7-11-2015

# Design of a Control System for a Reconfigurable Engine Assembly Line

Hamid Tabti
*University of Windsor*

# Design of a Control System for a Reconfigurable Engine Assembly Line

By

**Hamid Tabti**

A Thesis

Submitted to the Faculty of Graduate Studies through the
Department of Industrial and Manufacturing Systems Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2015

# Design of a Control System for a Reconfigurable Engine Assembly Line

By

**Hamid Tabti**

APPROVED BY:

_____

**J. Wu**, Outside Program Reader
Electrical & Computer Science

_____

**A. Djuric,** Program Reader
Wayne State University, Cross-appointed to IMSE

_____

**H. ElMaraghy**, Advisor
Industrial & Manufacturing System Engineering

May 15,2015

# DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

## ABSTRACT

Today's automotive manufacturing environment is dynamic, not long ago, plants produced engines for decades, with minor modifications warranting slight manufacturing line rework. Conversely, today's changing trends require machines and complete engine line overhauls rendering initial setups obsolete. Automakers compete to satisfy government regulations for best mileage and also lower manufacturing cost, thus the adoption of Reconfigurable Manufacturing Systems (RMS). Information Technology (IT) and Controls are growing closer with the line of demarcation disappearing in manufacturing. Controls are benefiting from opportunities in IT, hardware and software. Component-based software suitable for RMS modularity and plug-and-play hardware/software components has gained decades of popularity in the software industry. This thesis implements distributed controls imbedding component-based technology and IEC 61311-3 function block standard for automotive engine assembly, which will contribute to these developments. The control architecture provides reconfigurability which is lacking in current manufacturing systems. The research imbeds: 1- Reconfigurability - Fitting RMS-designed hardware towards new manufacturing, 2- Reusability - Building software library for reuse across assembly lines, and 3- Plug-and-Play - Embedding easy to assemble software components (function blocks).

# DEDICATION

*To My Parents*

*To My Wife and Kids*

# ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to Ford Motor Company for their continuous and generous support to complete this Master thesis. Special thanks go to Mr. Mike Bastian, Controls Manager at Ford Motor Company for his support, help and interest in the research. The years I spent at the IMSE center of the University of Windsor were a great enlightening learning experience for me. I certainly refined my knowledge of manufacturing systems and acquired an edge to explore research and development opportunities while applying my experience on the reality of the plant floor against academic principles. The research in this thesis was inspired by the organization needs and enhanced by academia's perpetual constructive feedback.

I would like to express my genuine gratitude to my supervisor, Dr. Hoda ElMaraghy, and to my supervisory committee members, Dr. A.Djuric and Dr. J. Wu. I believe that my supervisor provided me with her best of assistance, advice, and guidance to getting my research done. Dr. ElMaraghy's vision, experience, and directions were essential to establish the major frame to this work. Special thanks go to Dr.Waguih ElMaraghy as well, for support, help and constructive criticism allowing me to stay the course of completing this research.

I am also very grateful to my collaboration all along with other researchers at the IMSE laboratory. I had indeed the opportunity to work with highly talented and extremely intelligent researchers, from whom I learned a lot. The combined academic-industrial fertile research environment I worked in hosted the research reported in this thesis

Finally I would like to thank my wife for patiently sacrificing her time as in providing full attention to our five children, which in particular allowed me to stay late at the university, after working hours and over weekends, progressing with my research. I dedicate this modest achievement to them and to my family in Algeria

whom while away encouraged me to persevere in completing this degree. This thesis is offered to my wife, my children, my late mother, my father, my siblings, whom committed much of their time not to say their entire lives help shaping the person I am with unconditional love.

# Table of Contents

LIST OF TABLES

LIST OF FIGURES

# LIST OF ABBREVIATIONS/SYMBOLS

| | |
|---|---|
| IPA | Application Programming Interface |
| ASRS | Automatic Storage and Retrieval Systems |
| AGV | Automatic-Guided Vehicles |
| AD | Axiomatic Design |
| BOM | Bill of Materials |
| CBS | Component-based Software Technology |
| CAD | Computer-aided Design |
| CAFE | Corporate Average Fuel Economy |
| CR | Customer Requirements |
| DM | Design Matrix |
| DFM | Design for manufacturability |
| DP | Design Parameters |
| DARTS | Design Approach for Real-Time Systems |
| DS | Docking Station |
| FPS | Ford Production System |
| FR | Functional Requirements |
| FB | Function Blocks |
| FBD | Function Block Diagram |
| FC | Function Call |
| HMI | Human Machine Interfaces |
| HMS | Holonic Manufacturing Systems |
| IT | Information Technology |

| | |
|---|---|
| I/O | Input/output |
| ISO | International Standard Organization |
| IL | Instruction List |
| IEEE | Institute of Electrical and Electronics Engineers |
| LD | Ladder Diagram |
| LLD | Ladder Logic Diagrams |
| MODICON | Modular Digital Controller |
| NHTSA | National Highway Traffic and Safety Administration |
| NBS | National Bureau of Standards |
| OMAC | Open Modular Architecture Controls at GM Powertrain |
| OOP | Object Oriented Programming |
| OSE | Open System Environment |
| OAC | Open Architecture Control |
| PN | Petrinet |
| PLC | Process Logic Controller |
| RFID | Radio Frequency Identification |
| RMS | Reconfigurable Manufacturing Systems |
| RTSM | Real-Time Specification Methodology |
| RAC | Reusable Automation Components |
| SFC | Sequential Function Charts |
| ST | Structured Text |
| UML | Unified Modeling Language |
| TPS | Toyota uses Toyota Production System |

# CHAPTER 1:

# Engine Assembly Lines and Machines

## 1 Introduction

This chapter focuses on automobile engine manufacturing plants. It presents different layouts for engine assembly and stations composing production lines. The need for Reconfigurability is also discussed.

## 1.1 Engine Assembly Overview

An engine assembly line is a complex manufacturing system composed of a few hundred stations linked with a conveyor on which pallets travel from station to station. Each station performs an assembly operation by adding a component or a subassembly to the engine manufactured. The process begins by loading an engine block to an empty pallet, then parts are added in sequence till final assembly; it ends by offloading the finished and tested engine to a shipping rack.

## 1.2 Engine Assembly Layouts

The layout chosen for an engine assembly depends on production volumes. Two different layouts are common in the industry, single and multiple loops.

**A Single Loop Layout:** This type of layout is better suited for low volume lines, where cycle time is long and disturbances resulting from downtime have little effects. An example of a single layout for engine assembly is shown on Figure 1.1.

**B Multiple Loops Layout:** An engine assembly is divided into multiple loops, usually three or four. A first short block loop consists of engines without cylinder heads and timing component. A second long block loop may encompass two loops for an engine before dressing. A later or final dress loop installs wire harness and vacuum hoses. Figure 1.2 illustrates an example of multiple loop engine assembly.

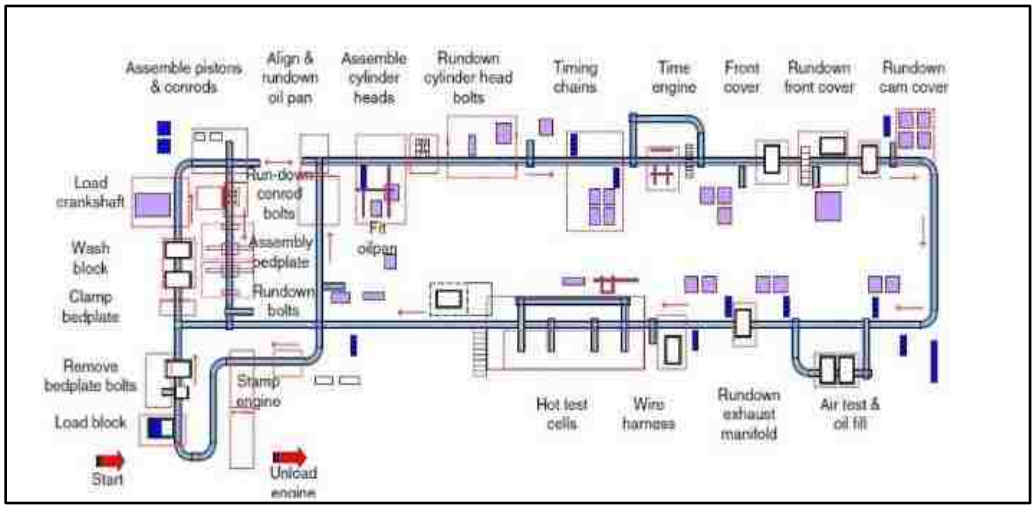



Figure 1.1: Single Loop Engine Assembly Line (*from J.A. Krause Machinenfabrik GmbH*)

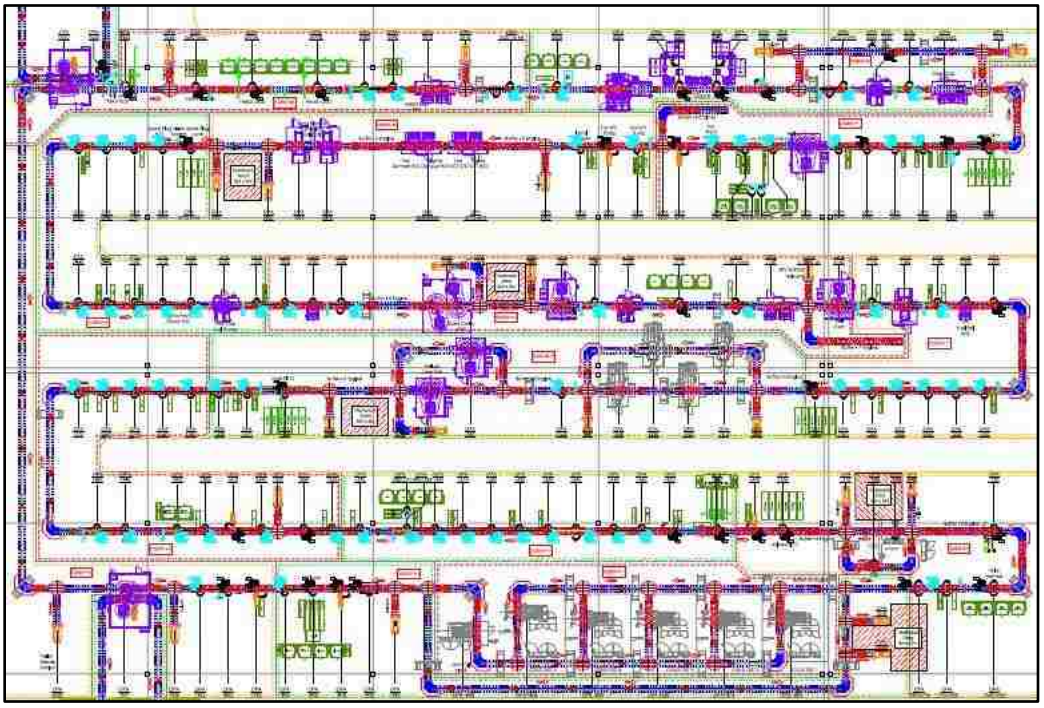


Figure 1.2: Three-loop Engine Assembly Line (private communication)

## 1.3 Engine Assembly Station Types

There are two types of engine assembly stations: 1- Manual and 2- Automatic.

**A Manual Assembly Stations:** Today's engine assembly lines involve 150 to 180 workers performing manual tasks using specific tools for each station (powered equipment like rundown tools, or manual ones to handle parts for assembly). This type of station is the most flexible, as operators adapt easily to any situation. A human has better dexterity than any machine or robot; he or she also learns and gains experience by helping detect potential anomalies and developing solutions. An example of a manual station is shown on Figure 1.3.

Figure 1.3: Typical Manual Work Station used in assembly line

Each manual station is built as a module, fitted with a Human Machine Interface (HMI) controlled by a Programmable Logic Controller (PLC), while hosting tooling suitable for the assembly task.

**B Automatic Assembly Stations:** These stations include fully automatic machines which complete tasks without operator intervention, and semi-automatic stations where machines perform portions of an assembly leaving the remainder for operators to complete by unloading or loading the assembly in progress to other machines for performing complementary tasks. Automation is usually assigned to repetitive and tedious tasks which are difficult on operators. Engine assembly lines include many types of automatic stations such as:

- Robotic Cells: Typically consisting of an industrial six-axis robot performing tasks, for instance a material handling function such as pick and place;
- Conveyors: Properly fitted with stops and controlled by PLC to move pallets from station to station;
- Buffers: Usually used to store pallets in process;
- Elevators: Properly placed to transfer pallets over lines between loops;
- Bolt and Dowel Feeders: Automatically embedded to sort bolts or dowels and feed them to machines for install and rundown;
- Poke Yoke: Relied upon to assure Quality Control (QC) functions, in terms of simple probing or vision system detecting assembly errors;
- Test Stations: Introduced at certain stages of an assembly to check functionality throughout production, for leaks, compression, running torque, or complete engine also called Cold Test using an electrical drive to test the produced engine at certain speeds;
- Rundown Stations: Automatically inserted to run down bolts for instance, to specification;
- Press: Crafted to consistently press dowels into an engine block or cylinder head following a specified depth and/or force;
- Room Temperature Vulcanizing (RTV) Applications: Engineered robot stations applying a controlled amount of RTV to a surface -

Vision systems are used to guide presence and location of RTV - An example of an automatic station is shown on Figure 1.4.


Figure 1.4: Automatic Assembly Cell (*Hirata Assembly Systems, http://www.hirata.co.jp/en/*)

Automatic stations are more complex than manual stations; they are also built as modules with each machine hosting PLC and HMI.

## 1.4 Engine Variants

A typical engine assembly line is designed to run many variants, usually 2+1; where +1 refers to a future engine within the same family yet to be designed. An example of an existing line is illustrated on Figure 1.5. Note that beginning with one engine block could result in multiple engine derivatives. The first stations on the short block handle one part type while stations in the first and final handle all engine derivatives. Derivatives multiply by adding more engine blocks to the production line, and could reach 15 to 20 at any point in the life cycle of the manufacturing system. Consequently the +1 variant forces the engine assembly designers to implement the RMS paradigm to facilitate integration of anticipated changes and variants.

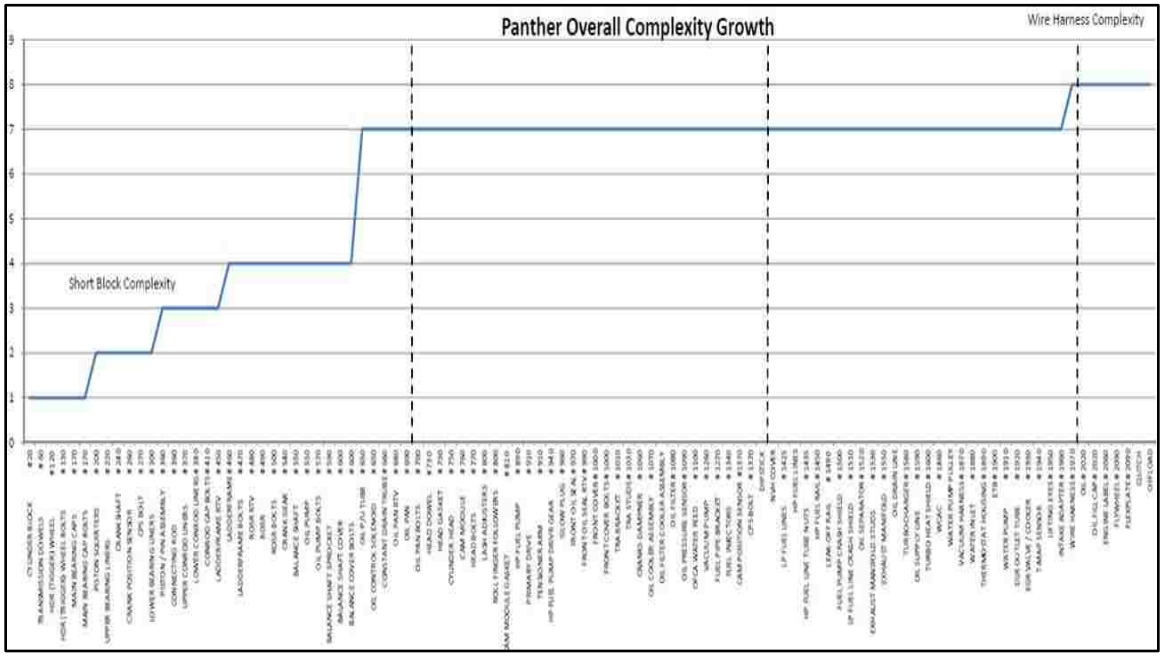



Figure 1.5: Engine Variants through a Typical Engine Assembly Line

## 1.5 Enablers for Reconfigurability in Engine Assembly Lines

Many characteristics of Reconfigurable Manufacturing Systems (RMS) are implemented in the design of new assembly lines in order to manage perceived complexities resulting from variants and future products. A new engine line needs hardware and software enablers to produce required variants. Hard enablers typically used in engine assembly lines are listed below:

a) Nested Pallet: Designers use adaptor plates with two faces to handle potentially completely different engine blocks when running multiple engine blocks on a same pallet: Each face is designed with proper locators for a given block, to rotate when another block type is produced;
b) Modular Station: Modularity is an RMS characteristic allowing relocation of stations if required, while duplicating stations and installing them in parallel for volume increases;

c) Space Protect: Spaces are left for future stations handling +1 variety with conveyors to insert strategically throughout a line in order to accommodate additional stations.

d) Manual Stations: Operator stations are most flexible: Humans handle any assembly with proper tooling.

e) Industrial Robots: Fitting a six DOF robot in an assembly cell with a changeable end effector provides necessary flexibility to handle multiple variants.

Today's trends in controls software evolve towards modularity to accomplish desired architectures. Figure 1.6 shows an example of Enterprise PC distributed Control System.

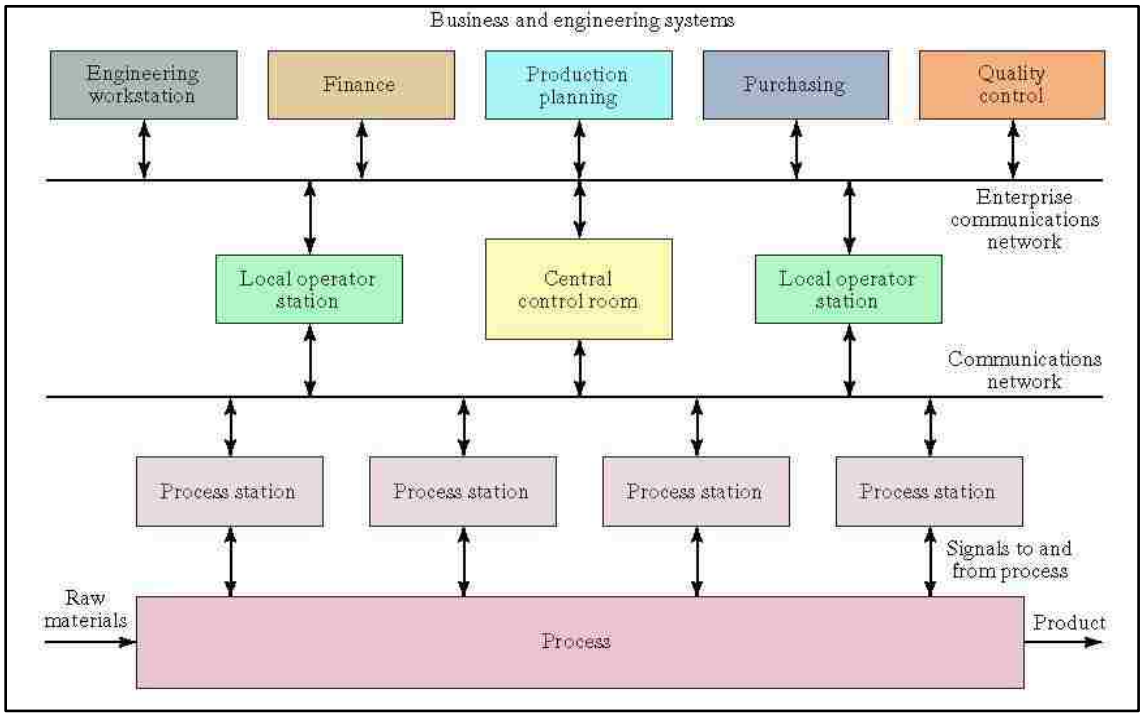



Figure 1.6: Enterprise PC Distributed Control System (*MGroover, 2000*)

A controls system of an engine assembly line is complex and composed of hardware elements connected through a communication bus. Figure 1.7 depicts typical hardware for a distributed control system. Such hardware multiplies by at least a hundred for a typical engine assembly system indicating associated complexities.

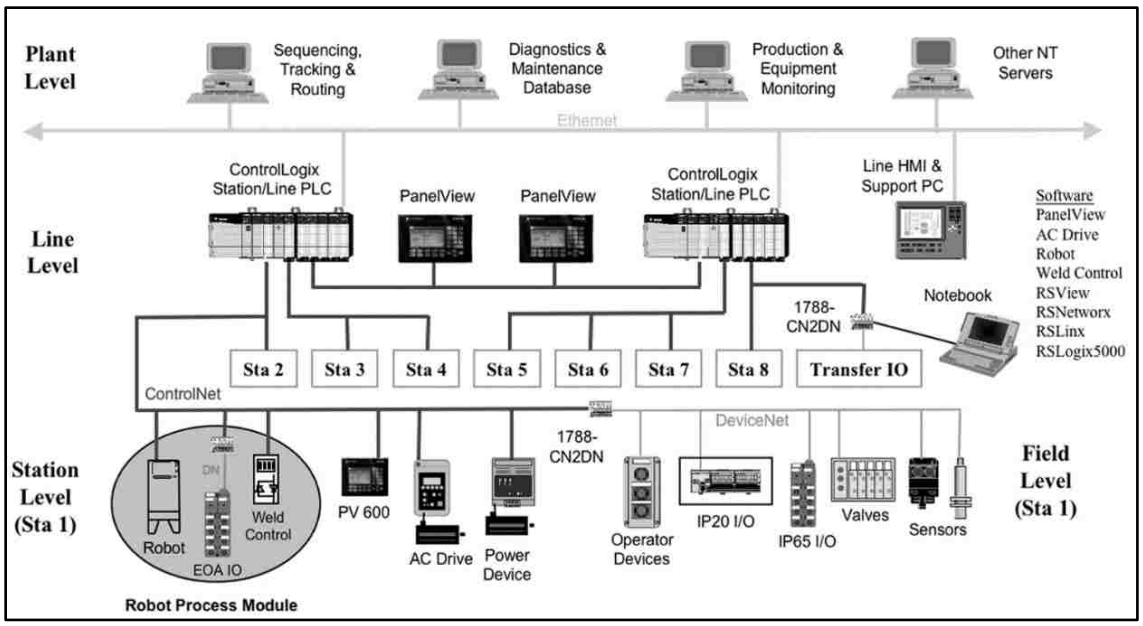



Figure 1.7: Distributed Control System Hardware *(http://ab.rockwellautomation.com)*

## 1.6 Motivation for Research

In current automotive assembly circles, control software is usually customized to each machine; similar machines could have different pieces of control logic. Programmers usually use experience from previous projects. They consider similar machines coded in the past then modify the associated coding to suit the new machinery. The result is always a tailored program that is rigid, station specific, and inflexible to change in case of machine hardware modification. This approach triggers high costs and decreases innovation. Control systems need to adopt the latest developments in IT for increased effectiveness while responding quickly to reconfigurability requirements. Modularity is an essential ingredient in controls software to translate from station to station.

According to Chan *et al.* (2000), Object-oriented Programming (OOP) is the dominantly used software technology in the design of manufacturing controls; Mehrabi *et al.* (2002) believes that OOP is also the choice for RMS controls. Agent-based technologies are very effective in dealing with unexpected events. Agents could be added or replaced with ease, leaving the built system flexible and adaptive (Ferber, 1999) (Kendall, 2000). Agent-based systems are not an exception to the rule. In spite of many desirable features, they still have shortcomings, like the inability of global optimization and an unpredictable

system performance. A potential problem for large systems is deterioration of performance from excessive required communications between agents. Component-based software technology became widely popular because of ease of development and integration. Component-based software architecture provides reconfigurable software coding for RMS controls, and is able to adapt to physical changes in manufacturing.

## 1.7 Objectives and Problem Statement

It is imperative to seek designs of reconfigurable controls to build true RMS lines producing families of products with many variants of engines. The possibility of reusing the same coding with minimal interventions makes this as important as the development of the manufacturing system hardware. The literature is unfortunately limited in applying reconfigurable control to automotive manufacturing. Very few publications are found on engine assembly manufacturing, because of the proprietary nature of such undertakings by each manufacturer.

This research aims at studying and developing a modular control architecture using newly emerged information technologies - such as component-based software technology and IEC 61311-3 Standard - towards building function blocks for controls. The proposed software architecture offers all RMS characteristics of modularity, flexibility, and robustness. Features used in coding the control methodologies used in this thesis include:

- Reconfigurability: Coding suited for RMS designed hardware, which considers modularity and ease of adaption to new manufacturing scenarios.;
- Reusability: Software collecting a library of components to reuse across entire assembly lines;
- Robustness: Programming which maintains system operability to counter malfunctions;
- Immunity to Disturbances: Controls handling machine malfunctions/errors yet retaining production, and
- Plug and Play: Software components (function blocks) which are easy to assemble, to build control systems for Reconfigurable ManufacturingS systems.

Benefits of the proposed architecture are demonstrated on an actual case study from the industry, in chapter 5.

# CHAPTER 2

## Literature Survey

Controls' architecture design should mimic the hardware design of a system. Evolutions in manufacturing paradigms and innovations of the microprocessor and Information Technology (IT) further transformed controls in manufacturing. The latter introduced hardware, software solutions and advanced algorithms for machines and systems while the former dictated controls' architecture. This chapter presents literature related to controls architecture and software.

## 2.1 Traditional Control Architectures

(Diltis et al. (1991) traced the evolution of controls' structures for Automated Manufacturing Systems (AMS). Three main controls' architectures were presented: 1- Centralized, 2- Hierarchical, and 3- Heterarchical. The authors also reviewed characteristics, advantages, and drawbacks of each of the topologies.

### 2.1.1 Centralized Architecture

This appears to be widely used in continuous process controls, to concentrate planed and processed information in a single decision node. The architecture requires powerful processing to handle large amounts of recourses. Advantages include simplicity of central startup, shutdown, and program archiving. Coding is extensive and difficult to develop and maintain, making codes unsuitable for Reconfigurable Manufacturing Systems (RMS). Figure 2.1 presents an example of centralized architecture.

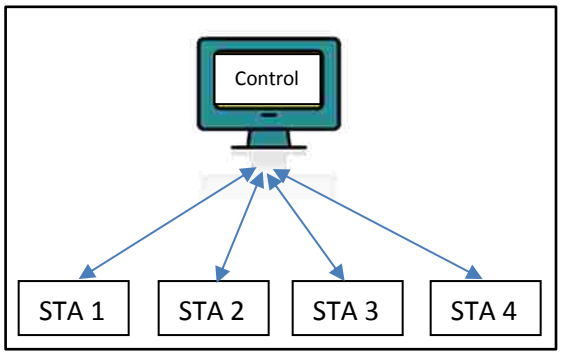



Figure 2.1: Centralized Architecture System Sample

### 2.1.2 Hierarchical Architecture

The International Standard Organization (ISO) and National Bureau of Standards (NBS) establish hierarchical controls models (Bauer *et al.*, 1994). Both are similar from top to bottom, but differ in the number of stages. The former breaks the organization into six levels: 1- Enterprise, 2- Plant, 3- Area, 4- Cell, 5- Station, and 6- Equipment, while the latter into five levels: 1- Facility, 2- Shop, 3- Cell, 4- Station, and 5- Equipment. This architecture is recognized for its efficiency and robustness, due to an easy structure. Figure 2.2 demonstrates an example of hierarchical architecture.

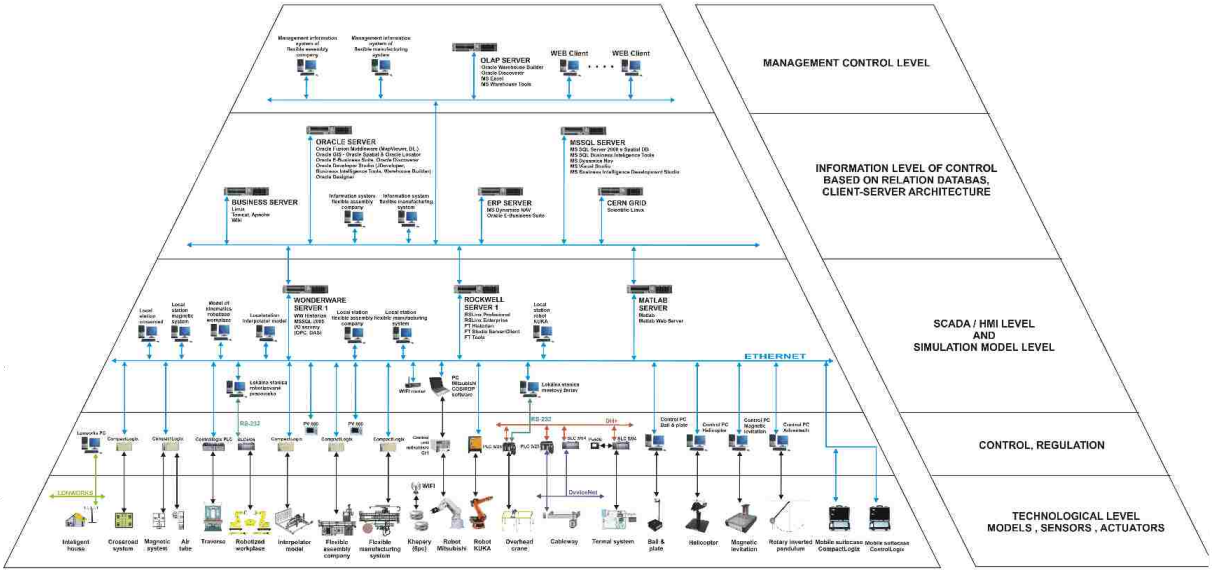



Figure 2.2: Hierarchical Controls System Sample *(http://ab.rockwellautomation.com)*

### 2.1.3 Heterarchical Architecture

Heterarchical architecture consists of a distributed control system, grouped independently but cooperating as "agents". Tasks are performed by exchanging information among agents. Duffie and Prabhu (1996) presented major works and design principals relating to heterarchical controls to promote extensibility, self-configuration, and adaptation, to real-time events such as in equipment failures. Figure2.3 shows an example of heterarchical architecture.

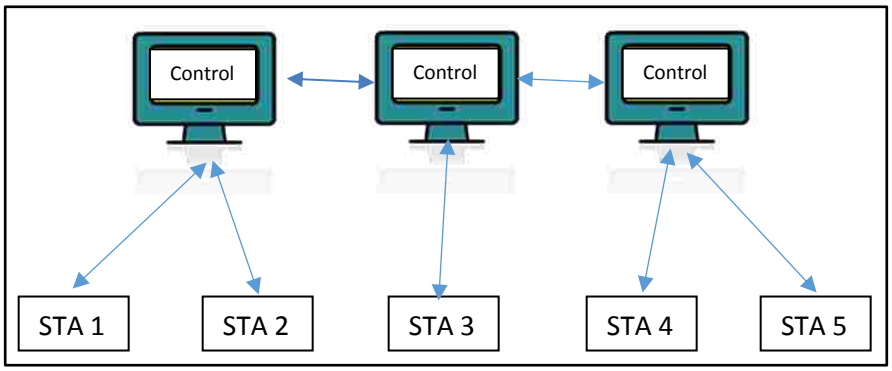



Figure 2.3: Heterarchical Controls System Sample

## 2.2 Holonic Manufacturing Systems (HMS)

Koestler developed the Holon concept, inspired by social organizations and living organisms. He was inspired by the hybrid form of modules or components in actual these systems. Koestler (1969) defined Holons as self-contained wholes of subordinated parts, related inversely. The HMS consortium arranged Koestler's proposal into a series of concepts for manufacturing systems. The objective was to enable production outcomes according to Holonic organisms and societies in life that have similar characteristics in terms of stability in adversity, and adaptability and flexibility to fluctuations (Van Brussel *et al.*, 1998). Valckenaers *et al.* (1994) define HMS as:

- Holon: An autonomous and cooperative block for manufacturing, such as in transformation, transportation, storage, data validation, and actual objects like date and processing components - A Holon can be part of another Holon as well.
- Holarchy: A series of Holons readily cooperating for a target or purpose, setting fundamental rules to associate Holons and establishing freedoms.

Farid (2004) surveyed HMS literature related to architectures, methodologies, protocols, algorithms, and interactions. He also highlighted open research challenges and roadblocks to industrial adoption. Babiceanu *et al.* (2006) reviewed the public domain as well, and included development and applications of Holonic systems.

The HMS concept combines best features of hierarchical and heterarchical organizations (Diltis *et al.*, 1991), in academic circles. However applications in industry are scarce, with a single study by Bussmann and Sieverding (2001). The latter applied the Holonic paradigm for Mercedes-Benz V6 and V8 engines. The study consists of: 1- Introduction of flexible buffers to decouple production line, 2- Addition of concurrent multi-functional stations for machine backup to overcome potential breakdown or production scalability, and 3- Development of Holonic controls for assembly, as in a Holon per Docking Station (DS Holon), per Manufacturing station (MF Holon), per Engine Block buffer (EB Holon), and Automatic-guided Vehicle (AGV Holon). Figure 2.4 shows the resulting layout. The new assembly profile and control system contrasted an existing assembly system based on simulation and plant data. Results showed the new layout to deliver a more robust and scalable system.

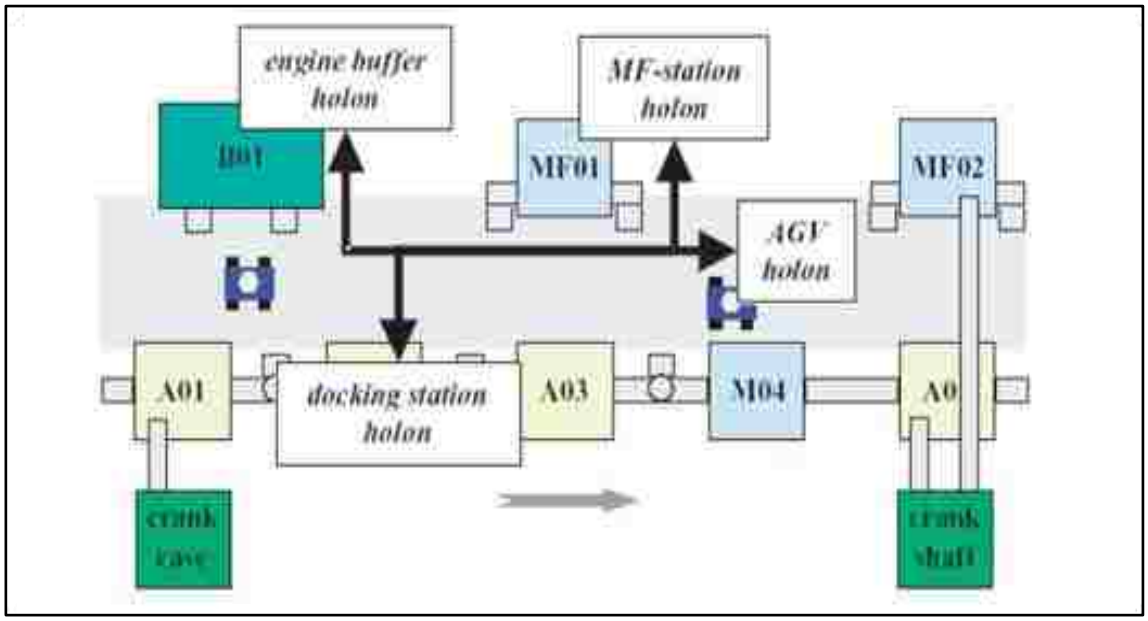



Figure 2.4: Holonic System Layout *(Bussmann and Sieverding, 2001)*

## 2.3 RMS Controls System State-of-the-art

Mehrabi *et al.* (2000) presented two main characteristics for RMS withmost influence on systems' software architecture:

a- Modularity: To guide design components of all systems, codes, and supporting platforms, as modular, and

b- Integrability: To design with readiness for new technology introduction into components and systems.

Software design modularity and integrability co-exist. The former enables scalability and simplicity in software design. The latter facilitates plug-and-play concepts. Open-architecture controllers proved essential for reconfiguration in manufacturing (Proctor, 1998) (Koren, 1999).

### 2.3.1 Open Control Architecture Systems

The IEEE Guide POSIX Open System Environment (OSE) defines "an open system" in section 2.2.2.28 as one that imbeds general specifications and standards related to interfaces, as well as services and supporting formats, to rightfully create application software in terms of:

- Portability with minimal changes across supplier systems;
- Interoperability with applications to in-house and third-party systems as well, and
- Interaction with operators facilitating portability.

Pritschow *et al.* (2001) indicate that software fulfilling IEEE's requirement should satisfy: 1- Vendor neutrality, 2- Consensus drivability (interest group), 3- Standards-basis (National/international), and 4- Free availability.

Since 1990, several research projects (OSACA/HUMNOS, OMAC, OSEC, OCEAN, ORCOS, JOP) have targeted Open Architecture Control or OAC for machines (Brecher *et al.*, 2010) (Pritschow *et al.*, 2001) (Katz *et al.*, 2000) (West, 2003). The Unified Reconfigurable Open Controls Architecture (UROCA) borrowed a concept from human's left/right brain intelligence. ElBeheiry and ElMaraghy (2006) used a Design Approach

for Real-Time Systems (DARTS) and Real-Time Specification Methodology (RTSM) to propose UROCA. One relevant project, the Open Modular Architecture Controls at GM Powertrain (OMAC), was adopted by all major automakers. Publication Motors (1996) describes OMAC's concept architecture and standard Application Programming Interface (IPA) as externally linked to various OMAC building blocks. A generic PC running Windows operating system is recommended for hardware platform; Profibus DP and Interbus are selected for networking and specify flowchart/IEC 61131-3 compatible languages for programming.

## 2.4 Control Software Development for RMS

Design and implementation research for RMS control software is common in literature. All modelling methodologies and programing languages are implemented as controls to reduce the gap with IT.

**2.4.1 Object Oriented Programming (OOP):** OPP is a software technology established to design production controls, and is popular in academia (Grabot and Huguet, 1996) (Howard *et al.,* 1998). Ka *et al.* (1998) highlighted a simulation framework based on objects to develop and evaluate multi-agent manufacturing architectures. Chan *et al.* (2000) established a similar architecture to design and implement reconfigurable controls. Holons, in Holonic manufacturing systems, are often programmed as objects using OOP.

**2.4.2 Unified Modeling Language (UML):** UML is an open smulation language used to create abstract models for systems. Advantages of UML diagrams include simplicity and standardization. Huang *et al.* (2001) designed modular real-time control system architectures based on UML. Panjaitan and Frey (2007) combined UML and IEC 61499 in a distributed control system.

**2.4.3 Petrinets (PNs):** A PN is a modeling script in distributed control system and process analysis. Petrinets proved important in modeling, analysis, and simulation and control of industrial automated systems (Peng and Zhou, 2004). Park *et al.* (1998) used Petrinets to combine modular logic controls with Sequential Function Charts (SFC) implementation logic. Holloway *et al.*, (2000)

allowed Petrinets in PC software to control systems. Lee and Hsu (2000) designed logics with Petrinets, as Ladder diagrams for industrial PLCs

**2.4.5 Agent-based methodology:** Some developers consider agents as objects; others differentiate between agents and objects even if commonalities are shared. Both approaches however envision using objects and agents together in developing software systems (Odel, 2002). Cândido *et al.* (2007) described a multi-agent implementation to manufacturing floor controls, with plug-and-play and system reconfiguration. Shop floor components were identified for improved adaptability and interaction to environmental requests. Monostori *et al.* (2006) introduced agents and multi-agent systems in coding for manufacturing applications. Their comprehensive survey emphasised methodological issues and agent deployment in industrial systems. Agent technologies and manufacturing evolutions were to proceed together. Vrba *et al.* (2011) presented methodologies to design agent-based control systems, related tools supporting implementation and validation, and agent applications for industrial systems. Metzger and Polakow (2011) surveyed technical applications in automating continuous industrial processes. Analysis of the literature followed main trends in research, such as agent-based supervisory controls shifting interests to low-level agent-based control algorithms.

## 2.5 Component-based Software Technology (CBS)

Reuse and development of Component-based Software (CBS) improve productivity and software quality. Building distributed systems based on component software increased over the past two decades (Mei *et al.*, 2003). Chirn and Duncan (2000) implemented CBS in an automatic assembly cell with plug-and-play through the Internet. Morton *et al.* (2002) introduced a methodology to design and implement software components as building blocks. Brennan *et al.* (2002) described a scheme for dynamic and intelligent reconfiguration of distributed control systems for IEC61499 function blocks. Xia *et al.* (2004) called on IEC61499 standard to specify components and implement control system. Harisson *et al.* (2006) harmonized modularity in reconfigurable automation systems using functional analysis (dual space and bag-definition). Vyatkin (2013)

reviewed software engineering in industrial automation, standards and norms, as well as models and methods and strategies to develop industrial software. The survey was geared to academia not industry. Mahmood *et al.* (2007) presented an extensive literature survey where many facets of CBD were presented in the software development field, including risks, benefits, selection and identification methodologies, in addition to means and tools of development. The survey concluded that CBD still requires support and research to achieve a full potential and economically viability.

## 2.6 Industrial Control in Automotive Sector: State-of-the-art

Clearly, PLCs are widespread in the Automotive Industry. They were introduced when General Motors (GM) looked for a robust replacement of the relay logic. A first commercial PLC by Bedford Associates was the Modular Digital Controller (MODICON) (Segovia and Theorin, 2012). Both PLC hardware and programing have since evolved throughout the manufacturing industry. Initially PLCs were programmed using Ladder logic, consisting of a structure similar to the relay logic which simulated electrical wiring control circuits. Today, PLC programing follows the IEC 61131-3 Standard and five programming languages (Karl-Heinz and Tiegelkamp, 2010):

1. The Ladder Diagram (LD): A graphical language widespread in automotive plants throughout North America due to\ simplicity of use by maintenance staff for troubleshooting and adjustments;
2. The Function Block Diagram (FBD): A graphical language used to encapsulate pieces of logic for reuse where the logic itself can be written in Ladder or other languages;
3. Structured Text (ST): An advanced programming method resembling Pascal in PCs, used for implementing complex algorithms to control uncommon machinery;
4. Instruction List (IL): A language similar to machine language when programing microprocessors, that uses registers and basic logic instruction, and
5. Sequential Function Chart (SFC): More of a sequencing tool than a programming language, with graphical representation in Grafcet.

Programmers use more than one of the IEC 61133-3 languages to write a PLC program for a complex machine. Ljungkrantz *et al.* (2010) studied controls coding for PLC programming in two Swedish automotive companies, Volvo and Saab. They reported that: 1- PLC programs are mainly written in Ladder diagrams and SCF, frequently as reused function blocks, and 2- Although function blocks were adopted, their behaviors were lightly detailed. Di Giovanni *et al.* (2013) described a methodology for automatic generation of Ladder Logic Diagrams (LLDs). The five-step method relies on a unique model generated from the entire manufacturing process specifications. Lee *et al.* (2006) overviewed frameworks for automatic generation of PLC codes. The project was not implemented, in spite of cooperation with Ford Motor Company, mainly because of a need for manual intervention to adapt and integrate coding with existing programs. Ryssel *et al. (*2009) proposed a methodology that generates function blocks based on web technologies. The approach faced hurdles for adoption in the Automotive Industry, the major being a need for high programing skills - not readily the case among maintenance personnel in production plants. Breslin *et al.* (2010) introduced another methodology that promotes the use of Semantic Web Computing (SWC), and aside from similar shortfalls encountered by Ryssel *et al.*, the authors agree that the methodology needs to mature before any further expansion. Ljungkrantz *et al.* (2010) proposed a framework called Reusable Automation Components (RAC). The methodology offers verification tools for programing, applied only to basic examples so more development is required to satisfy industry requisites. Hirsch (2010) proposed the use of SysML technology which was implemented using IEC61499 Standard on a modular manufacturing cell, but such technology remains challenging for plant personnel to support and use. Many of the presented concepts induce a paradigm shift in software controls' development methods rooted in the industry thus the resistance to adoption.

## 2.7 Summary

Topics and subject areas relevant to controls architecture and software were reviewed in this chapter. The literature is abundant in academia, but does not directly satisfy industry requirements or allow building a controls system for a machine fulfilling promises of

Reconfigurable Manufacturing Systems control systems. Most existing publications use Object Oriented Programming to develop manufacturing control systems. The IEC 61499 Function block standard gained popularity in academia but remains unused in the Automotive Industry because of perceived steep learning curve. The following chapter presents a methodology to design reconfigurable control software for machines. Subsequent chapters include details and a case study.

# CHAPTER 3

## System Engineering Approach to Control Software Design

The primary objective of this research is to design a reconfigurable software control system for an engine assembly line. Because of the design complexity of such a manufacturing system, requiring teams of engineers to work concurrently to create a reliable production line following multi-phase processing, an effective and systematic methodology remains greatly needed.

### 3.1 System Engineering to Design Controls Software

Over the past few decades new paradigms were introduced, like Flexible manufacturing, Changeable manufacturing, Reconfigurable manufacturing, in response to manufacturing systems and controls system's continuous increases in complexities. Limitations in programming languages readily understood by shop floor personnel when creating control systems in the manufacturing industry remain current. Academia is more advanced regarding controls modeling and coding methodologies, but they are yet to demonstrate efficiency when implemented on bigger scales in industrial settings. Controls must be included as an integral part of systems engineering design. Traditionally complex software used systems engineering for design, testing, and implementation. Many systems engineering methodologies and approaches are widely used in software engineering literature, such as the Vee model, Waterfall or Linear model, the Spiral or Incremental model (Kossiakoff, 2011). Each approach has applications, benefits, and limitations. The Vee model was chosen in this research, due to simplicity of use and most importantly testing before coding which saves time as defects are identified at early stages. In addition, system and user requirements are clear and do not change during the life cycle of a project (Sage and Cuppan, 2001). Figure 3.1 presents the traditional Vee model as per the Institute of Electrical and Electronics Engineers (IEEE) for software engineering. Some steps shown on the graph do not apply to developing the model in this thesis, because of specificity of controls logic software. The Vee model consists of two main steps: 1) A top to bottom design, starting with the definition of software requirements and design parameters, then 2) A bottom-up design, namely software modules compilation and testing (Ould, 1990).

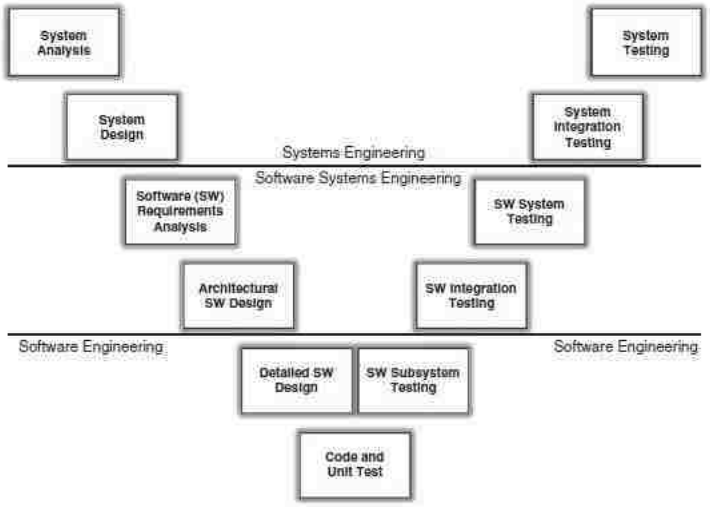



Figure 3.1: IEEE Vee Model for Software Systems Engineering Process *(Kossiakoff, 2011)*

## 3.2 Design Steps for Engine Assembly

With product design reaching a specific millstone, manufacturing teams get involved in reviewing product design for manufacturability (DFM) prior to beginning the design of manufacturing system. Subsequently, many teams join efforts and work concurrently to design an engine assembly line. The main steps of such an undertaking are:

### 3.2.1 Design of Assembly Process

Assembly sequence and precedence graph are first defined (Henrioud *et al.*, 2003). Product differentiation is always delayed in processing, for a family of engines (AlGeddawy and ElMraraghy, 2010) Tasks are identified and assigned to multiple stations, requiring manual operators, or semi-automatic to fully automatic operations, depending on the complexity of operations, ergonomics, and safety factors. Process designers tend to allocate repetitive and most demanding operations to automatic stations, to prevent operators' injuries. Quality assurance test stations are also identified and strategically located throughout an assembly process. Every engine manufacturer uses its specific internal processes; for example Ford Motor Company uses Ford's Production System (FPS), and Toyota uses Toyota's Production System (TPS) - both encompass all production philosophy and influence the design of assembly process, and assembly line in general.

### 3.2.2 Design of Mechanical Machinery and Tooling

Machine designers start conceptualizing an associated assembly system upon completing the process design. They begin with an assembly line layout, conveyor system, and pallets to use in producing an engine. A conveyor system includes "pallet rotates" and "pallet transfers". Some engine assembly systems are equipped with Automatic Guided Vehicles (AGVs) for part deliveries and pallet transportations to add parts routing flexibility to the line. Mechanical designers use the same base design and footprint for all manual stations and tooling. Automatic stations are more complex to design and depending on the operation a designer has few options. Robotic cells using industrial robot arms are most common stations for flexibility, but dedicated machines offer specialized tasks such as dowel or cup plug presses, more repeatable and reliable when built on rigid structures. Part and pallet transfers for long distances require the use of gantries. Test stations are semi-automatic with operator interventions to connect an engine to test probes using electrical wires or air hoses, but at times they can be fully automatic. Buffers and Automatic Storage and Retrieval Systems (ASRS) are also part of an engine assembly system. Finally, mechanical parts, like tools, are designed by tooling engineers for the engine assembly line. These range from a simple hand tool an operator uses for a determined task, to a complex end effector of a robot arm.

### 3.2.3 Design of Controls Hardware System

Controls engineering teams work with machine designers to comprehend intended functionality and determine sensors for controls system, such as limit switches, speed sensors, proximity switches, temperature sensors, etc. Other items to design include actuators, like valves for pneumatic cylinders, servo motors, contactors, and relays. Such electrical components are controlled by PLCs. The mechanical assembly system dictates a controls' architecture. A distributed controls system best fits a RMS system. All stations have the same functionality of processing a part from a controls system's perspective where each station is fitted with a Process Logic Controller (PLC) that orchestrates physical devises (actuators) and is responsible for the station's functionality. A communication routine is also chosen for networking the PLCs. Each PLC manufacturer has proprietary protocols. Profinet is a main protocol dominantly used for deterministic characteristics. A Radio Frequency Identification (RFID) is used to track

an engine from station to station. Each pallet has a tag storing all production data for the engine carried over, and each station is also equipped with a reader/writer to scan tag data, determine required tasks, and update tags with status of operations (failed or successful). Tags hold serial numbers identifying engines as well.

### 3.2.4 Design of Controls Software System

Controls run from PLCs that scan all field sensors (inputs) for statuses, and execute logic program, and updates outputs to actuate all field actuators. All station PLCs are networked in an engine assembly system. Figure 3.2 highlights a grouping of PLCs communicating through Ethernet switches to exchange production data.

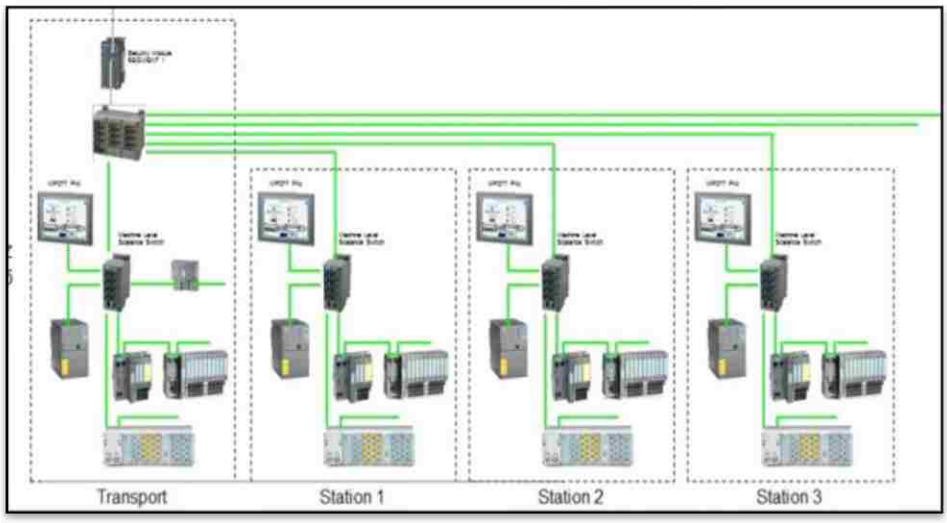



Figure 3.2: Network of PLCs in Engine Assembly

Designing controls begins with thorough reviews to understand process and functionality of each station. A station's functionality is decomposed into basic steps to follow while executing related tasks. The PLC logic mainly sequences steps to control statuses of machines, set before the commencement of coding (control logic writing). Programing language, machine interface, and machine interlock signals are chosen, and then a program structure is set prior to coding. Presently more than 90% of logic programmers in North America use Ladder logic (Bolton, 2009).

The process of engine assembly design is multi-phase. Figure 3.3 illustrates the Axiomatic Design (AD) model encompassing the four steps presented. It is worth mentioning that such steps are not completely sequential, as teams work simultaneously, yet interface constantly to accommodate changes and alleviate roadblocks potentially facing project's life cycle.

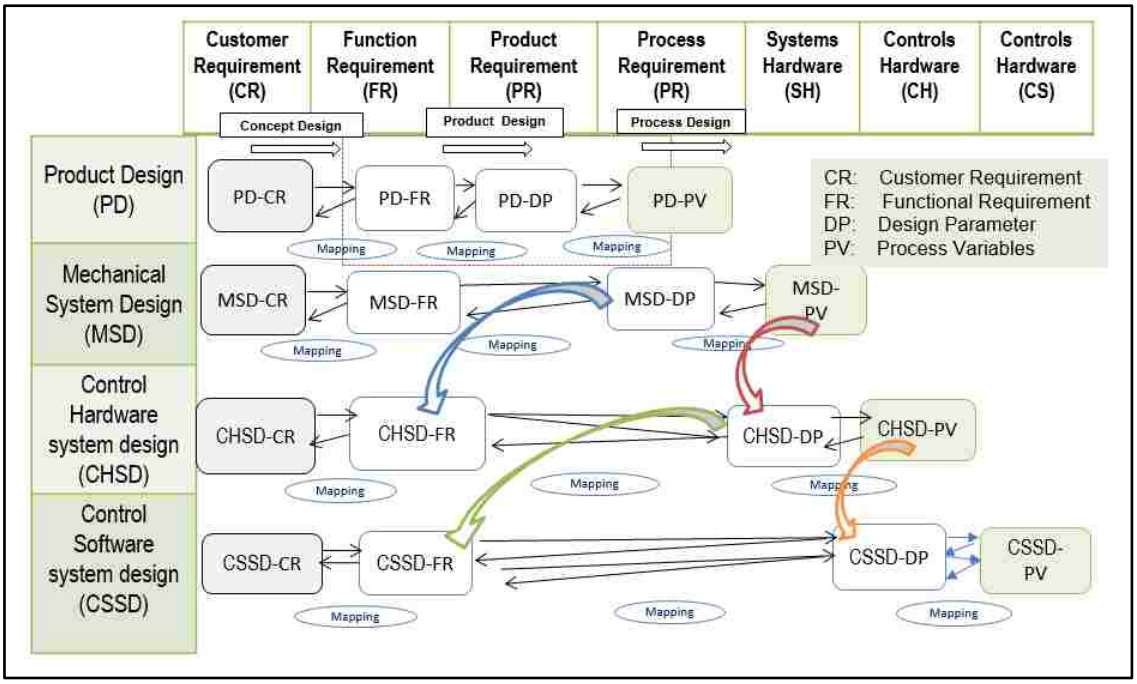



Figure 3.3: Design Framework Extended from Yien and Tsang (1996)

The last step of controls software system design is of primary importance in this thesis. Control Software System Design (CSSD-FR) denotes functionality of software modules and originates from Control Hardware System Design (CHSD-DP) or controls hardware modules in Figure 4.3. Any CSSD-DP reflects on the structure of controls system, the interaction of Input/output (I/O) and their statuses with program running. Process Variables PVs for CHSD-PV enable software design parameters and constrain CSSD-DP, while CSSD-PV represents controls software for program development.

### 3.3 Controls Software Design Process

This research focuses on the design of controls, using systems engineering and Axiomatic Design. The systematic design process is decomposed into a set of sequential activities:

1. Define user requirements for controls system;
2. Identify a controls hardware system;
3. Design a controls software system according to users' requirements (this is the most important step, to decompose in tasks);
4. Code and test Function blocks
5. Combine Function Blocks to form a machine program and test protocol;
6. Deploy machine programs and test the assembly line, and
7. Integrate and validate the system for the production line.

It is possible to shift back and forth between steps while designing a system, especially when testing fails. Experience confirms that changes to well-structured software are effortless, a matter of altering a few lines of code. Unfortunately experience also confirms the contrary, with unpredictable outcomes. A simple software change usually requires a complete retest of system (Blanchard *et al.*, 1990). Figure 3.4 shows design phases that form the present research's methodology.

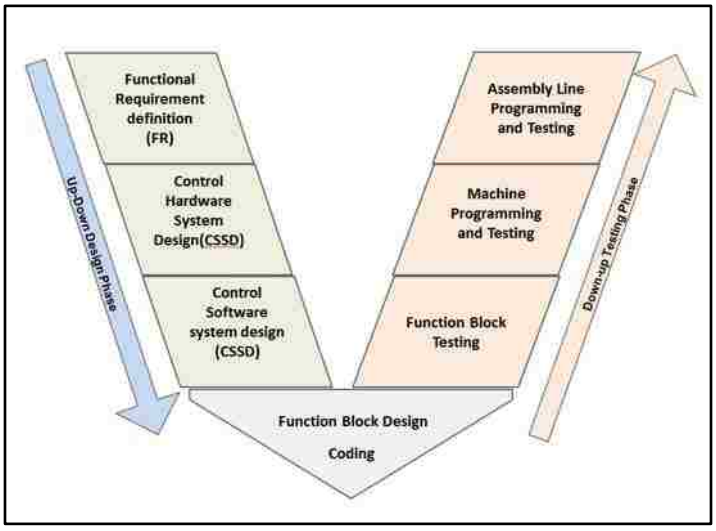



Figure 3.4: Vee Model Design of the Controls System

### 3.3.1 Software Requirements Definition

At this stage, customer requirements are documented, in terms of enumerating system's roles. The RMS controls system sought in this research must not only integrate the software, but the hardware system as well. By analogy to characteristics defined for an RMS system by Merhabi *et al.* (2000), five traits are proposed in reconfigurable control systems to enable integration to physical RMS. Such properties are to embed in the designed controls system for system re-configurability:

a) **Modularity:** System components should be modular and easy to build upon in a larger coherent system. The main enabler is the standard interface for control components. Modularity is achieved for controls software, using subroutines, databases, and Function Blocks that interface with one another.
b) **Integrability:** Integrating modules and components rapidly, with abilities for new technology assimilation, is quite important. Therefore a controls system should be designed with an open architecture.
c) **Diagnosibility:** The potential to identify anomalies quickly for corrections and resuming operations is crucial. Controls software should have fault messaging displays on Human Machine Interfaces (HMIs) besides faults' storage on servers for historical tracing.
d) **Convertibility:** Transforming functionality of existing systems to allow for quick product changes is key. A controls system should thus track each model to produce, by identifying each product with a Radio Frequency Identification (RFID) tag or a barcode to scan.
e) **Customization:** The ability to respond to changing production capacity and flexibility within a product family falls under customization. A controls system should orchestrate speeds of different motions within machines and transport systems such as conveyors to cope with volume demand and product variants.

Table 3.1: Hardware and Software Characteristic Requirements for RMS System

| RMS Characteristic Customer requirements (CR) | Hardware characteristic | Control software characteristic |
|---|---|---|
| Modularity | -Modular machines<br>-Modular system | -Use of subroutines<br>-Use of function blocks |
| Integrability | -Using ISO components same foot print, can be interchangeable | -Open architecture |
| Diagnosability | -Use sensors for feedback on all actuators | -HMI to display faults<br>-Server to store faults |
| Convertibility | -Flexible tooling,<br>-Changeable end effectors<br>-Flexible layout | -Use RFID technology for tools and end effector identification |
| Customization | -Ability to scale production volume<br>-Ability to run a family of product variants | -Use variable speed control on transport system<br>-Use servo slides for motions<br>-Use RFID technology to identify variants |

### 3.3.2 Controls Hardware Identification

By analogy to human anatomy, controls hardware represents the nervous system. A controls hardware system is usually dictated by mechanical design. Configurability, entailing modularity of design for system and individual machines, as a main characteristic, is to build from the ground up. Using reconfigurable machines definitely helps establish a reconfigurable system. Selected field sensors and actuators must assure good functionality of equipment and safe use of machinery. Safety of operators is crucial in manufacturing and controls industry, and is incorporated in many Government regulations such as OSHA in North America or the Machine Directive and ISO Standards in Europe. Safety requirements are well documented and guidelines should be followed by designers and programmers alike. In this phase, interlock signals, exchanged between

machines such as “machine is ready for part”, “machine cycle done”, etc., are defined and standardized. All engine assembly lines use RFID to track product evolution during production; stations are equipped with readers and pallets with tags. Figure 4.5 shows a typical machine with PLC, sensors, and interfaces.

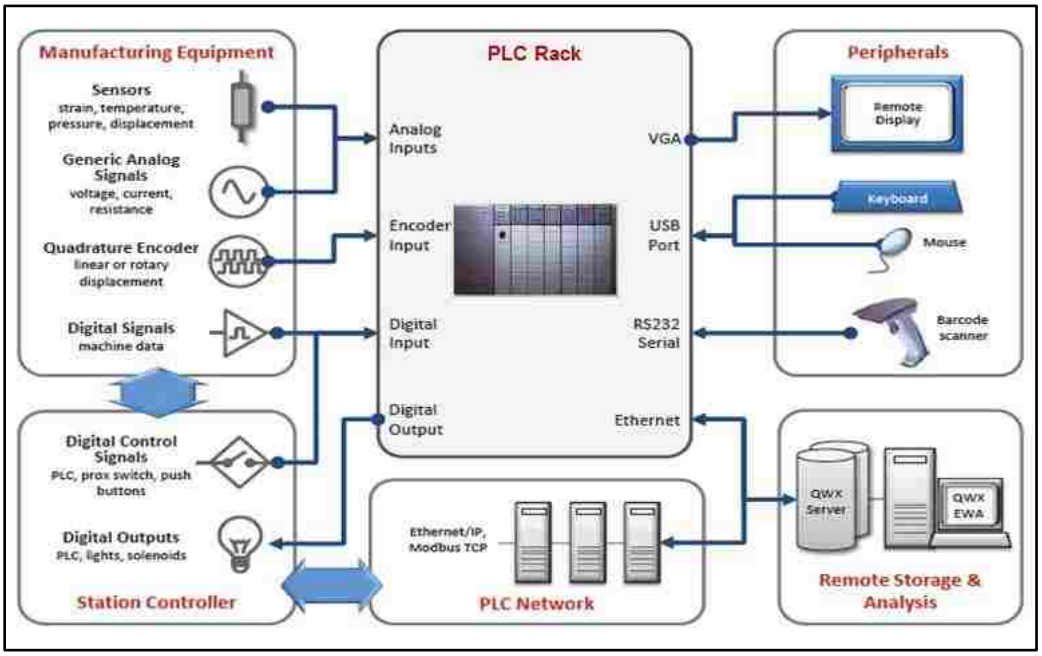



Figure 3.5: PLC Rack Connected to Field Sensors and IT

### 3.3.3 Control Software Design

An objective in controls development is the ease of rewriting or modifying software each time a Reconfigurable Manufacturing System changes, thus the need for a reconfigurable controls program. Major automotive manufacturers plan to run multiple engine architectures on engine production lines for two reasons: 1) To optimize the enormous investment required to build a new engine line, and 2) To cope with the short life cycle of internal combustion engines according to the increasingly stringent regulations for fuel consumption. An engine life cycle has over last decade been shortened from 10 years on average to only three years. Automotive manufacturers consequently retool lines every two to three years. It is challenging to retool a manufacturing facility such as an engine assembly line in full production, hence, the search for control system flexibility and reconfigurabily. The industry has a “2+1 specification” for new engine lines, which is

the capability to build two current products architectures and still able to run a future model. This is possible by embedding reconfiguration principles in both system software and hardware. A literature survey within this research field indicates attempts to develop required logics for RMS using different methodologies and languages. Nevertheless, none could successfully replace the mature PLC Ladder logic which electricians master. The methodology chosen for this research is inspired by component-based software technology, meeting requirements for re-configurability, while being based on Ladder logic for programming.

### 3.3.4 PLC Functions in Production Machine

PLCs dominate controls and automation as the brains to a production machine, since their invention. They have evolved from basic functions as replacements to the relay logic, to more sophisticated devices, with multiples functions, due to technological advances in IT and electronics and the constant demand for additional functionality. The major tasks that a PLC supports are:

a- **Operation sequencing**: Hardware design of PLCs and physical I/O cards allow for signal collections from controlled systems and generation of output signals applying to the system. Inputs are usually discrete signals from different components of machinery occurring in responding to PLC outputs or external factors. PLC programs produce desired outputs in right sequence for machines control.

b- **Safety devices monitoring**: Every production machine has devices ensuring safety of operators and preventing damage to components in case of unforeseen events or malfunctions. Emergency stops, light curtains, safety mats, etc. are constantly monitored.

c- **Error handling**: A PLC program should detect any malfunction in a machine component, and act according to a programmed response to the fault. The response could be an error message, a controlled shutdown of machine, or other preprogrammed reactions.

d- **HMI messages**: The most automated cells in production still require human intervention to reset a fault from malfunction or simply run a manual cycle for maintenance: HMI are used to enable such human-machine interactions. They display cell statuses and faults and other data of interest to production or maintenance personnel. The function of a PLC is to generate these messages and send them to the HMI; a PLC also takes inputs from HMI and processes them.

e- **Part tracking**: PLCs are gaining similar computing power and memory to those of PCs, due to advances in technology. Tasks are also becoming more complex as programmers are constantly requested to add new features to satisfy machine users. Part tracking is an example of task a PLC controls.

f- **Data transfer via Ethernet to a corporate network for reports**: PLCs are integral in the corporate network. They are monitored and programmed remotely from anywhere, with programs uploaded for archiving and data exchange for production supervision or otherwise, as deemed necessary.

Each of the above functions is programmed in PLCs, based on many function blocks for each associated task.

## 3.4 Component-based Software Technology

Component-based software is a mature concept in software engineering, as it aims at reusing proven coding to ensure savings and guaranty reliability. Databases of built software components render new program developments a matter of assembling standard components (Szyperski, 1998). Software components emerge as communication network boxes linked with connecting wires, similar to physically interconnected hardware components (Cox and Song, 2001).

### 3.4.1 Software Components Definition

A software component is an independent entity that executes a predefined task. It is a standard software element fitting a model with the ability of independent deployment without modifications (Councill, 1998). A software component is a unit with specified interfaces and content facilitating integration of third-party developments.

### 3.4.2 Software Component Characteristics

Coding for Software components should be:

- Standard, yet conform to class specifications like interface, meta-data, programming language, documentation, and means of deployment;
- Independent, for launch and composition without needing other specific components - Components need external services, at instances, in which case they should be declared and grouped separately;
- Composable, interacting with environments with predefined interfaces, yet allowing external components to access information and attributes;
- Deployable, to function alone as self-contained entities, and
- Documented, for users: Documentation should encompass interface and complete programming details including syntax and semantics.

Logic function blocks if used correctly, as described in IEC 61131-3, have all of the above characteristics, and could be geared at developing a component-based controls system.

## 3.5 Summary

In this chapter, benefits of using systems engineering in the design of controls systems for hardware and software are discussed. The Vee model was chosen in this thesis due to its simplicity. Component-based software was also introduced with advantages elucidated. A methodology of software control system is introduced in next chapter

# CHAPTER 4

## Control Software Development Methodology

A methodology to design and develop controls software is presented in this chapter. A case study "Manual work station"; is detailed, all steps are demonstrated sequentially.

### 4. Component-based Approach to Machine Controls Software

Distributed architecture accommodates best this proposed research: Each machine is controlled separately, to run in a standalone mode, and can also be considered as a module in building an engine assembly line. Machines are mechanically built as modular as possible, for

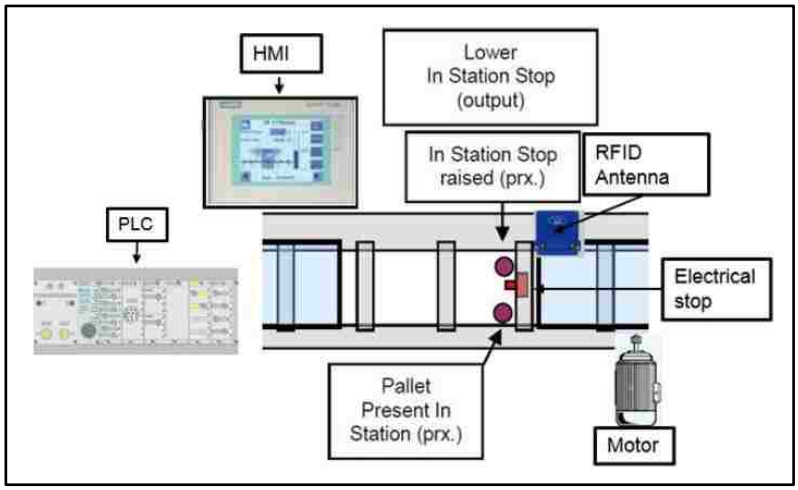



Figure 4.1: Modular Design for Manual Workstation

reconfigurability. Controls logic should be modular for each mechanical module. The granularity of a controls system depends on that of the mechanical design. Granularity means to what extent to decompose a mechatronic system. Granularity of reconfigurable controls software must be equal or lower than that of the mechanical design. A cladogram methodology is used herein to decompose controls software into required granularity. Figure 4.1 presents a modular design for a manual work station equipped with all controls hardware. A pragmatic approach should create a system with least granularity while providing all necessary system variants, like minimizing the number of

modules required within a given system and still remaining able to build any desired machine configuration (Harrison *et al.*, 2007). In practice, special attention should be made to the desired functionality for reuse, while maintaining modularity.

Mechanically, a manual station as that on Figure 4.6 decomposes into:

1. A conveyor body;
2. A pallet stop, and
3. An electric motor.

Controls wise the station can be decomposed into:

1. A PLC station controller (Controller/Brains);
2. An electrical motor control (Output 1);
3. A pallet stop control (Output 2);
4. A HMI for operator interface (Output3/Input 4);
5. A RFID antenna for pallet identification (Output4/Input 5);
6. A proximity switch to detect presence (Input 1);
7. A proximity switch to detect any stop raised state (Input 2);
8. A proximity switch to detect stop lowered state (Input 3);
9. Feedback to indicate motor running (Input4);
10. A feedback for motor faulted (Input5), and
11. A feedback for stop faulted (Input 6).

## 4.1 Decomposition of Controls System

Distributed controls architecture for hardware system is chosen for this research, to allow creation of modular controls system satisfying RMS needs. Moreover, PLC programming should follow components-based software technology as a prerequisite to satisfy RMS principals, hence, the need to decompose the controls system into optimum granularity. It is possible to create more complex products varieties by defining a finite number of basic production tasks. A production system can be seen as a set of controls components with devices in control of basic tasks, then combining the basic mechatronic components can result in complex assemblies. Basic mechatronic components can be reused to create different assembly systems with complex activities, merely by

reconfiguring simple components (Harrison *et al.*, 2007). A cladogram could be used to granulize controls and decompose it into basic components.

### 4.1.1 Decomposition of Controls System using Cladistics

Many methodologies are used in the literature to decompose systems. Systematically, DSM is frequently used in systems engineering for modularization. Systems are dealt with easily if decomposed into basic elements. Such approach is very effective for analysis, representation, and modeling (Browning, 2001). An original method was used by AlGeddawy and ElMaraghy (2013) for system decomposition and optimum granularity for modular product design. The methodology lends itself very well to the current research seeking the best granularity of built blocks (Logic Function Block) for designing complex systems. The manual station of Figure 4.1 is used as a case study to explain the Methodology which is divided in multiple sequential steps:

#### A STEP 1 – Identification of Control System Components

Table 4.2 contains all controls hardware components of a manual assembly system. Associated numbers or letters are used to ease the manipulations. The table shows the modules for the manual work station in Figure 4.1.

Table 4.1: BOM for Manual Work Station

| Number | Component Name | Symbol |
|---|---|---|
| 1 | An electrical motor control | A |
| 2 | A pallet stop control | B |
| 3 | A HMI for operator interface | C |
| 4 | An RFID antenna for pallet identification | D |
| 5 | Proximity switches to detect pallet presence | E1,E2 & E3 |
| 6 | A proximity switch to detect stop raised state | F1 and F2 |
| 7 | A proximity switch to detect stop lowered state | G |
| 8 | A Feedback to indicate motor running | H |
| 9 | A feedback for electrical motor faulted | I |
| 10 | A feedback for stop faulted | J |

## B STEP 2 – Building DSM to Capture Data Relationships

Interaction between components includes many types: Spatial interaction consists of components sharing the same or adjacent space to be linked mechanically; by exchanging energy such as electrical power; or by exchanging material or information, namely data and signals (Eppinger and Browning, 2012). A controls system is more of an information exchange as used in this case study: "1" is chosen to represent an element exchanging information while "0" is used for no information exchange.

The original DSM matrix is shown in Table 4.2. The matrix is 13x13 in size per system architecture and few inputs are shared by different modules or appear duplicated.

Table 4.2: Original DSM Matrix

| | A | B | C | D | E1 | F1 | E2 | F2 | E3 | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| B | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| C | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| D | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| E1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| F1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| F2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| G | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| H | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| I | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| J | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## C STEP 3– Cladogram Generation using Phylip Software

The Original DSM matrix is inputted to the Phylip cladistics software analysis tool to build a most parsimonious classification (Cladogram) for system components (http://www.phylip.com). The result is shown on Figure 4.2.

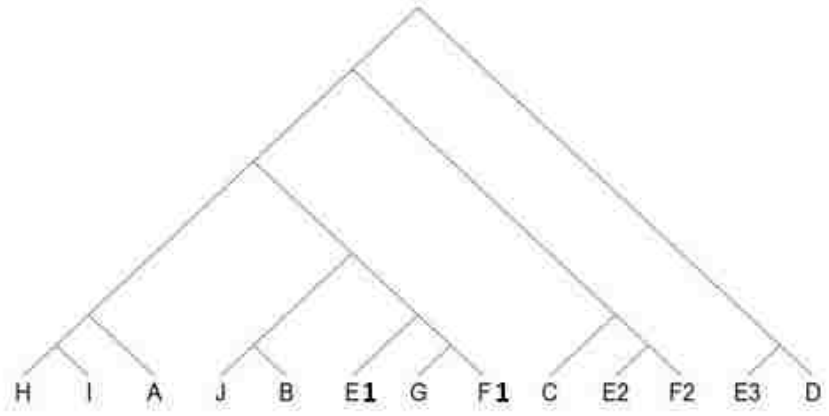



Figure 4.2: clustering results from the Phylip Software Analysis tool (http://evolution.genetics.washington.edu/phylip.html)

The cladogram of Figure 4.3 shows five levels of granularity.

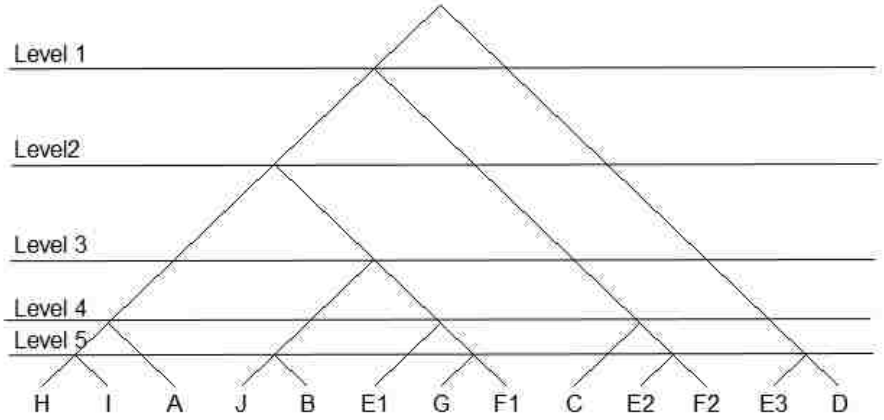



Figure 4.3: Manual Station Cladogram with Levels of Modularity

## D STEP 4 – DSM Matrix Rearrangement

The DSM matrix is rearranged according to the cladogram results, from left to right. Table 4.3 shows the rearranged DSM matrix.

Table 4.3: Rearranged DSM Matrix

| | H | I | A | J | B | E1 | G | F1 | C | E2 | F2 | E3 | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| F1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| E2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| F2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| E3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

## E STEP 5 – Calculation of Modularity Index

To determine the optimum granularity for a system, a modularity index, MI, is calculated for each level of granularity. The smallest MI corresponds to the best granularity.

$$\mathbf{MI = I + Z} \qquad (4.1)$$

I: Is the number of "1" elements in the DSM outside a given cluster, and

Z: is the Number of "0" elements inside the cluster.

Granularity level one results in two clusters shown on Table 4.4: MI1= 86+2 = 88.

Table 4.4: Granularity Level 1

| | H | I | A | J | B | E1 | G | F1 | C | E2 | F2 | E3 | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| F1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| E2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| F2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| E3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Granularity level two as presented on Table 4.5, indicates MI2 = 44+2 = 46.

Table 4.5: Granularity Level 2

| | H | I | A | J | B | E1 | G | F1 | C | E2 | F2 | E3 | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| F1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| E2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| F2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| E3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Granularity level three is shown on Table 4.6, and MI3 = 14+8 = 22.

Table 4.6: Granularity Level 3

| | H | I | A | J | B | E1 | G | F1 | C | E2 | F2 | E3 | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| F1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| E2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| F2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| E3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Granularity level four is shown on Table 4.7, leading to MI4 = 4+18 = 22.

Table 4.7: Granularity Level 4

| | H | I | A | J | B | E1 | G | F1 | C | E2 | F2 | E3 | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| F1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| E2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| F2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| E3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Granularity level five is presented on Table 4.8, and MI5 = 4+22 = 26.

Table 4.8: Granularity Level 5

| | H | I | A | J | B | E1 | G | F1 | C | E2 | F2 | E3 | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| F1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| E2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| F2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| E3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Levels 3 and 4 have similar modularity indexes with the only difference being one connection. Level 3 is adopted in this case as system decomposition to four elements. The extra component generated by Level 4 (J, B) does not have a specific functionality by itself. Components (H, I, A), (J, B, E, G, F), (C, E2, F2), and (E3, D) are basic components in the manual workstation: (Motor), (Stop), (RFID), and (HMI).The system encompasses four Function Blocks

- FB200: Motor Control Function Block;
- FB220: Stop Function Block;
- FB230: HMI Function Block, and
- FB240: RFID Function Block.

Function blocks are developed in Chapter 5 as applied to a case study. The decomposition methodology presented is straightforward, and can be applied to any control system, regardless of size.

## 4.2 Conceptual Design of PLC Logic using Axiomatic Design

Controls programs are designed and implemented based on programmers' experiences, similar to software design. Programs often begin with a program's draft copied from a similar existing machine which is then adapted, tested many times, and subjected to extensive debugging. At times, resources (time and money) are wasted unnecessarily, and projects' budgets end-up exceeding estimates. This results from the lack of use of fundamental principles and methodologies for software design, despite the availability of various methodologies (Suh et al., 1999). Axiomatic Design was first introduced as a general solution to design then adapt software design. It imposes a systematic thinking to satisfy customers' requirements. The methodology in itself is simple, but requires experience and practice to master.

### 4.2.1 Axiomatic Design Principles

The Axiomatic Design methodology is built on two "axioms":

1) Independence Axiom: This assures the independence of functional requirements, with best designs being decoupled, and
2) Information Axiom: This keeps information content to a minimum, where best designs have the highest probability of satisfying functional requirements.

Axiomatic Design divides relies on four domains, as shown on Figure 4.4.

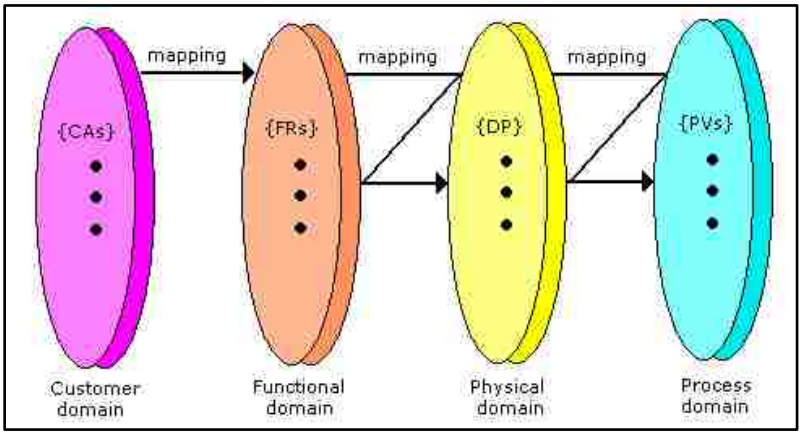



Figure 4.4: Axiomatic Design Domains *(http://www.axiomaticdesign.com)*

Mapping between four domains is referred to as "Zigzagging" and brings system's decomposition to a cellular level. Cladograms decompose a control system to the best granularity. Axiomatic Design further decomposes it to basic components:

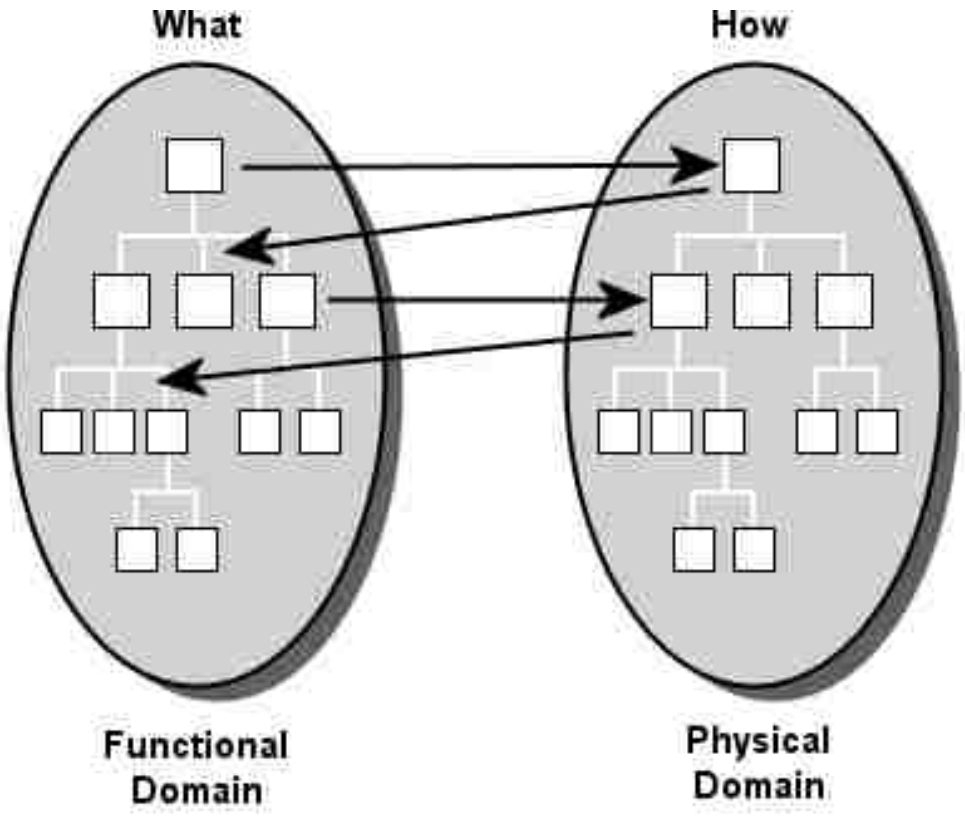



Figure 4.5: Zigzagging between Domains in Axiomatic Design *(http://www.axiomaticdesign.com)*

Axiomatic Design decomposes manual work station components. Figure 4.6 represents the matrix and Figure 4.7 shows zigzagging.

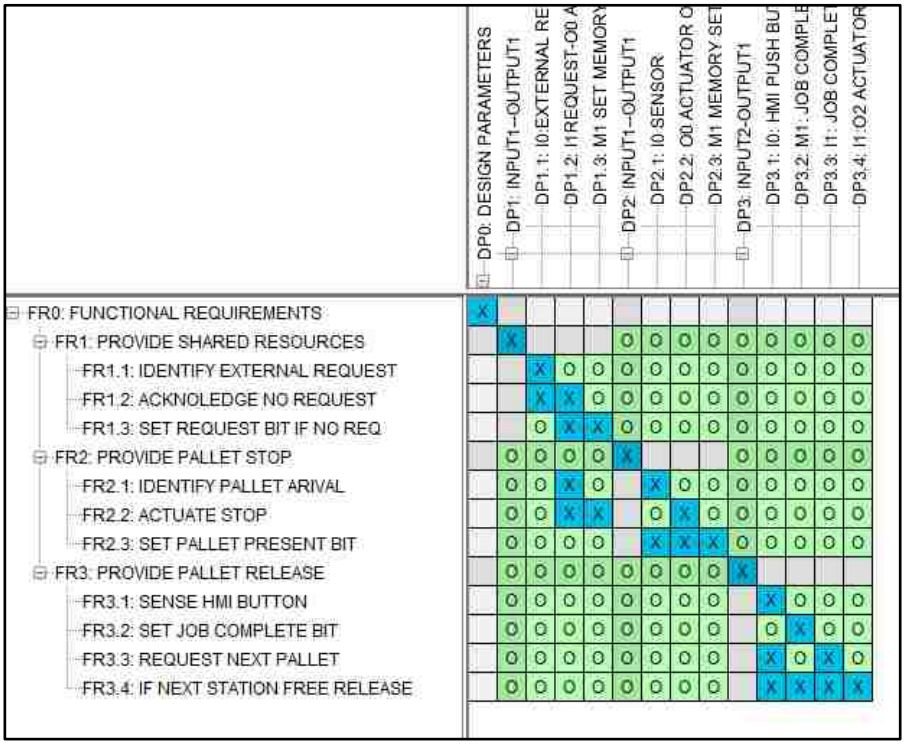



Figure 4.6: Axiomatic Design Matrix for Assembly Station Stop in Acclaro

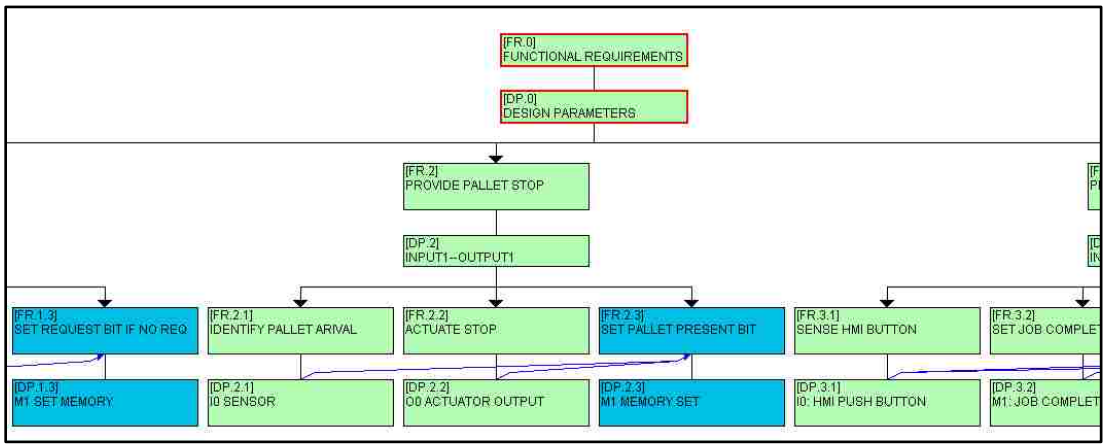



Figure 4.7: Tree FR-DP Zigzagging

## 4.3 Control Logic Programming Language

The literature review on controls within this research proved to be abundant in programming languages especially in academia, but unfortunately none of such languages are used in industry. The reasons for this were discussed previously, but most important being limitations of available languages in PLC platforms provided by major manufacturers. This consequently limited alternatives for controls language development in this research. All major PLC manufacturers follow IEC 61131-3 Standard for programming, established by the International Electrotechnical Commission (IEC). The intent is achieving program portability from one PLC brand to another, without changes, which is a goal yet to be attained. The standard defines five standard programming languages for PLC:

a- Ladder Logic (LL): This originated in the USA, and uses graphics in Relay Ladder Logic (RLL) - It is the most used language due to simplicity;

b- Sequential Function Charts (SFC): These were developed from the Grafcet theory, and break sequential tasks into Steps, Transitions, and Actions - Advantages include easiness of following a machine sequence, with the approach usually combining with other languages to produce complete programs;

c- Function Block Diagram (FBD): This is widely used in process industry to express the behavior of functions, like circuit diagrams in electronics - It sees a system as a flow of signals through processing elements;

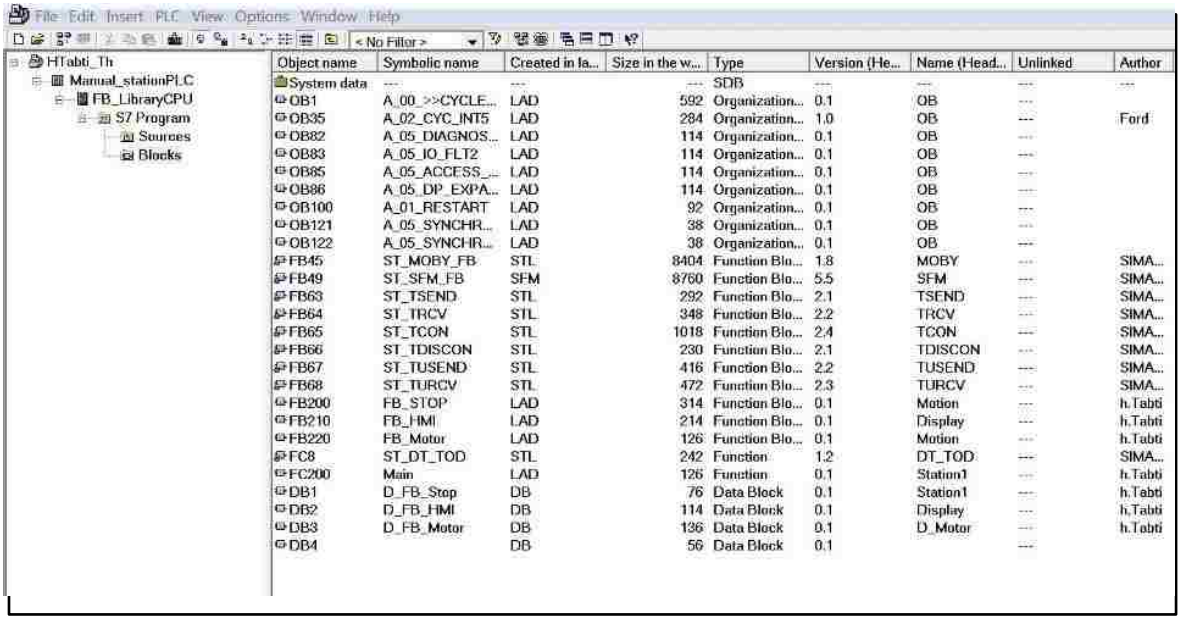
d- Instruction List (IL): This is a machine language that resembles an assembler programming for microprocessors and uses text for complex functions, and

e- Structured Text (ST): This can be seen as an advanced language, with modern essential elements for coding. It is very effective in defining complex function blocks for third-party usage.

These Programming languages divide into two distinct categories, graphical and textual, yet can be combined into the same program if needed (Hajarnavis and Young, 2008). Hajarnavis and Young evaluated the use of PLC programming languages by five teams with different levels of experience in implementing simple process changes. They concluded that Ladder logic was the easiest to use and still produced the best results. Hence, Ladder logic was adopted as a programming language in this research.

### 4.4. Function Block Development

A limitation of this research is programming language. The thesis is focused on industrial




| Object name | Symbolic name | Created in la... | Size in the w... | Type | Version (He... | Name (Head... | Unlinked | Author |
|---|---|---|---|---|---|---|---|---|
| System data | --- | --- | --- | SDB | --- | --- | --- | --- |
| OB1 | A_00_>>CYCLE... | LAD | 592 | Organization... | 0.1 | OB | --- | |
| OB35 | A_02_CYC_INT5 | LAD | 284 | Organization... | 1.0 | OB | --- | Ford |
| OB82 | A_05_DIAGNOS... | LAD | 114 | Organization... | 0.1 | OB | --- | |
| OB83 | A_05_IO_FLT2 | LAD | 114 | Organization... | 0.1 | OB | --- | |
| OB85 | A_05_ACCESS_... | LAD | 114 | Organization... | 0.1 | OB | --- | |
| OB86 | A_05_DP_EXPA... | LAD | 114 | Organization... | 0.1 | OB | --- | |
| OB100 | A_01_RESTART | LAD | 92 | Organization... | 0.1 | OB | --- | |
| OB121 | A_05_SYNCHR... | LAD | 38 | Organization... | 0.1 | OB | --- | |
| OB122 | A_05_SYNCHR... | LAD | 38 | Organization... | 0.1 | OB | --- | |
| FB45 | ST_MOBY_FB | STL | 8404 | Function Blo... | 1.8 | MOBY | --- | SIMA... |
| FB49 | ST_SFM_FB | SFM | 8760 | Function Blo... | 5.5 | SFM | --- | SIMA... |
| FB63 | ST_TSEND | STL | 292 | Function Blo... | 2.1 | TSEND | --- | SIMA... |
| FB64 | ST_TRCV | STL | 348 | Function Blo... | 2.2 | TRCV | --- | SIMA... |
| FB65 | ST_TCON | STL | 1018 | Function Blo... | 2.4 | TCON | --- | SIMA... |
| FB66 | ST_TDISCON | STL | 230 | Function Blo... | 2.1 | TDISCON | --- | SIMA... |
| FB67 | ST_TUSEND | STL | 416 | Function Blo... | 2.2 | TUSEND | --- | SIMA... |
| FB68 | ST_TURCV | STL | 472 | Function Blo... | 2.3 | TURCV | --- | SIMA... |
| FB200 | FB_STOP | LAD | 314 | Function Blo... | 0.1 | Motion | --- | h.Tabti |
| FB210 | FB_HMI | LAD | 214 | Function Blo... | 0.1 | Display | --- | h.Tabti |
| FB220 | FB_Motor | LAD | 126 | Function Blo... | 0.1 | Motion | --- | h.Tabti |
| FC8 | ST_DT_TOD | STL | 242 | Function | 1.2 | DT_TOD | --- | SIMA... |
| FC200 | Main | LAD | 126 | Function | 0.1 | Station1 | --- | h.Tabti |
| DB1 | D_FB_Stop | DB | 76 | Data Block | 0.1 | Station1 | --- | h.Tabti |
| DB2 | D_FB_HMI | DB | 114 | Data Block | 0.1 | Display | --- | h.Tabti |
| DB3 | D_FB_Motor | DB | 136 | Data Block | 0.1 | D_Motor | --- | h.Tabti |
| DB4 | | DB | 56 | Data Block | 0.1 | | --- | |

Figure 4.8: Manual Station Function Blocks

use, thus the choice of Ladder logic amongst five programming languages offered in IEC61311-3. Decomposition of manual work station resulted in four distinct objects: Motor, Stop, HMI, and RFID. Figure 4.8 lists the function blocks developed for the manual workstation.

The aim of this thesis is to create a library of Function Blocks for each possible mechatronic component on an engine assembly line. A survey of controls hardware was completed by a major car manufacturer in North America. Table 4.9 lists the mechatronic

components used. The plant adopts Siemens hardware. The logic for simple components is easy to write, then encapsulate as function blocks and store in a library for use when

needed. Architecture has to be defined for block communication. Block input and output signals must be identified and carried out through the project. Upon finishing building the project's library, writing logics becomes easier, consisting of assembling logic blocks in a sequence to form a machine program: Sequential Function Charts are adopted for this purpose. Figure 4.9 shows an example of program using SFCs for a manual work station; it calls and executes FBs sequentially.

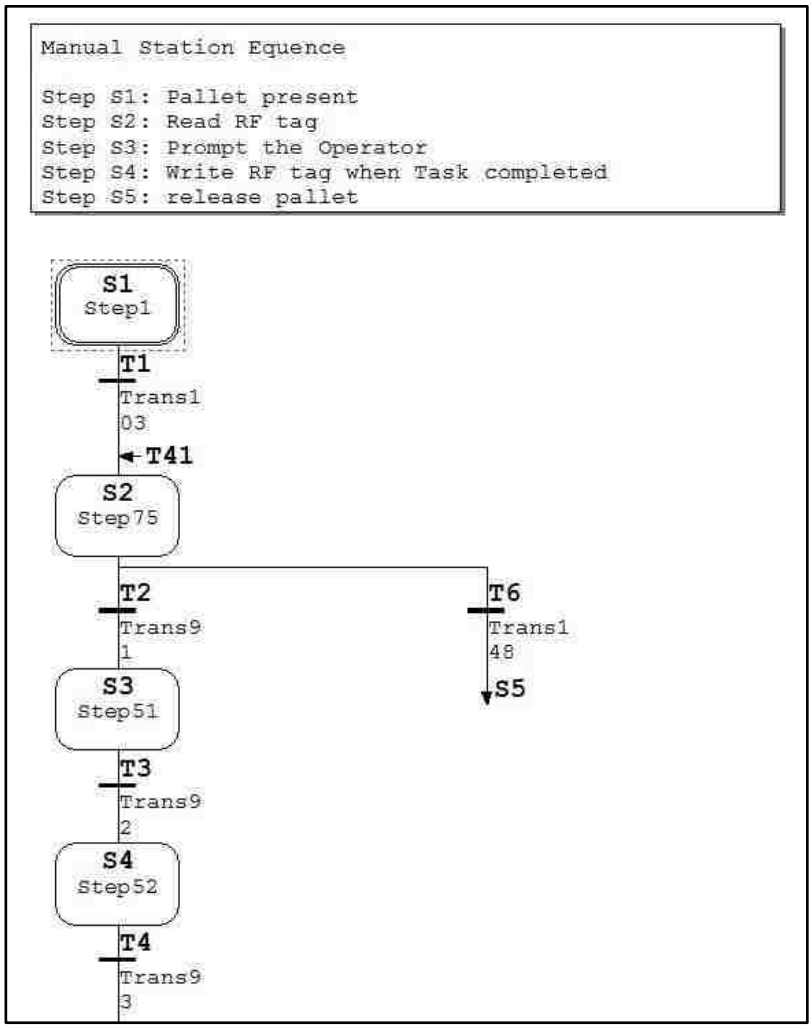



Figure 4.9: Sequential Flow Chart for a Manual Work Station

## 4.5 Function Block Testing

Each function block needs debugging and testing before implementation. Every PLC manufacturer has built-in debugging functionality in programmed logics. The program detects syntax errors and signals them during compilation. Some software include simulation to validate written programs. Siemens Step 7 programming software for instance has a module for simulation to run without updating all physical outputs - allowing a programmer to try a program without fear of damaging a machine in case of coding error. Indeed it is easier to debug simple Function Blocks before running them on actual machinery, but testing complicated FBs such as RFID blocks remains challenging and requires field trials because of complexity.

## 4.6 Machine Logic Testing

Function blocks are assembled in sequence to form a machine program after they are written. Simple machines can be easily debugged and tested, while complex machinery requires time and efforts to prove the functionality of programming. Many tools are available to test machine logic before trials in the field. Hardware in the loop consists of creating 3D simulation for a complete machine. All Computer-aided Design (CAD) files for standard equipment like robots are provided by manufacturers, for simple imports for simulations and integrations with other devices for complete systems. An Input/Output (I/O) map is then generated to link each component in the system to be prepared to run on a computer screen. An HMI is often added to validate manual functions of the machine, and screen messages. Such tool produces value to the development of software (Gu *et al.*, 2007) in terms of:

- Helping evaluate control strategies implemented very early and before a machine is physically built;
- Assisting failure testing without fear of damaging equipment;
- Allowing simulations of worst case scenarios without potentially harming operators or machinery;
- Helping investigate interaction between all devices, and

- Ensuring high quality levels.

## 4.7 Engine Assembly General Controls Hardware

Many mechatronic devices are used in an engine assembly. A survey by a major car manufacturer in North America helped focus this research on key components. Table 4.9 lists the related hardware. Each of these components requires at least one Function Block for controls. It is worth compiling them in a library to use when required.

Table 4.9: Survey on Mechatronic Components in an Assembly Line

| Controls Device | Maker |
|---|---|
| Industrial Robots | ABB |
| RFID | Siemens |
| Servomotors | Siemens |
| AC Motors | Eurodrive |
| Vision systems | Cognex |
| Press | Promess |
| Variable Frequency Drive | Siemens |
| Machine Human Interface | Siemens |
| Pneumatic valves | Festo |
| Pallet Stop | ABB |
| Audible alarm | Siemens |
| Barcode reader | EMS |
| Barcode Printer | Sato |
| Contactors | Siemens |
| Bolt Rundown | Atlas Copco |

Each machine is mainly composed of a few basic components listed in Table 4.9. The objective is to create a library of function blocks for each component, to store for use when needed.

## 4.8 Formalization of Controls Software Modularity Method

The methodology presented in this chapter involves systems engineering design for controls software. It can be recapitulated in:

- Step 1: Define the mechatronic devices used in an assembly machine;
- Step 2: Follow the DSM methodology presented in this research to decompose the mechatronic system into the best granularity;
- Step 3: Use Axiomatic Design to decompose the resulting DSM module into cellular level: I/O and holding memories;
- Step 4: Use Ladder logic (IEC61131-3) to create logics for each module and encapsulate the resulting code in a Function Block;
- Step 5: Debug the FBs using Siemens Step 7 software then store them in a library;
- Step 6: Use Sequential Function Charts programming language to build a machine controls program from the designed library;
- Step 7: Test machine control logics with hardware in the loop simulation then on the physical machine, and
- Step 8: Document each FB after validation and deploy the rest of machines in the assembly line.

## 4.9 Summary

This chapter presented a methodology for modular and reconfigurable control software design and implementation. A Vee model process was adopted from a systems engineering approach to develop a RMS controls system. A Design Structure Matrix for the system was built allowing decomposition of the mechatronic system into basic modules using cladistics. The control system modules were then further decomposed using Axiomatic Design. All design parameters were identified and used to write the controls Function Blocks (FB). Resulting FBs were stored in a library then retrieved as needed to build a control program for a manual work station. This way writing machine programs became a matter of using ready and tested FBs, thus achieving the goal of modular design that satisfies the reconfigurability requirements of a changeable engine assembly system. Chapter 5 is dedicated to implementing the developed methodology in a case study and presents related results.

# CHAPTER 5

## Manual Workstation Code Generation

The manual station presented in this chapter constitutes a case study #1 of building a complete program for the station. Function Blocks (FBs) are generated using Axiomatic Design (AD) before assembly to form a machine program. Case study #2 consists of modifying the machine hardware used by including a new component; a stop is chosen for this purpose. The ability to modify the controls system easily and allow the new functionality is demonstrated in this chapter.

## 5 Case Study 1: Manual Workstation Program Generation

### 5.1 Motor Control FB Development

The process of developing a motor control FB begins with the design of logic. Axiomatic Design is used to define the machine's Functional Requirements (FRs), Design Parameters (DPs), and Design Matrix (DM).Acclaro software is used to help generate a flowchart showing relationships between components. The process is demonstrated step by step for one Function Block, FB 200, or "Motor Control".

### 5.2 Design Matrix for Motor Control FB

The design matrix in AD builds relationships between FR and DP vectors. Customer Requirements (CRs) derive from FRs by asking 'What' and DPs by asking 'How' questions. Matrix D represents mapping between domains, and can be written as:

$$\{FR\} = [D]\{DP\} \qquad 5.1$$

{FR} represents the FR vector;
{DP} gathers the DP vector, and
[D] is the design matrix.
Acclaro Software V5.3 is used to represent design matrix and flow chart. First, FRs and DPs are generated through series of what and how queries while zigzagging between the two domains; Table 5.1 shows Motor Control FB decomposition:

Table 5.1: Motor Control FB Decomposition FR1- DP1

| | |
|---|---|
| FR1: Drive Conveyor | DP1: Run Conveyor Forward |
| FR2: Provide Motor Status | DP2: Display Status on HMI |

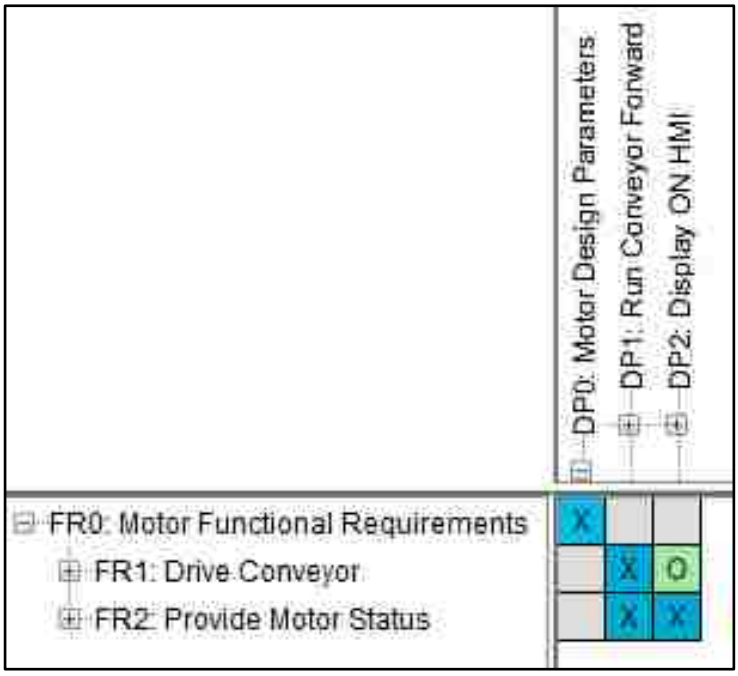



Figure 5.1: FR1 - DP1 Motor Design Matrix

FR1 decomposes into:

Table 5.2 Motor Control FB Decomposition FR1Xs- DP1Xs

| | |
|---|---|
| FR11: Start Motor | DP11: HMI Start Button |
| FR12: Stop Motor | DP12: HMI Stop Button |
| FR13: Jog Motor | DP13: HMI Jog Button |

Each FR follows the same process, as shown on Figure 5.2 shows the next level of decomposition FR1X DP1X .

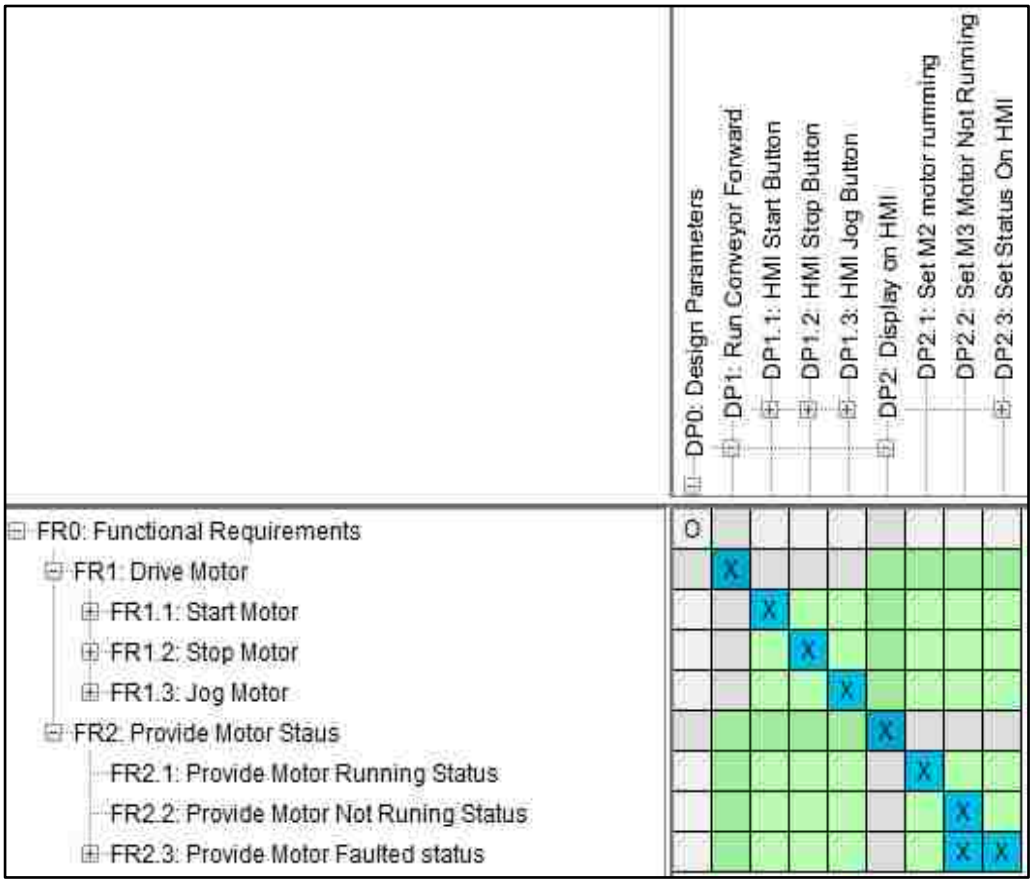



Figure 5.3: FR1X - DR1X Motor Design Matrix

Decomposition continues till all FRs and DPs are determined, resulting in a complete design matrix for Motor Control. The resulting DM is shown on Figure 5.4. It is clear that the system is uncoupled, which will result in a viable design. A flow diagram is generated using Acclaro software, to serve as a blueprint for the FB development. The flow diagram shows sequences and system decomposition as well as Inputs and Outputs and internal memory registry to hold data for faults and HMI. Figure A.1 and A.2 attached in Appendix A present more detail on such developments.

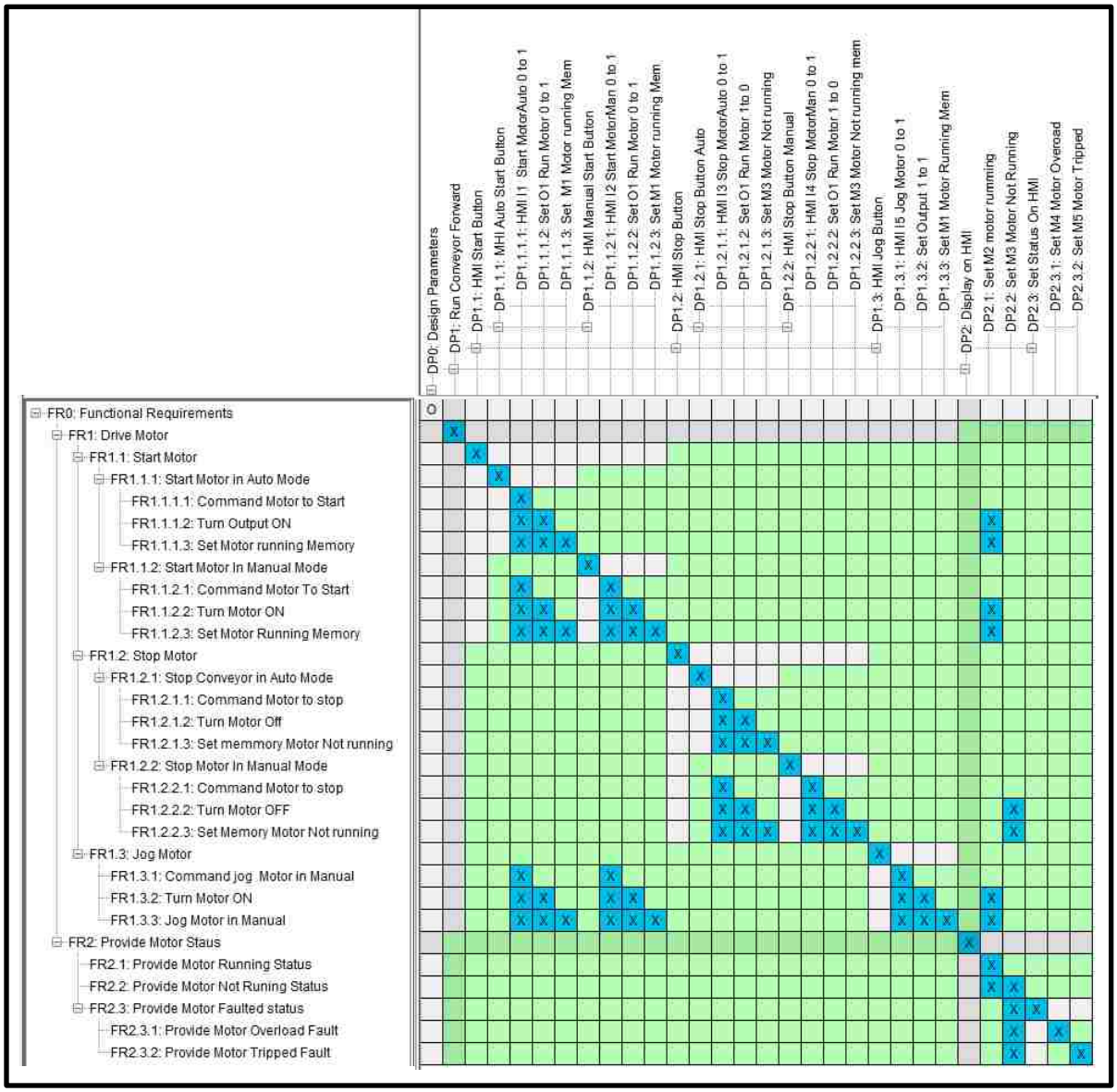



Figure 5.4: Complete Motor Design Matrix

## 5.3 Motor Control FB Development

The flow chart in Appendix A delimits all elements the Motor Control FB encompasses, I/Os, and internal holding memory registers for the program to develop. The flow of the program is also demonstrated. The diagram provides the programmer with a roadmap for FB coding. The FB program is included in Appendix B for reference. , FB 200 (Motor Control) is archived and stored for future use.

The same steps are duplicated to generate FB 220 (Stop Control FB), FB 230 (HMI Control), and FB 240 (RFID Control). These blocks are stored in a library, as shown on Figure 5.6.

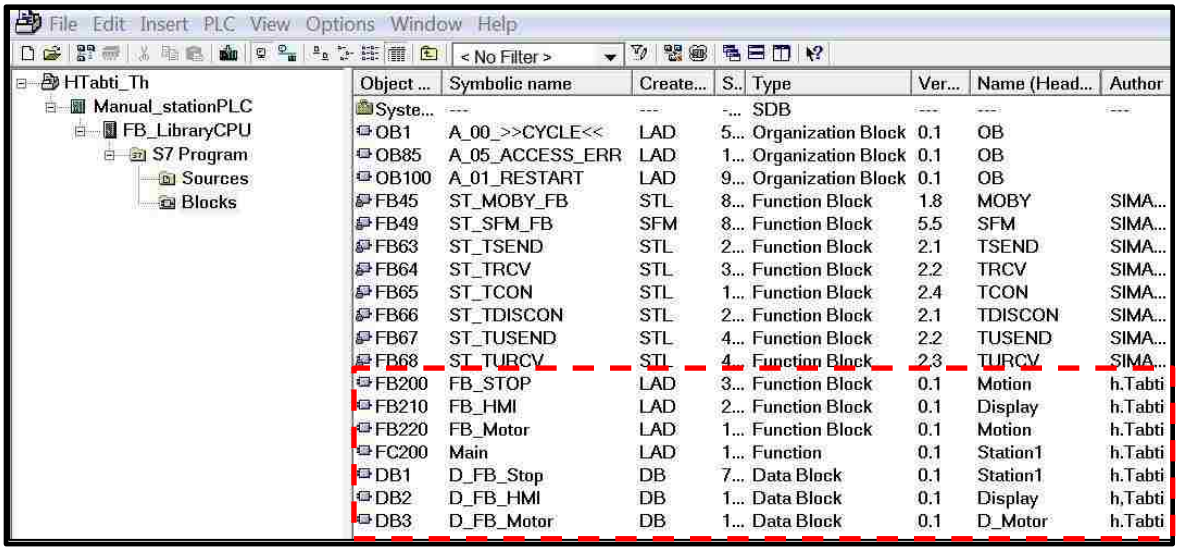



Figure 5.6: Function Block Library for Created Function Blocks

## 5.4 Manual Work Station Program Building

Creation of machine program is effortless after gathering all building blocks. Siemens S7 software is used to create a program that calls all developed FBs. The task relies on a FC (Function Call) block that reaches out to all FBs following a machine sequence. A manual work station series of operations is described hereafter.

A pallet enters a station, while a mechanical stop holds it in position with a proximity

switch detecting its presence; then a 'RFID read' is trigged to identify the required task. Once the work is completed, the FRID updates the status to the tag, and an operator releases the pallet by pressing a programed button on the HMI. The stop is actuated and the pallet is set for transfer. Once a pallet leaves the station, the cycle repeats. Figure 5.7 illustrates the related FC program.

In sequence:

- FB200: Motor Control Function Block runs the machine conveyor;
- FB220: Stop Function Block controls the stop;
- FB240: RFID Function Block reads all required tasks;
- Work In progress completes;
- FB240: RFID Function Block writes operations' status;
- FB230: HMI Function Block triggers display and releases the pallet, and
- FB220: Stop Function Block releases the pallet.

The complete program is attached in Appendix C.

## 5.5 Case Study 2: Addition of Stop to Work Station

### 5.5.1 Re-configurability Capability

Reconfigurability is the ability to add flexibility on demand. This case study demonstrates reconfigurability by adding a stop to the manual work station case study. One goal of this research is to highlight the claim of reconfigurability of methodology developed by including a new module. A stop is selected herein because of the availability of its FB. All FBs can be easily added and deleted when a machine is altered due to the program structure modularity.

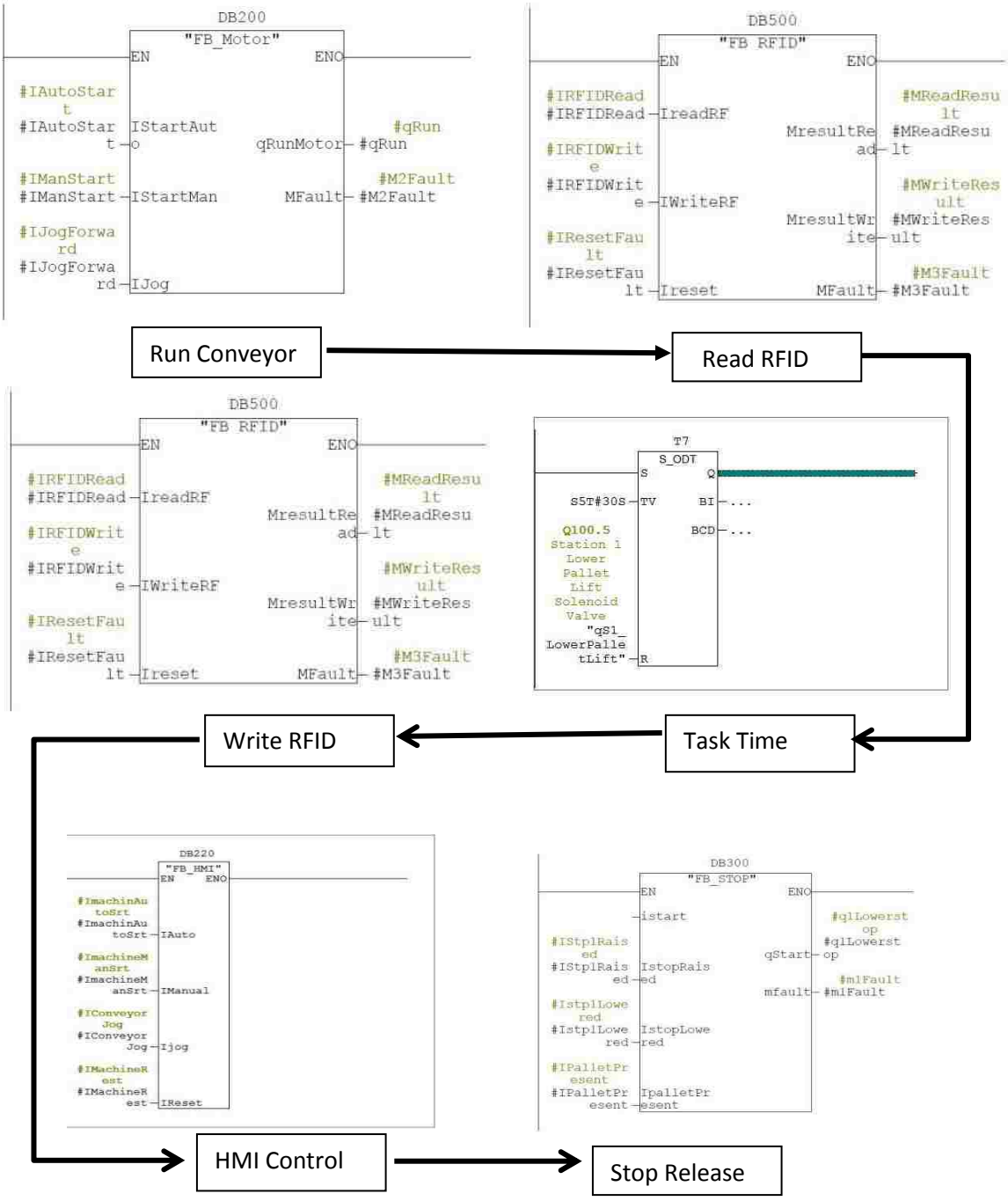



Figure 5.7: Manual Station FC Program Sequence

### 5.5.2 Addition of New Module to Machine

Controls software updates are required when adding hardware to a machine. Today in the Automotive Industry, any hardware change unfortunately necessitates extensive programming, due to programs' architecture and structure. Programs are written in Ladder Logic, and are specifically coded for prescribed tasks. It is certainly very challenging to rework a machine, by adding functionality, which entails addition of mechanical components that in turn drive changes in programming. Downtime is not permitted in manufacturing, especially in the Automotive Industry, because of associated prohibitive costs. The purchase of complete new machines is in fact preferred in manufacturing plants as they get delivered with prescribed software programs (already written).

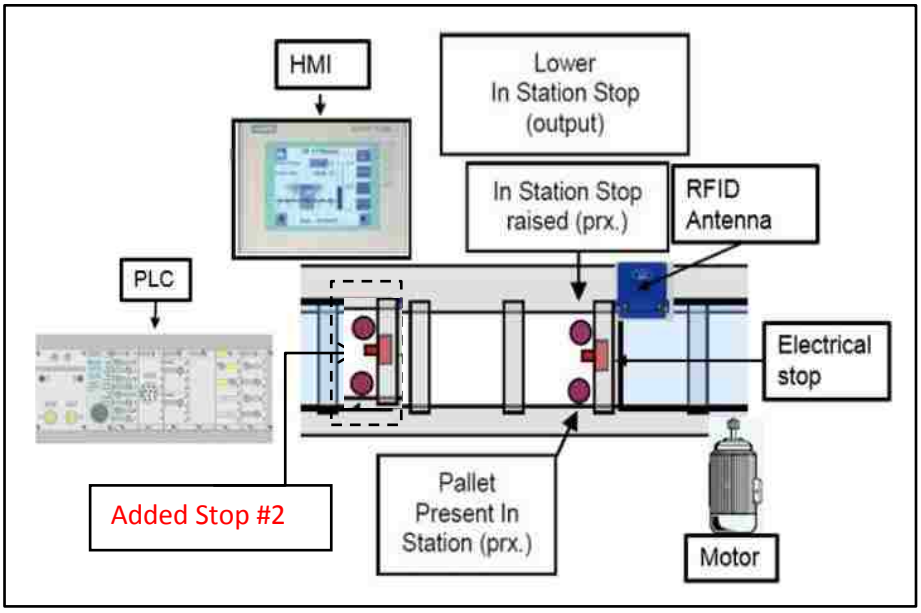


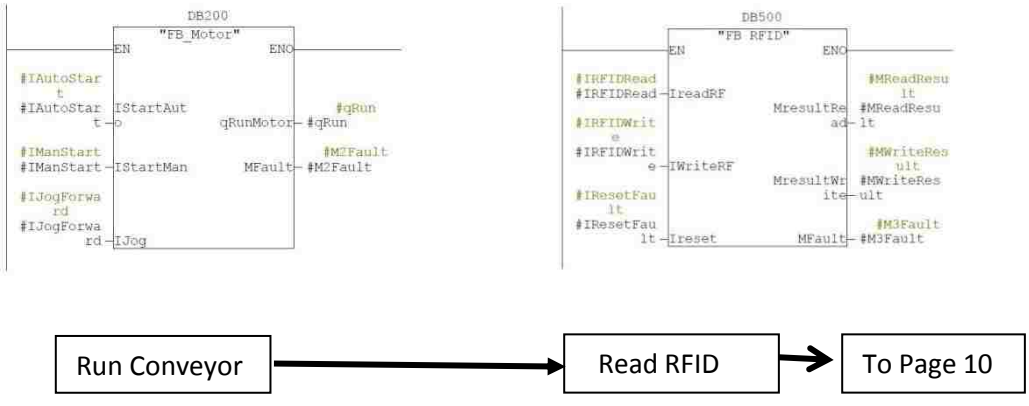

Figure 5.8: Reconfigured Manual Work Station

Introducing such new preprogrammed machines to production lines is better seen by production personnel instead of reworking or reprogramming existing production machinery existing on production floors. The work proposed in this research alleviates this problem: The addition of a stop for the manual station case study is shown to result in a station with two positions that can handle more tasks. Figure 5.5 represents the new machine layout.

An approach similar to that in 5.1.4 is to use, first, by defining the sequence of operations then designing a FC responsible for calling all FBs in sequence to accomplish the new predefined tasks. In this instance the RFID is assumed not to be required: No task is getting done at the station; it is only a trafficking stop. The said stop is to add downstream of engine flow, keeping the first sequence of operation for the first stop intact. Only additional stop FB needs incorporation at the end of the first program: Figure 5.9 highlights the program sequence; the new station is delimited by a red dashed line.

- FB200: Motor Control Function Block runs the machine conveyor;
- FB240: RFID Function Block reads the required tasks;
- Work In progress completes;
- FB240: RFID Function Block writes the operations' status;
- FB230: HMI Function Block displays the status and releases the pallet;
- FB220: Stop Function Block releases the pallet;
- FB220: Stop Function Block controls the stop;
- FB230: HMI Function Block, triggers display again and to release once more the pallet, and
- FB220: Stop Function Block releases the pallet.

The new machine setup only requires minor program changes on the FC already running the station. With countable keyboard stokes the new FC gets modified. It then becomes a matter of inserting the FBs sequentially. The new FC is attached as Appendix C.



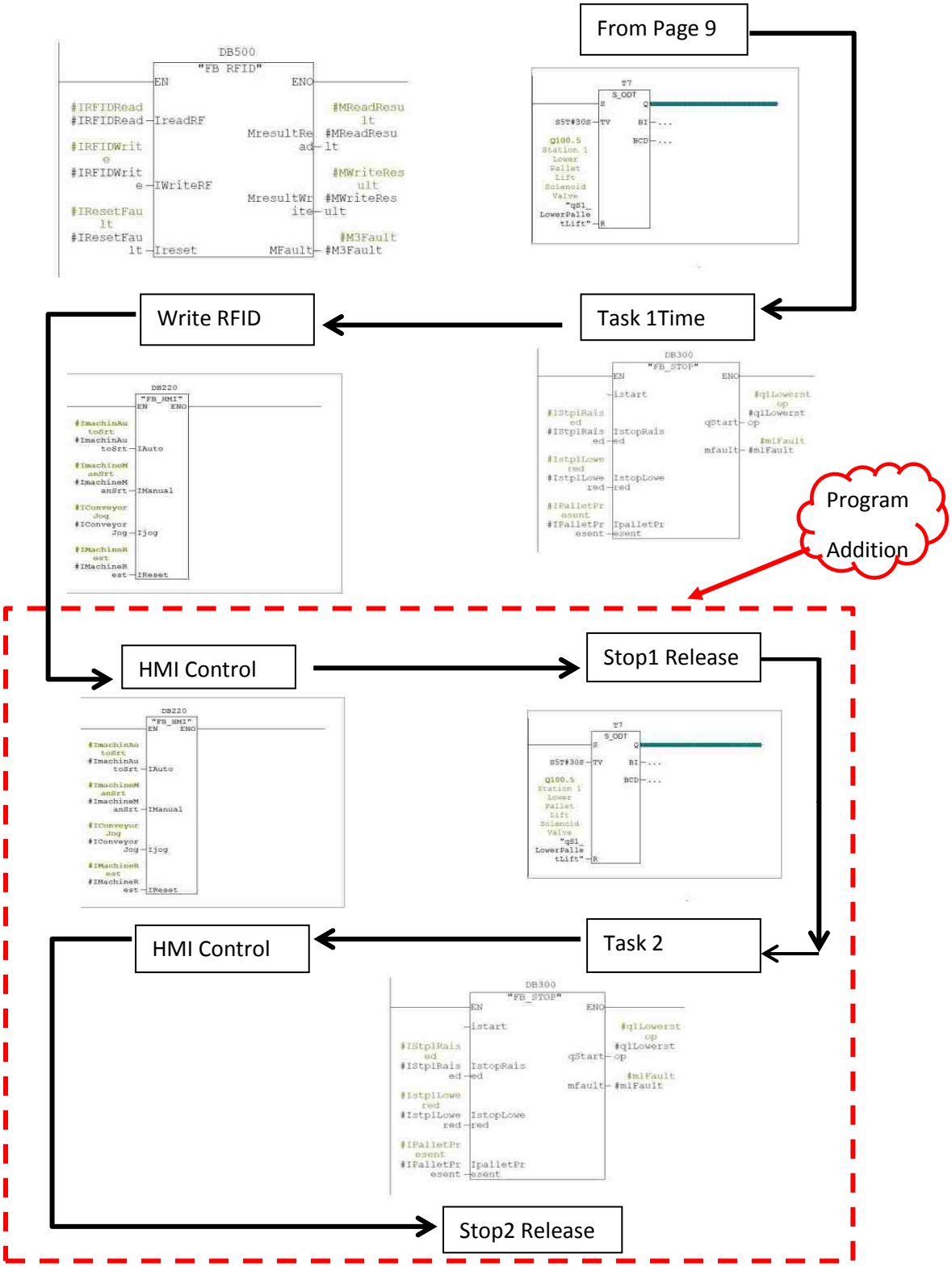



Figure 5.9: Reconfigured Manual Workstation Program

## 5.6 Summary

This chapter discussed two case studies. Case study #1 introduced in Chapter 4 explained a methodology essential to this research, and all stages were discussed. A logic program for a machine is easy to write as proven herein, once a mechatronic library is completed. Related programs are left to appendices because of size limitations. Case study #2 proves that modifying the controls logic of a running program is simple. All steps are also discussed here, while programs are left to the appendices.

# CHAPTER 6

## Conclusions and Future Work

## 6 Conclusions

A design process is presented in this thesis to help controls engineers develop reconfigurable codes for reconfigurable machines and systems. Chapter 1 introduced the motivation for the design of a reconfigurable control system for an engine assembly line, and outlined objectives and motivation for the thesis. Chapter 2 presented a thorough literature survey investigating various academic methods and researches, but none of the methodologies were industry ready. Chapter 3 investigated systems engineering approach for control software design. Chapter 4 proposed a novel method to design and develop a reconfigurable and industry-applicable controls system. A case study to support the methodology was presented in Chapter 5. Chapter 6 contains conclusions and discusses the entire thesis while proposing continuations to this work.

The proposed design method for in this thesis requires reconfigurability of mechanical system as a condition for use, in terms of modularity. A distributed architecture for controls is also required. Best results are obtained if mechanical hardware and controls are closely coordinated at the development stage. The methodology proposed in this thesis offers several benefits, such as time savings for control software designs, retooling time minimization for machinery compared to current industrial practices, maintenance of actual tools and programming languages known on production floors. The proposed metrology resulting from this research leads to a robust control software, especially when reused many times. Ladder logic is currently the most valuable element for plant floor personnel and the only language every electrician and most controls engineers use and master per resemblance to electrical diagrams. Other high level language seems to add abstraction to reasoning, and can cause major challenges to production if and when problems arise during the life cycle of manufacturing system. The second axiom in the Axiomatic Design encourages minimization of information content in any design, and considers as best design that containing the least amount of information, or the simplest. The methodology presented herein is systematic and easy to use, and can conclusively be

adapted to develop coding, using existing tools and programming methods, in a systematic manner geared to production reconfigurability.

## 6.1 Contributions

Main contributions from academic and industrial perspectives of the research presented herein include:

- Adoption of Component Based Software ideas to develop a machine control program;
- Introduction of DSM matrix and Cladograms, modularity measurement, and modularity indexing as a new contribution to control logic design;
- Implementation of a systematic methodology for the decomposition of mechatronic devises into modules for the creation of logic Function Blocks;
- Conceptualization of design for Function Blocks using Axiomatic Design, to simplify designs resulting in robust and reliable coding with minimum information;
- Development of conceptual design further into the creation of Library of Function Blocks using Ladder Logic;
- Creation of a machine program using Function Blocks, by means of Function Call (FC) readily available in the IEC 61311-3 (similar to calling subroutines in high level coding languages). The FBs sequentially call to execute given tasks;
- Demonstration of Ladder logic's ability to using FBs in creating control programs for a Reconfigurable Manufacturing System, and
- Reduction in control software design, significantly, by using available/book-shelved FBs, leading to substantial savings in budgeting assembly lines.

## 6.2 Research Significance

The methodology introduced controls program development supporting the re-configurability paradigm. An engine assembly production line is selected as a case study. The proposal is simple, systematic, and easy to implement. Systems engineering methodology, namely the "Vee model", was adopted to the process. The approach consists of two main phases: 1- Decomposition of mechatronic systems to best

Granularity, and 2- Axiomatic Design as a conceptual design for FB creations. The case study demonstrated the use of Ladder logic to develop control programs for reconfigurable machines.

Modularity accompanies granularity. A DSM matrix is created for the mechatronic system with information exchange as a link; then a Cladogram is generated to decompose the system in visually distinguishable modules with modularity indexes calculated for best result validation. It is a very important phase dictating the number of modules to be used for each machine. Still, the methodology could be fully automated.

Conceptual design of FBs is simplified with the use of Axiomatic Design where each module is decomposed to basic components. A flow diagram is then generated using Acclaro software. The flow chart is used as a blueprint in subsequent coding as it saves time in designing and debugging controls software. The software uses CRs, FRs, and DPs a designer identifies. The software then generates a design tree and flow diagrams.

The challenge remains in choosing a programming language to develop FBs. Ladder logic and IEC 61311-3 Standard govern PLC programing languages, even if they lack academia's consensus for use within reconfigurable manufacturing. Ladder logic is well-developed with a proven performance record in industry. This thesis demonstrates that Ladder logic can still be used to build modular programs that are easily debugged and reused. Ladder logic suits the latest PLCs, becoming very powerful in programming interfaces, with abilities to handle any task.

## 6.3 Industrial Significance

The research presented in this thesis is geared towards industry, and the case study is an actual manual work station. Many benefits could be cited. The control cost for a manufacturing system such as an automotive engine assembly is 30 to 40 %, where about 15 to 20 % is in software development. Creating a library of Function Blocks by itself, to use in an entire project, can easily save about 10% in development costs. Validated control modules for reuse should result in robust and fault-free programs. The use of Ladder logic saves on training costs for maintenance personal. Axiomatic design help

programmers produce codes along a systematic foundation contrarily to the current method of copy, paste, and modify until reaching a working (not an optimal) outcome.

## 6.4 Limitations

The methodology presented in this thesis still seems to have, as of now, a few limitations:

- Complex functions that require math and calculations will be challenging to code in Ladder logic. In these special cases, the methodology stands but the coding has to be completed using STL or appropriate language for the application. An example would be RFID function block that requires the manipulation of data as well as commands that is easily handled by STL language.
- The methodology is only applicable for distributed controls architecture as it assumes modularity of hardware, which is a prerequisite for a reconfigurable manufacturing system; the control software is built to suit the machine hardware.
- The machine programs require manual changes when modules are added or removed, the new program has to be edited, but only following copies and drops of FBs from libraries.

## 6.5 Future Work

Automating function block generation with Ladder logic should but increase the value of the methodology presented in this thesis from a reconfigurability perspective. A machine can potentially generate its program each time a module is added. IEC64199 is another standard well perceived in research, and investigating its usability and easiness of implementation in the developed methodology appears promising.

# REFERENCES/BIBLIOGRAPHY


AlGeddawy, Tarek, and Hoda ElMaraghy. "Optimum granularity level of modular product design architecture." *CIRP Annals-Manufacturing Technology* 62.1 (2013): 151-154.

AlGeddawy, Tarek, and Hoda ElMaraghy. "Design of single assembly line for the delayed differentiation of product variants." *Flexible services and manufacturing journal* 22.3-4 (2010): 163-182.

Babiceanu, Radu F., and F. Frank Chen. "Development and applications of holonic manufacturing systems: a survey." *Journal of Intelligent Manufacturing*17.1 (2006): 111-131.

Bauer, A., Browden, R., Browne, J., Duggan, J., & Lyons, G., (1994) "Shop Floor Control Systems - From design to implementation", 2nd edition, Chapman & Hall, ISBN: 0412581507.

Blanchard, Benjamin S., Wolter J. Fabrycky, and Wolter J. Fabrycky. Systems engineering and analysis. Vol. 4. Englewood Cliffs, New Jersey: Prentice Hall, 1990.

Bolton, William. *Programmable logic controllers*. Newnes, 2009.

Brecher, Christian, et al. "Open control systems: state of the art." Production Engineering 4.2-3 (2010): 247-254.

Breslin, J. G., O'Sullivan, D., Passant, A., & Vasiliu, L. (2010). Semantic Web computing in industry. *Computers in Industry*, *61*(8), 729-741.

Brennan, Robert William, Martyn Fletcher, and Douglas H. Norrie. "An agent-based approach to reconfiguration of real-time distributed control systems."*Robotics and Automation, IEEE Transactions on* 18.4 (2002): 444-451.

Browning, T. R. (2001). Applying the design structure matrix to system decomposition and integration problems: a review and new directions. *Engineering Management, IEEE Transactions on, 48*(3), 292-306.

Bussmann, Stefan, and Jorg Sieverding. "Holonic control of an engine assembly plant: an industrial evaluation." *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*. Vol. 1. IEEE, 2001.

Cândido, Gonçalo, and José Barata. "A multiagent control system for shop floor assembly." *Holonic and Multi-Agent Systems for Manufacturing*. Springer Berlin Heidelberg, 2007. 293-302.

Chan, Felix TS, et al. "Object-oriented architecture of control system for agile manufacturing cells." *Management of Innovation and Technology, 2000. ICMIT 2000. Proceedings of the 2000 IEEE International Conference on*. Vol. 2. IEEE, 2000.

Chirn, Jin-Lung, and Duncan C. McFarlane. "A holonic component-based approach to reconfigurable manufacturing control architecture." *Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on*. IEEE, 2000.

Dilts, D. M., Boyd, N. P., and Whorms, H. H. (1991). "The evolution of control Architectures for automated manufacturing systems." *Journal of Manufacturing Systems*, 10(1), 79-93.

Duffie, N. A. "Heterarchical control of highly distributed manufacturing systems." *International Journal of Computer Integrated Manufacturing* 9.4 (1996): 270-281.

ElBeheiry, ElSayed, Waguih ElMaraghy, and Hoda ElMaraghy. "The structured design of a reconfigurable control process." Advances in Design. Springer London, 2006. 559-571.

Eppinger, S. D., & Browning, T. R. (2012). *Design structure matrix methods and applications*. MIT press.

Farid, A. "A review of holonic manufacturing systems literature." *University of Cambridge Institute for Manufacturing, Cambridge UK, Tech. Rep* (2004).

Grabot, Bernard, and Pierre Huguet. "Reference models and object-oriented method for reuse in production activity control system design." *Computers in industry* 32.1 (1996): 17-31.

Hajarnavis, Vivek, and Ken Young. "An investigation into programmable logic controller software design techniques in the automotive industry." *Assembly Automation* 28.1 (2008): 43-54.

Harrison, R., et al. "Reconfigurable modular automation systems for automotive power-train manufacture." *International Journal of Flexible Manufacturing Systems* 18.3 (2006): 175-190.

Henrioud, J-M., L. Relange, and C. Perrard. "Assembly sequences, assembly constraints, precedence graphs." *Assembly and Task Planning, 2003. Proceedings of the IEEE International Symposium on*. IEEE, 2003.

Hirata corporation http://www.hirata.co.jp/en/ accessed March 10. 2014

Rockwell Automation http://ab.rockwellautomation.com. Accessed April 15. 2014

Phylip Software http://evolution.genetics.washington.edu/phylip.html. Accessed April 2. 2015

Acclaro software http://www.axiomaticdesign.com. Accessed May 1. 2015

Hirsch, M. (2010). *Systematic design of distributed industrial manufacturing control systems* (Vol. 6). Logos Verlag Berlin GmbH.

Holloway, Lawrence E., et al. "Automated synthesis and composition of taskblocks for control of manufacturing systems." *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 30.5 (2000): 696-712.

Howard, Alexander, Ashok Kochhar, and John Dilworth. "An objective approach for

generating the functional specification of manufacturing planning and control systems." *International Journal of Operations & Production Management* 18.8 (1998): 710-726.

Huang, Hui-Min, et al. "Open system architecture for real-time control using an uml-based approach." *Proceedings of the 1st ICSE Workshop on Describing Software Architecture with UML*. 2001.

Ka, B., László Monostori, and E. Szelke. "An object-oriented framework for developing distributed manufacturing architectures." *Journal of Intelligent Manufacturing* 9.2 (1998): 173-179.

Karl-Heinz, J., Tiegelkamp, M., IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids. Springer, 2010.

Katz, R., Min, B.-K., and Pasek, Z. (2000). "Open Architecture Control Technology Trends." ERC/RMS Report 35.

Kim, Sun-Jae, Nam P. Suh, and Sang-Gook Kim. "Design of software systems based on axiomatic design." *Robotics and Computer-Integrated Manufacturing* 8.4 (1991): 243-255.

Koestler, Arthur. "THE GHOST IN THE MACHINE." (1968).

Koren, Y., 1999, “Reconfigurable manufacturing systems,” Annals of the CIRP, Vol. 48, pp. 213.

Koren Y, Ulsoy AG. Reconfigurable manufacturing systems. Engineering Research Center for Reconfigurable Machining Systems. ERC/RMS report #1. Ann Arbor; 1997

Lee, Jin-Shyan, and Pau-Lo Hsu. "A PLC-based design for the sequence controller in discrete event systems." *Control Applications, 2000. Proceedings of the 2000 IEEE International Conference on*. IEEE, 2000.

Lucas, M. R., and D. M. Tilbury. "A study of current logic design practices in the automotive manufacturing industry." International Journal of Human-Computer Studies 59.5 (2003): 725-753.

Ljungkrantz, Oscar, Knut Åkesson, and Martin Fabian. "Practice of industrial control logic programming using library components." *Programmable logic controller, Intech* (2010): 17-32.

Ljungkrantz, O., Akesson, K., Fabian, M., & Yuan, C. (2010). Formal specification and verification of industrial control logic components. *Automation Science and Engineering, IEEE Transactions on*, *7*(3), 538-548.

Metzger, Mieczyslaw, and Grzegorz Polakow. "A survey on applications of agent technology in industrial process control." Industrial Informatics, IEEE Transactions on 7.4 (2011): 570-581.

Mei, Hong, et al. "ABC: An architecture based, component oriented approach to software development." *Journal of Software* 14.4 (2003): 721-732.

Mehrabi, Mostafa G., A. Galip Ulsoy, and Yoram Koren. "Reconfigurable manufacturing systems: key to future manufacturing." Journal of Intelligent Manufacturing 11.4 (2000): 403-419.

Mehrabi, Mostafa G., A. Galip Ulsoy, and Yoram Koren. "Reconfigurable manufacturing systems: Key to future manufacturing." *Journal of intelligent Manufacturing* 11.4 (2000): 403-419.

Monostori, László, József Váncza, and Soundar RT Kumara. "Agent-based systems for manufacturing." *CIRP Annals-Manufacturing Technology* 55.2 (2006): 697-720.

Motors, General. "Open, Modular Architecture Controls at GM Powertrain." May14 (1996): 39.

Odell, James. "Objects and agents compared." *Journal of object technology* 1.1 (2002): 41-53.

Orio, Giovanni Di, et al. "Control System Software Design Methodology for Automotive Industry." *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*. IEEE, 2013.

Panjaitan, S. D., and Georg Frey. "Development process for distributed automation systems combining UML and IEC 61499." International Journal of Manufacturing Research 2.1 (2007): 1-20.

Peng, Shih Sen, and Meng Chu Zhou. "Ladder diagram and Petri-net-based discrete-event control design methods." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 34.4 (2004): 523-531.

Pritschow, Günter, et al. "Open controller architecture–past, present and future." *CIRP Annals-Manufacturing Technology* 50.2 (2001): 463-470.

Proctor, F., 1998, "Practical open architecture controllers for manufacturing application," In: Open Architecture Control Systems (Koren, Y., et al. editors), IITA Publ., pp. 103-113.

Ryssel, U., Dibowski, H., & Kabitzsch, K. (2009, September). Generation of function block based designs using semantic web technologies. In *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on* (pp. 1-8). IEEE.

Segovia, Vanessa Romero, and Alfred Theorin. "History of control–History of PLC and DCS"." (2012).

Seungjoo Lee, Mark Adam Ang, and Jason Lee. Automatic generation of logic control. Technical report, Ford Motor Co., Univ. of Michigan and Loughborough Univ., 2006.

Valckenaers, P., Bonneville, F., Van Brussel, H., Bongaerts, L., Wyns, H., Results of the Holonic Control System Benchmark at the K.U. Leuven, Proc. of the CIMAT Conference (Computer Integrated Manufacturing and Automation Technology), Rensselaer Polytechnic Institute, Troy, NY, 10–12 October 1994, pp. 128–133.

Van Brussel, Hendrik, et al. "Reference architecture for holonic manufacturing systems: PROSA." *Computers in industry* 37.3 (1998): 255-274.

Vrba, Pavel, et al. "Rockwell Automation's Holonic and Multiagent Control Systems Compendium." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 41.1 (2011): 14-30.

Vyatkin, Valeriy. "Software Engineering in Industrial Automation: State-of-the-Art Review." *Industrial Informatics, IEEE Transactions on* 9.3 (2013): 1234-1249.

Vyatkin, V. (2013). Software engineering in industrial automation: State-of-the-art review. *Industrial Informatics, IEEE Transactions on*, *9*(3), 1234-1249.

West, J. (2003). "How open is open enough? Melding proprietary and open source platform strategies." Research Policy, 32(7), 1259.

Xia, Feng, Zhi Wang, and Youxian Sun. "Towards component-based control system engineering with IEC61499." *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*. Vol. 3. IEEE, 2004.

APPENDICES

## Appendix A: Motor Control Flow Chart Figures

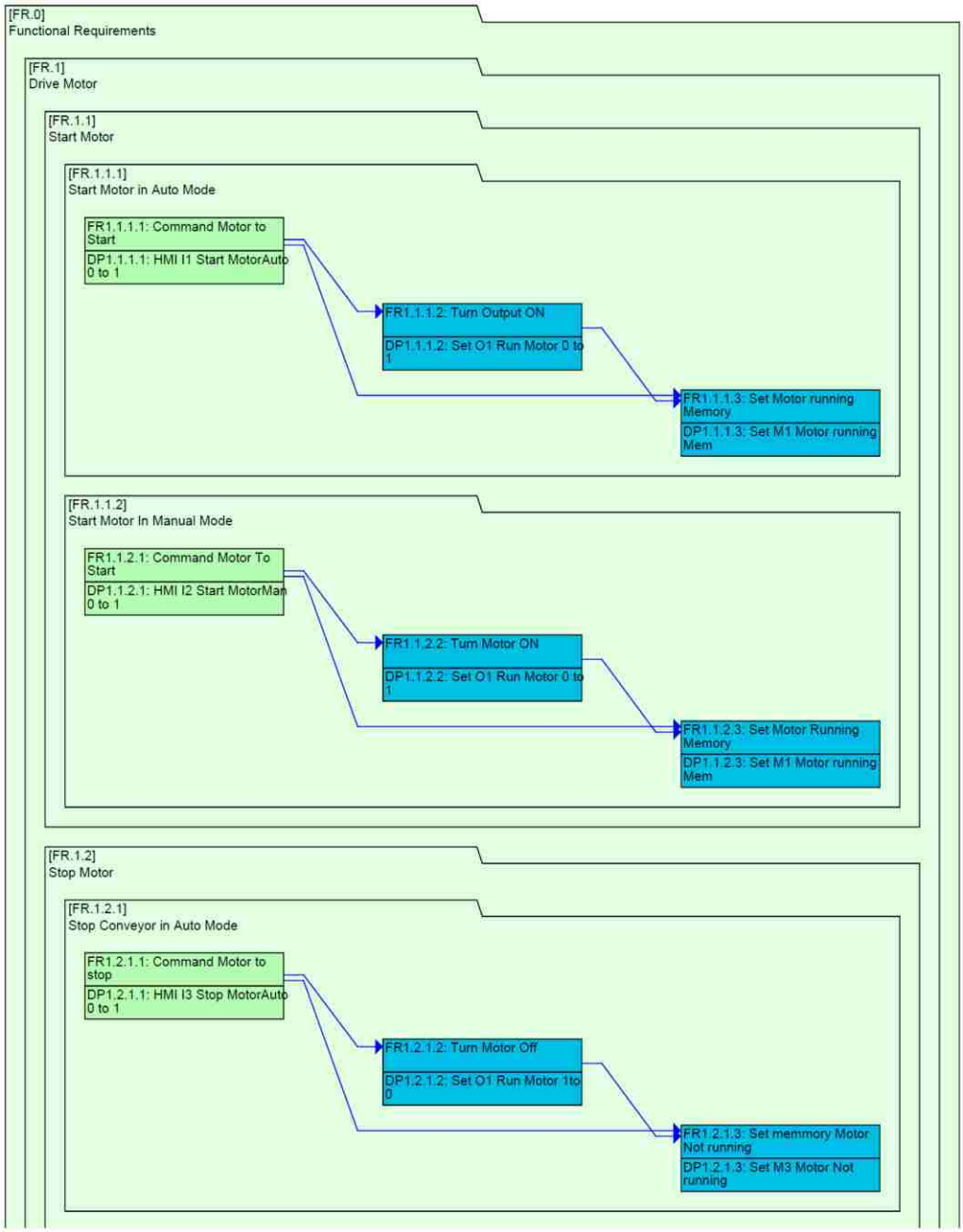



Figure A1: Motor Control Flow Chart

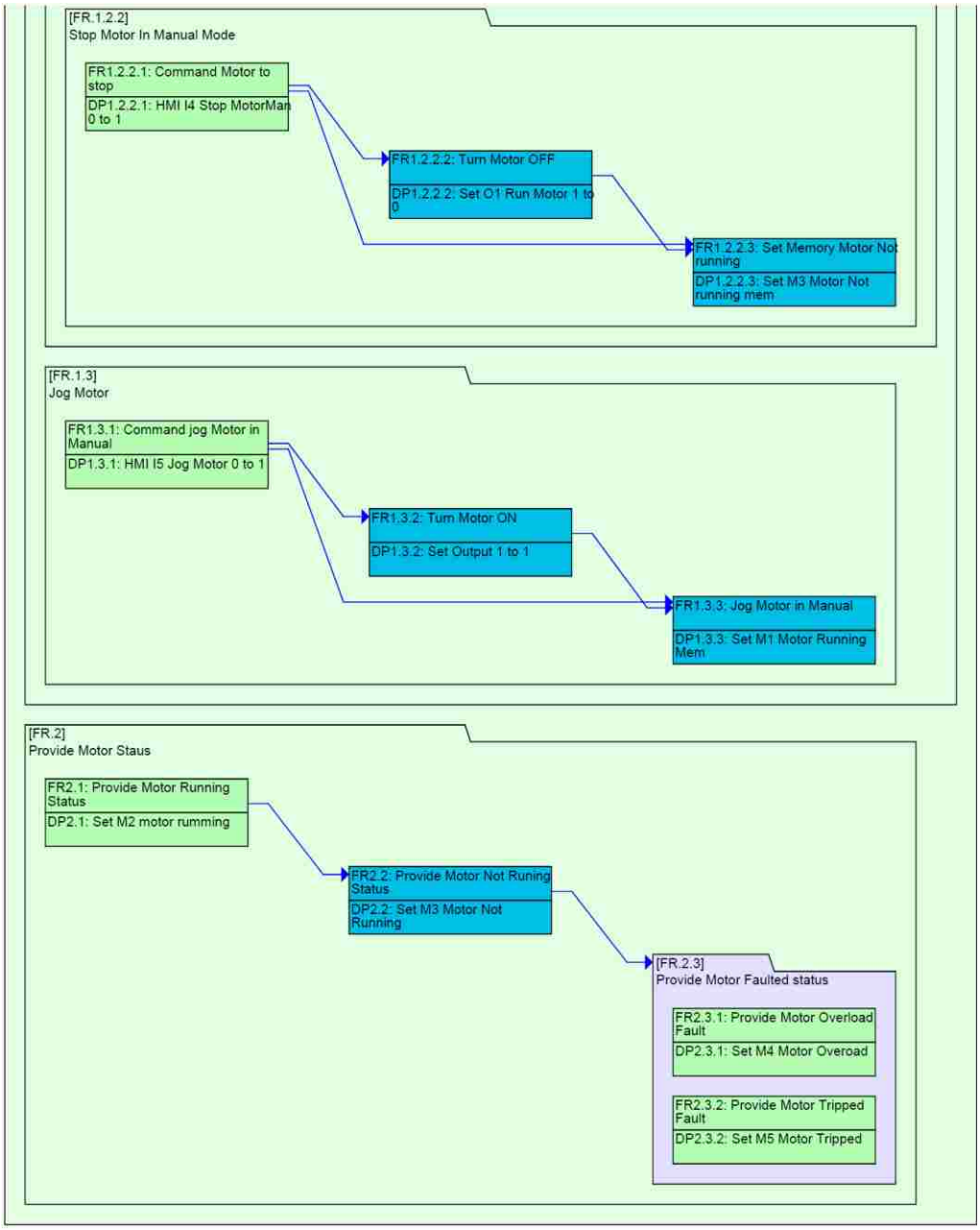



Figure A1: Motor Control Flow Chart (continued)

**Appendix B:** PLC Program for Manual Station in Figure 4.1



**FC200 - <offline>**
"Main" Manual station
**Name:** Station1 **Family:** Thesis
**Author:** h.Tabti **Version:** 0.1
**Block version:** 2
**Time stamp Code:** 05/07/2015 12:18:23 AM
**Interface:** 05/07/2015 12:18:23 AM
**Lengths (block/logic/data):** 00212 00106 00006

**Object properties:**
S7_language 9(1) English (United States) 05/07/2015 12:18:25 AM

| Name | Data Type | Address | Comment |
|---|---|---|---|
| IN | | 0.0 | |
| IStp1Raised | Bool | 0.0 | Stop#1 raised Input |
| IStp1Lowered | Bool | 0.1 | Stop#1 Lowered Input |
| IPalletPresent | Bool | 0.2 | Pallet Front Stop#1 |
| IAutoStrt | Bool | 0.3 | Start Conveyor Auto Mode |
| IManStart | Bool | 0.4 | Start Conveyor Man Mode |
| ICvyrJog | Bool | 0.5 | Jog Conveyor Man Mode |
| IReadRFIDCmd | Bool | 0.6 | Command RFID read |
| IWriteRFIDCmd | Bool | 0.7 | Command RFID write |
| IResetRFID | Bool | 1.0 | Fault Reset |
| IMachineAutoStrt | Bool | 1.1 | Machine in Auto mode |
| IMachineManStrt | Bool | 1.2 | Machine in Manual mode |
| ICvyrJog_1 | Bool | 1.3 | Cvyr Joging |
| IMachineFaulted | Bool | 1.4 | Machine Faulted Status |
| OUT | | 0.0 | |
| qlLowerstop | Bool | 2.0 | Lower Stop #1 |
| mlFault | Bool | 2.1 | Stop#1 Faulted |
| qlRunCvyr | Bool | 2.2 | Run Motor Cvyr |
| MlCvyrFault | Bool | 2.3 | Cvyr Motor Faulted |
| MIRFReadRslt | Bool | 2.4 | Read Result RFID |
| MIRFWriteRslt | Bool | 2.5 | Write Result RFID |
| MIRFStatusFlt | Bool | 2.6 | Fault status RFID |
| IN_OUT | | 0.0 | |
| TEMP | | 0.0 | |
| RETURN | | 0.0 | |
| PBT_VAL | | 0.0 | |

**Block: FC200 Manual Work Station Program**

This is a Program to run a Manual Work Station On Figure 4.1.

Network: 1 Call FB220 to run conveyor motor

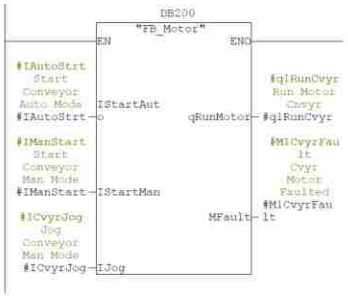
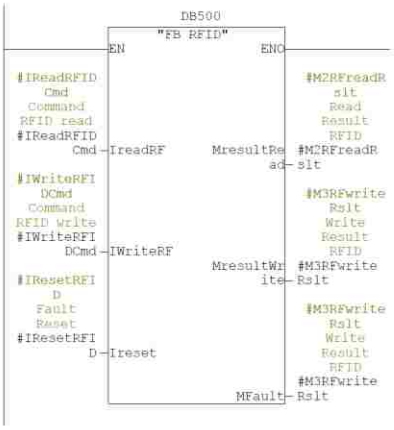





Figure B1: PLC Program for Manual Station in Figure 4.1

Network: 2 Call FB230 RFID to Read RF Tag




Network: 3 30 Second Timer for Task Completion

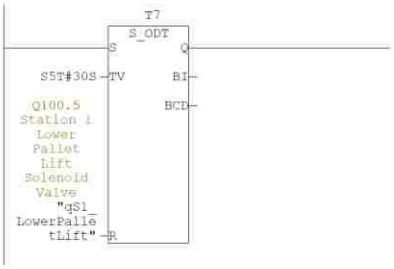
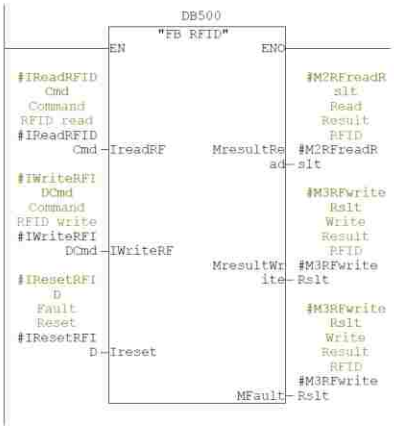





Figure B1: PLC Program for Manual Station in Figure 4.1 (continued)

Network: 4 Call FB230 RFID to Write RF Tag




Network: 5 Call FB 210 for HMI display and Control

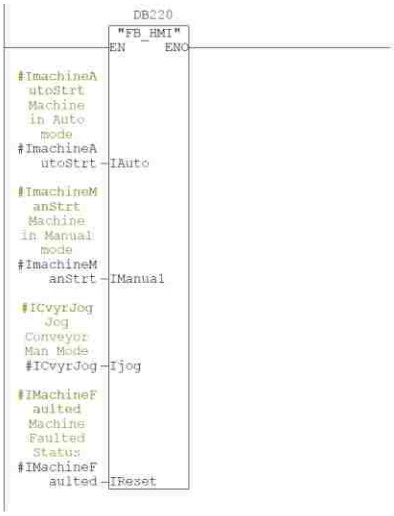





Figure B1: PLC Program for Manual Station in Figure 4.1 (continued)

Network: 6 Call FB200 for stop Control(pallet Release)

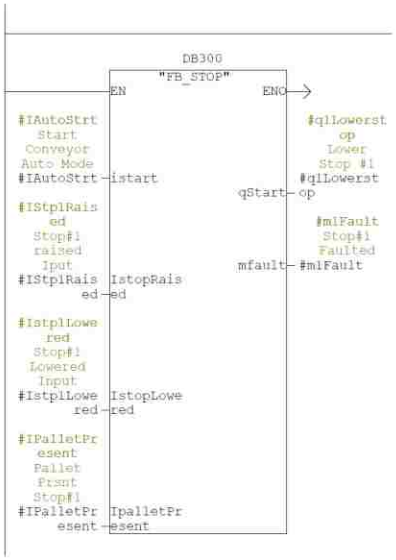





Figure B1: PLC Program for Manual Station in Figure 4.1 (continued)

**Appendix C**: PLC Program for Manual Station in Figure 5.8



**FC200 - <offline>**
"Main" Manual station
**Name:** Station1 **Family:** Thesis
**Author:** h.Tabti **Version:** 0.1
**Block version:** 2
**Time stamp Code:** 05/07/2015 12:18:23 AM
**Interface:** 05/07/2015 12:18:23 AM
**Lengths (block/logic/data):** 00254 00106 00006

**Object properties:**
S7_language 9(1) English (United States) 05/07/2015 12:18:25 AM

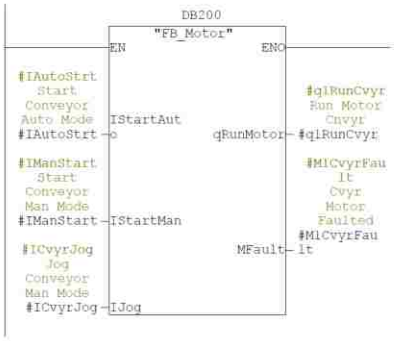
| Name | Data Type | Address | Comment |
|---|---|---|---|
| IN | | 0.0 | |
| IStp1Raised | Bool | 0.0 | Stop#1 raised Iput |
| IstpILowered | Bool | 0.1 | Stop#1 Lowered Input |
| IPallatPresent | Bool | 0.2 | Pallet Front Stop#1 |
| IAutoStrt | Bool | 0.3 | Start Conveyor Auto Mode |
| IManStart | Bool | 0.4 | Start Conveyor Man Mode |
| ICvyrJog | Bool | 0.5 | Jog Conveyor Man Mode |
| IReadRFIDCmd | Bool | 0.6 | Command RFID read |
| IWriteRFIDCmd | Bool | 0.7 | Command RFID write |
| IResetRFID | Bool | 1.0 | Fault Reset |
| ImachineAutoStrt | Bool | 1.1 | Machine in Auto mode |
| ImachineManStrs | Bool | 1.2 | Machine in Manual mode |
| ICvyrJog_1 | Bool | 1.3 | Cvyr Joging |
| IMachineFaulted | Bool | 1.4 | Machine Faulted Status |
| IStp2Raiseded | Bool | 1.5 | Stop#2 raised Input |
| IPallatPrent2 | Bool | 1.6 | Pallet Present at Stop#2 |
| IstpILowered | Bool | 1.7 | Stop#2 Lowered Input |
| OUT | | 0.0 | |
| qiLowerstop | Bool | 2.0 | Lower Stop #1 |
| miFault | Bool | 2.1 | Stop#1 Faulted |
| qiRunCvyr | Bool | 2.2 | Run Motor Cvryr |
| MiCvyrFault | Bool | 2.3 | Cvyr Motor Faulted |
| MCPFreadRslt | Bool | 2.4 | Read Result RFID |
| MCPFwriteRslt | Bool | 2.5 | Write Result RFID |
| MARFStatusFlt | Bool | 2.6 | Fault status RFID |
| IPallatPsent1 | Bool | 2.7 | Pallet Present at Stop#1 |
| qiLowerstop_1 | Bool | 3.0 | Lower Stop #2 |
| miFault_1 | Bool | 3.1 | Stop#2 Faulted |
| IN_OUT | | 0.0 | |
| TEMP | | 0.0 | |
| RETURN | | 0.0 | |
| RET_VAL | | 0.0 | |

| Block: FC200 Manual Work Station Program |
|---|
| This is a Program to run a Manual Work Station On Figure 4.1. |



Figure C1: PLC Program for Manual Station in Figure 5.8

Network: 1 Call FB220 to run conveyor motor




Network: 2 Call FB230 RFID to Read RF Tag

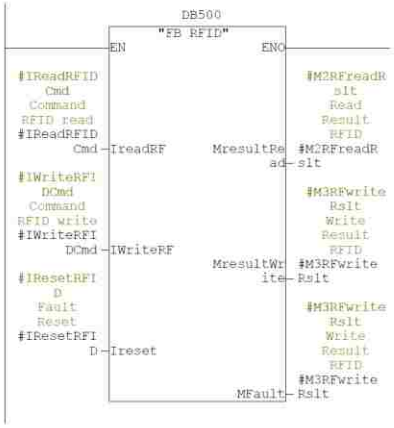
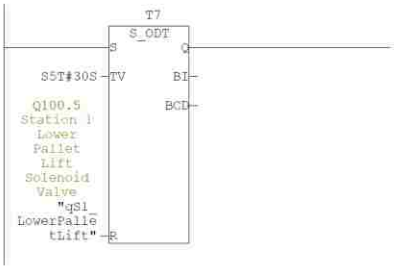





Figure C1: PLC Program for Manual Station in Figure 5.8 (Continued)

Network: 3 30 Second Timer for Task#1 Completion




Network: 4 Call FB230 RFID to Write RF Tag

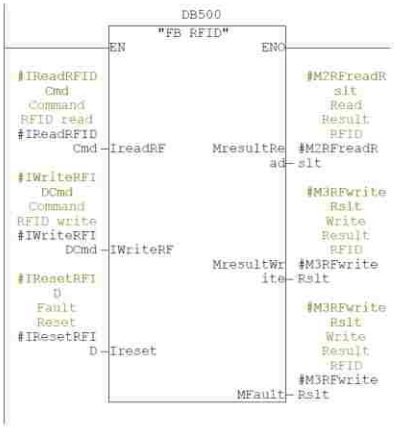
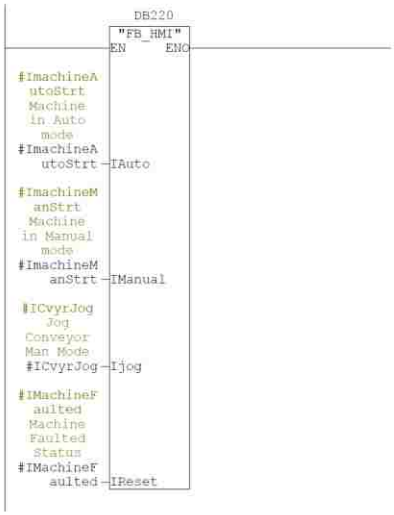





Figure C1: PLC Program for Manual Station in Figure 5.8 (Continued)

Network: 5 Call FB 210 for HMI display and Control




Network: 6 Call FB200 for stop#1 Control(pallet Release at Stop #1)

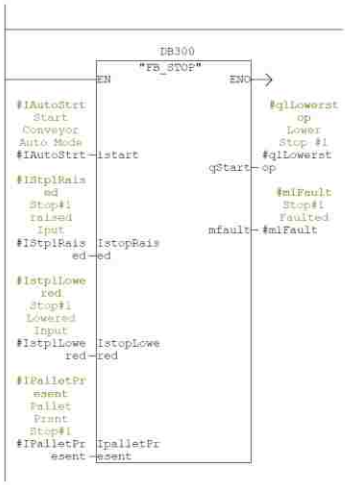
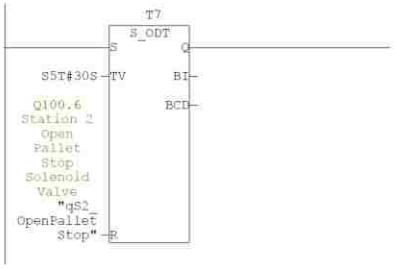





Figure C1: PLC Program for Manual Station in Figure 5.8 (Continued)

Network: 7 30 Second Timer for Task#2 Completion




Network: 8 Call FB 210 for HMI display and Control

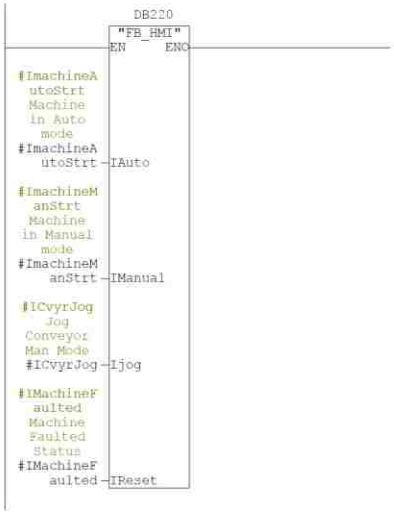





Figure C1: PLC Program for Manual Station in Figure 5.8 (Continued)

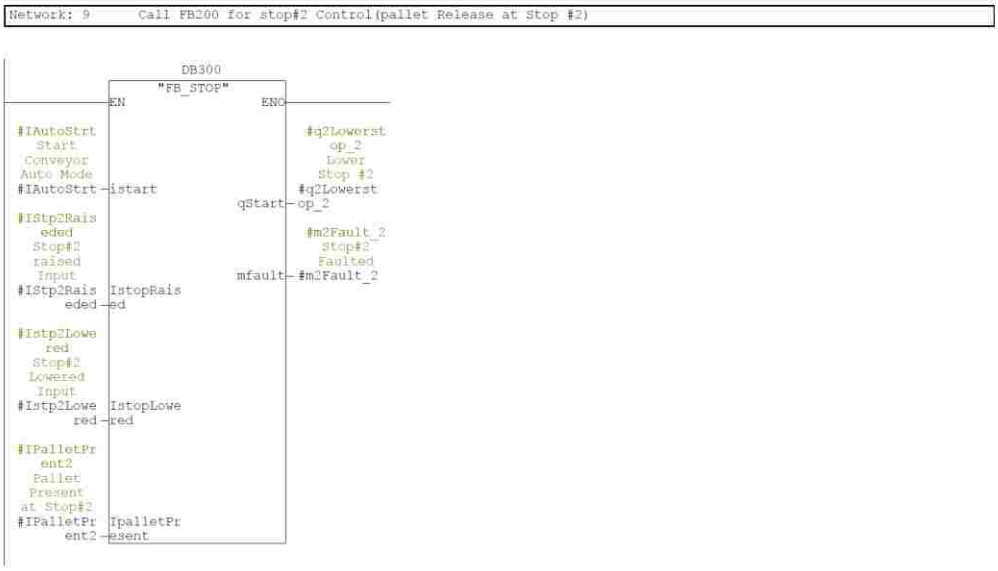





Figure C1: PLC Program for Manual Station in Figure 5.8 (Continued)

# VITA AUCTORIS

| | | |
|---|---|---|
| NAME: | Hamid Tabti | |
| PLACE OF BIRTH: | Algiers Algeria | |
| YEAR OF BIRTH: | 1966 | |
| EDUCATION: | Baccalaureat Science Harrach | 1985 |
| | Ecole National Polytechnique | 1990 |