CODEN:LUTEDX/(TEIE-3121)/1-87/(2023)

Industrial Electrical Engineering and Automation

Utveckling av automationsstandard för spårbarhetshantering

Joel Hedlund
Isak Larsson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Utveckling av automationsstandard för spårbarhetshantering

Kandidatuppsats:
Joel Hedlund
Isak Larsson

LUNDS UNIVERSITET
Lunds Tekniska Högskola

Avdelningen för Industriell Elektroteknik och Automation
Lunds Tekniska Högskola, Lunds universitet
SE-221 00 Lund, Sweden

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

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

Printed in Sweden
Lunds universitet
Lund 2023

”Det har jag aldrig provat tidigare så det klarar jag helt säkert.”
- *Pippi Långstrump*

Sammanfattning

Denna rapport behandlar ett examensarbete som genomförts hos Backer AB i Sösdala. Backer AB är specialiserat på att utveckla och tillverka högkvalitativa uppvärmningslösningar för olika branscher.

Idag har det blivit allt viktigare för företag att använda spårbarhetssystem som har möjlighet att följa produkter och råvaror genom hela leveranskedjan. Krav på hållbarhet och socialt ansvarstagande, tillsammans med ökad efterfrågan från konsumenter och krav från myndigheter, har lett till ett ökat tryck på företag att implementera och förbättra sina spårbarhetssystem. Med utgångspunkt från detta har examensarbetet utvecklat ett spårbarhetssystem som kan vara användbart i flera delar av företagets produktion. Det utvecklade spårbarhetssystemet består av ett styrsystem, en etikettskrivare, en databas och ett Javaprogram.

Arbetet med att ta fram ett spårbarhetssystem har delats upp i flera olika steg. Det första steget bestod av att skapa funktioner till ett styrsystem bestående av en PLC och ett HMI. Dessa funktioner gjorde det möjligt att hantera märkning av produkter med hjälp av en etikettskrivare. Etikettskrivaren skriver ut streckkoder som individmärker en produkt. Med hjälp av HMI:et kunde en fabrikslinje simuleras där en produkt passerar genom hela produktionskedjan.

Det andra steget bestod av att undersöka det befintliga spårbarhetssystemet som Backer AB använder. Denna undersökning behandlade innehållet i spårbarhetssystemet för att kunna skapa en motsvarande databas. Därefter utvecklades databasen, vilket gjorde det möjligt att lagra information gällande en produkts tillverkningsprocess, där bland annat tester som en produkt genomgått och komponenter som använts i tillverkningen lagras.

Det tredje steget utgjordes av att utveckla ett Javaprogram som hanterar lagring av information i databasen och även hämtning av information från databasen.

Dessa ingående delar i spårbarhetssystemet resulterade i att ett fungerande och anpassningsbart spårbarhetssystem kunde utvecklas, vilket kan användas inom flera delar av företagets produktion.

Nyckelord: Automation, märkning av produkt, spårbarhet, Java, PLC, databas

Abstract

This report discusses a bachelor's thesis that was done at Backer AB in Sösdala. Backer AB specializes in developing and manufacturing high quality heating solutions for different branches.

In recent times it has become more important for corporations to use systems to be able to trace products and materials that have been used during production. Demands on durable products and social responsibilities, together with increased demands from consumers and authorities, has led companies to want to implement and improve their traceability systems. As a result of this, this bachelor's thesis has developed a traceability system that can be used in multiple areas of the company's production. The developed traceability system consists of a database, a label printer, a Java program and a control system.

Multiple steps have been taken to create the traceability system. The first step was to create functions for the control system that consists of a PLC and an HMI. These functions made it possible to create a unique individual marking of a product with the help of a label printer. With the help of an HMI a factory line could be simulated where a product undergoes its production.

The second step was to investigate the already existing traceability system that Backer AB uses. This investigation took the traceability system's content to create a corresponding database. The database was developed and made it possible to store information regarding a products' manufacturing process, showing tests and components that were used during the manufacturing process.

The third step was to develop a Java program that manages storing of information in the database and also retrieves information from the database.

These components of the traceability system resulted in the development of a functioning and adaptable traceability system, that can be used in several areas of the company's production.

Keywords: Automation, marking of a product, traceability, Java, PLC, database

Förord

Vi vill ta detta tillfälle i akt för att framföra ett stort tack till Backer AB för att ni gav oss möjligheten att utföra detta väldigt lärorika och roliga examensarbete hos er. Vi har känt oss otroligt bra omhändertagna och vi vill således tacka all personal för ert fina välkomnande och bemötande.

Ett extra stort tack till vår guide, lunchkamrat och handledare Hans-Göran Künkel för all din hjälp som lett oss på rätt väg under examensarbetet. Utan din vägledning hade det inte varit möjligt att genomföra detta examensarbete.

Ytterligare tack till vår handledare Mats Lilja som guidat oss genom uppsatsarbetet och gett oss värdefull konstruktiv kritik. Vi vill även tacka dig för din fina undervisning under vår tid på LTH, Campus Helsingborg.

Slutligen vill vi bara säga att vi under dessa veckor har lärt oss enormt mycket om hur ingenjörsyrket fungerar i verkligheten. Examensarbetet har verkligen stärkt vår motivation att fortsätta jobba inom elektroteknik och automation.

Terminologi

API	”Application Programming Interface”, är en mjukvarukomponent som låter applikationer kommunicera och utbyta data med varandra på ett standardiserat sätt.
Code-128	Streckkod som kan koda de första 128 första tecken i ASCII-kodningen.
EAN-13	”European Article Number”, är en streckkod som består av 13 siffror. Används främst för att märka förpackningar.
EOL	”End-Of-Line”, är en term som används för att beskriva den sista stationen eller momentet i en fabrikslinje.
HMI	”Human-Machine-Interface”, är ett gränssnitt mellan maskin och människa som sammanbinder en operatör och en maskin.
JDBC	”Java Database Connectivity”, är en API som gör det möjligt för ett Javaprogram att hantera databaser.
MES-modul	”Manufacturing Execution System”-modul, är en hårdvara som kan hantera databaser. Modulen är utvecklad av Mitsubishi och kan användas ihop med vissa av Mitsubishis PLC-modeller.
PLC	”Programmable Logic Controller”, är ett programmerbart styrsystem som främst används för att styra tillverkningsprocesser av olika slag.
SSMS	”SQL Server Management Studio”, är en databashanteringsmjukvara utvecklad av Microsoft som används för att hantera SQL Server databaser.
ST	”Strukturerad Text”, är ett programmeringsspråk designat för PLC-programmering.
ZPL	”Zebra Programming Language”, är ett programmeringsspråk som används för att skapa etiketter och streckkoder för utskrift.

Innehållsförteckning

Figurförteckning

1. Inledning
Detta kapitel beskriver bakgrund, syfte, mål- och problemformulering samt avgränsningar för examensarbetet.

1.1 Bakgrund
Detta avsnitt beskriver företaget som examensarbetet utförts hos, generell teori gällande spårbarhet samt hur spårbarheten sker på företaget idag.

1.1.1 Företaget
Företaget som detta examensarbete utförts på är Backer AB i Sösdala. Backer AB grundades år 1949 och är specialiserade på tillverkning av produkter och system för elektrisk uppvärmning. Företaget är huvudsätet för Backergruppen som idag består av över 70 bolag som tillsammans har över 10000 anställda och gör Backergruppen till den främsta aktören inom sin bransch.

1.1.2 Spårbarhet
Spårbarhet är en viktig faktor inom många olika branscher och sammanhang. Begreppet spårbarhet syftar på att kunna synliggöra historik, händelser och platser för olika objekt i en leveranskedja. Detta ger möjlighet att bland annat kunna spåra en produkts förflyttning från punkt A till punkt B, varje process den passerar och var produkten befinner sig. Denna information ska kunna lagras för att vid ett senare tillfälle kunna redovisas för alla involverade parter [1]. Fördelarna som ett välimplementerat spårbarhetssystem bär med sig är bland annat att flödena i tillverkningsprocessen av en produkt kan förbättras men även att ett ökat förtroende uppnås hos konsumenter genom transparent information om de aktuella produkterna.

1.1.3 Spårbarhet på företaget
På företaget sker spårbarheten av produkter idag mer eller mindre analogt där dokumentering av produkten sker i pappersformat. Denna dokumentation innehåller bland annat produktens tillverkningsorder och anteckningar på tester som utförts under tillverkningen. Sedan följer denna dokumentation med produkten ända fram till sista steget i produktionen. Dokumentationen av produkten är inte individuell utan detta dokument omsluter flertalet produkter som ingår i samma tillverkningsorder. Under produktens färd genom produktionen bockas varje steg av med handburna terminaler. Med hjälp av dessa terminaler sparar företaget ned vem som jobbat med respektive del i produktionen i det digitala affärssystemet. När produkten levererats sparas detta dokument undan i ett arkiv och sparas där i två år. Företaget sparar även undan vem som jobbat med respektive jobb i produktionen digitalt i deras affärssystem. Tillsammans med dokumentet och affärssystemet kan en helhetsbild fås av produktens tillverkning. Det huvudsakliga problemet som företaget vill lösa är att hanteringen kring spårbarheten inte är standardiserad och att man därför vill underlätta och effektivisera arbetet när ny produktionsutrustning tas fram.

Lösningarna i detta examensarbete kommer att ge Backer AB ett standardiserat och modulbaserat system som kan säkerställa spårbarheten i produktionen. Detta examensarbete kommer även att bestå av att ta fram en metodik, det vill säga en allmän beskrivning för hur systemet ska fungera, för implementering av systemet i företagets produktion.

1.2 Syfte

Examensarbetet syftar till att ta fram en modell för ett standardiserat spårbarhetssystem för produktionen inom Backer AB. Syftet är vidare att ta fram de metoder som är bäst lämpade för Backer AB gällande etikettutskrifter, inhämtning av data och lagring av data. Detta görs genom att ta fram standardiserade funktioner och mallar som kan återanvändas i produktionsutrustningen. Det förväntade resultatet är att med hjälp av ett standardiserat bibliotek med funktionsblock och en databas kunna standardisera hanteringen gällande spårbarhet.

1.3 Målformulering

Målet med examensarbetet är att utveckla en modell för ett spårbarhetssystem som i ett senare skede är möjligt att implementera i produktionen på Backer AB. Under examensarbetet ska program skapas vars huvudfunktion är att möjliggöra spårbarheten i produktionen, där dessa program ska kunna användas var som helst i produktionen. För att möjliggöra spårbarheten i produktionen kommer utgångspunkten för detta examensarbete bestå av att skapa ett bibliotek med funktionsblock till en PLC och en SQL-databas.

1.4 Problemformulering

Följande frågor skulle besvaras under examensarbetet:

1. Hur ska spårbarhetssystemet utformas för att kunna vara användbart i hela produktionskedjan?
2. Hur ska funktionsblocken utformas för att kunna återanvändas?
3. Hur ska produkterna individmärkas?
4. Vilken information ska sparas i databasen?
5. Hur ska information om en produkt sparas i en databas?
6. Hur ska information om en produkt hämtas ur en databas?

1.5 Motivering av examensarbetet

Under stora delar av utbildningen har automationsteknik och programmering stått i centrum. Det kändes därför naturligt att rikta in examensarbetet mot dessa ämnen och att försöka hitta ett företag som byggt sin verksamhet just runt dem. Det visade sig att Backer AB uppfyllde dessa kriterier och vid en förfrågan om att göra examensarbetet hos dem ställde de sig positiva. Inriktningen på examensarbetet grundar sig i ett förslag på upplägg som Backer AB presenterade och som omfattar just dessa ämnen, det vill säga automationsteknik och programmering. Examensarbetet känns som en stor möjlighet att utveckla kunskaperna inom

bland annat databas- och PLC-programmering samt kommunikation mellan hårdvara och mjukvara.

Från företagets perspektiv finns många fördelar med att ta fram ett spårbarhetssystem inom ramen för ett examensarbete. Den främsta fördelen är att kunna uppnå full spårbarhet i alla led i produktionskedjan. Spårbarheten innebär att om ett fel uppstår någonstans i produktionen, så ska alla produkter i en produktionsbatch kunna spåras, vilket i slutändan leder till att slutprodukternas kvalité kan säkerställas. Examensarbetet innebär således att processen runt spårbarhet blir standardiserad i alla delar av företagets produktion.

Ur ett samhällsperspektiv finns många fördelar, bland annat ger ett spårbarhetssystem trygghet i form av en ökad kundsäkerhet. Ett spårbarhetssystem ger möjlighet för företag och privatpersoner att rapportera felaktigheter hos en produkt till producenten som sedan kan spåra alla produkter i samma batch. Detta i sig ger ett ökat förtroende mellan kunder och företag eftersom eventuella fel som uppstår hos en produkt kan återrapporteras till leverantören och samtidigt åtgärdas.

1.6 Avgränsningar

I detta avsnitt definieras avgränsningarna för examensarbetet.

- Programmen är enbart utvecklade för Mitsubishi PLC:er.
- Streckkoderna är framtagna med hjälp av ZPL-språket, därför kommer utskrift av streckkoderna enbart fungera med skrivare som är kompatibla med detta språk.
- Databashanteraren är enbart kompatibel med programmeringsspråket Java.

1.7 Resurser

För att genomföra examensarbetet kommer resurser av olika slag krävas. De resurser som använts presenteras i följande punktlista:

- Mjukvara:
    - GX Works 3
    - GT Works 3
    - Microsoft SQL Server
    - SQL Server Management Studio
    - Eclipse
    - TSC Console
- Hårdvara:
    - TSC etikettskrivare
    - Mitsubishi styrsystem
    - Dator - lånas ut av Backer AB

De resurser som använts i examensarbetet beskrivs närmare i nästa kapitel, teknisk bakgrund.

2. Teknisk bakgrund

I detta kapitel beskrivs hårdvara, mjukvara samt tekniker som använts för att genomföra examensarbetet.

2.1 Hårdvara

I detta avsnitt förklaras vilken hårdvara som använts i examensarbetet.

2.1.1 PLC

Inom detta examensarbete har stora delar bestått av att designa och utveckla program till en PLC av modellen ”Mitsubishi FX5U”, där PLC står för ”Programmable Logic Controller” vilket är ett programmerbart styrsystem. En PLC är en dator som är specifikt anpassad för automation och som exempelvis används för att styra processer av olika slag inom industrin [2].

2.1.2 HMI

HMI som står för ”Human-Machine Interface” och är en slags instrumentpanel. Ett HMI möjliggör bland annat för en användare att kunna kommunicera med en maskin, ett datorprogram eller ett system [3]. Inom detta examensarbete har ett HMI använts för att testa framtagna funktioner och program, samt för att simulera en viss tillverkningsprocess. Med hjälp av ett programmeringsverktyg har HMI:et kunnat anpassas efter behov.

HMI:et som använts i detta examensarbete var en Mitsubishi Electric GT2104-RTBD [4].

2.1.3 Etikettskrivare

Etikettskrivaren som har använts i detta examensarbete är av modellen ”TSC TX200” och är en etikettskrivare lanserad av företaget Taiwan Semiconductor där den är en av sex modeller i den så kallade TX-serien. Användaren kan ansluta till skrivaren med hjälp av WiFi eller Bluetooth, det är även möjligt att ansluta till skrivaren med ethernet, USB eller med seriell koppling [5]. I detta examensarbete har skrivaren varit konfigurerad och ansluten till ett lokalt nätverk med hjälp av ethernet.

2.2 Mjukvara

I detta avsnitt förklaras vilken mjukvara som använts i examensarbetet.

2.2.1 Microsoft SQL Server och SQL Server Management Studio

Microsoft SQL Server är en databasprogramvara utvecklad av Microsoft. I detta examensarbete har denna databashanterare möjliggjort att kunna skapa databaser med tillhörande tabeller. Databasen som användes under examensarbetet var SQL Server 2022 [6].

För att hantera databasen och dess innehåll har programvaran ”SQL Server Management Studio”, förkortat SSMS, använts. SSMS tillåter användaren att mer detaljrikt hantera

databaser, där exempelvis användaren kan hämta specifik information med hjälp av ”SQL-queries”, skapa nya tabeller samt att ta bort information och lägga till ny information [7].

2.2.3 GX Works 3

GX Works 3 är en utvecklingsmiljö skapad av Mitsubishi Electric som används för att utveckla programvara till Mitsubishi PLC:er. GX Works 3 används specifikt för PLC-hårdvara i serierna ”MELSEC IQ-R” och ”MELSEC IQ-F”. Grundfunktionerna i GX Works 3 är att skapa program, ställa in rätt parametrar för de olika modulerna som ska användas, samt läsa och skriva data till och från CPU-modulen.

GX Works 3 gör det även möjligt för användaren att kunna se värden i realtid, debugga program och köra diagnostikfunktioner. GX Works 3 stödjer IEC 61131–3 standarden, det vill säga att program kan skrivas med ladderlogik, instruktionslistor, funktionsblock, strukturerad text och sekventiella funktionsdiagram [8]. Dessa språk kan användas samtidigt i samma program. I detta examensarbete har GX Works 3 använts för att programmera PLC:n, där funktionsblock och program har utvecklats.

2.2.4 GT Works 3

GT Works 3 är en utvecklingsmiljö utvecklad av Mitsubishi Electric som är specifikt anpassad för att designa HMI-program till Mitsubishis olika HMI:er. GT Works 3 gör det möjligt för användaren att lägga in funktioner i form av bland annat knappar, grafer, text – och numerisk inmatning [9]. I detta examensarbete har GT Works 3 främst använts för att kunna simulera inmatningar i fabrikslinjen från HMI:n till PLC:n.

2.2.5 TSC Console

”TSC Console” är en mjukvara utvecklad av TSC Printers som är anpassad för att hantera TSC-skrivare. TSC Console gör det möjligt att konfigurera skrivare, exempelvis att bestämma skrivarens kommunikationsprotokoll i form av ethernet, WiFi och så vidare. TSC Console gör det även möjligt att kalibrera utskriften baserat på etiketternas mått, men också att kunna hämta information om en skrivare. Exempelvis kan antalet etiketter som den anslutna skrivaren skrivit ut presenteras [10]. Denna mjukvara har främst använts för att bestämma skrivarens kommunikationsmedium, i detta examensarbete har detta bestämts till att vara ethernet.

2.3 Zebra Programming Language

Zebra Programming Language, förkortat ZPL, är ett programmeringsspråk framtaget av Zebra Technologies som primärt används för att designa etiketter. ZPL-språket innehåller en uppsättning kommandon som låter användaren designa etikettens beståndsdelar i form av text, former, streckkoder och bilder för att senare kunna skriva ut dessa.
Generellt börjar alltid kommandona med antingen ett ^ eller ~, och är en eller två bokstäver långt.

^XA: Säger åt programmet var etiketten börjar. Skrivs alltid längst upp.

^XZ: Säger åt programmet var etiketten slutar. Skrivs alltid längst ned.

^FO X, Y: Detta kommando används för att bestämma ett objekts position på etiketten. X-värdet bestämmer positionen för objektet i antalet pixlar från den vänstra kanten på etiketten. Y-värdet i sin tur bestämmer positionen för objektet i antalet pixlar från den översta kanten på etiketten.

^A F, X, Y: Detta kommando sköter textformatet i form av textfont, fonthöjd respektive fontbredd. Exempelvis ger **^A 0,40,40** en text med fonten Helvetica som är 40 pixlar hög samt 40 pixlar bred.

^FD: Detta kommando betyder ”Field Data”, på svenska fältdata, och följs av en parameter, denna parameter kan exempelvis vara en text-sträng eller en sifferkombination.

^FS: Detta kommando används oftast i kombination med **^FD** och säger till programmet var fältdatans innehåll avslutas.

^BY w, r, h: Detta kommando används främst för att ändra bredden på en streckkod med hjälp av parametrarna **w**, **r** och **h**. Dessa parametrar sköter bredden, förhållandet mellan de breda och de smala linjerna respektive höjden.

För att skapa streckkoder finns flera olika metoder. Nedan följer ett exempel:

^BE o, h, f, g: Detta kommando skapar en ”EAN-13” streckkod, som främst används för att märka konsumentförpackningar [11]. Efter kommandot **^BE** följer fyra olika parametrar, **o**, **h**, **f** och **g**. Orienteringen kännetecknas av **o** och säger om streckkoden ska vara vriden ett antal grader eller inte, exempelvis kan **o** sättas till ”N” för att vara vriden noll grader eller till ”R” som roterar streckkoden 90 grader. Vidare bestäms höjden på streckkoden i antalet pixlar i ett intervall på 1 till 32000 pixlar. De två sista parametrarna **f** och **g** bestämmer om den mänskligt läsbara texten skall skrivas ut eller inte respektive om texten skall skrivas ovanför streckkoden. Dessa parametrar bestäms av ”Y” för ”yes” samt ”N” för ”no”.

För att skriva ut en streckkod av typen EAN-13 med noll graders rotation och höjden 100 pixlar, där läsbar text ska ingå och där denna text ska skrivas under streckkoden skrivs följande kommando: **^BEN, 100, Y, N**

För att sedan skapa en fullständig etikett används dessa kommandon i en struktur som presenteras i figur 1 nedan.

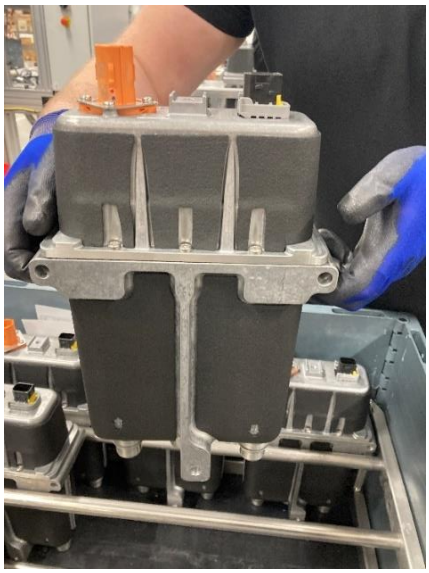
Figur 1: Exempel för att skapa en etikett med hjälp av ZPL-kommandon.

2.4 Fabrikslinjen som utgör basen för spårbarhetssystemet

Den tillverkningsprocess som spårbarhetssytemet skulle behandla valdes ut av företaget och utgörs av en fabrikslinje som tillverkar batteriuppvärmare till lastbilar av olika märken. Tillverkningen av dessa produkter sker i flera olika moment i flera olika stationer. Figur 2 visar en del av fabrikslinjen där produkterna tillverkas. Denna linje är under utveckling och i framtiden kommer linjen att vara uppkopplad till ett styrsystem som exempelvis ska sköta in- och utsignaler till och från linjen.

Figur 2: Del av fabrikslinjen

Huvudmålet för detta examensarbete är främst att ta fram en allmän metod för att kunna spåra dessa produkter, som visas i figur 3, både under och efter produktionen, men även lägga grunden för att kunna implementera spårbarhet i andra delar av fabriken.



Figur 3: Färdig komponent som tillverkats i fabrikslinjen

3. Metod

I detta kapitel beskrivs arbetsprocessen för examensarbetet.

3.1 Förberedelser

I detta avsnitt presenteras de olika faserna som uppstarten av examensarbetet bestod av samt det arbete som utfördes under varje fas.

3.1.1 Planering och förstudier

Under uppstarten av examensarbetet spenderades en stor del av tiden med att planera samt studera relevant material. Eftersom utgångspunkten för examensarbetet var att använda Mitsubishi-hårdvara så lades till en början stort fokus på att studera datablad och manualer. Under arbetet samlades information in om tekniska detaljer i befintlig hårdvara och mjukvara. Denna information hämtades bland annat från produktens egna hemsida och direkt från verktyg som fanns tillgängliga i mjukvaran. Denna inhämtning av information gjordes främst för att bekanta sig med den befintliga utrustningen, men även för att undersöka vilka funktioner som fanns tillgängliga.

Under denna tid hölls även flera möten med handledare och annan berörd personal på företaget. Här presenterades detaljer kring produktionen på företaget, genomgång av programvaror genomfördes och det gjordes även gemensamma rundvandringar i fabriken.

3.1.2 Systemdesign

Efter att grundläggande fakta om produktionen och andra tekniska detaljer samlats in erhölls en ökad förståelse för hur ett system skulle kunna utformas. Med hjälp av detta togs en hårdvarulista fram som beskrev vilken hårdvara som behövdes för examensarbetet. När denna hårdvarulista godkänts av företaget kunde komponenterna hämtas ut. Därefter byggdes en testrigg bestående av en Mitsubishi FX5U PLC, en HMI av typen Mitsubishi GT2000, en TSC TX200 etikettskrivare samt en nätverksswitch av modellen Netgear ProSafe GS108.

Testriggens huvudsyfte var att kunna testa systemet offline, det vill säga utanför produktionslinjen och företagsnätverket. HMI:n används ej vid slutgiltig implementation utan används här för att simulera riktiga händelser i form utav knapptryck. Med hjälp av testriggen kontrollerades hur de olika hårdvarorna kunde kommunicera med varandra över ethernet.

3.2 Metodik

Detta avsnitt beskriver de valda metoderna som användes för att besvara frågorna i problemformuleringen i avsnitt 1.4.

3.2.1 Framtagande av ett spårbarhetssystem

Eftersom huvuduppgiften för examensarbetet var att ta fram ett spårbarhetssystem togs en inledande beskrivning fram för hur detta system kunde se ut. Eftersom fabrikslinjen som beskrevs i kapitel 2.4 stod i fokus under examensarbetet var denna linje utgångspunkten i framtagandet av beskrivningen. För att uppnå detta gjordes flödesscheman för hur spårbarhetssystemet i sin helhet kan fungera i en produktionslinje.

Eftersom detta examensarbete även bestod av att utveckla ett modulbaserat system bestämdes det att grunderna till spårbarhetssystemet initialt skulle bestå av tre funktioner. I samverkan med handledarna på företaget togs väldefinierade beskrivningar fram för de olika funktionerna som skulle ingå i spårbarhetssystemet. Den första funktionen skulle hantera etikettutskrift från styrsystemet till en ansluten skrivare. Den andra funktionen skulle hantera databasen, där denna funktion både ska kunna hämta och lägga in information i databasen. Den tredje funktionen skulle hantera en etikettläsare, där denna funktion ska kunna hämta information från en streckkod.

Denna beskrivning för spårbarhetssystemet med de ingående funktionerna användes senare för att kunna skapa de funktioner och program som gör det möjligt att spåra en produkt.

3.2.2 Utveckling av återanvändningsbara funktionsblock

Efter framtagandet av en allmän beskrivning för hur spårbarhetssystemet skulle fungera, togs ytterligare beskrivningar fram, i detta fall i form av flödesscheman för de tre ingående funktionerna. I samverkan med handledarna på företaget analyserades dessa flödesscheman, där bland annat innehåll och annat relevant diskuterades.

Efter att dessa allmänna beskrivningar var godkända kunde utvecklingen av funktionerna påbörjas. Prioriteringen här var att funktionerna skulle vara återanvändningsbara, vilket innebär att programmen utformats så att modifikationer med relativ enkelhet kan utföras. Variabelnamn, in- och utgångar samt funktioner namngavs i programmen så att det är tydligt vad en funktion gör och vilken funktion varje variabel är knuten till.

3.2.3 Individmärkning av en produkt

När väl grundfunktionerna var implementerade påbörjades framtagandet av metoden för att individmärka en produkt. Individmärkningen är en fundamental del av systemet som gör det möjligt att både kunna spåra en specifik produkt genom produktionen samt i ett senare syfte kunna gå tillbaka till en produkts historik ifall företaget får en reklamation av produkten.

För att kunna individmärka produkten har en etikettskrivare av modellen TSC TX200 använts. Innan etikettskrivarens funktion kunde programmeras konfigurerades denna med hjälp av TSC Console där bland annat kommunikationsmediet konfigurerades, i detta fall valdes

kommunikationsmediet till ethernet med en fast IP-adress och portnummer. Genom att sedan följa manualer och internetbaserade guider kring anslutningar från PLC till externa enheter kunde anslutningen till skrivaren med hjälp av PLC-programmeringsmjukvaran upprättas. Därefter programmerades grundfunktionen, i detta fall i form av en etikettutskrift när användaren trycker på en knapp på HMI:et. Det viktigaste här var att säkerställa så att individmärkningen av en produkt verkligen är unik och inte kan blandas ihop med en annan produkt.

3.2.4 Framtagande av databas

För att kunna veta vilken information som ska sparas i databasen undersöktes detaljer kring hur nuvarande spårbarhet i produktionen går till, därefter kunde grunderna för en databas skapas. Prioriteringen här var att information gällande en produkt som ska sparas måste följa det system som företaget redan har och möjliggöra så att data med enkelhet kan hämtas från databasen. För att enkelt kunna hämta information ur databasen valdes här att skapa tre tabeller som hanterar olika delar av produktens tillverkning.

3.2.5 Lagring av information i databas

När valet av vilken information som skulle sparas i databasen stod klart var nästa steg att reda ut hur denna information skulle sparas i databasen. Till en början utforskades vilka möjligheter som fanns tillgängliga och hur detta skulle kunna tänkas gå till. Eftersom utgångspunkten för examensarbetet var att skapa funktionsblock i GX Works 3 för att skicka information till databasen blev detta således första alternativet. Det andra alternativet var att skapa en direkt kommunikation mellan databasen och PLC:n med hjälp av en extern hårdvarumodul, i detta fall en så kallad ”MES-modul”. Det tredje alternativet var att utveckla en så kallad ”mellanhand”, i detta fall ett Java-program som ansluts till databasen via ”Java Database Connectivity”, vilket är en API enbart utvecklad för detta syfte. De olika möjligheterna utforskades och i slutändan valdes det alternativ som innebär att utveckla en mellanhand mellan PLC:n och databasen, i detta fall ett Javaprogram. Detta beslut grundade sig i problem med de två första alternativen. Det första problemet som uppstod var att efter utveckling och testning av det första alternativet uppstod problem med kommunikationen mellan databasen och PLC:n. Problemet som uppstod var att PLC:n inte kunde bibehålla anslutningen till databasen och ingen information kunde skickas. Detta problem beskrivs närmare i avsnitt 4.6.1. Det andra problemet var att efter det andra alternativet hade undersökts visade det sig att inköpspriset var för högt och leveranstiden var för lång för de nödvändiga komponenterna.

3.2.6 Hämtning av information ur en databas

När väl information kunde sparas i en databas var nästa steg att göra det möjligt att hämta information ur databasen. För att kunna hämta information ur en databas fanns ett antal möjligheter. Ett av de alternativen som togs fram var att bygga ut Javaprogrammet så att information kunde presenteras i ett grafiskt gränssnitt, där användaren kan söka efter en streckkod för att således kunna presentera information som tillhör denna produkt. Det andra alternativet var att inte göra ett grafiskt gränssnitt utan sköta hämtningen av information direkt via databashanteringsmjukvaran ”SSMS”. Båda alternativen ansågs vara lämpade för

examensarbetet, där information snabbt kan hämtas med hjälp av Javaprogrammet och mer specifika SQL-kommandon kan köras i databashanteringsmjukvaran. Här utvecklades ett enkelt grafiskt gränssnitt där användaren kan söka på en streckkod, vilket leder till att information hämtas ur databasen och presenteras.

3.3 Simulering

Eftersom produktionslinjen är under utveckling fanns ett designdokument av linjen tillgängligt. I detta dokument kunde tekniska detaljer utläsas och även vad som är tänkt att utföras rent praktiskt i varje station. Tillverkningsprocessen består av 10 stationer. Exempelvis utförs tester i varje station där dessa tester genomförs av en operatör och sedan ska operatören trycka på en knapp som skickar en signal till ett styrsystem. Denna signal kan antingen vara att testet var ”OK” eller ”NOK” (det vill säga ”Not OK”).

I HMI:et kunde de olika stationerna simuleras, inklusive de tester som utförs i varje station. Här gjordes även en simulering av en skanner, detta för att kunna bestämma grundläggande detaljer kring systemets uppbyggnad men även för att senare kunna simulera tillverkningsprocessen ytterligare. Dessa simulerade knappar och skannrar möjliggjorde att simulering av hela fabrikslinjen kunde utföras.

Utvecklingen av simuleringen skedde parallellt med de problemställningar som presenterades i avsnitt 3.2.

3.4 Kommunikation

Kommunikationen mellan företaget och författarna innefattade till en början en del möten där företaget informerade om vilka önskemål som fanns gällande uppbyggnaden av det slutgiltiga systemet. Därefter hade författarna möten med handledaren på företaget för att stämma av hur arbetet fortlöpte.

Författarna hade även kontakt med andra anställda på företaget som bidrog med kunskap gällande utförandet av examensarbetet.

Författarna hade ett gemensamt kontor under hela examensarbetet vilket innebar att kommunikationen sinsemellan skedde kontinuerligt under alla moment i arbetet.

3.5 Källkritik

Samtliga källor som hämtats under examensarbetet anses vara pålitliga. Detta eftersom källorna hämtats från produkttillverkarens egen hemsida, eller från företag som är experter inom det sökta området.

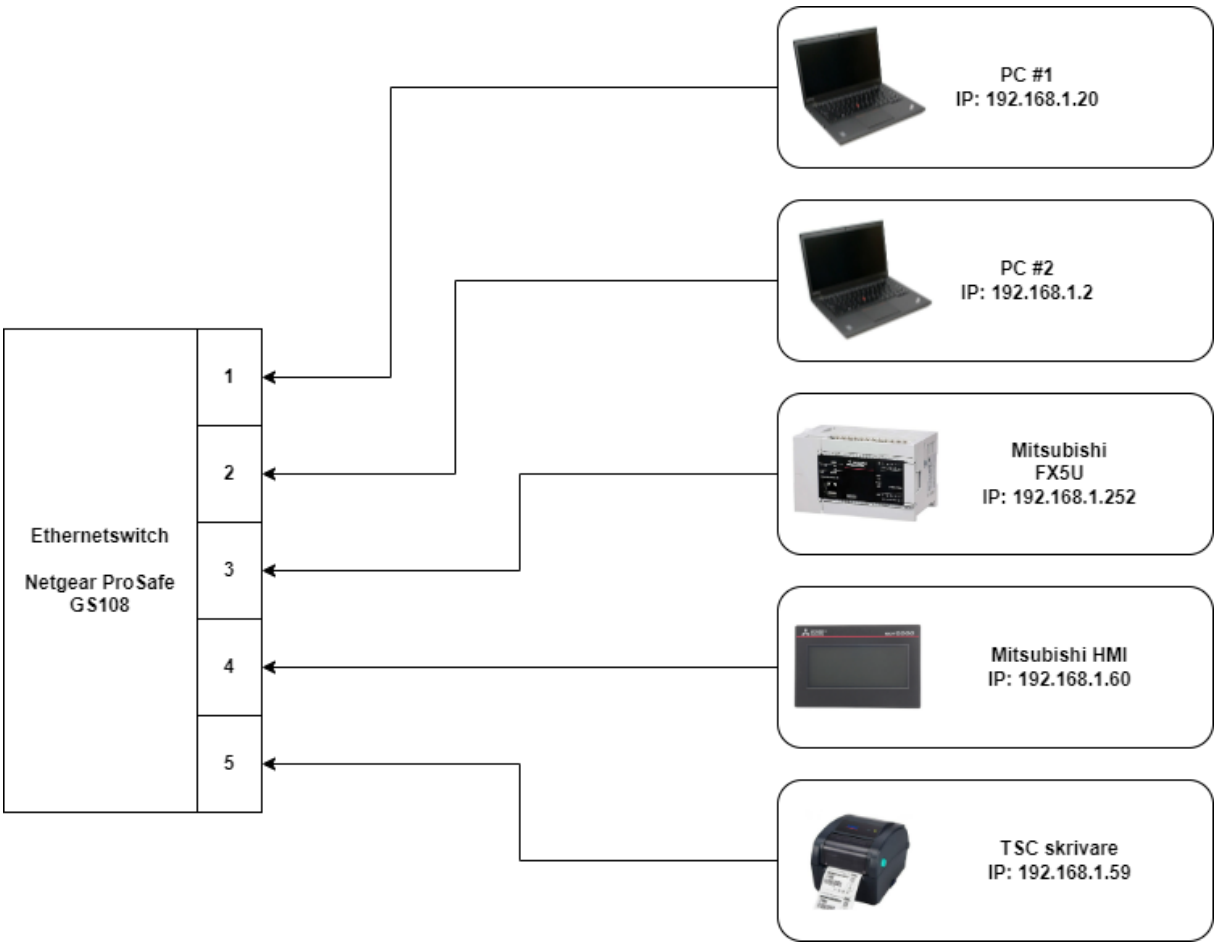
4. Analys

Detta kapitel beskriver utförandet av examensarbetet merdetaljerat, hur det utfördes och vilka val som gjorts. Detta kapitel kommer även att behandla problem som uppstått och hur dessa problem lösts.

4.1 Systemdesign

Efter att hårdvara hämtats ut och testriggen hade byggts upp var nästa steg var att konfigurera hårdvaran så att alla komponenter hamnar på samma lokala nätverk. Detta för att de olika hårdvarukomponenterna skulle kunna kommunicera med varandra. Konfigurationen av hårdvaran gjordes kontinuerligt allt eftersom programmering av respektive hårdvara påbörjats. Under konfigurationen av hårdvarans nätverksadress valdes här att adresserna ska bestämmas enligt formatet *192.168.1.XYZ*, där *XYZ* är en unik identifierare för varje komponent i nätverket. När IP-adresserna hade bestämts visade det sig fungera som tänkt, då alla enheter kunde kommunicera med varandra.

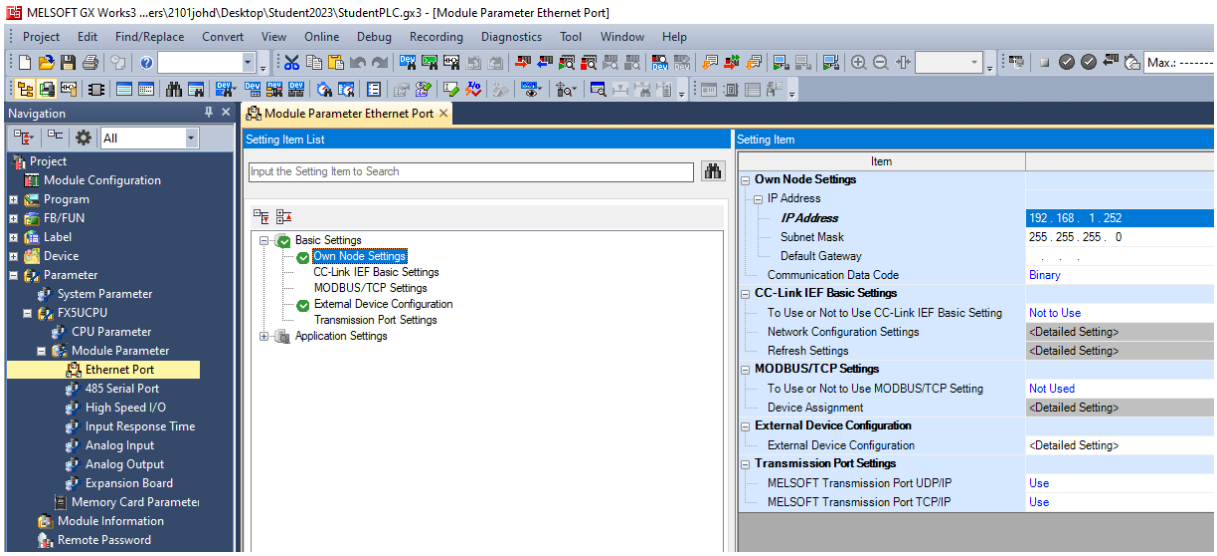
Figur 4 nedan visar testriggens slutgiltiga uppbyggnad samt de olika hårdvarornas fasta IP-adresser. Hur dessa konfigurerades presenteras i senare avsnitt i detta kapitel.



Figur 4: Testriggens slutgiltiga uppbyggnad

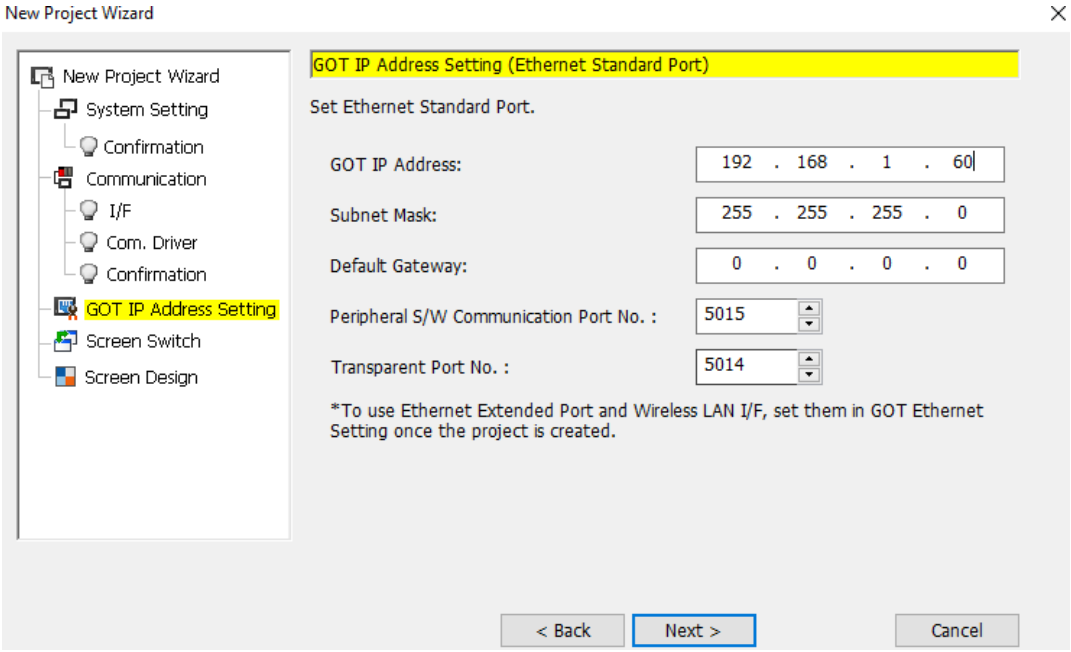
4.1.1 Konfiguration av PLC och HMI

För att konfigurera PLC:n så att denna hamnar i ett nätverk där all hårdvara kan kommunicera sinsemellan, så programmerades IP-adressen i mjukvaran GX Works 3. Här valdes PLC:ns IP-adress till 192.168.1.252. Fönstret där IP-adressen konfigurerades visas i figur 5.



Figur 5: Fönstret i programmeringsmjukvaran där IP-adressen för PLC:n konfigureras

Konfigurationen av HMI:n utförs när ett nytt projekt skapas i GT Designer 3. Därefter följdes instruktionerna ända fram tills dess att skärmen i figur 6 visas. Där konfigurerades IP-adressen till 192.168.1.60, även detta för att kunna kommunicera med annan hårdvara i nätverket, men främst för att kunna kommunicera med PLC:n.



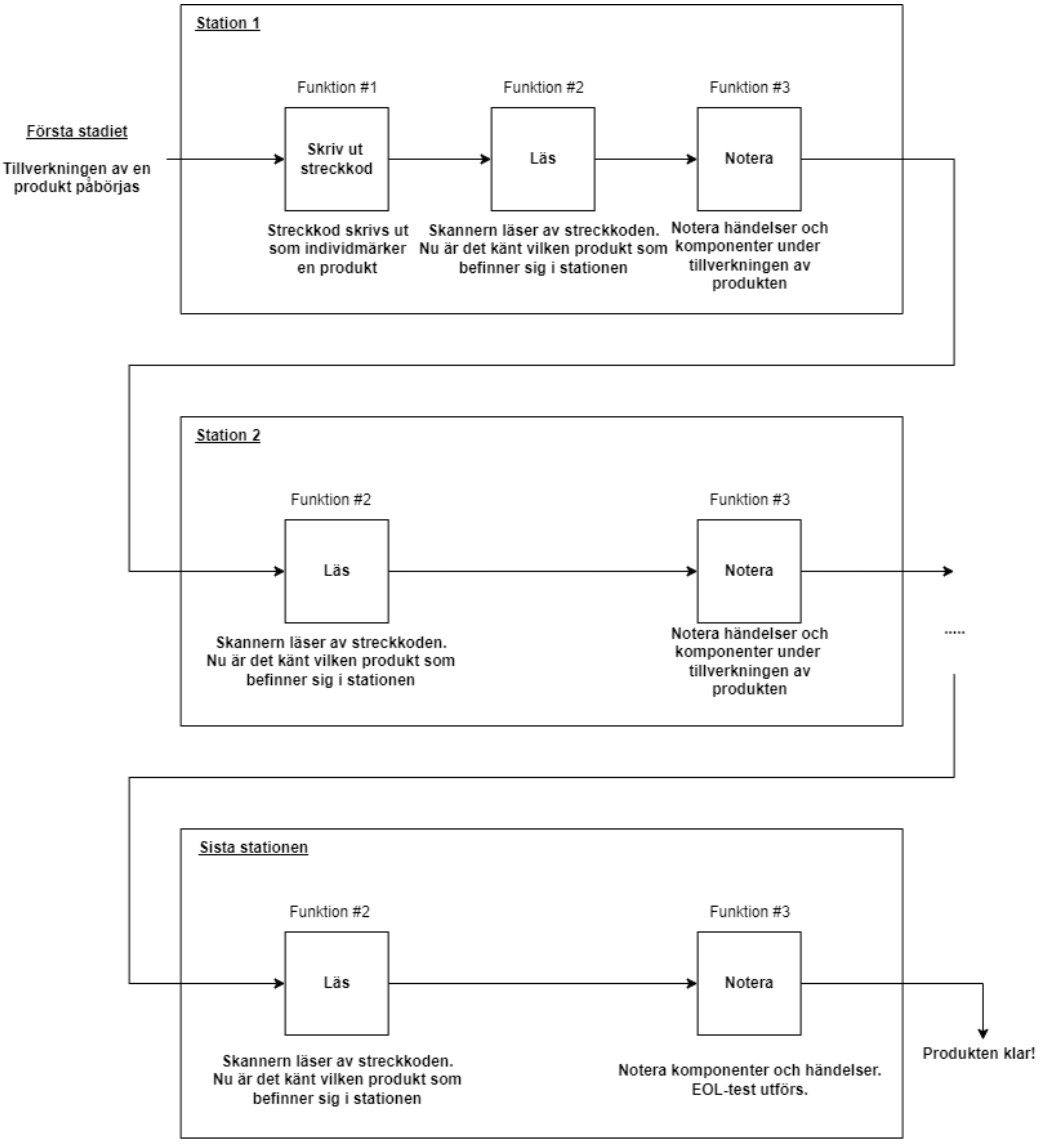
Figur 6: Fönstret under konfigurationen av HMI:n där IP-adressen bestäms

4.2 Framtagande av ett spårbarhetssystem

För att ta fram en allmän metod och beskrivning för ett spårbarhetssystem, vilket utgör basen för detta examensarbete, skapades två flödesscheman, där det ena beskriver den allmänna metoden för spårbarhet och det andra beskriver metoden för spårbarheten för en individuell produkt.

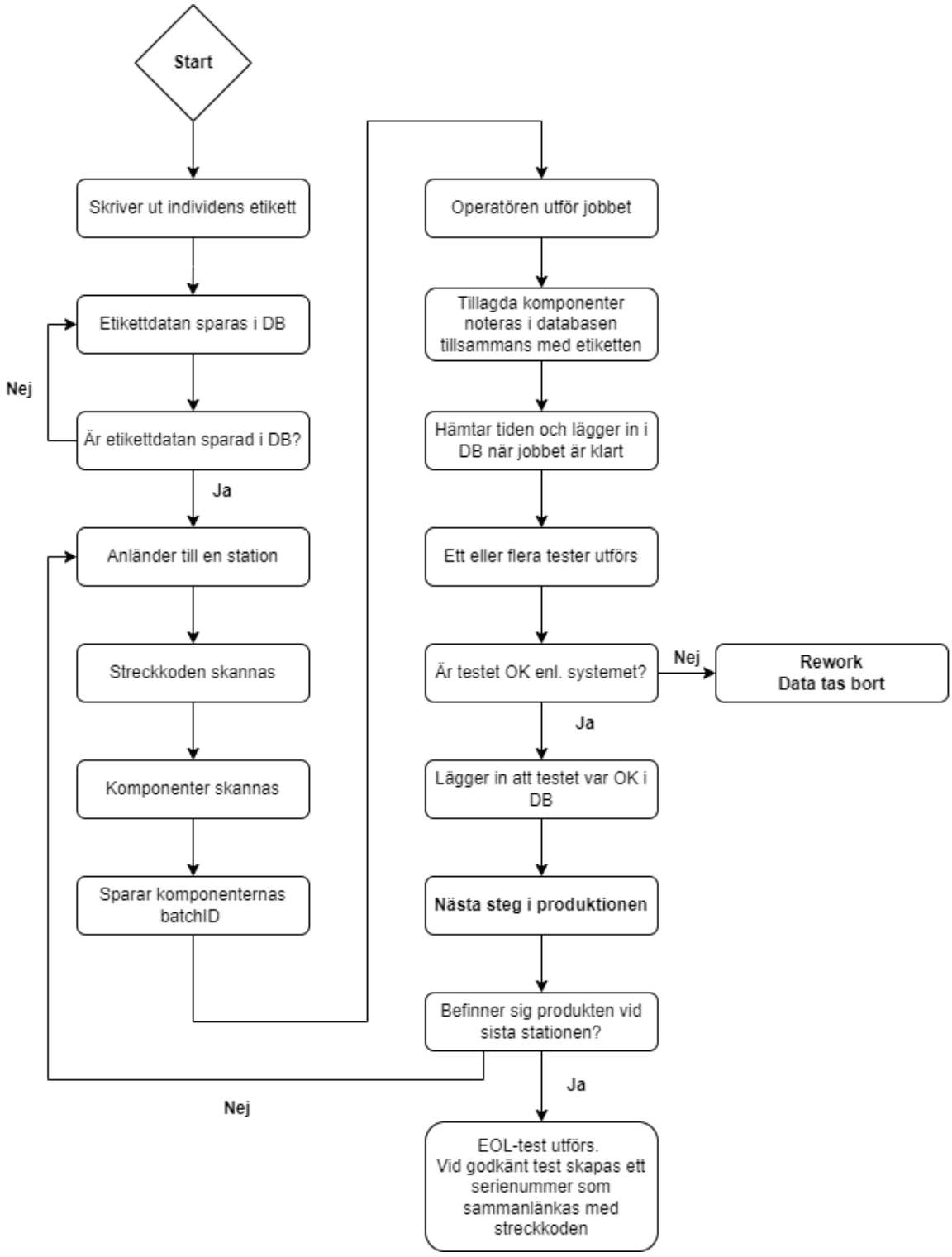
4.2.1 Generell uppbyggnad av ett spårbarhetssystem

För att bygga upp ett spårbarhetssystem bestämdes först de funktioner som skulle ingå. De funktioner som spårbarhetssystemet skulle bestå av bestämdes till att vara etikettutskrift, etikettläsare och möjlighet att notera relevant information i en databas. Dessa funktioner valdes i samverkan med handledarna på företaget. Detta innebar att etikettskrivaren skulle skriva ut en etikett med en streckkod som senare läses av etikettläsaren och som därefter noteras i databasen. Därefter valdes det att göra ett flödesschema som beskrev detta system. Detta flödesschema presenteras i figur 7. Denna beskrivning utgjorde grunden för de program och funktioner som beskrivs senare i detta kapitel.



Figur 7: Uppbyggnad av spårbarhetssystemet som består av en etikettskrivare, skannrar och en databas

För att beskriva hur den allmänna metoden för spårbarhet specifikt tillämpas i den aktuella fabrikslinjen togs flödesschemat i figur 8 fram. Denna metod beskriver alla ingående moment som ska utföras i de olika stationerna och vilka funktioner som ska användas under varje moment. Denna beskrivning togs fram för att fungera som en bas för det arbete som utfördes under examensarbetet.



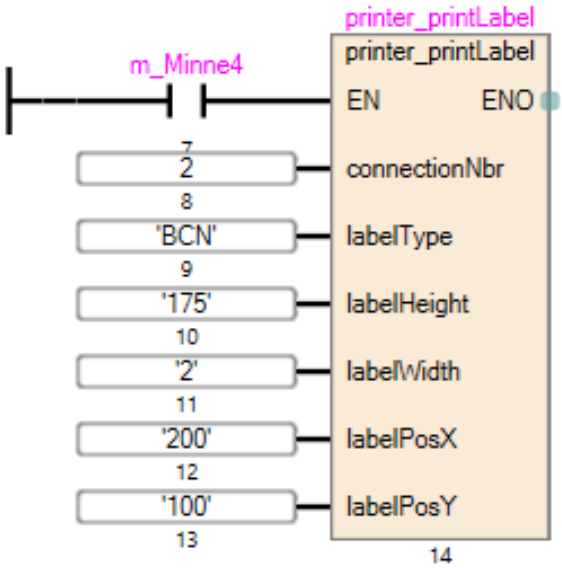
Figur 8: Flödesschema för spårbarhetssystemet i detalj för en enskild produkt

4.3 Utveckling av återanvändningsbara funktionsblock

Under uppstarten av utvecklingen av de olika funktioner som bestämts i samarbete med företaget var prioriteringen att dessa ska kunna återanvändas. Med återanvändning avses att kunna återanvända dessa funktionsblock i nya applikationer. Exempelvis när ny produktionsutrustning ska tas fram.

För att skapa funktionsblock som kan återanvändas krävs det att anpassningsbarhet och tydlighet i funktionsblockens uppbyggnad prioriteras. Detta innebär att variabelnamn samt in- och utgångar namnges för att sedan kunna knyta dessa till en funktion, men även underlätta för framtida modifikation. För att kunna återanvända program och funktioner är det även viktigt att dessa är dokumenterade, exempelvis i form av kommentarer i utvecklingsmiljön och beskrivningar för funktioner i form av flödesscheman. Därför har även detta prioriterats under examensarbetet.

De funktioner som skulle skapas skulle hantera en etikettskrivare, en databas och etikettläsare där alla funktioner kan modifieras efter behov med relativ enkelhet. I figur 9 nedan visas en av funktionerna som skapats med utgångspunkt att kunna återanvändas. Detta funktionsblock har sex ingångar där olika typer av inställningar kan utföras. Exempelvis är det möjligt att skapa nya streckkoder med andra mått och utseenden. Detta gör att en utvecklare inte nödvändigtvis behöver ändra innehållet i funktionsblocken, i form av kod eller andra funktionsblock, vilket möjliggör att på ett relativt smidigt sätt kunna implementera dessa i nya applikationer.



Figur 9: Funktionsblocket som hanterar utskrift av streckkod

4.4 Individmärkning av en produkt

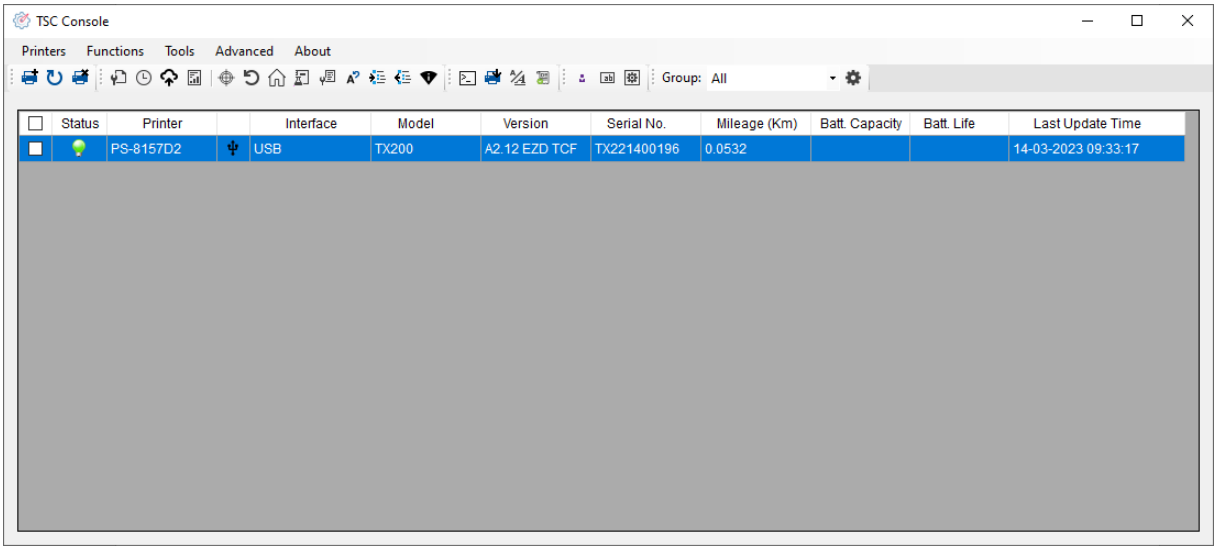
Arbetet för att ta fram en funktion som kan individmärka en produkt har delats upp i flera olika steg. Dessa steg presenteras i följande avsnitt.

4.4.1 Konfiguration av etikettskrivare

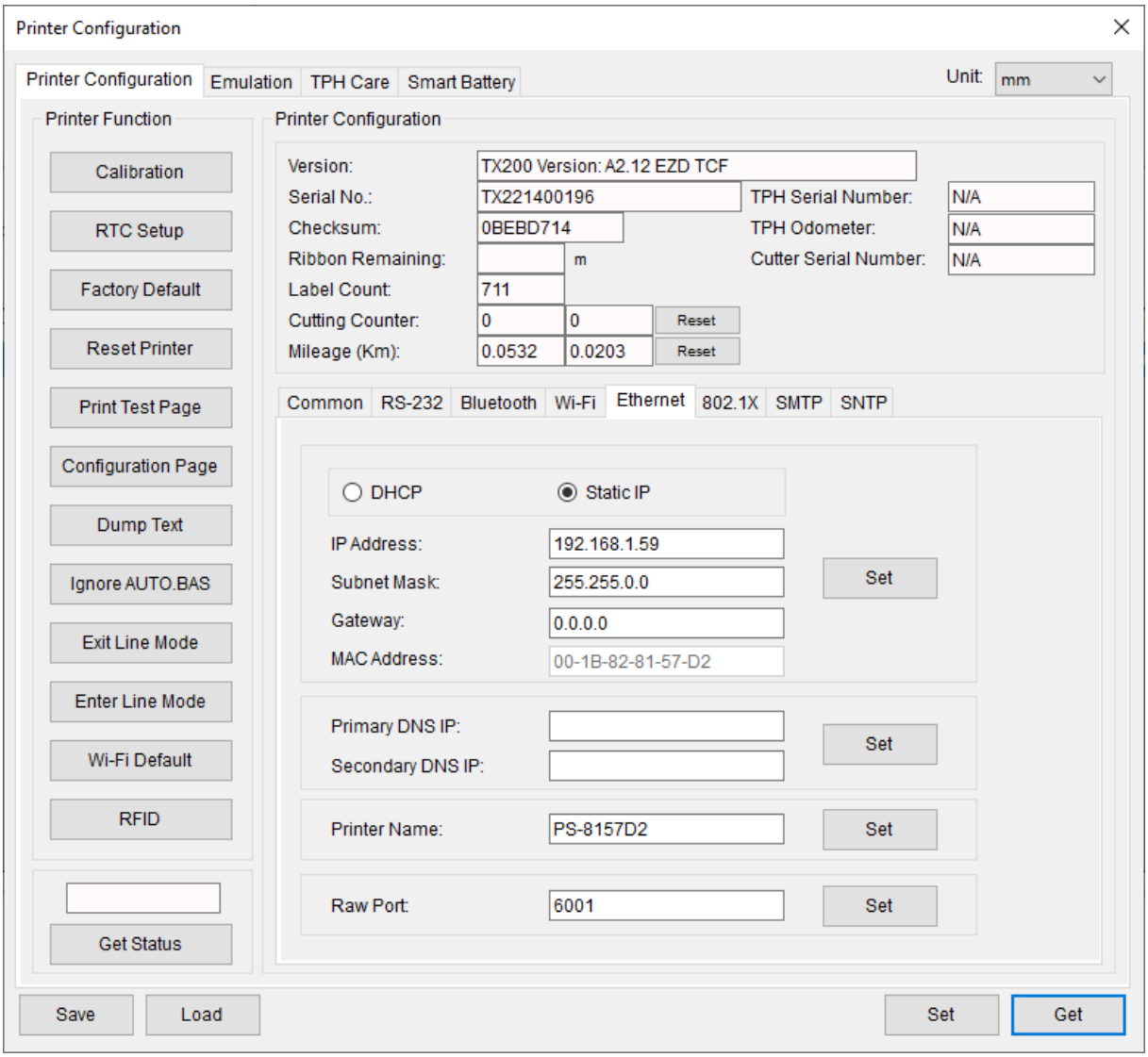
I det första skedet av framtagandet av metoden för att kunna individmärka produkten konfigurerades skrivaren med hjälp av TSC Console. Eftersom de olika hårdvarukomponenterna skulle ha möjlighet att kommunicera med varandra i ett lokalt nätverk valdes här att bestämma en fast IP-adress och portnummer.

För att kunna konfigurera skrivaren krävdes det först att skrivaren anslöts med hjälp av en USB A till USB B-kabel. När etikettskrivaren upptäcktes av TSC Console kunde skrivarens konfiguration bestämmas. I detta examensarbete har skrivaren konfigurerats till att använda ethernet där andra enheter kan kommunicera med skrivaren, främst tack vare att IP-adressen och portnumret bestämts. När detta gjorts visade det sig att kommunikationen mellan enheterna fungerade som planerat.

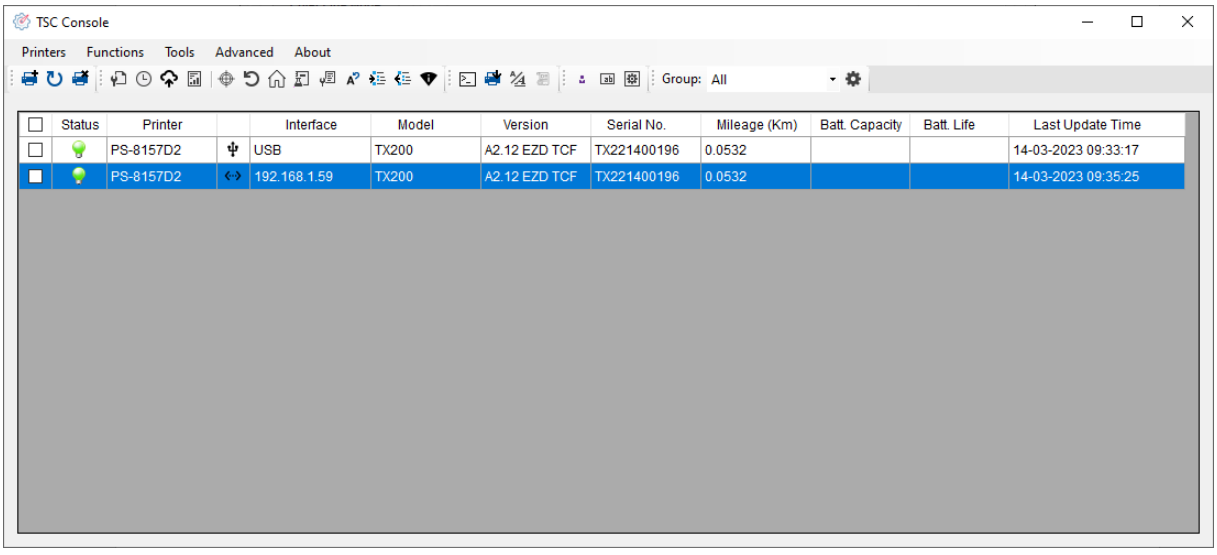
Figur 10–12 visar TSC Console programmet och vilka steg som utförts för att konfigurera skrivaren.



Figur 10: Fönstret i TSC Console när skrivaren lagts till med USB-kabeln



Figur 11: Fönster där skrivarens ethernet-inställningar konfigureras

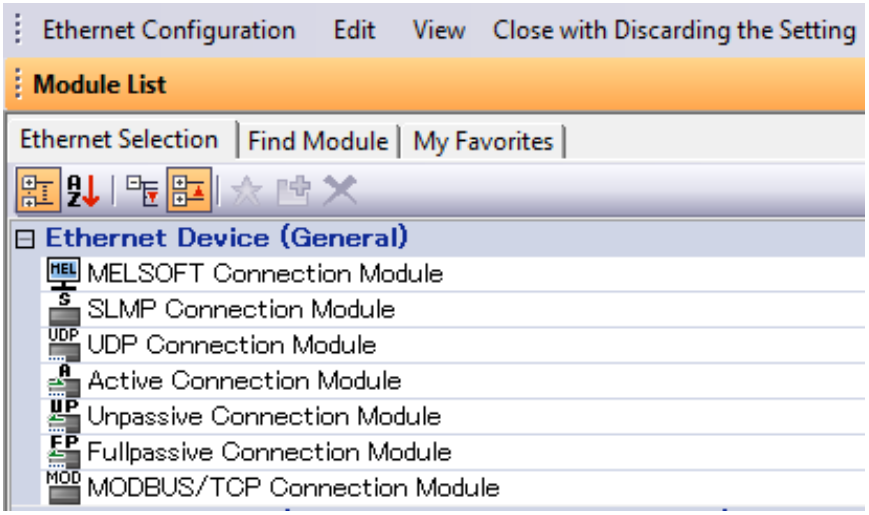


Figur 12: Fönstret när konfigurationen av skrivaren slutförts

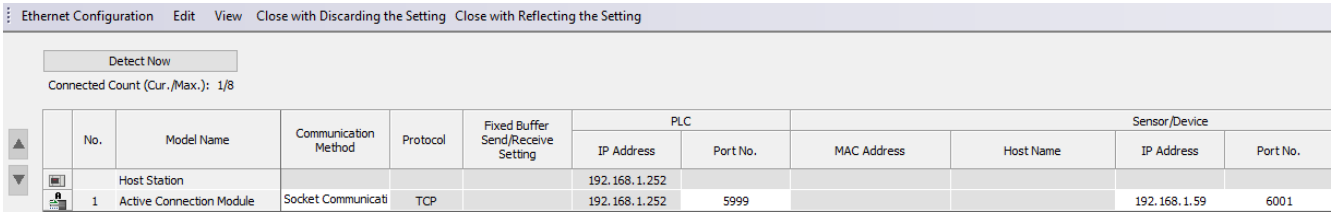
4.4.2 Anslutning mellan PLC och skrivare

När väl skrivaren, HMI:n och PLC:n var konfigurerade med respektive IP-adresser var nästa steg att ansluta PLC:n till skrivaren. För att kunna göra detta genomgicks flera steg. Under förstudierna upptäcktes det att det fanns möjlighet att konfigurera olika typer av kommunikationsvägar mellan PLC och så kallade ”externa enheter”, det vill säga enheter som befinner sig i samma lokala nätverk som PLC:n. För att öppna kommunikationsvägen mellan PLC och skrivaren konfigurerades detta i GX Works 3 via ”external device configuration” i ethernetinställningarna. Här valdes modulen ”Active Connection Module” då denna använder kommunikationsprotokollet TCP, vilket är kompatibelt med skrivaren. Figur 13 presenterar de olika modulerna som finns tillgängliga för kommunikation mellan PLC:n och externa enheter via ethernet. I denna modul konfigurerades dels PLC:ns portnummer och dels IP-adressen samt portnumret till skrivaren som bestämts i föregående steg, detta moment presenteras i figur 14.

Allmänt gäller det att med hjälp av en ethernetmodul kan en specifik uppkoppling till en komponent upprättas. Detta innebär att om fler komponenter ska anslutas krävs det att flera moduler används och att denna modul programmeras individuellt.

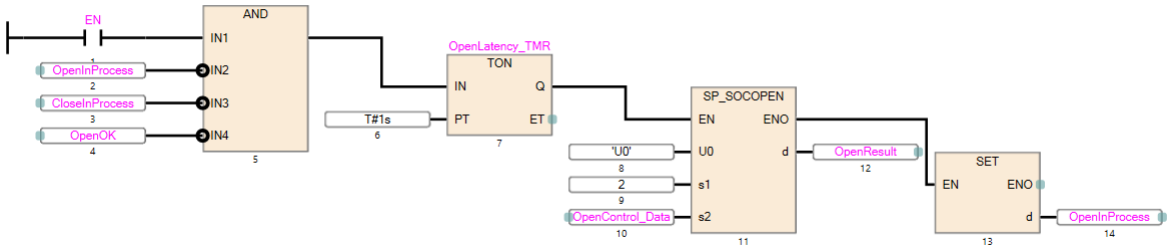


Figur 13: De olika ethernetmodulerna som finns tillgängliga i programmeringsmjukvaran



Figur 14: Fönstret när konfigurationen av ethernetmodulen slutförts, i detta fall en ”Active Connection Module” som använder TCP-protokollet

Efter att kommunikationsvägen mellan PLC:n och skrivaren öppnats kunde programmeringen av kommunikationen påbörjas. Med hjälp av funktionsblock kunde kommunikationen öppnas mellan skrivaren och PLC:n, kodexemplet som visas i figur 15 utför öppningen av kanalen när användaren trycker på en knapp på HMI:et. Detta visade sig fungera precis som tänkt då PLC:n bibehåller anslutningen till skrivaren fram till dess att data skickas över kanalen. Därefter kan en ny anslutning upprättas för att sedan skicka ny information.



Figur 15: Funktionsblocken i programmeringsmjukvaran som gör det möjligt att ansluta till skrivaren

4.4.3 Utskrift av streckkod från PLC

När en anslutning upprättats från PLC:n till etikettskrivaren var nästa steg att programmera en funktion som skickar information om etikettutskrift till skrivaren.

För att kunna skriva ut streckkoder krävs det först och främst att en streckkod med hjälp av ZPL-programmeringsspråket skapas. Under förstudierna hämtades information in om hur streckkoder skapas och designas, men även vilka typer av streckkoder som bäst lämpar sig för detta examensarbete. Här togs två alternativ fram, ett alternativ var att använda sig av streckkoden ”EAN-13” och det andra alternativet var att använda streckkoden ”Code 128”. I figur 16 och figur 17 nedan visas de två olika streckkoderna.

Figur 16: EAN-13 streckkod

Figur 17: Code 128 streckkod

Här gjordes en grundlig jämförelse mellan de två alternativen och slutligen föll valet på att använda streckkoden ”Code 128”. Detta främst på grund av att denna typ av streckkod är mer anpassningsbar och dynamisk än ”EAN-13” streckkoden. Detta innebär bland annat att ”Code 128” streckkoden anpassar sin bredd automatiskt beroende på vad som streckkoden innehåller. Detta ansågs passa bäst för utvecklingen av spårbarhetssystemet då streckkoden behöver innehålla en unik identifierare som individmärker produkten, som i sin tur ska kunna anpassas efter behov. Exempelvis kan streckkoden vara uppbyggd efter formatet ”*ÅÅÅÅMMDDXYZ*” eller ”*ÅÅMMDDX*”. Eftersom de olika formaten är olika långa anpassar streckkoden sig efter detta. Figur 18 och figur 19 visar exempel på hur denna streckkod anpassar bredd efter innehållet.

Figur 18: Code 128 streckkod som innehåller 123456789

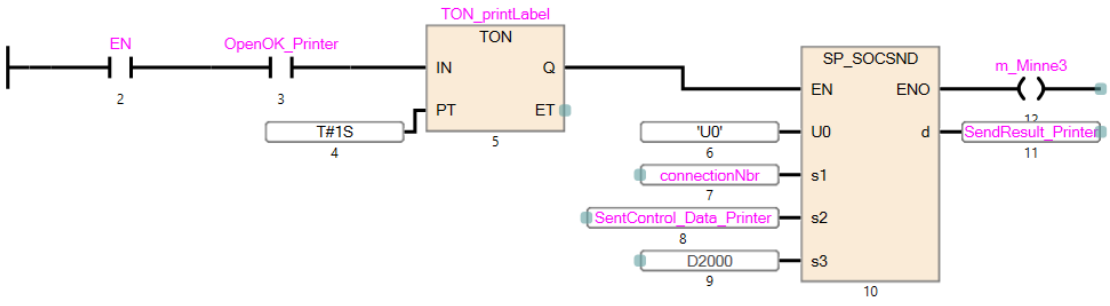
Figur 19: Code 128 streckkod som innehåller 123

För att skriva ut denna streckkod från PLC:n skrevs ST-kod som presenteras i figur 20.

```
IF EN AND OpenOK_Printer THEN;

    MOVP(TRUE, 84, D2000);
    STRINGMOVP(TRUE, '^', D2001);
    STRINGMOVP(TRUE, 'XA', D2002);
    STRINGMOVP(TRUE, '^', D2004);
    STRINGMOVP(TRUE, 'FO', D2005);
    STRINGMOVP(TRUE, labelPosX, D2007);
    STRINGMOVP(TRUE, ',', D2009);
    STRINGMOVP(TRUE, labelPosY, D2010);
    STRINGMOVP(TRUE, '^', D2011);
    STRINGMOVP(TRUE, 'BY', D2012);
    STRINGMOVP(TRUE, labelWidth, D2014);
    STRINGMOVP(TRUE, '^', D2015);
    STRINGMOVP(TRUE, labelType, D2016);
    STRINGMOVP(TRUE, ',', D2019);
    STRINGMOVP(TRUE, labelHeight, D2020);
    STRINGMOVP(TRUE, '^', D2022);
    STRINGMOVP(TRUE, 'FD', D2023);
    STRINGMOVP(TRUE, '^', D2038);
    STRINGMOVP(TRUE, 'FS', D2039);
    STRINGMOVP(TRUE, '^', D2041);
    STRINGMOVP(TRUE, 'XZ', D2042);

END_IF;
```

Figur 20: Kod för att skapa streckkoden i programmeringsmjukvaran med ZPL-kommandon

Med hjälp av denna kod läggs sedan kommandona för att skapa streckkoden i D2000-registret. För att senare kunna skicka denna information till skrivaren valdes här att använda funktionen ”SP_SOCSND” med tillhörande funktionsblock, vilket presenteras i figur 21. Streckkodens innehåll, i form av en sifferkombination, görs i ett Javaprogram och lagras i D2025. Javaprogrammet beskrivs i avsnitt 4.5.2. När detta var färdigt kunde streckkoder skrivas ut och det blev även möjligt att skriva ut flera streckkoder efter varandra.



Figur 21: Funktionsblocken i programmeringsmjukvaran som används för att skicka streckkoden till skrivaren

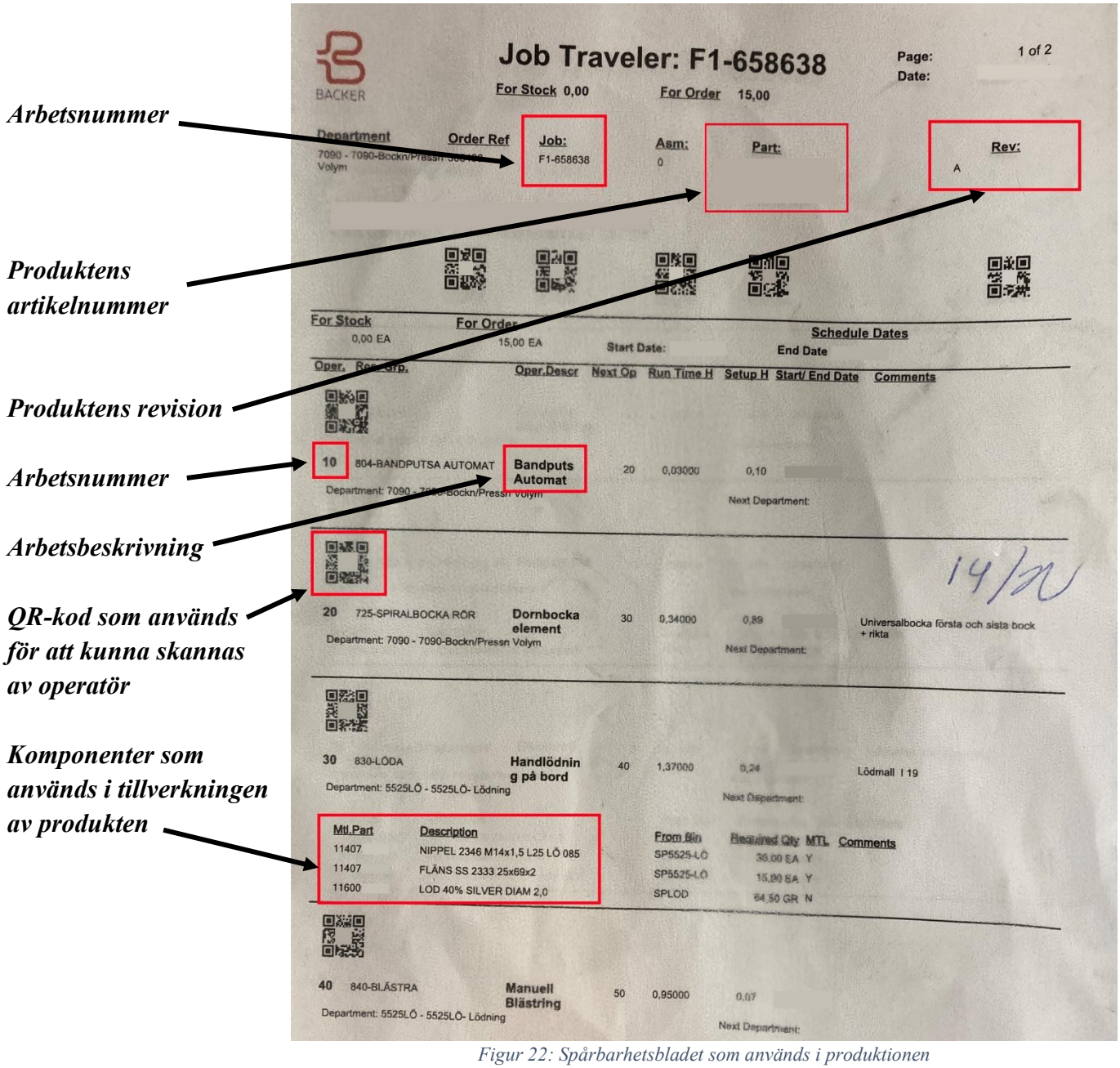
4.5 Framtagande av databas

För att utveckla en databas som ska innehålla information om en produkt har flera steg genomförts, dessa steg presenteras i följande avsnitt.

4.5.1 Undersökning av nuvarande spårbarhetssystem

För att kunna implementera ett nytt spårbarhetssystem med information kopplad till en produkt utfördes en undersökning om hur det nuvarande spårbarhetssystemet är uppbyggt. I denna process hämtades information in om vilken produktinformation som ska sparas. Det befintliga spårbarhetssystemet har ingen individuell märkning på enskilda produkter utan här används istället enbart jobbnumret. Alla produkter med samma jobbnummer läggs på samma pall och transporteras genom fabriken tillsammans. Komponenter och moment som genomfördes under tillverkningen noteras på papper som sedan arkiveras. Det nuvarande Spårbarhetssystemet innehåller även produktens artikelnummer och dess revision. Med hjälp av artikelnumret och revisionen går det att veta vilken typ av produkt som ska tillverkas.

Figur 22 presenterar spårbarhetsbladet som undersökningen behandlade och dess innehåll. I figuren har viss information maskerats bort på grund av sekretesskäl.



Figur 22: Spårbarhetsbladet som används i produktionen

4.5.2 Implementering av databas

Efter att en undersökning utförts på hur det nuvarande spårbarhetssystemet fungerar och vad som sparas i företagets affärssystem kunde i samarbete med personal på företaget den viktigaste informationen som ska sparas i databasen väljas i enlighet med följande punktlista:

- ***Tillverkningsordernummer***
- ***Produktens artikelnummer***
- ***Revison***
- ***Arbetsnummer***
- ***Komponentnummer***

För att sedan kunna koppla denna information till en produkt, och således skapa ett spårbarhetssystem, krävdes det att följande information också lades till:

- ***Streckkodsnummer***
- ***Serienummer***
- ***Tid***
- ***Om ett test i en station under produktionen var OK eller NOK***

När det bestämts vilken information som ska sparas så skapades tre tabeller som innehåller informationen i ovanstående punktlistor, detta för att sortera data i olika tabeller.

Den första tabellen fick namnet ”storedLabels”, och kan sägas vara den överordnade tabellen där information om de olika produkterna presenteras överskådligt. Denna tabell utformades för att innehålla följande attribut:

- *factoryID*
    - Kopplas till siffran på den utskrivna streckkoden. Detta attribut är en primärnyckel.
- *serialNbr*
    - Serienumret är en unik identifierare för en individuell produkt. Produkten tilldelas denna identifierare efter att produkten genomgått alla moment i produktionen och ett godkänt ”End-Of-Line test” genomförts. När produkten levererats till kunden är serienumret kundens referens som kan användas för att söka information gällande en individuell produkt. Således uppnås ett komplett spårbarhetssystem där produkten kan spåras under produktionen men även efter produktionen.
- *prodID*
    - Tillverkningsordernumret som tillhör produkten.
- *partNum*
    - Detta attribut beskriver vad produktens artikelnummer är.

- *revision*
    - Detta attribut beskriver revisionen av den produkt som ska tillverkas.
- *timeAdded*
    - Tid och datum när streckkoden skrevs ut och således när en produkts tillverkning påbörjades i fabriken.

FabriksID:t tillsammans med serienumret möjliggör sökning i de andra tabellerna efter önskad information.

Den andra tabellen fick namnet ”operationsOfLabel” och visar alla moment produkten gått igenom, vilka komponenter produkten består av samt dess revision, och slutligen datum och tid när produkten befann sig i en station i fabriken. Här valdes således att denna tabell ska innehålla följande attribut:

- *factoryID*
    - Foreign key som kopplas till ”factoryID” i ”storedLabels”-tabellen. Kopplas till siffran på den utskrivna streckkoden som skannas i varje station.
- *operationNbr*
    - Beskriver vilket arbetsnummer som produkten genomgått.
- *partNbr*
    - Beskriver vilken komponent, det vill säga artikelnummer, som använts för att tillverka produkten
- *revision*
    - Detta attribut beskriver revisionen av den komponent som produkten tillverkats med.
- *timeOfOperation*
    - Tid och datum när streckkoden befann sig i en station.

Den tredje och sista tabellen fick namnet ”testOfLabel”, där denna tabell visar alla tester genomförda på samtliga stationer som en produkt har tagit sig igenom innan End-Of-Line testet och datum och tid för testet. Denna tabells attribut valdes till:

- *factoryID*
    - Foreign key som kopplas till ”factoryID” i ”storedLabels”-tabellen. Kopplas till siffran på den utskrivna streckkoden som skannas i varje station.
- *stationNbr*
    - Kopplas till den station där testet genomförts.
- *ok_nok*
    - Är en beskrivning för ett test och om detta test var OK eller NOK.
- *timeOfTest*
    - Tid och datum när ett test på en produkt utfördes.

Med hjälp av dessa tabeller i den framtagna databasen uppnåddes ett komplett spårbarhetssystem där information om en produkt kan lagras och hämtas, men även göra det möjligt att presentera informationen om en produkt på ett överskådligt sätt. Detta innebär att exempelvis moment som en produkt genomgått i produktionen presenteras i tabellen

”testOfLabel” och komponenter som använts för att tillverka produkten presenteras i tabellen ”operationsOfLabel”.

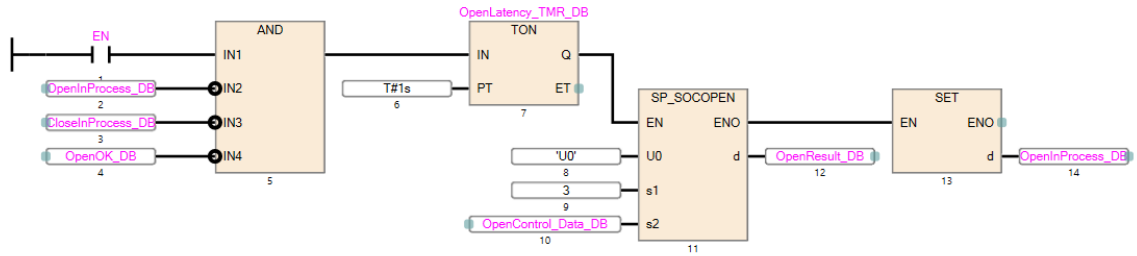
Dessa tabeller innehåller enbart den grundläggande informationen som krävs för att skapa ett spårbarhetssystem. Det är därför möjligt att med hjälp av denna databas lägga till fler attribut i databasen efter de behov som finns vid en verklig implementation. En verklig implementation kräver även att företagets affärssystem kopplas ihop med databasen. Detta eftersom attribut som används i denna databas finns tillgängliga i företagets affärssystem.

4.6 Lagring av information i databas

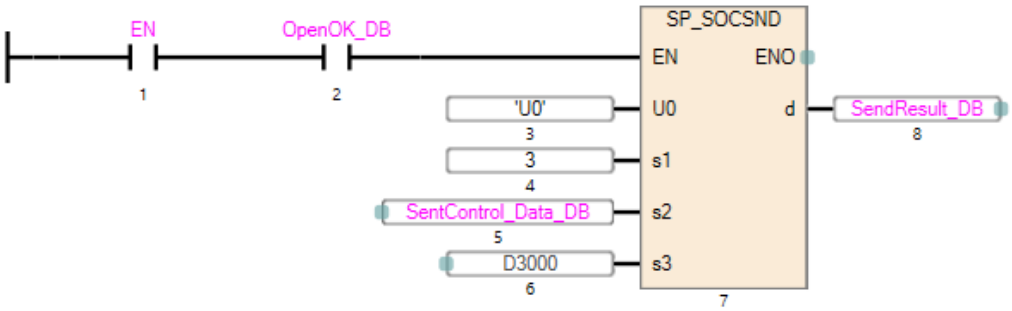
För att kunna spara information i en databas togs flera beslut kring hur detta skulle gå till. Det första alternativet var att programmera funktionsblock i GX Works 3 men problem uppstod under utvecklingen.

4.6.1 Problem med kommunikationen mellan databasen och PLC:n

För att konfigurera anslutningen mellan databasen och PLC:n genomgicks samma steg som för skrivaren. Även här lades en ”Active Connection Module” till som därefter fick en kanal tilldelad. På samma sätt som för skrivaren programmerades öppningen av anslutningen samt sändningen av data, vilket i detta fall bestod av att först skicka information för inloggning till databasen för att sen skicka SQL-kommandon som lägger in information i en tabell. De funktioner som programmerades visas i figur 23 och figur 24.

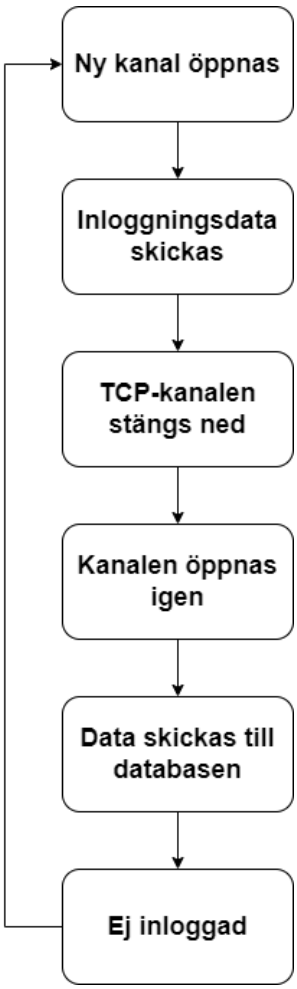


Figur 23: Funktionsblocken som skulle användas för att ansluta PLC:n till databasen



Figur 24: Funktionsblocket som skulle användas för att skicka information till databasen

Problemet som uppstod här är att så fort som data skickas till databasen stängs anslutningen ned. Således måste ny information om inloggning åter skickas till databasen, vilket resulterade i att programmet fastnade i detta läge och att ingen data som ska läggas in i tabellerna skickas. I figur 25 illustreras problemet.



Figur 25: Illustration av problemet med anslutningen till databasen

Det fanns då två olika lösningar på detta problem, att skifta fokus och istället utveckla en mellanhand som hanterar databasen eller köpa in en så kallad ”MES-modul” som hanterar databaser direkt från PLC:n.

På grund av inköpspris och leveranstider för en MES-modul valdes här att istället utveckla ett Javaprogram som hanterar data som skickas från PLC:n men även hanterar inläggning och hämtning av information i databasen.

4.6.2 Utveckling av databashanteraren

Javaprogrammet som utvecklades i detta moment agerar som en mellanhand mellan PLC och databasen. Programmet utvecklades så att SQL-kommandon kan skickas med data som hämtats från PLC:n. För att hämta data från PLC:n användes en API vid namn ”HslCommunication”. Denna API tillåter en utvecklare bland annat att ansluta till en PLC men även att hämta och skriva data.

HslCommunication innehåller flera funktioner som gör det möjligt att kommunicera med en PLC och att skriva och hämta information. Exempel på funktioner som HslCommunication API:n innehåller presenteras nedan:

- ***connectServer(String ipAddress, int port)***
  - Ansluter till en PLC med dess IP-adress och portnummer
- ***ReadBool(String address, short length)***
  - Läser ett booleskt värde för exempelvis en knapp. Returnerar ”true” vid värdet ett och ”false” vid värdet 0.
- ***ReadString (String address, short length)***
  - Läser en sträng som lagrats i exempelvis ett dataregister i PLC:n.
- ***Write(String device, int value)***
  - Skriver ett heltal till exempelvis ett dataregister.

I figur 26 nedan presenteras ett exempel på hur *ReadString* kan användas i en Javametod.

```
public static String readStringFromDevice(String device) {

    OperateResultExOne<String> read = melsec_net.ReadString(device, (short) 10);

    str = new StringBuilder();

    if (read.IsSuccess) {

        String readString = read.Content;

        for (int i = 0; i < readString.length(); i++) {

            if (readString.charAt(i) == '\0') {
                break;
            }

            str.append(readString.charAt(i));

        }

        return str.toString();
    } else {

        System.out.println("Read string failed" + read.Message);
        return "";
    }

}
```

Figur 26: Exempel på en metod som skapats för att hantera inläsning av en sträng från PLC:n med hjälp av API:n HslCommunication

För att hantera databasen hämtades ytterligare en API, denna vid namn ”mssql-jdbc”. Denna API gör det möjligt att kommunicera med en databas, samt att lagra och hämta information.

Med hjälp av dessa API:er implementerades flera metoder i olika klasser i Javaprogrammet som gjorde det möjligt att ansluta till PLC:n, hämta och skriva information från och till PLC:n, samt att hantera databasen i form av lagring och hämtning av information.

I följande punktlista presenteras de klasser som skapades och dess huvudfunktion förklaras:

- ***sqlServerConnector***
    - Denna klass hanterar anslutningen till databasen.
- ***PLCmanagement***
    - Denna klass hanterar anslutningen till PLC:n, skrivande av information till PLC:n samt hämtning av information från PLC:n.
- ***DBmanagement***
    - Denna klass hanterar funktioner gällande lagring av information i databasen och hämtning av data från databasen.
- ***gui***
    - Denna klass hanterar det grafiska gränssnittet som programmet innehåller.
- ***Main***
    - Main-klassen hanterar körningen av programmet där de olika metoderna från klasserna används.

Med hjälp av Javaprogrammet valdes det här att skapa streckkodens innehåll. I detta fall valdes det att skapa en sifferkombination enligt formatet ”ÅÅÅÅMMDDXYZQW”, där år, månad, datum och ett tal på fem siffror inom intervallet 1 till 99999 skickas till PLC:n. För att göra detta möjligt så valdes det att skapa en funktion som skapar en identifierare som börjar på ett, sedan när nästa streckkod skrivs ut ökar programmet detta värde med ett. Det vill säga att den första streckkoden som skrivs ut exempelvis innehåller ”2023040100001”, nästa streckkod som skrivs ut innehåller således ”2023040100002” och så vidare. För varje ny dag börjar denna räkning om på ett.

För att motverka problem som kan uppstå, exempelvis om programmet startas om, och alltid skapa unika identifierare valdes det att skapa en funktion som kontrollerar så att denna streckkod inte finns i databasen. Detta innebar att totalt 100.000 stycken unika identifierare kan skapas varje dag.

En viktig reflektion kring denna lösning är att i en verklig implementering är denna metod inte den bäst lämpade. Detta på grund av att detta spårbarhetssystem bygger på att en extern dator används för att köra Javaprogrammet. Detta medför extra underhåll av datorer, exempelvis med systemuppdateringar och liknande samt skötsel av programmet efter exempelvis en programkrasch. För att skapa ett system som inte bygger på att använda en extern dator är det lämpligare att sköta all hantering av information i PLC:n, där exempelvis skötsel av databasen utförs av en MES-hårdvarumodul.

4.7 Hämtning av information ur en databas

För att kunna hämta information ur en databas undersöktes möjligheterna kring hur detta skulle kunna gå till. Det första alternativet var att vidareutveckla Javaprogrammet så att programmet kunde presentera den information som databasen innehåller gällande en produkt. Det andra alternativet var att inte utveckla Javaprogrammet ytterligare utan att hantera sökning efter information i databashanteringsmjukvaran. Här ansågs båda alternativen vara lämpliga där enkel sökning efter en specifik streckkod görs med Javaprogrammet och specifika sökningar, exempelvis sökning efter alla streckkoder som skapats under ett visst datum, sköts av databashanteringsmjukvaran.

Vidareutvecklingen av Javaprogrammet innebar då att implementera ett grafiskt gränssnitt där användaren kan skriva in en streckkod för att få information gällande en produkt. Därefter hämtas all information från de tre tabellerna i databasen som tidigare skapats, där denna information är kopplad till en individuell produkt. Grunden till detta beslut var främst viljan att skapa ett enklare gränssnitt mellan databasen och Javaprogrammet. Det vill säga att inga tidigare kunskaper om SQL-kommandon krävs utan information kan hämtas efter en inmatning av streckkoden.

Det andra alternativet bortsett från att vidareutveckla Javaprogrammet var att sköta specifik inhämtning av information från databasen med hjälp av databashanteringsmjukvaran, i detta fall SSMS. Eftersom databashanteringsmjukvaran specifikt hanterar databaser och sökning efter information är det därför möjligt att exempelvis göra mer specifika sökningar efter information. Det är exempelvis möjligt att söka efter produkter som producerats under ett visst datum, med en specifik komponent, eller med en specifik tillverkningsorder. Figur 27 nedan visar exempel på ett SQL-kommando som kan användas för att söka efter alla moment som en produkt med ett visst serienummer genomgått under tillverkningen.

```
--Sökning efter moment som en produkt genomgått under tillverkningen med ett visst serienummer
SELECT
operationsOfLabel.factoryID AS Streckkodsnummer,
operationNbr AS Operation,
partNbr AS Komponent_artNummer,
timeOfOperation AS TidUtfördOperation
FROM
operationsOfLabel
INNER JOIN
storedLabels
ON
storedLabels.factoryID = operationsOfLabel.factoryID
WHERE
serialNbr = 1;
```

Figur 27: Exempel på SQL-kommandon för att söka efter information i databasen

5. Resultat

Detta kapitel beskriver resultatet av arbetet med att ta fram ett spårbarhetssystem inom ramen för examensarbetet. Arbetet har resulterat i ett spårbarhetssystem som har gjort det möjligt att kunna spåra en produkt genom en simulerad fabrikslinje där bland annat komponenter som ingår i tillverkningen kan skannas och dokumenteras i en databas. I simuleringen kan även tester utföras och dokumenteras. Det framtagna spårbarhetssystemet består av flera delar där dess funktioner presenteras i följande avsnitt.

5.1 Uppbyggnad av spårbarhetssystemet

Det slutgiltiga spårbarhetssystemet består av flera delar som gör det möjligt att spåra en produkt både under och efter tillverkningen. Detta spårbarhetssystem består av ett styrsystem i form av en PLC och ett HMI, en etikettskrivare, skannrar, ett Javaprogram och en databas.

I detta spårbarhetssystem ansvarar styrsystemet för de fundamentala funktionerna som krävs för att hantera information gällande en produkts tillverkning. Funktionerna som styrsystemet hanterar presenteras i följande punktlista:

- Anslutning till etikettskrivaren.
- Nedkoppling från etikettskrivaren.
- Utskrift av individmärkningsetikett.
- Hantering av tester som utförs i fabrikslinjen.
- Hantering av skannrar.
- Lagring av information från fabrikslinjen.

För att hantera databasen och vissa funktioner i PLC:n togs ett Javaprogram fram. Detta Javaprogram hanterar och innehåller följande funktioner:

- Lagring av information i databasen.
- Hämtning av information ur databasen.
- Skapande av streckkodens innehåll.
- Ett grafiskt gränssnitt som presenterar information för användaren.
- Hämtning av information som lagrats i PLC:n.

Sammantaget har det med hjälp av dessa funktioner resulterat i att ett komplett och fungerande spårbarhetssystem skapats. Det slutgiltiga spårbarhetssystemet presenteras i appendix 8.1.

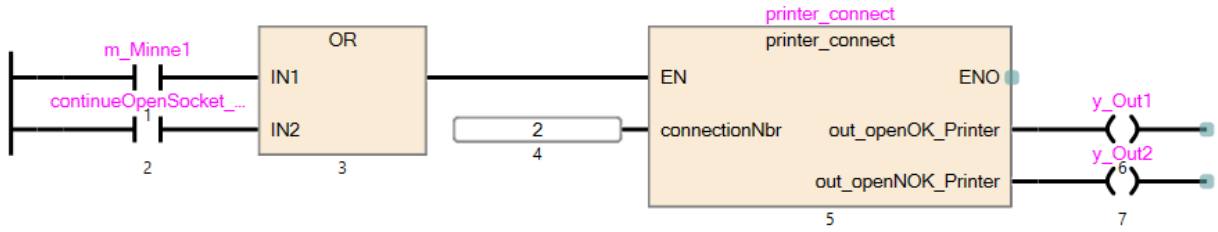
5.2 Styrsystemet

För att skapa ett fungerande spårbarhetssystem har flera funktioner skapats i styrsystemet. Dessa funktioner presenteras i följande avsnitt.

5.2.1 Individmärkning av en produkt

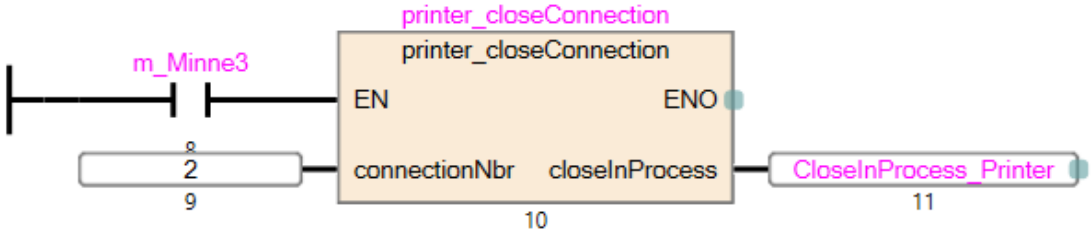
Den första funktionen ska enligt flödesschemat i figur 7 kunna skriva ut en etikett med en streckkod som individmärker en produkt. I PLC-programmeringsmjukvaran delades denna funktion upp i tre delar, ”printer_connect”, ”printer_closeConnection” och ”printer_printLabel”. Dessa funktionsblock presenteras och förklaras i följande punktlista.

- ***Printer_connect***
  - Detta funktionsblock startar processen för att kunna upprätta en kommunikationskanal mellan etikettskrivaren och PLC:n, detta med hjälp av funktionen ”*SP_SOCOPEN*”, en timer och grindlogik. Efter att en anslutning mellan PLC:n och skrivaren upprättats sätts out_openOK_Printer till true. Detta funktionsblock presenteras i figur 28 och tillhörande kod presenteras i appendix 8.9.



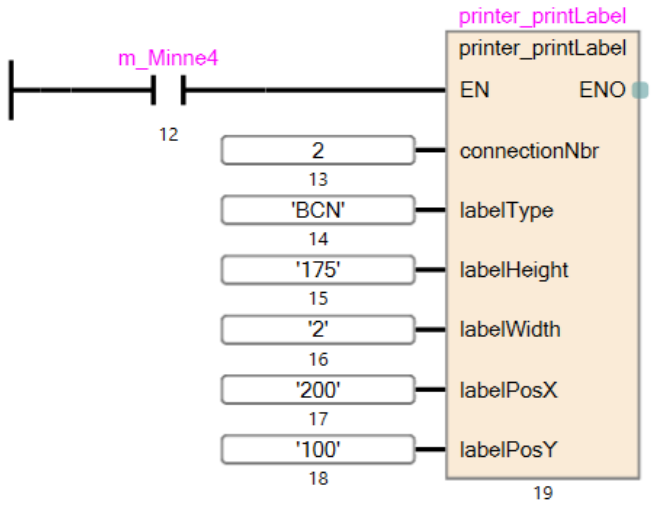
Figur 28: Funktionsblocket som ansluter till skrivaren

- ***Printer_closeConnection***
  - Kopplar ner anslutningen till skrivaren med hjälp av funktionsblocket ”*SP_SOCCLOSE*” när en användare trycker på en knapp på HMI:et. Detta funktionsblock presenteras i figur 29 och tillhörande kod presenteras i appendix 8.10.



Figur 29: Funktionsblock som stänger ner anslutningen till skrivaren

- ***Printer_printLabel***
    - Detta funktionsblock gör det möjligt att, efter en anslutning till skrivaren upprättats, kunna skriva ut en etikett med en streckkod. För att skriva ut unika etiketter ändras streckkodens fältdata. I detta fall ändras fältdatan enligt formatet ”*ÅÅÅÅMMDDXYZWC*” som skapas i Javaprogrammet. Javaprogrammet lägger även in etikettdatan i databasen efter att denna knapp tryckts ned. Detta funktionsblock presenteras i figur 30 och tillhörande kod presenteras i appendix 8.11.



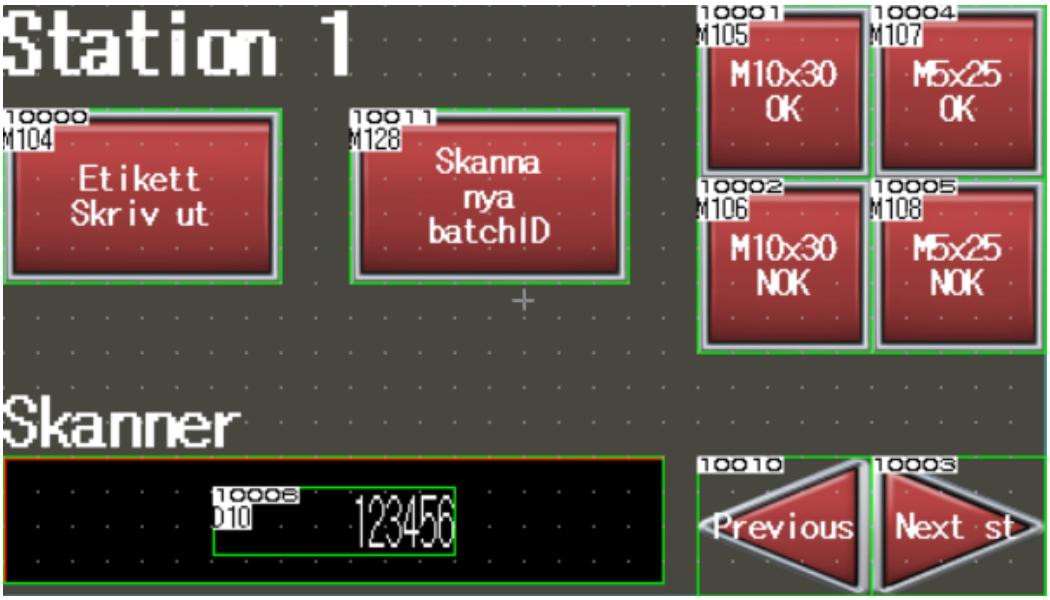
Figur 30: Funktionsblock som skapar en streckkod och som sedan skriver ut en etikett med denna streckkod

Etiketten som kan skrivas ut enligt formatet ”*ÅÅÅÅMMDDXYZWC”* visas i figur 31.

Figur 31: Exempel på etikett som kan skrivas ut

5.2.2 Hantering av information från HMI

Fabrikslinjen som detta examensarbete har utgått från har en väldefinierad arbetsprocess där det ingår flera olika moment. För att kunna utveckla funktioner som kan hantera information kring dessa moment i fabrikslinjen simulerades denna linje, bestående av alla tio stationer, i HMI:n. Hur den första stationen simulerades visas i figur 32. De andra stationerna bygger på samma principer förutom att etikett enbart skrivs ut i den första stationen.



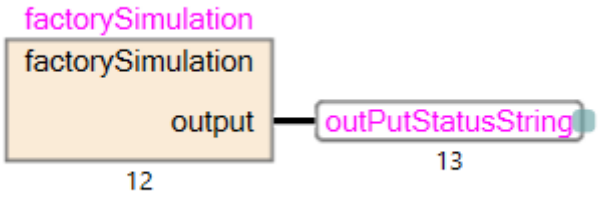
Figur 32: Simulering av station ett i HMI:n

Simuleringen består således av de moment som utförs i produktionen idag och innehåller även nya metoder som utvecklats under examensarbetet, exempelvis hur information gällande komponenter som ska användas i produktionen ska skannas och hanteras av styrsystemet samt utskrift av etikett. För att lagra information om vilka komponenter som används i tillverkningen skapades en knapp i HMI:n kallad ”Skanna nya batchID” som sedan öppnar fönstret som visas i figur 33. Här är det möjligt för användaren att skanna in nya komponenter som sedan sparas.

Figur 33: Fönstret i HMI:n som gör det möjligt att skanna in nya komponenter som ska användas i tillverkningen

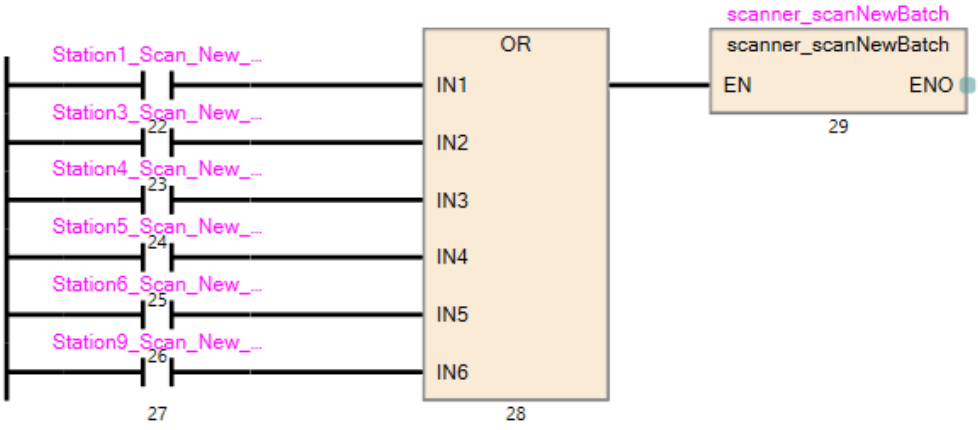
I samband med att dessa funktioner i HMI:n utvecklades togs det fram fler funktionsblock som gjorde så att dessa funktioner fungerade som tänkt. De funktionsblock som genomför testerna, respektive hämtar information gällande en komponent i HMI:n presenteras i följande punktlista:

- ***factorySimulation***
    - Detta funktionsblock som presenteras i figur 34 gör det möjligt för en användare att trycka på en knapp på HMI:et som simulerar att ett test har utförts och är OK eller NOK. Tillhörande kod presenteras i appendix 8.14.



Figur 34: Funktionsblocket som gör det möjligt att läsa insignaler från knappar i ett HMI som sedan meddelar om ett test var OK eller NOK

- **Scanner_scanNewBatch**
    - Detta funktionsblock som presenteras i figur 35 gör det möjligt att användaren, med hjälp av HMI:et, kan skanna in nya komponenter som används under tillverkningen av en produkt i fabrikslinjen. Skulle exempelvis en låda med skruvar ta slut under tillverkningen kan användaren trycka på en knapp som skannar in en ny låda med skruvar. Tillhörande kod presenteras i appendix 8.16.



Figur 35: Funktionsblocket som gör det möjligt att skanna nya komponenter och spara dessa

5.3 Databasen

Den databas som skapats och hur information kan hämtas ur denna presenteras i följande avsnitt.

5.3.1 Uppbyggnad av databasen

Databasen är utvecklad för att på ett enkelt vis kunna söka efter information gällande en produkt samt lagra den information som är viktigast för att uppnå ett komplett spårbarhetssystem. Resultatet av den slutgiltiga uppbyggnaden av databasen som togs fram i kapitel 4.5 och dess tabeller presenteras i följande punktlista:

- ***storedLabels***
  - *factoryID* (Primärnyckel)
  - *serialNbr*
  - *prodID*
  - *partNum*
  - *revision*
  - *timeAdded*

- ***operationsOfLabel***
  - *factoryID* (Främmande nyckel)
  - *operationNbr*
  - *partNbr*
  - *revision*
  - *timeOfOperation*

- ***testOfLabel***
  - *factoryID* (Främmande nyckel)
  - *ok_nok*
  - *stationNbr*
  - *timeOfTest*

5.3.2 Hämtning av information ur databasen

Utvecklingen av den databas som skapats i detta examensarbete gjorde det möjligt att söka efter och hämta information om en produkt, exempelvis vilka moment en produkt genomgått och vilka komponenter som har använts under tillverkningen. Exempel på information som kan hämtas presenteras i följande punktlista:

- Sökning efter moment som en produkt genomgått under tillverkningen med ett visst serienummer.
- Sökning efter tester som genomförts på en produkt med ett visst serienummer.
- Sökning efter tester som genomförts på en produkt med en viss streckkod.
- Sökning efter streckkoden som tillhör ett visst serienummer, till exempel om etiketten är borttagen från produkten.
- Sökning efter produkter som har använt en viss komponent under tillverkningen.
- Sökning efter produkter som har en viss tillverkningsorder.
- Sökning efter produkter som har ett visst artikelnummer.

- Sökning efter produkter som tillverkats mellan två datum.

Exempel på utdrag av information ur databasen efter en simulering av fabrikslinjen presenteras i figur 36 och figur 37.

	factoryID	serialNbr	prodID	partNum	revision	timeAdded
1	2023041100001	1	588807	6051155611	D	2023-04-11 14:20:27.973
2	2023041100002	2	588807	6051155611	D	2023-04-11 14:20:32.300
3	2023041100003	3	581321	6051155611	A	2023-04-11 14:20:49.793

Figur 36: Exempel på utdrag från tabellen "storedLabels" där alla skapade streckkoder lagras med tillhörande information.

	factoryID	operationNbr	partNbr	revision	timeOfOperation
1	2023041100001	10	1	A	2023-04-11 14:21:40.743
2	2023041100001	10	2	A	2023-04-11 14:21:40.760
3	2023041100001	10	3	A	2023-04-11 14:21:40.773
4	2023041100001	10	4	A	2023-04-11 14:21:40.787
5	2023041100001	10	5	A	2023-04-11 14:21:40.797
6	2023041100002	10	1	A	2023-04-11 14:22:00.617
7	2023041100002	10	2	A	2023-04-11 14:22:00.630
8	2023041100002	10	3	A	2023-04-11 14:22:00.640
9	2023041100002	10	4	A	2023-04-11 14:22:00.650
10	2023041100002	10	5	A	2023-04-11 14:22:00.657
11	2023041100003	10	1	A	2023-04-11 14:22:22.650
12	2023041100003	10	2	A	2023-04-11 14:22:22.667
13	2023041100003	10	3	A	2023-04-11 14:22:22.690
14	2023041100003	10	4	A	2023-04-11 14:22:22.707
15	2023041100003	10	5	A	2023-04-11 14:22:22.730
16	2023041100001	30	6	A	2023-04-11 14:23:31.610
17	2023041100001	30	7	A	2023-04-11 14:23:31.627
18	2023041100001	30	8	A	2023-04-11 14:23:31.637
19	2023041100001	30	9	A	2023-04-11 14:23:31.653
20	2023041100001	30	10	A	2023-04-11 14:23:31.667
21	2023041100001	30	11	A	2023-04-11 14:23:31.680
22	2023041100002	30	6	A	2023-04-11 14:23:53.420
23	2023041100002	30	7	A	2023-04-11 14:23:53.437
24	2023041100002	30	8	A	2023-04-11 14:23:53.460
25	2023041100002	30	9	A	2023-04-11 14:23:53.470
26	2023041100002	30	10	A	2023-04-11 14:23:53.480
27	2023041100002	30	11	A	2023-04-11 14:23:53.493
28	2023041100003	30	12	A	2023-04-11 14:24:26.200
29	2023041100003	30	13	A	2023-04-11 14:24:26.217
30	2023041100003	30	11	A	2023-04-11 14:24:26.227
31	2023041100003	30	25	A	2023-04-11 14:24:26.240
32	2023041100003	30	10	A	2023-04-11 14:24:26.253
33	2023041100003	30	36	A	2023-04-11 14:24:26.270

Figur 37: Exempel på utdrag från tabellen "operationsOfLabel" där information gällande komponenter som använts under tillverkningen av en produkt lagras.

5.4 Javaprogrammet

Javaprogrammet är utvecklat för att kunna lagra information i databasen samt hämta information från databasen. Javaprogrammet är passivt i förhållande till styrsystemet och så länge programmet körs väntar programmet på att något ska hända, exempelvis i form av knapptryck på HMI:n. Beroende på vilken knapp som trycks ned körs sedan en eller flera metoder i Javaprogrammet.

I följande avsnitt presenteras de Javaklasser som skapats och dess metoder förklaras. Sammantaget har detta Javaprogram främst resulterat i att information kan lagras i databasen. Utvecklingen av Javaprogrammet har även resulterat i att viss information kan hämtas från databasen och kan presenteras för användaren.

5.4.1 Mainklassen

Main-klassen för programmet som innehåller följande metoder:

- ***main()***
  - Här initieras globala variabler och andra metoder definieras.
- ***runTraceabilitySystem()***
  - I denna metod körs Javadelen av spårbarhetssystemet där Javaprogrammet skannar efter händelser i PLC:n och kör relevanta metoder när en bestämd handling utförts.

Denna klass presenteras i appendix 8.3.

5.4.2 PLCmanagement

Utvecklingen av denna klass gjorde det möjligt att hantera PLC:n. Bland annat för att hämta data från PLC:n och skriva data till PLC:n, samt ansluta till PLC:n. Denna klass innehåller följande metoder:

- ***void connectToPLC(String ipNbr, int partNbr)***
  - Denna metod ansluter till PLC:n specificerad med dess IP-adress och portnummer.
- ***String verifyConnection()***
  - Denna metod bekräftar för användaren att en anslutning upprättats eller inte.
- ***int readIntFromDevice(String device)***
  - Denna metod läser ett heltal från ett dataregister specificerat med *device*, metoden kan exempelvis läsa ett heltal från *D100.*
- ***long readDoubleFromDevice(String device)***
  - Denna metod läser ett flyttal på 32-bitar från ett dataregister specificerat med *device*, metoden kan exempelvis läsa ett flyttal från *D100.*
- ***boolean readBoolFromDevice(String device)***
  - Denna metod kan läsa om ett booleskt värde från ett minne specificerat med *device*, metoden kan exempelvis läsa om M101 är sann eller falsk.
- ***String readStringFromDevice(String device)***
  - Denna metod kan läsa en sträng från ett dataregister specificerat med *device*, metoden kan exempelvis läsa en sträng lagrad i *D102.*

- ***void addLabelToDB(String button)***
    - Denna metod skapar en unik streckkod och lagrar denna i databasen.
- ***String createNewLabel()***
    - Denna metod skapar en unik streckkod och returnerar denna. Används främst i metoden *addLabelToDB*.
- ***String createNewSerialNumber()***
    - Denna metod skapar ett nytt serienummer som bestäms när en produkt i den simulerade fabrikslinjen når sista stationen och EOL-testet är godkänt.
- ***void createNewJob(String button)***
    - Denna metod används för simuleringssyften där information gällande en produkt skapas. Här skapas ett nytt arbetsnummer, ett nytt artikelnummer och en ny revision.
- ***void addStationInfoToDB(String stationNbr)***
    - Denna metod är själva huvudfunktionen för spårbarhetssystemet där en streckkod kan matas in i HMI:et och därefter görs en sökning i databasen om denna streckkod finns i databasen. Finns streckkoden i databasen sparas denna streckkod i registret kopplat till den stationen. Detta gör det möjligt att veta vilken streckkod, det vill säga vilken individuell produkt, som befinner sig i en station.
- ***void getBatchID(String button)***
    - Denna metod hämtar information som sparats i PLC:n gällande vilken komponent som används i tillverkningen och som skannats in med hjälp av HMI:et.
- ***void readNokOkFromPlc(String stationNbr, String okButton, String nokButton)***
    - Denna metod läser av om ett test var OK eller NOK gällande ett test i en viss station. Denna metod läser av de knappar som behandlar om ett test var OK eller NOK under simuleringen av tillverkningen.

Denna klass presenteras i appendix 8.4.

5.4.3 DBmanagement

Utvecklingen av denna klass resulterade i att programmet kunde hantera hämtning av information från databasen och lagring av information i databasen. De metoder som skapats presenteras i följande punktlista:

- ***String selectAllFromTable(long searchedLabelNbr)***
    - Denna metod gör det möjligt för en användare att söka efter en utskriven streckkod och presentera all information gällande denna streckkod som hämtas från de tre tabellerna som databasen består av.
- ***long getLabelNbrFromDatabase(String table, string column, long equals)***
    - Denna metod söker efter en streckkod i en tabell specifierad av *equals*. Finns den sökta streckkoden i databasen returnerar denna metod den sökta streckkoden, annars returnerar metoden noll.
- ***void insertIntoTable(String table, String column, String value)***
    - Denna metod gör det möjligt att lägga in information i en tabell med hjälp av att specificera kolumnen och värdet som ska läggas in. Följande SQL-kommando körs: *INSERT INTO table (column) VALUES (value)*.

- ***void insertIntoTableTwoColumns(String table, String column1, String column2, String value1, String value2)***
    - Denna metod gör det möjligt att lägga in information i två kolumner i en tabell. Med hjälp av att specificera två värden som ska läggas in körs följande SQL-kommando: *INSERT INTO table(column1, column2) VALUES (value1, value2)*.
- ***void insertIntoTableThreeColumns(String table, String column1, String column2, String column3, String value1, String value2, String value3)***
    - Denna metod gör det möjligt att lägga in information i tre kolumner i en tabell genom att specificera de kolumner och de tre värden som ska läggas in. Därefter körs följande SQL-kommando: *INSERT INTO table(column1, column2, column3) VALUES (value1, value2, value3)*.
- ***void updateTable(String table, String column1, String column2, String value, String equals)***
    - Denna metod gör det möjligt att uppdatera en kolumn i en tabell där följande SQL-kommando körs: *UPDATE table SET column1 = value WHERE column2 = equals*
- ***String selectLastRow()***
    - Denna metod gör det möjligt att läsa den senaste inlagda raden i databasen. Denna metod används främst för att säkerställa att unika etiketter alltid skrivs ut. Detta då Javaprogrammet ska kunna startas om och fortsätta utskriften av unika streckkoder.

Denna klass presenteras i appendix 8.5.

5.4.3 GUI

Framtagandet av denna klass gjorde det möjligt att skapa ett grafiskt gränssnitt som gör det möjligt att presentera information för användaren, exempelvis gällande vilken data som lagts till i databasen samt presentation av information efter sökning efter en streckkod. Denna klass innehåller följande metoder:

- ***void createGUI()***
    - Denna metod skapar och lägger till grundfunktionerna till det grafiska gränssnittet. Det vill säga att knappar skapas och läggs till, inmatningsruta läggs till och fönstret skapas.
- ***void updateGUI(String updateText)***
    - Denna metod hämtar texten som matats in i attributet *updateText* och presenterar denna text i det grafiska gränssnittet.
- ***void buttonsOfGUI()***
    - Denna metod hanterar knapparna som används i det grafiska gränssnittet. Dessa knappar är *clearWindowButton* som rensar texten i det grafiska gränssnittet, *clearTextButton* som rensar texten som matats in i textfältet och *sendSQLbutton* som söker efter den inmatade streckkoden i databasen.

Denna klass presenteras i appendix 8.6.

5.4.4 sqlServerConnector

Utvecklingen av denna klass gjorde det möjligt att ansluta till den skapade databasen med hjälp av JDBC-API:n som använts i detta examensarbete. Denna klass innehåller en metod som presenteras i följande punktlista:

- *void connectToDatabase()*
    - Denna metod ansluter till databasen med hjälp av JDBC-API:n med hjälp av en ”connectionstring”.

Denna klass presenteras i appendix 8.8.

6. Slutsats

Examensarbetet har utvecklat ett spårbarhetssystem som består av tre delar:

1. Ett styrsystem bestående av en PLC och ett HMI.
2. Ett Javaprogram som hämtar, behandlar och skickar information från och till PLC:n samt databasen.
3. En databas som lagrar information skickat från Javaprogrammet.

Det slutgiltiga spårbarhetssystemet presenteras i appendix 8.1.

6.1 Frågor och svar

Följande frågor har besvarats under examensarbetet.

Hur ska spårbarhetssystemet utformas för att kunna vara användbart i hela produktionskedjan?

För att spårbarhetssystemet ska vara användbart i hela produktionskedjan har en allmän metod för hantering av spårbarhet i produktionen tagits fram. Denna metod består av tre huvudfunktioner:

1. Etikettutskrift
2. Läsning av etikett
3. Lagring av information i en databas

Dessa funktioner har under examensarbetet utvecklats främst för att göra det möjligt att hantera individuella produkter som tillverkas i fabrikslinjen, vilket beskrevs i kapitel 2.4. Eftersom utgångspunkten var att utveckla återanvändningsbara funktioner kan dessa och databasen ändras efter behov så att de är applicerbara i andra delar i företagets produktion.

Hur ska funktionsblocken utformas för att kunna återanvändas?

För att skapa återanvändningsbara funktionsblock, som i detta examensarbete var funktionsblock som hanterar etikettutskrift, krävdes det att flera aspekter beaktades. Dessa aspekter var:

- *Anpassningsbarhet*
    - I form av att ut- och ingångar skapas som kan ändra enklare detaljer. Exempelvis vilken streckkodstyp som ska användas och dess position på etiketten.
- *Tydlig uppbyggnad*
    - I form av ordningen av funktioner som ett funktionsblock exekveras med.
- *Dokumentation*
    - I form av kommentarer som förklarar en viss funktion.

Hur ska produkterna individmärkas?
För att individmärka en produkt har detta examensarbete tagit fram funktioner till en etikettskrivare. Detta i form av utskrift av en etikett som innehåller en streckkod. Streckkoden är unik och skapas i Javaprogrammet där formatet är enligt följande: *ÅÅÅÅMMDDXYZWC*.

Vilken information ska sparas i databasen?
I databasen sparas följande information:

- Streckkoden på den utskrivna etiketten.
- Serienumret efter ett godkänt EOL-test utförts.
- Tillverkningsordernumret för en produkt.
- Artikelnumret för en produkt.
- Aktuell revision för en produkt.
- Stationsnummer där ett visst test utfördes.
- Stationsnummer där komponenter lades till.
- Komponenter som använts i produktionen.
- Utförda tester som genomförts i de olika stationerna.
- Tid för utskriven etikett.
- Tid för utfört test på en produkt.
- Tid då komponenter användes för att tillverka en produkt.

Hur ska information om en produkt sparas i en databas?
För att spara information om en produkt i en databas har ett Javaprogram utvecklats. Detta Javaprogram lagrar information i en databas med hjälp av en JDBC-API som kan hantera SQL Server databaser. Information från den simulerade fabrikslinjen hämtas av Javaprogrammet som sedan lagrar denna information i databasen.

Hur ska information om en produkt hämtas ur en databas?
För att hämta information ur databasen finns två alternativ tillgängliga. Det första alternativet är att söka efter en streckkod i det grafiska gränssnittet som Javaprogrammet hanterar. I detta gränssnitt presenteras all information ur de tre tabellerna från databasen som gäller en specifik streckkod. Det andra alternativet är att hämta information med hjälp av SSMS, här kan många olika typer av SQL-kommandon köras för att hämta relevant information.

6.2 Etiska aspekter

Det spårbarhetssystem som utvecklats inom ramen för examensarbetet ger många fördelar, men även några nackdelar.

Genom att samla in information i en industriell miljö kan flera etiska problem uppstå, exempelvis kan systemet upplevas som en form av arbetsplatsövervakning. Skulle detta examensarbete utvecklas vidare skulle det kunna vara möjligt att övervaka och lagra information exempelvis gällande enskilda människors arbetsinsatser. Således skulle det kunna uppstå problem runt dessa personers integritet. Exempelvis skulle det kunna vara möjligt att utläsa information om hur effektivt en person arbetar, hur många fel i produktionen denna person orsakat och så vidare. Detta kan utsätta personer för ökad press, då de ska både arbeta felfritt och effektivt.

Å andra sidan medför det även fördelar med att samla in information i en industriell miljö. Uppstår det många fel i en fabrikslinje, oberoende på vem som utfört arbetet, kan det leda till att denna tillverkningsprocess ses över och förbättras. En annan fördel är att det kan leda till förbättringar rent arbetsmiljömässigt. Det vill säga att det kan vara möjligt att utläsa om vissa personer arbetar för mycket och för länge. Detta gör det möjligt att förbättra arbetsvillkoren och arbetsmiljön på arbetsplatsen.

6.3 Brister

De brister som uppstått under utvecklingen av detta examensarbete presenteras nedan.

1. Eftersom de externa enheterna konfigurerades med ethernetmoduler finns det ett begränsat antal av dessa som kan användas beroende på vilken PLC som används. I PLC:n som användes under detta examensarbete finns det endast möjlighet till att ansluta åtta externa enheter, exempelvis en för etikettutskrift och sju etikettskannrar. Detta är en brist då systemet är uppbyggt för att använda nio etikettskannrar och en etikettskrivare, det vill säga totalt tio externa enheter. Detta innebär att om detta system ska implementeras i verkligheten kräver det ett annat val av PLC. De nyare Mitsubishi PLC-serierna ”IQ-R”-, ”Q”- eller ”L”-serien gör det möjligt att använda totalt 16 externa enheter.
2. En verklig skanner har ej kunnat testas då den inte fanns tillgänglig. Således har inga funktionsblock kunnat skapas.

6.3 Framtida utvecklingsmöjligheter

Det spårbarhetssystem som har utvecklats i detta examensarbete har många framtida utvecklingsmöjligheter. Flera delar av systemet kräver en vidareutveckling före en verklig implementation. Några av dessa utvecklingsmöjligheter presenteras i detta avsnitt.

6.3.1 Etikettskanner

Då det inte var möjligt att få tillgång till en skanner under examensarbetet beskrivs några möjliga skannrar nedan. Då systemet utvecklas kan dessa skannrar användas och på så sätt bidra till en utveckling av systemet.

Två olika skannrar som kan användas är Mitsubishis egna skannrar, exempelvis ”CF37”, alternativt en ”OMRON V430-F”. Båda dessa använder ethernet som kommunikationsmedium, vilket innebär att ett ”skannerfunktionsblock” kommer behövas till varje enskild skanner i PLC:ns utvecklingsmiljö. Varje enskild skanner kommer ha sin tilldelade kommunikationskanal då de har olika IP-adresser. De kommer därför att behöva konfigureras så som det visas i figur 14.

6.2.3 Implementering med MES-modul

Istället för att använda sig av Javaprogrammet är ett alternativ att använda en extern hårdvarumodul som kallas ”MES”-modul. Denna modul gör det möjligt att kommunicera direkt med en databas i form av lagring och hämtning av information, det vill säga utan att det finns en dator mellan PLC:n och databasen. ”MES” står för Manufacturing Execution System och kan användas tillsammans med PLC-serien ”IQ-R”. Det finns även möjlighet att använda sig av andra PLC-serier men funktionaliteten kan inte bekräftas för dessa. MES-modulen stödjer de största SQL-databaserna, exempelvis Microsoft SQL Server och MySQL. SQL-kommandona som en MES-modul kan utföra är INSERT, SELECT, UPDATE, DELETE, Multi-INSERT och STORED PROCEDURE [12].

Det finns flera fördelar med att bygga ut spårbarhetssystemet med en MES-modul istället för att använda en mellanhand som ansvarar för databasen. En av dessa fördelar är att mängden hårdvara som behövs för ett fungerande system minimeras, vilket resulterar i ett enklare system. Till följd av att systemet består av färre komponenter leder det till att mindre underhåll av systemet krävs vilket då också minskar den totala kostnaden för systemet.

En annan fördel med att använda en MES-modul är att när den kommunicerar med databasen görs inte någon onödig polling. Detta är en fördel eftersom detta leder till att MES-modulen inte belastar nätverket mer än nödvändigt vid sändning av information, vilket är viktigt att ha i åtanke om flera PLC:s med tillhörande MES-modul ska implementeras på flera ställen i fabriken.

7. Källförteckning

[1] Visure Solutions, Inc. *Vad är spårbarhet?*. URL: https://visuresolutions.com/sv/vad-%C3%A4r-sp%C3%A5rbarhet/ (Hämtad 2023-03-28).

[2] ABB. *Styrsystem i det mindre formatet*. URL: https://new.abb.com/se/om-abb/teknik/sa-funkar-det/plc (Hämtad 2023-01-29).

[3] COPADATA. *Vad är HMI?*. URL: https://www.copadata.com/sv/produkter/zenon-software-platform/visualisering-kontroll/vad-aer-hmi-human-machine-interface-maenniska-maskingraenssnitt-copa-data/ (Hämtad 2023-01-29).

[4] Mitsubishi Electric. *GT2104-RTBD*. URL: https://emea.mitsubishielectric.com/fa/products/hmi/got/got2000/gt21/gt2104-rtbd.html (Hämtad 2023-02-04).

[5] TSC Printers. *TX Series 4-Inch Performance Desktop Printers*. URL: https://emea.tscprinters.com/en/products/tx-series-4-inch-performance-desktop-printers (Hämtad 2023-02-24).

[6] Microsoft. *SQL Server 2022*. URL: https://www.microsoft.com/en-us/sql-server/sql-server-2022 (Hämtad 2023-02-07).

[7] Microsoft. *What is SQL Server Management Studio (SSMS)?*. URL: https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16 (Hämtad 2023-02-07).

[8] Mitsubishi Electric. *GX Works3*. URL: https://www.mitsubishielectric.com/fa/products/cnt/plceng/smerit/gx_works3/index.html (Hämtad 2023-01-30).

[9] Mitsubishi Electric. *GT Works3*. URL: https://emea.mitsubishielectric.com/fa/products/hmi/got/gt_works3/gt_works3 (Hämtad 2023-01-30).

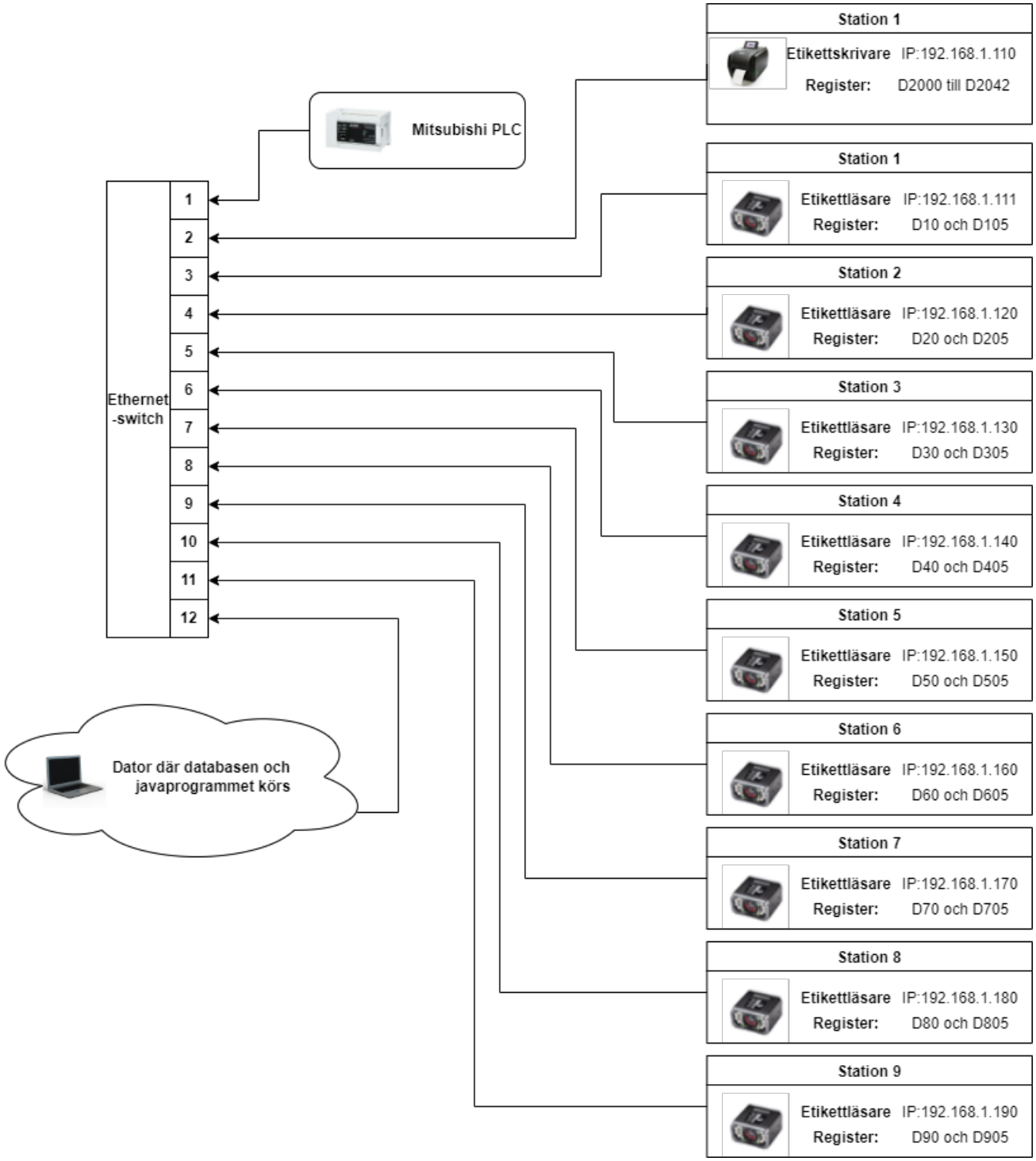
[10] TSC Printers. *TSC Console*. URL: https://info.tscprinters.com/tsc-console-en (Hämtad 2023-03-02).

[11] GS1. *EAN-13 och EAN-8*. URL: https://gs1.se/standarder-och-tjanster/ean-13/ (Hämtad 2023-01-30).

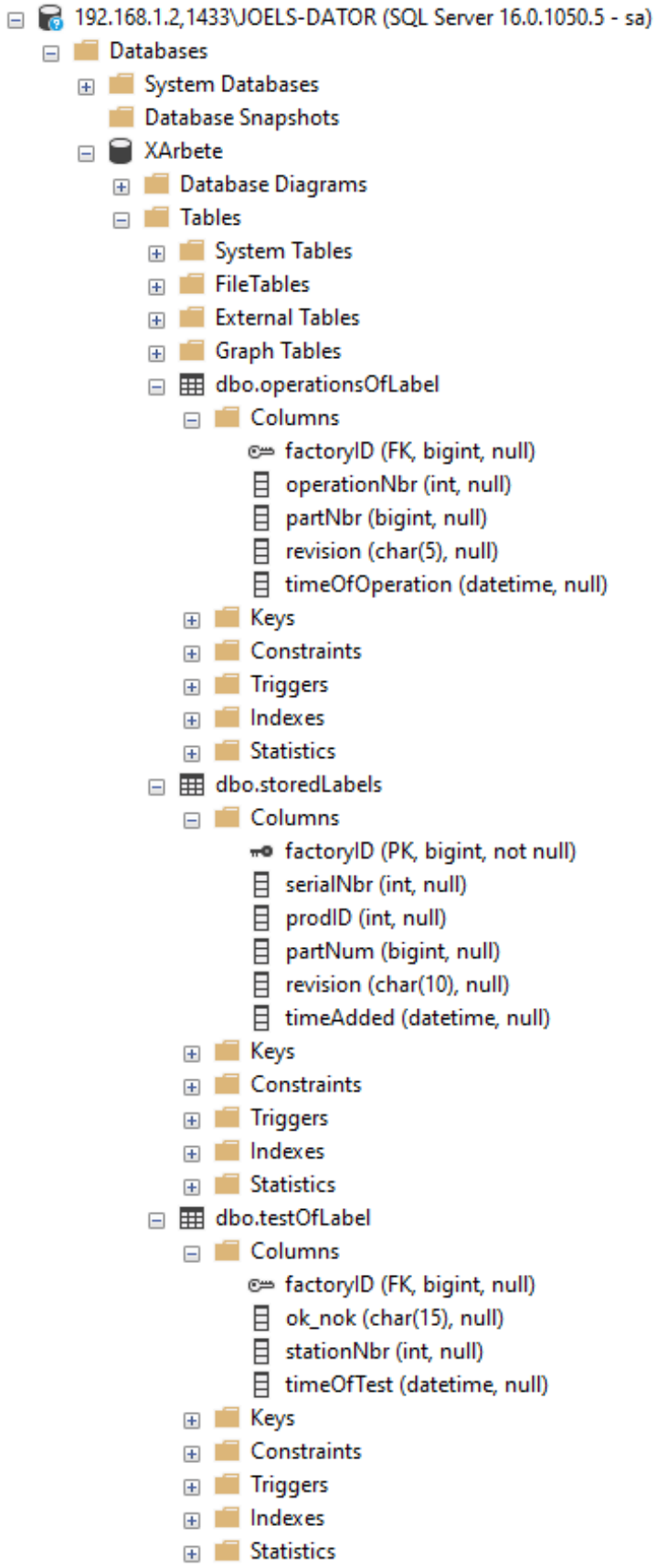
[12] Mitsubishi Electric. *MES interface module*. URL: MES Interface MELSEC iQ-R Series Product Features Programmable Controllers MELSEC | MITSUBISHI ELECTRIC FA (Hämtad 2023-03-16).

8. Appendix

8.1 Slutgiltiga spårbarhetssystemet



8.2 Databasens uppbyggnad



8.3 Mainklassen i Javaprogrammet

```
package Traceabilitysystem;

public class main {

		static PLCmanagement plcManager;
		static sqlServerConnector sqlServerConnector;
		static gui menuGui;

public static void main(String[] args) throws InterruptedException {
		sqlServerConnector = new sqlServerConnector();
		sqlServerConnector.connectToDatabase();
		plcManager = new PLCmanagement();
		plcManager.connectToPLC("192.168.1.252", 6000);
		plcManager.verifyConnection();
		menuGui = new gui();
		menuGui.createGUI();
		menuGui.updateGUI(sqlServerConnector.statusString);
		menuGui.updateGUI(plcManager.verifyConnection());
		runTraceabilitySystem();
}

public static void runTraceabilitySystem() throws InterruptedException {

while (PLCmanagement.MitsubishiFX5U.getConnectionId() != null) {
		gui.buttonsOfGUI();

		PLCmanagement.createNewJob("M102");

		PLCmanagement.addLabelToDB("M104");

		PLCmanagement.addStationInfoToDB("10");
		PLCmanagement.addStationInfoToDB("20");
		PLCmanagement.addStationInfoToDB("30");
		PLCmanagement.addStationInfoToDB("40");
		PLCmanagement.addStationInfoToDB("50");
		PLCmanagement.addStationInfoToDB("60");
		PLCmanagement.addStationInfoToDB("70");
		PLCmanagement.addStationInfoToDB("80");
		PLCmanagement.addStationInfoToDB("90");
		PLCmanagement.addStationInfoToDB("100");

		PLCmanagement.getBatchID("M161");
		PLCmanagement.getBatchID("M162");
		PLCmanagement.getBatchID("M163");
		PLCmanagement.getBatchID("M164");
		PLCmanagement.getBatchID("M165");
		PLCmanagement.getBatchID("M166");

		PLCmanagement.readNokOkFromPlc("10", "M105", "M106");
		PLCmanagement.readNokOkFromPlc("10", "M107", "M108");
		PLCmanagement.readNokOkFromPlc("20", "M109", "M110");
		PLCmanagement.readNokOkFromPlc("30", "M111", "M112");
		PLCmanagement.readNokOkFromPlc("30", "M113", "M114");
		PLCmanagement.readNokOkFromPlc("40", "M115", "M116");
		PLCmanagement.readNokOkFromPlc("50", "M117", "M118");
		PLCmanagement.readNokOkFromPlc("60", "M119", "M120");
		PLCmanagement.readNokOkFromPlc("70", "M121", "M122");
		PLCmanagement.readNokOkFromPlc("90", "M124", "M125");
		PLCmanagement.readNokOkFromPlc("100", "M126", "M127");

			}

		}
}
```

8.4 PLCmanagement-klassen i Javaprogrammet

```
package Traceabilitysystem;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import HslCommunication.Core.Types.OperateResult;
import HslCommunication.Core.Types.OperateResultExOne;
import HslCommunication.Profinet.Melsec.MelsecMcNet;

public class PLCmanagement {

		public static MelsecMcNet MitsubishiFX5U;
		public static StringBuilder stringBuilder;
		static String batchBarcode = "";
		static String labelBarcode = "";
		static int currentJobID = 0;
		static long partNumber = 0;
		static String revisionNumber = "";
		static int buttonPressCounter = 1;
		private static int fiveDigitBarcodeNumber = 1;
		private static int serialNumber = 1;

		static DBmanagement databaseManager = new DBmanagement();

public void connectToPLC(String ipNbr, int portNbr) {
		MitsubishiFX5U = new MelsecMcNet(ipNbr, portNbr);
		MitsubishiFX5U.ConnectServer();
		MitsubishiFX5U.setConnectTimeOut(100);

}

public String verifyConnection() {
		OperateResult connectionResult = MitsubishiFX5U.ConnectServer();

if (connectionResult.IsSuccess) {
		return "Connection to PLC successful";

} else {
		return "Connection to PLC failed";
		}

}

public static int readIntFromDevice(String device) {
		OperateResultExOne<short[]> read =
				MitsubishiFX5U.ReadInt16(device, (short) 10);

if (read.IsSuccess) {
		int nbrRead = read.Content[0];
		return nbrRead;

} else {
		System.out.println("Read int failed");
		return 0;
		}
}
```

```
public static long readDoubleFromDevice(String device) {
            OperateResultExOne<int[]> read = MitsubishiFX5U.ReadInt32(device, (short) 1);

if (read.IsSuccess) {
            long nbrRead = read.Content[0];
            return nbrRead;

} else {
            System.out.println("Read int failed");
            return 0;
            }
}

public static boolean readBoolFromDevice(String device) {
            OperateResultExOne<boolean[]> read =
                        MitsubishiFX5U.ReadBool(device, (short) 10);

if (read.IsSuccess) {
            boolean readBool = read.Content[0];
            return readBool;

} else {
            System.out.println("Read bool failed");
            return false;
            }

}


public static String readStringFromDevice(String device) {
            OperateResultExOne<String> read =
                        MitsubishiFX5U.ReadString(device, (short) 10);

            stringBuilder = new StringBuilder();

if (read.IsSuccess) {
            String readString = read.Content;

for (int i = 0; i < readString.length(); i++) {

            if (readString.charAt(i) == '\0') {
            break;
            }

            stringBuilder.append(readString.charAt(i));

}

            return stringBuilder.toString();
} else {

            System.out.println("Read string failed" + read.Message);
            return "";
            }

}
```

```
public static void addLabelToDB(String button) throws InterruptedException {
            boolean isButtonPressed = readBoolFromDevice(button);

if (isButtonPressed == true) {
            String createdNewLabel = createNewLabel();

            MitsubishiFX5U.Write("D2025", createdNewLabel);
            databaseManager.insertIntoTableTwoColumns(
            "storedLabels", "factoryID", "prodID", createdNewLabel,
            String.valueOf(currentJobID));

            databaseManager.updateTable(
            "storedLabels", "partNum", "factoryID",
            String.valueOf(partNumber), createdNewLabel);

            databaseManager.updateTable("storedLabels", "revision", "factoryID",
            String.valueOf(revisionNumber), createdNewLabel);

            Thread.sleep(1000);

            MitsubishiFX5U.Write(button, false);

            }
}

public static String createNewLabel() {

            java.util.Date currentDate = Calendar.getInstance().getTime();
            DateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");
            String stringDate = dateFormat.format(currentDate);

if (!(databaseManager.selectLastRow().isBlank())) {
            boolean compareDates =
            databaseManager.selectLastRow().substring(0, 8).equals(stringDate);

            if (!compareDates) {
            fiveDigitBarcodeNumber = 1;
            }
}
            String createdLabel =
            new SimpleDateFormat("yyyyMMdd").format(Calendar.getInstance().getTime())
            + String.format("%05d", fiveDigitBarcodeNumber);

while (databaseManager.getLabelNbrFromDatabase("storedLabels", "factoryID",
Long.parseLong(createdLabel)) != 0) {

            createdLabel =
            new SimpleDateFormat("yyyyMMdd").format(Calendar.getInstance().getTime())
            + String.format("%05d", fiveDigitBarcodeNumber);
            fiveDigitBarcodeNumber++;

            }

            return createdLabel;
}
```

```
public static String createNewSerialNumber() {

		String createdSerialNumber = String.format("%05d", serialNumber);
		serialNumber++;

while (databaseManager.getLabelNbrFromDatabase("storedLabels", "serialNbr",
Long.parseLong(createdSerialNumber)) != 0) {

		createdSerialNumber = String.format("%05d", fiveDigitBarcodeNumber);
		serialNumber++;

		}
		System.out.println(createdSerialNumber);
		return createdSerialNumber;

}

public static void createNewJob(String button) {
		boolean isButtonPressed = readBoolFromDevice(button);

if (isButtonPressed) {
		buttonPressCounter = buttonPressCounter + 1;
		gui.updateGUI("New job created! JobID = " + String.valueOf(currentJobID));
		gui.updateGUI("New partNum! partNum = " + String.valueOf(partNumber));
		gui.updateGUI("New revision! revisionNum = " + revisionNumber);

		}

switch (buttonPressCounter) {
case 1:
		currentJobID = 581321;
		partNumber = 6051155611L;
		revisionNumber = "A";
		break;
case 2:
		currentJobID = 588807;
		partNumber = 6051155611L;
		revisionNumber = "D";
		break;
case 3:
		currentJobID = 590337;
		partNumber = 6051155611L;
		revisionNumber = "001";
		break;
case 4:
		currentJobID = 591680;
		partNumber = 6051155611L;
		revisionNumber = "F";
		break;
case 5:
		currentJobID = 593019;
		partNumber = 6051155611L;
		revisionNumber = "C";
		break;
case 6:
		currentJobID = 667680;
		partNumber = 6051155611L;
		revisionNumber = "A";
		break;
case 7:
		currentJobID = 667681;
		partNumber = 6051155611L;
		revisionNumber = "A";
		break;
```

```
case 8:
                currentJobID = 600390;
                partNumber = 6051155611L;
                revisionNumber = "A";
                break;

default:
                buttonPressCounter = 0;
                break;
                }

}


public static void addStationInfoToDB(String stationNbr) {

                long readBarcode = MitsubishiFX5U.ReadInt64("D" + stationNbr).Content;

if (readBarcode != 0) {

                MitsubishiFX5U.Write("D" + stationNbr, 0.0);

if (databaseManager.getLabelNbrFromDatabase("storedLabels", "factoryID", readBarcode) == 0)
{
batchBarcode = String.valueOf(readBarcode);

} else {

                labelBarcode = String.valueOf(readBarcode);
                MitsubishiFX5U.Write("D" + stationNbr + "5", readBarcode);

                                }
                }

}

public static void getBatchID(String button) {

                boolean buttonPressed = readBoolFromDevice(button);

if (buttonPressed == true && button.equals("M161")) {

                for (int i = 1; i <= 5; i++) {

                String batchID = String.valueOf(readDoubleFromDevice("D1" + String.valueOf(i)
                + "1"));

                databaseManager.insertIntoTableThreeColumns("operationsOfLabel", "factoryID",
                "partNbr", "operationNbr", labelBarcode, batchID, "10");

                databaseManager.updateTable("operationsOfLabel", "revision", "factoryID",
                revisionNumber, labelBarcode);



                }

                MitsubishiFX5U.Write("M161", false);

}
```

```
if (buttonPressed == true && button.equals("M162")) {

            for (int i = 1; i <= 6; i++) {

            String batchID = String.valueOf(readDoubleFromDevice("D3" + String.valueOf(i)
            + "1"));

            databaseManager.insertIntoTableThreeColumns("operationsOfLabel", "factoryID",
            "partNbr", "operationNbr", labelBarcode, batchID, "30");
            databaseManager.updateTable("operationsOfLabel", "revision", "factoryID",
            revisionNumber, labelBarcode);

            System.out.println(batchID);

            }

            MitsubishiFX5U.Write("M162", false);
}

if (buttonPressed == true && button.equals("M163")) {

            for (int i = 1; i <= 7; i++) {

            String batchID = String.valueOf(readDoubleFromDevice("D4" + String.valueOf(i)
            + "1"));

            databaseManager.insertIntoTableThreeColumns("operationsOfLabel", "factoryID",
            "partNbr", "operationNbr", labelBarcode, batchID, "40");

            databaseManager.updateTable("operationsOfLabel", "revision", "factoryID",
            revisionNumber, labelBarcode);

            System.out.println(batchID);

            }

            MitsubishiFX5U.Write("M163", false);

}

if (buttonPressed == true && button.equals("M164")) {

            for (int i = 1; i <= 7; i++) {
            String batchID = String.valueOf(readDoubleFromDevice("D5" + String.valueOf(i)
            + "1"));

            databaseManager.insertIntoTableThreeColumns("operationsOfLabel", "factoryID",
            "partNbr", "operationNbr", labelBarcode, batchID, "50");

            databaseManager.updateTable("operationsOfLabel", "revision", "factoryID",
            revisionNumber, labelBarcode);

            }
            MitsubishiFX5U.Write("M164", false);
}
```

```
if (buttonPressed == true && button.equals("M165")) {

            String batchID = String.valueOf(readDoubleFromDevice("D611"));

            databaseManager.insertIntoTableThreeColumns("operationsOfLabel", "factoryID",
            "partNbr", "operationNbr", labelBarcode, batchID, "60");

            databaseManager.updateTable("operationsOfLabel", "revision", "factoryID",
            revisionNumber, labelBarcode);

            MitsubishiFX5U.Write("M165", false);

}

if (buttonPressed == true && button.equals("M166")) {

            String batchID = String.valueOf(readDoubleFromDevice("D911"));

            databaseManager.insertIntoTableThreeColumns("operationsOfLabel", "factoryID",
            "partNbr", "operationNbr", labelBarcode, batchID, "90");

            databaseManager.updateTable("operationsOfLabel", "revision", "factoryID",
            revisionNumber, labelBarcode);

            System.out.println(batchID);

            MitsubishiFX5U.Write("M166", false);
            }

}


public static void readNokOkFromPlc(String stationNbr, String okButton, String nokButton) {

            String status = readStringFromDevice("D1000");

if (readBoolFromDevice(okButton)) {

            databaseManager.insertIntoTableThreeColumns("testOfLabel", "factoryID",
            "ok_nok", "stationNbr", labelBarcode, status, stationNbr);

if (readBoolFromDevice("M126")) {
            String createdSerialNumber = createNewSerialNumber();
            databaseManager.updateTable(
            "storedLabels", "serialNbr", "factoryID", createdSerialNumber, labelBarcode);

            }
            MitsubishiFX5U.Write(okButton, false);
}

if (readBoolFromDevice(nokButton)) {

            databaseManager.insertIntoTableThreeColumns("testOfLabel", "factoryID",
            "ok_nok", "stationNbr", labelBarcode, status, stationNbr);

            MitsubishiFX5U.Write(nokButton, false);
                        }

            }

}
```

8.5 DBmanagement-klassen i Javaprogrammet

```
package Traceabilitysystem;

import java.sql.*;

import javax.swing.JOptionPane;

public class DBmanagement {

		PreparedStatement preparedStatement = null;
		ResultSet resultSet = null;
		String result = "";
		int sqlStep = 0;
		sqlServerConnector databaseConnector = new sqlServerConnector();
		StringBuilder stringBuilder = new StringBuilder();

public String selectAllFromTable(long searchedLabelNbr) {

		boolean endSearch = false;

		PreparedStatement preparedStatement1 = null;
		PreparedStatement preparedStatement2 = null;
		PreparedStatement preparedStatement3 = null;

		ResultSet resultSet1 = null;
		ResultSet resultSet2 = null;
		ResultSet resultSet3 = null;

		long column1;
		String column2;
		String column3;
		String column4;
		String column5;
		String column6;


while (endSearch != true) {

try {

if (sqlStep == 0) {

		preparedStatement1 = databaseConnector.conn.prepareStatement(
		"SELECT * FROM storedLabels WHERE factoryID = " + searchedLabelNbr);

		resultSet1 = preparedStatement1.executeQuery();

if (resultSet1.next() == false) {

		gui.updateGUI("\nThe table storedlabels was empty!");
} else {

		gui.updateGUI("\nfactoryID , serialNbr , prodID , partNum , revision ,
		timeAdded");

do {
		column1 = resultSet1.getLong(1);
		column2 = resultSet1.getString(2);
		column3 = resultSet1.getString(3);
		column4 = resultSet1.getString(4);
		column5 = resultSet1.getString(5);
		column6 = resultSet1.getString(6);
```

```
                stringBuilder.append(column1 + "   ,   " + column2 + "   ,   " + column3 + "
,   " + column4 + "   ,   " + column5 + "   ,   " + column6 + "\n");

} while (resultSet1.next());
}

                result = stringBuilder.toString();

                stringBuilder.setLength(0);
                sqlStep++;
                return result;
}

if (sqlStep == 1) {

                preparedStatement2 = databaseConnector.conn.prepareStatement(
                "SELECT * FROM operationsOfLabel WHERE factoryID = " + searchedLabelNbr);

                resultSet2 = preparedStatement2.executeQuery();

                if (resultSet2.next() == false) {

                gui.updateGUI("The table operationsOfLabel was empty!");
} else {
                gui.updateGUI("factoryID , operationNbr, partNbr , timeOfOperation");

do {

                column1 = resultSet2.getLong(1);
                column2 = resultSet2.getString(2);
                column3 = resultSet2.getString(3);
                column5 = resultSet2.getString(5);

                stringBuilder.append(column1 + "   ,   " + column2 + "   ,   " + column3 + " ,
                " + column5 + "\n");

} while (resultSet2.next());
}

                result = stringBuilder.toString();

                stringBuilder.setLength(0);
                sqlStep++;
                return result;
}

if (sqlStep == 2) {

                preparedStatement3 = databaseConnector.conn.prepareStatement(
                "SELECT * FROM testOfLabel WHERE factoryID = " + searchedLabelNbr);

                resultSet3 = preparedStatement3.executeQuery();

if (resultSet3.next() == false) {

                gui.updateGUI("The table testOfLabel was empty!");
} else {

                gui.updateGUI("factoryID , ok_nok , stationNbr, timeOfTest");
do {

                column1 = resultSet3.getLong(1);
                column2 = resultSet3.getString(2);
                column3 = resultSet3.getString(3);
                column4 = resultSet3.getString(4);
```

```
                stringBuilder.append(column1 + "    ,    " + column2 + "    ,    " + column3 + "
                ,    " + column4 + "\n");

} while (resultSet3.next());
}

                result = stringBuilder.toString();

                stringBuilder.setLength(0);

                endSearch = true;
                sqlStep = 0;

                return result;

}

} catch (SQLException e) {

                JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
                JOptionPane.ERROR_MESSAGE);

}

}

                return "";
}




public long getLabelNbrFromDatabase(String table, String column, long equals) {
                long labelNumber = 0;
try {
                preparedStatement = databaseConnector.conn.prepareStatement(
                "SELECT * FROM " + table + " WHERE " + column + " = ?");

                preparedStatement.setLong(1, equals);

                resultSet = preparedStatement.executeQuery();

if (resultSet.next() == false) {
                                return 0;
} else {
do {
                labelNumber = resultSet.getLong(1);

} while (resultSet.next());
                }
} catch (SQLException e) {

                JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
                JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();

                }

                return labelNumber;

}

public void insertIntoTable(String table, String column, String value) {

try {
```

```
            Statement insertStatement = databaseConnector.conn.createStatement();

            String sqlQuery = "INSERT INTO " + table + " (" + column + ")" + " VALUES " +
            "('" + value + "')";

            insertStatement.executeUpdate(sqlQuery);

            } catch (SQLException e) {

            JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
            JOptionPane.ERROR_MESSAGE);
                        e.printStackTrace();

            }

}

public void insertIntoTableTwoColumns(String table, String column1, String column2, String
value1, String value2) {

try {
            Statement insertStatement = databaseConnector.conn.createStatement();
            databaseConnector.conn.setAutoCommit(false);
            String sqlQuery = "INSERT INTO " + table + "(" + column1 + "," + column2 + ")"
            + "VALUES " + "(" + value1 + "," + "'" + value2 + "')";
            insertStatement.addBatch(sqlQuery);
            gui.updateGUI(sqlQuery);
            insertStatement.executeBatch();
            databaseConnector.conn.commit();



} catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
            JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
            }

}

public void insertIntoTableThreeColumns(String table, String column1, String column2,
String column3, String value1, String value2, String value3) {

try {
            Statement insertStatement = databaseConnector.conn.createStatement();

            databaseConnector.conn.setAutoCommit(false);

            String sqlQuery = "INSERT INTO " + table + "(" + column1 + "," + column2 + ","
+ column3 + ")" + "VALUES " + "(" + value1 + ",'" + value2 + "','" + value3 + "')";

            insertStatement.addBatch(sqlQuery);

            gui.updateGUI(sqlQuery);

            insertStatement.executeBatch();

            databaseConnector.conn.commit();

} catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
            JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
            }
```

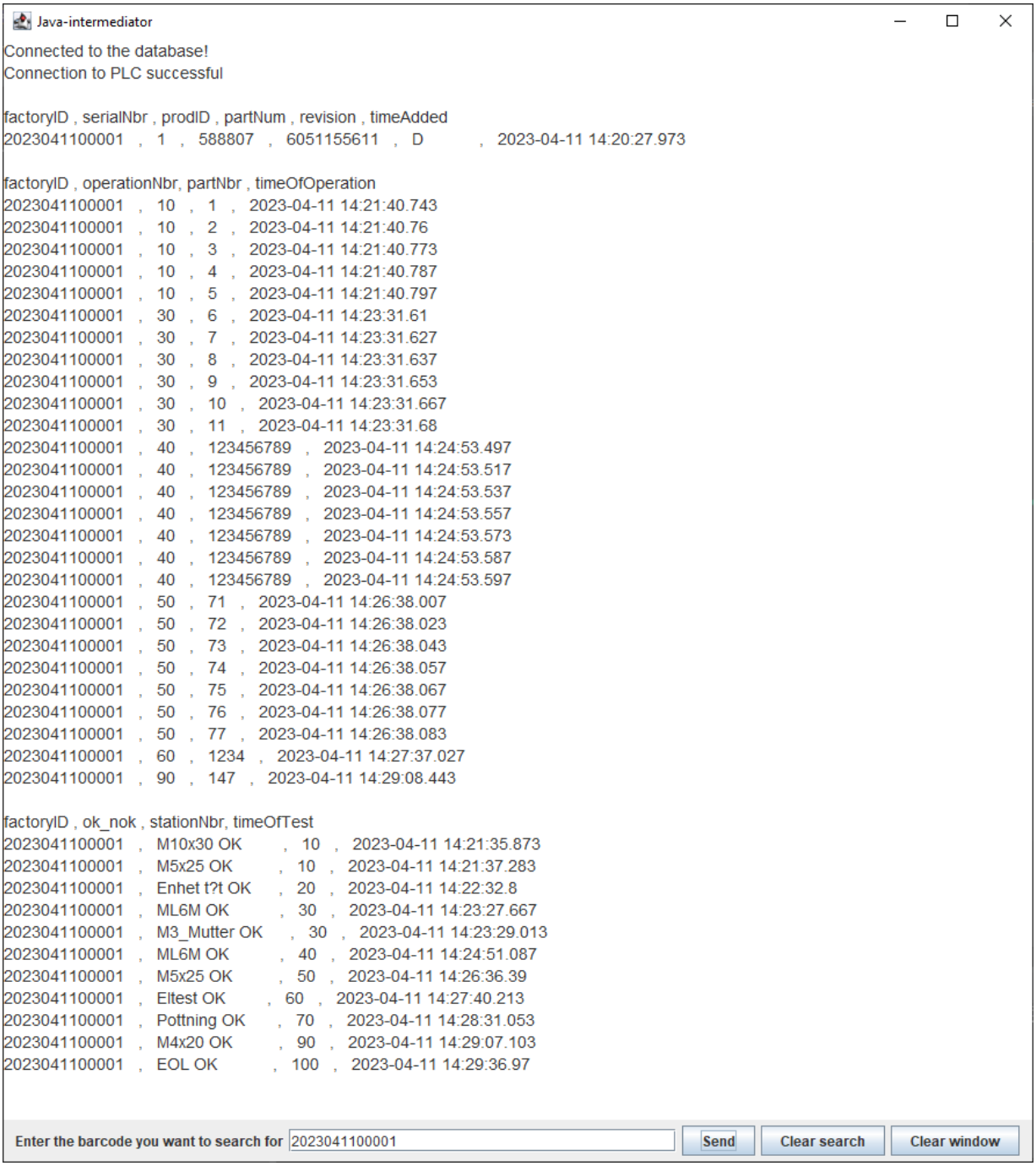
```
}

public void updateTable(String table, String column1, String column2, String value, String
equals) {
try {

              Statement updateStatement = databaseConnector.conn.createStatement();
              databaseConnector.conn.setAutoCommit(false);

              String sqlQuery = "UPDATE " + table + " SET " + column1 + " = " + "'" + value
              + "'" + " WHERE " + column2 + " = " + equals + ";";

              updateStatement.addBatch(sqlQuery);

              updateStatement.executeBatch();

              databaseConnector.conn.commit();
              gui.updateGUI(sqlQuery);
} catch (Exception e) {
              JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
              JOptionPane.ERROR_MESSAGE);
              e.printStackTrace();
              }
}

public String selectLastRow() {

              String sql = "SELECT TOP 1 * FROM storedLabels ORDER BY factoryID DESC;";
              String value = "";
try {
              PreparedStatement statement = databaseConnector.conn.prepareStatement(sql);
              ResultSet resultSet = statement.executeQuery();
              value = "";
if (resultSet.next()) {
              value = resultSet.getString(1);

              return value;
}
statement = databaseConnector.conn.prepareStatement(sql);

} catch (SQLException e) {
              e.printStackTrace();
              }

              return value;

              }
}
```

8.6 GUI-klassen i Javaprogrammet

```
package Traceabilitysystem;

import javax.swing.*;
import java.awt.*;

public class gui {

		private static JFrame frame;
		private static JPanel panel;
		private static JLabel label;
		private static JTextField textField;
		private static JButton sendSQLbutton;
		private static JButton clearTextButton;
		private static JButton clearWindowButton;
		private static JTextArea textArea;
		private final static String newline = "\n";
		private static DBmanagement dbMngr;

public static void createGUI() {
		frame = new JFrame("Java-intermediator");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setSize(1250, 750);

		textArea = new JTextArea();
		panel = new JPanel();
		label = new JLabel("Enter the barcode you want to search for");
		textField = new JTextField(30);

		sendSQLbutton = new JButton("Send");
		clearTextButton = new JButton("Clear search");
		clearWindowButton = new JButton("Clear window");

		panel.add(label);
		panel.add(textField);
		panel.add(sendSQLbutton);
		panel.add(clearTextButton);
		panel.add(clearWindowButton);

		frame.getContentPane().add(BorderLayout.SOUTH, panel);
		frame.getContentPane().add(BorderLayout.CENTER, textArea);
		frame.setVisible(true);

		dbMngr = new DBmanagement();

}

public static void updateGUI(String updateText) {

		textArea.append(updateText + newline);
}


public static void buttonsOfGUI() {

if (sendSQLbutton.getModel().isPressed() && ! textField.getText().equals(""))) {
		long searchedLabelNbr = Long.parseLong(textField.getText());

		for (int i = 0; i < 3; i++) {
		 updateGUI(dbMngr.selectAllFromTable(searchedLabelNbr));
		}
}
```

```
if (clearTextButton.getModel().isPressed()) {
            textField.setText("");
            }

if (clearWindowButton.getModel().isPressed()) {
            textArea.setText("");
}

            }
}
```

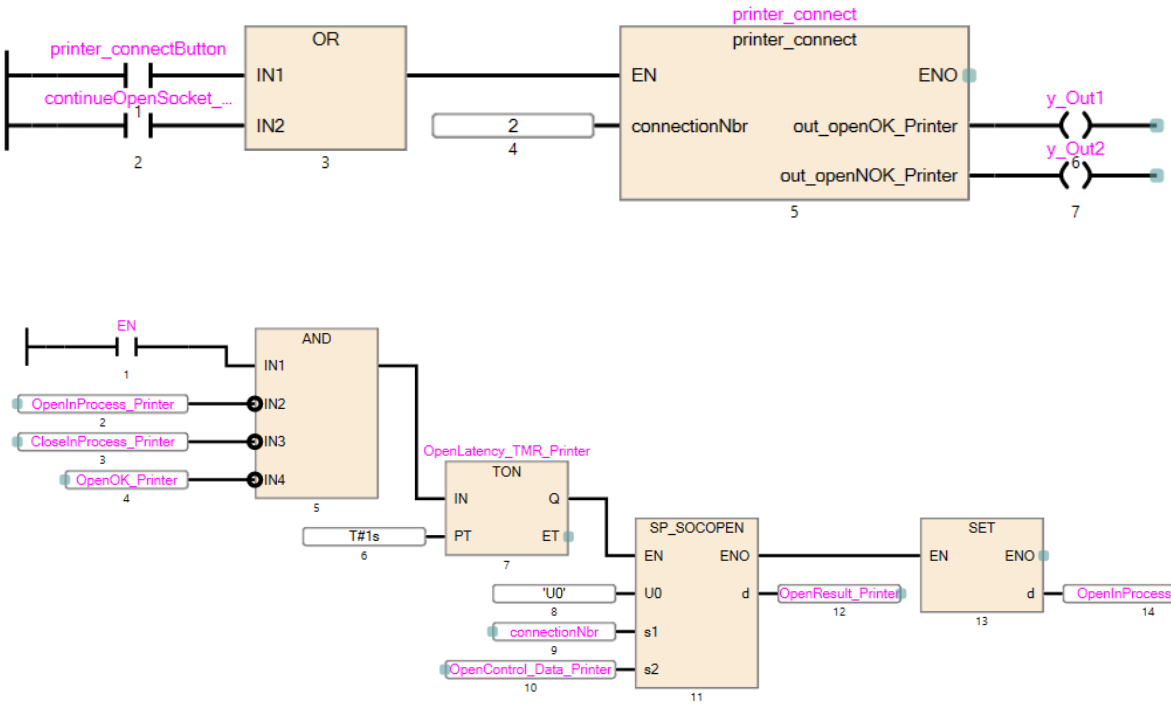
8.7 GUI:t efter sökning utförts



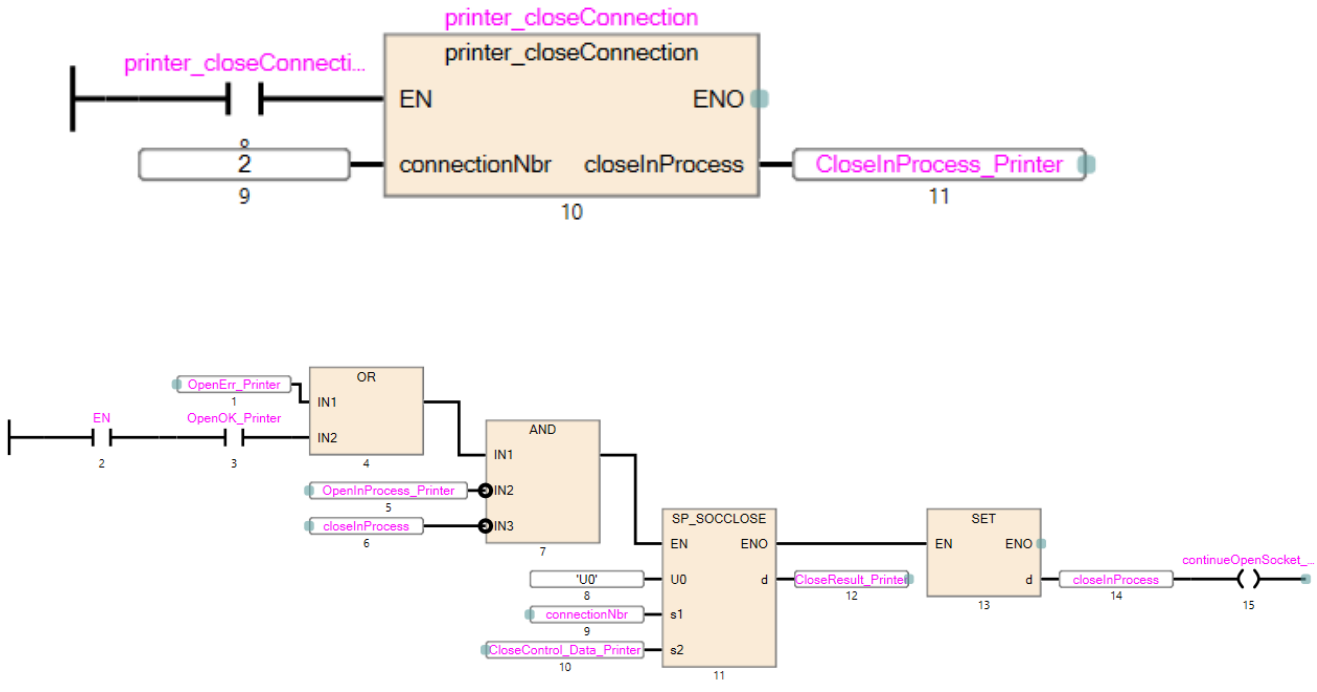
8.8 sqlServerConnector-klassen i Javaprogrammet

```
package Traceabilitysystem;

import java.sql.DriverManager;

public class sqlServerConnector {

		public static java.sql.Connection conn;

		public static String statusString;


public void connectToDatabase() {

try {
		Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

		conn = DriverManager.getConnection("jdbc:sqlserver://JOELS-
		DATOR;DatabaseName=XArbete;user=XYZW;password={ABCDE};encrypt=True;trustServer
		Certificate=true;allowMultiQueries=true");

} catch (Exception e) {
		e.printStackTrace();
		statusString = e.toString();
		}

if (conn != null) {
		statusString = "Connected to the database!";
		}

}
}
```

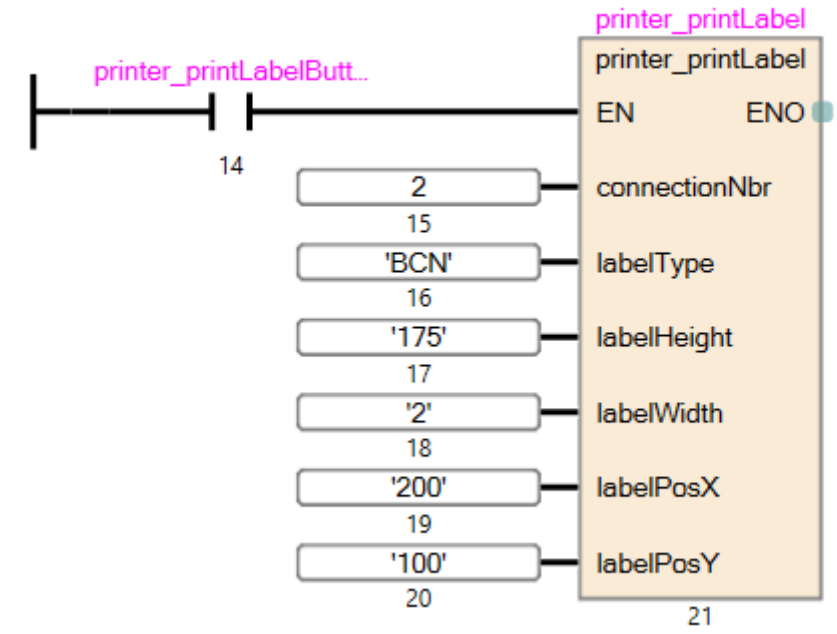
8.9 Printer_connect funktionsblocket

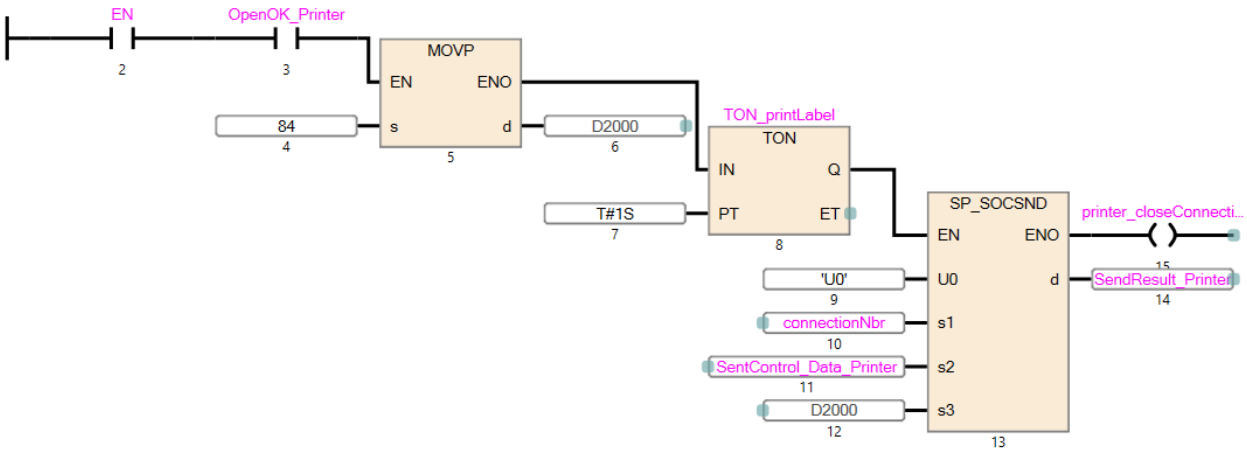


8.10 Printer_closeConnection funktionsblocket



8.11 Printer_printLabel funktionsblocket

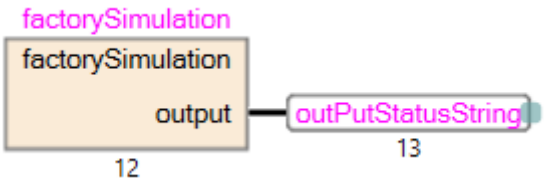




8.12 ST-kod i printer_printLabel för att kunna skriva ut etikett

```
IF EN AND OpenOK_Printer THEN;

            MOVP(TRUE, 74, D2000);
            STRINGMOVP(TRUE, '^', D2001);
            STRINGMOVP(TRUE, 'XA', D2002);
            STRINGMOVP(TRUE, '^', D2004);
            STRINGMOVP(TRUE, 'FO', D2005);
            STRINGMOVP(TRUE, labelPosX, D2007);
            STRINGMOVP(TRUE, ',', D2009);
            STRINGMOVP(TRUE, labelPosY, D2010);
            STRINGMOVP(TRUE, '^', D2011);
            STRINGMOVP(TRUE, 'BY', D2012);
            STRINGMOVP(TRUE, labelWidth, D2014);
            STRINGMOVP(TRUE, '^', D2015);
            STRINGMOVP(TRUE, labelType, D2016);
            STRINGMOVP(TRUE, ',', D2019);
            STRINGMOVP(TRUE, labelHeight, D2020);
            STRINGMOVP(TRUE, '^', D2022);
            STRINGMOVP(TRUE, 'FD', D2023);
            //MOVP(TRUE, YYYYMMDDXYZWC, D2025);
            STRINGMOVP(TRUE, '^', D2033);
            STRINGMOVP(TRUE, 'FS', D2034);
            STRINGMOVP(TRUE, '^', D2036);
            STRINGMOVP(TRUE, 'XZ', D2037);

END_IF;
```

8.13 factorySimulation funktionsblocket



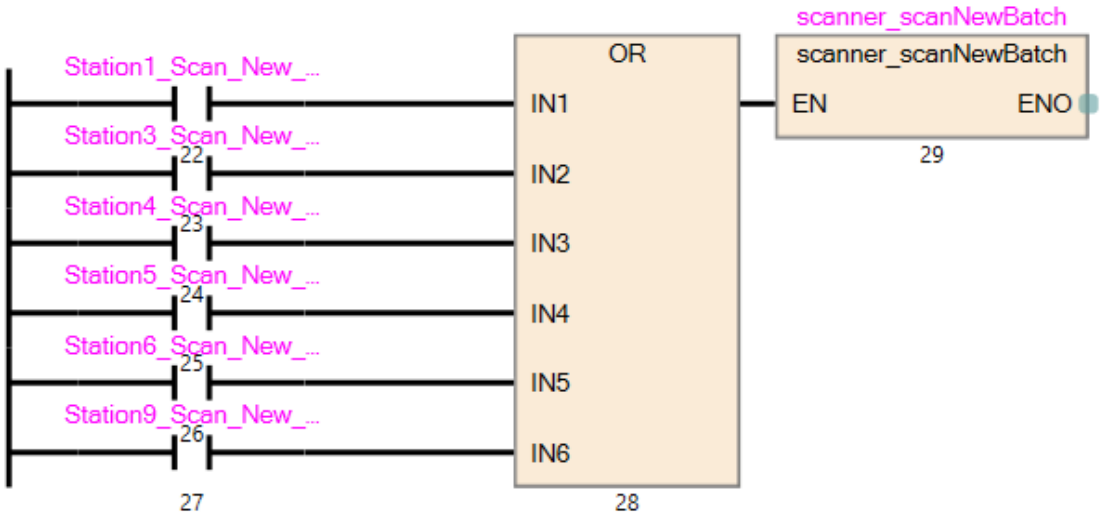
8.14 ST-kod i factorySimulation

```
//M10x30 OK
IF Station1_M10x30_OK_Button = TRUE THEN;
            output := 'M10x30 OK';
END_IF;

//M10x30 NOK
IF Station1_M10x30_NOK_Button = TRUE THEN;
            output := 'M10x30 NOK'
END_IF;

//M5x25 OK
IF Station1_M5x25_OK_Button = TRUE THEN;
            output := 'M5x25 OK';
END_IF;

//M5x25 NOK
IF Station1_M5x25_NOK_Button = TRUE THEN;
            output := 'M5x25 NOK';
END_IF;
//Enhet tät OK
IF Station2_Enhet_tät_OK_Button = TRUE THEN;
            output := 'Enhet tät OK';
END_IF;

//Enhet tät NOK
IF Station2_Enhet_tät_NOK_Button = TRUE THEN;
            output := 'Enhet tät NOK';
END_IF;

//ML6M OK
IF Station3_ML6M_OK_Button = TRUE THEN;
            output := 'ML6M OK';
END_IF;

//ML6M NOK
```

```
IF Station3_ML6M_NOK_Button = TRUE THEN;
            output := 'ML6M NOK';
END_IF;

//M3_Mutter OK
IF Station3_M3_Mutter_OK_Button = TRUE THEN;
            output := 'M3_Mutter OK';
END_IF;

//M3_Mutter NOK
IF Station3_M3_Mutter_NOK_Button = TRUE THEN;
            output := 'M3_Mutter NOK';
END_IF;

//ML6M OK
IF Station4_ML6M_OK_Button = TRUE THEN;
            output := 'ML6M OK';
END_IF;

//ML6M NOK
IF Station4_ML6M_NOK_Button = TRUE THEN;
            output := 'ML6M NOK';
END_IF;

//M5x25 OK
IF Station5_M5x25_OK_Button = TRUE THEN;
            output := 'M5x25 OK';
END_IF;

//M5x25 NOK
IF Station5_M5x25_NOK_Button = TRUE THEN;
            output := 'M5x25 NOK';
END_IF;

//Eltest OK
IF Station6_Eltest_OK_Button = TRUE THEN;
            output := 'Eltest OK';
END_IF;

//Eltest NOK
IF Station6_Eltest_NOK_Button = TRUE THEN;
            output := 'Eltest NOK';
END_IF;

//Pottning OK
IF Station7_Pottning_OK_Button = TRUE THEN;
            output := 'Pottning OK';
END_IF;
```

```
//Pottning NOK
IF Station7_Pottning_NOK_Button = TRUE THEN;
            output := 'Pottning NOK';
END_IF;

//Härdtid OK
IF Station8_Härdtid_OK_Button = TRUE THEN;
            output := 'Härdtid OK';
END_IF;

//M4x20 OK
IF Station9_M4x20_OK_Button = TRUE THEN;
            output := 'M4x20 OK';
END_IF;

//M4x20 NOK
IF Station9_M4x20_NOK_Button = TRUE THEN;
            output := 'M4x20 NOK';
END_IF;

//EOL OK
IF Station10_EOL_OK_Button = TRUE THEN;
            output := 'EOL OK';
END_IF;

//EOL NOK
IF Station10_EOL_NOK_Button = TRUE THEN;
            output := 'EOL NOK';
END_IF;
```

8.15 scanner_scanNewBatch funktionsblocket



8.16 ST-kod i scanner_scanNewBatch för att kunna skanna och spara batcher

```
IF EN THEN;

IF Station1_Scan_New_Batch_Button THEN;

scanner_currentValue := D10;

IF Station1_Scan_Tank_BatchID_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D10, D111);
            RST(TRUE, D10);
            RST(TRUE, D11);
            RST(TRUE, Station1_Scan_Tank_BatchID_Button);
END_IF;

IF Station1_Scan_M10x30_BatchID_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D10, D121);
            RST(TRUE, D10);
            RST(TRUE, D11);
            RST(TRUE, Station1_Scan_M10x30_BatchID_Button);
END_IF;

IF Station1_Scan_EPDM_Seal_BatchID_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D10, D131);
            RST(TRUE, D10);
            RST(TRUE, D11);
            RST(TRUE, Station1_Scan_EPDM_Seal_BatchID_Button);
```

```
END_IF;

IF Station1_Scan_Patron_BatchID_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D10, D141);
            RST(TRUE, D10);
            RST(TRUE, D11);
            RST(TRUE, Station1_Scan_Patron_BatchID_Button);
END_IF;

IF Station1_Scan_M5x25_Skruv_BatchID_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D10, D151);
            RST(TRUE, D10);
            RST(TRUE, D11);
            RST(TRUE, Station1_Scan_M5x25_Skruv_BatchID_Button);
END_IF;

END_IF;

IF Station3_Scan_New_Batch_Button THEN;

scanner_currentValue := D30;

IF Station3_Scan_ML6M_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D30, D311);
            RST(TRUE, D30);
            RST(TRUE, D31);
            RST(TRUE, Station3_Scan_ML6M_Batch_Button);
END_IF;

IF Station3_Scan_Kylpad_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D30, D321);
            RST(TRUE, D30);
            RST(TRUE, D31);
            RST(TRUE, Station3_Scan_Kylpad_Batch_Button);
END_IF;

IF Station3_Scan_IsolerHylsor_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D30, D331);
            RST(TRUE, D30);
            RST(TRUE, D31);
            RST(TRUE, Station3_Scan_IsolerHylsor_Batch_Button);
END_IF;

IF Station3_Scan_Spacers_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D30, D341);
            RST(TRUE, D30);
            RST(TRUE, D31);
            RST(TRUE, Station3_Scan_Spacers_Batch_Button);
END_IF;
```

```
IF Station3_Scan_Brickor_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D30, D351);
            RST(TRUE, D30);
            RST(TRUE, D31);
            RST(TRUE, Station3_Scan_Brickor_Batch_Button);
END_IF;

IF Station3_Scan_M3_Mutter_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D30, D361);
            RST(TRUE, D30);
            RST(TRUE, D31);
            RST(TRUE,  Station3_Scan_M3_Mutter_Batch_Button);
            END_IF;

END_IF;

IF Station4_Scan_New_Batch_Button THEN;

scanner_currentValue := D40;

IF Station4_Scan_ML6M_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D40, D411);
            RST(TRUE, D40);
            RST(TRUE, D41);
            RST(TRUE, Station4_Scan_ML6M_Batch_Button);
END_IF;

IF Station4_Scan_KabelE11_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D40, D421);
            RST(TRUE, D40);
            RST(TRUE, D41);
            RST(TRUE, Station4_Scan_KabelE11_Batch_Button);
END_IF;

IF Station4_Scan_KabelE12_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D40, D431);
            RST(TRUE, D40);
            RST(TRUE, D41);
            RST(TRUE, Station4_Scan_KabelE12_Batch_Button);
END_IF;

IF Station4_Scan_KabelE21_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D40, D441);
            RST(TRUE, D40);
            RST(TRUE, D41);
            RST(TRUE, Station4_Scan_KabelE21_Batch_Button);
END_IF;
```

```
IF Station4_Scan_KabelE22_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D40, D451);
            RST(TRUE, D40);
            RST(TRUE, D41);
            RST(TRUE, Station4_Scan_KabelE22_Batch_Button);
END_IF;

IF Station4_Scan_TempSensor_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D40, D461);
            RST(TRUE, D40);
            RST(TRUE, D41);
            RST(TRUE, Station4_Scan_TempSensor_Batch_Button);
END_IF;

IF Station4_Scan_Kopplingsblack_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D40, D471);
            RST(TRUE, D40);
            RST(TRUE, D41);
            RST(TRUE, Station4_Scan_Kopplingsblack_Batch_Button);
END_IF;


END_IF;

IF Station5_Scan_New_Batch_Button THEN;

scanner_currentValue := D50;

IF Station5_Scan_Lock_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D50, D511);
            RST(TRUE, D50);
            RST(TRUE, D51);
            RST(TRUE, Station5_Scan_Lock_Batch_Button);
END_IF;

IF Station5_Scan_Packning_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D50, D521);
            RST(TRUE, D50);
            RST(TRUE, D51);
            RST(TRUE, Station5_Scan_Packning_Batch_Button);
END_IF;

IF Station5_Scan_HVKontakt_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D50, D531);
            RST(TRUE, D50);
            RST(TRUE, D51);
            RST(TRUE, Station5_Scan_HVKontakt_Batch_Button);
END_IF;
```

```
IF Station5_Scan_LVKontakt_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D50, D541);
            RST(TRUE, D50);
            RST(TRUE, D51);
            RST(TRUE, Station5_Scan_LVKontakt_Batch_Button);
END_IF;

IF Station5_Scan_Tycolas_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D50, D551);
            RST(TRUE, D50);
            RST(TRUE, D51);
            RST(TRUE, Station5_Scan_Tycolas_Batch_Button);
END_IF;

IF Station5_Scan_M5_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D50, D561);
            RST(TRUE, D50);
            RST(TRUE, D51);
            RST(TRUE, Station5_Scan_M5_Batch_Button);
END_IF;

IF Station5_Scan_M5x25_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D50, D571);
            RST(TRUE, D50);
            RST(TRUE, D51);
            RST(TRUE, Station5_Scan_M5x25_Batch_Button);
END_IF;

END_IF;

IF Station6_Scan_New_Batch_Button THEN;

scanner_currentValue := D60;

IF Station6_Scan_PCBID_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D60, D611);
            RST(TRUE, D60);
            RST(TRUE, D61);
            RST(TRUE, Station6_Scan_PCBID_Batch_Button);
END_IF;

END_IF;
```

```
IF Station9_Scan_New_Batch_Button THEN;

scanner_currentValue := D90;

IF Station9_Scan_M4x20_Batch_Button AND scanner_currentValue <> 0 THEN;
            DMOV(TRUE, D90, D911);
            RST(TRUE, D90);
            RST(TRUE, D91);
            RST(TRUE, Station9_Scan_M4x20_Batch_Button);
END_IF;

END_IF;

END_IF;
```