt för hela anläggningen så programmerades en SR-vippa i *Common Tracks*. I varje banmodul lades in en likadan SR-vippa som i *Common Tracks* med ingångssignaler från *Common Tracks* som styrde set och reset som i sin tur låg som krav för olika funktioner skulle få startas. Set styrdes genom nödstoppsknappen och reset styrdes genom RTO knappen från HMI:et. Därefter konfigurerades inställningarna i *Alarm Classes* hur larmet presenterades på HMI:et genom olika färger för om larmet var kvitterat eller inte, inkommande eller utgående.

Det andra sättet kan ses som det nya sättet vilket användes för de övrig larmen då det upplevdes som smidigare. Det nya sättet programmerades så att det använde sig utav av ett färdigt *Program_Alarm* block tillsammans med *Get_AlarmState* block i PLC:n.*Program_Alarm* blocket aktiverades av en signal när larmet är aktivt som i sin tur aktiverade *Get_AlarmState*. Larmet hade då skapats och det som konfigurerades sedan var presentationsinställningar som *Alarm Classes* och informationstexter var larmet utlöstes någonstans och tillvägagångssätt för att åtgärda larmet. Se figur 13 för alla programmerade larm med *Program_Alarm* metoden.



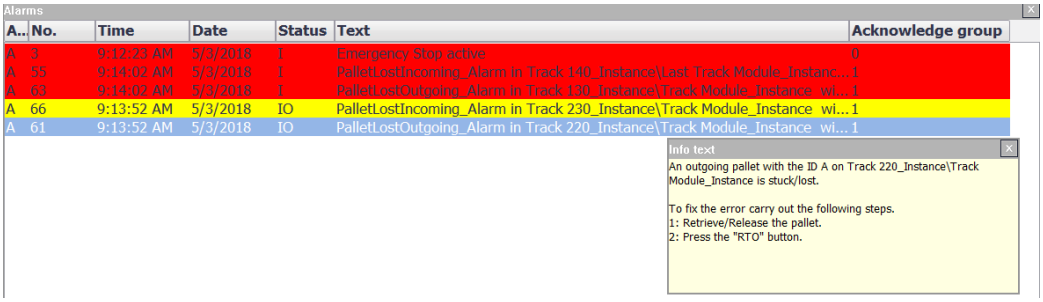
Alarm types

	Name	Type	Location	Alarm text	Info text	Alarm class	Acknowledgment
1	Fuse_alarm	PLC alarm	Motor Module	<Keyword: Name> in <Keyword: Path>	Fuse Malfunction in <Keyword: Path>.¶¶To	Acknowledgement	☑
2	CircuitBreaker_alarm	PLC alarm	Motor Module	<Keyword: Name> in <Keyword: Path>	Circuit Breaker Malfunction in <Keyword: P	Acknowledgement	☑
3	PalletLostIncoming_Alarm	PLC alarm	Track Module	<Keyword: Name> in <Keyword: Path> wit	An incoming pallet with the ID @1%s@ on	Acknowledgement	☑
4	PalletLostOutgoing_Alarm	PLC alarm	Track Module	<Keyword: Name> in <Keyword: Path> wit	An outgoing pallet with the ID @1%s@ on	Acknowledgement	☑
5	PalletLostIncoming_Alarm	PLC alarm	T Track Module	<Keyword: Name> in <Keyword: Path> wit	An incoming pallet with the ID @1%s@ on	Acknowledgement	☑
6	PalletLostOutgoing_Alarm	PLC alarm	T Track Module	<Keyword: Name> in <Keyword: Path> wit	An outgoing pallet with the ID @1%s@ on	Acknowledgement	☑
7	PalletLostOutgoing_Alarm	PLC alarm	First Track Module	<Keyword: Name> in <Keyword: Path> wit	An outgoing pallet with the ID @1%s@ on	Acknowledgement	☑
8	PalletLostIncoming_Alarm	PLC alarm	Last Track Module	<Keyword: Name> in <Keyword: Path> wit	An incoming pallet with the ID @1%s@ on	Acknowledgement	☑
9	Info_Maintenance	PLC alarm	Motor Module	Maintenance required with <Keyword: Pat	The motor of <Keyword: Path> has ran for	No Acknowledgem...	☐

Figur 13: *Sammanställning av alla* Program_Alarm

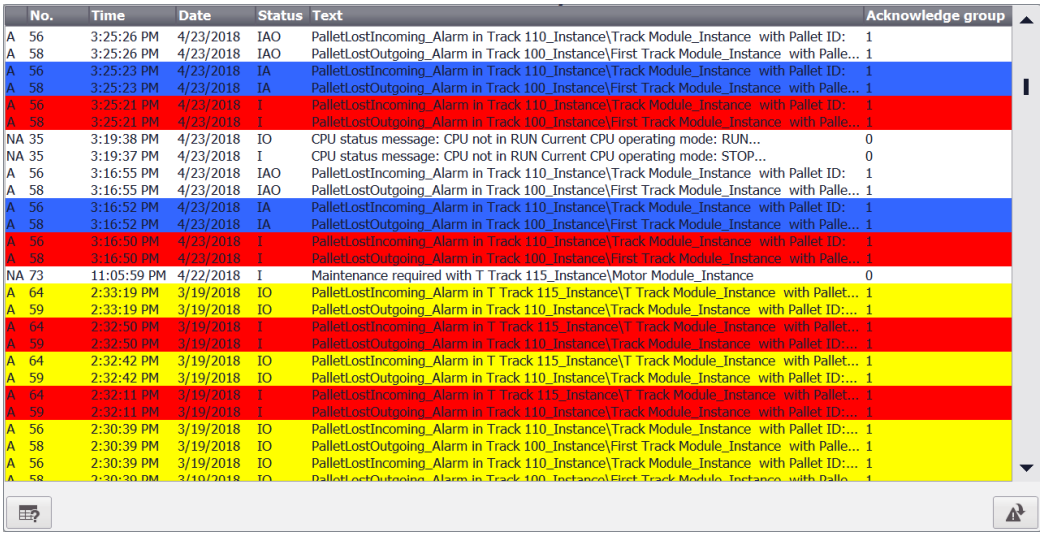
För larm med SR-vippor så lagrades dess data i ett *Single Instance* datablock och för larm som användes sig utav *Program_Alarm* så lagrades dess data i *Multi Instance* datablock.

Larmen *PalletLostOutgoing_Alarm* och *PalletLostIncoming_Alarm* som motsvaras av att en pall har fastnat på en sektion eller trillat av sektionen inaktiveras när programmet kördes i manuellt läge.
Om ett nytt larm inkommer så aktiveras larmskärmen och visualiseras som ett extrafönster. Objekten för där larmet utlöstes sattes i ett trippat läge tills det återställdes. Ifall ett larm uppstod visualiserades även felet med en tydlig förändring i HMI:et. T.ex. Om motorskyddsbrytaren har gått så blir motorn för denna sektion röd. Larmet låg kvar i extrafönstret tills det kvitterades. Kvitterades bara larmen så startades inte objekten igen utan befanns sig i det trippade läget och det krävdes att RTO knappen också trycktes ner. Extrafönstret skapades utav ett befintligt grafiskt verktyg, *Alarm View*, för presentation av larmen. I *Alarm View* så bockades det in vilka larmklasser som åstundades att presenteras. Se figur 14 för larm representerade i *Alarm View*.



Figur 14: *Extrafönstret med aktiva larm*

Det skapades också ett register med all larmhistorik som sparades undan som en TXT fil på hårddisken. Registret för larmhistorik gjordes genom *Alarm View* fast det valdes att den skulle spara undan larmen istället för att det visades aktiva larm. Se figur 15 för larmregister med den senaste larmhistoriken.



Figur 15: *Register med larmhistorik*

4.12 Fas 12: Pallregister

För varje bansektion upprättades ett register över alla pallar som varit på sektionen. Första delen av detta avsnitt beskriver hur det konstruerats och den andra delen hur det presenteras i HMI.

4.12.1 Konstruktion

I detta avsnitt beskrivs hur ett pallregister konstruerats. Först skapades en UDT bestående av all pallinformation som lagras. Den döptes till *PalletEntry* och benämns som pallinlägg i rapporten. Ett pallinlägg består av ID, löpnummer, datum och tid då pallen ankom samt hur lång tid pallen befann sig på sektionen. Se figur 16.

PalletEntry

	Name	Data type	Default value	Comment
	PalletID	String	''	The pallet id
	DateTime	Date_And_Time	DT#1990-01-01-00:00:00	Date and time the pallet arrived at the trackunit
	Nbr	Int	0	The pallet running number
	TimeOnTrack	Time	T#0s	The amount of time the pallet spent on the trackunit

Figur 16: *User-Defined Data Type PalletEntry.*

För att upprätta ett register bildades en UDT vid namn *PalletLog* som består av en vektor med pallinlägg. Det vill säga UDT:n *PalletLog* består av en vektor med UDT:n *PalletEntry*. Storleken för alla pallregister i programmet fastställs av storleken som anges på vektorn i UDT:n *PalletLog*. Se figur 17 för en bild på *PalletLog*.

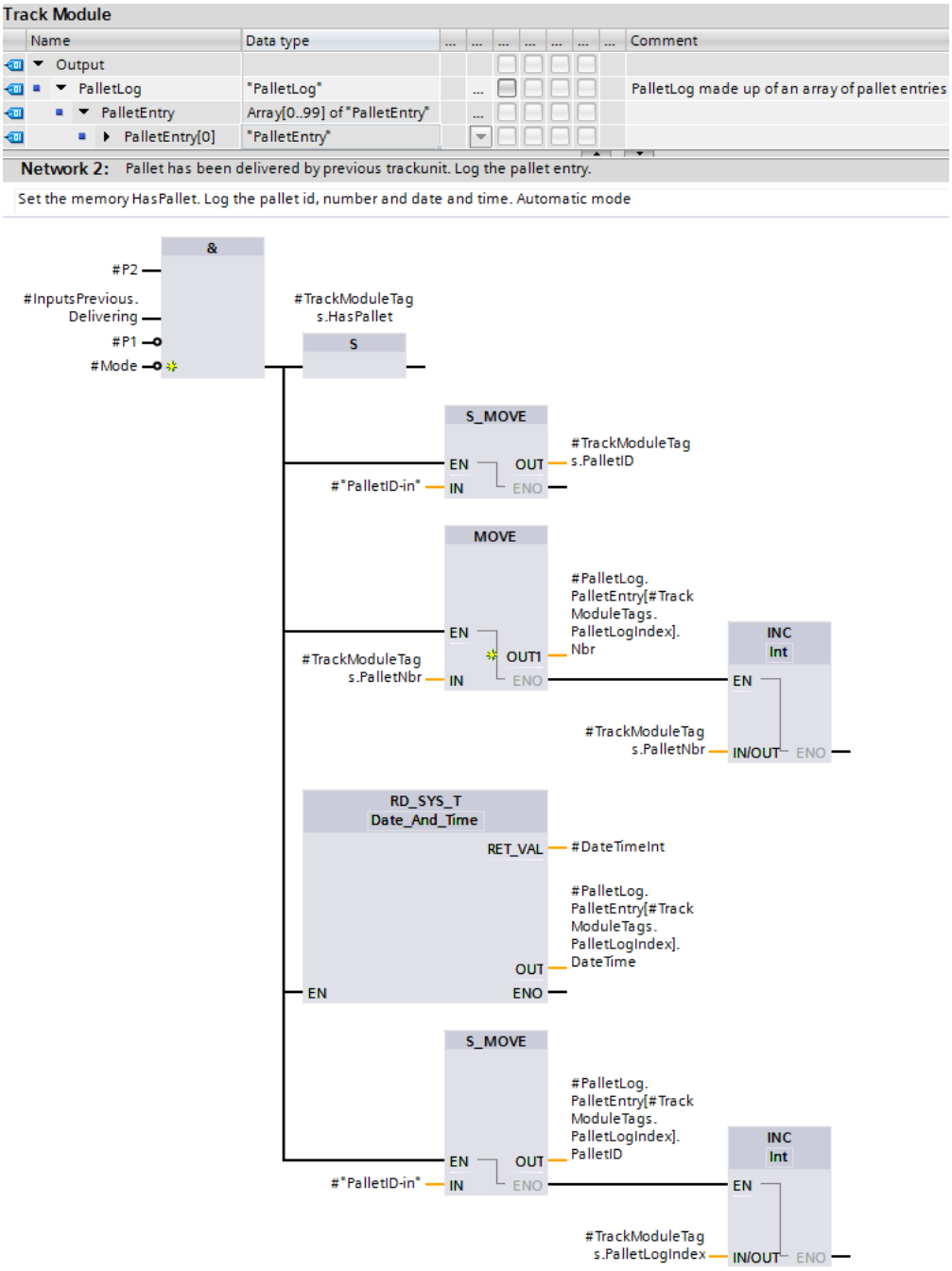
PalletLog

Name	Data type	Comment
PalletEntry	Array[0..99] of "PalletEntry"	
PalletEntry[0]	"PalletEntry"	
PalletID	String	The pallet id
DateTime	Date_And_Time	Date and time the pallet arrived at the trackunit
Nbr	Int	The pallet running number
TimeOnTrack	Time	The amount of time the pallet spent on the trackunit
PalletEntry[1]	"PalletEntry"	
PalletID	String	The pallet id
DateTime	Date_And_Time	Date and time the pallet arrived at the trackunit
Nbr	Int	The pallet running number
TimeOnTrack	Time	The amount of time the pallet spent on the trackunit
PalletEntry[2]	"PalletEntry"	
PalletEntry[3]	"PalletEntry"	

Figur 17: *User-Defined Data Type PalletLog.*

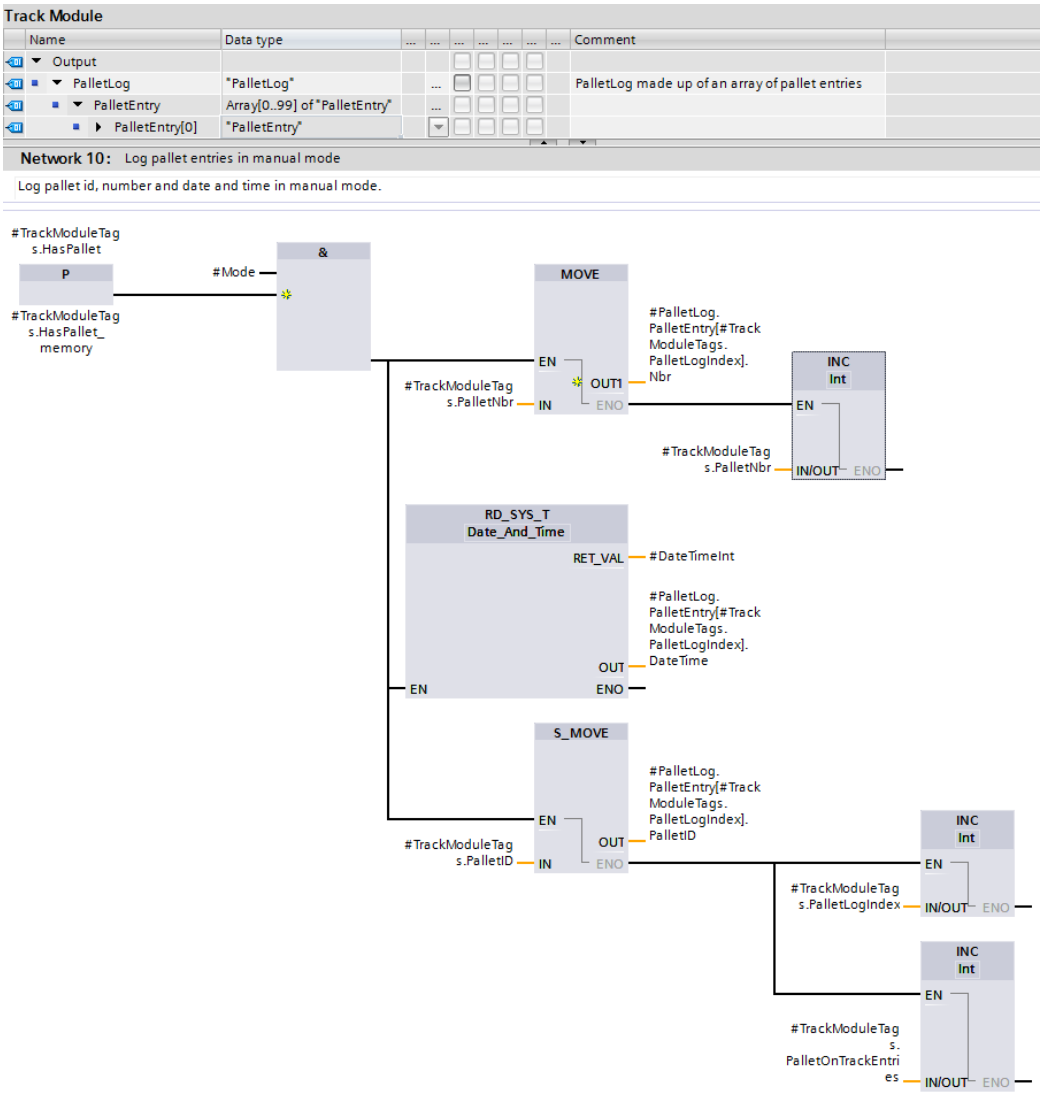
Uppbyggnad av ett pallregister i en banstyrningsmodul sker snarlikt i varje modul. Då modulen *Track Module* som utgör en mellanliggande sektion är den mest förekommande beskrivs konstruktionen av ett pallregister inuti den.

I banstyrningsmodulerna skapades en utgång vid namn *PalletLog*. Utgångens datatyp är UDT:n PalletLog i figur 17. På så sätt bildades ett pallregister som utgång från banstyrningsmodulerna. Inuti banstyrningsmodulerna detekteras varje pallankomst och då registreras pallinformationen. Logiken för att detektera pallankomst i automatiskt läge utvecklades i fas fyra. Därmed kunde registreringen av pallinformation läggas till på samma plats. För logiken rörande registrering av pallinformation i automatiskt läge se figur 18.



Figur 18: *Track Module, detektera pallankomst i automatiskt läge och registrera pallinformation.*

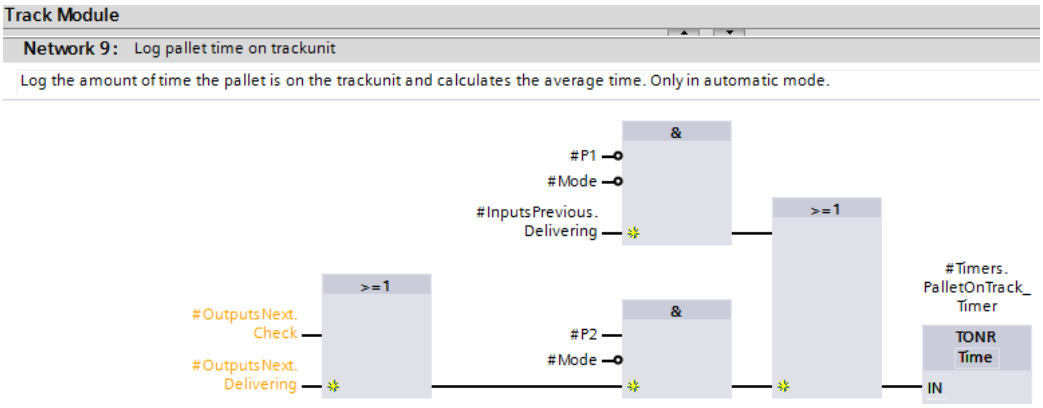
För detektering av pallankomst i manuellt läge bildades ett nytt nätverk i banstyrningsmodulen. Med den booleska variabeln *HasPallet* anges det om en pall finns på sektionen, därför bevakas den för en signalförändring från falsk till sann. När signalförändringen sker och programmet är i manuellt läge registreras pallinformationen. För logiken rörande registrering av pallinformation i manuellt läge se figur 19.



Figur 19: *Track Module, detektera pallankomst i manuellt läge och registrera pallinformation*

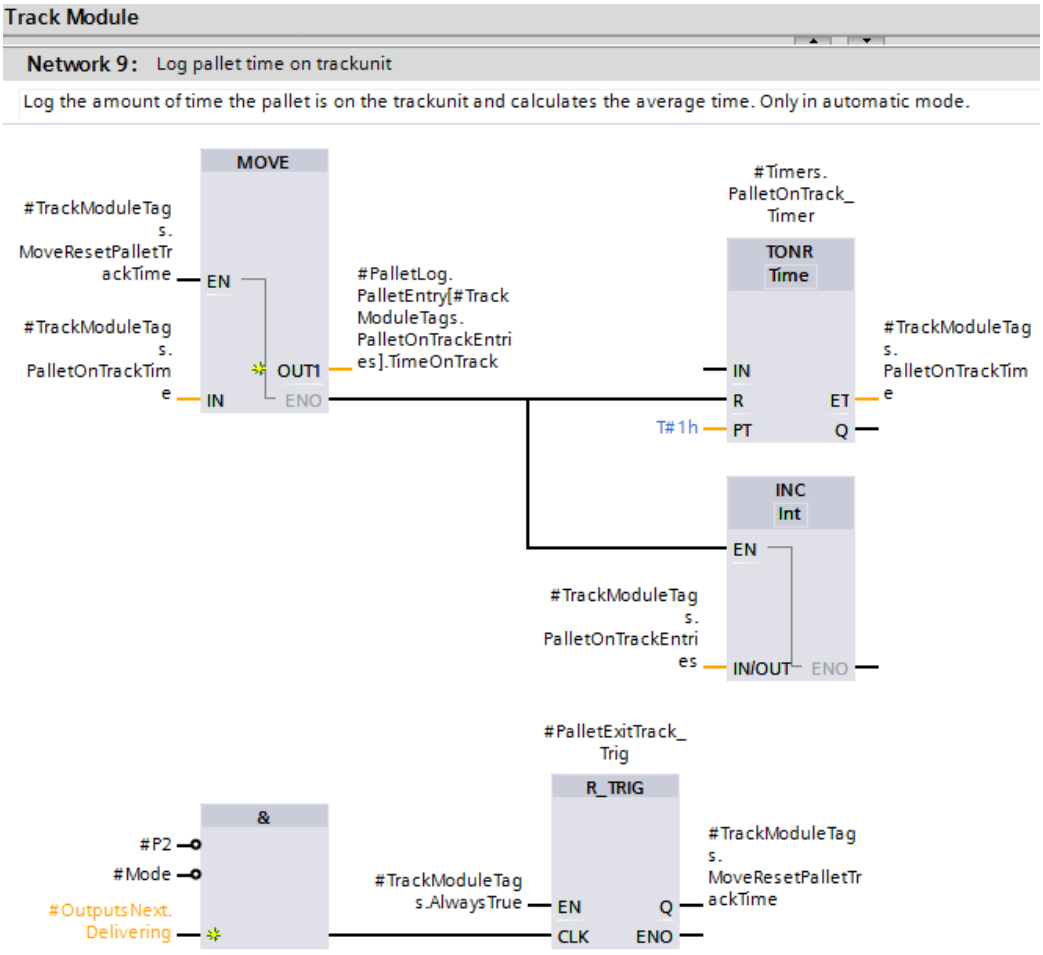
Tiden som en pall befunnit sig på en sektion registreras enbart när anläggningen körs i automatisk läge och sker på följande sätt. När föregående bansektion signalerar att den levererar en pall samt dess fotocell (P1) är inaktiv innebär det att hela pallen kommit in på den aktuella

sektionen. Timern aktiveras och tid ackumuleras så länge pallen befinner sig på sektionen. Pallen befinner sig på den aktuella sektionen så länge fotocellen (P2) och en av signalerna *Check* eller *Delivering* till nästkommande sektion är aktiva. Se figur 20.



Figur 20: *Track Module, aktivering av TONR timer som beräknar hur lång tid en pall har befunnit sig på bansektionen i automatiskt läge.*

När en pall lämnat bansektionen återställs timern och det ackumulerade tidsvärdet flyttas till sektionens pallregister. Ett pall utträde detekteras då sektionen signalerar till nästkommande sektion att en pall levereras och dess fotocell (P2) blir inaktiv. När utträde detekterats blir signalen *MoveReset-PalletTrackTime* sann under ett cykelvarv och triggar en förflyttning av den ackumulerade tiden samt återställer timern. Se figur 21.



Figur 21: *Track Module, förflyttning av ackumulerad tid till pallregister samt återställning av timern.*

Två stycken heltal *PalletLogIndex* och *PalletOnTrackEntries* används som index så att pallinformation förflyttas till korrekt pallinlägg i banstyrningsmodulens register. Orsaken till att två indexvärden används är för att pallinformation som lagras skiljer sig åt beroende på om anläggningen är i automatiskt eller manuellt läge. Indexvärdena pekar på den plats i registret där nästa insättning kommer att ske och när den utförts ökas värdena med ett, detta går att se i figur 18, 19 och 21. När indexvärdena motsvarar registrets storlek återställs de till noll och gamla pallinlägg börjar då att

skrivas över. Anledningen till detta är att i register presenteras enbart den senaste pallinformationen, den fullständiga är tänkt att lagras i ett överordnat system.

För lagringen av bansektionernas pallregister bildades ett datablock *Trackunit PalletLogs*. Inuti datablocket skapades ett register för varje sektion med UDT:n *PalletLog*. Utgången *PalletLog* från bansektionerna knöts sedan till tillhörande register i datablocket.

4.12.2 Åskådliggörande i HMI

Inledningsvis utformades grunden av det som behövdes i HMI:et för visningen av pallregister. Grunden bestod av en tabell med tio stycken pallinlägg, en drop-down meny för att välja register, en knapp för att bläddra fem inlägg bakåt i registret samt en för att bläddra fem framåt. Tanken var att det skulle skapas en HMI-tagg som innehöll en vektor med de tio pallinlägg som visas i tabellen. Därefter skulle det skrivas ett program i PLC:en där det hanteras vilka tio pallinlägg som HMI-taggens vektor pekar på.

En UDT som innehöll alla de variabler som kom att användas i HMI:et framställdes och döptes till *ViewPalletLogHMI*, för en bild på den se figur 22.

ViewPalletLogHMI

	Name	Data type	Default value	Comment
	Pallets	Array[0..9] of "PalletEntry"		HMI viewing array with pallet entries
	Forward	Bool	false	Button browse to more recent pallet entries
	Backward	Bool	false	Button browse to older pallet entries
	PalletLogSelectedByHMI	Int	0	Int used by Case statement to select which DB log to view
	Refresh	Bool	false	Button to leave browsing mode and fetch new pallet entries
	Browsing	Bool	false	In browsing mode when true
	NbrOfPagesSelectedLog	Int	0	Number of pages in the selected DB log, 10 entries per page
	ViewingPage1	Int	0	Page from DB log being viewed on the HMI
	ViewingPage2	Int	0	Page from DB log being viewed on the HMI
	PageNbrHyphen	String	''	A hyphen made visible when viewing 2 pages simultaneously
	Backslash	String	'/'	A backslash character for page displayment

Figur 22: *User-Defined Data Type ViewPalletLogHMI.*

I PLC:en skapades ett nytt funktionsblock som betecknats *Pallet Log Viewer*, på så sätt har tillämpningen för visning av pallregister i HMI:et hamnat utanför banstyrningsmodulerna. Tillämpningen har skapats separat

för möjligheten att välja bort den i fall då en överordnad databas impelementerats.

I funktionsblocket bildades en PLC-tagg med UDT:n *ViewPalletLogHMI* som struktur och döptes till *LogSelectedByHMI_1*. Taggen innehåller bland annat heltalet *PalletLogSelectedByHMI*, med detta kommuniceras det till PLC:n vilket register som valts i HMI:et. Varje register sammanlänkas med ett unikt värde på *PalletLogSelectedByHMI*, se figur 23. I funktionsblocket skapades sedan ett tomt register *SelectedLog_1*. När ett register valts i HMI:et sätts värdet på *PalletLogSelectedByHMI* och används i en case sats för att kopiera över det valda registret ur datablocket till *SelectedLog_1*.

Text lists

Name	Selection
PalletLogs	Value/Range

Text list entries

Default	Value	Text
○	1	Track 100 - Ingoing Pallets
○	2	T-Track 115
○	3	Track 110
○	4	Track 130
○	5	Track 220
○	6	Track 230
○	7	Track 140 - Outgoing Pallets Station Not A
○	8	Track 240 - Outgoing Pallets Station A
◉	Default entry	Select which pallet log to view and browse

Figur 23: *Textlistan PalletLog.*

När det valda registret kopierats över till *SelectedLog_1* sorteras vektorn med den enkla sorteringsalgoritmen bubbelsortering med avseende på löpnummer. På så sätt fås vektorn sorterad i fallande ordning där det senaste pallinlägget alltid finns på förstaplatsen i vektorn.

I taggen *LogSelectedByHMI_1* finns en boolesk variabel *Browsing*, med den uppges det ifall registret bläddras igenom av användaren. Om *Browsing* är falsk har inte bläddring påbörjats, registret betraktas då i realtid till följd av att varje cykelvarv kopieras det valda registret ur datablocket över till

SelectedLog_1. För insättning av pallinlägg från *SelectedLog_1* till vektorn *LogSelectedByHMI_1.Pallets* med de tio inlägg som ska visas i HMI:et används *Index_1*. När *Browsing* är falsk sätts alltid *Index_1* till nio, det görs för att de tio senaste inläggen ur registret ska kopieras över till vektorn *LogSelectedByHMI_1.Pallets*.

Knapparna i HMI:et för att bläddra igenom det valda registret har knutits till de booleska variablerna *Forward* och *Backward* i PLC-taggen *LogSelectedByHMI_1*. Dessa variabler övervakas för en signalförändring från falsk till sann, när signalförändringen detekteras så hanteras bläddringen i PLC-programmet.

När knappen för att bläddra bakåt trycks in sätts *Browsing* till sann, registret betraktas då inte längre i realtid utan innehåller de inlägg som fanns vid den tidpunkt då bläddring påbörjades. Heltalet *Index_1* ökas med fem och sedan kontrolleras det att värdet på *Index_1* inte överstiger registrets storlek eller pekar på ett tomt pallinlägg. För koden där knapptryck av bläddra bakåt knappen hanteras se figur 24.

Pallet Log Viewer

```
(*
Browse backwards log 1
Scan for positive signal edge on downwards arrow button on HMI screens "PalletLogs" & "TwoPalletLogs".
When the button is pressed, the viewieng array displays 5 older pallet entries.
Once the earliest entry in the logg is reached nothing happens when pressing the button.
*)
#BrowseBackwardLog_1(CLK := #LogSelectedByHMI_1.Backward,
                     Q => #PalletLogViewerTags.Back_1);
 // If (PalletEntry[0].Nbr < 10) nothing to browse backwards to.
IF #PalletLogViewerTags.Back_1 AND #SelectedLog_1.PalletEntry[0].Nbr > 10 THEN
    #LogSelectedByHMI_1.Browsing := TRUE;
    #PalletLogViewerTags.Index_1 := #PalletLogViewerTags.Index_1 + 5;
    // If (index > array size) -> index = last entry in array.
    IF #PalletLogViewerTags.Index_1 > (#PalletLogSize - 1) THEN
        #PalletLogViewerTags.Index_1 := (#PalletLogSize - 1);
    END_IF; // If Index is pointing to an empty entry decrease index until it points to the oldest entry.
    IF #SelectedLog_1.PalletEntry[#PalletLogViewerTags.Index_1].Nbr = 0 THEN
        FOR #i := 0 TO #PalletLogViewerTags.Index_1 DO
           IF #SelectedLog_1.PalletEntry[#PalletLogViewerTags.Index_1].Nbr = 0 THEN
               #PalletLogViewerTags.Index_1 := #PalletLogViewerTags.Index_1 - 1;
           END_IF;
        END_FOR;
    END_IF;
END_IF;
```

Figur 24: *Pallet Log Viewer, kod där knapptryck av bläddra bakåt i registret knappen hanteras.*

Knappen för att bläddra framåt synliggörs av att variabeln *Browsing* blivit sann, först måste det alltså bläddrats bakåt för att det ska finnas något att bläddra fram till. När knappen trycks in kontrolleras det om de senaste pallinlägget i registret finns först i vektorn *LogSelectedByHMI_1.Pallets*. Om det senaste inlägget inte finns först i vektorn minskas *Index_1* med fem och det säkerställs att värdet aldrig blir mindre än nio. Om knappen tryckts och det senaste inlägget finns först i vektorn så sätts *Browsing* till falsk och registret betraktas återigen i realtid. Det finns även en *Refresh* knapp som synliggörs när *Browsing* är sann. Med *Refresh* knappen kan användaren direkt återgå till att betrakta registret i realtid. För koden där bläddra framåt knappen hanteras se figur 25.

Pallet Log Viewer

```
117 (*
118 Browse forwards log 1
119 Scan for positive signal edge on upwards arrow button on HMI screen "PalletLogs".
120 When the button is pressed, the viewieng array displays 5 more recent pallet entries.
121 Once the most recent entry in the logg is reached and the button is pressed
122 browsing becomes false.
123 *)
124 #BrowseForwardLog_1(CLK := #LogSelectedByHMI_1.Forward,
125                     Q => #PalletLogViewerTags.Forward_1);
126 IF #PalletLogViewerTags.Forward_1 THEN
127     // If the newest log entry isn't displayed in the HMI viewing array.
128     IF #SelectedLog_1.PalletEntry[0].Nbr <> #LogSelectedByHMI_1.Pallets[0].Nbr THEN
129         #PalletLogViewerTags.Index_1 := #PalletLogViewerTags.Index_1 - 5;
130         // Index = 9 --> 10 newest pallet entries displayed in the HMI viewing array.
131         IF #PalletLogViewerTags.Index_1 < 9 THEN
132             #PalletLogViewerTags.Index_1 := 9;
133         END_IF;
134     // Newest log entry is displayed in the HMI viewing array, can't browse further forward.
135     ELSIF #SelectedLog_1.PalletEntry[0].Nbr = #LogSelectedByHMI_1.Pallets[0].Nbr THEN
136         #LogSelectedByHMI_1.Browsing := FALSE;
137         RETURN;
138     END_IF;
139 END_IF;
```

Figur 25: *Pallet Log Viewer, kod där knapptryck av bläddra framåt i registret knappen hanteras.*

Då värdet på *Index_1* fastställts (så) avgörs det vilka pallinlägg ur *Selected-Log_1* som vektorn *LogSelectedByHMI_1.Pallets* ska peka på genom koden i figur 26.

Pallet Log Viewer

```
141 (*
142 Insert pallet entries into the HMI viewing array LogSelectedByHMI_1.Pallets.
143 The 10 pallet entries displayed from the SelectedLog_1 is decided by the Index_1 value.
144 *)
145 FOR #i := #PalletLogViewerTags.Index_1 TO (#PalletLogViewerTags.Index_1 - 9) BY -1 DO
146     #LogSelectedByHMI_1.Pallets[#i - (#PalletLogViewerTags.Index_1 - 9)] := #SelectedLog_1.PalletEntry[#i];
147 END_FOR;
```

Figur 26: *Pallet Log Viewer, kod för att ange de pallinlägg som ska visas i HMI:et.*

Antalet sidor i registret som kopierats över till *SelectedLog_1* beräknas och därigenom klargörs det hur många pallinlägg det finns att bläddra igenom. Eftersom det i HMI:et visas tio inlägg fastställdes det att tio inlägg utgör en sida. För koden där antalet sidor i det valda registret beräknas se figur 27.

Pallet Log Viewer

```
72  // Calculate the number of pages in the SelectedLog_1, 10 entries per page.
73 IF #SelectedLog_1.PalletEntry[0].Nbr = 0 THEN // The DB log is empty --> 0 pages.
74     #LogSelectedByHMI_1.NbrOfPagesSelectedLog := 0;
75 ELSIF #SelectedLog_1.PalletEntry[0].Nbr < 11 THEN // (Entries < 11) in the DB log --> 1 page.
76     #LogSelectedByHMI_1.NbrOfPagesSelectedLog := 1;
77 ELSIF #SelectedLog_1.PalletEntry[0].Nbr >= #PalletLogSize THEN //Calculates the number of pages when the DB log is full.
78     #LogSelectedByHMI_1.NbrOfPagesSelectedLog := (#PalletLogSize / 10);
79     IF (#PalletLogSize MOD 10) > 0 THEN
80         #LogSelectedByHMI_1.NbrOfPagesSelectedLog := #LogSelectedByHMI_1.NbrOfPagesSelectedLog + 1;
81     END_IF;
82 ELSE  // Calculates the number of pages when (10 < entries < #PalletLogArraySize).
83     #LogSelectedByHMI_1.NbrOfPagesSelectedLog := (#SelectedLog_1.PalletEntry[0].Nbr / 10);
84     IF (#SelectedLog_1.PalletEntry[0].Nbr MOD 10) > 0 THEN
85         #LogSelectedByHMI_1.NbrOfPagesSelectedLog := #LogSelectedByHMI_1.NbrOfPagesSelectedLog + 1;
86     END_IF;
87 END_IF;
```

Figur 27: *Pallet Log Viewer, kod för att beräkna antalet sidor i det valda pallregistret.*

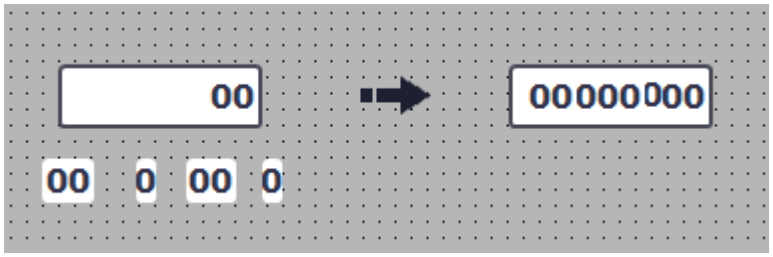
För att göra användaren medveten om var i registret den befinner sig beräknas det vilken eller vilka sidor ur registret som visas i HMI:et. I de fall då pallinlägg från två olika sidor visas så synliggörs båda sidnummer med ett bindestreck emellan. För koden där sidnummer som visas i HMI:et beräknas se figur 28.

Pallet Log Viewer

```
149  // Calculate which page numbers from SelectedLog_1 currently being viewed in the HMI.
150 IF #LogSelectedByHMI_1.NbrOfPagesSelectedLog = 0 THEN // The DB log is empty -> 0 pages.
151     #LogSelectedByHMI_1.ViewingPage1 := 0;
152     #LogSelectedByHMI_1.ViewingPage2 := 0;
153     #LogSelectedByHMI_1.PageNbrHyphen := '';
154 // (Entries < 11) in the DB log --> 1 page.
155 ELSIF #LogSelectedByHMI_1.NbrOfPagesSelectedLog = 1 THEN
156     #LogSelectedByHMI_1.ViewingPage1 := 1;
157     #LogSelectedByHMI_1.ViewingPage2 := 0;
158     #LogSelectedByHMI_1.PageNbrHyphen := '';
159 //Calculates the viewed page number when the DB log has more than one page.
160 ELSIF #LogSelectedByHMI_1.NbrOfPagesSelectedLog > 1 THEN
161     // If entry[0] has (pallet nbr % 10) = 0 the page nbr viewed is (pallet nbr / 10).
162     IF (#LogSelectedByHMI_1.Pallets[0].Nbr MOD 10) = 0 THEN
163         #LogSelectedByHMI_1.ViewingPage1 := (#LogSelectedByHMI_1.Pallets[0].Nbr / 10);
164         #LogSelectedByHMI_1.ViewingPage2 := 0;
165         #LogSelectedByHMI_1.PageNbrHyphen := '';
166     ELSIF (#LogSelectedByHMI_1.Pallets[0].Nbr MOD 10) <> 0 THEN
167         #LogSelectedByHMI_1.ViewingPage2 := (#LogSelectedByHMI_1.Pallets[0].Nbr / 10);
168         #LogSelectedByHMI_1.ViewingPage1 := #LogSelectedByHMI_1.ViewingPage2 + 1;
169         #LogSelectedByHMI_1.PageNbrHyphen := '-';
170     END_IF;
171 END_IF;
```

Figur 28: *Pallet Log Viewer, kod för att beräkna sidnummer ur pallregistret som visas i HMI:et.*

Vid visning av sidnummer i HMI:et används fem variabler, dessa finns att se längst ner i figur 22. När variablerna ska åskådliggöras i HMI:et krävs det fem separata I/O fält. För att de ska presenteras enhetligt och se ut som ett I/O fält så har ett huvudfält med kantlinje skapats. Huvudfältet innehåller variabeln *NbrOfPagesSelecredLog* och dess tecken har placerats längst åt höger. Resterande fyra I/O fält har skapats utan kantlinje och minskats i storlek, sedan har det placerats ovanpå huvudfältet, för en bild på detta se figur 29. Variablerna *ViewingPage2* och *PageNbrHyphen* synliggörs enbart när två olika sidor ur register betraktas.



Figur 29: *Fem separata I/O fält sammansatta till att se ut som ett.*

I slutet av programmet tilldelas heltalsvariabeln *PreviouslyViewed_1* värdet på *PalletLogSelectedByHMI*, d.v.s vilket register som betraktades under det gångna cykelvarvet. När programmet sedan körs i nästa cykelvarv jämförs de två heltalsvariablerna med varandra, på så sätt undersöks det om registret som betraktades i det föregående cykelvarvet är ett annat än i det nuvarande. Om användaren har bytt register töms vektorn *LogSelectedByHMI_1.Pallets* innan den fylls med nya pallinlägg och *Browsing* sätts till falsk.

När tillämpningen för att åskådliggöra och bläddra igenom ett pallregister i HMI:et var färdigt beslutades det att skapa en till skärm där två register kan betraktas samtidigt. De nödvändiga variablerna för att betrakta ännu ett register introducerades och sedan kunde den befintliga koden kopieras och återanvändas. För skärmen med visualisering av två register kopierades den som skapats för att visualisera en. Tabellen med inlägg fick modifieras lite för att två stycken tabeller skulle få plats.

4.13 Fas 13: Metadata

I det här avsnittet redovisas tillvägagångssättet för insamling, behandling och presentation av metadatan.

4.13.1 Insamling

I detta delkapitlet förklaras hur insamlingen av metadatan gjordes.

Det skapades först två UDT:er för insamlingen av metadata, *AllTracks-ModuleData* och *MotorData*. Där metadatan för en bansektion samlades in i *AllTracksMoudleData* och metadatan för en motormodul samlades

in i *MotorData*. UDT:erna lades som en utgång i datainstansen för varje banmodul respektive motormodul. *Motor Data* skickades från utgången till bansektionens *AllTracksModuleData*. Det valdes att all tänkbar metadata samlades in fast enbart viss information visades. Se figur 29 för UDT:n *AllTracksModuleData* för all insamlad metadata.

	Name	Data type	Comment
1	NbrOfPallets	Int	Number of pallets that has been on the track
2	NbrOn/Off_Photcell	Int	Number of times the photocell has been switched on/off
3	NbrOfIncomingPalletAlarms	Int	Number of incoming pallet lost/stuck alarms
4	NbrOfOutgoingPalletAlarms	Int	Number of outgoing pallet lost/stuck alarms
5	NbrOn/Off_T-out	Int	Number of times T-out has been switched on/off
6	NbrOn/Off_T-in	Int	Number of times T-in has been switched on/off
7	AveragePalletTimeOnTrack	DInt	Average time pallets spend on the trackunit in milliseconds
8	PalletsToA	Int	Number of pallets to station A
9	PalletsNotToA	Int	Number of pallets to not station A
10	MotorData	"MotorData"	Tags for logging motor data
11	NbrOfOn/OffSwitches	Int	Number of times the motor has been turned on/off
12	NbrOfFuseAlarms	Int	Number of fuse alarms that's occured
13	NbrOfCBAlarms	Int	Number of CB alarms that's occured
14	Maintenance	Bool	Maintenance recommended. Used to switch between displaying time unil/since maintenance
15	TotalRuntime	String	Total runtime of the motor as Hours Minutes Seconds
16	TimeUntil/SinceMaintenance	String	Time until/since a maintenance is/was recommended

Figur 30: *User-Defined Data Type AllTracksModuleData*

Insamling för heltal började med att en önskad variabel triggades på en positiv flank. Då ökades en variabel med ett, i en av det tidigare nämnda UDT:erna som svarade för statistiken av just den variabeln. Variablerna av typen av de ovannämnda UDT:erna lagrades i datablocket *Trackunit Datalogs*.

För statistik gällande tidsvariabler startades en timer när en önskad signal blev aktiv. Timern returnerade en variabel av datatypen time som konverterades till ett heltal som i sin tur konverterades till en sträng. Detta gjordes för sekunder, minuter samt timmar. Strängarna lades in i en vektor av strängar som därefter konverterades ihop till en sträng. Därefter upprepades processen för lagringen som för insamling utav heltal. Om manuellt läge var till så räknades motorns körtid endast av tidsvariablerna.

Det valdes också att ta med sektionens nuvarande status på fotocellen och pallen samt nuvarande pall-ID som kopierades över till en egen UDT, *TrackunitStatus*. Därefter lades det sedan in i datablocket *Trackunit Datalogs* från

banmodulen.

4.13.2 Behandling

I det här delavsnittet beskrivs behandlingen av den insamlade metadatan.

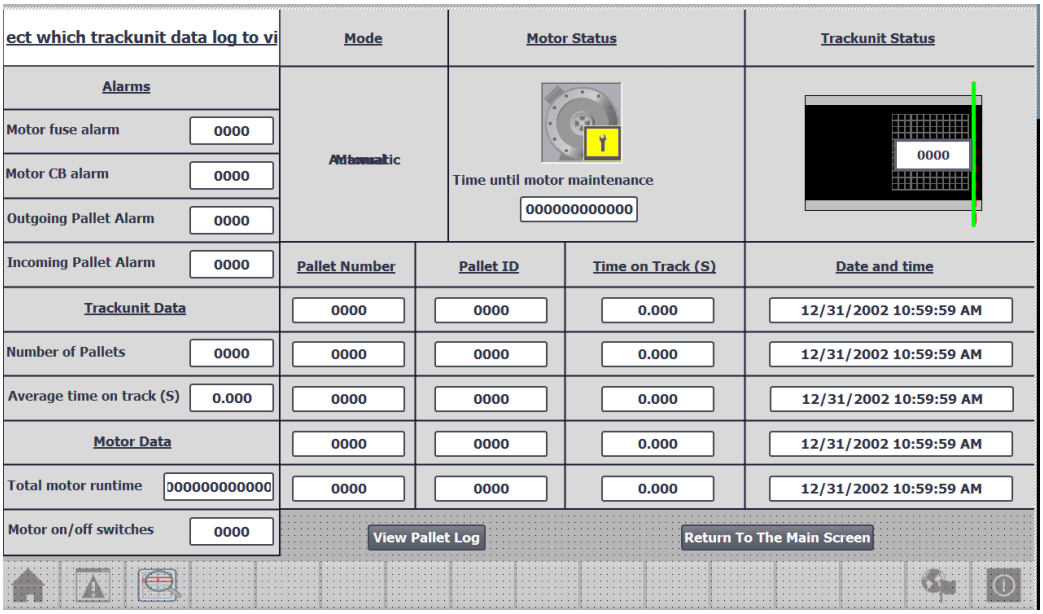
Ett nytt funktionsblock, *Trackunit Data Viewer*, skapades för att ta hand om visningen från PLC:n till HMI:et. På grund av att det endast visades en sektion åt gången skapades endast en HMI-tagg av typen *AllTracksModuleData*. Funktionsblocket bestod endast av case-satser där det avgjordes vilken sektions data som skickades från datablocket *Trackunit Datalogs* till HMI-taggen.

4.13.3 Presentation

I detta delkapitel förklaras hur metadatan valdes att presenteras.

Från huvudskärmen valdes det vilken sektions metadata som visades genom att trycka på knappen *i*. När knappen trycktes sattes HMI-taggen *DataLogSelectedByHMI* till ett unikt värde för varje sektion. Taggen kopplades till den tidigare nämnda case-satsen i *Tackunit Data Viewer*.

För visning av metadatan skapades en ny skärm, *TrackData*. Den bestod av en metadata-del, en status-del och en kontinuerlig uppdatering av de fem senaste pallarna i from av ett pallregister enligt figur 31.



Figur 31: *TrackData, skärmen som visar status och metadata för en sektion*

För metadatan skapades fyra olika faceplates, en för varje sorts bansektion då metadatan som samlades in var olika. Det gjordes som en tabell där värdet kopplades till ett I/O field och med ett beskrivande textfält. I och med att det var fler än en gjordes en visibility funktions koppling med taggen *DataLogSelectedByHMI* så bara en sort visades åt gången. Motorstatusen hämtades från den redan gjorda faceplaten *FaceplateMotor*. Underhållsstatusen för motorn presenterades som en varning i *Alarm view* och som en underhållssymbol i motorns faceplate. Lägesstatusen kopplades bara till den redan använda HMI-taggen *Mode* och det lades till en visibility funktion med ett textfält med motsvarande text utav värdet. Pallregistret kopierades från *PalletLogs*-skärmen men med endast kontinuerlig uppdatering av de senaste fem inkomna pallarna. Det skapades en faceplate för *Track Status* som byggdes på den redan tidigare gjorda faceplaten *FaceplateTrackModule*. Det som skilde sig var att enbart informationen från UDT:n *Trackunit Status* togs med. Slutligen gjordes två navigationsknappar som navigerade användare till önskad skärm.

5 Resultat

I detta kapitel redovisas resultaten av faserna som introducerats i metodkapitlet 3.1 och vars genomförande har skildrats i kapitel 4. Fas 6 och 9 omfattar två typer av simulering och har därför inga fristående resultat, de har använts för att uppnå resultaten i faserna 4, 5, 7, 8, 10, 11, 12 och 13.

5.1 Fas 1: Förberedande arbete

Resultatet av *Fas 1: Förberedande arbete* blev den digitala arbetsmiljö som examensarbetet kom att utföras i. Den digitala arbetsmiljön omfattar två virtuella maskiner som är sammankopplade via ett LAN-nätverk. Båda de virtuella maskinerna innehöll TIA Portalen V14 och de nödvändiga licenserna för STEP 7 och WinCC.

I delkapitlet 4.1 där genomförandet av fas 1 beskrivs det hur en Multiuser-server implementerades och ett Multiuser-projekt skapades på servern. Examensarbetet utfördes till en början i ett Multiuser-projekt fram tills simulering påbörjades. Det upptäcktes då att prestandan påverkades negativt och programmet arbetade oerhört mycket långsammare än vid simulering i ett vanligt projekt. Då arbetet utfördes gemensamt på ett kontor sågs inte längre något syfte i att fortsätta arbeta i ett Multiuser-projekt. Det redan utförda arbetet kopierades över till ett vanligt projekt i TIA Portalen som examensarbetet sedan fortlöpte i.

5.2 Fas 2: Programstruktur

Här presenteras resultatet av *Fas 2: Programstruktur* implementerades under programmeringen i de resterande faserna. Se figur 3 i avsnitt 4.2 för en bild över programstrukturen.

Implementationen av denna struktur med *Multi Instance* datablock resulterade i ett organiserat och lättnavigerat program. Det innebär att felsökning och åtgärdande av eventuella fel kunde göras enkelt och fort. Genom programmets modulariserade struktur var det även enkelt att göra ändringar och tillägg.

Utöver de datablock som innehåller larm, register och metadata har enbart ett *Single Instance* datablock skapats. Detta datablock heter *Common Tracks_DB* och genererades när funktionsblocket *Common Tracks* som utgör hela bansystemet anropades i huvudprogrammet *OB1*. I detta *Single Instance* datablocket hittar man alla *Multi Instance* datablock för anrop av bansektionerna samt deras anrop av moduler. För att få en bild över detta se figur 32.

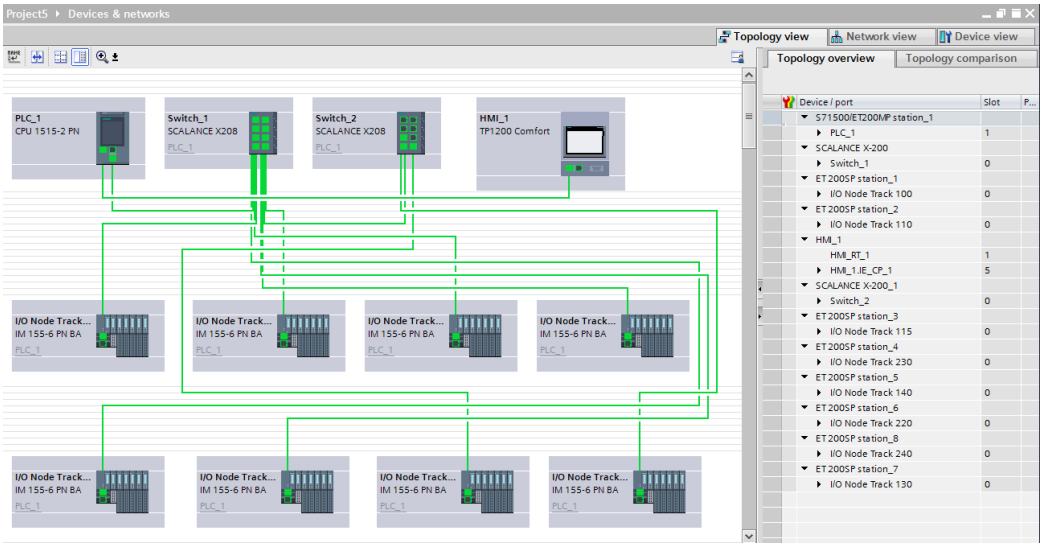
Common Tracks_DB

	Name	Data type	Comment
1	Input		
2	Output		
3	InOut		
4	Static		
5	RTOButtonAppearance	Int	RTO button made blue by modules
6	Track 110_Instance	"Track 110"	
7	Input		
8	Output		
9	InOut		
10	Static		
11	Track Module_Instance	"Track Module"	DB for the call instance of the FB "Track Module"
12	Motor Module_Instance	"Motor Module"	DB for the call instance of the FB "Motor Module"
13	Track 220_Instance	"Track 220"	
14	Track 100_Instance	"Track 100"	
15	Input		
16	Output		
17	InOut		
18	Static		
19	First Track Module_Instance	"First Track Module"	DB for the call instance of the FB "First Track Module"
20	Motor Module_Instance	"Motor Module"	DB for the call instance of the FB "Motor Module"
21	Track 130_Instance	"Track 130"	
22	Track 140_Instance	"Track 140"	
23	T Track 115_Instance	"T Track 115"	
24	Input		
25	Output		
26	InOut		
27	Static		
28	T Track Module_Instance	"T Track Module"	DB for the call instance of the FB "T Track Module"
29	Motor Module_Instance	"Motor Module"	DB for the call instance of the FB "Motor Module"
30	Track 230_Instance	"Track 230"	
31	Track 240_Instance	"Track 240"	
32	Input		
33	Output		
34	InOut		
35	Static		
36	Last Track Module_Instance	"Last Track Module"	DB for the call instance of the FB "Last Track Module"
37	Motor Module_Instance	"Motor Module"	DB for the call instance of the FB "Motor Module"
38	Pallet Log Viewer_Instance	"Pallet Log Viewer"	
39	Trackunit Data Viewer_Instance	"Trackunit Data Viewer"	

Figur 32: *Common Tracks_DB med instansdatablock för alla bansektioners samt instansdatablocken för bansektioners anrop av moduler.*

5.3 Fas 3: Virtuell hårdvara

Resultatet av *Fas 3: Virtuell hårdvara* är ett PROFINET nätverk med virtuell hårdvara. Hårdvaran består av en 1515-2 soft PLC, en TP 1200 Comfort panel, åtta stycken ET200SP distribuerade I/O noder och två stycken SCALANCE X208 switchar. Nätverket avbildar på så sätt en fysisk anläggning med åtta bansektioner där enbart ethernet kablar dras fram till sektionernas I/O noder. För att få en helhetsbild av nätverket med den virtuella hårdvaran se figur 33.



Figur 33: *Topologisk vy av den virtuella hårdvaran*

5.4 Fas 4-5: Programmering av Moduler och signalgränssnitt

I detta delkapitel redovisas resultatet av faserna *Fas 4: Programmering av Moduler* och *Fas 5: Signalgränsnitt mellan bansektioner*. Ett av resultatet är de fyra nedanstående banstyrningsmodulerna.

- *First Track Module* - Imatningsstation
- *Track Module* - Mellanliggande bansektion
- *T Track Module* - T-bansektion
- *Last Track Module* - Utmatningsstation

Banstyrningsmodulerna kan i ett program anropas hur många gånger som helst och därav teoretiskt sätt användas för att skapa oändligt stora bansystem. Modulerna kan återanvändas i andra program för andra anläggningar. Modulen *Track Module* som utnyttjats som en ram för att utveckla resterande banstyrningsmoduler finns bifogad i *Appendix 1*.

De fyra UDT:erna *InputsPrevious*, *InputsNext*, *OutputsPrevious* och *OutputsNext* är det resulterande signalgränssnittet. De har medfört att de olika anropen av banstyrningsmodulerna kan utföra handskakningar med varandra och överlämna pallar.

Slutligen har en modul för motorstyrning tillverkats. Även denna kan återanvändas i ett program genom flertalet anrop samt användas i andra program. Modulen *Motor Module* för motorstyrning finns bifogad i *Appendix 2*.

5.5 Fas 7-8: Gränssnitt från Modul till HMI och Faceplates

I det här avsnitt presenteras resultatet från *Fas 7: Gränssnitt från Modul till HMI* och *Fas 8: Faceplates*.

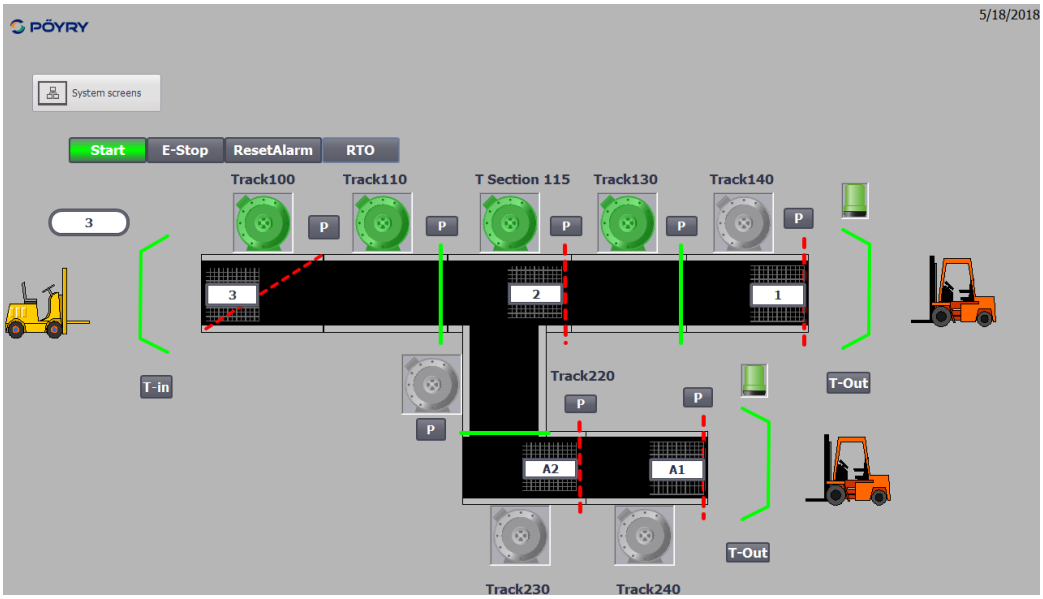
Resultatet från det första steget av HMI simuleringen blev framförallt att kontrollera att kopplingarna var korrekt gjorda samt att det skapades en

helhetsbild över systemet i realtid. Till att börja med testades enkla taggar och faceplates för att komma in på rätt tankebanor. Successivt simulerades fler och fler funktioner tills en helhetsbild hade framtagits. Resultatet blev att systemet beskrevs i en sorts stegvis simulering då det inte blev en helt jämn förflyttning av pallarna. Resultatet av pallförflyttningen blev då att en pall hade ett läge per sektion. I det här systemet prioriterades funktionalitet och inte en fräck grafisk display.

I och med att allting gjordes i faceplates så upptäcktes det att felmarginalen samt att lokalisera var eventuella fel uppstod minskades drastiskt. Ifall ett fel uppstod så syntes det i alla sektioner av den typen, eller så var det kopplingen som var kopplad till en annan sektion. I vilket fall så märktes felet tydligt.

Resultatet från ändring av programspråket FBD till SCL underlättade också felsökningen för då sågs det lättare om det var fel med uppkopplingen till HMI eller själva koden. Det upplevdes också som smidigare för koden blev betydligt mer kompakt framförallt för for-satser. Koden blev också kompakt tack vare att allt som skulle kopplas till HMI samlades i en UDT för varje sektion.

Den stora resultatskillnaden från simulering med watchtable jämfört med simulering i HMI var att det blev mer lättöverskådligt samt lättare att se processen. Det gavs också en helhetsbild som inte watchtable kunde leverera. Dock så krävdes det en uppkoppling till ett annat system vilket också ökade risken att det kunde uppstå ett fel. Som tidigare nämnts så var det enkelt att lokalisera var det eventuella felet dök upp någonstans samt att fixa felet i och med programstrukturen. Se figur 34 för resultat av simulering i HMI.



Figur 34: *Resultat av simulering i HMI.*

5.6 Fas 10: Manuellt läge

I detta avsnitt skildras resultatet av *Fas 10: Manuellt läge*.

Huvudsakliga resultatet som gavs från att implementera ett manuellt läge blev att det kunde skapas en pall var som helst i systemet. Eftersom det tidigare bestämdes att om det inte fanns en pall så var pall-ID:et en tom sträng. Detta användes nu till att definiera för manuellt läge om det fanns en pall eller inte. Genom ett gemensamt beslut så bestämdes det att en pall måste ha ett pall-ID. Skrivs det in ett pall-ID i manuellt läge så skapas då en pall på den sektion där pall-ID:et skrevs in. En pall utan ID kunde innan köras in från inmatnings stationen men den funktionen togs nu bort eftersom det ansågs att om en pall inte har ett pall-ID så är det en oönskad pall eller ett oönskat föremål som har kommit in på bansektionerna.

I manuellt läge upptäcktes även att det gick betydlig mycket snabbare att simulera programmet, då en pall kunde skapas var som helst i systemet. Detta underlättade på så vis att det gick betydligt snabbare att testa en

specifik bana.

Resultatet från implementeringarna av de semi-automatiska säkerhetsfunktionerna blev att det skapades en mer verklighetstrogen miljö, än att låta allting vara i helt manuellt läge. Ifall motorn startades i manuellt läge och fotocellen slogs till så stoppades motorn då det skulle simulera ifall ett oönskat föremål kom in på bana som kunde skada maskinen eller operatören.

5.7 Fas 11: Larmhantering

I det här avsnittet redovisas resultat av *Fas 11: Larmhantering*.

När ett larm uppstod så dök det upp ett extrafönster som visade larmet så länge det var aktivt. Det resulterade i att det krävdes en kvittering från HMI-skärmen för att larmet skulle försvinna från extrafönstret. Dock om RTO knappen trycktes ner så kunde fortfarande bansektionerna köras. Så extrafönstret ger endast resultatet av ett larm och är inte nödvändig för uppstart av maskinen igen. För varje larm fanns det möjligheten att få upp en informations ruta för information kring larmet och vad som krävdes för uppstart. Det fanns också en larmhistorik skärm där alla larm och dess olika statusar visades i kronologisk ordning. Dessutom sparades alla larm och dess olika statusar som en TXT fil på datorns hårddisk. Det resulterade i att det blev enkelt att spåra ett larm samt att lokalisera var och vad som är fel.

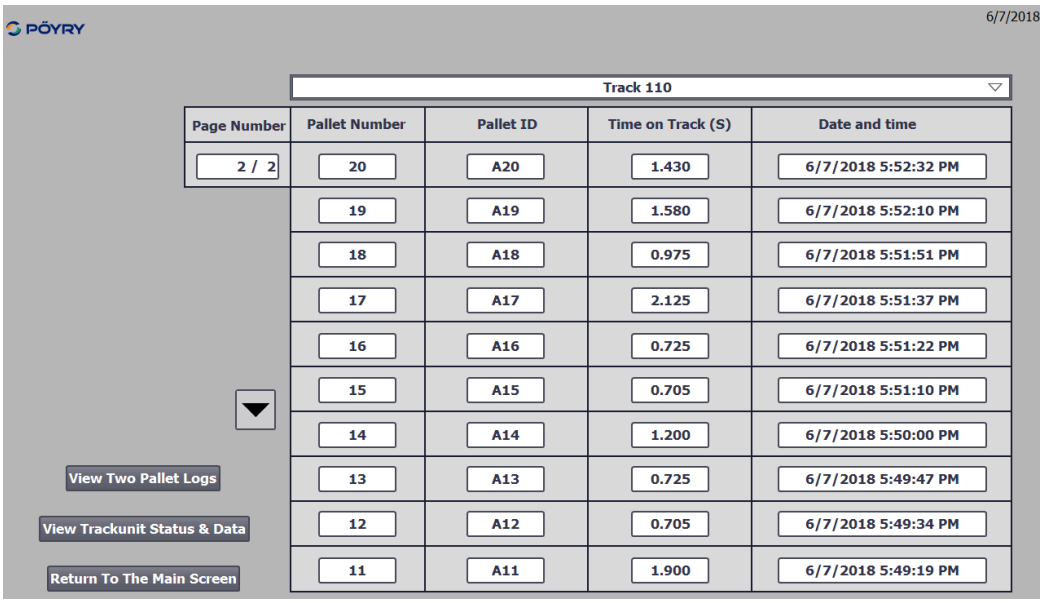
Resultat som gavs från larmhantering var att det upptäcktes hur mångsidigt och lätthanterlig det var att implementera larm i TIA portalen samt att visa dem i HMI. Extrafönstret och larmskärmen var färdigprogrammerade inuti WinCC. Det enda som krävdes av programmeraren var att kryssa i rutorna för vad för typ av larm som skulle visas.

Skillnaden mellan att använda det gamla sättet mot att använda det nya blev att det tog mindre tid att koda ett larm med det nya sättet. Rent resultatmässigt blev det ingen skillnad.

5.8 Fas 12: Pallregister

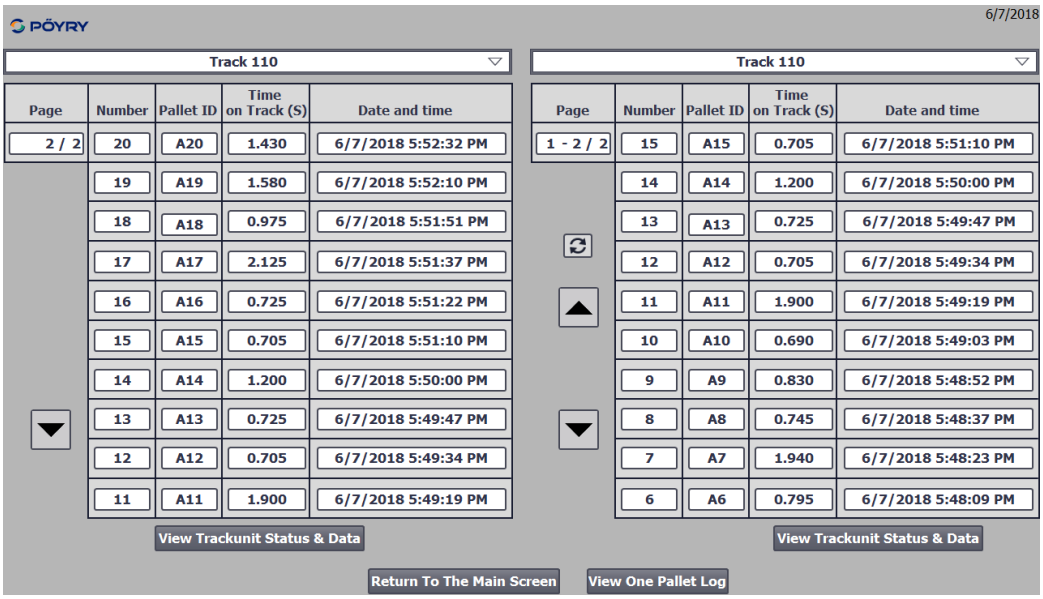
Här presenteras resultatet av *Fas 12: Pallregister*. Inuti banstyrningsmodulerna integrerades FBD kod så att ett pallregister upprättas för varje bansektion. Register består av information kring varenda pall som befunnit sig på sektionen och lagras i ett datablock *PalletLogs*. Ett funktionsblock *Pallet Log Viewer* har bildats som hanterar visning av pallregister i HMI:et. Funktionsblocket *Pallet Log Viewer* finns bifogad i *Appendix 3*.

I HMI:et har två skärmar för att granska pallregister skapats. Den ena skärmen *PalletLogs* består av en stor tabell där en bansektions register kan betraktas i realtid samt historik. Se figur 35.



Figur 35: *HMI-skärm PalletLogs*

Den andra skärmen *TwoPalletLogs* består av två mindre tabeller där två register kan betraktas. Detta medför att två olika sektioners register kan granskas samtidigt eller att ett register kan betraktas både i realtid och historik samtidigt. Se figur 36.



Figur 36: *HMI-skärm TwoPalletLogs*

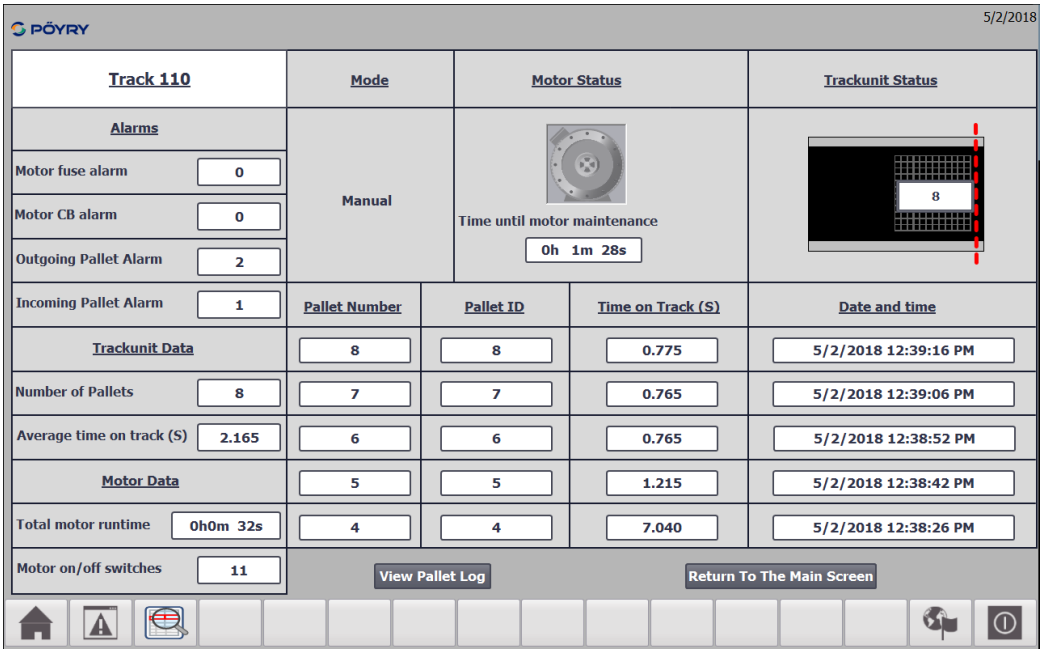
5.9 Fas 13: Metadata

I detta avsnitt redovisas resultat som gavs av *Fas 13: Metadata*.

Resultatet från insamlingen av metadatan blev att all tänkbar information samlades in. Alltihop samlades i en UDT, *AllTrackModuleDataHMI*, som sedan skickades till HMI:et. Allting samlades in för det skulle avspegla ett mångsidigt verklighetstroget system. Därefter valdes vad som skulle visas men alla värden var fortfarande sparade ifall en specifik kund skulle vilja ha info om t.ex antal påslag från fotocellerna som här ansågs inte viktigt. Resultatet från insamlingen blev lite mager då det inte fanns med en specifik typ av motor. Därför valdes det att ta med statusen från motorn och bansektion och för att det passade detta examensarbetet bättre.

Resultatet från behandlingen blev att allting samlades in i en tagg som tidigare nämnt fast till faceplaten så valdes bara de önskade värden som kopplades ihop med ett I/O fält. De andra värdena sparades och kan fortfarande hämtas men visades inte i HMI:et.

Resultatet från presentationen blev att det endast fanns en skärm och för att rätt sektion skulle visas så satte taggen *DataLogSelectedByHMI* som var kopplad till en grafisk lista. När *i* knappen på en sektion trycktes ner så sattes den till motsvarande värde i den grafiska listan och rätt sektion visades. Slutresultat blev att metadataskärmen gav en mer helhetsbild än vad som tänkt från början då från början var det endast tänkt som en statisk sida fast nu blev det både statisk och status. Se figur 37 för resultatet av presenterad metadatainsamling i HMI:et.



Figur 37: *Resultat av insamlad metadata för Track 110.*

6 Slutsats

I detta avsnitt återges först en sammanfattning av resultatet sedan en diskussion kring examensarbetets på förhand förväntade resultat och det slutgiltiga resultatet. Därefter presenteras en slutsats av problemformuleringen, en reflektion kring etiska aspekter samt en kring framtida utvecklingsmöjligheter.

6.1 Sammanfattning av resultat

I det här delkapitlet ges en sammanfattning av resultatet som skildrats i kapitel 5.

I examensarbetet har fyra stycken banstyrningsmoduler som hanterar styrning av olika typer av bansektioner på ett säkert sätt genererats. Banstyrningsmodulerna omfattar automatisk samt manuell drift, gränssnitt till HMI, larmhantering, etablering av pall- och metadataregister.

En motorstyrningsmodul som hanterar styrning av en motor på ett säkert sätt har genererats. Motorstyrningsmodulen omfattar automatisk samt manuell drift, gränssnitt till HMI, larmhantering och registrering av motordata.

Modulerna kan återanvändas i andra TIA Portal program.

Med modulerna har ett bansystem med åtta bansektioner bildats. Bansystemet kan snabbt och enkelt expanderas med fler sektioner genom att anropa modulerna fler gånger. Programmet har fått en lättnavigerad och överskådlig programstruktur som gör det lätt att felsöka och modifiera programmet. Virtuell hårdvara i form av distribuerade I/O noder för var bansektion är också en del av programmet.

Ett HMI har konstruerats som visualiserar och styr bansystemet i realtid. HMI:et har framställts genom att skapa faceplates som även dem kan återanvändas inom andra TIA Portal program. Genom HMI:et kan man granska historisk data i form av larmhistorik, pallregister samt processens metadata.

Pallregister och metadata lagras i separata datablock. Larmhistoriken lagras i en TXT fil. Historik och data har lagrats på detta sätt för att underlätta

en framtida implementation av en överordnad databas.

6.2 Diskussion

Här diskuteras hur programmet förväntades att bli från grundtankarna till det faktiska resultatet med anknytningar till hur målen uppnåddes.

Grundidén för programmet, med undantag för metadatan, fullföljdes och målen 1-4 uppnåddes. Funktionaliteten för programmet blev mer djupgående och avancerad än vad som var förväntat. Det var inte tänkt från början att ha med en T-sektion fast i och med målet *Testa skalbarheten genom att på minimal tid expandera anläggningen* så gjordes en T-sektion och även två mellanliggande och en ut sektion med tillhörande motorer. Resultatet blev att det endast tog två timmar att programmera sektionerna, tack vare att allting hade gjorts i modultänk. Det resulterade i att systemet blev mer verklighetstroget men också att det lades ner mer tid än vad som var planerat och metadata delen hann inte göras enligt planeringen.

För metadata-delen var den ursprungliga idén att programmet skulle sammankopplas med ett överordnad databassystem. Tanken var att den överordnade databasen skulle hämta metadatan och därifrån presenteras. Eftersom metadata-delen gjordes sist så upptäckes det att tiden inte hade räckt till för att sammankoppla programmet med ett överordnat databassystem. Då gjordes undersökningar för hur metadatan kunde samlas in och presenteras inom TIA Portalen. Det fanns möjligheten att presentera metadatan på en egengjord hemsida. TIA portalen har färdigprogrammerade tillvägagångssätt för skapandet av hemsidan och det var endast att bestämma layouten och välja innehållet. Problemet var dock att det krävdes en fysisk PLC. Som tidigare nämnts så innehåller examensarbetet endast hantering av virtuell hårdvara och eftersom tidsmarginalerna var för små för att koppla upp ett fysiskt system så eliminerades detta alternativ. Det enda rimliga alternativ som kvarstod för att målen 5-7 skulle uppnås var att lagra metadatan i ett datablock och presentera metadatan i HMI:et. Genom att alternativet fullföljdes uppnåddes fortfarande målen kring metadatan fast inte på ett helt optimalt tillvägagångssätt. Lagring och presentation av pallregister omfattar enbart den senaste tiden och inte en komplett historik vilket hade uppnåtts med en överordnad databas.

6.3 Slutsats av problemformulering

Här presenteras vad för slutsatser som har tagits fram från de grundläggande problem som examensarbetet skulle lösa.

- Hur ska man styra ett bansystem på ett säkert och effektivt sätt för en tänkt fabrik utefter förbestämda förutsättningar?

  Processen styrs på ett säkert sätt genom att motorerna förreglas så att de enbart körs under säkra omständigheter samt genom implementationen av RTO vid av uppstart efter utlöst larm.

- På vilket sätt väljer man att utföra kodningen så man på ett enkelt och smidigt sätt kan utöka anläggningen?

  Genom att kodningen utfördes i moduler kunde anläggningen enkelt och smidigt utökas genom att återanvända modulerna.

- Hur kan man följa processen, både i realtid och via historik?

  HMI:et har en huvudskärm där processen visualiseras i realtid och det har skapats skärmar där processens historiska data kan granskas.

- Hur ska man kunna upptäcka fel som håller på att ske?

  Genom att analysera den insamlade metadatan. Larm av varningstyp implementeras för att informera via HMI:n att fel är på väg att uppstå.

- Vilken metadata kan man samla från processen?

  För bansektioner; Antal till/frånslag på fotocellen och signalslingan, antal pallar och genomsnittlig tid för pall på sektionen.

  Rörande pallar; Id, löpnummer, tid på sektion samt ankomst- tid och datum.

  Angående motorer; Körtid, tid tills/sedan underhåll rekommenderas och antal till/frånslag.

  Gällande larm; Antal och tidpunkt.

Metadata för varje enskild bansektion, pallar, motorer och larm har insamlats.

- Vad kan denna metadata användas till?

  Metadatan kan användas till att upptäcka och förhindra fel innan de uppstår, samt till att optimera systemets effektivitet.

- Hur kan man presentera metadatan på ett systematiskt och lättöverskådligt sätt?

  Metadata presenteras sektionsvis genom att trycka på en sektions info knapp på HMI:ets huvudskärm. Presentation av metadata görs i överskådliga tabeller på en separat skärm.

6.4 Reflektion över etiska aspekter

Angående reflektioner över etiska aspekter har det valts att reflektera kring datalagring och datainsamling, därför att det kändes mest relevant för examensarbetet.

Det gäller framförallt metadatan för mindre företag. Då metadatan måste samlas in och lagras någonstans så kommer det bli för kostsamt för mindre företag att göra det själva. Tillverkarna av styrsystem kommer då att erbjuda molntjänster i prenumerationsform. Problemet som uppstår är då vem som äger och har tillgång till den insamlade metadatan. Är det då förteget själv som äger metadatan och ska ha rätten att bestämma vem som får ha användning av den? Eller är det tillverkarna som står för lagringen och insamlingen? Ska tillverkarna få använda metadatan från en fabrik för sin egen vinning?

6.5 Framtida utvecklingsmöjligheter

I det här delkapitlet diskuteras möjliga tillägg och examensarbetets utvecklingsmöjligheter.

För programmet finns det en hel del utvecklingsmöjligheter. Fler banstyrningsmoduler skulle kunna tillverkas. Exempelvis moduler för sektioner som kan ta emot pallar från två och tre olika sektioner. Även en modul för en

sektion som kan skicka pallar till tre olika sektioner skulle kunna bildas. Man skulle även kunna introducera arbetsstationer med maskiner som ska utföra nån form av arbete på pallarna, exempelvis plasta in dem. Då skulle man även kunna utvidga pallregister till att innehålla variabler som anger huruvida arbetet på en viss station utförts samt datum och tid då arbetet utfördes.

En överordnad databas skulle kunna implementeras och kopplas ihop med PLC-programmet. Bansystemets pallregister samt metadata kan då skickas till databasen och lagras där. Man skulle då också kunna utveckla ett grafiskt användargränssnitt till databasen där all information och metadata presenteras.

Programmet kan och borde testas på en fysisk anläggning. Det skulle då kunna visa sig andra möjligheter för optimering av programmet som inte uppenbarat sig vid simulering i en virtuell miljö.

7 Källförteckning

Soft PLC

[**1**] (Siemens, *Virtual commissioning of automation solutions*, Available: https://www.siemens.com/global/en/home/products/automation/industry-software/automation-software/tia-portal/virtual-commissioning.html, (2018-05-14))

TIA Manual

[**2**] (Siemens AG, 2018-02-21, *TIA Portal - An Overview of the Most Important Documents and Links - Controller*, Available: https://support.industry.siemens.com/cs/document/65601780/tia-portal-an-overview-of-the-most-important-documents-and-links-controller?dti=0&lc=en-WW, (2018-05-14))

TIA → HMI

[**3**] (Siemens AG, 2018-05, Siemens *Simatic S7-1500 Getting started*, Available: https://www.automation.siemens.com/salesmaterial-as/interactive-manuals/getting-started_simatic-s7-1500/documents/EN/software_complete_en.pdf, (2018-05-14))

HMI

[**4**] (Siemens AG, 2017-08-09, *Creating Faceplates with WinCC Runtime Advanced and Comfort Panels*, Available: https://support.industry.siemens.com/cs/document/68014632/creating-faceplates-with-wincc-runtime-advanced-and-comfort-panels?dti=0&lc=en-WW, (2018-

05-14))

Multiuser

[**5**] (Siemens AG, 2017-02-24, *Multiuser Engineering in TIA Portal*, Available: https://support.industry.siemens.com/cs/document/109740141/multiuser-engineering-in-tia-portal?dti=0&lc=en-WW, (2018-05-14))

Spiralmodellen [**6**] (Barry Boehm, 2000-07, *Spiral Development: Experience, Principles, and Refinements*, Available: https://resources.sei.cmu.edu/asset_files/SpecialReport/2000_003_001_13655.pdf, (2018-05-14))

S7-1500 HB

[**7**] (Hans Berger,2014-05, *Automating with SIMATIC S7-1500 - Configuring, Programming and Testing with Step 7 Professional*, Available: http://plc4good.org.ua/files/02_materials/book/Automating%20with%20SIMATIC%20S7-1500_%20Configuring,%20Programming%20and%20Testing%20with%20STEP%207%20Professional-Publicis%20(2014).pdf, (2018-05-14))

VMware

[**8**] (VMware, *Simplify and Automate Virtualization Management*, Available: https://www.vmware.com/solutions/virtualization/virtualization-management.html, (2018-05-14))

8 Appendix

Funktionsblocken som är bifogade i appendix har skrivits ut med print kommandot i TIA Portalen. När man gör en utskrift av ett block i TIA Portalen inkluderas blockgränssnittet med varje flik utvidgad. Detta leder till att oerhört många sidor med intetsägande variabler skrivs ut. Dessa sidor har klippts bort och ersatts med en skärmdump på blockgränssnittet med alla flikar komprimerade.

8.1 Appendix 1: Track Module

Track Module

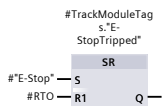
	Name	Data type	Comment
1	Input		
2	InputsPrevious	"InputsPrevious"	Handshake signals from the previous trackunit
3	InputsNext	"InputsNext"	Handshake signals from the next trackunit
4	E-Stop	Bool	Emergency stop
5	RTO	Bool	Reset tripped object
6	P1	Bool	Photocell on the previous trackunit
7	P2	Bool	Photocell on the trackunit
8	Mode	Bool	False = Auto | True = Manual
9	MotorFeedback	Bool	Feedback from the motor
10	PalletID-in	String	Pallet ID received from previous trackunit
11	PalletLogSize	Int	Number of Pallet entries in the array PalletLog
12	Output		
13	PalletLog	"PalletLog"	PalletLog made up of an array of pallet entries
14	OutputsPrevious	"OutputsPrevious"	Handshake signals to the previous trackunit
15	OutputsNext	"OutputsNext"	Handshake signals to the next trackunit
16	DataLog	"AllTracksModuleData"	Trackunit data
17	StartMotor	Bool	Signal to start the motor
18	PalletID-out	String	Pallet ID to send along with the pallet to the next trackunit
19	PalletLostMotor	Bool	Signal to motor that a PalletLost_Alarm has been generated
20	InOut		
21	Static		
22	PalletLost	Struct	Signals that generates the PalletLost_Alarms
23	TrackModuleTags	"TrackModuleTags"	Tags for logic within the module
24	HMI	"TrackModuleHMI"	Tags for communication with the HMI
25	TrackunitStatusHMI	"TrackunitStatus"	Status displayment of the trackunit for HMI screen TrackData
26	Timers	Struct	Timers used in the module
27	PalletLostIncoming_Alarm	Program_Alarm	Generates the PalletLostIncoming_Alarm
28	PalletLostOutgoing_Alarm	Program_Alarm	Generates the PalletLostOutgoing_Alarm
29	PalletExitTrack_Trig	R_TRIG	Detect a positive signal edge of a pallet exiting the trackunit
30	Temp		
31	Output_error	Bool	Whether or not an error occured during execution of Get_AlarmState
32	Output_status	Word	Display of status during execution of Get_AlarmState
33	AlarmState	Byte	Status of the Program_Alarm as a bit array
34	DateTimeInt	Int	Status of the Read time-of-day instruction

Figur 38: *Track Module blockgränssnitt.*

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
F2Button	Bool	false	Non-retain	True	True	True	False		Button to activate photocell 2
PalletIDFromHMI	String	''	Non-retain	True	True	True	False		Enter pallet id in manual mode
ManualMotorStart	Bool	false	Non-retain	True	True	True	False		Start motor in manual mode
▼ TrackunitStatusHMI	"TrackunitStatus"		Non-retain	True	True	True	False		Status displayment of the trackunit for HMI screen TrackData
HasPallet	Bool	false	Non-retain	True	True	True	False		Pallet on/off track
PalletID	String	''	Non-retain	True	True	True	False		The id of the pallet on the trackunit
Photocell	Bool	false	Non-retain	True	True	True	False		Photocell on the trackunit
▼ Timers	Struct		Non-retain	True	True	True	False		Timers used in the module
▼ PalletOnTrack_Timer	TONR_TIME		Non-retain	True	True	True	False		Timer to log the time each pallet spends on the trackunit
PT	Time	T#0ms	Non-retain	True	True	True	False		
ET	Time	T#0s	Non-retain	True	False	True	False		
IN	Bool	false	Non-retain	True	True	True	False		
Q	Bool	false	Non-retain	True	False	True	False		
▼ PalletLostOutgoing_Timer	TONR_TIME		Non-retain	True	True	True	False		Timer for the PalletLostOutgoing_Alarm
PT	Time	T#0ms	Non-retain	True	True	True	False		
ET	Time	T#0ms	Non-retain	True	False	True	False		
IN	Bool	false	Non-retain	True	True	True	False		
Q	Bool	false	Non-retain	True	False	True	False		
▼ PalletLostIncoming_Timer	TONR_TIME		Non-retain	True	True	True	False		Timer for the PalletLostIncoming_Alarm
PT	Time	T#0ms	Non-retain	True	True	True	False		
ET	Time	T#0ms	Non-retain	True	False	True	False		
IN	Bool	false	Non-retain	True	True	True	False		
Q	Bool	false	Non-retain	True	False	True	False		
PalletLostIncoming_Alarm	Program_Alarm			True	True	True	False		Generates the PalletLostIncoming_Alarm
PalletLostOutgoing_Alarm	Program_Alarm			True	True	True	False		Generates the PalletLostOutgoing_Alarm
▼ PalletExitTrack_Trig	R_TRIG			True	True	True	False		Detect a positive signal edge of a pallet exiting the trackunit
▼ Input									
CLK	Bool	false	Non-retain	True	True	True	False		
▼ Output									
Q	Bool	false	Non-retain	True	True	True	False		
InOut									
▼ Static									
Stat_Bit	Bool	false	Non-retain	True	True	True	False		
▼ Temp									
Output_error	Bool								Whether or not an error occured during execution of Get_AlarmState
Output_status	Word								Display of status during execution of Get_AlarmState
AlarmState	Byte								Status of the Program_Alarm as a bit array
DateTimeInt	Int								Status of the Read time-of-day instruction
Constant									

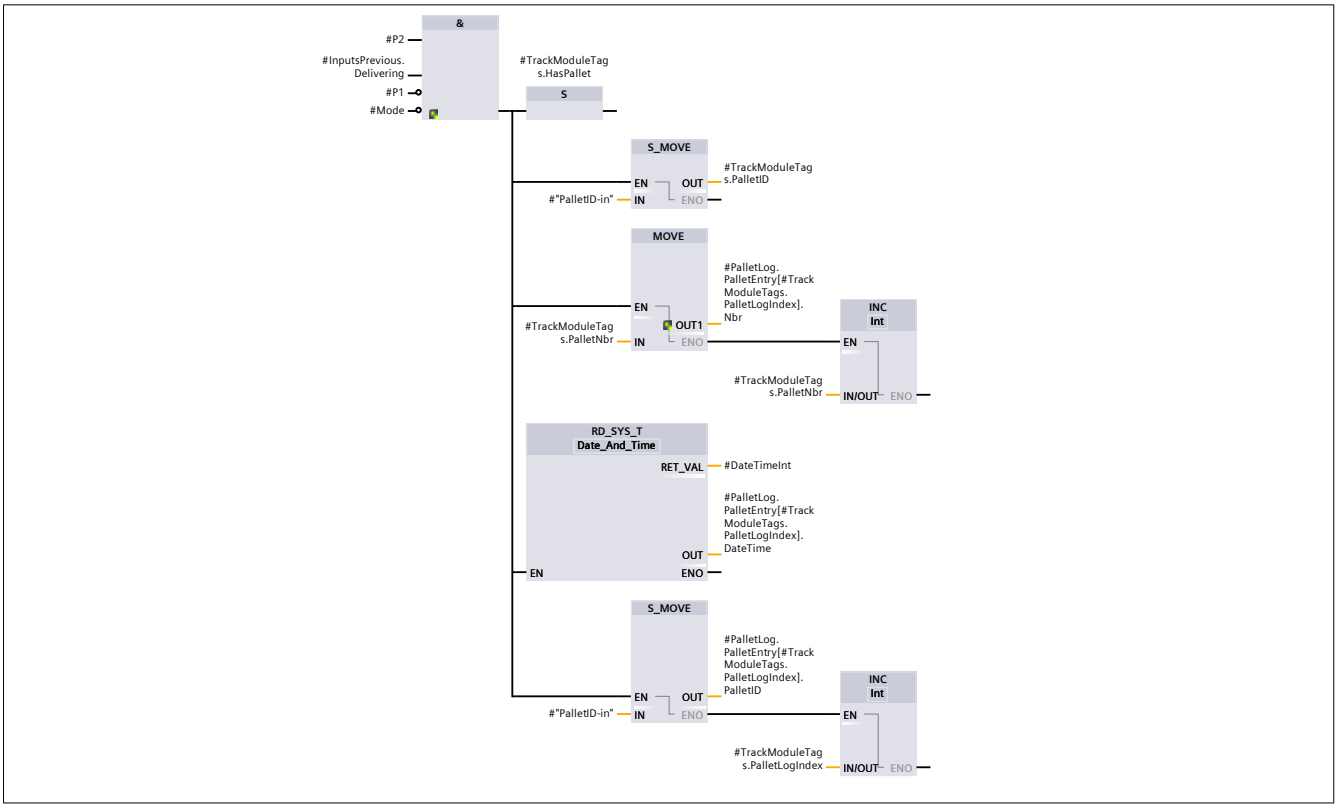
Network 1: Emergency stop

Emergency stop sets the memory E-stopTripped. Reset memory with GlobalRTO when emergency stop is false.



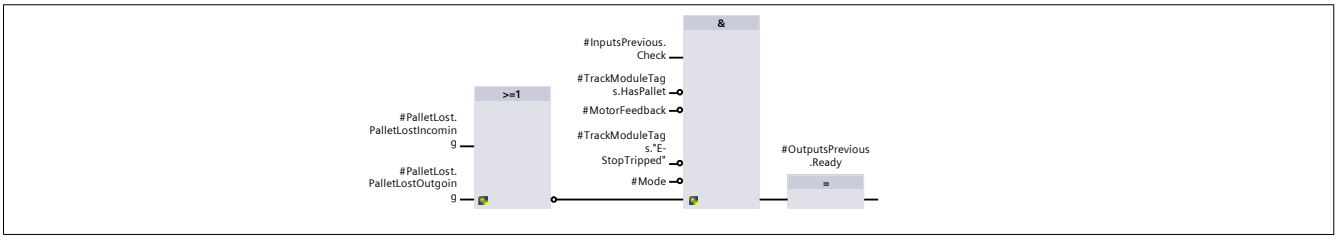
Network 2: Pallet has been delivered by previous trackunit. Log the pallet entry.

Set the memory HasPallet. Log the pallet id, number and date and time. Automatic mode



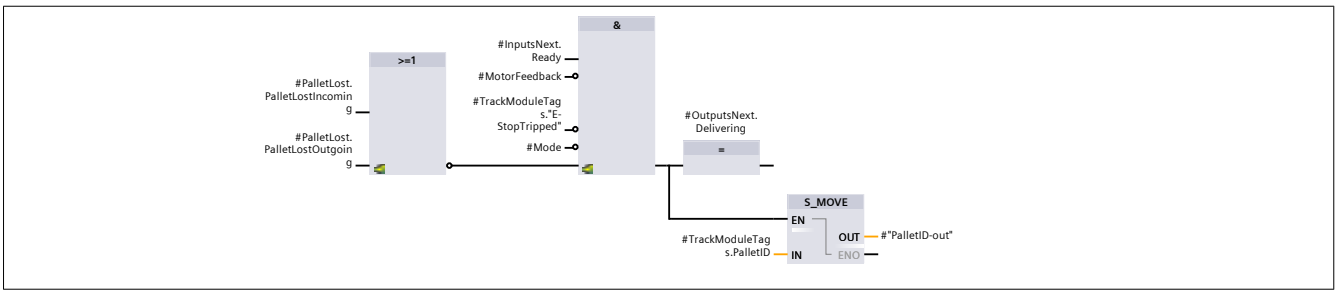
Network 3: Ready to receive a pallet from the previous trackunit

Check if the trackunit is ready to receive a pallet. Set the output signal Ready used for handshake with the previous trackunit.



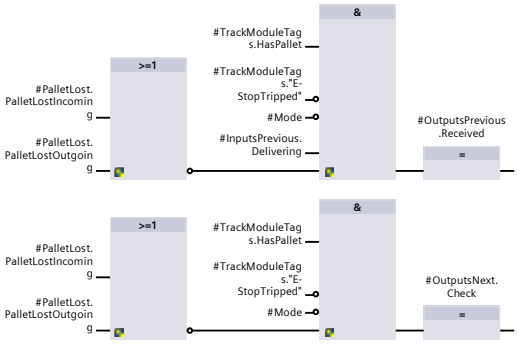
Network 4: Delivering a pallet to the next trackunit

Set the output signal Delivering used for handshake with the next trackunit.



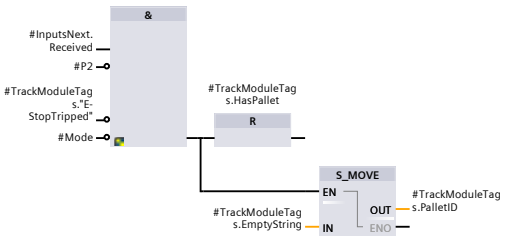
Network 5: Pallet has been received.

Set the output signal Received used for handshake with the previous trackunit.
Set the output signal Check used for handshake with the next trackunit.



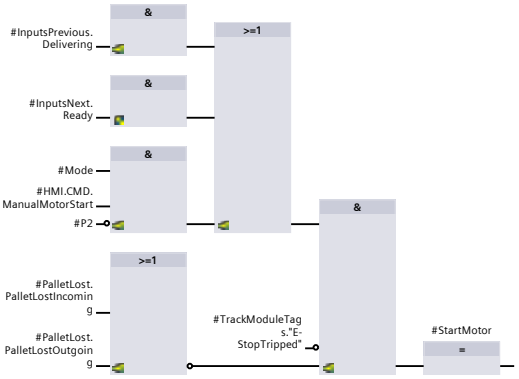
Network 6: No pallet on trackunit

Pallet transfer complete, reset the memory HasPallet. Overwrite the PalletID with an empty string.



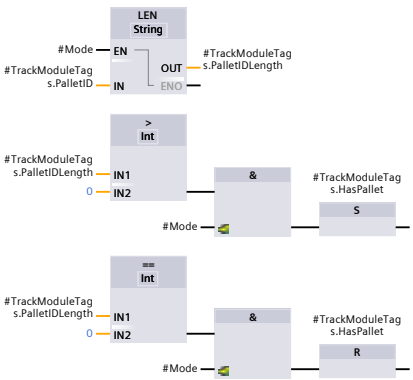
Network 7: Start motor

The trackunit is either receivng a pallet from the previous trackunit or delivering a pallet to the next trackunit.
Set the output StartMotor.



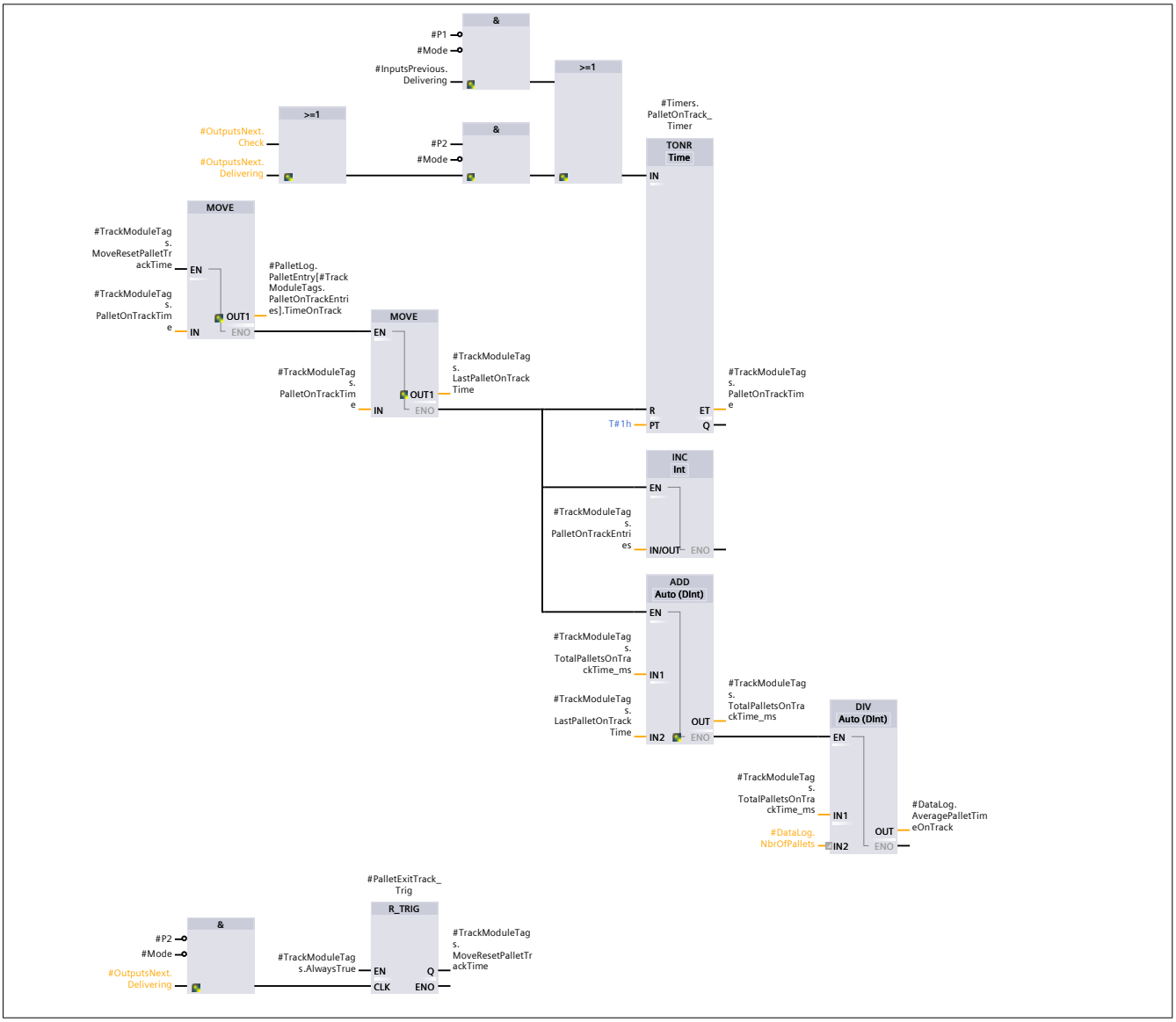
Network 8: Pallet id in manual mode

Handle the entered pallet id from the HMI in manual mode. If an empty string is entered the trackunit has no pallet.



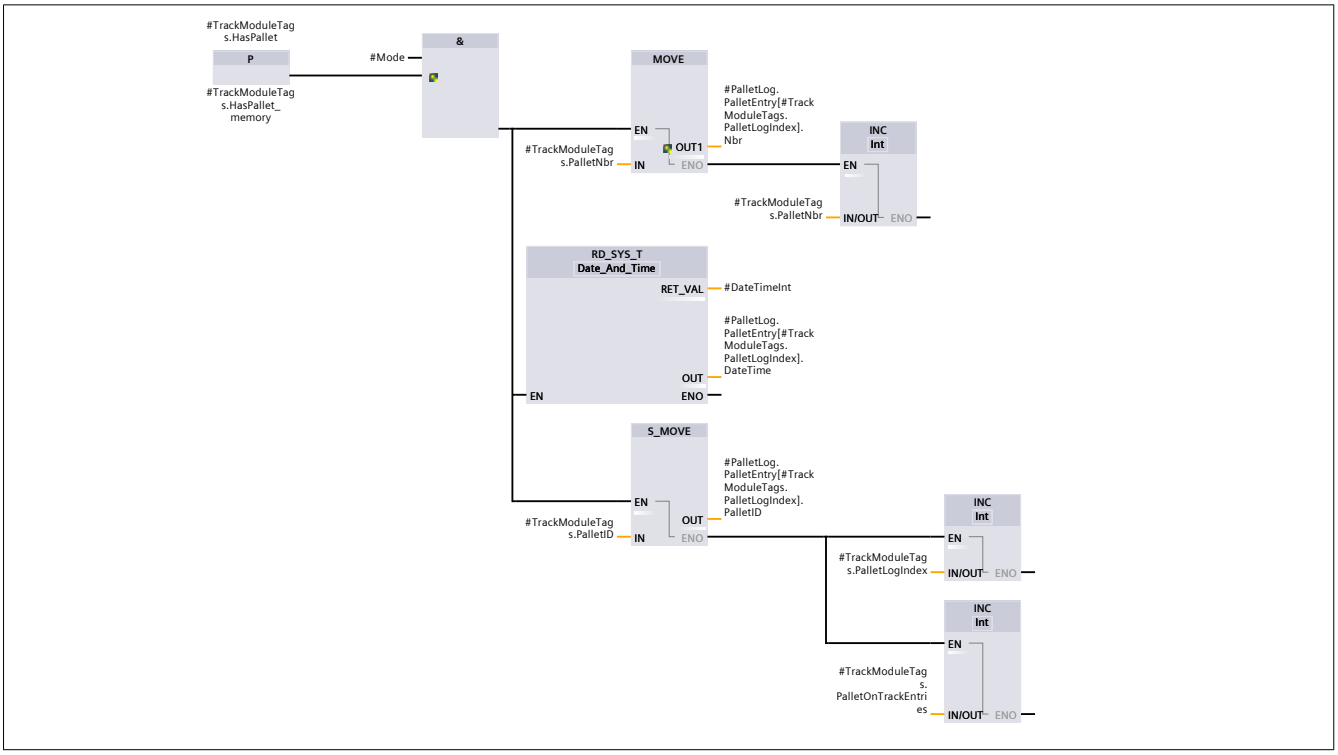
Network 9: Log pallet time on trackunit

Log the amount of time the pallet is on the trackunit and calculates the average time. Only in automatic mode.



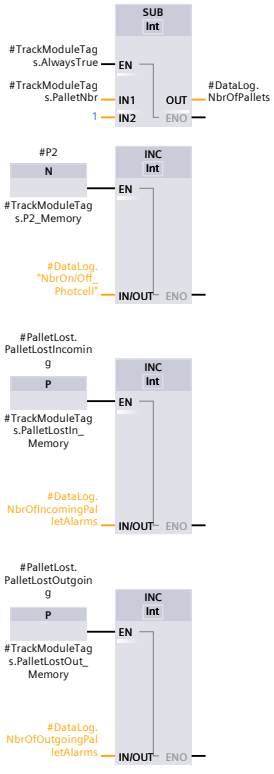
Network 10: Log pallet entries in manual mode

Log pallet id, number and date and time in manual mode.



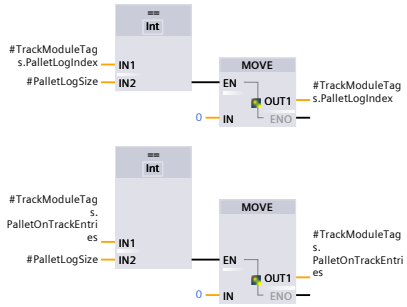
Network 11: Log data

Log number of pallets, on/off photocell, incoming and outgoing pallet stuck/lost alarm.



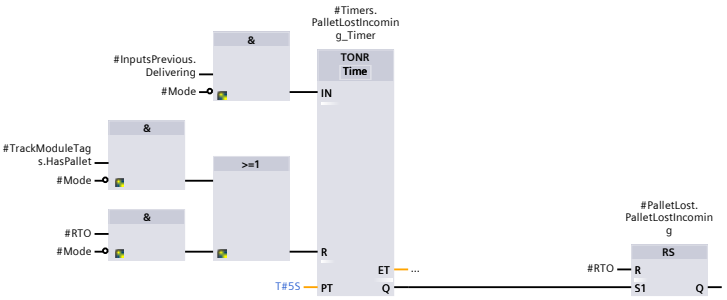
Network 12: Reset PalletLogIndex and PalletOnTrackEntries

Reset PalletLogIndex and PalletOnTrackEntries used for logging pallet entries in the PalletLog. PalletLogSize is the size of the log array.



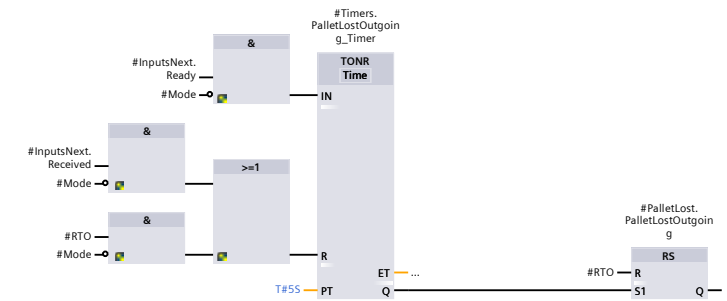
Network 13: Timer for PalletLostIncoming_Alarm

Timer checks that a pallet that's being received from the previous trackunit has been received within the expected time, if not a timer sets signal PalletLostIncoming.



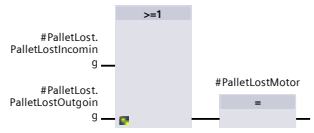
Network 14: Timer for PalletLostOutgoing_Alarm

Timer checks that a pallet that's being delivered to next trackunit has been delivered within the expected time, if not a timer sets signal PalletLostOutgoing.



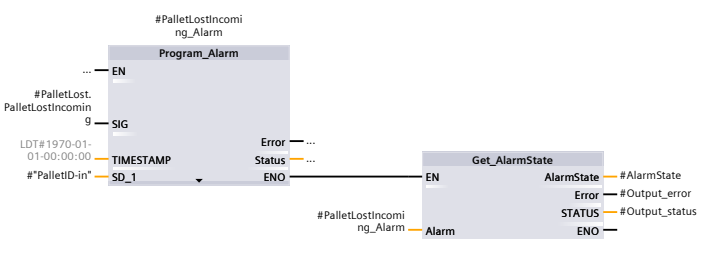
Network 15: Pallet lost signal to motor

If a PalletLost alarm is active send the signal PalletLostMotor to stop the motor.



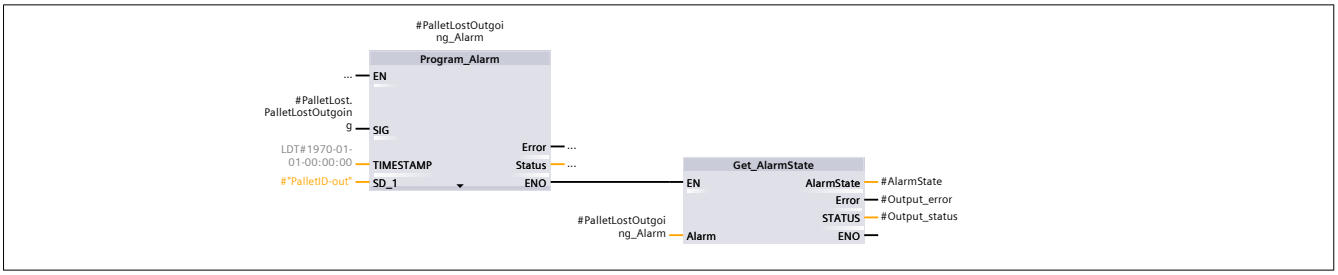
Network 16: Program_Alarm for PalletLostIncoming

When the signal PalletLostIncoming is set by timer a PalletLostIncoming_Alarm is generated by the Program_Alarm function block.



Network 17: Program_Alarm for PalletLostOutgoing

When the signal PalletLostOutgoing is set by timer a PalletLostOutgoing_Alarm is generated by the Program_Alarm function block.



Network 18: HMI

SCL network for communication betwen module and HMI.

```
0001 // Mode selected by HMI. Automatic by default
0002 #HMI.Status.Mode := #Mode;                // False = Auto | True = Manual
0003
0004 // Set pallet status for HMI
0005 IF NOT #HMI.Status.Mode THEN                  // Automatic mode
0006     #HMI.CMD.ManualMotorStart := FALSE;
0007     #HMI.Status.PalletID := #TrackModuleTags.PalletID;
0008 ELSE                                          // Manual mode
0009     #TrackModuleTags.PalletID := #HMI.Status.PalletID;
0010 END_IF;
0011
0012 IF #TrackModuleTags.HasPallet THEN
0013     #HMI.Status.HasPallet := TRUE;
0014 ELSE
0015     #HMI.Status.HasPallet := FALSE;
0016     ;
0017 END_IF;
0018
0019 // Update HMI Status tag, used by faceplate
0020 IF NOT #"E-Stop" AND NOT #TrackModuleTags."E-StopTripped" THEN
0021     IF NOT #HMI.Status.HasPallet THEN
0022         #HMI.Status.Status := 1;
0023     ELSE
0024         #HMI.Status.Status := 2;
0025         ;
0026     END_IF;
0027 END_IF;
0028
0029 // Emergency stop activated
0030 IF #"E-Stop" THEN
0031     #HMI.Status.Status := 3;
0032     ;
0033 END_IF;
0034
0035 // Tripped by emergency stop
0036 IF NOT #"E-Stop" AND #TrackModuleTags."E-StopTripped" THEN
0037     #HMI.Status.Status := 4;
0038     ;
0039 END_IF;
0040
0041 // Reset tripped, reset status
0042 IF NOT #"E-Stop" AND #RTO THEN
0043     IF NOT #HMI.Status.HasPallet THEN
0044         #HMI.Status.Status := 1;
0045     ELSE
0046         #HMI.Status.Status := 2;
0047         ;
0048     END_IF;
0049         ;
0050     END_IF;
0051
0052 // For displayment of the trackunit status to an HMI screen
0053 #TrackunitStatusHMI.HasPallet := #TrackModuleTags.HasPallet;
0054 #TrackunitStatusHMI.PalletID := #TrackModuleTags.PalletID;
0055 #TrackunitStatusHMI.Photocell := #P2;
```

8.2 Appendix 2: Motor Module

Motor Module

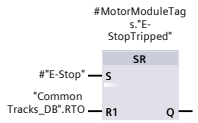
		Name	Data type	Comment
1		Input		
2		E-Stop	Bool	Emergency stop
3		MotorCircuitBreaker	Bool	Motor Protective Circuit Breaker
4		Fuse	Bool	Fuse
5		AlarmReset	Bool	Reset Fuse/CB alarm
6		PalletLost	Bool	Alarm signal that a pallet is stuck or lost
7		RTO	Bool	Reset tripped object after an alarm has been acknowledged
8		StartMotor	Bool	Signal from the track module to start the motor
9		MaintenanceTimeInMinutes	USInt	Minutes before a maintenance check is recommended
10		Output		
11		Motor	Bool	Motor that runs the track
12		MotorData	"MotorData"	Motor Data
13		Feedback	Bool	Feedback to the track module
14		InOut		
15		Static		
16		MotorModuleTags	"MotorModuleTags"	Tags used for logic within the module
17		MotorAlarms	"MotorAlarms"	Signals for generating the motor alarms
18		HMI	"MotorModuleHMI"	Tags for communication with the HMI
19		Timers	Struct	Timers used in the module
20		Fuse_alarm	Program_Alarm	Generates the Fuse_Alarm
21		CircutBreaker_alarm	Program_Alarm	Generates the CircuitBreaker_Alarm
22		Info_Maintenance	Program_Alarm	Generates the Info_Maintenance_Alarm
23		Temp		
24		Output_error	Bool	Whether or not an error occured during execution of Get_AlarmState
25		Output_status	Word	Display of status during execution of Get_AlarmState
26		AlarmState	Byte	Status of the Program_Alarm as a bit array

Figur 39: *Motor Module blockgränssnitt.*

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
Fuse_alarm	Program_Alarm			True	True	True	False		Generates the Fuse_Alarm
CircutBreaker_alarm	Program_Alarm			True	True	True	False		Generates the CircuitBreaker_Alarm
Info_Maintenance	Program_Alarm			True	True	True	False		Generates the Info_Maintenance_Alarm
Temp									
Output_error	Bool								Whether or not an error occured during execution of Get_AlarmState
Output_status	Word								Display of status during execution of Get_AlarmState
AlarmState	Byte								Status of the Program_Alarm as a bit array
Constant									

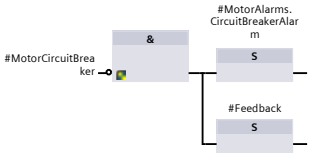
Network 1: Emergency stop

Emergency stop sets memory S_E-stopTripped. Reset tag with GlobalRTO when emergency stop is false.



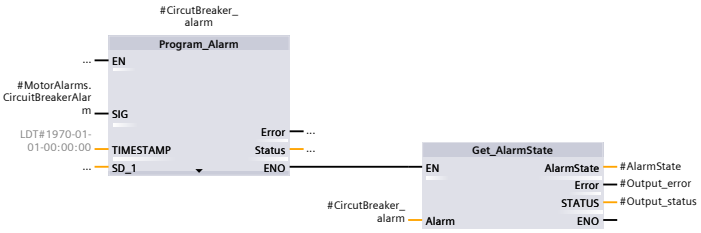
Network 2: Motor Protective Circuit Breaker Alarm

MotorCircuitBreaker input sets the motor protective CircuitBreakerAlarm signal and the Feedback signal to the track unit.



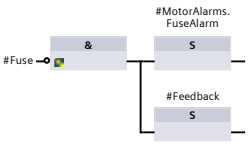
Network 3: Program_Alarm for the CB alarm

When the CircuitBreakerAlarm signal is set a CircuitBreaker_Alarm is generated by the Program_Alarm. The alarm status is then outputted by the Get_AlarmState.



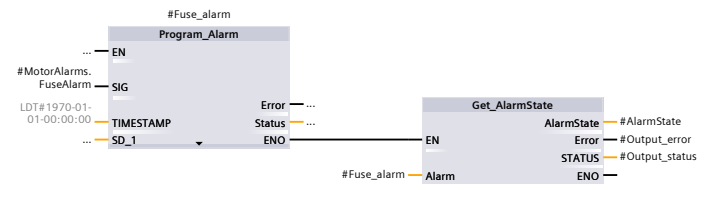
Network 4: Fuse Alarm

Fuse input sets the FuseAlarm signal and the Feedback signal to the track unit.



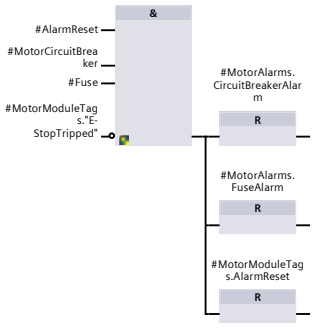
Network 5: Program_Alarm for the Fuse_Alarm

When the FuseAlarm signal is set a Fuse_Alarm is generated by the Program_Alarm. The alarm status is the outputted by the Get_AlarmState.



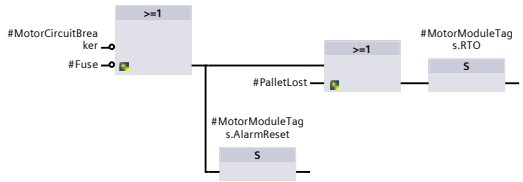
Network 6: Reset alarms

Reset the CircuitBreakerAlarm and/or the FuseAlarm.



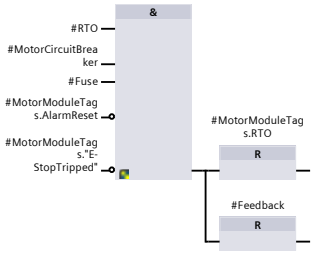
Network 7: Static memory RTO set

Static memory RTO set by an alarm to prevent immediate start after an alarm reset.



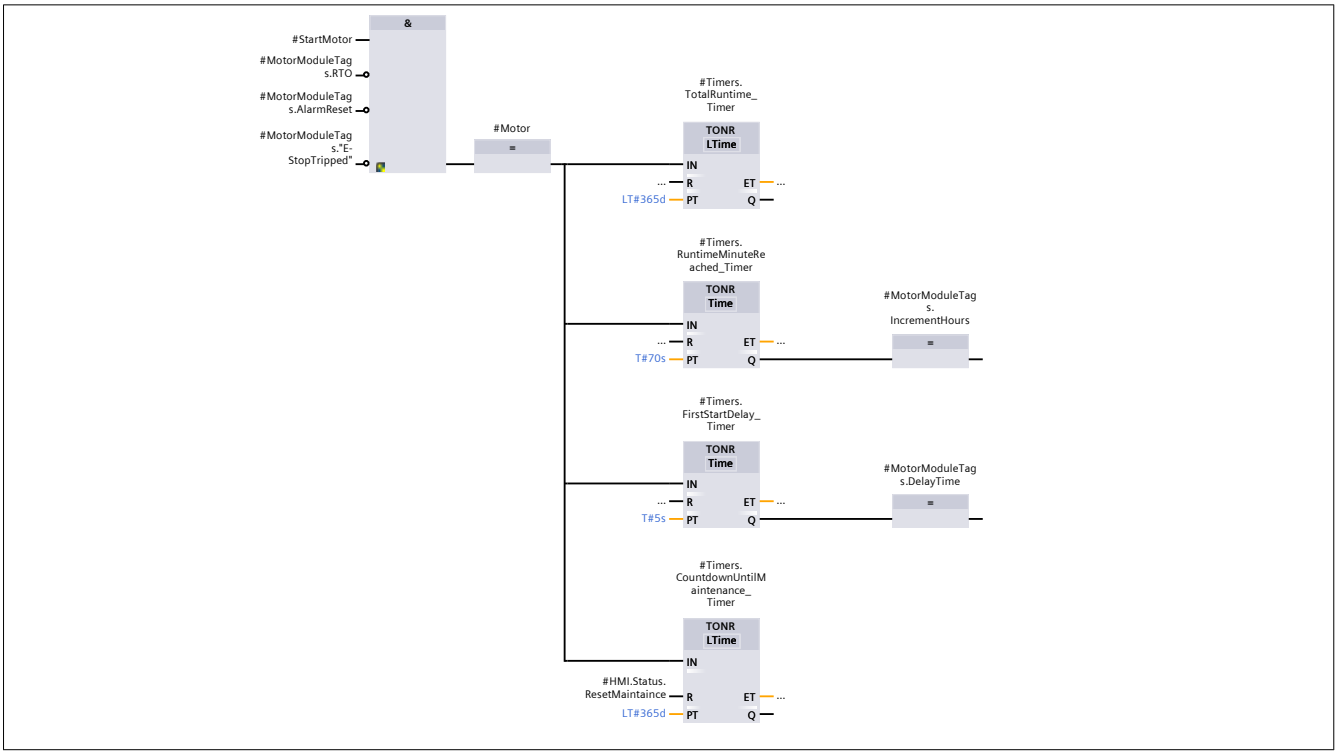
Network 8: Static memory RTO reset

Static memory RTO reset by input RTO to enable start up.



Network 9: Contactor

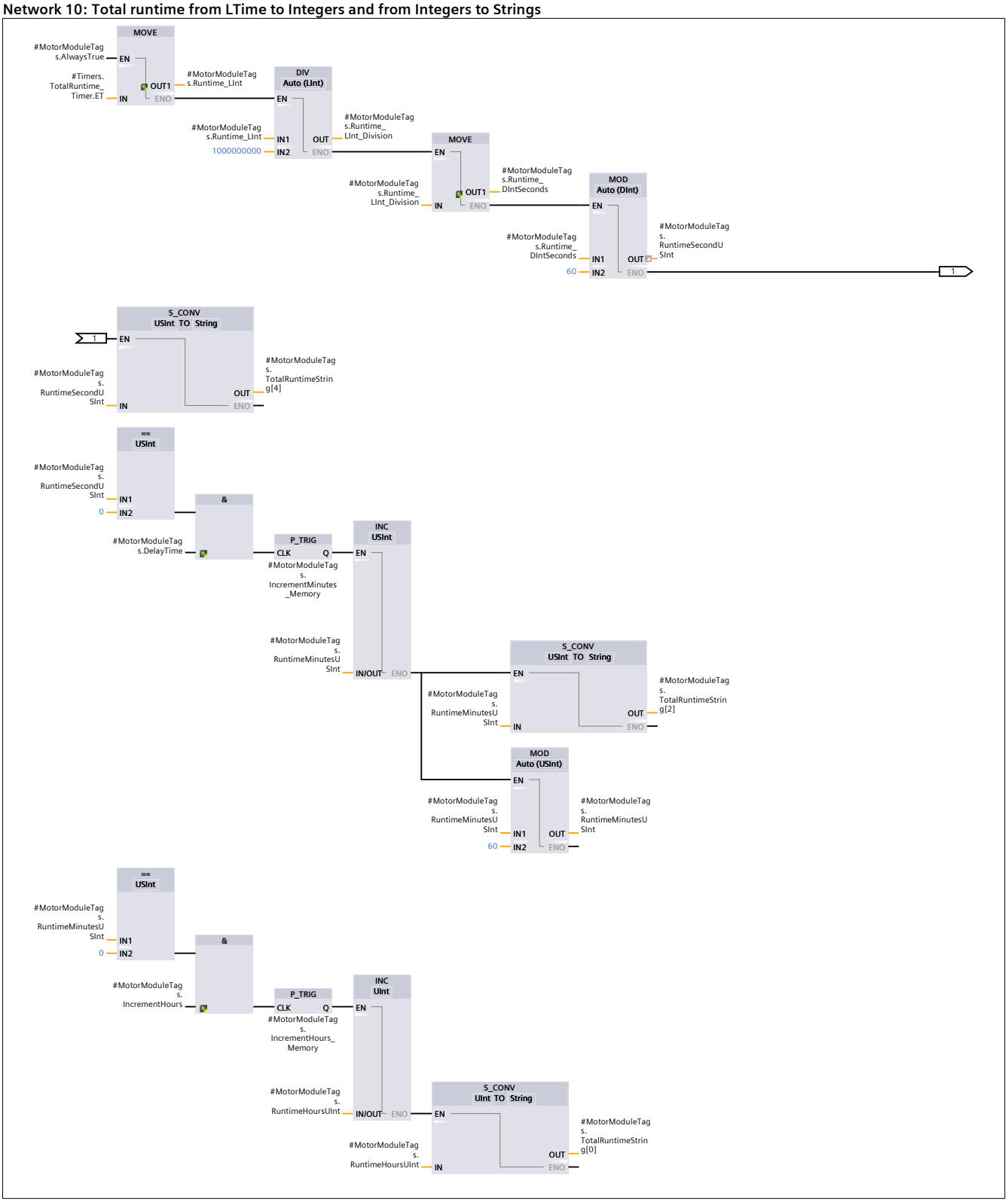
Enable output Motor. Log the motor runtime since the last maintenance check.



Network 10: Total runtime from LTime to Integers and from Integers to Strings

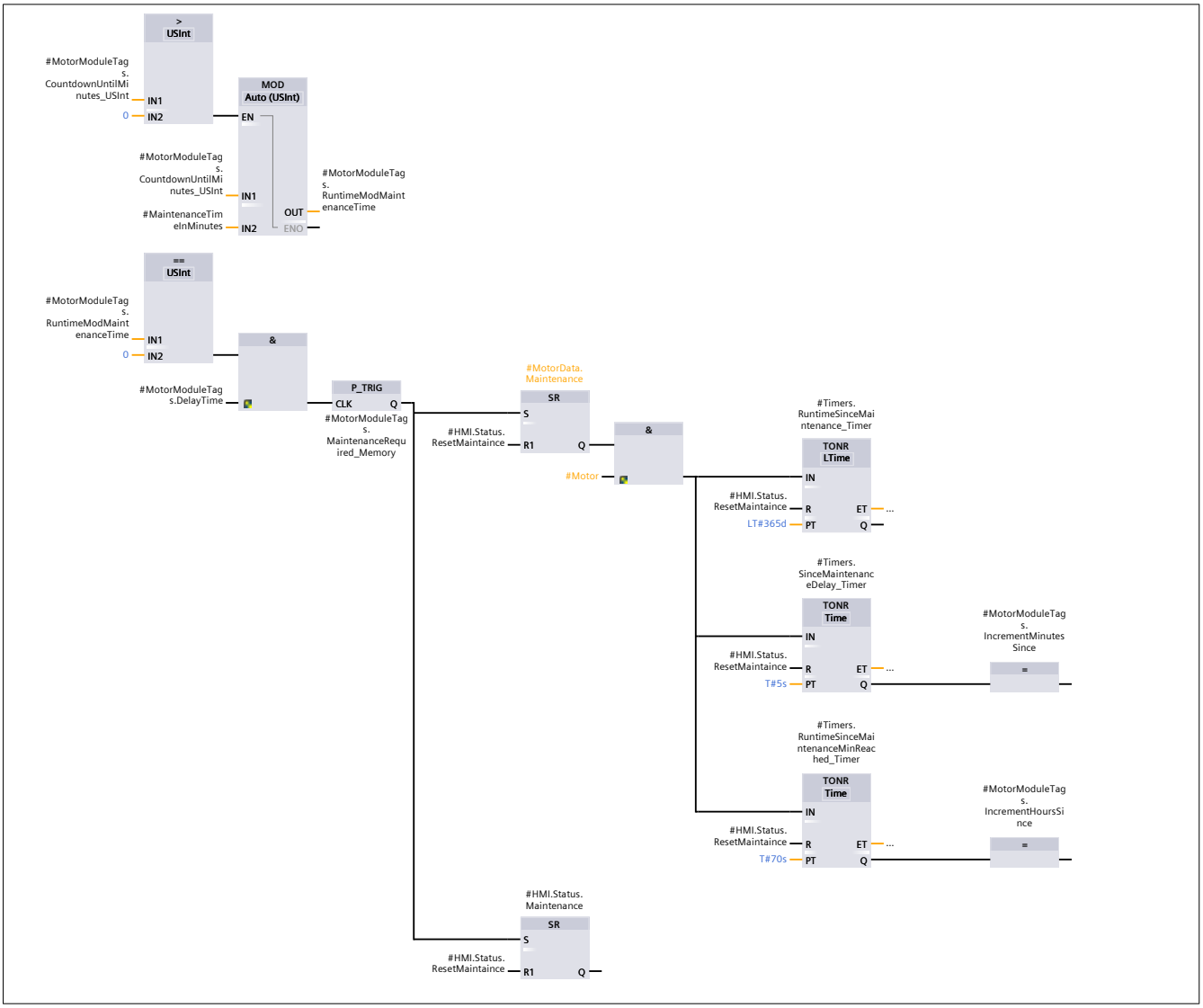
Convert the runtime from LTime to a Integers for minutes, seconds and hours. Then convert each Integer to a String.

Network 10: Total runtime from LTime to Integers and from Integers to Strings



Network 11: Maintenance

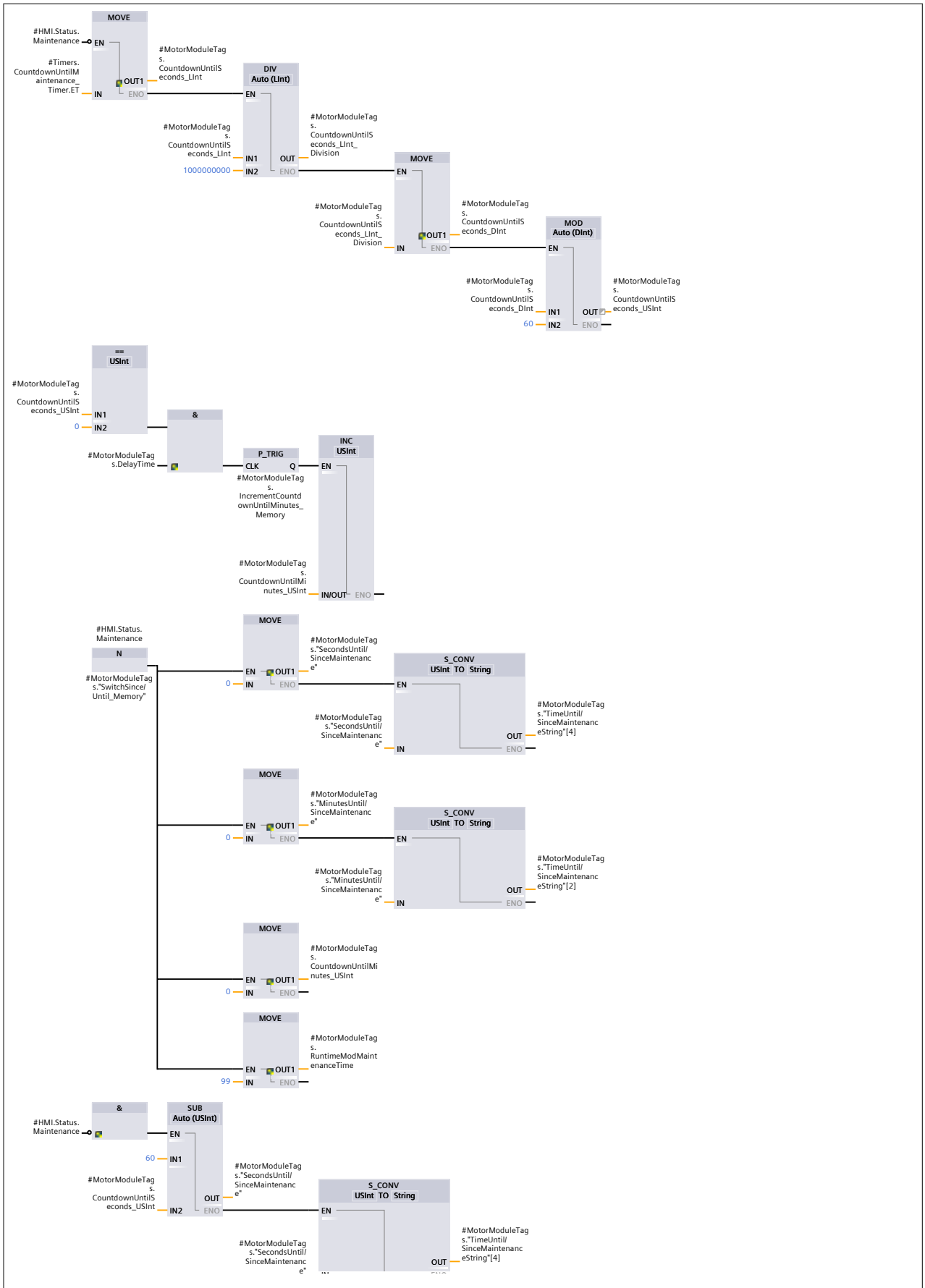
Checks if the runtime has reached the time value for a maintenance. Set signal Maintenance, the signal is reset by the "Serviced" button the HMI.



Network 12: Time until maintenance is recommended

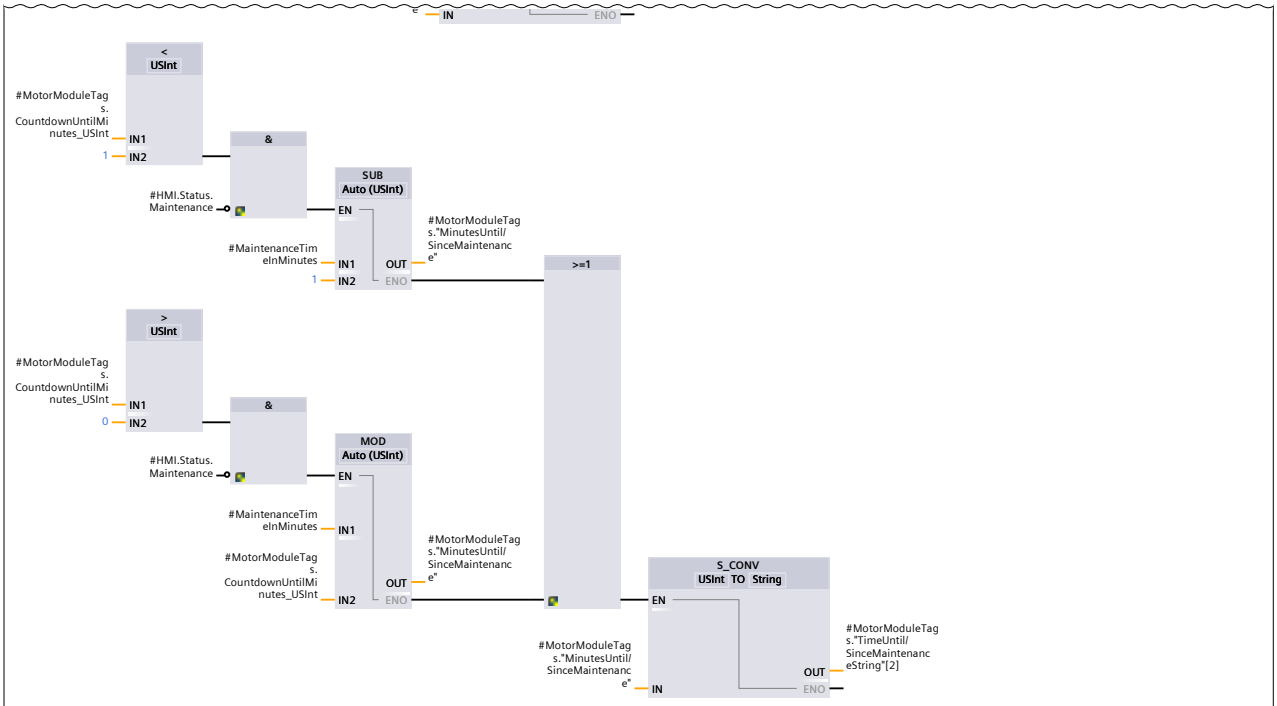
Calculate the time until a maintenance is recommended

Network 12: Time until maintenance is recommended (1.1 / 2.1)



Network 12: Time until maintenance is recommended (2.1 / 2.1)

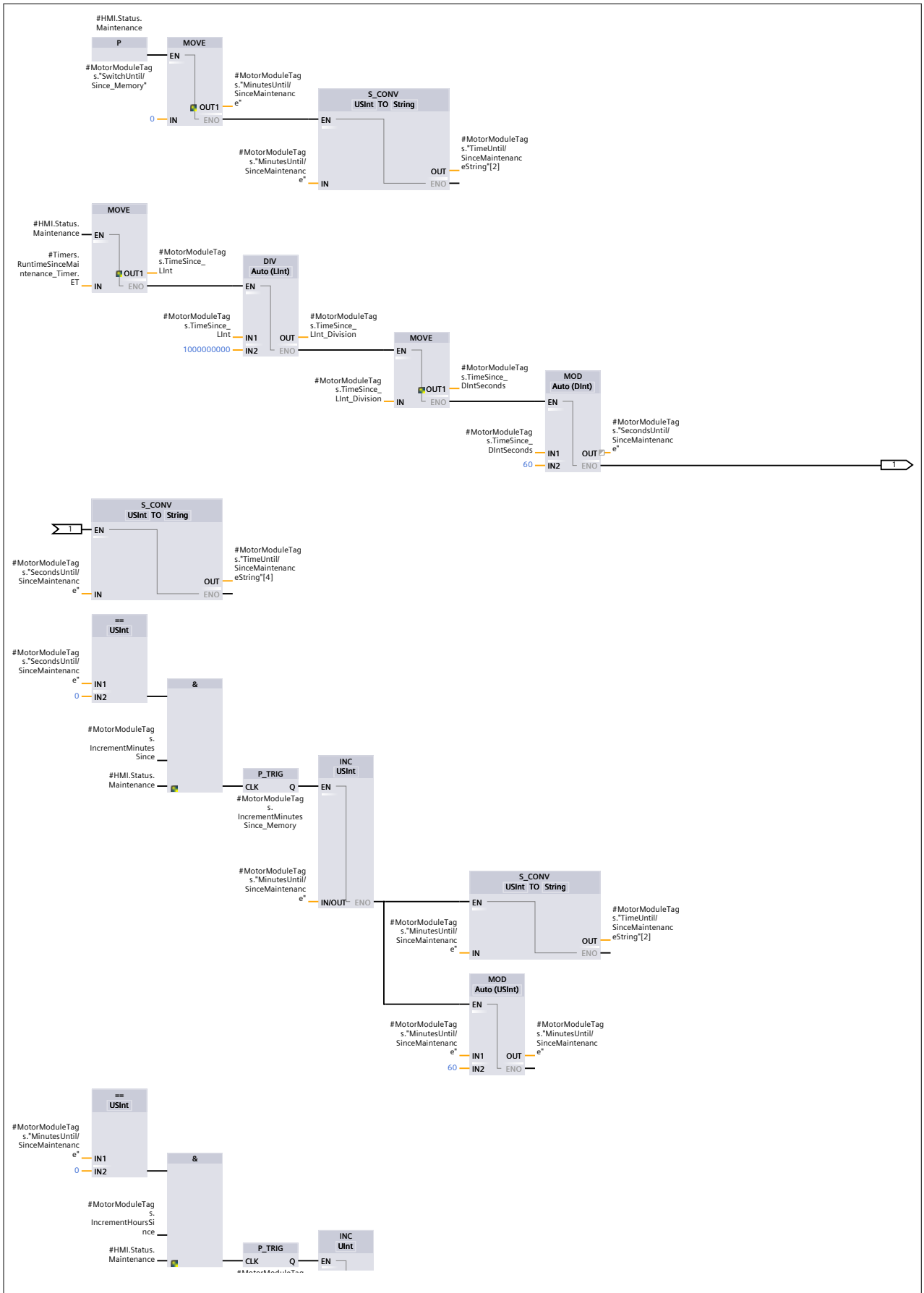
1.1 (Page1 - 9)



Network 13: Time since maintenance was recommended

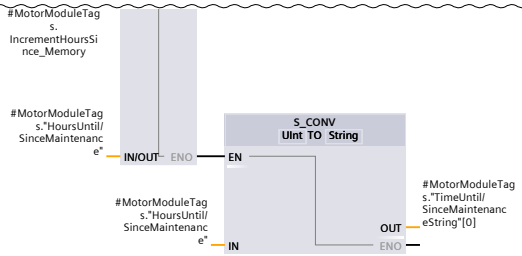
Calculate the time elapsed since a maintenance was recommended.

Network 13: Time since maintenance was recommended (1.1 / 2.1)



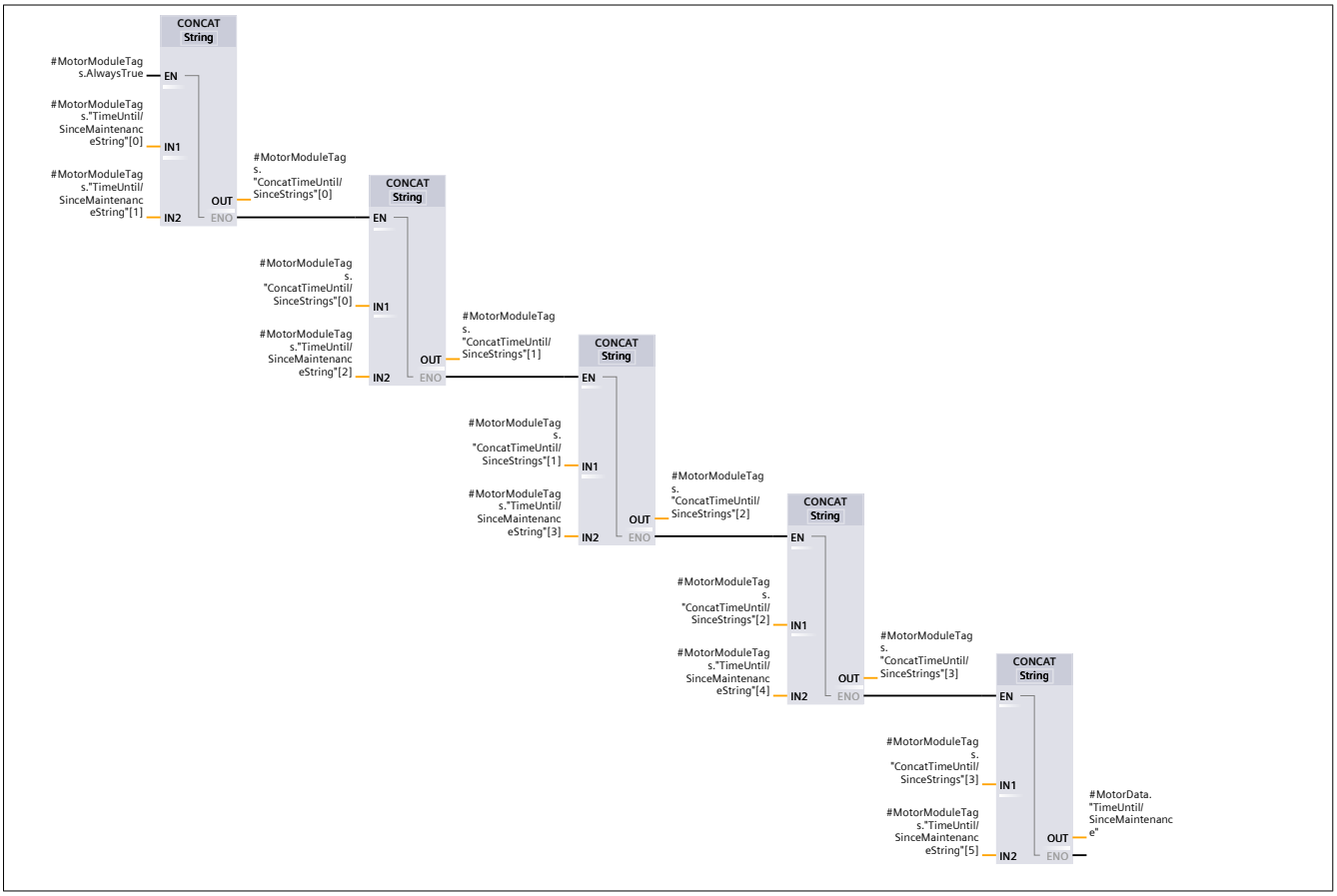
Network 13: Time since maintenance was recommended (2.1 / 2.1)

1.1 (Page1 - 12)



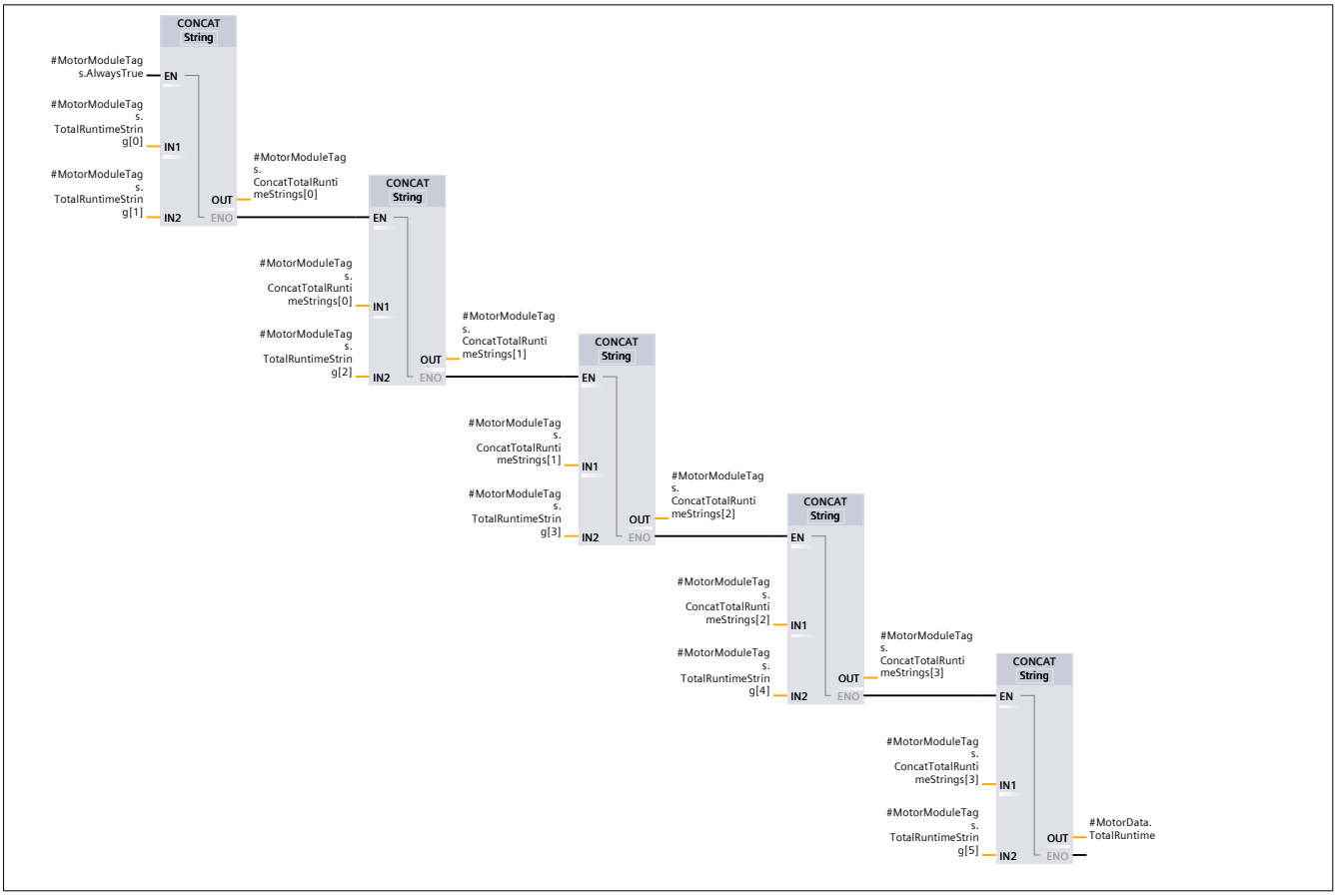
Network 14: Concatenate TimeUntilMaintenance String

Concatenates the separate Strings for hours, minutes and seconds into one String TimeUntilMaintenance.



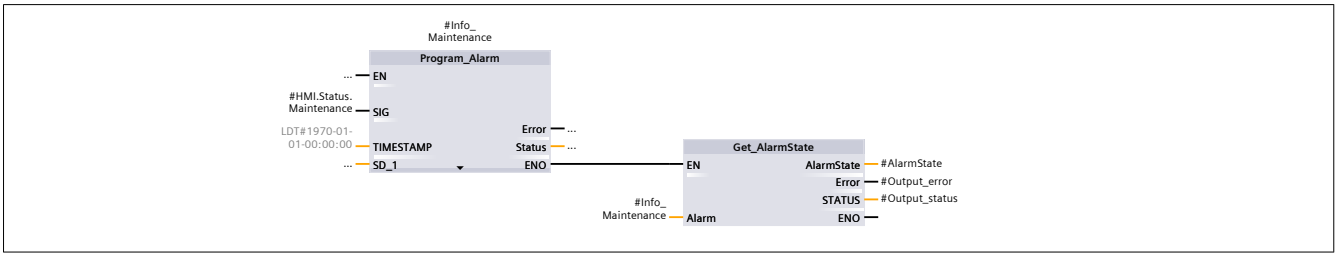
Network 15: Concatenate TotalRuntime String

Concatenates the separate Strings for hours, minutes and seconds into one String TotalRuntime.



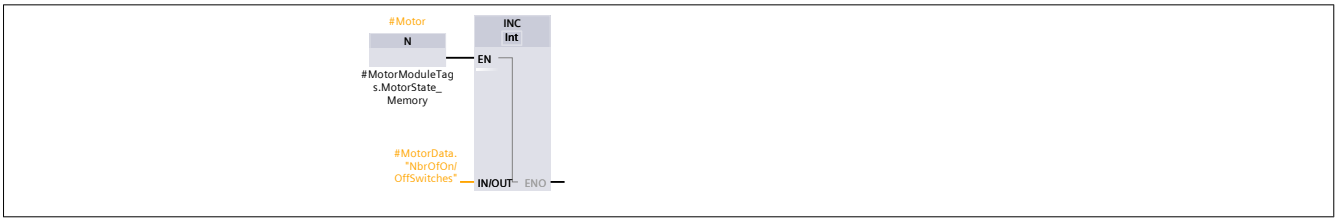
Network 16: Program_Alarm for Info_Maintenance

When the TotalTimeMin reaches the set value an Info_Maintenance is generated by the Program_Alarm. The alarm status is the outputted by the Get_AlarmState.



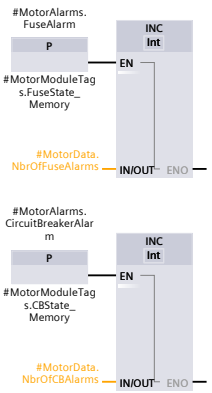
Network 17: Motor on/off

Counts the number of times the motor has been turned on/off.



Network 18: Fuse & CB alarm data

Counts the number of times a fuse/CB alarm has occured.



Network 19: HMI

SCL network for all communication betwen module and HMI.

```
0001 // Active alarm
0002 IF #MotorAlarms.CircuitBreakerAlarm OR #MotorAlarms.FuseAlarm OR #PalletLost THEN
0003     #HMI.Status.ActiveAlarm := TRUE;
0004     #HMI.Status.Activated := FALSE;
0005     #HMI.Status.Tripped := FALSE;
0006     #HMI.Status.Inactive := FALSE;
0007     ;
0008 END_IF;
0009 // Activated
0010 IF #Motor THEN
0011     #HMI.Status.Activated := TRUE;
0012     #HMI.Status.Tripped := FALSE;
0013     #HMI.Status.Inactive := FALSE;
0014     #HMI.Status.ActiveAlarm := FALSE;
0015     ;
0016 END_IF;
0017 // Inactive
0018 IF NOT #MotorAlarms.CircuitBreakerAlarm AND NOT #MotorAlarms.FuseAlarm AND NOT #PalletLost AND NOT #Motor THEN
0019     #HMI.Status.Inactive := TRUE;
0020     #HMI.Status.Activated := FALSE;
0021     #HMI.Status.Tripped := FALSE;
0022     #HMI.Status.ActiveAlarm := FALSE;
0023     ;
0024 END_IF;
0025 // Tripped
0026 IF #MotorModuleTags.RTO AND NOT #MotorAlarms.CircuitBreakerAlarm AND NOT #MotorAlarms.FuseAlarm AND NOT #PalletLost
0027     OR ( NOT #"E-Stop" AND #MotorModuleTags."E-StopTripped") THEN
0028     #HMI.Status.Tripped := TRUE;
0029     #HMI.Status.Activated := FALSE;
0030     #HMI.Status.Inactive := FALSE;
0031     #HMI.Status.ActiveAlarm := FALSE;
0032     ;
0033 END_IF;
0034
0035 // Update HMI Status tag, used by faceplate
0036 IF #HMI.Status.Inactive THEN
0037     #HMI.Status.Status := 1;
0038     ;
0039 END_IF;
0040
0041 IF #HMI.Status.Activated THEN
0042     #HMI.Status.Status := 2;
0043     ;
0044 END_IF;
0045
0046 IF #HMI.Status.ActiveAlarm THEN
0047     #HMI.Status.Status := 3;
0048     ;
0049 END_IF;
0050
0051 IF #HMI.Status.Tripped THEN
0052     #HMI.Status.Status := 4;
0053     ;
0054 END_IF;
0055
0056 // Emergency stop activated
0057 IF #"E-Stop" THEN
0058     #HMI.Status.Status := 3;
0059     ;
0060 END_IF;
0061
0062 // Tripped by emergency stop
0063 IF NOT #"E-Stop" AND #MotorModuleTags."E-StopTripped" THEN
0064     #HMI.Status.Tripped := TRUE;
```

```
0065     #HMI.Status.ActiveAlarm := FALSE;
0066     #HMI.Status.Activated := FALSE;
0067     #HMI.Status.Inactive := FALSE;
0068     "Common Tracks_DB".RTOButtonAppearance := 1;
0069     ;
0070 END_IF;
0071
0072 // Reset tripped
0073 IF NOT #"E-Stop" AND "Common Tracks_DB".RTO THEN
0074     #HMI.Status.Inactive := TRUE;
0075     #HMI.Status.Activated := FALSE;
0076     #HMI.Status.Tripped := FALSE;
0077     #HMI.Status.ActiveAlarm := FALSE;
0078     ;
0079 END_IF;
0080
0081 // After an alarm reset of a fuse or CB alarm, make the RTO button on the HMI blue
0082 IF #MotorModuleTags.RTO AND NOT #MotorModuleTags.AlarmReset THEN
0083     "Common Tracks_DB".RTOButtonAppearance := 1;
0084     ;
0085 END_IF;
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
```

8.3 Appendix 3: Pallet Log Viewer

Pallet Log Viewer

	Name	Data type	Comment
1	Input		
2	PalletLogSize	Int	Number of Pallet entries in the DB PalletLogs
3	Output		
4	InOut		
5	Static		
6	LogSelectedByHMI_1	"ViewPalletLogHMI"	HMI buttons and the viewing array for pallet log 1
7	LogSelectedByHMI_2	"ViewPalletLogHMI"	HMI buttons and the viewing array for pallet log 2
8	SelectedLog_1	"PalletLog"	The selected Log 1 from PalletLogs DB2
9	SelectedLog_2	"PalletLog"	The selected Log 2 from PalletLogs DB2
10	BrowseForwardLog_1	R_TRIG	Detect positive signal edge upwards arrow button on HMI for log 1
11	BrowseBackwardLog_1	R_TRIG	Detect positive signal edge downwards arrow button on HMI for log 1
12	BrowseForwardLog_2	R_TRIG	Detect positive signal edge upwards arrow button on HMI for log 2
13	BrowseBackwardLog_2	R_TRIG	Detect positive signal edge downwards arrow button on HMI for log 2
14	PalletLogViewerTags	"PalletLogViewerTags"	Tags used for logic within the FB
15	Temp		
16	i	Int	Used by for loops
17	j	Int	Used by for loops

Figur 40: *Pallet Log Viewer blockgränssnitt.*

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
Back_1	Bool	false	Non-retain	True	True	True	False		Browse backwards through SelectedLog_1
Forward_2	Bool	false	Non-retain	True	True	True	False		Browse forwards through SelectedLog_2
Back_2	Bool	false	Non-retain	True	True	True	False		Browse backwards through SelectedLog_2
Index_1	Int	0	Non-retain	True	True	True	False		Used to add pallet entries from SelectedLog_1 to LogSelectedByHMI_1
Index_2	Int	0	Non-retain	True	True	True	False		Used to add pallet entries from SelectedLog_2 to LogSelectedByHMI_2
▼ Temp									
i	Int								Used by for loops
j	Int								Used by for loops
Constant									

```
0001 (*
0002 Check if the SelectedLog_1 viewed in the last scan cycle is the same as the one selected in the current or if the refresh button on the HMI is pressed.
0003 If a new log has been chosen or the refresh button is pressed, empty the LogSelectedByHMI_1 array before filling it with new values
0004 and reset the Integer containing the number of pages in SelectedLog_1.
0005 *)
0006 IF (#PalletLogViewerTags.PreviouslyViewed_1 <> #LogSelectedByHMI_1.PalletLogSelectedByHMI) OR #LogSelectedByHMI_1.Refresh THEN
0007   #LogSelectedByHMI_1.Browsing := FALSE;
0008   #LogSelectedByHMI_1.NbrOfPagesSelectedLog := 0;   //Reset the Integer containing the number of pages in the selected DB log
0009   FOR #i := 0 TO 9 DO
0010     #LogSelectedByHMI_1.Pallets[#i] := #PalletLogViewerTags.EmptyPalletEntry;
0011   END_FOR;
0012 END_IF;
0013
0014 (*
0015 Case branch to assign a pallet log from Trackunit PalletLogs [DB2] to SelectedLog_1.
0016 Which log to view is selected by pressing the info button of the desired trackunit on the main screen
0017  or by the drop-down menu on the screens "PalletLogs" & "TwoPalletLogs".
0018 *)
0019 CASE #LogSelectedByHMI_1.PalletLogSelectedByHMI OF
0020   1:  // Track 100 - Ingoing pallets
0021     IF NOT #LogSelectedByHMI_1.Browsing THEN
0022       #SelectedLog_1 := "Trackunit PalletLogs"."Track 100 - IngoingPallets";
0023     END_IF;
0024
0025   2:  // T-Track 115
0026     IF NOT #LogSelectedByHMI_1.Browsing THEN
0027       #SelectedLog_1 := "Trackunit PalletLogs"."T Track 115";
0028     END_IF;
0029
0030   3:  // Track 110
0031     IF NOT #LogSelectedByHMI_1.Browsing THEN
0032       #SelectedLog_1 := "Trackunit PalletLogs"."Track 110";
0033     END_IF;
0034
0035   4:  // Track 130
0036     IF NOT #LogSelectedByHMI_1.Browsing THEN
0037       #SelectedLog_1 := "Trackunit PalletLogs"."Track 130";
0038     END_IF;
0039
0040   5:  // Track 220
0041     IF NOT #LogSelectedByHMI_1.Browsing THEN
0042       #SelectedLog_1 := "Trackunit PalletLogs"."Track 220";
0043     END_IF;
0044
0045   6:  // Track 230
0046     IF NOT #LogSelectedByHMI_1.Browsing THEN
0047       #SelectedLog_1 := "Trackunit PalletLogs"."Track 230";
0048     END_IF;
0049
0050   7:  // Track 140 - Outgoing pallets station not A
0051     IF NOT #LogSelectedByHMI_1.Browsing THEN
0052       #SelectedLog_1 := "Trackunit PalletLogs"."Track 140 - OutgoingPallets_NotA";
0053     END_IF;
0054
0055   8:  // Track 240 - Outgoing pallets station A
0056     IF NOT #LogSelectedByHMI_1.Browsing THEN
0057       #SelectedLog_1 := "Trackunit PalletLogs"."Track 240 - OutgoingPallets_A";
0058     END_IF;
0059 END_CASE;
0060
0061 //Sort the SelectedLog_1 by pallet number in descending order, newest pallet entry at index [0].
0062 FOR #i := 0 TO (#PalletLogSize - 1) DO
```

```
0063    FOR #j := (#i + 1) TO (#PalletLogSize - 1) DO
0064      IF #SelectedLog_1.PalletEntry[#i].Nbr <  #SelectedLog_1.PalletEntry[#j].Nbr THEN
0065        #PalletLogViewerTags.TempPalletEntry := #SelectedLog_1.PalletEntry[#i];
0066        #SelectedLog_1.PalletEntry[#i] := #SelectedLog_1.PalletEntry[#j];
0067        #SelectedLog_1.PalletEntry[#j] := #PalletLogViewerTags.TempPalletEntry;
0068      END_IF;
0069    END_FOR;
0070  END_FOR;
0071
0072  // Calculate the number of pages in the SelectedLog_1, 10 entries per page.
0073  IF #SelectedLog_1.PalletEntry[0].Nbr = 0 THEN // The DB log is empty --> 0 pages.
0074    #LogSelectedByHMI_1.NbrOfPagesSelectedLog := 0;
0075  ELSIF #SelectedLog_1.PalletEntry[0].Nbr < 11 THEN // (Entries < 11) in the DB log --> 1 page.
0076    #LogSelectedByHMI_1.NbrOfPagesSelectedLog := 1;
0077  ELSIF #SelectedLog_1.PalletEntry[0].Nbr >= #PalletLogSize THEN //Calculates the number of pages when the DB log is
      full.
0078    #LogSelectedByHMI_1.NbrOfPagesSelectedLog := (#PalletLogSize / 10);
0079    IF (#PalletLogSize MOD 10) > 0 THEN
0080      #LogSelectedByHMI_1.NbrOfPagesSelectedLog := #LogSelectedByHMI_1.NbrOfPagesSelectedLog + 1;
0081    END_IF;
0082  ELSE  // Calculates the number of pages when (10 < entries < #PalletLogArraySize).
0083    #LogSelectedByHMI_1.NbrOfPagesSelectedLog := (#SelectedLog_1.PalletEntry[0].Nbr / 10);
0084    IF (#SelectedLog_1.PalletEntry[0].Nbr MOD 10) > 0 THEN
0085      #LogSelectedByHMI_1.NbrOfPagesSelectedLog := #LogSelectedByHMI_1.NbrOfPagesSelectedLog + 1;
0086    END_IF;
0087  END_IF;
0088
0089  // If Browsing mode for LogSelectedByHMI_1 isn't activated then Index = 9 to display the 10 newest pallet entries by
      default.
0090  IF NOT #LogSelectedByHMI_1.Browsing THEN
0091    #PalletLogViewerTags.Index_1 := 9;
0092  END_IF;
0093
0094  (*
0095  Browse backwards log 1
0096  Scan for positive signal edge on downwards arrow button on HMI screens "PalletLogs" & "TwoPalletLogs".
0097  When the button is pressed, the viewieng array displays 5 older pallet entries.
0098  Once the earliest entry in the logg is reached nothing happens when pressing the button.
0099  *)
0100  #BrowseBackwardLog_1(CLK := #LogSelectedByHMI_1.Backward,
0101             Q => #PalletLogViewerTags.Back_1);
0102  IF #PalletLogViewerTags.Back_1 AND #SelectedLog_1.PalletEntry[0].Nbr > 10 THEN // If (PalletEntry[0].Nbr < 10) nothing
      to browse backwards to.
0103    #LogSelectedByHMI_1.Browsing := TRUE;
0104    #PalletLogViewerTags.Index_1 := #PalletLogViewerTags.Index_1 + 5;
0105    IF #PalletLogViewerTags.Index_1 > (#PalletLogSize - 1) THEN // If (index > array size) -> index = last entry in ar-
      ray.
0106      #PalletLogViewerTags.Index_1 := (#PalletLogSize - 1);
0107    END_IF; // If Index is pointing to an empty entry decrease index until it points to the oldest entry.
0108    IF #SelectedLog_1.PalletEntry[#PalletLogViewerTags.Index_1].Nbr = 0 THEN
0109      FOR #i := 0 TO #PalletLogViewerTags.Index_1 DO
0110        IF #SelectedLog_1.PalletEntry[#PalletLogViewerTags.Index_1].Nbr = 0 THEN
0111          #PalletLogViewerTags.Index_1 := #PalletLogViewerTags.Index_1 - 1;
0112        END_IF;
0113      END_FOR;
0114    END_IF;
0115  END_IF;
0116
0117  (*
0118  Browse forwards log 1
0119  Scan for positive signal edge on upwards arrow button on HMI screen "PalletLogs".
0120  When the button is pressed, the viewieng array displays 5 more recent pallet entries.
0121  Once the most recent entry in the logg is reached and the button is pressed
0122  browsing becomes false.
0123  *)
0124  #BrowseForwardLog_1(CLK := #LogSelectedByHMI_1.Forward,
0125            Q => #PalletLogViewerTags.Forward_1);
0126  IF #PalletLogViewerTags.Forward_1 THEN
0127    // If the newest log entry isn't displayed in the HMI viewing array.
0128    IF #SelectedLog_1.PalletEntry[0].Nbr <> #LogSelectedByHMI_1.Pallets[0].Nbr THEN
0129      #PalletLogViewerTags.Index_1 := #PalletLogViewerTags.Index_1 - 5;
0130      // Index = 9 --> 10 newest pallet entries displayed in the HMI viewing array.
0131      IF #PalletLogViewerTags.Index_1 < 9 THEN
0132        #PalletLogViewerTags.Index_1 := 9;
0133      END_IF;
0134    // Newest log entry is displayed in the HMI viewing array, can't browse further forward.
0135    ELSIF #SelectedLog_1.PalletEntry[0].Nbr = #LogSelectedByHMI_1.Pallets[0].Nbr THEN
0136      #LogSelectedByHMI_1.Browsing := FALSE;
0137      RETURN;
0138    END_IF;
0139  END_IF;
0140
0141  (*
0142  Insert pallet entries into the HMI viewing array LogSelectedByHMI_1.Pallets.
0143  The 10 pallet entries displayed from the SelectedLog_1 is decided by the Index_1 value.
0144  *)
0145  FOR #i := #PalletLogViewerTags.Index_1 TO (#PalletLogViewerTags.Index_1 - 9) BY -1 DO
0146    #LogSelectedByHMI_1.Pallets[#i - (#PalletLogViewerTags.Index_1 - 9)] := #SelectedLog_1.PalletEntry[#i];
```

```
0147 END_FOR;
0148
0149 // Calculate which page numbers from SelectedLog_1 currently being viewed by LogSelectedByHMI_1.
0150 IF #LogSelectedByHMI_1.NbrOfPagesSelectedLog = 0 THEN // The DB log is empty --> 0 pages.
0151   #LogSelectedByHMI_1.ViewingPage1 := 0;
0152   #LogSelectedByHMI_1.ViewingPage2 := 0;
0153   #LogSelectedByHMI_1.PageNbrHyphen := '';
0154 ELSIF #LogSelectedByHMI_1.NbrOfPagesSelectedLog = 1 THEN // (Entries < 11) in the DB log --> 1 page.
0155   #LogSelectedByHMI_1.ViewingPage1 := 1;
0156   #LogSelectedByHMI_1.ViewingPage2 := 0;
0157   #LogSelectedByHMI_1.PageNbrHyphen := '';
0158 ELSIF #LogSelectedByHMI_1.NbrOfPagesSelectedLog > 1 THEN //Calculates the viewed page number when the DB log has more
      than one page.
0159   IF (#LogSelectedByHMI_1.Pallets[0].Nbr MOD 10) = 0 THEN  // If entry[0] has (pallet number % 10) = 0 the page nbr
      viewed is (pallet number / 10).
0160     #LogSelectedByHMI_1.ViewingPage1 := (#LogSelectedByHMI_1.Pallets[0].Nbr / 10);
0161     #LogSelectedByHMI_1.ViewingPage2 := 0;
0162     #LogSelectedByHMI_1.PageNbrHyphen := '';
0163   ELSIF (#LogSelectedByHMI_1.Pallets[0].Nbr MOD 10) <> 0 THEN
0164     #LogSelectedByHMI_1.ViewingPage2 := (#LogSelectedByHMI_1.Pallets[0].Nbr / 10);
0165     #LogSelectedByHMI_1.ViewingPage1 := #LogSelectedByHMI_1.ViewingPage2 + 1;
0166     #LogSelectedByHMI_1.PageNbrHyphen := '-';
0167   END_IF;
0168 END_IF;
0169
0170 #PalletLogViewerTags.PreviouslyViewed_1 := #LogSelectedByHMI_1.PalletLogSelectedByHMI; // Set the currently viewed Se-
      lectedLog_1 as PreviouslyViewed.
0171
0172 (*
0173 Check if the SelectedLog_2 viewed in the last scan cycle is the same as the one selected in the current or if the re-
      fresh button on the HMI is pressed.
0174 If a new log has been chosen or the refresh button is pressed then empty the LogSelectedByHMI_2 array before filling
      it with new values.
0175 *)
0176 IF (#PalletLogViewerTags.PreviouslyViewed_2 <> #LogSelectedByHMI_2.PalletLogSelectedByHMI) OR #LogSelectedByHMI_2.Re-
      fresh THEN
0177   #LogSelectedByHMI_2.Browsing := FALSE;
0178   #LogSelectedByHMI_2.NbrOfPagesSelectedLog := 0;   //Reset the Integer containing the number of pages in the selected
      DB log
0179   FOR #i := 0 TO 9 DO
0180     #LogSelectedByHMI_2.Pallets[#i] := #PalletLogViewerTags.EmptyPalletEntry;
0181   END_FOR;
0182 END_IF;
0183
0184 (*
0185 Case branch to assign a pallet log from Trackunit PalletLogs [DB2] to SelectedLog_2.
0186 Which log to view is selected by the drop-down menu on the screen "PalletLogs".
0187 *)
0188 CASE #LogSelectedByHMI_2.PalletLogSelectedByHMI OF
0189   0:  // When the user leaves the screen "TwoPalletLogs" empty the SelectedLog_2.
0190     FOR #i := 0 TO (#PalletLogSize - 1) DO
0191       #SelectedLog_2.PalletEntry[#i] := #PalletLogViewerTags.EmptyPalletEntry;
0192     END_FOR;
0193
0194   1:  // Track 100 - Ingoing pallets
0195     IF NOT #LogSelectedByHMI_2.Browsing THEN
0196       #SelectedLog_2 := "Trackunit PalletLogs"."Track 100 - IngoingPallets";
0197     END_IF;
0198
0199   2:  // T-Track 115
0200     IF NOT #LogSelectedByHMI_2.Browsing THEN
0201       #SelectedLog_2 := "Trackunit PalletLogs"."T Track 115";
0202     END_IF;
0203
0204   3:  // Track 110
0205     IF NOT #LogSelectedByHMI_2.Browsing THEN
0206       #SelectedLog_2 := "Trackunit PalletLogs"."Track 110";
0207     END_IF;
0208
0209   4:  // Track 130
0210     IF NOT #LogSelectedByHMI_2.Browsing THEN
0211       #SelectedLog_2 := "Trackunit PalletLogs"."Track 130";
0212     END_IF;
0213
0214   5:  // Track 220
0215     IF NOT #LogSelectedByHMI_2.Browsing THEN
0216       #SelectedLog_2 := "Trackunit PalletLogs"."Track 220";
0217     END_IF;
0218
0219   6:  // Track 230
0220     IF NOT #LogSelectedByHMI_2.Browsing THEN
0221       #SelectedLog_2 := "Trackunit PalletLogs"."Track 230";
0222     END_IF;
0223
0224   7:  // Track 140 - Outgoing pallets station not A
0225     IF NOT #LogSelectedByHMI_2.Browsing THEN
0226       #SelectedLog_2 := "Trackunit PalletLogs"."Track 140 - OutgoingPallets_NotA";
0227     END_IF;
```

```
0228
0229    8:  // Track 240 - Outgoing pallets station A
0230      IF NOT #LogSelectedByHMI_2.Browsing THEN
0231        #SelectedLog_2 := "Trackunit PalletLogs"."Track 240 - OutgoingPallets_A";
0232      END_IF;
0233  END_CASE;
0234
0235  //Sort the SelectedLog_2 by pallet number in descending order, newest pallet entry at index [0].
0236  FOR #i := 0 TO (#PalletLogSize - 1) DO
0237    FOR #j := (#i + 1) TO (#PalletLogSize - 1) DO
0238      IF #SelectedLog_2.PalletEntry[#i].Nbr < #SelectedLog_2.PalletEntry[#j].Nbr THEN
0239        #PalletLogViewerTags.TempPalletEntry := #SelectedLog_2.PalletEntry[#i];
0240        #SelectedLog_2.PalletEntry[#i] := #SelectedLog_2.PalletEntry[#j];
0241        #SelectedLog_2.PalletEntry[#j] := #PalletLogViewerTags.TempPalletEntry;
0242      END_IF;
0243    END_FOR;
0244  END_FOR;
0245
0246  // Calculate the number of pages in the SelectedLog_2, 10 entries per page.
0247  IF #SelectedLog_2.PalletEntry[0].Nbr = 0 THEN // The DB log is empty --> 0 pages.
0248      #LogSelectedByHMI_2.NbrOfPagesSelectedLog := 0;
0249  ELSIF #SelectedLog_2.PalletEntry[0].Nbr < 11 THEN // (Entries < 11) in the DB log --> 1 page.
0250      #LogSelectedByHMI_2.NbrOfPagesSelectedLog := 1;
0251  ELSIF #SelectedLog_2.PalletEntry[0].Nbr >= #PalletLogSize THEN //Calculates the number of pages when the DB log is
      full.
0252      #LogSelectedByHMI_2.NbrOfPagesSelectedLog := (#PalletLogSize / 10);
0253      IF (#PalletLogSize MOD 10) > 0 THEN
0254        #LogSelectedByHMI_2.NbrOfPagesSelectedLog := #LogSelectedByHMI_2.NbrOfPagesSelectedLog + 1;
0255      END_IF;
0256  ELSE  // Calculates the number of pages when (10 < entries < #PalletLogArraySize).
0257      #LogSelectedByHMI_2.NbrOfPagesSelectedLog := (#SelectedLog_2.PalletEntry[0].Nbr / 10);
0258      IF (#SelectedLog_2.PalletEntry[0].Nbr MOD 10) > 0 THEN
0259        #LogSelectedByHMI_2.NbrOfPagesSelectedLog := #LogSelectedByHMI_2.NbrOfPagesSelectedLog + 1;
0260      END_IF;
0261    END_IF;
0262
0263  // If Browsing mode for LogSelectedByHMI_2 isn't activated then Index = 9 to display the 10 newest pallet entries by
      default.
0264  IF NOT #LogSelectedByHMI_2.Browsing THEN
0265    #PalletLogViewerTags.Index_2 := 9;
0266  END_IF;
0267
0268  (* Browse backwards log 2
0269  Scan for positive signal edge on downwards arrow button on HMI screen "PalletLogs".
0270  When the button is pressed, the viewieng array displays 5 older pallet entries.
0271  Once the earliest entry in the logg is reached nothing happens when pressing the button.
0272  *)
0273  #BrowseBackwardLog_2(CLK:=#LogSelectedByHMI_2.Backward,
0274            Q=>#PalletLogViewerTags.Back_2);
0275  IF #PalletLogViewerTags.Back_2 AND #SelectedLog_2.PalletEntry[0].Nbr > 10 THEN // If (PalletEntry[0].Nbr < 10) there's
      nothing to browse backwards to.
0276    #LogSelectedByHMI_2.Browsing := TRUE;
0277    #PalletLogViewerTags.Index_2 := #PalletLogViewerTags.Index_2 + 5;
0278    IF #PalletLogViewerTags.Index_2 > (#PalletLogSize - 1) THEN // If (index > array size) then index = last entry in
      the array.
0279      #PalletLogViewerTags.Index_2 := (#PalletLogSize - 1);
0280    END_IF;
0281    IF #SelectedLog_2.PalletEntry[#PalletLogViewerTags.Index_2].Nbr = 0 THEN // If index is pointing to an empty entry
      decrease index until it points to the earliest entry.
0282      FOR #i := 0 TO #PalletLogViewerTags.Index_2 DO
0283        IF #SelectedLog_2.PalletEntry[#PalletLogViewerTags.Index_2].Nbr = 0 THEN
0284          #PalletLogViewerTags.Index_2 := #PalletLogViewerTags.Index_2 - 1;
0285        END_IF;
0286      END_FOR;
0287    END_IF;
0288  END_IF;
0289
0290  (* Browse forwards log 2
0291  Scan for positive signal edge on upwards arrow button on HMI screen "PalletLogs".
0292  When the button is pressed, the viewieng array displays 5 more recent pallet entries.
0293  Once the most recent entry in the logg is reached nothing happens when pressing the button.
0294  *)
0295  #BrowseForwardLog_2(CLK := #LogSelectedByHMI_2.Forward,
0296            Q => #PalletLogViewerTags.Forward_2);
0297  IF #PalletLogViewerTags.Forward_2 THEN
0298    IF #SelectedLog_2.PalletEntry[0].Nbr <> #LogSelectedByHMI_2.Pallets[0].Nbr THEN // If the newest log entry isn't dis-
      played in the HMI viewing array.
0299      #PalletLogViewerTags.Index_2 := #PalletLogViewerTags.Index_2 - 5;
0300      IF #PalletLogViewerTags.Index_2 < 9 THEN  // Index = 9 results in the 10 newest pallet entries being displayed in
      the HMI viewing array.
0301        #PalletLogViewerTags.Index_2 := 9;    // Therefore Index can't be smaller than 9.
0302      END_IF;
0303    ELSIF #SelectedLog_2.PalletEntry[0].Nbr = #LogSelectedByHMI_2.Pallets[0].Nbr THEN // Newest entry in the log is dis-
      played in the HMI viewing array, can't browse further forward.
0304      #LogSelectedByHMI_2.Browsing := FALSE;
0305      RETURN;
0306    END_IF;
0307  END_IF;
```

```
0308
0309 (*
0310 Insert 10 pallet entries into the LogSelectedByHMI_2 array.
0311 The 10 pallet entries displayed from the SelectedLog_2 is decided by the Index_2 value.
0312 *)
0313 FOR #i := #PalletLogViewerTags.Index_2 TO (#PalletLogViewerTags.Index_2 - 9) BY -1 DO
0314     #LogSelectedByHMI_2.Pallets[#i - (#PalletLogViewerTags.Index_2 - 9)] := #SelectedLog_2.PalletEntry[#i];
0315  END_FOR;
0316
0317 // Calculate which page numbers from SelectedLog_2 currently being viewed by LogSelectedByHMI_2.
0318 IF #LogSelectedByHMI_2.NbrOfPagesSelectedLog = 0 THEN // The DB log is empty --> 0 pages.
0319   #LogSelectedByHMI_2.ViewingPage1 := 0;
0320   #LogSelectedByHMI_2.ViewingPage2 := 0;
0321   #LogSelectedByHMI_2.PageNbrHyphen := '';
0322 ELSIF #LogSelectedByHMI_2.NbrOfPagesSelectedLog = 1 THEN // (Entries < 11) in the DB log --> 1 page.
0323   #LogSelectedByHMI_2.ViewingPage1 := 1;
0324   #LogSelectedByHMI_2.ViewingPage2 := 0;
0325   #LogSelectedByHMI_2.PageNbrHyphen := '';
0326 ELSIF #LogSelectedByHMI_2.NbrOfPagesSelectedLog > 1 THEN //Calculates the viewed of page number when the DB log has
      more than one page.
0327   IF (#LogSelectedByHMI_2.Pallets[0].Nbr MOD 10) = 0 THEN  // If the pallet entry [0] has (pallet number % 10) = 0 the
      page nbr viewed is (pallet number / 10)
0328     #LogSelectedByHMI_2.ViewingPage1 := (#LogSelectedByHMI_1.Pallets[0].Nbr / 10);
0329     #LogSelectedByHMI_2.ViewingPage2 := 0;
0330     #LogSelectedByHMI_2.PageNbrHyphen := '';
0331   ELSIF (#LogSelectedByHMI_2.Pallets[0].Nbr MOD 10) <> 0 THEN
0332     #LogSelectedByHMI_2.ViewingPage2 := (#LogSelectedByHMI_2.Pallets[0].Nbr / 10);
0333     #LogSelectedByHMI_2.ViewingPage1 := #LogSelectedByHMI_2.ViewingPage2 + 1;
0334     #LogSelectedByHMI_2.PageNbrHyphen := '-';
0335   END_IF;
0336 END_IF;
0337
0338 #PalletLogViewerTags.PreviouslyViewed_2 := #LogSelectedByHMI_2.PalletLogSelectedByHMI; // Set the currently viewed Se-
      lectedLog_2 as PreviouslyViewed.
0339
```

Symbol	Address	Type	Comment
"Trackunit PalletLogs"."T Track 115"		Block_UDT	Log of pallets on Trackunit 115
"Trackunit PalletLogs"."Track 100 - IngoingPallets"		Block_UDT	Log of pallets that entered the conveyor belt system
"Trackunit PalletLogs"."Track 110"		Block_UDT	Log of pallets on Trackunit 110
"Trackunit PalletLogs"."Track 130"		Block_UDT	Log of pallets on Trackunit 130
"Trackunit PalletLogs"."Track 140 - OutgoingPallets_NotA"		Block_UDT	Log of pallets that exited the conveyor belt system at station Not A
"Trackunit PalletLogs"."Track 220"		Block_UDT	Log of pallets on Trackunit 220
"Trackunit PalletLogs"."Track 230"		Block_UDT	Log of pallets on Trackunit 230
"Trackunit PalletLogs"."Track 240 - OutgoingPallets_A"		Block_UDT	Log of pallets that exited the conveyor belt system at station A
#BrowseBackwardLog_1		Multi_FB	Detect positive signal edge downwards arrow button on HMI for log 1
#BrowseBackwardLog_2		Multi_FB	Detect positive signal edge downwards arrow button on HMI for log 2
#BrowseForwardLog_1		Multi_FB	Detect positive signal edge upwards arrow button on HMI for log 1
#BrowseForwardLog_2		Multi_FB	Detect positive signal edge upwards arrow button on HMI for log 2
#i		Int	Used by for loops
#j		Int	Used by for loops
#LogSelectedByHMI_1.Backward		Bool	Button browse to older pallet entries
#LogSelectedByHMI_1.Browsing		Bool	In browsing mode when true
#LogSelectedByHMI_1.Forward		Bool	Button browse to more recent pallet entries
#LogSelectedByHMI_1.NbrOfPages-SelectedLog		Int	Number of pages in the selected DB log, 10 entries per page
#LogSelectedByHMI_1.PageNbrHyphen		String	A hyphen made visible when viewing 2 pages simultaneously
#LogSelectedByHMI_1.PalletLogSelectedByHMI		Int	Int used by Case statement to select which DB log to view
#LogSelectedByHMI_1.Pallets[0].Nbr		Int	The pallet running number
#LogSelectedByHMI_1.Pallets[*]		Block_UDT	HMI viewing array with pallet entries
#LogSelectedByHMI_1.Refresh		Bool	Button to leave browsing mode and fetch new pallet entries
#LogSelectedByHMI_1.ViewingPage1		Int	Page from DB log being viewed on the HMI
#LogSelectedByHMI_1.ViewingPage2		Int	Page from DB log being viewed on the HMI
#LogSelectedByHMI_2.Backward		Bool	Button browse to older pallet entries
#LogSelectedByHMI_2.Browsing		Bool	In browsing mode when true
#LogSelectedByHMI_2.Forward		Bool	Button browse to more recent pallet entries
#LogSelectedByHMI_2.NbrOfPages-SelectedLog		Int	Number of pages in the selected DB log, 10 entries per page
#LogSelectedByHMI_2.PageNbrHyphen		String	A hyphen made visible when viewing 2 pages simultaneously
#LogSelectedByHMI_2.PalletLogSelectedByHMI		Int	Int used by Case statement to select which DB log to view
#LogSelectedByHMI_2.Pallets[0].Nbr		Int	The pallet running number
#LogSelectedByHMI_2.Pallets[*]		Block_UDT	HMI viewing array with pallet entries
#LogSelectedByHMI_2.Refresh		Bool	Button to leave browsing mode and fetch new pallet entries
#LogSelectedByHMI_2.ViewingPage1		Int	Page from DB log being viewed on the HMI
#LogSelectedByHMI_2.ViewingPage2		Int	Page from DB log being viewed on the HMI
#PalletLogSize		Int	Number of Pallet entries in the DB PalletLogs
#PalletLogViewerTags.Back_1		Bool	Browse backwards through SelectedLog_1
#PalletLogViewerTags.Back_2		Bool	Browse backwards through SelectedLog_2
#PalletLogViewerTags.EmptyPalletEntry		Block_UDT	An empty pallet entry used to empty arrays of type "PalletEntry"
#PalletLogViewerTags.Forward_1		Bool	Browse forwards through SelectedLog_1
#PalletLogViewerTags.Forward_2		Bool	Browse forwards through SelectedLog_2