UNIVERSITY OF BRITISH COLUMBIA

# MECH 527 - Mechatronic System Design Project

## Data Harvesting, Automation, and Controls

- Project Report -

Prepared by: Jonas Chianu (31298391)

Supervisor: Dr. Christoph Sielmann

Submitted on: August 26,2021

## Table of Contents

**Acknowledgement**

The MECH 540B original lab guideline and instructions, in addition to the necessary lab equipment, were prepared by Dr. Christoph Sielmann.

The MANU (Manufacturing Engineering) automation kit was designed and fabricated by the joint effort of Dr. Sielmann and the department of manufacturing engineering for use in the MANU 386 labs at the University of British Columbia (UBC).

The infrastructure plans for the UBC data management system in project 2 were already developed by Dr. Sielmann. My role involved findings ways to extract data from the different equipment as well as what useful data could be extracted.

I would like to express my gratitude and appreciation for everyone that has helped and contributed to the completion of my project. I am most grateful for all the help I got from my supervisor, Dr. Sielmann, and for all the guidance he provided and for always being available to answer my question and breaking down any problems I had. I would also like to thank Casey Keulen, a Ph.D student in the department of materials engineering, for taking time to work through how to operate the different equipment in the lab. Lastly, I would love to thank my parents and sibling, for their constant encouragement throughout my projects.

## 1. Introduction

Data harvesting, automation, and controls are very important topics in not only mechatronics but also in various other fields such as manufacturing, mechanical engineering, chemical engineering and various STEM (Science, technology, engineering, and mathematics) disciplines. As industries have increasingly become more data driven and more automated, knowledge and expertise in these areas have become increasingly more in demand. Data harvesting and collection involves extracting useful real-time data from various equipment and storing the data in a data server such as SCADA (Supervisory Control and Data Acquisition) for monitoring and control purposes to deal with real time and critical events. The automation kit project primarily involves control work on a PLC (Programmable Logic Controller). Working with a PLC is an important skill required in industries such as manufacturing for controlling different systems and when paired with SCADA or another data server allows for the full control of all the equipment in a plant. The FPGA lab upgrade projects involves knowledge of digital systems and software diagnostics.

A picture of the automation kit is shown in figure 1.1. The automation kit is primarily used for MANU 386 labs to teach the students about developing a PLC project and automating the discrete control of pneumatic valves for controlling a pneumatic cylinder with position feedback.

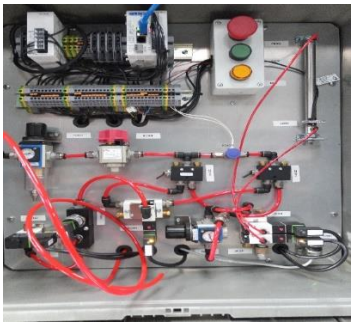


*Figure 1.1: Automation Kit*

## Purpose and Overview

The report details three different projects that were conducted. The first project involves updating and modernizing some of the labs, concerning digital systems and FPGA, in MECH 540B. The goal is to upgrade the lab to a newer version of Quartus which is a programmable logic device design software. This involves changing the lab procedures to reflect the changes in software and detailing any added dependencies and changes required.

The second project involves programming a ladder logic using a PLC software to control the automation kits piston level. The automation kits are designed to be operated horizontally but it is preferable to operate them vertically. This involves factoring in the effect of gravity on the piston cylinder. So overall the goal is to control the position of the pneumatic cylinder and negate the effects of gravity by controlling the pneumatic valves.

The third project involves data harvesting from different equipment in the Frank Forward 219 lab. The goal is to compile a list of communication interfaces and useful data that can be extracted from all the equipment of interest. This information will be very important in future implementation of a data server that can be used to store all this information and will allow for the monitoring of all the equipment of all the lab equipment in UBC (University of British Columbia) Vancouver and UBC Okanagan. A Scada application was also implemented for one of the equipment.

The deliverables are three updated lab manuals for the first project (MECH 540B labs), the Ladder logic program for the automation kit, a complete list of communication interfaces and useful data for different equipment in the lab, a final report and a final presentation summarizing the work done for each of the projects.

## 2. Project A: MECH 540B Lab Upgrade

The first lab involves updating and modernising lab 3, 4 and 5 for MECH 540B. These are the FPGA labs, concerned with providing student a hands on experience on digital systems. The goal is to update the lab instruction from Quartus 16 to Quartus 20. Lab 3 is an introduction to FPGA and creating an application on an FPGA. Lab 4 pairs the FPGA with an ADC by implementing a VHDL state machine. Finally, Lab 5 involves using the FPGA to build a microcontroller (MCU).

### 2.1. Equipment

There are several equipment necessary to conduct all three labs detailed as followed.

A laptop with internet capability, figure 2.1, is needed to download documentations, software, and source codes.

*Figure 2.1: Laptop with internet*

DE1-SoC Development board includes a robust hardware platform built around the Altera System-on-Chip (SoC) FPGA. It is used to assists users in designing an application using the FPGA, because it contains many additional capabilities including increased re-configurability and a higher performance. Figure 2.2 shows the board with all the different components available and figure 2.3 is a block diagram of the board, displaying the different components in an easier manner.

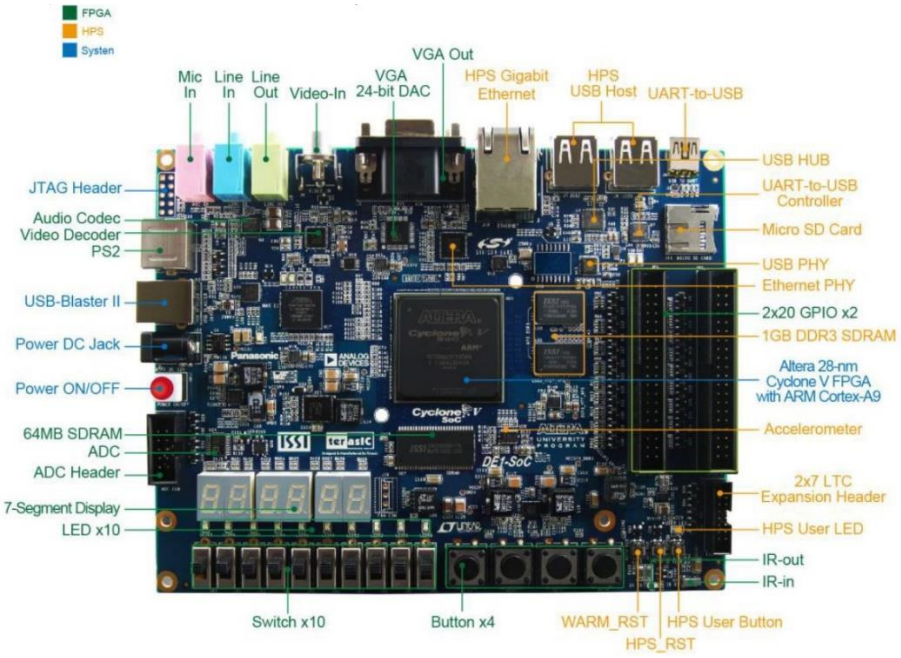



*Figure 2.2: DE1-SoC development board [1]*

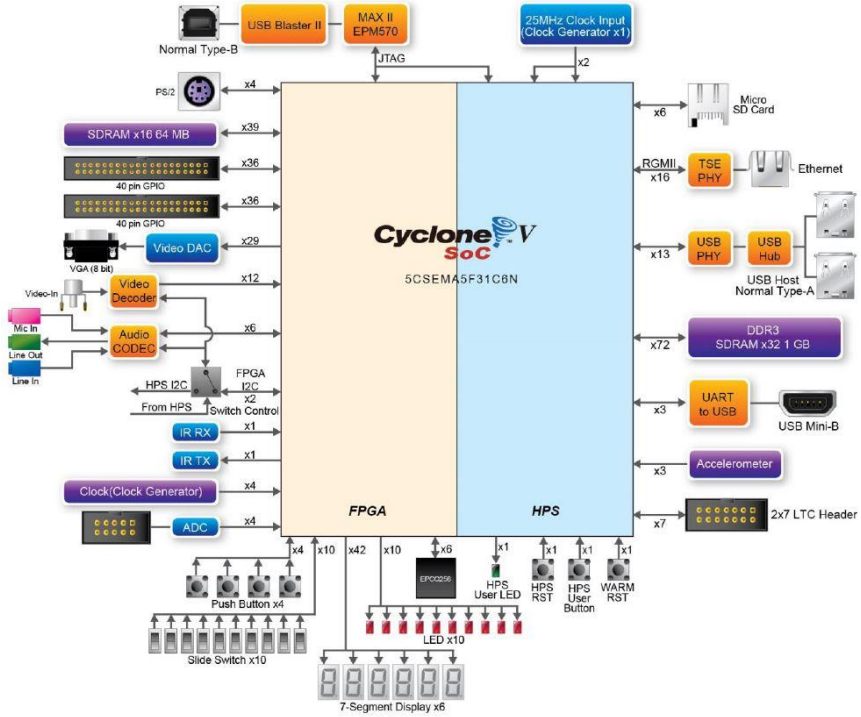



*Figure 2.3: Block diagram of DE1-SoC [1]*

The ADC connector break-out cable, figure 2.4, is used to connect the ADC on the DE1-SoC board to a power source. The ADC allows the user to read the voltage reading on a power supply and the ability to display the readings on the 7-segment  display of the DE1-Soc board.

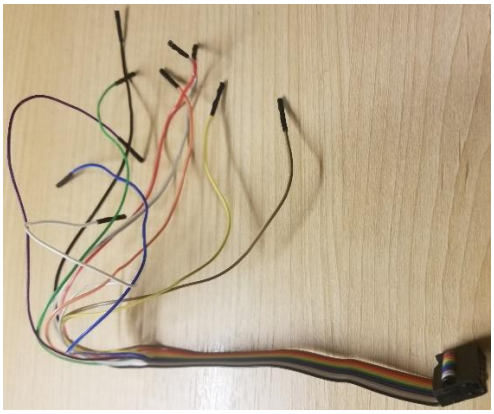


*Figure 2.4: ADC connector break-out cable*

Resistor kit, capacitor kit and prototyping board (breadboard) shown in figures 2.5, 2.6, 2.7, with additions of wires and a power supply, are used to construct circuits.


*Figure 2.5: Resistor Kit*


*Figure 2.6: Capacitor Kit*

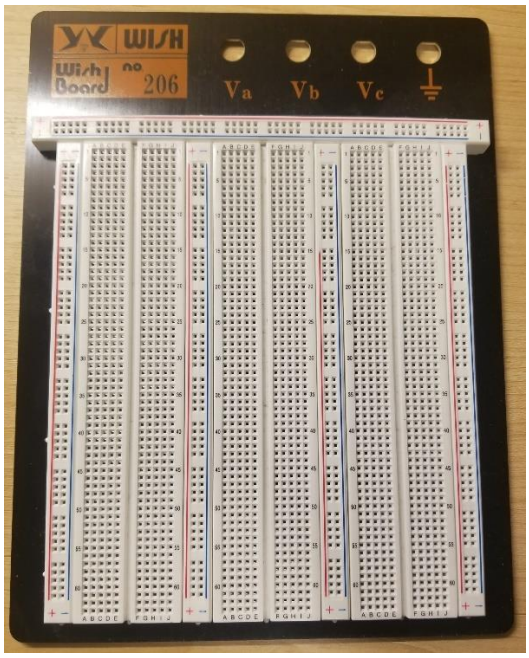



*Figure 2.7: Prototyping Board*

The Analog Discovery (AD2) board, figure 2.8, is a USB oscilloscope, logic analyzer and variable power supply. It is used primarily in this project to provide DC and AC power supply.


*Figure 2.8: Analog Discovery (AD2) board*

### 2.2. Quartus Installation instructions

This is an entirely new documents to help in the installation of Quartus 20 and Nios II Software Build Tool for Eclipse. The document can be found in Appendix A ("Quartus Installation Instructions"). Quartus Prime Lite Edition 20 can be downloaded from the URL in the installation instructions.

Unlike Quartus 16 which comes with Nios II Software Build Tool already installed, Nios II must be manually installed and configured for Quartus 20. Step 1 provides a link to download and install Eclipse C/C++ Development Tooling (CDT), which is an Eclipse IDE for Mars.2, by following the instructions in the website. The next step involves installing Windows Subsystem (WS) for Linux. This allows developers the ability to run a GNU/Linux environment directly on Windows, unmodified, without the overhead of a traditional virtual machine or dual-boot setup. The link provided in step 2 walks the student through the installation process. It is key to install WS1 instead of WS2 due to the former having better support. If the instructions from step 2 were followed, Ubuntu (or another Linux distribution) should be installed, and step 3 requires the student to type and execute a couple commands in the Ubuntu shell as listed in the instruction sheet. At this point the Nios II Software Build Tool should work.

### 2.3. Lab 3: Introduction to FPGA

There are no changes to the instructions for lab 3 so the lab document is the same. However, section 1 ask that the student follow the "DE1-Soc My First FPGA" guide, attached with the original lab 3 instruction and can also be found online, and there are a few differences here. The differences mainly concern section 2.1 (Assign the Device) of the guide. On page 10 of the guide "Family and Device Settings" is listed as page 3, but in version 20 it is page 4. This change is very minimal but

worth noting, nonetheless. There are also some changes the UI and layout, but this are also not big changes that are worth going into details on. The updated version can be found in Appendix A.

### 2.4. Lab 4: Pairing FPGA with an ADC

Two changes were made to the instructions for Lab 4 and these changes are listed below.

Changes:

1. Step 7 on section 1 was deleted. This asks the student to activate TalkBack. This feature will allow the student to use many licensed features of the software, including multicore compilation and integrated logic analyzer. Unlike version 16 where TalkBack has to be activated manually, in version 20, this feature is automatically activated so this step can be skipped.
2. There is a small change is the name of the SignalTap. In version 16, it is called SignalTap II, while in version 20, it is just called SignalTap.

The updated version can be found in Appendix A.

### 2.5. Lab 5: Building a Microcontroller (MCU)

Lab 5 had the most extensive changes to the manual and these changes are listed below.

Changes:

1. As mentioned previously in section 2.4, Nios II Software Build Tool must have previously been installed manually by the student.
2. QSYS, a tool for combining IP cores on common buses, has been renamed to Platform Designer.

3. There are some changes in step 1 of section 3 due to Nios II having to be installed manually. Nios II can no longer be open directly from Quartus, instead the Nios II executable must be run using the Nios II command shell. This ensures that the BSP generation will work on Eclipse with no errors. This will open Eclipse, which is an open-source IDE for software development.

4. An additional sub step must be added between step 8d and 8e of section 3. The system id checks must both be checked before closing the window or else there will be errors when the students run the project. An additional image, figure 2.9, showing the System id checked, is provided.

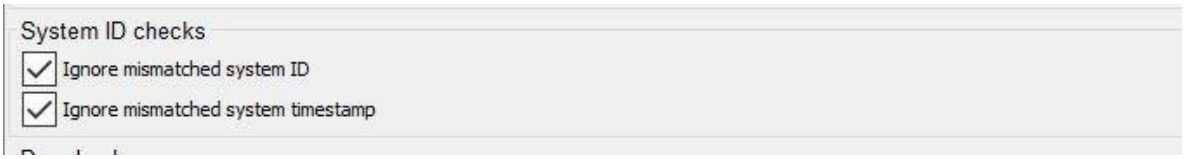



*Figure 2.9: System ID checks*

The updated version can be found in Appendix A.

## 3. Project B: MANU Automation Kit Controls

The second project involves automating the discrete control of pneumatic valves for controlling the position level of a pneumatic cylinder. The current automation kit was designed to be operated horizontally, but for spatial configuration reasons, it is preferable to operate the kit vertically. The only issue is the fact that the pneumatic cylinder/piston is subjected to gravity when positioned vertically, so that must be taken into account in the automated controls and PLC program.

### 3.1. Overview of Automation Kit

A picture of the automation kit is shown in figure 3.1 with all the components labelled with standard P&ID abbreviations. Figure 3.2 presents a panel layout of the automation kit which is easier to understand. Starting with the PLC (Programmable Logic Controller) module, which is akin to the brains of the system and all the required control actions are programmed into the PLC. It is like a computer but adapted for manufacturing processes but has uses in many different industries. It is very reliable, easy to program and easy to diagnose and maintain. ClickPLC was chosen as the PLC software. The software uses ladder logic, which is the most popular programming language used with PLCs. It is a visual programming language and very easy to develop and adapt.

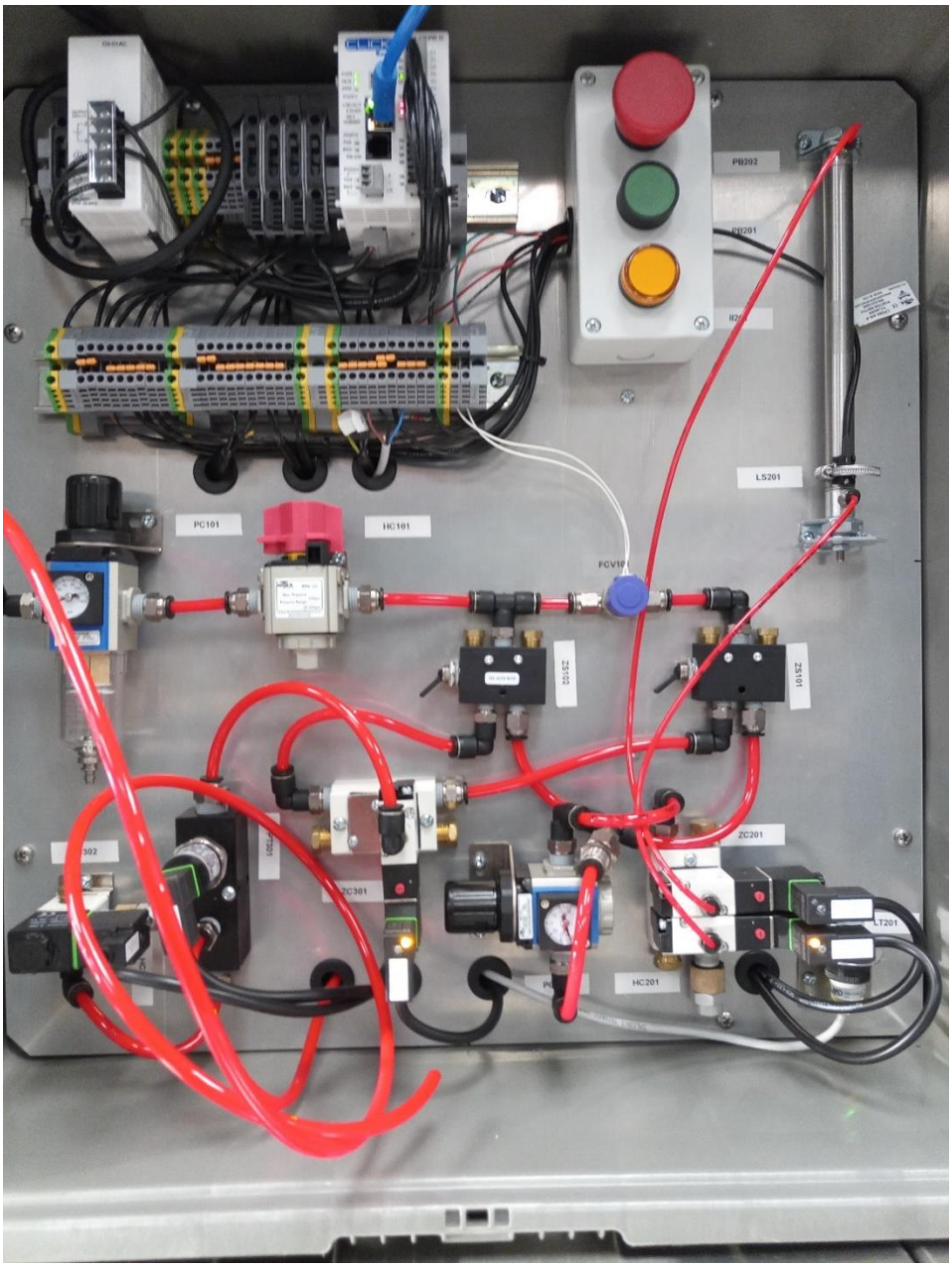



*Figure 3.1: Picture of Automation Kit*

There are a few other components that are critical in the automation kit. The DIN rail terminal block module provides a clean, safe, and efficient way for providing connecting between electrical components and for connecting and identifying two or more wires to a circuit. It connects to a pneumatic system that flows into the pressure regulator/filter. The pressure regulator is

connected to the pressure release valves which reduces the pressure if needed. The piston cylinder on the right of figure 3.2 is the key component that needs to be controlled. It has on/off piston actuators on both ends. If pressure is applied at the top of the cylinder it moves downward and the opposite if pressure is applied at the bottom. An ultrasonic position transmitter/sensor (not labelled in the panel layout), as seen on the bottom right of figure 3.1, provides the position of the cylinder over 4-20 mA to PLC input. The full-scale measurement range of the sensor is 40 – 300 mm corresponding to 2 – 62 units on the ClickPLC software (4.333 mm = 1 unit). There is around a 250 ms measurement latency from the position sensor. The pressure sensor measures the pressure. The proportional valve or flow control valve controls the flow rate of the air actuating the piston. The pushbutton enclosure is used to manually start and stop the system and it also indicates if the system is running.

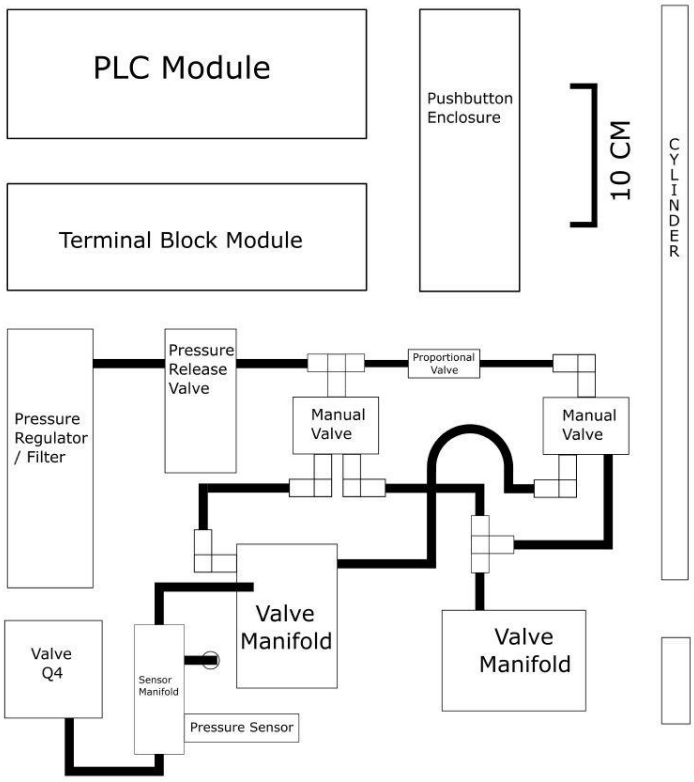



*Figure 3.2: Panel Layout*

Another important drawing worth looking at to better understand how the automation kit works is the P&ID drawing as shown in figure 3.3 and table 3.1 which provides the key component names for P&ID abbreviations. LT-201 is an ultrasonic position transmitter and was discussed above. LS-201 is a limit switch connected to PLC input and turns on when the cylinder is fully extended. ZC-201 is a pneumatic valve which when open, flows air into the top of the piston cylinder causing it to extend and move down. It is connected to PLC output. ZC-202 is a pneumatic valve which when open, flows air into the bottom of the piston cylinder causing it to retract and go up. Both valves are connected to PLC outputs. FCV-101 is the Proportional/flow control valve. ZS-101 and ZS-102 are the manual valves. ZC-301 and ZC-302 are the valve manifolds. PT-301 is the pressure sensor. HC-101 is the pressure release valve. PC-101 is the pressure regulator or filter. PC-201 is a pressure reducing valve. If inlet pressure at PC-201 drops too low then no air will pass through the valve.

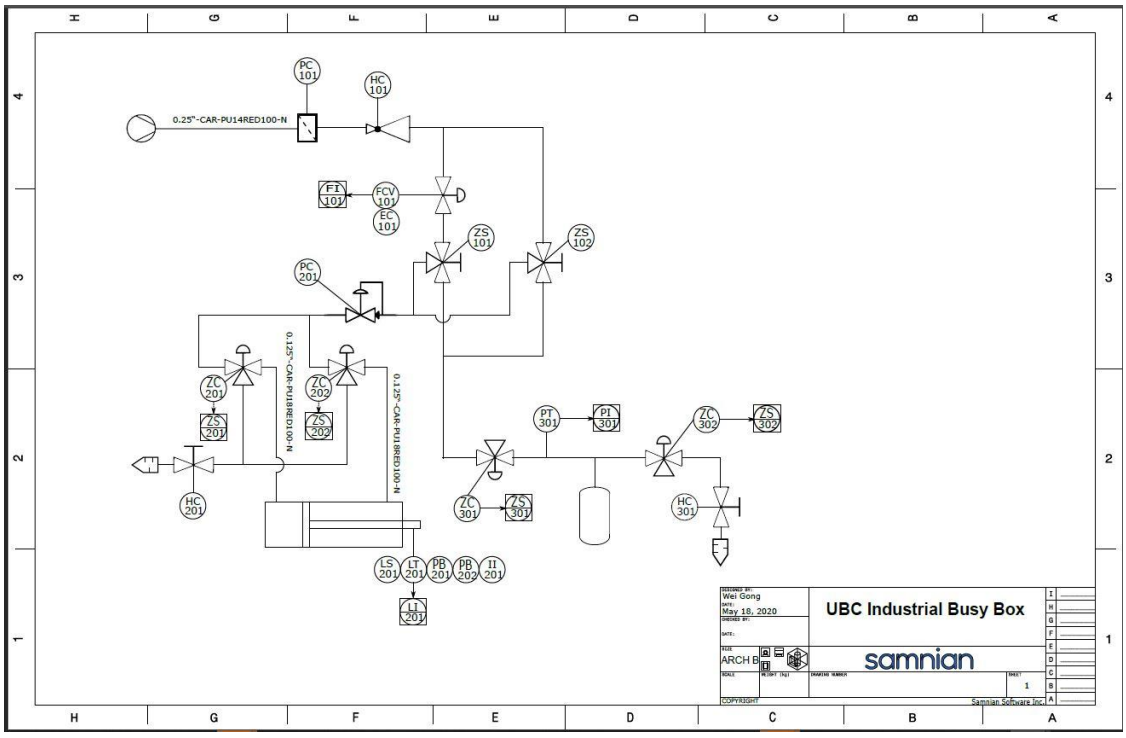



*Figure 3.3: Formal P&ID*

*Table 3.1: P&ID Abbreviation Definitions*

| **P&ID Abbreviation** | **Component** |
|---|---|
| EC-101 | Voltage controller |
| FCV-101 | Proportional/flow control valve |
| FI-101 | Proportional/flow control indicator |
| HC-101 | Pressure release valve |
| HC-201 | Pneumatic valve controller |
| HC-301 | Valve Q4 controller |
| II-201 | Dimmable LED lamp |
| LI-201 | Ultrasonic position indicator |
| LS-201 | Limit switch |
| LT-201 | Ultrasonic position transmitter |
| PB-201 | Green pushbutton |
| PB-202 | Red pushbutton |
| PC-101 | Pressure regulator/filter |
| PC-201 | Pressure reducing valve |
| PI-301 | Pressure indicator |
| PT-301 | Pressure sensor |
| ZC-201 | Pneumatic valve (top of piston) |
| ZC-202 | Pneumatic valve (bottom of piston) |
| ZC-301 | Valve manifold |
| ZC-302 | Valve manifold |
| ZS-101 | Manual valve |
| ZS-102 | Manual valve |
| ZS-201 | Pneumatic valve switch (top of piston) |
| ZS-202 | Pneumatic valve switch (bottom of piston) |
| ZS-301 | Valve manifold switch |
| ZS-302 | Valve manifold switch |

Panel drawings can be found in Appendix B. These give additional information on the power supply and plc chassis layout, terminal block layout, power distribution, and analog and digital I/O wiring.

### 3.2. Built-in I/O Setup

The I/O must first be setup on the ClickPLC software. Figure 3.4 shows the I/O available on this PLC model. The address names for each input and output is also given. The I/O addresses are connected to components of the automation kit for inputting and outputting actions.

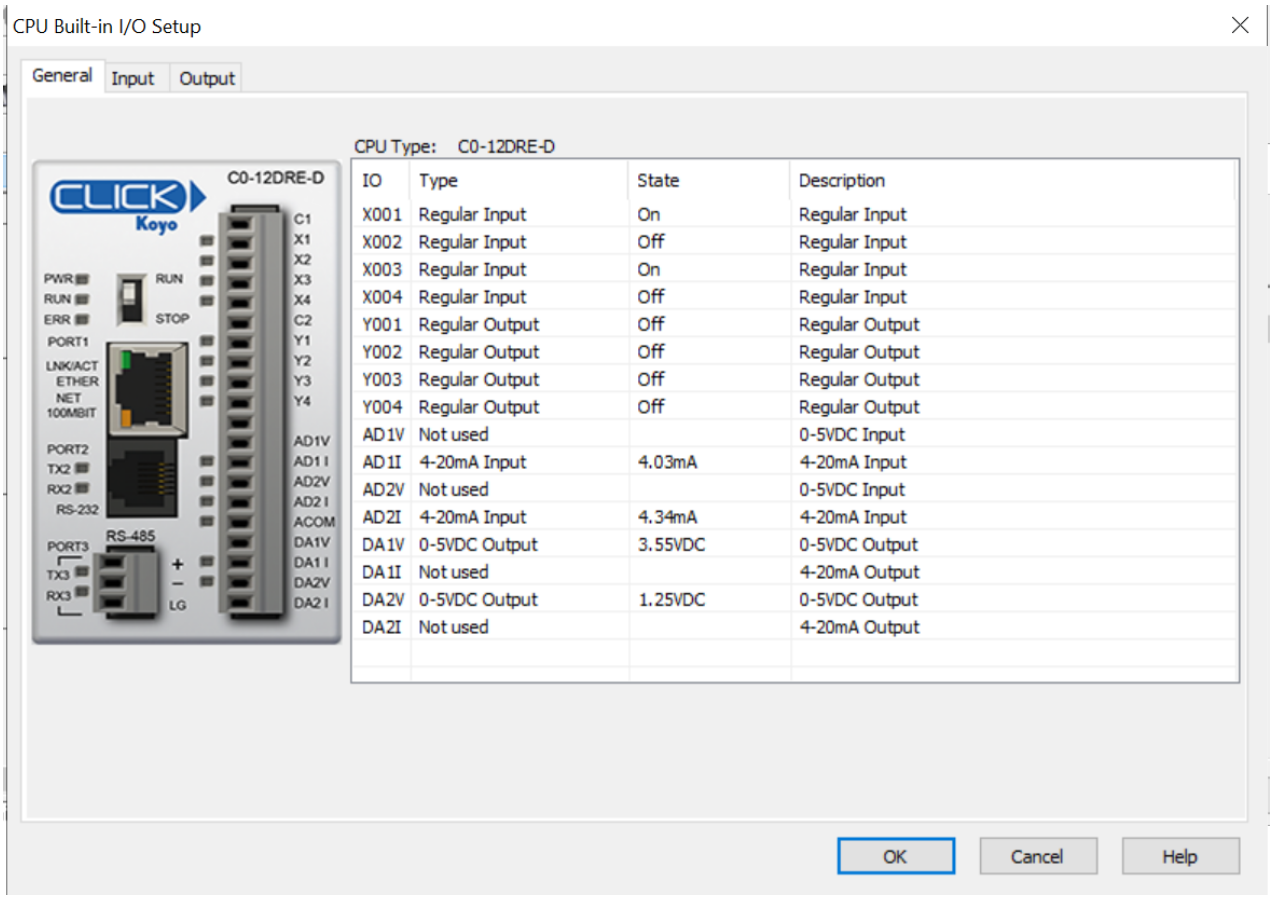



*Figure 3.4: General CPU Built-in I/O Setup*

There are 6 Built-in Input selections, figure 3.5, but only a few of these are important. The green pushbutton or operator button (PB-201) is connected to discrete input X002. The pressure sensor (PT-301) is connected to input AD1I which in turn is assigned to data register DF1. The ultrasonic position transmitter/sensor (LT-201) is connected to input AD2I which in turn is assigned to data register DF2.

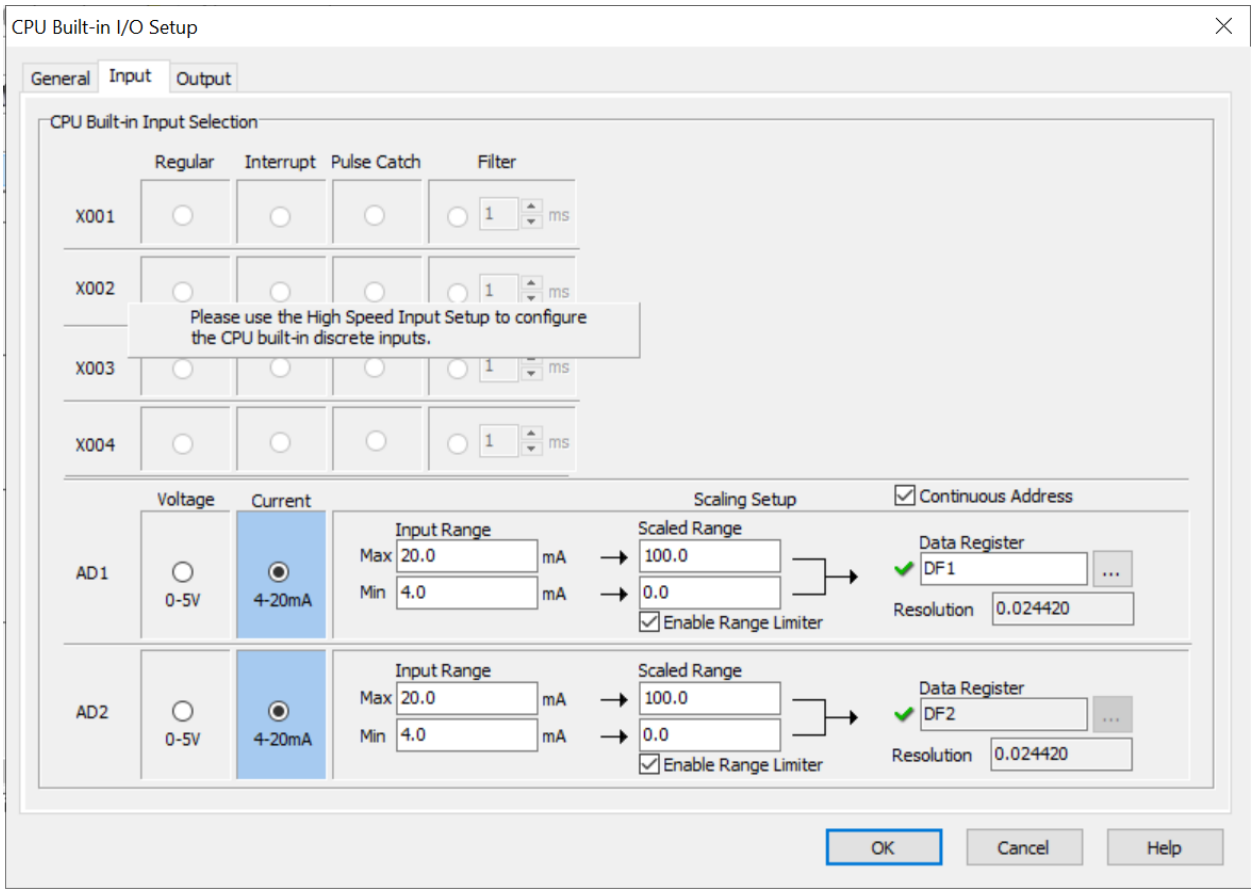



*Figure 3.5: CPU Built-in Input Selection*

There are 6 Built-in output selections, figure 3.6, but only a few of these are important. The pneumatic valve (ZC-201) which when opened causes the piston to extend downwards is connected to discrete output Y001. The pneumatic valve (ZC-202) which when opened causes the piston to extend downwards is connected to discrete output Y002. Proportional/flow control valve (FCV-101) is connected to analog output DA1V which in turn is assigned to data register DF3. Dimmable LED lamp (II-201) is connected to analog output DA2V which in turn is assigned to data register DF4.

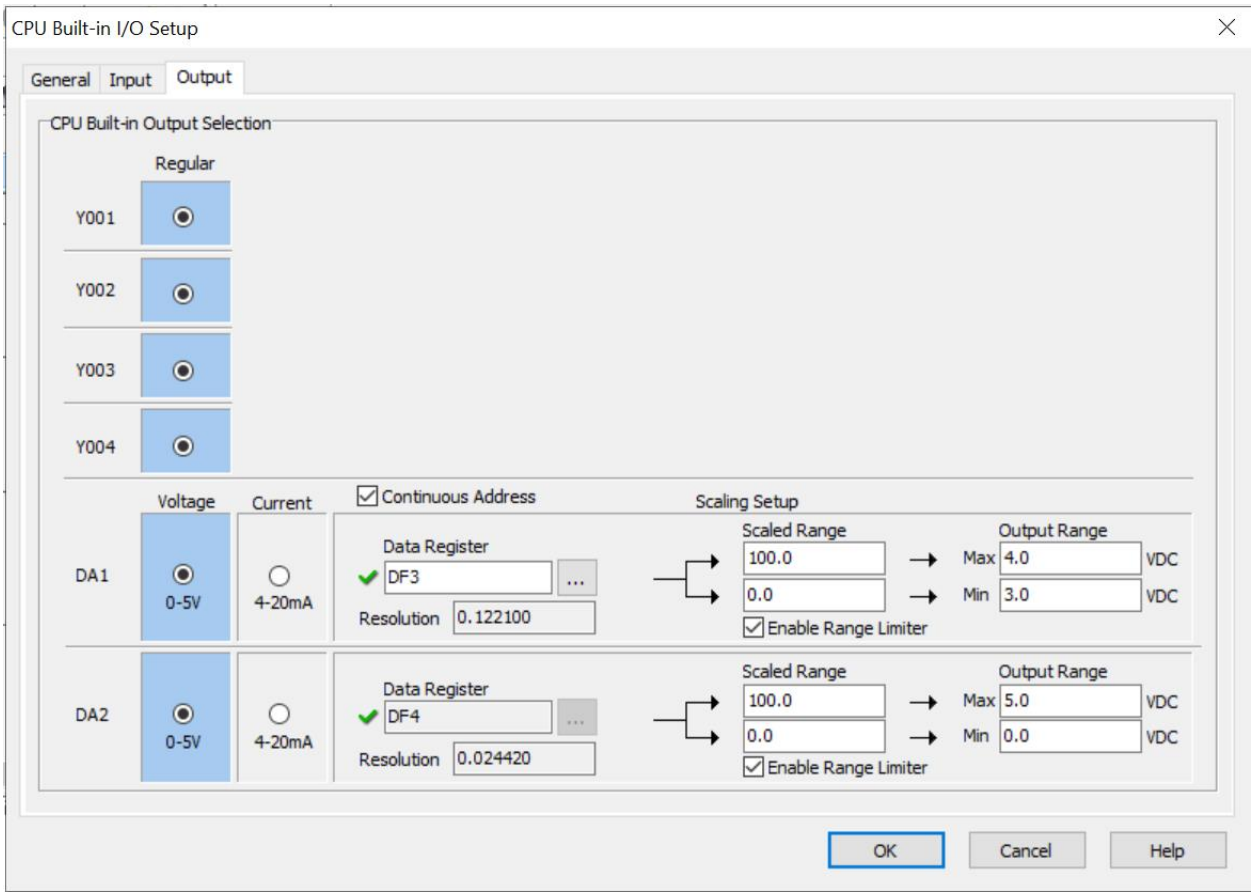



*Figure 3.6: CPU Built-in Output Selection*

### 3.3. Control Logic Design and Implementation

A ladder logic was first written for to automatically control the piston when the automation kit was oriented horizontally, to eliminate the effects of gravity. The program will later be adapted for the automation kit oriented vertically with the effect of gravity present. The complete ladder logic program is given in Appendix B.

The first step was to program when the pneumatic valves output (Y001 and Y002) open and close as shown in figure 3.7. Both the extension and retraction valve can be both closed or only one can be open at a time. Both pneumatic valves should not be open at the same time. The ladder logic is shown below.  C stands for control relays. C103 allows the user to manually start and stop the entire program by controlling if the valves should be open. When C100 (piston down) is toggled on and C101 (piston up) is toggled off, Y001 is open and Y002 is closed forcing the piston down. The opposite is true when C101 is toggled on and C100 is toggled off. When C100 and C101 are both toggled off, both Y001 and Y002 are closed and there will be no air pushing on the piston.

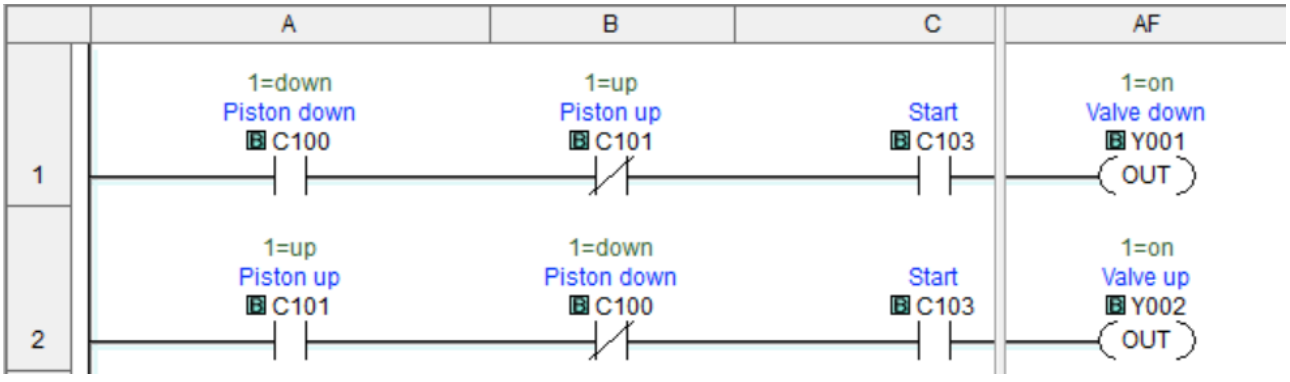



*Figure 3.7: Pneumatic Valve Operation Logic*

One of the pneumatic valves needs to be operation when the piston position is outside the target tolerance and both pneumatic valves need to be closed when the piston is within the target tolerance. The logic is given figure 3.8 below. The distance from the target is given by the absolute value difference between DF5 (target position) and DF2 (piston position). If the distance from target from the target is greater than the target tolerance, then one of the pneumatic valve will be toggled on and so C102 is toggled on to activate/turn on the system.

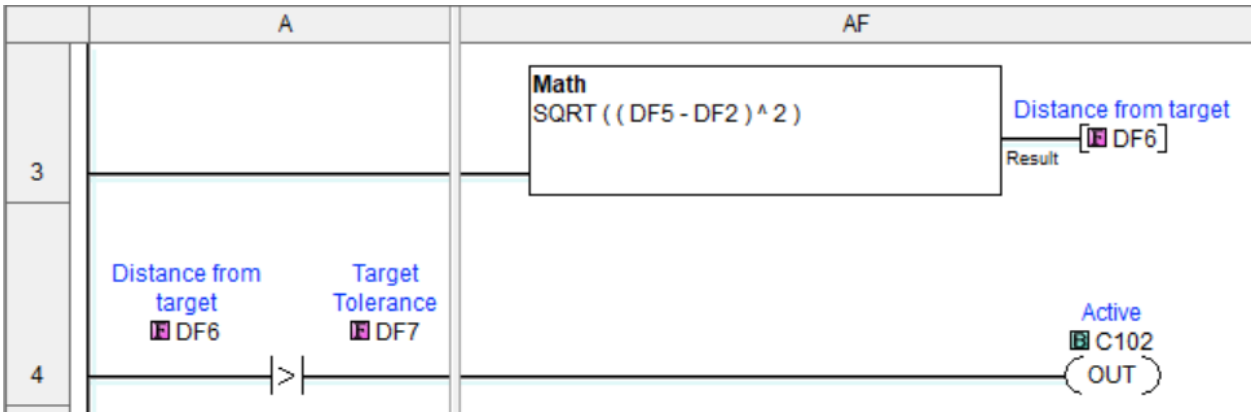



*Figure 3.8: Position of piston and Target Tolerance*

The timing was next to be dealt with. The pneumatic valves left to its own device will fully open and close in 2-3 seconds which is too long. The piston position would move too far in amount of time and it will be difficult to get the piston position to settle around the target position. To solve this issue, very precise timings are imposed on the pneumatic valves according to the logic in figure 3.9 below. On rung 5, A 10 ms clock (SC4) is used along with a counter that has a counter number of 4. As the counter counts to 4, which takes a time of 40 ms to complete, CT1 (counter high) is off, this triggers rungs 9 and 10 in figure 3.11 which require a CT1 to be false. This in turn causes the piston to move up or down. Hence CT1 is called counter high because one of the valves is open. When the current counter number is 4, CT1 is turned on. These triggers rung 7 which operates in the same way as rung 5. While CT2 is counts up, CT2 is in an off state and CT1 is in

an on state and both pneumatic valves are not operational for 40ms. Hence CT2 is called counter low because both valves closed. This process of turning the system on for 40ms then off for another 40ms, resembles a square wave and the timing diagram is shown in figure 3.10. When the counter number of CT2 reaches 4, CT2 turns on. CT1 and CT2 toggling from off to on after counting to 4 is also necessary to reset the counters for the next pneumatic valve action. Both counters are reset in rungs 6 and rungs 8. CT1 and CT2 in rungs 6 and 8 are rising edge contact, which turns on when the bit is turned from off to on. CT1 toggling on triggers C105 (counter low reset) and is used to reset CT2 before it starts counting again. Likewise, CT2 toggling on triggers C104 (counter high reset) and is used to reset CT1 before it starts counting again.

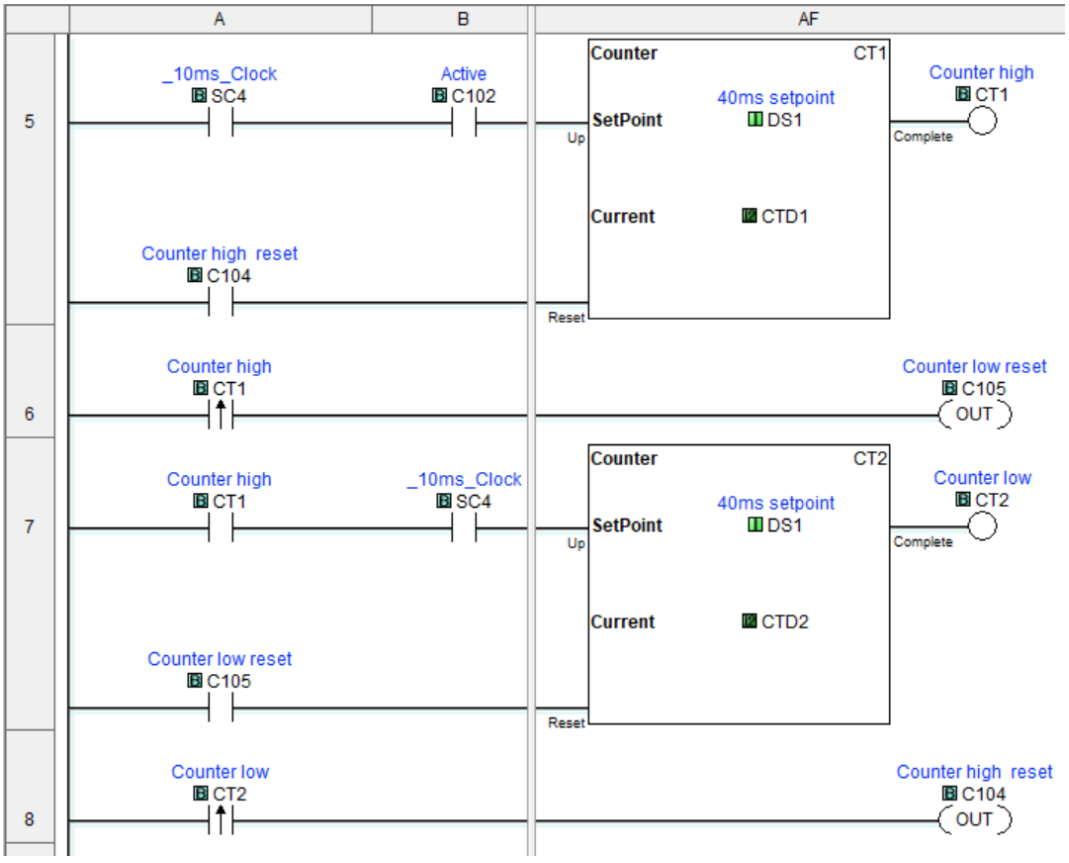



*Figure 3.9: Counter Timing Logic*

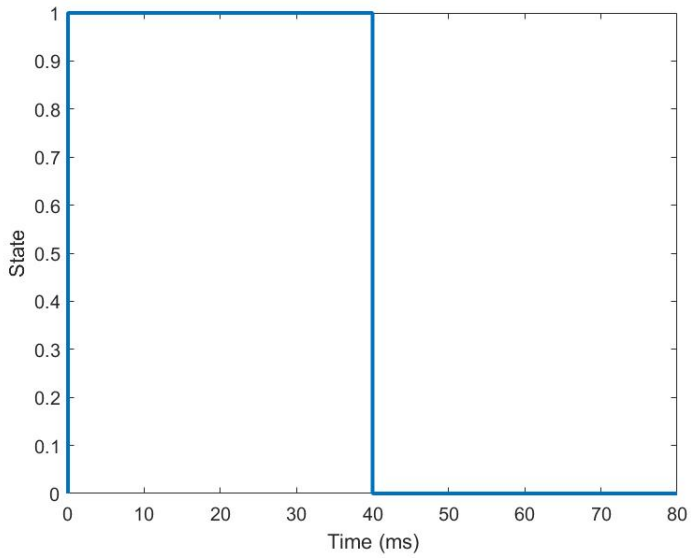



*Figure 3.10: Counter Timing diagram*

Finally, the last 2 rungs, shown in figure 3.11, dictate if the piston goes up or down. While counter CT1 is counting, the counter is in an off state so rung 9 and 10 are triggered. C102 is also in an on state due to piston position not being within tolerance. If DF2 (piston position) is greater than DF5 (target position), C100 is triggered, and the piston moves down. The opposite is true if DF2 is less than DF5.

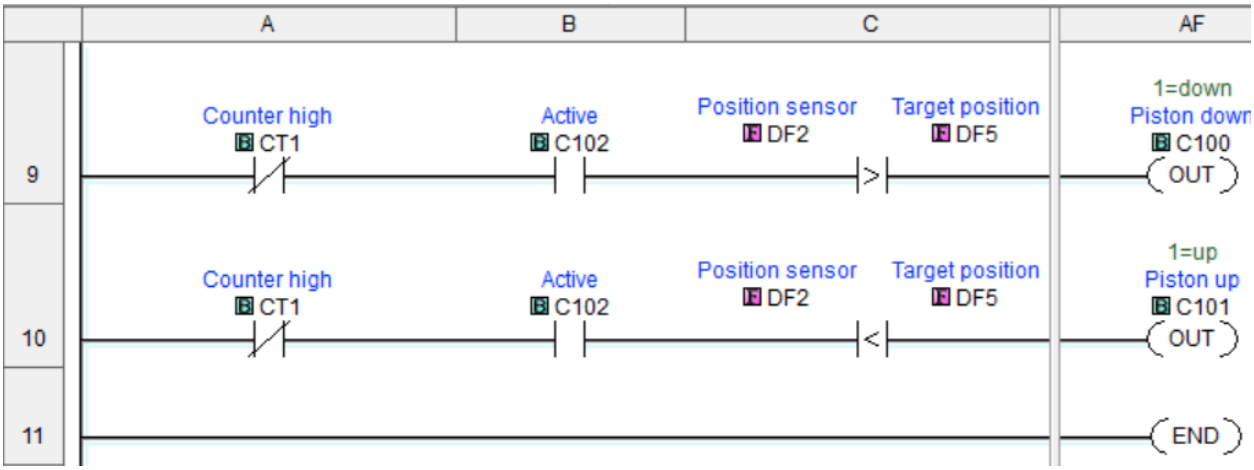
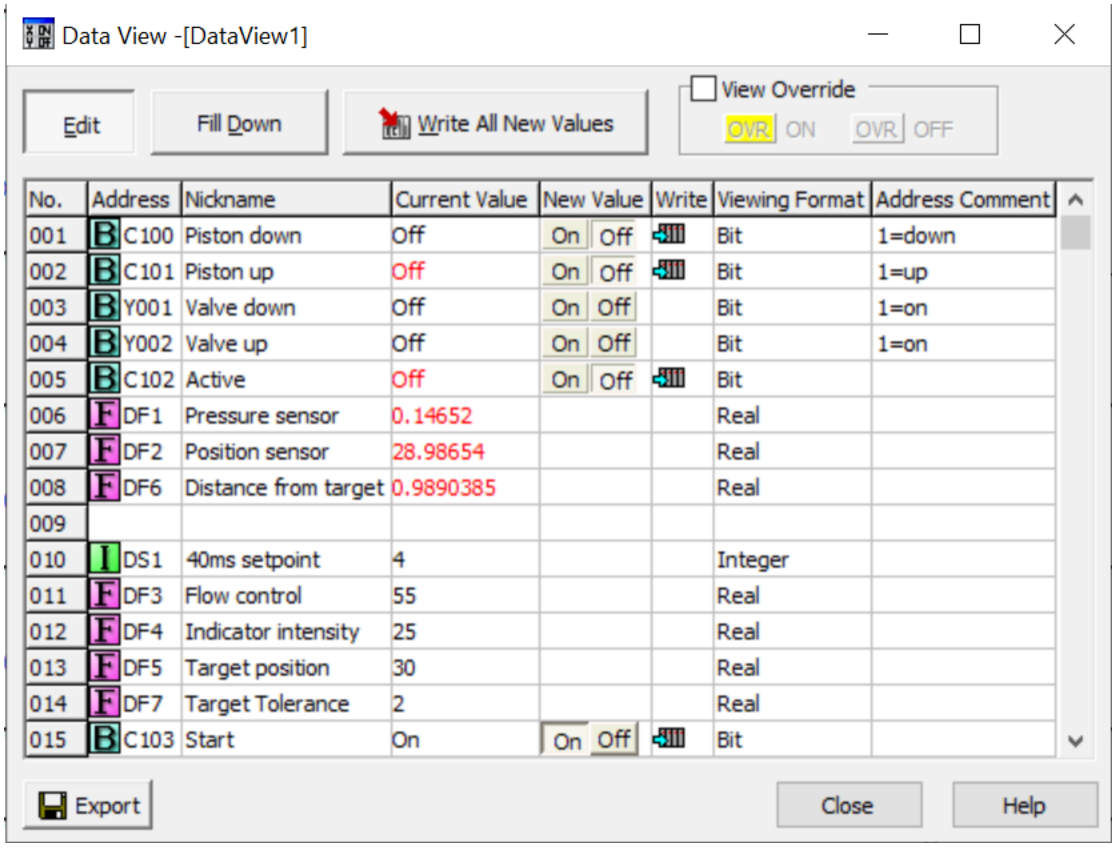



*Figure 3.11: Piston Control Logic*

Data View, figure 3.12, on ClickPLC is used for monitoring and changing the values of the addresses. Address 1-8 are addresses that are monitored while addresses 10-15 are addresses that can be changed and controlled directly by the user. DS1, counter setpoint was chosen to be 4, corresponding to a 40 ms counter time. DF3, flow control, was set to 55% of the valve is closed, therefore only 45% of the maximum flow rate flows through the pneumatic valves. DF4 is the indicator intensity. The indicator informs the students if the system is running or not. The higher the indicator intensity, the brighter the indicator. DF7, target tolerance, is set to 2 (+/- 8.666 mm). DF5, the target position, should be chosen by the student to be between 2 – 62 which corresponds to 40 – 300 mm. C103 is used to manually start/stop the system. All the value that are set are taking into account an automation kit that has been positioned vertically.




| No. | Address | Nickname | Current Value | New Value | Write | Viewing Format | Address Comment |
|---|---|---|---|---|---|---|---|
| 001 | B C100 | Piston down | Off | On Off | | Bit | 1=down |
| 002 | B C101 | Piston up | Off | On Off | | Bit | 1=up |
| 003 | B Y001 | Valve down | Off | On Off | | Bit | 1=on |
| 004 | B Y002 | Valve up | Off | On Off | | Bit | 1=on |
| 005 | B C102 | Active | Off | On Off | | Bit | |
| 006 | F DF1 | Pressure sensor | 0.14652 | | | Real | |
| 007 | F DF2 | Position sensor | 28.98654 | | | Real | |
| 008 | F DF6 | Distance from target | 0.9890385 | | | Real | |
| 009 | | | | | | | |
| 010 | I DS1 | 40ms setpoint | 4 | | | Integer | |
| 011 | F DF3 | Flow control | 55 | | | Real | |
| 012 | F DF4 | Indicator intensity | 25 | | | Real | |
| 013 | F DF5 | Target position | 30 | | | Real | |
| 014 | F DF7 | Target Tolerance | 2 | | | Real | |
| 015 | B C103 | Start | On | On Off | | Bit | |

*Figure 3.12: Data View*

Vertical Assembly

The PLC logic for the automation kit in a vertical position is the same as for a horizontal position, although the value set in data view are different and are shown in figure 3.12. DS1, the counter number, was reduced from 5 to 4. The flow control was increased by reducing DF3 from 65 to 55. Finally, the target tolerance was kept at 2, because the amount of movement of the piston due to gravity should be kept at a minimum as it drops down before the bottom valve is triggers and pushes it back up. This unfortunately causes an oscillating behavior but keeps the piston very close to the target setpoint.

## 4. Project C: Manufacturing Equipment Data Harvesting

The last project involves finding means of data harvesting/acquisition and listing useful data from the manufacturing equipment in the UBC Frank Forward 219 Lab. In particular, the interest was on performance and diagnostic data. The goal is to put together a list of communication interfaces, useful data, individual data addresses and other important information that can be used to communicate and retrieve data from the equipment. These data will then be gathered, monitored, analyzed and controlled using a SCADA system to deal with critical real time issues of the equipment. This project is just the first stage in building an ERP (Enterprise Resource Planning) / MES (Manufacturing Execution System)/PLM(Planning Lifecycle Manufacturing Management), as shown in figure 4.1, for MANU. It will, in the future, be extended to include all the manufacturing lab equipment in UBC Vancouver and Okanagan campuses. This will help to manage lab classes like companies, create new courses with data analytics focus and it will help to bridge many MANU courses.

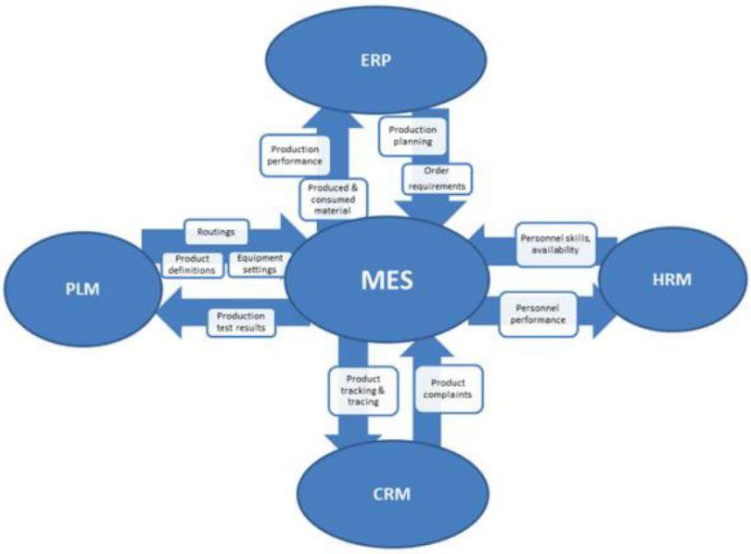



*Figure 4.1: Manufacturing Execution System [2]*

There are seven equipment that will be investigated in this project involving data harvesting and acquisition. The equipment of interest are summarized in table 4.1. Some of the equipment act solely as controllers for other equipment.

The injection moulding machine is used for manufacturing plastic products using an injection moulding process. The microprocessor-based temperature process control is used as a controller for the Thermolyne Furnatrol muffle furnace. The furnace is used for melting down of materials in a controlled and isolated way. The load frame is used to perform tensile and compression test. The 3D printer is an industrial grade 3D printer. The automation kit was already discussed in section 3. It is used for automating a pneumatic piston using 2 pneumatic valves. The Haas CNC controller is the controller that is used to operate the Haas CNC Machine. The temperature and process controller controls a standard mechanical convection oven.

*Table 4.1: Data Harvesting Equipment of Interest*

| **Equipment of interest** | **Model** | ***(Equipment Being Controlled)*** |
|---|---|---|
| Injection Moulding Machine (Procan Alpha 2 Control System) | BOY 22 A PRO | N/A |
| Microprocessor Based Temperature Process Control | Omega CN7200 | Thermolyne Furnatrol II Muffle Furnace (F-A1740-1) |
| Load Frame | Instron 3369 | N/A |
| 3D Printer | Raise3D Pro2 Plus | N/A |
| Automation Kit | Samnian BB-01 | N/A |
| Haas CNC Mill (Controller and Simulator) | Haas | N/A |
| Temperature and Process Controller | Watlow F4T | Blue M Standard Mechanical Convection Oven (M146 Series, Model DC-1406) |

To show the implementation of how SCADA can be used to gather and monitor the equipment, a further SCADA application and UI will also be created for the injection moulding machine. This will also provide a UI (User Interface) for the operator.

### 4.1. Infrastructure

The automation pyramid is shown in figure 4.2. The equipment are on the bottom level and these are controlled using a PLC/PAC. A SCADA software is connected to the PLC/PAC and is used to supervise the equipment. The MES manages everything and is used to document the manufacturing history. At the very top is the ERP used for inventory management. Aside from MES, PLM, and instrumentation and IoT database, there are other additional requirements for a successful ERP system, and these includes SCM (Supply Chain Management), and CRM (Client Relationship Management).

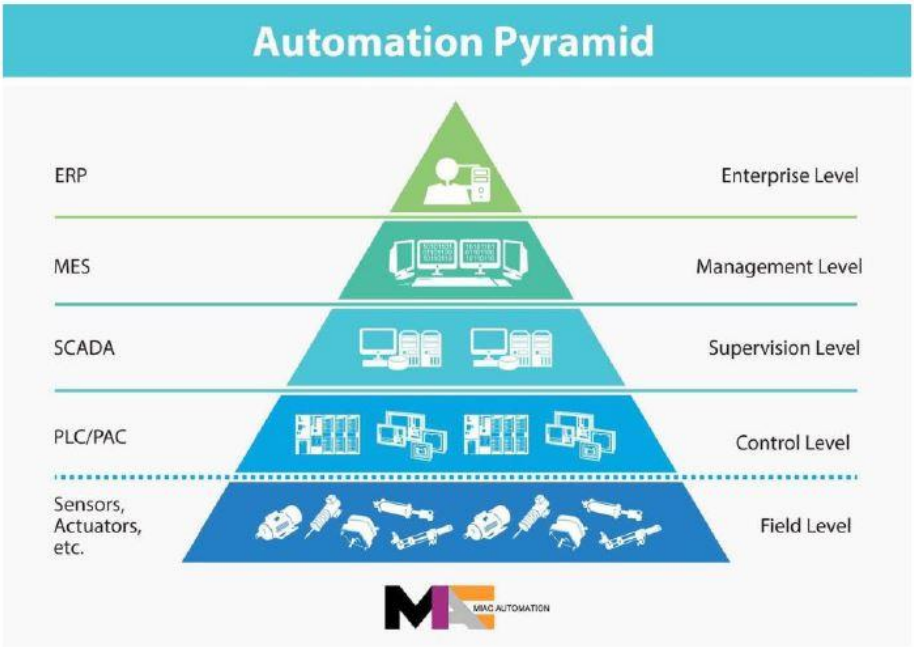



*Figure 4.2: Automation Pyramid [3]*

The proposed UBC MANU infrastructure in shown in figure 4.3 below. The proposed data server for all seven equipment is shown in table 4.2 below. Scada Historian is the data server for most of the equipment and the SCADA Historian of choice is VTSCADA.

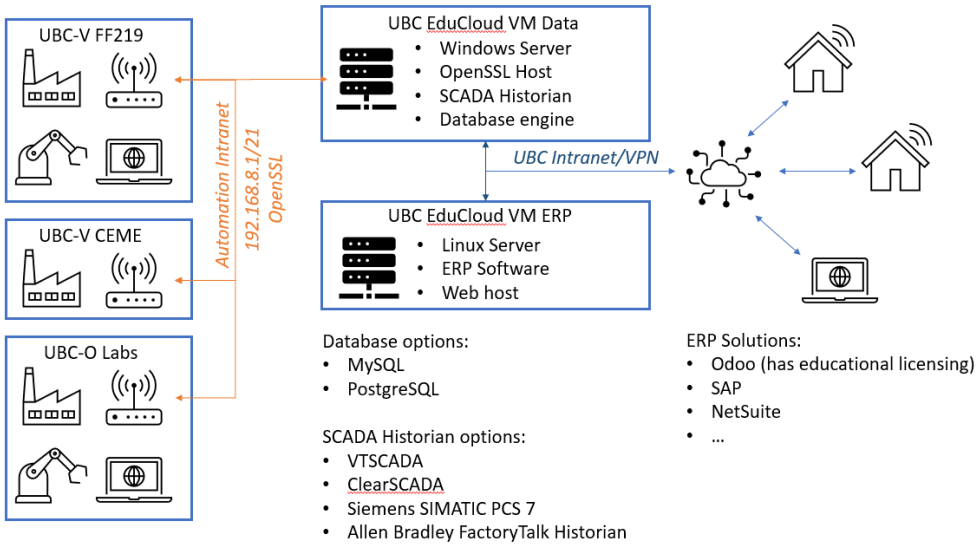



*Figure 4.3: MANU Infrastructure*

*Table 4.2: UBC-V Equipment Data Servers*

| **Equipment** | **Data Server** |
|---|---|
| Injection Moulding Machine (Procan Alpha 2 Control System) | SCADA Historian |
| Microprocessor Based Temperature Process Control | MOXA MGate 5217 / SCADA Historian |
| Load Frame | Data Server |
| 3D Printer | Data Server (image store) |
| Automation Kit | SCADA Historian |
| Haas CNC Mill (Controller and Simulator) | Ethernet / Data Server |
| Temperature and Process Controller | SCADA Historian |

### 4.2. BOY Injection Moulding Machine

The BOY 22 A PRO Injection Moulding Machine, figure 4.4, is used for manufacturing plastic products through means of an injection moulding process. The equipment takes in raw plastic pellets through the feed hopper at the top. When the pellets have been melted, the screw pushes the molten plastic through the nozzle and into the mould at a very high pressure and the completed piece is then ejected.


*Figure 4.4: Injection Moulding Machine*

#### 4.2.1. Communication Technology and Tags

The communication interface for the injection moulding machine is Ethernet OPC (Unified Architecture). OPC is used for machine-to-machine communication primarily for industrial automation. OPC communication is possible by connecting a PC to the injection moulding machine using an ethernet cable.

The internet protocol version 4 (TCP/IPv4) properties, which can be found in the ethernet connection properties should be changed to match the network configuration on the Procan Alpha 2 system (14.21 Machine data, basic setup). The last set of numbers of the IP Address should be 1 number less when entered into the IPv4 properties. Figure 4.5 demonstrates this idea.

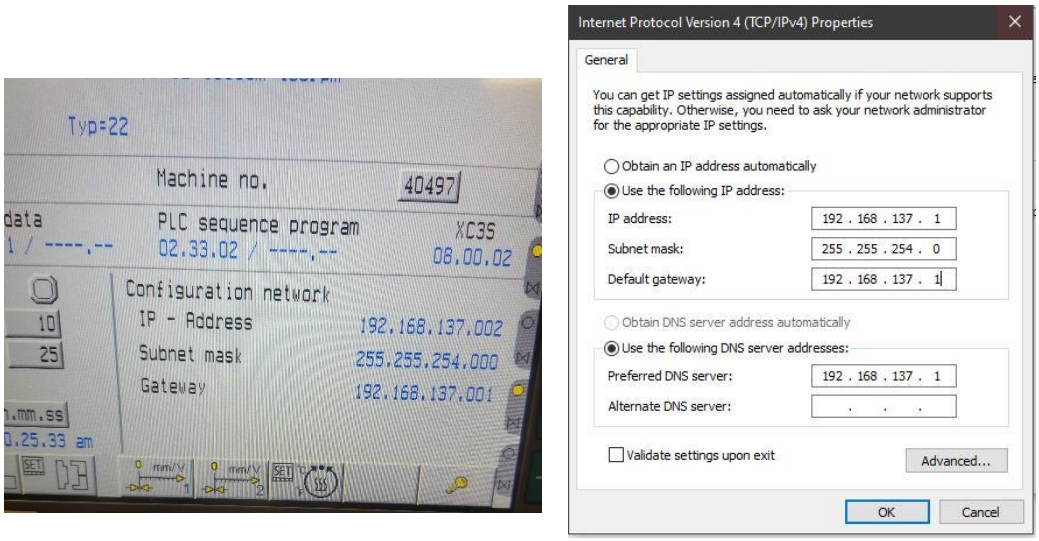



*Figure 4.5: Setting up Ethernet Connection*

To access all the OPC variable of Procan Alpha 2 control system, KEPServerEX software by Kepware with CODESYS V2.3 Ethernet driver must be used. KEPServerEX is a connectivity platform that leverages OPC and allows the users to connect, manage, monitor and control different tags from various machine through one interface. CODESYS driver paired with KEPServerEX allows the user to connect to operations that use CODESYS, which PLC runtime and development environment.

After downloading KEPServerEX and CODESYS, a new project needs to be created on the KEPServerEX platform. The first step is to add a new channel, under “Connectivity”, to the project as shown in figure 4.6. The channel must be given a name, in this case it was named “Channel 1”, and the CODESYS driver must be selected. All other settings should stay as default.

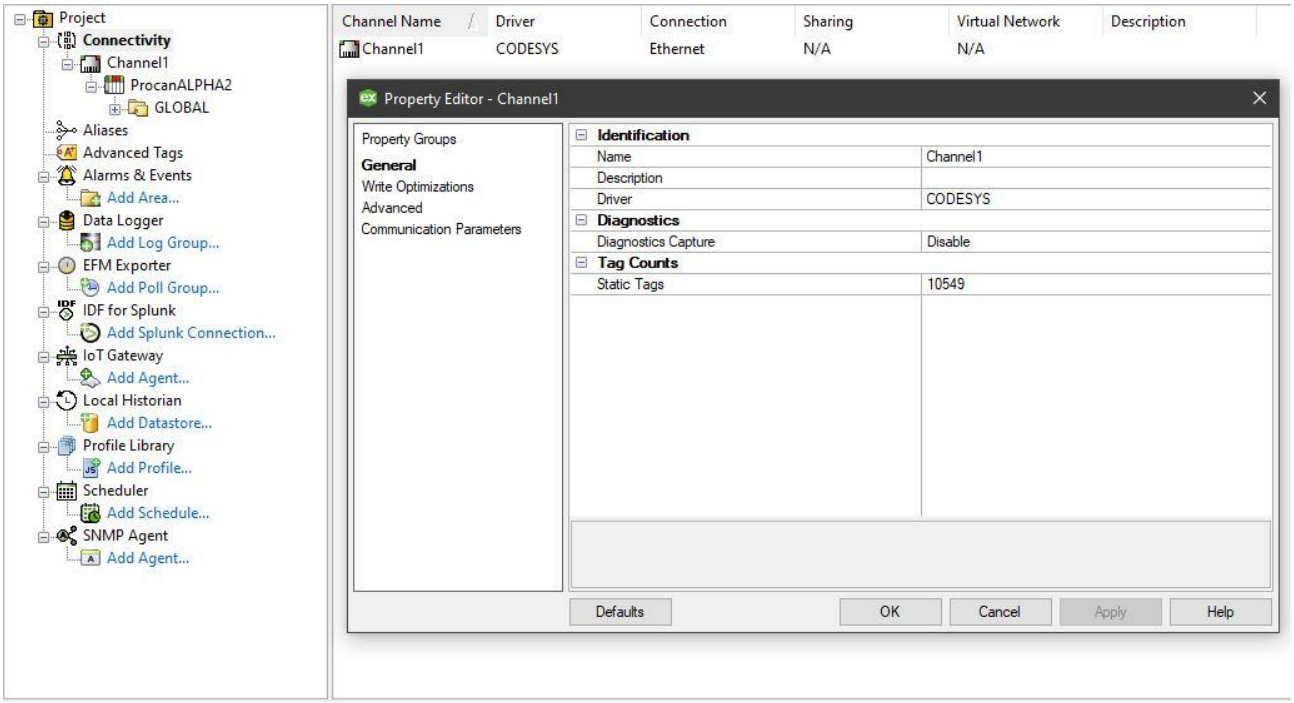



*Figure 4.6: Adding a Channel*

The next step is to add a new device to the created channel as shown in figure 4.7. The device is the Procan Alpha 2 control system. The driver model is CODESYS V2.3 Ethernet. The IP address should be the same as those on the Procan Alpha 2 system. All other settings should stay as default. After the device is created, in the device properties (Tag Generation tab), click on “Create Tags”.

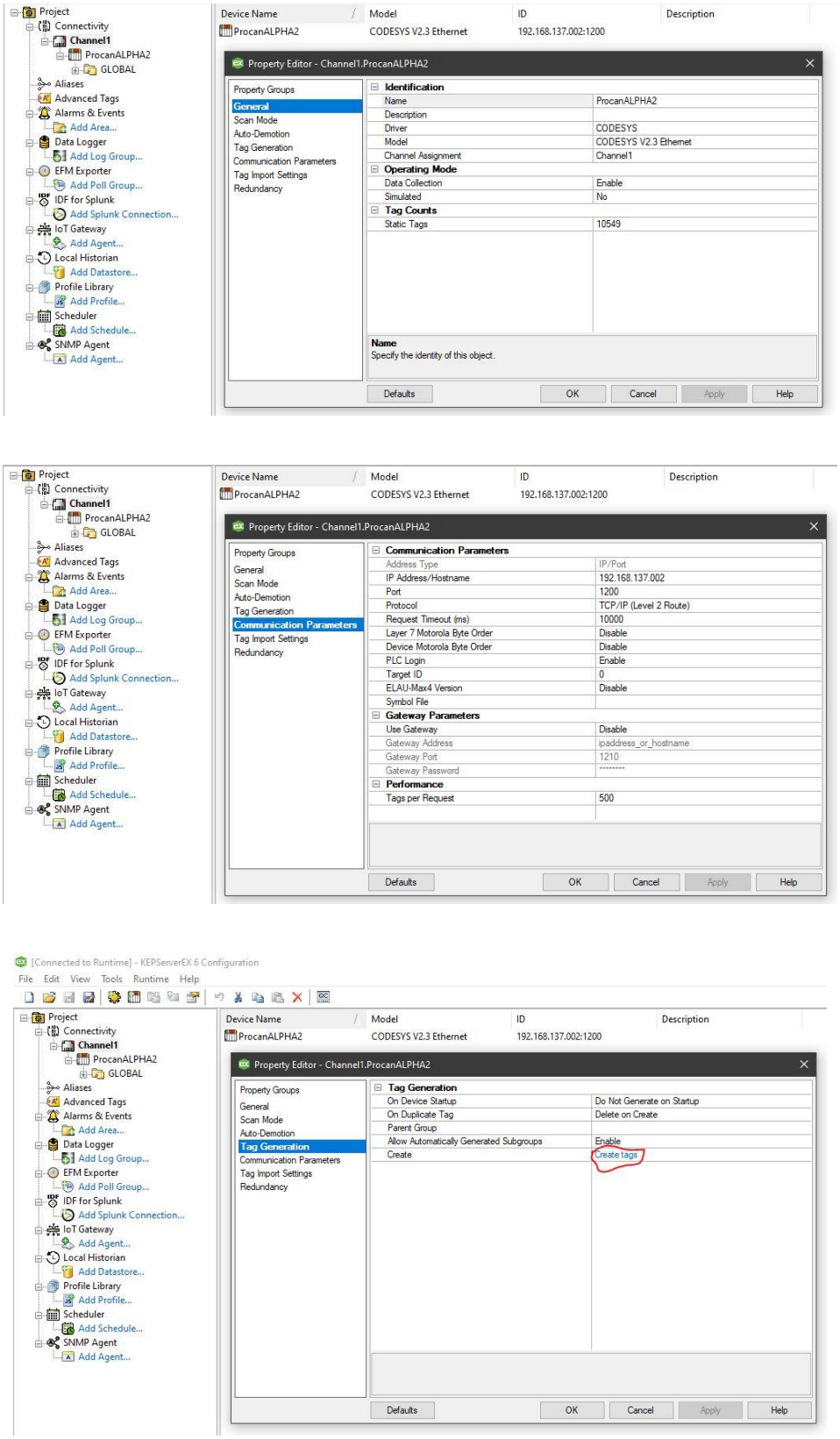



*Figure 4.7: Add Device and Generate Tags*

All the tags (variables) available for the injection moulding will be generated and is viewed using OPC Quick Client (QC) in the GLOBAL section as shown in figure 4.8 and it is now a matter of choosing the tags relevant to performance and diagnostics.

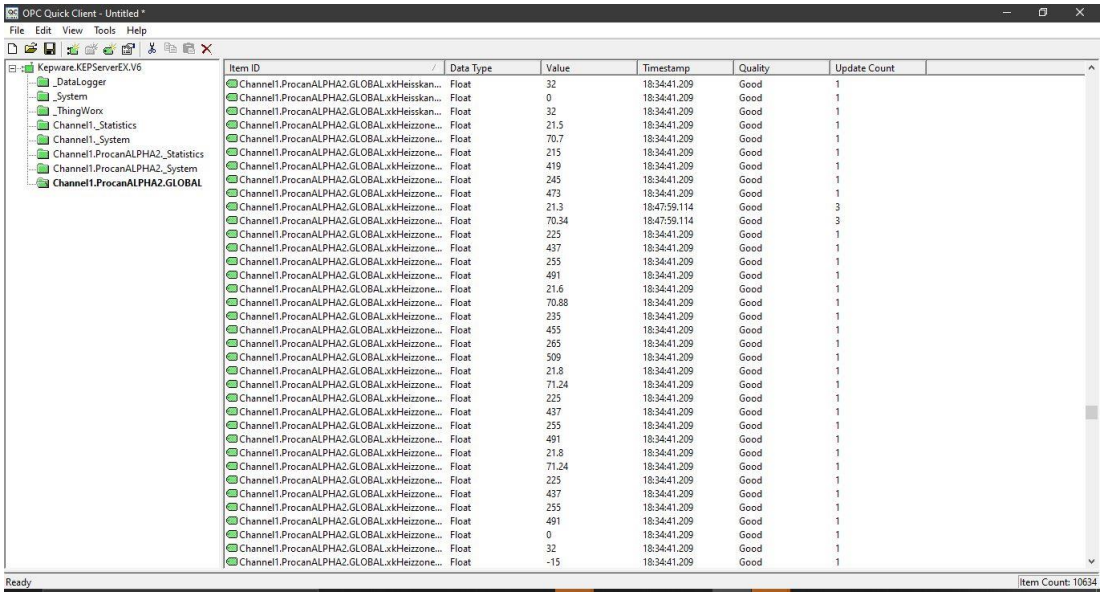



*Figure 4.8: All Available Tags (Variables) for Injection Moulding Machine*

The Item ID is the address of a tag and it is in the given format:

"Custom_Channel_Name"."Custom_Device_Name".GLOBAL."Tag Name"

The tag/variable name can be interpreted using a variable definition spreadsheet attached provided by the company. A list of chosen tags, showing the sensor endpoint, instrumentation data sources, addresses, units, engineering units and conversion is shown in the table 4.3 below.

*Table 4.3: List of Tags for Injection Moulding Machine*

| **Sensor Endpoint** | **Instrumentation Data Sources** | **Addresses (just the tag name)** | **Units** | **Conversions** | **Engineering Units** |
|---|---|---|---|---|---|
| **Communication Interface:** Ethernet OPC via KEPServerEX | | | | | |
| Clamping Force | Pressure sensor B30D | .xFWkzSchliessen_kN | kN | None | kN |
| Max Injection Pressure | Pressure sensor | .xpSpritzMax_psi | psi | 1 psi = 6.89476 kPa | kPa |
| Feeding zone temperature | Thermocouple (Type J, FeCuNi) | .xkEinzugszone_C | deg C | None | deg C |
| Heating zone 1 temperature | Thermocouple (Type J, FeCuNi) | .xkHeizzone1_C | deg C | None | deg C |
| Heating zone 2 temperature | Thermocouple (Type J, FeCuNi) | .xkHeizzone2_C | deg C | None | deg C |
| Heating zone 3 temperature | Thermocouple (Type J, FeCuNi) | .xkHeizzone3_C | deg C | None | deg C |
| Heating zone 4 (Nozzle) temperature | Thermocouple (Type J, FeCuNi) | .xkHeizzone4_C | deg C | None | deg C |
| Heating zone 5 (Mould) temperature | Thermocouple (Type J, FeCuNi) | .xkHeizzone5_C | deg C | None | deg C |
| Ejector Position | Position sensor | .xsAuswerfer_mm | mm | None | mm |
| Screw Position | Position sensor | .xsSchnecke_mm | mm | None | mm |

| | | | | | |
|---|---|---|---|---|---|
| Screw speed | Speed sensor | .xvSchnecke_1min | 1 / min | 1 1/min = 1 rpm | rpm |
| Average Cycle Time | N/A | .xtZykluszeitMittelwert | sec | None | sec |
| Effectiveness | N/A | .xnEffektivitaet | % | None | % |
| Estimated Production Time | N/A | .xtProduktionsdauer | sec | None | sec |
| Material Throughput | N/A | .xmMaterialdurchsatz_kgh | kg/hr | 1 kg/hr = (1/60) kg/sec | kg |
| Poor Parts | N/A | .xnStueckzaehlerSchlecht | None | None | None |
| Total Parts | N/A | .xnStueckzaehlerGesamt | None | None | None |

The clamping force and max injection pressure tells the user how much pressure the mould is under. A higher pressure means a more accurate part. The feeding zone temperature and heating zone 1 – 5 temperatures give insight on the temperature at different point of the injection process. The feeding zone temperature has a set point of $50\ ^\circ C$, the heating zone 1 has a set point of $230\ ^\circ C$, the heating zones 2, 4 and 5 have a set point of $230\ ^\circ C$, the heating zone 3 has a set point of $250\ ^\circ C$. The tags will be monitored to make sure that they reach the target setpoint. The ejector position, and the screw position and speed are other useful data to keep track of. On the manufacturing side, we can keep track of the average cycle time, effectiveness of production, estimated production time based on past production time, and material throughput which tells us how much mould is being used over time or the rate of production. Lastly data is collected on the number of poor parts vs. the total number of parts.

### 4.2.2. SCADA Application

After establishing connection between the KEPServerEX platform and the PLC (Connected to Procan Alpha 2 system) of the injection moulding machine, and getting the useful tags, the next step is to set up a SCADA application. The SCADA Application will be connected to the KEPServerEX. The KEPServer EX serves as an OPC server. The overall connection scheme is shown in figure 4.9 below.

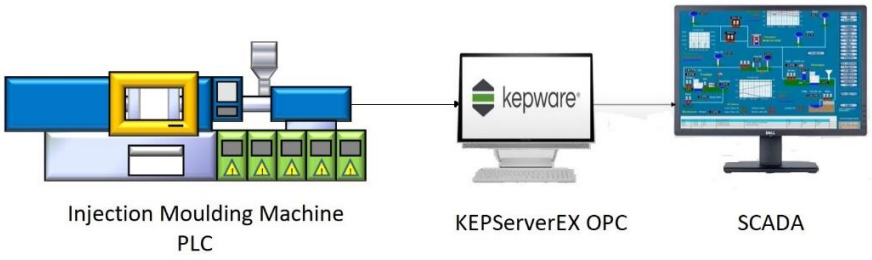



*Figure 4.9: SCADA and Injection Moulding PLC Communication via KEPServerEX OPC Server*

The SCADA Historian software that is used is VTSCADA. A TCP/IP Port is first created and the TCP/IP name/address is the IP address of the Procan Alpha 2 system and the TCP/IP port number is by default 1200 as shown in figure 4.10 below.

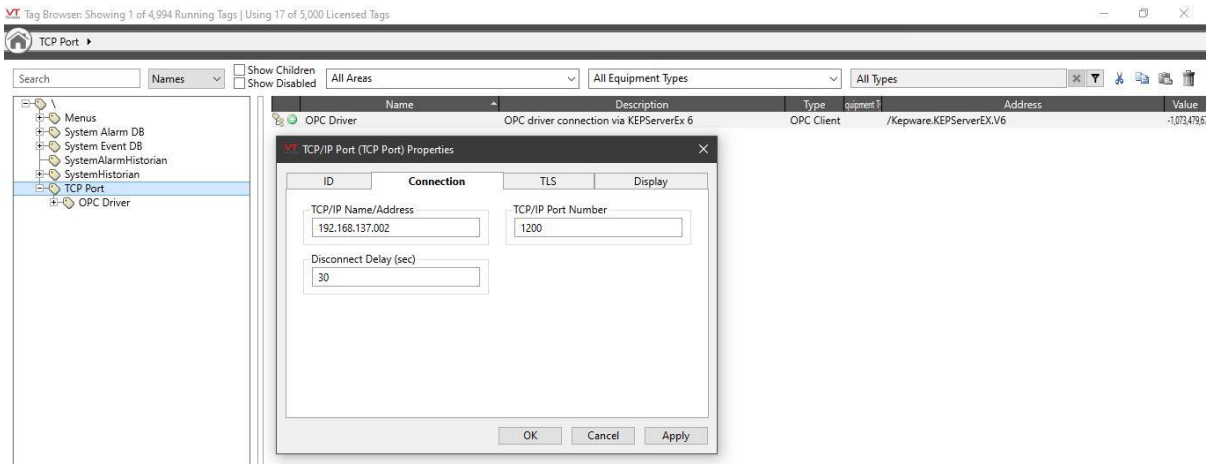



*Figure 4.10: TCP/IP Port Properties*

The driver is created under the TCP port, as shown in figure 4.11, and it is an OPC Client with the OPC Server name being “Kepware.KEPServerEX.V6”.

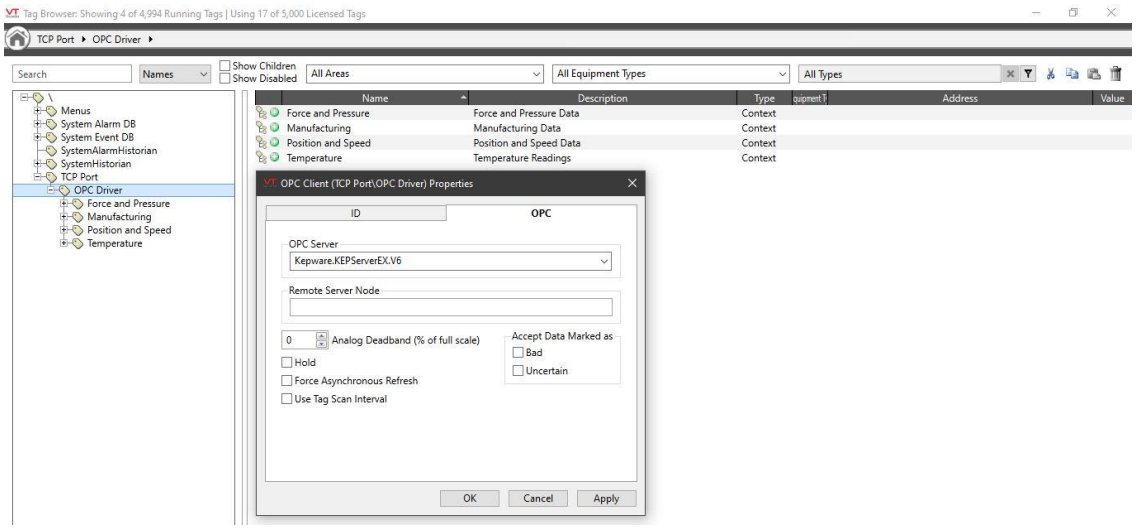



*Figure 4.11: OPC Client Driver Properties*

All the I/O tags chosen are analog I/O because all the chosen tags are of analog data type. The read address is the address of tag. Figure 4.12 below shows an example of the I/O tag for clamping force.

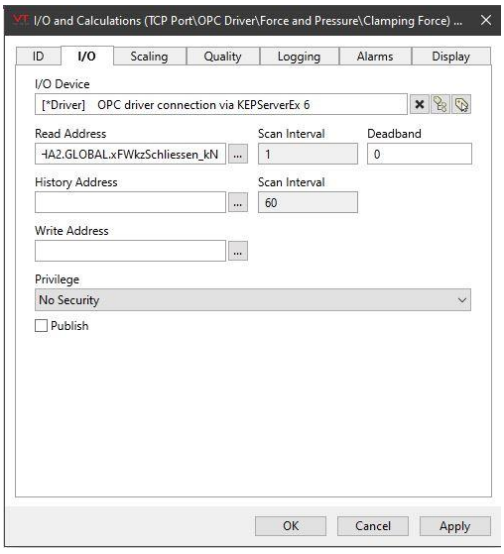



*Figure 4.12: I/O and Calculations Properties (Clamping Force)*

The VTSCADA tags and values while running is shown below in figures 4.13 – 4.16. I/O tags of different types are organized into 4 context tags, Force and Pressure, Manufacturing, Position and Speed, and Temperature.

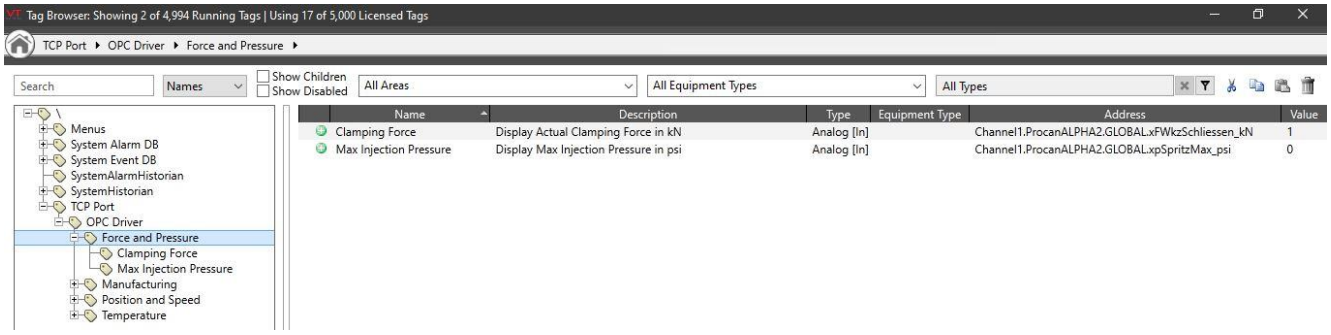



*Figure 4.13: Force and Pressure I/O Tags*

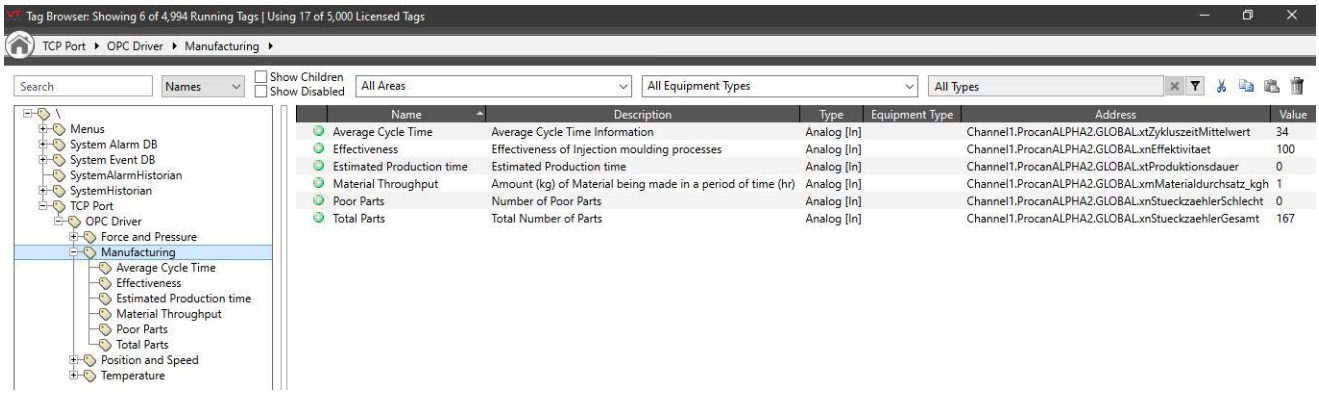



*Figure 4.14: Manufacturing Data I/O Tags*

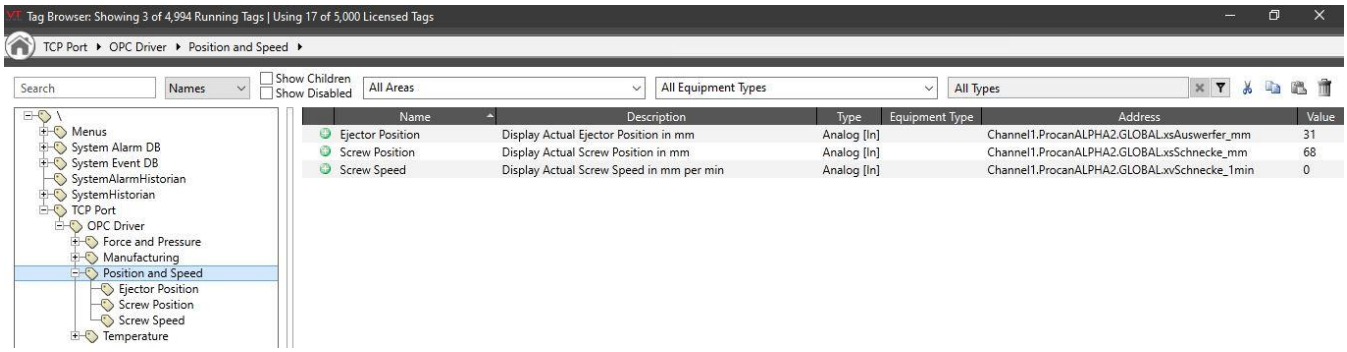



*Figure 4.15: Position and Speed I/O Tags*

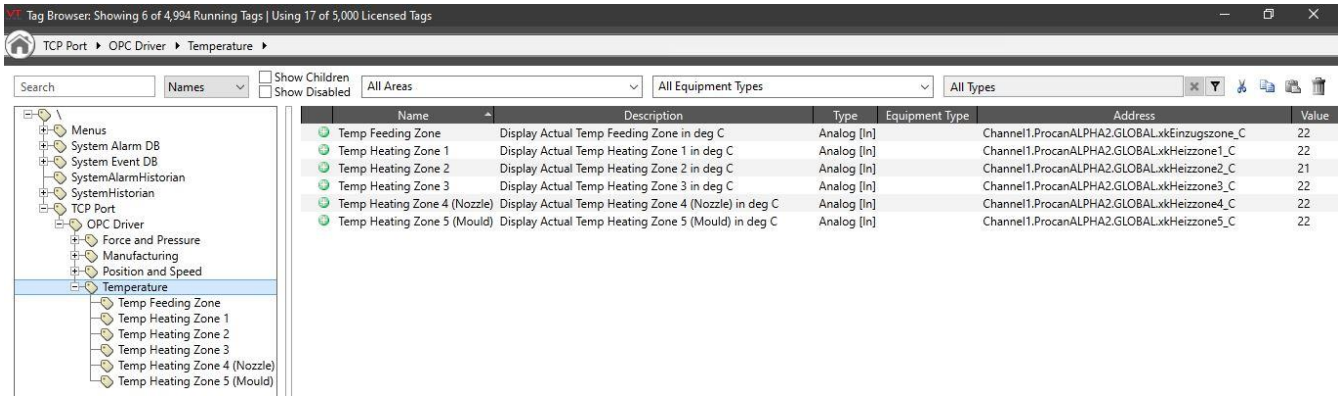



*Figure 4.16: Temperature (Heating Zones) I/O Tags*

A SCADA user interface is created using the VTSCADA Idea Studio function and the operator view is shown figure 4.17.

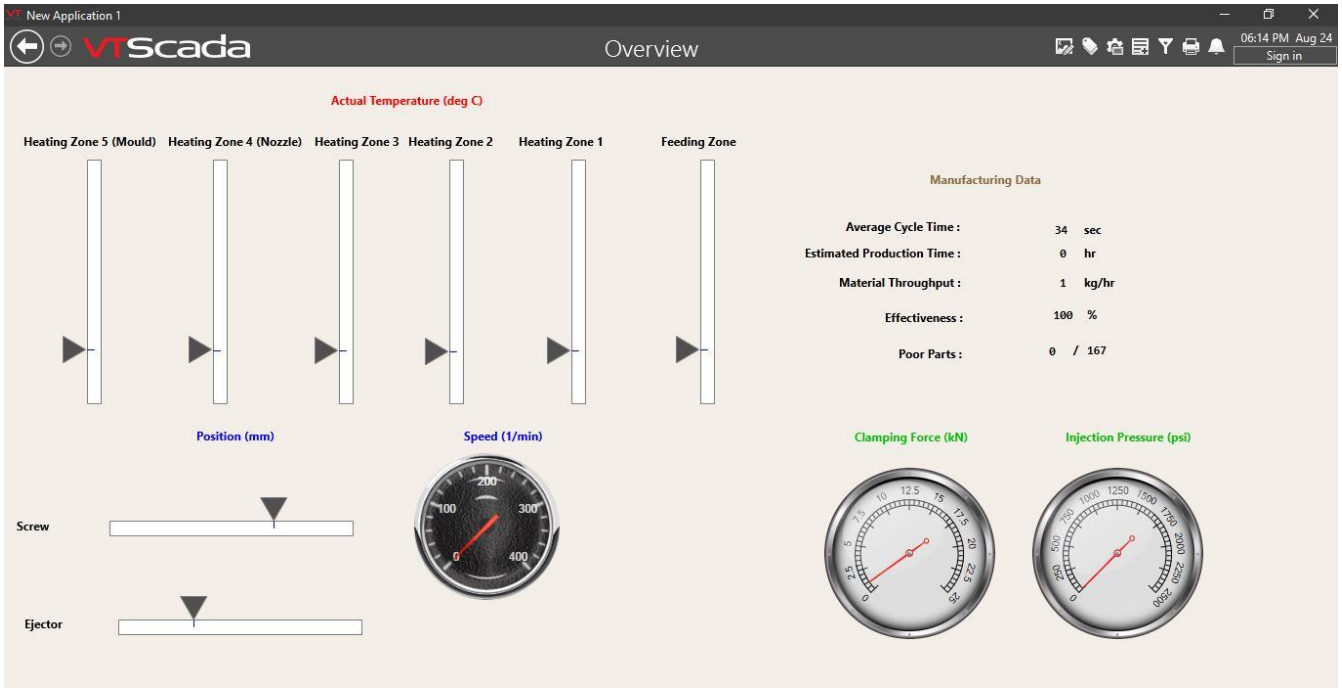



*Figure 4.17: SCADA Application Operator View*

By clicking on the tag widgets while in the operator view, it is possible to see the historical data/value of the tag, and this can be saved and stored. Figure 4.18 shows the historical data for the clamping force for the last 5 mins. The clamping force value hovers around 1 kN.

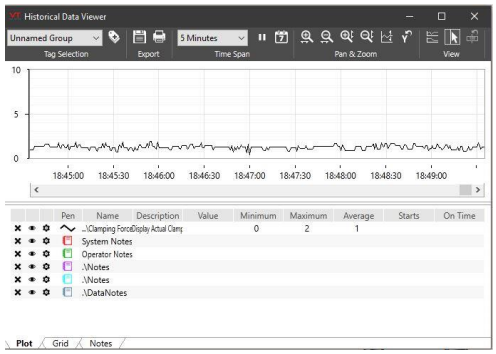



*Figure 4.18: Historical Data for Clamping Force Tag*

### 4.3. Samnian Automation Kit (Communication Technology and Tags)

The Automation Kit, figure 4.19, is used for automating a pneumatic piston using two pneumatic valves at the top and bottom for the extension and retraction of the piston cylinder. The pneumatic valves and other components can be controlled using the PLC installed with the kit. More information about the kit can be found in section 3 above.

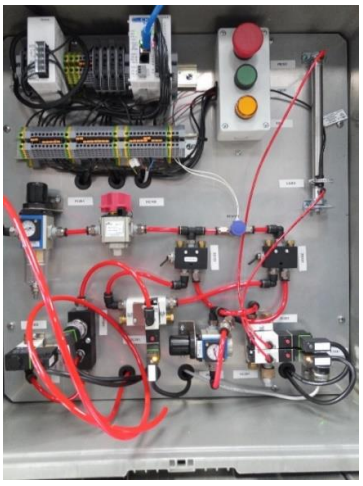


*Figure 4.19: Automation Kit*

The communication interface for the automation kit is Modbus TCP connectivity. Modbus TCP is a variant of Modbus for communication over TCP/IP network. Modbus is used for supervision and control of automation equipment.

The automation kit has numerous components (Figure 3.3 and Table 3.1) such as valves and sensors connected to a PLC and each such connection has a unique I/O address and therefore the relevant tags/variables are chosen from the list of available I/O for the PLC. A list of chosen tags, showing the sensor endpoint, instrumentation data sources, addresses, units, engineering units and conversion is shown in table 4.4.

*Table 4.4: List of Tags for Automation Kit*

| Sensor Endpoint | Instrumentation Data Sources | Addresses (*Data Register Address*) | Units | Conversions | Engineering Units |
|---|---|---|---|---|---|
| **Communication Interface:** Modbus TCP Connectivity | | | | | |
| Pressure | Pressure sensor (PT-301) | AD1I (*DF1*) | N/A | N/A | N/A |
| Piston Position | Ultrasonic position transmitter/sensor (LT-201) | AD2I (*DF2*) | cm | 1cm = 10 mm | mm |
| on/off | Pneumatic valve (ZC-201) | Y001 | None | None | None |
| on/off | Pneumatic valve (ZC-202) | Y002 | None | None | None |

The pressure is useful for knowing how much force is being applied to the piston/cylinder. The piston position is the most important variable because the goal of the automation kit project is to automate the piston so that it settles at a target position and therefore always knowing the actual position is necessary. The offset between the target and actual position lets the user know how well the ladder logic program can control the piston's motion. The pneumatic valves for extension (ZC-201) and retraction (ZC-202) are also very important to keep track of because they give

information about the piston's current state such as if it is going up, going down or not moving (or within the target tolerance).

### 4.4. Haas CNC Mill (Communication Technology and Tags)

The Haas CNC Mill, figure 4.20, uses a rotating cutter which translates and rotates in multiple axis to carve up and create details in the raw workpiece and turn it into a finished part according to the desired specifications. Most machines are capable of up to 6 axes of motion, 3 translational axes and 3 rotational axes. A controller and simulator are used to control the Haas CNC mill and important information about the mill's operations can be extracted from the controller.


*Figure 4.20: Haas CNC Mill*

The communication interface used by the Haas CNC Mill controller and simulator is an ethernet protocol via TCP Port. DPRNT or Machine Data Collection (MDC) are features included in the CNC program and when paired with PuTTY prints information (text/variable) either to a file or serial TCP port, in the case of DPRNT, or allows the interactive query/extraction of the CNC variable and macros using Q and E commands, in the case of MDC. Macros provide the

added capability of communicating with and extracting information from peripheral devices. G and M code can be written and printed out using DPRNT. On the other hand, MDC use Q600 command to request the content of any macro or system variable. They are like addresses. The Q600 commands that were relevant to performance and diagnostic were chosen.

A list of chosen tags, showing the sensor endpoint, instrumentation data sources, addresses, units, engineering units and conversion is shown in table 4.5.

*Table 4.5: List of Tags for Haas CNC Mill Controller and Simulator*

| **Sensor Endpoint** | **Instrumentation Data Sources** | **Addresses (*Q600 Commands*)** | **Units** | **Conversions** | **Engineering Units** |
|---|---|---|---|---|---|
| **Communication Interface:** Ethernet Protocol via TCP (DPRNT or MDC paired with PuTTY allow for printing or query of variables and macros) | | | | | |
| Maximum axis loads for X, Y, Z, A, and B Axes, respectively | N/A | Q600 #1064 - #1068 | N | None | N |
| Spindle load with Haas vector drive | N/A | Q600 #1098 | N | None | N |
| Maximum axis loads for C, U, V, W, and T-axes respectively | N/A | Q600 #1264 - #1268 | N | None | N |
| Maximum recorded vibrations of tools 1 through 200 | N/A | Q600 #1801 - #2000 | N/A | N/A | N/A |
| Tool length offsets of tools 1 through 200 | N/A | Q600 #2001 - #2200 | mm | None | mm |
| Tool length wear of tools 1 through 200 | N/A | Q600 #2201 - #2400 | mm | None | mm |
| Tool diameter/radius offsets of tools 1 through 200 | N/A | Q600 #2401 - #2600 | mm | None | mm |

| | | | | | |
|---|---|---|---|---|---|
| Tool diameter/radius wear of tools 1 through 200 | N/A | Q600 #2601 - #2800 | mm | None | mm |
| Spindle RPM | N/A | Q600 #3027 | rpm | None | rpm |
| Actual diameter for tools 1 through 200 | N/A | Q600 #3201 - #3400 | mm | None | mm |
| Tool life monitor counter | N/A | Q600 #5701 - #5800 | None | None | None |
| Pallet usage count | N/A | Q600 #7801 - #7806 | None | None | None |
| ATM percent of available tool life of all tools in the group | N/A | Q600 #8501 | % | None | % |
| ATM total available tool usage count in the group | N/A | Q600 #8502 | None | None | None |
| ATM total available tool hole count in the group | N/A | Q600 #8503 | None | None | None |
| ATM total available tool feed time (in seconds) in the group | N/A | Q600 #8504 | sec | None | sec |
| ATM percent of available tool life of the next tool | N/A | Q600 #8511 | % | None | % |
| ATM available usage count of the next tool | N/A | Q600 #8512 | None | None | None |
| ATM available hole count of the next tool | N/A | Q600 #8513 | None | None | None |
| ATM available feed time of the next tool (in seconds) | N/A | Q600 #8514 | sec | None | sec |
| Coolant level | N/A | Q600 #13013 | N/A | N/A | N/A |
| Estimated RPM for tools 1 through 200 | N/A | Q600 #50801 - #51000 | rpm | None | rpm |
| Estimated Feed rate for tools 1 through 200 | N/A | Q600 #51001 - #51200 | rpm | None | rpm |

The maximum axis loads, and spindle load give insight on the forces and stresses acting on the mill components such as the spindle and workpiece. Vibrations are generally unwanted, and it is important to keep track of the amount of vibrations at all times. The tool length and diameter offset, and wear gives insight into accuracy of the milling process and condition of the tool respectively. The actual diameter can also be compared with the offset. Spindle and estimated tool rpm along with the estimated tool feed rate should also be closely monitored for any deviation from the desired rpm or feed rate. This affects the quality of the finished product. Tool life counter simply monitors the life of the tool based on usage. Pallet usage counter gives a similar information but on the pallet. ATM stands for Advanced Tool Management and give a more in-depth analytical data to help in managing the tool. This includes advanced information on tool usage, tool life, number of holes made by the tool, and the available feed time. The final tag/variable to keep track of is the coolant level because it is very bad for the CNC Mill to overheat and it should be always properly cooled.

### 4.5. Temperature and Process Controller (Communication Technology and Tags)

The Watlow temperature and process controller shown in figure 4.21 is used to control the Blue M Standard Mechanical Convection oven shown in figure 4.22. Convection oven are capable of maintaining constant temperature using fan and exhaust to circulate hot air inside the oven. It is a dry oven and can also be used to dry or remove moisture from a product. Other uses include thermal testing, curing, and aging.


*Figure 4.21: Watlow F4T Temperature and Process Controller*


*Figure 4.22: Blue M Standard Mechanical Convection oven*

The communication interface used is an Ethernet Modbus TCP connectivity. A list of possible tag/variable that can be extracted from the controller is provided by the company through a register list spreadsheet. A list of chosen tags, showing the sensor endpoint, instrumentation data sources, addresses, units, engineering units and conversion is shown in table 4.6.

*Table 4.6: List of Tags for Watlow F4T Temperature and Process Controller*

| **Sensor Endpoint** | **Instrumentation Data Sources** | **Addresses (*Absolute*)** | **Units** | **Conversions** | **Engineering Units** |
|---|---|---|---|---|---|
| **Communication Interface:** Ethernet Modbus TCP connectivity | | | | | |
| Monitor the most recent load fault. | N/A | 406253 | N/A | N/A | N/A |
| Monitor the Timer block for input or timer errors. | N/A | 415051 | % | None | % |
| Monitor the variable output value. | Thermocouple | 415823 | deg C | None | deg C |
| Monitor/Set Target Set Point Loop 1 for current step | N/A | 416603 | deg C | None | deg C |
| Monitor/Set Target Set Point Loop 2 for current step | N/A | 416605 | deg C | None | deg C |
| Monitor/Set Target Set Point Loop 3 for current step | N/A | 416607 | deg C | None | deg C |
| Monitor/Set Target Set Point Loop 4 for current step | N/A | 416609 | deg C | None | deg C |
| Monitor/change ramp rate for control loop 1 | N/A | 416761 | N/A | N/A | deg C/s |
| Monitor/change ramp rate for control loop 2 | N/A | 416763 | N/A | N/A | deg C/s |
| Monitor/change ramp rate for control loop 3 | N/A | 416765 | N/A | N/A | deg C/s |
| Monitor/change ramp rate for control loop 4 | N/A | 416767 | N/A | N/A | deg C/s |
| Monitor control loop 1 precision | N/A | 416809 | N/A | N/A | N/A |
| Monitor control loop 2 precision | N/A | 416811 | N/A | N/A | N/A |

| | | | | | |
|---|---|---|---|---|---|
| Monitor control loop 3 precision | N/A | 416813 | N/A | N/A | N/A |
| Monitor control loop 4 precision | N/A | 416815 | N/A | N/A | N/A |

The load fault is monitored and a fault history can be generated. The variable output value is the temperature reading from inside the oven and is the most important tag/variable to monitor. The temperature reading should not deviate greatly from the set temperature. It is also very important to monitor the timer block to make sure that the timer error is kept to a minimum. The product being heated for too short or too long a time will lead to undesired results. The target temperature set point are monitored for all process loops at all steps to make sure that they are being met. The ramp rate will also be monitored for all process loops because the rate of temperature increase is critical. A very fast temperature increase can be very catastrophic on some product and a very slow temperature increase is not time efficient. The last set of tags that will be monitored are the precision of the control loops. The control loop precision should be high to allow for high repeatability of results.

### 4.6. Omega Temperature Process Control (Communication Technology and Tags)

The Omega microprocessor-based temperature process controller shown in figure 4.23 is used to control the Thermolyne Furnatrol II Muffle Furnace shown in figure 4.24. The furnace is used for melting down materials in a controlled and airtight environment, away from any impurities and isolated from fuel and any other products that can cause combustion. The furnace can get to very high temperatures, up to $1100\ °C$.


*Figure 4.23: Omega CN7200 Microprocessor Based Temperature Process Control*

*Figure 4.24: Thermolyne Furnatrol II Muffle Furnace*

The communication interface used are Modus ASCII or Modbus RTU over RS-485 serial Communications. In ASCII format, the messages are more readable but less efficient while in RTU format, the message are binary coded and is very difficult to read but the messages are smaller in size allowing for more data exchanges and a more efficient communication and thus the more popular choice.

A list of available RS-485 communication data register addresses is given in the controller's instruction sheet. A list of chosen tags, showing the sensor endpoint, instrumentation data sources, addresses, units, engineering units and conversion is shown in table 4.7.

*Table 4.7: List of Tags for the Microprocessor Based Temperature Process Control*

| Sensor Endpoint | Instrumentation Data Sources | Addresses | Units | Conversions | Engineering Units |
|---|---|---|---|---|---|
| **Communication Interface:** Modus ASCII or Modbus RTU over RS-485 serial Communications | | | | | |
| Temperature | Themocouple | 1000H | deg C or deg F | None | deg C |
| Proportional control offset error value | N/A | 100DH | % | None | % |

The temperature is the most important tag to monitor. It tells the user how well the furnace can maintain the set temperature. Another important tag to monitor is the proportional control offset error value. It tells the user if the proportional control is sufficiently controlling the temperature. This value should be as small as possible to have an accurate control.

### 4.7. Raise3D Pro2 Plus 3D Printer (Communication Technology and Reports)

The Raise3D Pro2 Plus, figure 4.25, is a 3D printer for large build volumes, while also still maintaining a high finish resolution and quality. Communication is established with the printer with using Raise3D IdeaMaker software, which connects the printer to the pc and allows the user to import or create CAD files on IdeaMaker and print them. Remote connection is established using RaiseCloud, which is an integrated 3D print management ecosystem that can be paired with IdeaMaker.


*Figure 4.25: Raise3D Pro2 Plus 3D Printer*

In addition to this, a remote access API (developer) is available to log data and generate reports from the printer which can then be stored in a database. The API documentation can be browsed for further information on how to use the API by visiting http://xxx.xxx.xxx.xxx:10800, with xxx.xxx.xxx.xxx being the printer's IP address.

Although individual tag/variable addresses are not available for the printer, the RaiseCloud platform provides the status of the printer and several in-dept reports which can be used to gauge the printer's performance and diagnose any issues. RaiseCloud stores reports from the last 3 months. These reports can be downloaded and saved into a database. A list of reports generated by RaiseCloud, showing the printer statistic, is shown in figure 4.26.

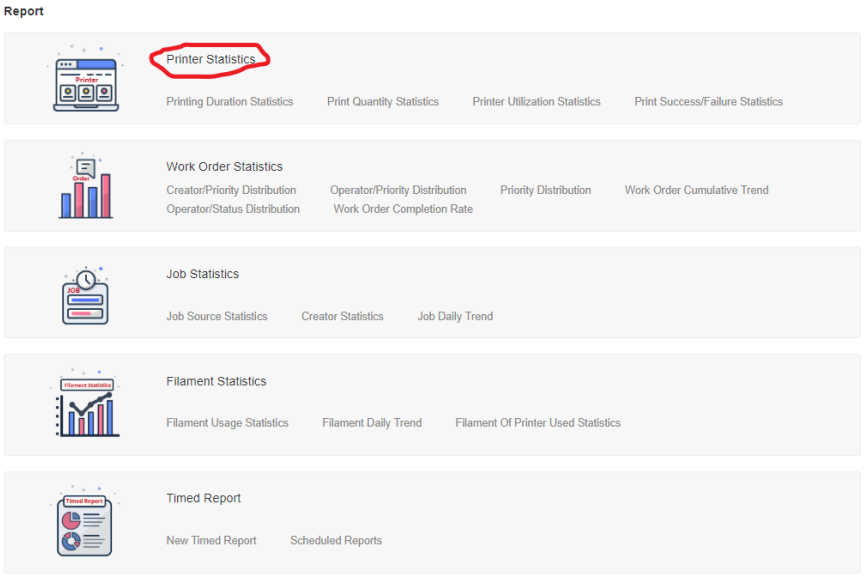



*Figure 4.26: RaiseCloud3D Report Module [4]*

The printer duration statistics gives information on how long different print jobs took, which can be used to gauge the efficiency of the job. Print quantity statistic gives information on how much material were used in different prints, which can also be used to gauge efficiency. The printer utilization is information of how much jobs are being conducted on the printer. Finally, the Print success/failure statistic is one of the most important reports because it provides information on jobs that failed, which is an indicator of effectiveness and might point to an issue with the printer or the user's operation of the printer.

## 4.8. Instron Load Frame (Communication Technology and Tags)

The Instron Load Frame, figure 4.27, is used to perform multiple different kinds of static testing such as tensile, compression, bend, shear, tear, cyclic, peel and other mechanical test and can generate forces of up to 50 kN. The load frame is connected to a computer by an ethernet protocol via TCP/IP, and the computer has the Instron Bluehill software, which is used to control the load frame and is used for running test and analyzing data.

*Figure 4.27: Instron 3369 Load Frame*

The connection between the Bluehill software on the computer and the load frame had previously been set up. The next task was to find a way of saving and exporting the data generated by the Bluehill software. The Bluehill 3 test data, especially those concerning displacement and load, can be exported into a CSV (semi-auto) and a custom import script can be written. All the information can then be linked and sent to a data server (image store).

## 5. Summary and Recommendations

This section summarizes the three projects and the report and recommends a few areas in which the projects could have been improved or conducted in a different manner. It also goes into details on the future works needed for these projects.

### 5.1. Summary

This report was written to detail the 3 projects that were conducted by the author during the summer. The first project involved updating the MECH 540B FPGA labs to reflect the newest version of the Quartus software. The second project involved controlling the piston level in the automation kit to be about the target set point by automating the pneumatic valves and other controlling valves while also accounting for the effects of gravity on the piston. The third project involved finding means of data acquisition from different equipment at the UBC Frank Forward 219 lab, by listing the communication interfaces, useful tags/variables that could be extracted, and other useful communication information. A SCADA application was also created for one of the equipment to demonstrate how the acquired tags from the equipment could be monitored using SCADA.

The first project involved updating 3 labs. A new Quartus 20 installation document was written up and it also had instructions on how to install Nios II software build tool which, unlike Quartus 16, had to be manually installed for Quartus 20. The first lab had no changes to the lab instructions. The only changes were to the “My First FPGA” guide which was an introduction guide to creating a new project on the DE1-SoC board using Quartus. The second lab had 2 changes. Talkback was no longer a feature that needed to be manually activated and the name of Signal Tap change to no longer including the “II” at the end. The third lab had the most changes, mostly involving the Nios

II software build tool. Nios II now had to be manually installed, and Nios II executable now had to be run from Nios II command shell. Other changes include QSYS was being renamed to Platform Designer, and System ID checks having to be checked before running the project.

The second project involved automating the MANU automation kit to get the piston level at the target position. A PLC ladder logic was designed to control the motion of the piston so that it settled at the target position. A ladder logic was first designed for the automation kit operating horizontally and then was adapted for a vertical positioning which factored in gravity. The extension and retraction of the piston was controlled by 2 pneumatic valves at either ends of the piston. The control logic involved the piston moving up or down towards the target position when the piston was outside the target tolerance. The opening and closing timing of the valves was controlled using high-low square wave 40ms ‘clock and counter’.

The last project involved finding means of communicating and extracting data from seven equipment in the lab and finding out what useful tags/variables related to performance or diagnostic that could be extracted from the equipment. A list of the chosen tags table, showing the sensor endpoint, instrumentation data sources, addresses, units, engineering units and conversion was created for each equipment. In addition, a VTSCADA application was made for the Injection moulding machine to demonstrate how the data can be extracted and monitored using SCADA. Instructions were provided on how to connect SCADA with the injection moulding PLC through KEPServerEX and how the SCADA application and operator view, showing the relevant tag, were set up.

### 5.2. Recommendations

There are no improvements needed for the first project but for simplicity's sake it might be preferable to use Quartus 16, because the NIOS II installation is very complicated and although version 20 is more modern, the same skills taught in the labs will be the same regardless of the version.

The ladder logic could have been further improved for the vertical positioning of the assembly kit. An improvement could have been to keep the retraction (upwards) valves open but at a lower flow rate instead of closing it completely when the piston was within the target tolerance. In this way the piston cylinder could be kept at the same level and wouldn't need to oscillate around the setpoint.

An improvement to the third project, would have been to have establish communication with all the equipment to confirm that the relevant tags were indeed able to be acquired. The injection moulding SCADA project could also have been improved by having a better operator view with a better looking and better organized UI. It would also have been good to have set up alarm and warning states for all the tags.

### 5.3. Future Work

For project 2, with the issue of gravity and automating to target setpoint being handled, it can be possible to incorporate a controller lab to the MANU 386 course that involves the implementation of various controller that can more efficiently get piston to the desired setpoint. For project 3, the next step would be to communicate with all the equipment and have the tags all in data servers such as SCADA Historian. The rest of the MANU Infrastructure detailed in figure 4.3 can then be implemented after.

## 6. References

1. T. Technologies, “SoC platform - cyclone - DE1-SoC Board,” Terasic. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836. [Accessed: 27-Aug-2021].

2. “AEROSPACE and defense software reviews,” Aerospace Export, 13-Dec-2020. [Online]. Available: https://aerospaceexport.com/best--manufacturingmanufacturing--executionexecution--systemsystem--mesmes--software/software/. [Accessed: 27-Aug-2021].

3. “MES / Oee / track and trace,” MIAC Automation. [Online]. Available: http://www.miac-automation.com/mes-oee-track-and-trace/. [Accessed: 27-Aug-2021].

4. “1.2. report,” Support Center | Raise 3D Technologies Inc. [Online]. Available: https://support.raise3d.com/RaiseCloud/1-2-report-6-40.html. [Accessed: 27-Aug-2021].

## Appendices

### Appendix A: MECH 540B Lab Manuals and Instructions

# Quartus Installation Instructions

**Quartus**

1. Download latest version of Quartus Prime Lite Edition (includes Nios II EDS) from the following URL and install it (download the latest version of edition 20 if available). https://fpgasoftware.intel.com/20.1.1/?edition=lite

**Nios II Software Build Tool for Eclipse**

1. Download CDT 8.8.1 which is Eclipse C/C++ IDE for Mars.2. Click to see installation instructions.
2. Install Windows Subsystem for Linux. Follow the installation steps. Use WSL 1 not WSL 2 because of better support.
3. With Ubuntu installed, performed the following commands in the Ubuntu shell
    a. sudo apt-get install update
    b. sudo apt-get install wsl
    c. sudo apt-get install dos2unix
    d. sudo apt-get install make

# MECH 540B Lab #3

Please ensure that you read both the procedures and questions for each section. **There is data requested in the Question sections which are not addressed in the procedure**.

**Purpose**

1. Build a simple application for an FPGA using schematic capture, Verilog, and VHDL.
2. Assign hardware pins and compile the project.
3. Execute the application on an FPGA and observe the results.

**Equipment**

1. DE1-SoC Development board
2. Laptop with Internet capability
3. Additional software/documentation:
    a. DE1-SoC_User_manual.pdf
    b. DE1-SoC schematic
    c. sevenseg.vhd

**Procedures**

Section 1: My First FPGA

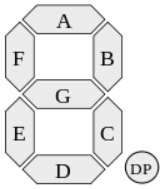
1. Follow up the guide attached to these instructions, completing all tasks therein.

Questions:

1. What would simple_counter.v be in VHDL?
2. Why is the NOT element necessary?
3. Why did the unused LEDs light dimly until you change the unused pin assignment?
4. At what frequency does the fastest LED transition with and without KEY[0] pressed?
5. What does KEY[1] do to the LEDs? Why?
6. How would you modify the code to count backwards?

Section 2: 7-Segment Displays

A 7-segment (or 9-segment) display maps binary coded decimal (BCD) to individual LEDs arrayed as a number 8. Lighting up the appropriate collection of LEDs displays a number. A 7-segment display VHDL file is provided that converts a 4-bit BCD input (0-9) into a binary output to display the number on a single display.



1. Add the VHDL code to your project.
2. Generate a symbol from the VHDL code.
3. Add the symbol to your schematic such that the output of the mux feeds the input of the 7-segment symbol, the clock is sourced from CLOCK_50, and the output goes to a 7-bit wide port (e.g. [6..0]).
4. Assign the new pins using the User Manual as a reference for pin numbers.
5. Compile and test the design.
6. Slow down the increment rate by passing counter[28..25] to the mux.
7. Compile and test the design again.
8. Examine the Compilation Report and answer the relevant questions.
9. Use the Chip Planner (under Tools) to examine your hardware utilization.

Questions:

1. What happens to the counter rate when KEY[0] is pressed?
2. What does the display show when an invalid binary value is supplied (i.e. >9)?
3. Why does the VHDL code set “0000001” (which maps to ABCDEFG) to turn on all LEDs except the centre? Why doesn’t a “1” light the LED?
4. What binary output would you use to create an “A”?
5. Include both a schematic (bdf) screen capture and VHDL equivalent (File -> Create HDL Design File from Current File…) of your main project file.
6. Is the VHDL clear to understand? Do you prefer schematic layout or VHDL?
7. What percentage of ALMs and pins were used? How many free PLLs are there?
8. Are all instantiated cells in the FPGA fabric immediately adjacent? If not, why are they separated?

# MECH 540B Lab #4 (***Updated***)

Please ensure that you read both the procedures and questions for each section. **There is data requested in the Question sections which are not addressed in the procedure**.

**Purpose**

1. Modify a VHDL state machine that communicates with an on-board ADC.
2. Use the SignalTap II Logic Analyzer to debug the program.
3. Interpret the ADC output and display measured voltage on your board.

**Equipment**

1. DE1-SoC Development board
2. ADC connector break-out cable
3. Laptop with Internet capability
4. Resistor kit
5. Additional software/documentation:
    a. DE1-SoC_User_manual.pdf + schematic
    b. Lab4Code.zip
    c. Lab3 project

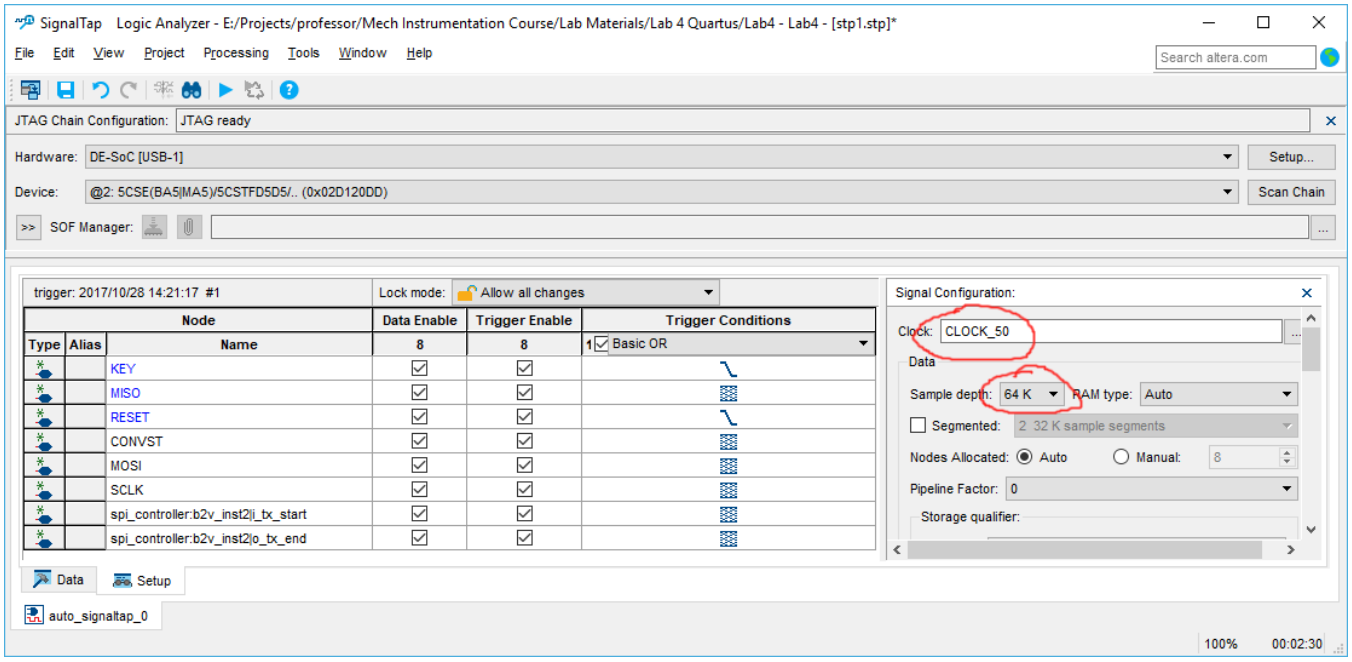
**Procedures**

<u>Section 1: Setup and Signal Analysis</u>

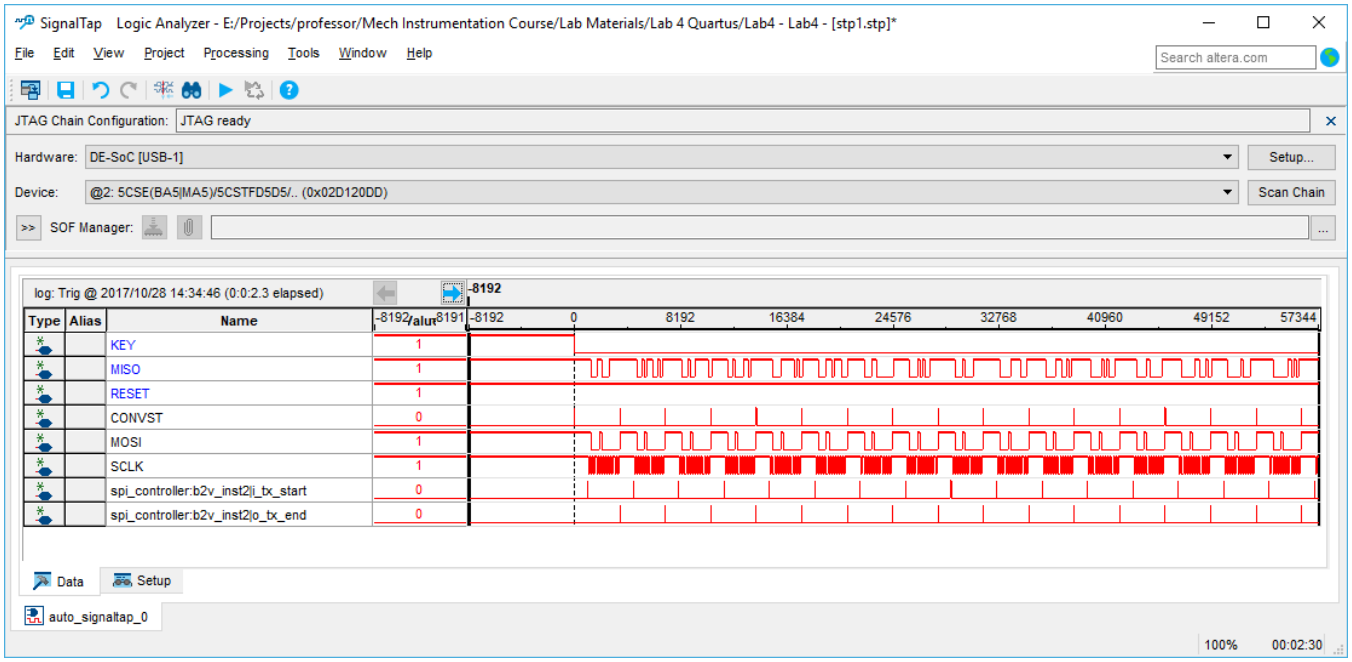
1. Create a new project as you did in Lab #3.
2. Copy Lab4.vhd and spi_controller.vhd into your project folder. Modify Lab4.vhd to ensure that your top level entity matches your project name.
3. Setup pins as shown:

| | | |
|---|---|---|
| in CLOCK_50 | Input | PIN_AF14 |
| out CONVST | Output | PIN_AJ4 |
| in KEY | Input | PIN_AA14 |
| out LEDS[9] | Output | PIN_Y21 |
| out LEDS[8] | Output | PIN_W21 |
| out LEDS[7] | Output | PIN_W20 |
| out LEDS[6] | Output | PIN_Y19 |
| out LEDS[5] | Output | PIN_W19 |
| out LEDS[4] | Output | PIN_W17 |
| out LEDS[3] | Output | PIN_V18 |
| out LEDS[2] | Output | PIN_V17 |
| out LEDS[1] | Output | PIN_W16 |
| out LEDS[0] | Output | PIN_V16 |
| in MISO | Input | PIN_AK3 |
| out MOSI | Output | PIN_AK4 |
| in RESET | Input | PIN_AA15 |
| out SCLK | Output | PIN_AK2 |

4. Synthesize the project and upload it to your FPGA. Reference your Lab #3 documentation for a reminder of the appropriate steps.
5. Pressure Button 0 (further from LEDs) to confirm your LEDs light.
6. Pressure Button 1 (no visible indication).
7. Open Tools → SignalTap Logic Analyzer. This tool will add logic to your FPGA that allows you to capture and view signals within your FPGA in real-time.
8. Setup your SignalTap Logic Analyzer as shown (you'll find the required signals under the SignalTap: Presynthesis filter):




9. Note the clock source, sample depth (number of samples/trigger), and trigger conditions shown for the selected nodes. You can select nodes by double clicking somewhere in the Setup page.
10. Save and click the blue arrow at the top of the window to recompile your project with SignalTap modules built in. Upload the new project to your FPGA.
11. Under Processing go to "Autorun Analysis". Go to the "Data" view and press Button 0 on your FPGA. Your FPGA will be communicating with the SignalTap Logic Analyzer via JTAG over USB and will trigger (like an oscilloscope) on a downward edge of either "KEY" or "RESET". An example capture is shown:




Questions:

1. What does each signal represent?
2. What do the LEDs represent?
3. Provide a logic analyzer screenshots after pressing “KEY” and “RESET”.
4. Provide a brief state diagram of (or otherwise describe) the behaviour of the state machine.

## Section 2: Optimizing the State Machine

1. Review the ADC datasheet in the appendix. The datasheet includes timing specifications for CONVST, which initiates conversion, serial clocking, and acquisition delays.
2. Update Lab4.vhd to increase the response speed of the ADC. This may involve increasing SCLK, reducing delays between states, changing the control register written to the ADC, and adding/removing components of the state machine to ensure acquisition as soon as the data is available.
3. Verify that your optimization state machine is working correctly by applying a voltage to ADC channel 0 subject to the following constraints. Reference the DE1-SoC datasheet to determine which pin (and colour) represents which signal:
    a. Always use the breakout header
    b. Measure +5V and GND to ensure that your orientation is correct.
    c. Build a resistor divider with output impedance >2 kOhm, <10 kOhm to generate your desired voltage using the +5V and GND available through the header.

4. Measurements at a given voltage should be stable (no LED flicker). GND should cause all LEDs to be off (except perhaps 1) and 5V should cause all LEDs to light up. **When you connect 5V to the ADC channel then do so through a 10 kOhm resistor.**

Questions:

1. Demonstrate your optimized configuration using the logic analyzer.
2. While holding down "KEY, how many samples can you acquire per second? Were you able to achieve the rated 500 ksp/s? Why or why not?
3. What is the largest constraint in ADC throughput?

Section 3: Display

1. Add in the 7segment display component from Lab #3.
2. Display the voltage being applied to the ADC pin (0V, 1V, 2V, etc.) on your 7-segment display. Only a single digit is required.

Questions:

1. What is the full scale voltage of the ADC?
2. What is the minimum resolution of the ADC?
3. What is the difference between 2s complement and binary outputs?

# MECH 540B Lab #5 (***Updated***)

Please ensure that you read both the procedures and questions for each section. **There is data requested in the Question sections which are not addressed in the procedure**.

**Purpose**

1. Build an analog ADC prefilter.
2. Modify the Lab 4 project to build a peripheral ADC interface for a Nios II CPU.
3. Build a Platform Designer-based VHDL project including Nios II Processor and peripherals.
4. Connect to the instantiated Nios II MCU and build a C project to acquire data from the ADC and transmit it to the PC.
5. Build a Matlab script to collect the data and support timing analysis.

**Equipment**

1. DE1-SoC Development board
2. AD2 board
3. ADC connector break-out cable
4. Laptop with Internet capability
5. Resistor and capacitor kit
6. Prototyping board
7. Additional software/documentation:
    a. DE1-SoC_User_manual.pdf + schematic
    b. Lab5Code.zip
    c. Lab4 project

**Procedures**

<u>Section 1: ADC prefilter</u>

1. Design a first order passive RC filter with a -3 dB threshold chosen for a 1000 sample/second ADC.
2. Implement the filter on your prototyping board.
3. Connect the function generator to ADC channel zero (as per Lab 4) via the filter.

Questions:

1. Did you remember to consider the Nyquist frequency when designing your filter?
2. What is the maximum fundamental frequency with which you can drive the ADC?

Section 2: Building a hardware peripheral

In Lab 4, you built and tuned VHDL code that acquired data from the 12-bit ADC on the DE1-SoC board. Here you will modify the project and create a peripheral device that can be linked to a CPU.

The main task is to add in a 16-bit wide parallel communication interface to the VHDL module. KEY, which was used to start and maintain acquisition, will be mapped to bit 0 of a new control register. Although you can continue writing the ADC results to a 7 segment display or LEDs, it must also be written to a new data register.

1. Create a new project in Quartus, as you have in Lab 3 and Lab 4.
2. Copy your Lab 4 VHDL files into the new project.
3. Open "Lab4forLab5.vhd" and modify your copied Lab4 files to suit:
    a. Add new ports:
        i. ADDRESS: Where 0 = Control register, 1 = Data register
        ii. READDATA: CPU bus that is "read" during a RD strobe
        iii. WRITEDATA: CPU bus that is "written" during a WR strobe
        iv. WR: Active low write strobe controlled by the CPU
        v. RD: Active low read strobe controlled by the CPU
        vi. CS: Active low chip select controlled by the CPU for selecting your ADC driver
    b. Add a new process in your architecture:

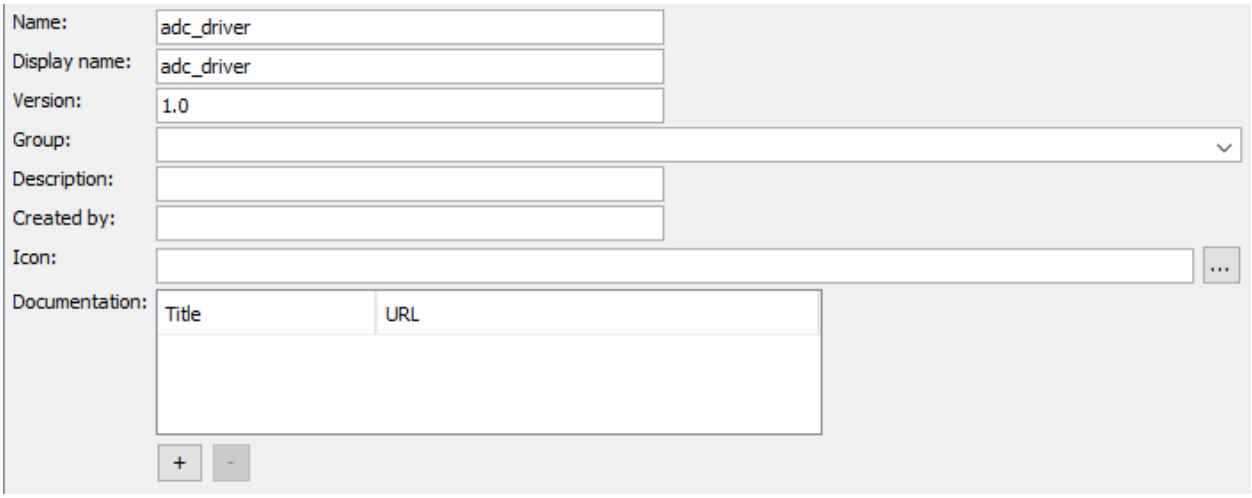
```
PROCESS(CLOCK_50, WR, RD, CS, ADDRESS, WRITEDATA, adc_value)
BEGIN
    --If the part is selected, on the rising edge of the system clock...
    IF rising_edge(CLOCK_50) and CS = '0' THEN

        --If writing to address 0 (control register)
        IF ADDRESS = '0' and WR = '0' THEN
            key <= WRITEDATA(0);

        --If reading from address 0 (control register)
        ELSIF ADDRESS = '0' and RD = '0' THEN
            READDATA <= "1010101010101010";     --Useful to activate this for testing
            READDATA <= "000000000000000" & key;

        --If reading from address 1 (data register)
        ELSIF ADDRESS = '1' and RD = '0' THEN
            READDATA <= "0101010101010101";     --Useful to activate this for testing
            READDATA <= "0000" & adc_value;
        END IF;
    END IF;
END PROCESS;
```

    c. Convert "KEY" from an entity-level port to a architecture-level signal. As you can see, "key" is then mapped to the LSB of our control register (address 0). WRITEDATA will contain the data written by the CPU when WR is strobed.

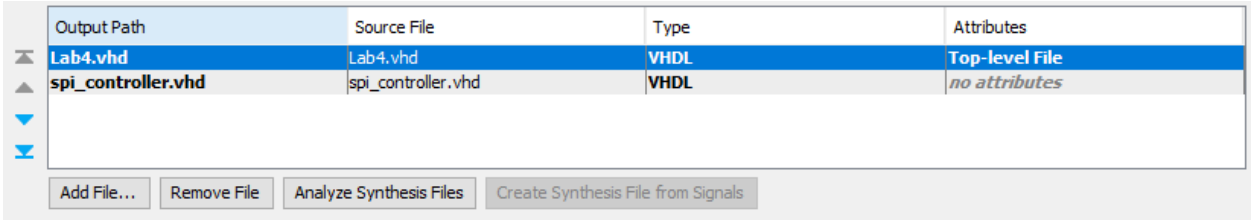
d. Invert the polarity of “RESET” in the spi_controller PORT MAP.

4. You may wish to copy your new file into your old Lab4 project (backup the original!) to see if you get any errors during synthesis. Don’t worry about missing pin assignments.
5. Once your code is error-free and with your Lab 5 Quartus project open, go to Tools→Platform Designer. Platform Designer is a tool for combining IP cores on common buses. It will synthesize all of the VHDL necessary for your Lab 5, including the Nios II CPU core.
6. Within Platform Designer, open Lab5.qsys (supplied in Lab5Code.zip). This basic starting project contains almost everything you need:
    a. clk_50: 50 MHz clock source.
    b. pll: Phase lock loop IP core that allows clock scaling.
    c. nios2_gen2_0: Nios II CPU core.
    d. onchip_memory2: Simple RAM/ROM memory module.
    e. sysid_qsys: Stores a compilation-specific build ID for diagnostics.
    f. timer: Hardware timer module that you can use if you wish.
    a. jtag_uart: JTAG UART module for providing communication between your Nios II CPU and a terminal application on your PC.
7. Top left of the screen, go to Project → *New Component…* and fill out the following windows.

    Component Type (feel free to add in your name):

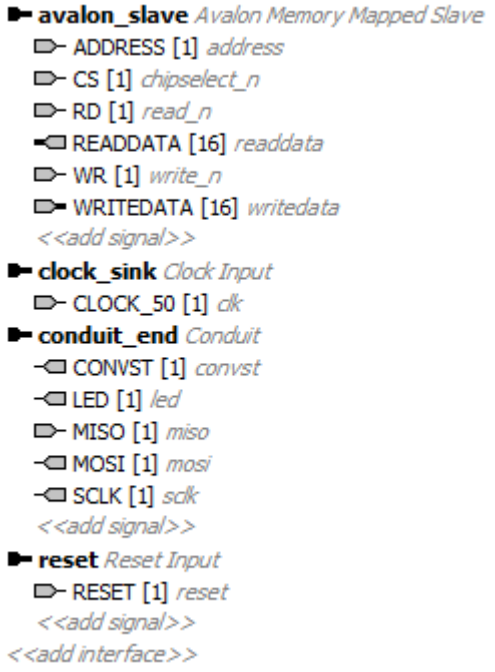



    Files (add any VHDL files required for Lab4):



Ensure that Lab4.vhd (or your equivalent) is at the top. Click **"Analyze Synthesis Files"**. This will process your input VHDL files, check for code correctness, and add signals to the "Signals & Interfaces" tab. You will probably get a lot of warnings and errors at the bottom of the screen. If you get an error stating "null" then there is an error in your Lab 4 VHDL that must be fixed!

Signals & Interfaces:




Assign your signals as shown. Note that your signals might differ slightly from mine if you've added segment display and other LEDs.

**Avalon_slave**: These signals are critical and dictate the buses and signals involved in passing information between the CPU and your VHDL project (adc driver).

**clock_sink**: This is where the 50 MHz that your project needs will come from.

**conduit_end**: These signals are routed to pins and will include any LEDs, keys, etc that your module will connect to on the board. It also includes all of the ADC communication signals.

**reset:** This is a system reset that may be triggered by the CPU on startup. It's passed to the SPI controller built into your project.

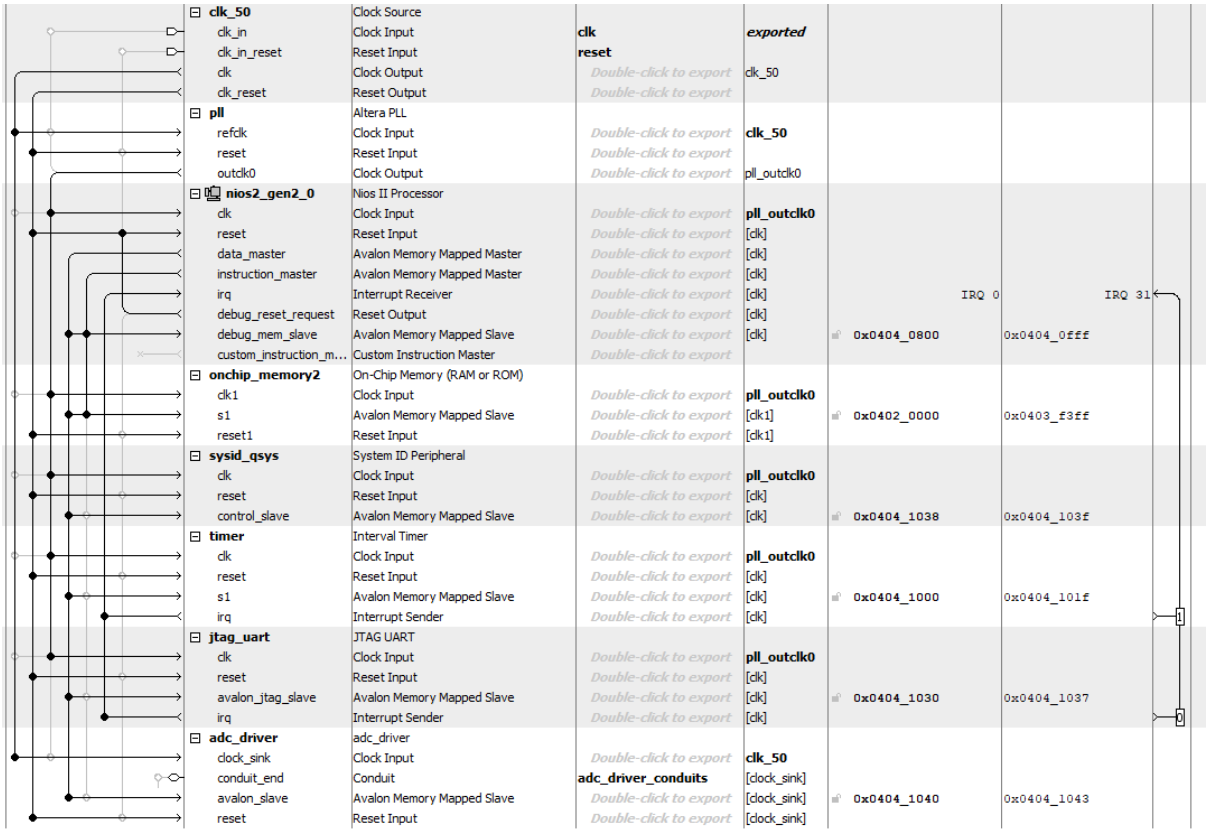
Signals can be dragged/dropped between interfaces. You will probably need to "add interface" a few times before you have all the categories that you need.

When ready to go, you should have no errors or warnings in your Messages window. Click "Finish…"

8. Add your new component to your system contents. Connect the various subcomponents as shown. The clock is sourced directly from the 50 MHz source, although this also requires that CPU be limited to 50 MHz. The analon_slave is connected to the data_master of the Nios CPU, and reset is tied into general system reset. You can make the connections by clicking on where the dots should be.

   Note that you must choose an address for your adc driver. This is an address in the 32-bit address space available to the Nios II core which will be address 0 for your driver.

You must also provide a name for your conduits to export them to the pin planner. See the column listing “Double-click to export”.




9. Save your Platform Designer project. Go to Generate→Generate HDL… Specify your project working directory.
10. Within Quartus, go to File→Open and select the Platform Designer file. Specify that you want to add the file to your project as the top level entity. Quartus will now examine the contents of your Platform Designer file and add all of the appropriate files and components to build the project.
11. Run “Start Analysis & Elaboration”.
12. Open the Pin Planner and assign your pins. Here is an example:

| Node Name | Direction | Location | I/O Bank | VREF Group | Fitter Location | I/O Standard |
|---|---|---|---|---|---|---|
| out adc_driver…its_convst | Output | PIN_AJ4 | 3B | B3B_N0 | PIN_AJ4 | 2.5 V |
| out adc_driver_conduits_led | Output | PIN_V16 | 4A | B4A_N0 | PIN_V16 | 2.5 V |
| in adc_driver…duits_miso | Input | PIN_AK3 | 3B | B3B_N0 | PIN_AK3 | 2.5 V |
| out adc_driver…duits_mosi | Output | PIN_AK4 | 3B | B3B_N0 | PIN_AK4 | 2.5 V |
| out adc_driver…duits_sclk | Output | PIN_AK2 | 3B | B3B_N0 | PIN_AK2 | 2.5 V |
| in altera_reserved_tck | Input | | | | PIN_AC5 | 2.5 V (default) |
| in altera_reserved_tdi | Input | | | | PIN_U8 | 2.5 V (default) |
| out altera_reserved_tdo | Output | | | | PIN_AB9 | 2.5 V (default) |
| in altera_reserved_tms | Input | | | | PIN_V9 | 2.5 V (default) |
| in clk_clk | Input | PIN_AF14 | 3B | B3B_N0 | PIN_AF14 | 2.5 V |
| in reset_reset_n | Input | PIN_AA14 | 3B | B3B_N0 | PIN_AA14 | 2.5 V |

13. Go to Assignments →Settings→EDA Settings→Simulation and set Tool name to "<None>". Do a full compile.
14. If everything runs well, then you will receive a window at the end of compilation informing you that there are time limited megafunctions in your design. This message is referring to the Nios II core. What this means that the Nios II core can only be run while your PC is connected and Quartus is running unless you buy the full license.
15. Program your FPGA using the same methods employed in Labs 3 and 4. An OpenCore Plus Dialog box will open up which allows the core on your FPGA to run.
16. Congratulations on making it this far!

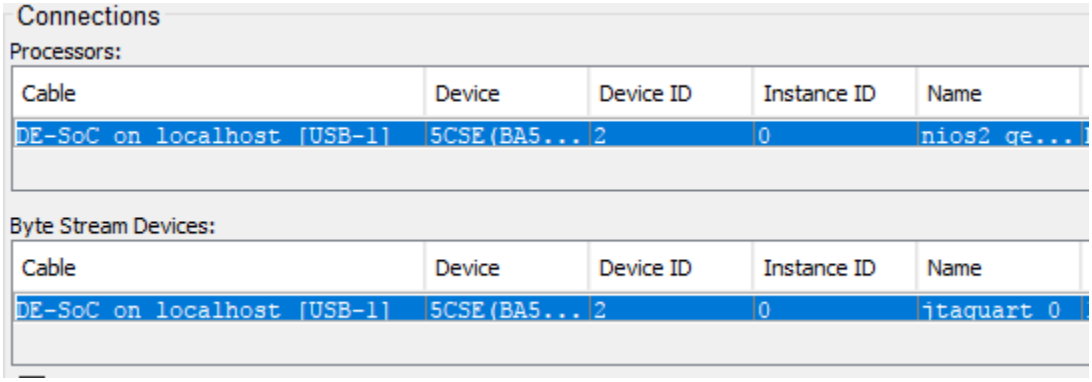
Questions:

1. How can you change the operating frequency of the MCU?
2. How would you add more memory (RAM/ROM) to your MCU?
3. How would you add general purpose I/O (GPIO) to your MCU?

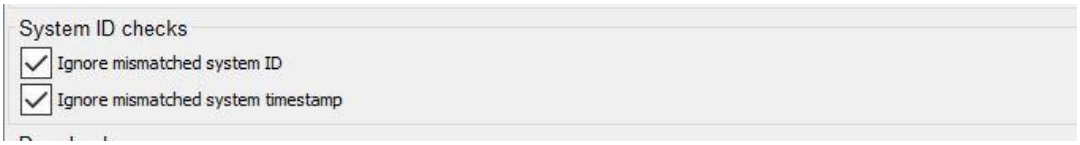
Section 3: Connecting to and programming your MCU

1. With your MCU core running on your FPGA, within Quartus, Open Nios II Command Shell. Navigate to the located /nios2eds/sdk2/bin folder and run eclipse-nios2.exe via the Shell, and NOT via Quartus itself, this ensures that your BSP generation will work on Eclipse with no errors This will open Eclipse, which is an open source IDE for software development.
2. Start a new project, specifically a "Nios II Application and BSP from Template".
3. The SOPC file was generated by Platform Designer and fully describes the hardware available to your CPU. Eclipse will parse that file and build a BSP library with all components and their corresponding memory locations appropriated mapped. Select the SOPC file from your project. Pick a project name and use a Blank Project template. Click finish.
4. Copy main.c into your Eclipse project workspace. Add it to your project.
5. Go to your Lab5_bsp project in the explorer on the left, right click, go to "Nios II", and click "Generate BSP". This step is required whenever you recompile your FPGA project. It will update your include files and libraries accordingly.
6. Open main.c.
    a. Read through the file and understand how it works. By setting your "control" register from 0x1 to 0x0, you are performing the same action through software as pressing key 0 from lab 4.
    b. Make sure that you change any references to "ADC_DRIVER_BASE" to the name that you chose for your adc driver instance in Platform Designer. You can check out the "system.h" file in your BSP project to see which peripherals were imported and how they are named.

7. Right-click on your Lab 5 project name and select “Build Project”. Address any errors that crop up. If you are not particularly strong with C then ask your TAs for assistance.
8. Click next to the green “Run” button in your toolbar, go to “Run Configurations”, and create a new launch configuration.
    a. Go to “NIOS II Hardware”
    b. Set Project name to your project name.
    c. The “ELF” file is the output file generated after successful compilation. It should be auto-populated once you select your project.
    d. Under “Target Connection” ensure that the DE-SoC is found. If it is not, then verify that your FPGA has been successfully programmed and that the Open Core Plus Status window is open (which allows the MCU to run).




    e. Check the System ID checks and apply




    f. Close the window.
9. Click the green “Run” button. This is will recompile your code (if necessary), stop the MCU, send your code to the MCU, activate the MCU, and open a Nios II JTAG UART console. Remember to regenerate your BSP when required!
10. If all went well, you should see numbers scrolling in your Nios II console representing ADC values acquired through your hardware ADC driver!

Questions

1. What are the minimum and maximum numbers that you can see by changing the voltage at the ADC input using your function generator?
2. What is the value of your adc driver base?
3. What is the value of the control register? How do you know?
4. Provide a screenshot of your Eclipse environment with data in your Nios II Console window.

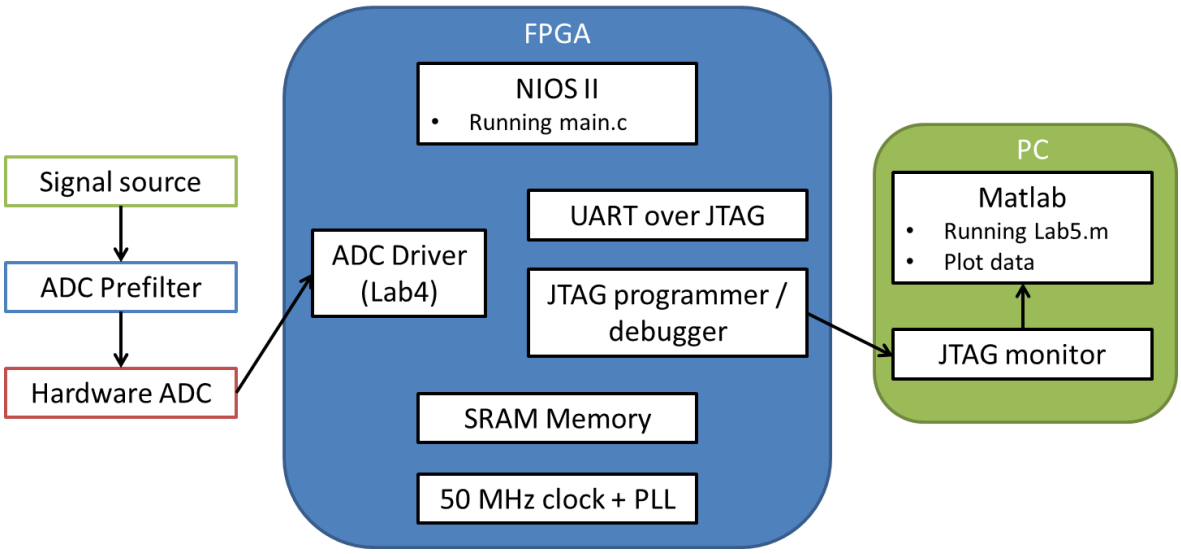
Section 4: Data acquisition using Matlab

By piping data to and from a command line utility, nios2-terminal.exe, it is possible to pipe data into Matlab in real-time.

1. With your MCU running, close the Nios II Console window in Eclipse. Only one service can access the JTAG UART at a time.
2. Launch Matlab.
3. Load Lab5.m in the Matlab editor.
4. Change the directory under "MON" in the file to reflect the location of your Quartus installation.
5. Run the file. The script will launch "nios2-terminal" in the background (invisibly) and pipe the data arriving from your MCU into Matlab variables (cmdout) line by line. The given implementation converts the strings provided into a floating point value and stores it in an array. At the end of the script, the array is plotted.
6. Note that the process is a bit finicky. If you get a lot of errors, trying restarting Matlab and/or Eclipse, or just keep trying.
7. Set a 10 Hz waveform on your AD2 function generator. Ensure that you have an appropriate DC offset and amplitude to suit the range of the ADC.
8. Run the Matlab script and look at the plot.
9. Change the delays in the Eclipse project and try again.
10. Adjust the function generator to produce 100 Hz, 250 Hz, and 400 Hz waveforms and capture the resulting plots.
11. Perform an FFT on captured data showing a single frequency spike (correctly identified on your axes). You may wish to increase the sample buffer size.
12. Attempt to maximize the sampling rate.
13. Try 10 and 11 for square and triangle waves.

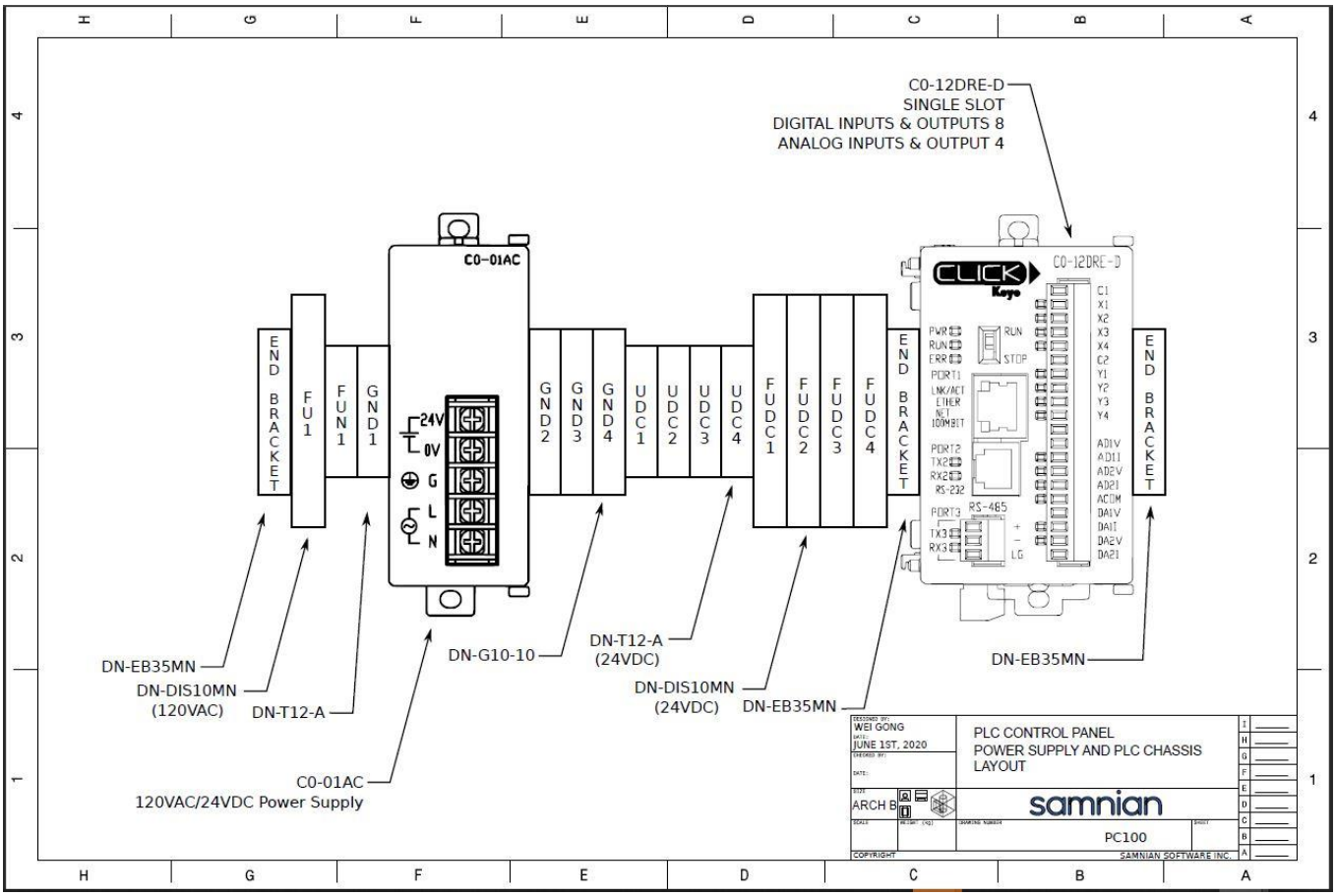
Questions:

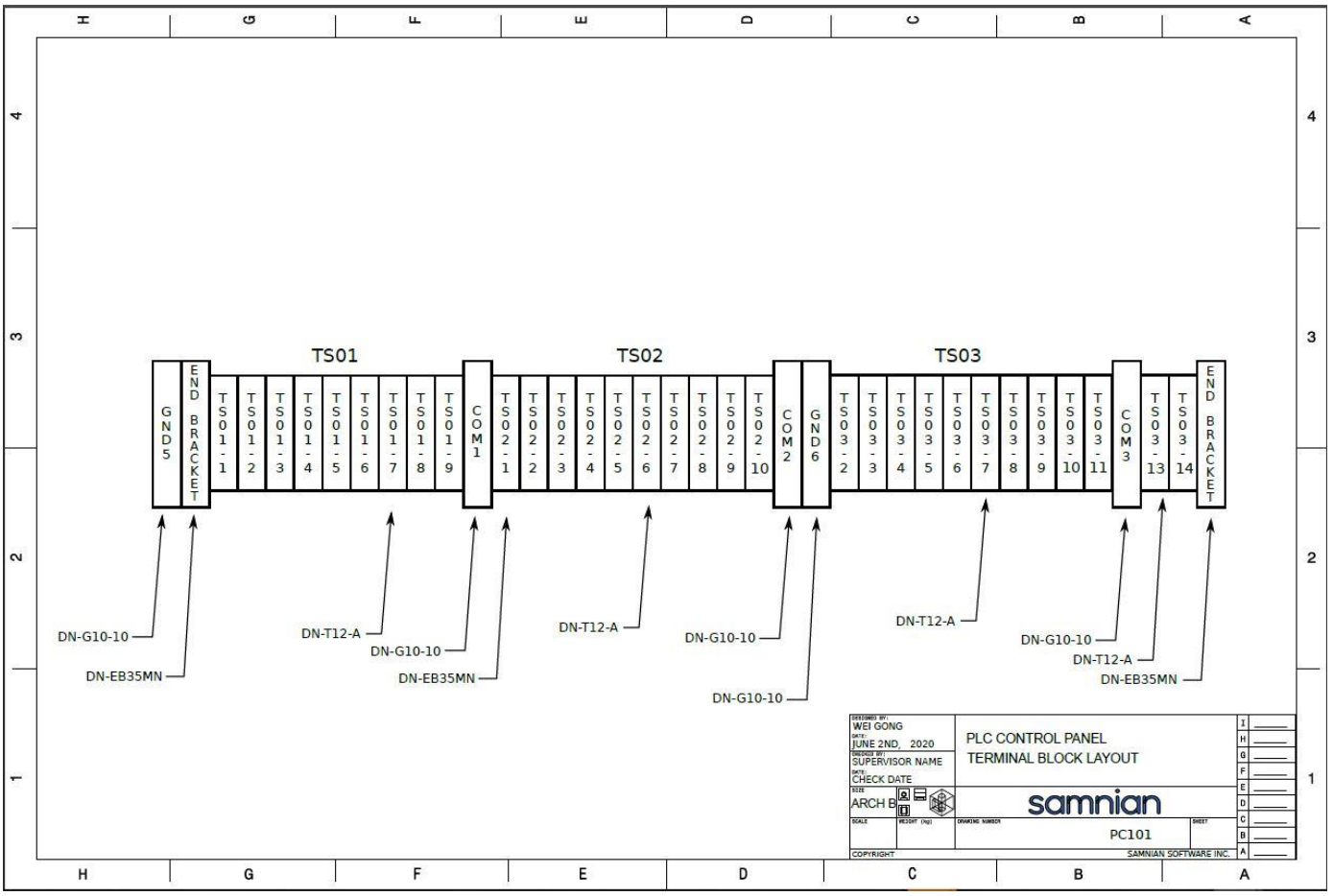
1. What is the sampling rate (acquisition speed) of your MCU code as seen by Matlab?
2. How fast can you get the acquisition rate? What steps did you take? What is the speed bottleneck?
3. Are acquired samples evenly spaced or is there significant jitter?
4. When performing 13, do the FFTs look like what you would expect? Why or why not?
5. If given more time, how would you further improve acquisition speed?

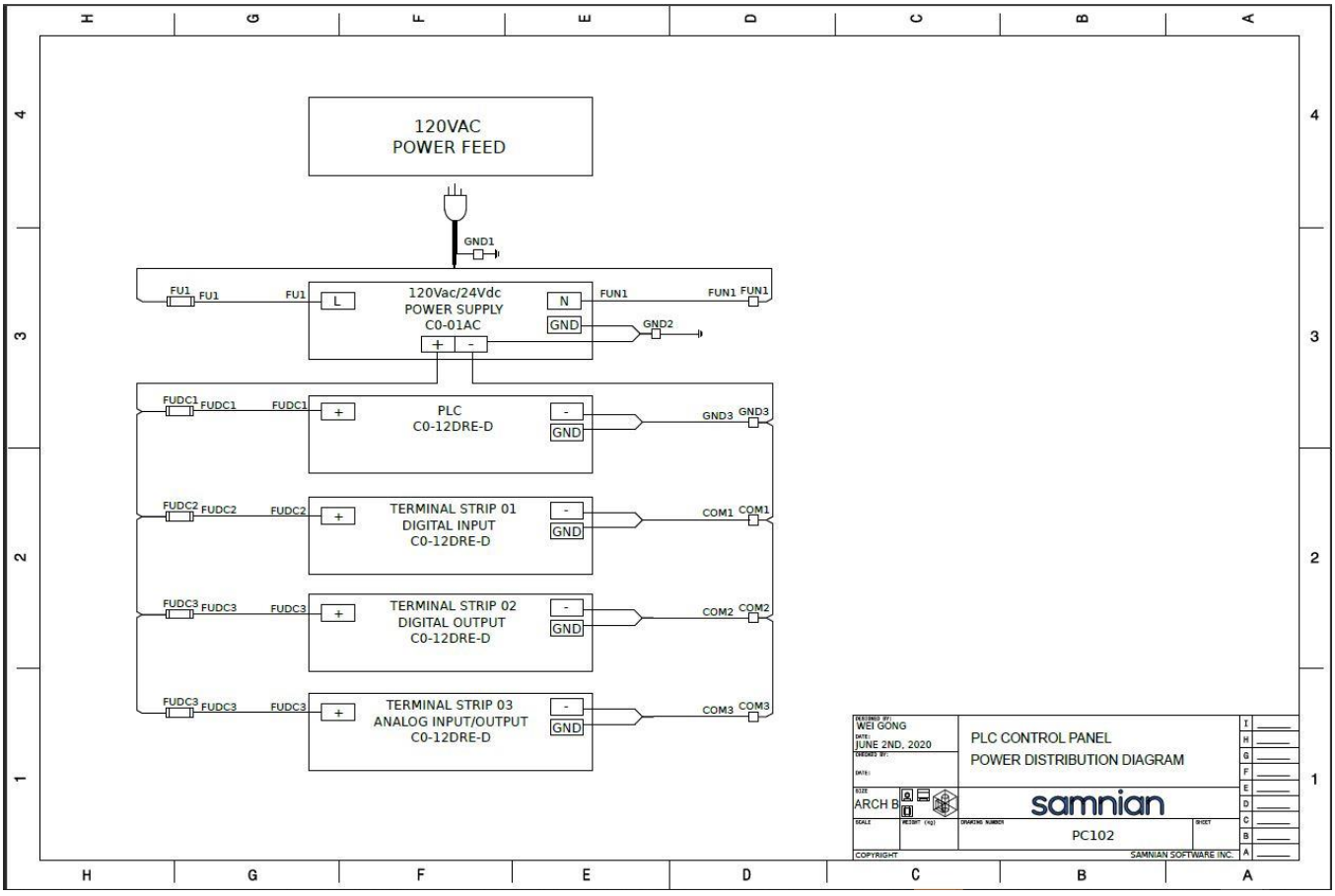

FPGA
NIOS II
• Running main.c
UART over JTAG
ADC Driver (Lab4)
JTAG programmer / debugger
SRAM Memory
50 MHz clock + PLL
Signal source
ADC Prefilter
Hardware ADC
PC
Matlab
• Running Lab5.m
• Plot data
JTAG monitor

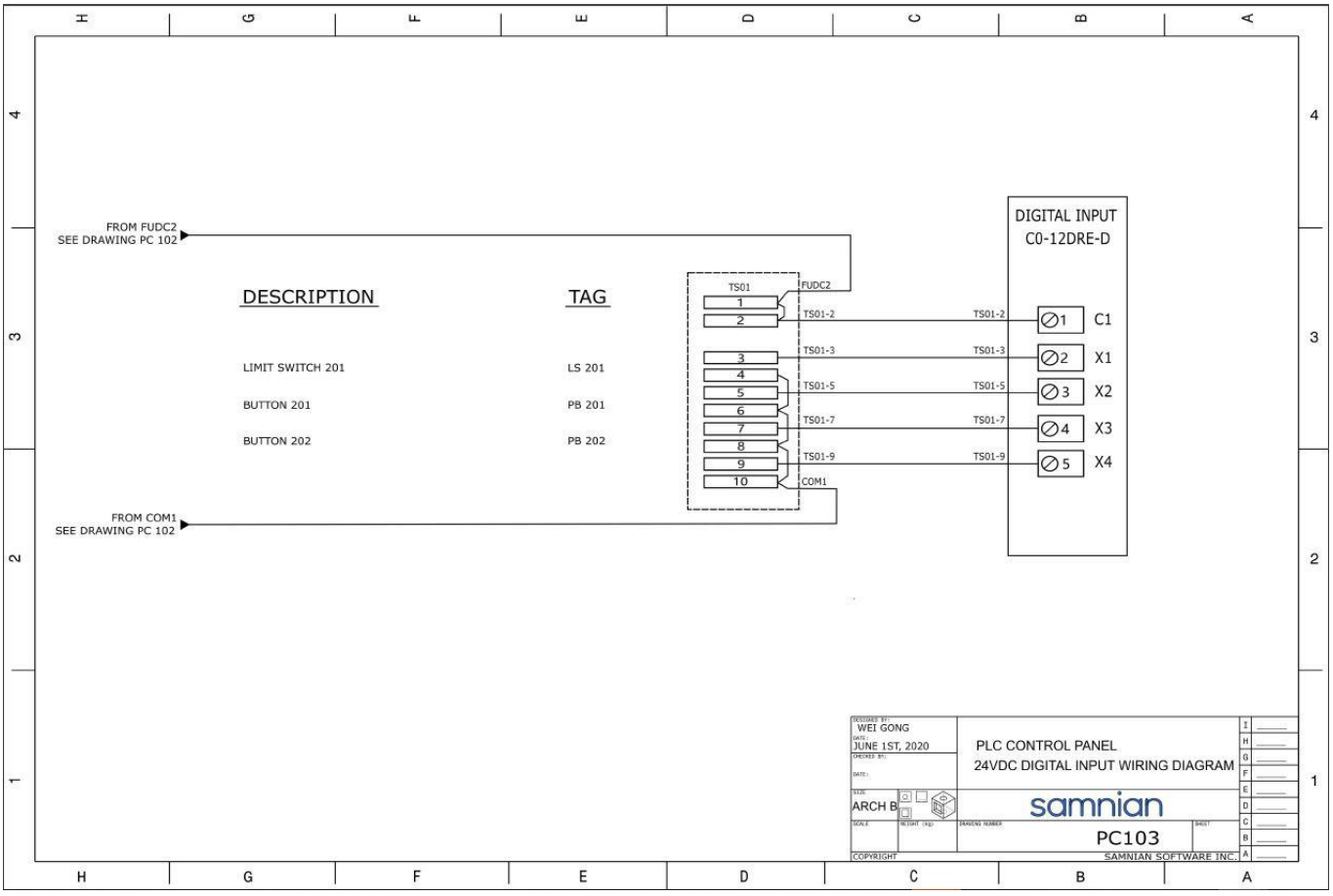
## Appendix B: PLC Information

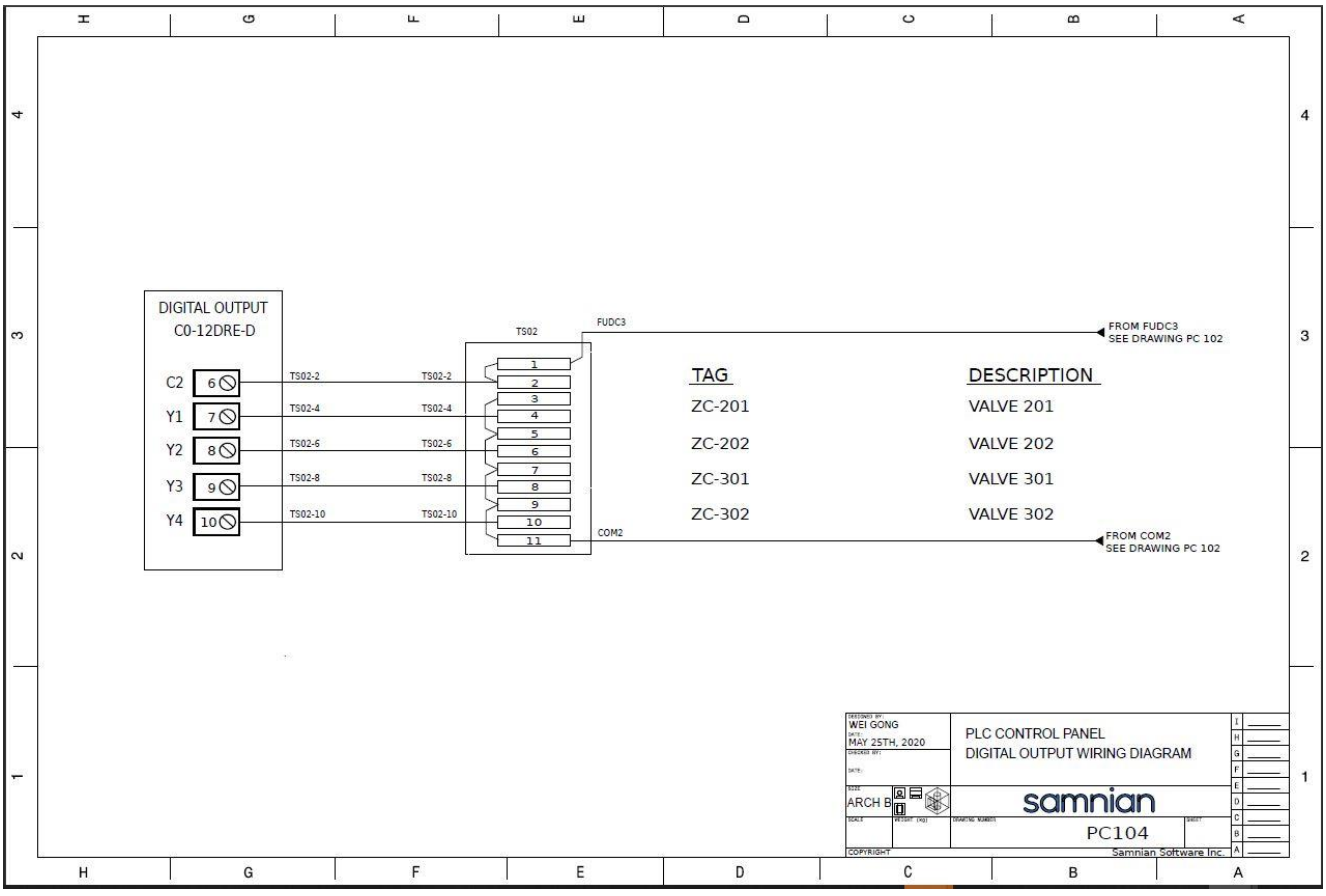
## Panel drawings

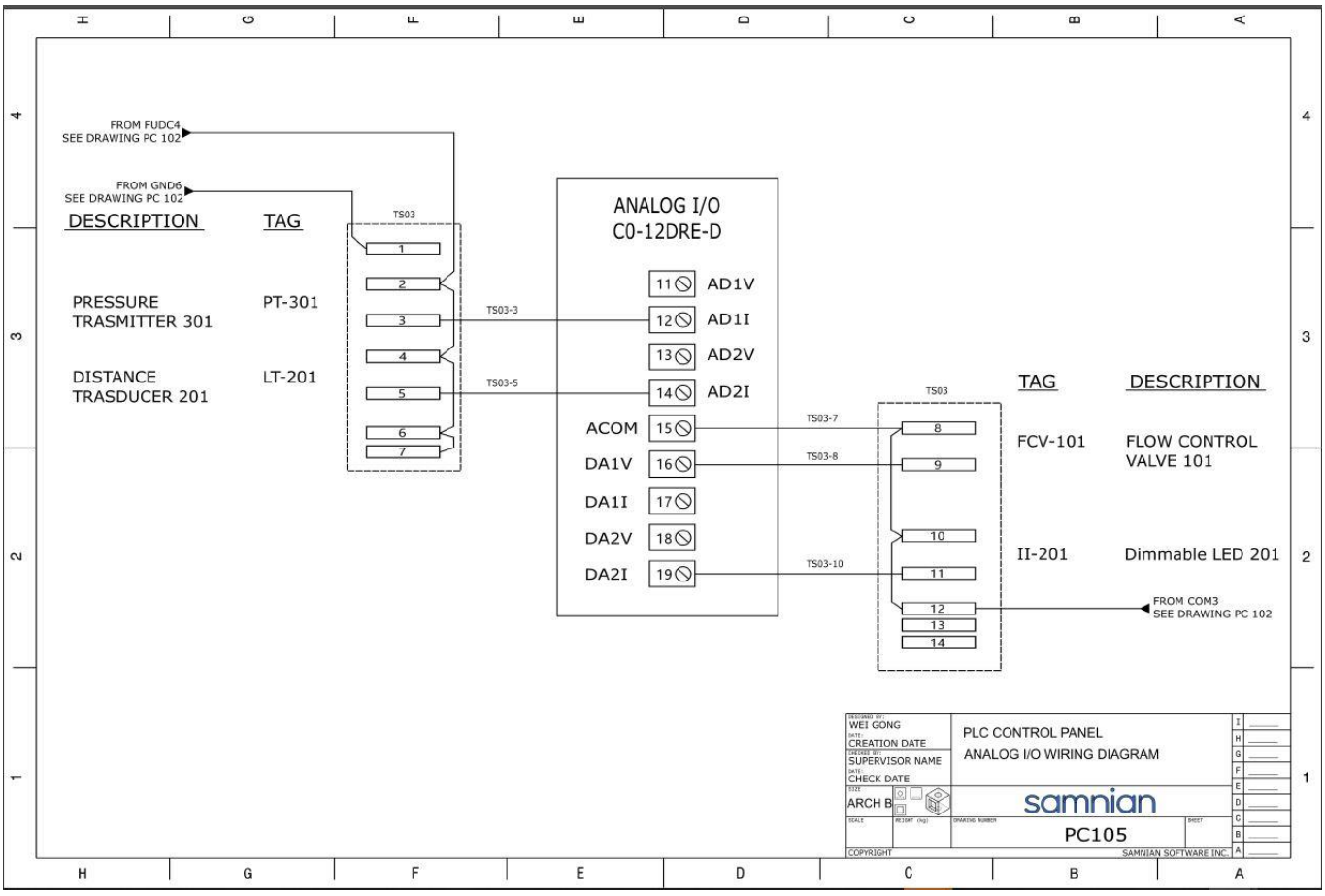


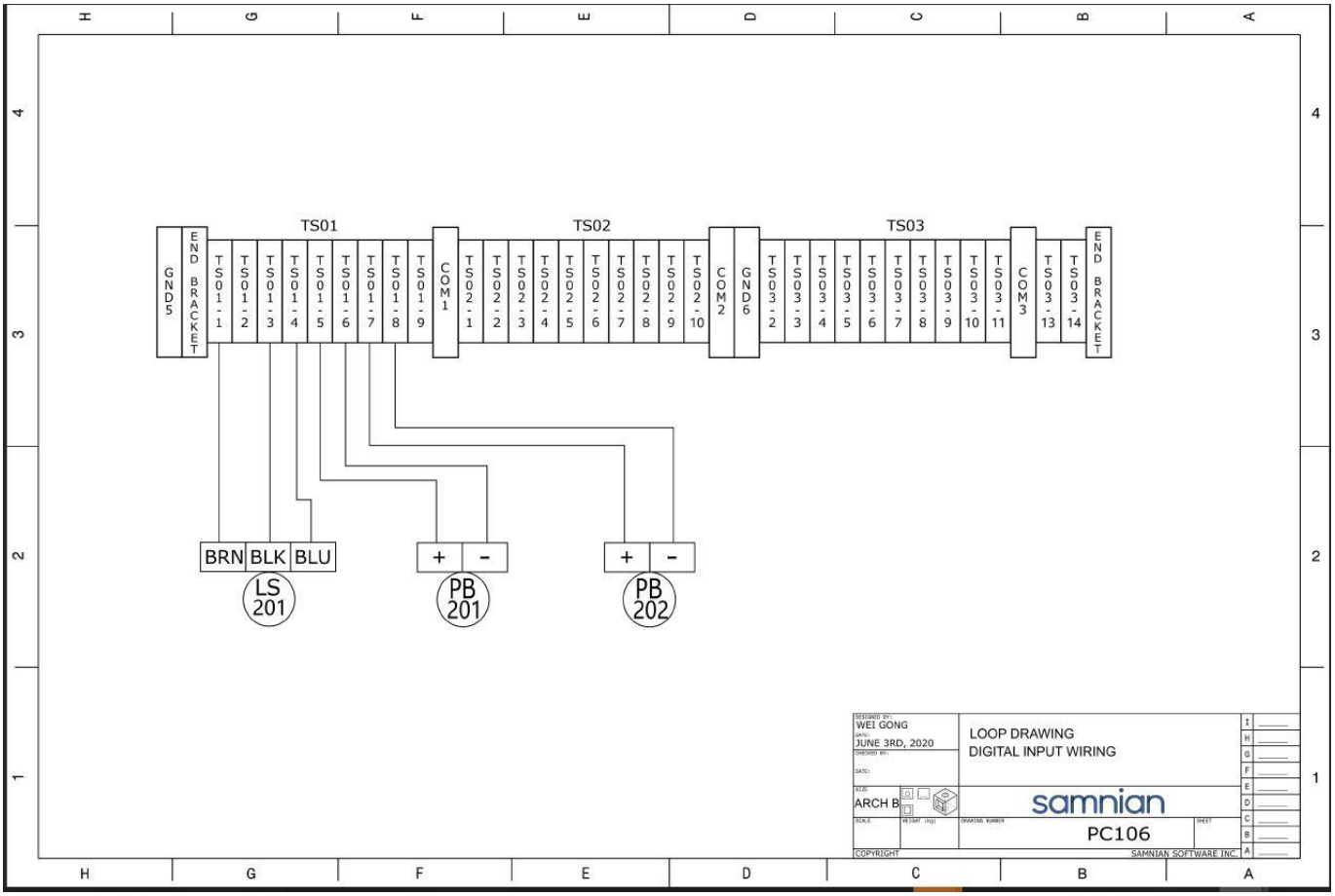

TS01
TS02
TS03
GND5
END BRACKET
TS01-1
TS01-2
TS01-3
TS01-4
TS01-5
TS01-6
TS01-7
TS01-8
TS01-9
COM1
TS02-1
TS02-2
TS02-3
TS02-4
TS02-5
TS02-6
TS02-7
TS02-8
TS02-9
TS02-10
COM2
GND6
TS03-2
TS03-3
TS03-4
TS03-5
TS03-6
TS03-7
TS03-8
TS03-9
TS03-10
TS03-11
COM3
TS03-13
TS03-14
END BRACKET
DN-G10-10
DN-EB35MN
DN-T12-A
DN-G10-10
DN-EB35MN
DN-T12-A
DN-G10-10
DN-G10-10
DN-T12-A
DN-G10-10
DN-T12-A
DN-EB35MN
DESIGNED BY: WEI GONG
DATE: JUNE 2ND, 2020
CHECKED BY: SUPERVISOR NAME
DATE: CHECK DATE
SIZE: ARCH B
SCALE
WEIGHT (kg)
PLC CONTROL PANEL
TERMINAL BLOCK LAYOUT
samnian
DRAWING NUMBER: PC101
SHEET
COPYRIGHT
SAMNIAN SOFTWARE INC.

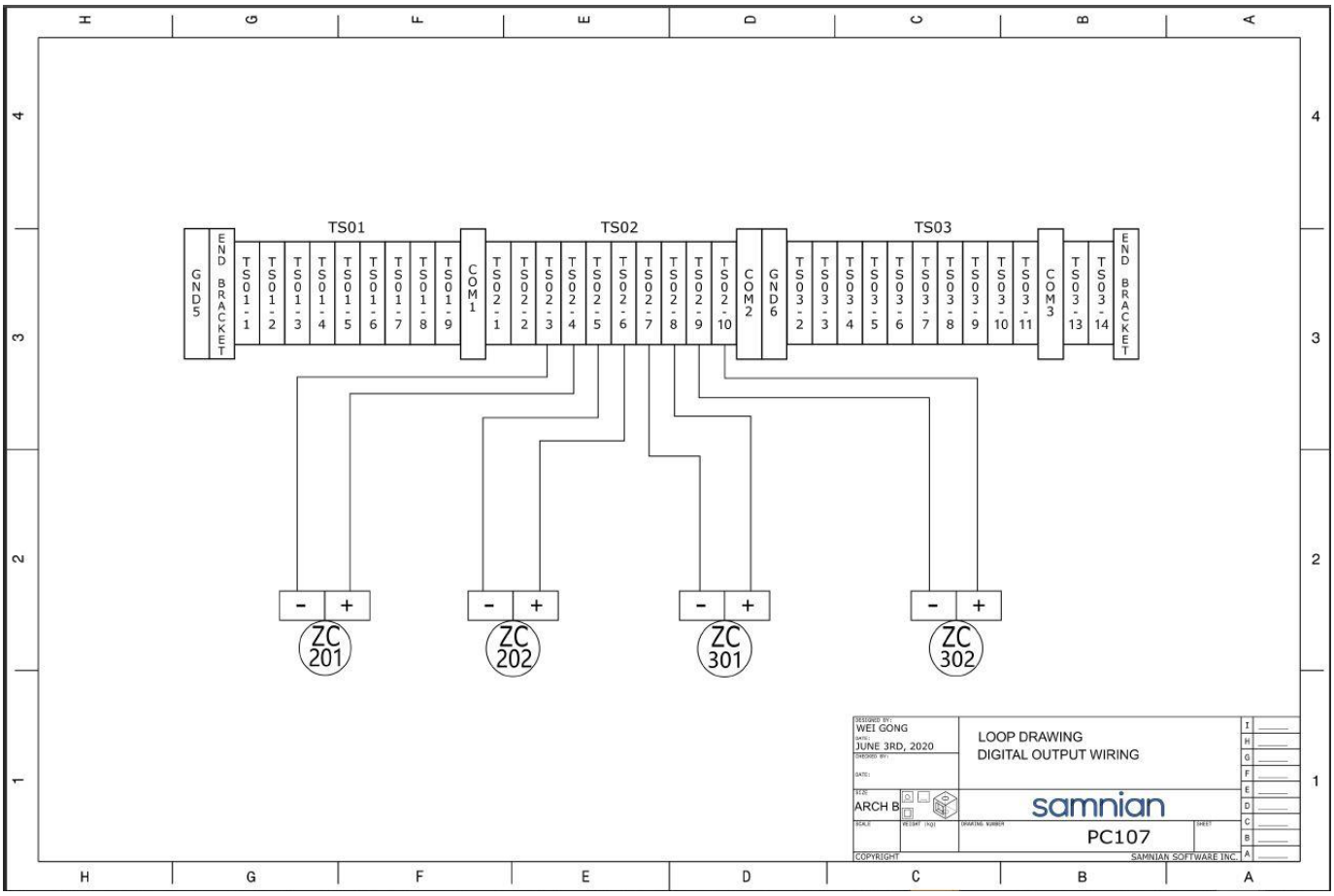

120VAC
POWER FEED
GND1
FU1
120Vac/24Vdc
POWER SUPPLY
C0-01AC
L
N
GND
FUN1
GND2
PLC
C0-12DRE-D
FUDC1
GND3
TERMINAL STRIP 01
DIGITAL INPUT
C0-12DRE-D
FUDC2
COM1
TERMINAL STRIP 02
DIGITAL OUTPUT
C0-12DRE-D
FUDC3
COM2
TERMINAL STRIP 03
ANALOG INPUT/OUTPUT
C0-12DRE-D
COM3
WEI GONG
JUNE 2ND, 2020
PLC CONTROL PANEL
POWER DISTRIBUTION DIAGRAM
ARCH B
samnian
PC102
COPYRIGHT
SAMNIAN SOFTWARE INC.

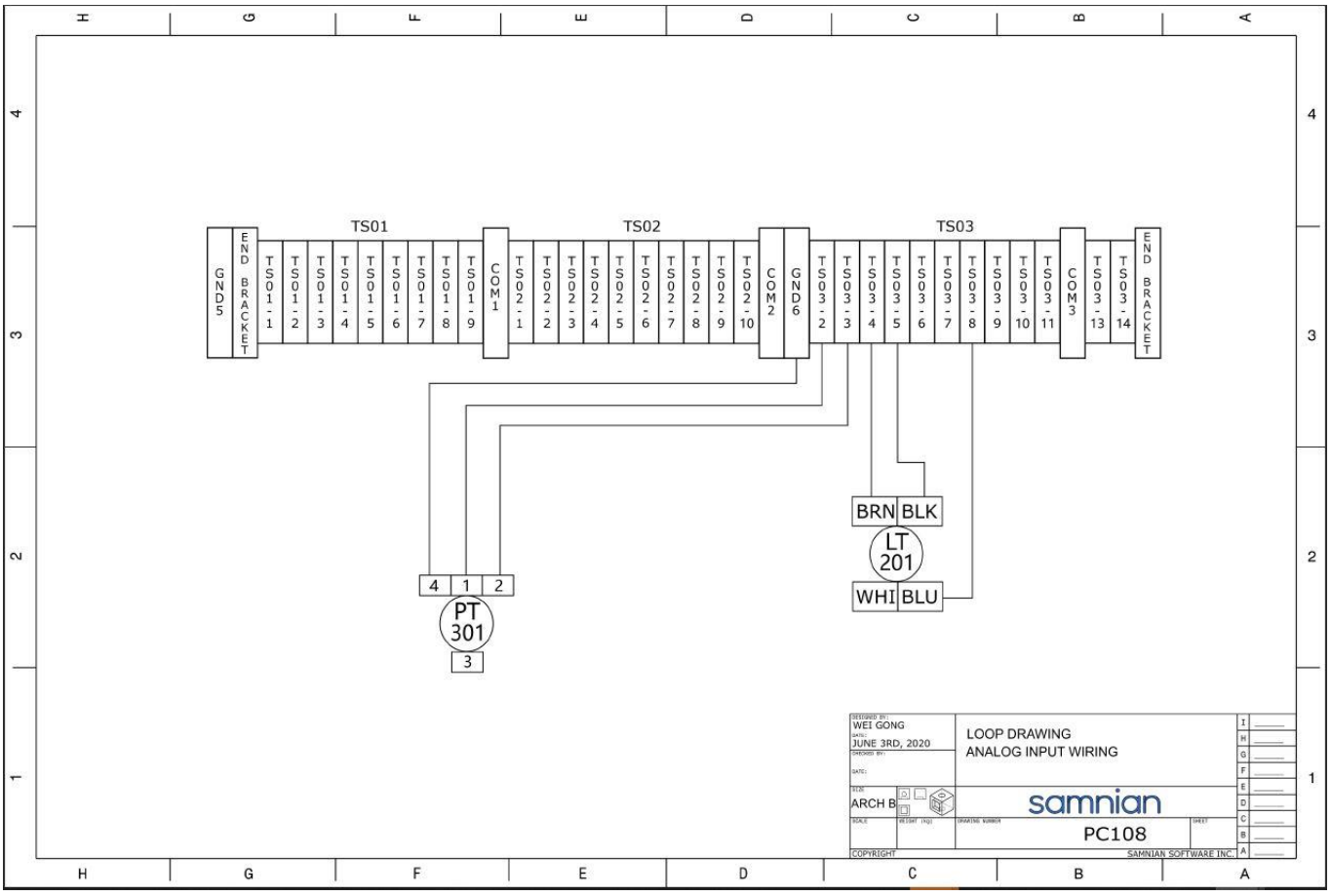

FROM FUDC2
SEE DRAWING PC 102
DESCRIPTION
TAG
LIMIT SWITCH 201
LS 201
BUTTON 201
PB 201
BUTTON 202
PB 202
TS01
FUDC2
TS01-2
TS01-3
TS01-5
TS01-7
TS01-9
COM1
DIGITAL INPUT
C0-12DRE-D
C1
X1
X2
X3
X4
FROM COM1
SEE DRAWING PC 102
WEI GONG
JUNE 1ST, 2020
PLC CONTROL PANEL
24VDC DIGITAL INPUT WIRING DIAGRAM
ARCH B
samnian
PC103
COPYRIGHT
SAMNIAN SOFTWARE INC.

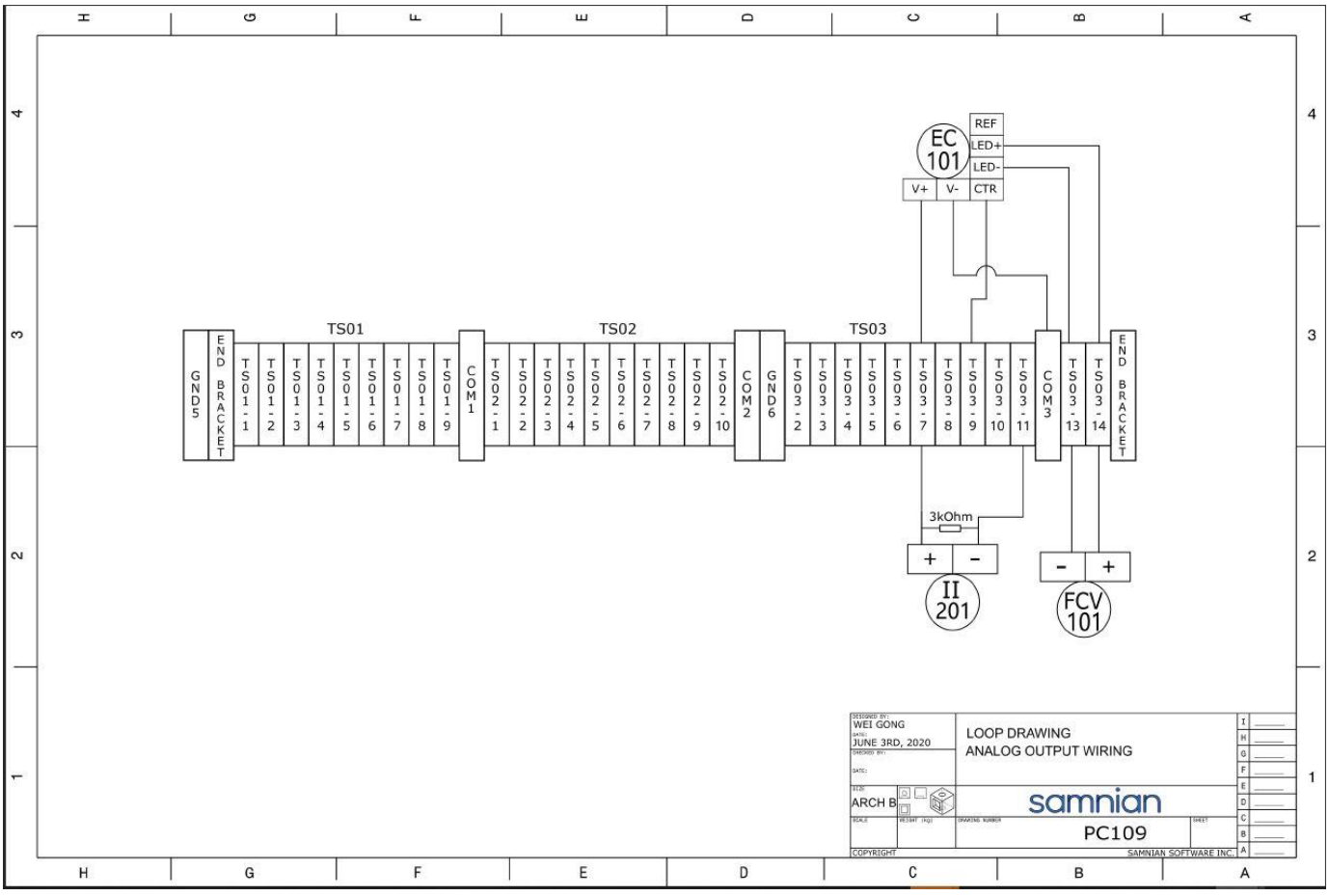

DIGITAL OUTPUT
C0-12DRE-D
C2
Y1
Y2
Y3
Y4
6
7
8
9
10
TS02-2
TS02-4
TS02-6
TS02-8
TS02-10
TS02
FUDC3
COM2
FROM FUDC3
SEE DRAWING PC 102
FROM COM2
SEE DRAWING PC 102
TAG
DESCRIPTION
ZC-201
VALVE 201
ZC-202
VALVE 202
ZC-301
VALVE 301
ZC-302
VALVE 302
WEI GONG
MAY 25TH, 2020
PLC CONTROL PANEL
DIGITAL OUTPUT WIRING DIAGRAM
ARCH B
samnian
PC104
COPYRIGHT
Samnian Software Inc.


FROM FUDC4
SEE DRAWING PC 102
FROM GND6
SEE DRAWING PC 102
DESCRIPTION
TAG
PRESSURE
TRASMITTER 301
PT-301
DISTANCE
TRASDUCER 201
LT-201
TS03
1
2
3
4
5
6
7
TS03-3
TS03-5
ANALOG I/O
C0-12DRE-D
11 AD1V
12 AD1I
13 AD2V
14 AD2I
ACOM 15
DA1V 16
DA1I 17
DA2V 18
DA2I 19
TS03-7
TS03-8
TS03-10
TS03
8
9
10
11
12
13
14
TAG
DESCRIPTION
FCV-101
FLOW CONTROL
VALVE 101
II-201
Dimmable LED 201
FROM COM3
SEE DRAWING PC 102
WEI GONG
CREATION DATE
SUPERVISOR NAME
CHECK DATE
ARCH B
PLC CONTROL PANEL
ANALOG I/O WIRING DIAGRAM
samnian
PC105
COPYRIGHT
SAMNIAN SOFTWARE INC.


TS01
TS02
TS03
GND5
END BRACKET
TS01-1
TS01-2
TS01-3
TS01-4
TS01-5
TS01-6
TS01-7
TS01-8
TS01-9
COM1
TS02-1
TS02-2
TS02-3
TS02-4
TS02-5
TS02-6
TS02-7
TS02-8
TS02-9
TS02-10
COM2
GND6
TS03-2
TS03-3
TS03-4
TS03-5
TS03-6
TS03-7
TS03-8
TS03-9
TS03-10
TS03-11
COM3
TS03-13
TS03-14
END BRACKET
BRN
BLK
BLU
LS 201
+
-
PB 201
+
-
PB 202
DESIGNED BY: WEI GONG
DATE: JUNE 3RD, 2020
CHECKED BY:
DATE:
SIZE: ARCH B
SCALE
VERSION (rev)
LOOP DRAWING
DIGITAL INPUT WIRING
samnian
DRAWING NUMBER: PC106
SHEET
COPYRIGHT
SAMNIAN SOFTWARE INC.


TS01
TS02
TS03
GND5
END BRACKET
TS01-1
TS01-2
TS01-3
TS01-4
TS01-5
TS01-6
TS01-7
TS01-8
TS01-9
COM1
TS02-1
TS02-2
TS02-3
TS02-4
TS02-5
TS02-6
TS02-7
TS02-8
TS02-9
TS02-10
COM2
GND6
TS03-2
TS03-3
TS03-4
TS03-5
TS03-6
TS03-7
TS03-8
TS03-9
TS03-10
TS03-11
COM3
TS03-13
TS03-14
END BRACKET
ZC 201
ZC 202
ZC 301
ZC 302
DESIGNED BY: WEI GONG
DATE: JUNE 3RD, 2020
CHECKED BY:
DATE:
SIZE: ARCH B
SCALE
WEIGHT (kg)
DRAWING NUMBER
SHEET
LOOP DRAWING
DIGITAL OUTPUT WIRING
samnian
PC107
COPYRIGHT
SAMNIAN SOFTWARE INC.


TS01
TS02
TS03
GND5
END BRACKET
TS01-1
TS01-2
TS01-3
TS01-4
TS01-5
TS01-6
TS01-7
TS01-8
TS01-9
COM1
TS02-1
TS02-2
TS02-3
TS02-4
TS02-5
TS02-6
TS02-7
TS02-8
TS02-9
TS02-10
COM2
GND6
TS03-2
TS03-3
TS03-4
TS03-5
TS03-6
TS03-7
TS03-8
TS03-9
TS03-10
TS03-11
COM3
TS03-13
TS03-14
END BRACKET
BRN
BLK
LT 201
WHI
BLU
4
1
2
PT 301
3
WEI GONG
JUNE 3RD, 2020
ARCH B
LOOP DRAWING
ANALOG INPUT WIRING
samnian
PC108
COPYRIGHT
SAMNIAN SOFTWARE INC.


EC 101
REF
LED+
LED-
V+
V-
CTR
TS01
TS02
TS03
GND5
END BRACKET
TS01-1
TS01-2
TS01-3
TS01-4
TS01-5
TS01-6
TS01-7
TS01-8
TS01-9
COM1
TS02-1
TS02-2
TS02-3
TS02-4
TS02-5
TS02-6
TS02-7
TS02-8
TS02-9
TS02-10
COM2
GND6
TS03-2
TS03-3
TS03-4
TS03-5
TS03-6
TS03-7
TS03-8
TS03-9
TS03-10
TS03-11
COM3
TS03-13
TS03-14
END BRACKET
3kOhm
+
-
II 201
-
+
FCV 101
DESIGNED BY: WEI GONG
DATE: JUNE 3RD, 2020
CHECKED BY:
DATE:
SIZE: ARCH B
SCALE
WEIGHT (KG)
DRAWING NUMBER
SHEET
LOOP DRAWING
ANALOG OUTPUT WIRING
samnian
PC109
COPYRIGHT
SAMNIAN SOFTWARE INC.

## Ladder Logic Programming