الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Ecole Nationale Supérieure de Technologie
Département : Génie Electrique et Informatique Industrielle

Mémoire de projet de fin d'études
Pour le mémoire d'Ingéniorat en Automatique

Spécialité
Automatique et Informatique Industrielle

Conception d'une plateforme de prototypage pour automate industriel

Réalisé par

Yannis AMMER
Abdelmadjid CHERGUI

Présenté et défendu publiquement le (06/07/2022)

Composition du Jury :

Président	M. Tarek CHERIFI,	M.C.B	ENST
Encadrant	M. Abdelhalim ZAOUI,	Prof.	ENST
Examinateur	M. Redha DERMOUCHE,	M.A.A	ENST
Examinateur	Mme. Lamia HABANI,	M.A.A	ENST
Co-Encadrant	M .Abdelmalek BELLOULA,		SMARTEST

Année universitaire 2021 –2022

الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
National Higher School of Technology
Department: Electrical Engineering and Industrial Computing

Thesis for the end of studies project
For the dissertation in Automation Engineering
Specialization
Automation and Industrial Computing

Design of a prototyping platform for industrial PLC

Realized by

Yannis AMMER
Abdelmadjid CHERGUI

Presented and publicly defended on (07/06/2022)

Composition of the Jury :

President	M. Tarek CHERIFI,	M.C.B	ENST
Supervisor	M. Abdelhalim ZAOUI,	Prof.	ENST
Examinator	M. Redha DERMOUCHE,	M.A.A	ENST
Examinator	Mme. Lamia HABANI,	M.A.A	ENST
Co-Supervisor	M .Abdelmalek BELLOULA,		SMARTEST

Année universitaire 2021 –2022

Dedication

“

*To my dearest mother **Nabila** the one that always believes in me, supports me, guides me and the source of my happiness,*

*To my life inspiration, my dear father **Samir**, for his encouragement, support, and especially for the sacrifices he made to see his son grow and be the man i am today,*

*To my little sister **Tinhinene**,*

*To **AMMER** and **GADOURI** families,*

*To my closest friends **Ali, Ahmed, Anis, Djamel, Imad, Nassim, Sisaleh, Tayeb, Youcef** and many other friends whom i encountered during my journey to be among the people who want to see me succeed,*

*And finally, To my supervisors, **Abdelmalek BELLOULA** and **Abdelhalim ZAOUI** for their guidance and support throughout this project.*

Thank You.

”

- Yannis

“

”A loved soul never leaves us’.

I dedicate this modest work to my grand parents ” khedidja, Hadj Ali and hadj Bouzid ” who left us early. ,

To my mother Rabia, who supports me in all my projects and always present for my ups and downs to make me smile. My mother, I hope that this work will be the testimony of my gratitude and my deepest respect towards the support, the sacrifices and all the efforts that she made for me. May God keep you and protect you for us inchallah. ,

To my father Si Aissa, thank you for always being my ally in the wars with my life. I love you. Nothing in my life would have been possible without your fight. Nothing I can say can truly express how I feel. Nothing will show all the gratitude I have for you. Dad, It is because of you that I have never missed anything in life. ,

To my older and younger brothers, Amine & yacine, who despite the distances always encourages me to go beyond my dreams. ,

To my beautiful aunts Oumeria, Samira and Zoulikha and my second mother (my grandmother) Saadia, for the enthusiasm and the support that you gave me. ,

To CHERGUI and BOUDJALEL families and all my friends ,

And finally, To my supervisors, ***Abdelmalek BELLOULA*** *and* ***Abdelhalim ZAOUI*** *for their guidance and support throughout this project.*

”

- Abdelmadjid

Acknowledgment

First and foremost, we want to thank Allah for giving us strength, patience, desire, bravery, and health to complete this project.

We'd also want to thank our parents and families for their support and encouragement during our long academic career.

We would like to express our gratitude and appreciation to our supervisors, ***Mr.Abdelmalek BELLOULA*** *for giving us the opportunity do an internship at* ***SPA SMARTEST*** *and for assisting us through every difficult obstacle in our way, and* ***Abdelhalim ZAOUI*** *who supported and assisted us throughout the project's many phases and whose experience was crucial in developing the research questions and methods.*

And without forgeting the jury members or the whole department of electrical engineering and industrial computing.

Finally, thanks to everyone in my promotion team and everyone who has helped us both morally and materially.

Contents

Dedication . . . . . **I**

Acknowledgment . . . . . **III**

List of Figures . . . . . **VIII**

List of Tables . . . . . **IX**

List of Acronyms . . . . . **X**

General introduction **1**

I PLC's Fundamentals **4**
- I.1 Introduction 5
- I.2 PLCs Brief History 5
- I.3 PLC: Programmable Logic Controller 6
- I.4 PLCs Architecture 7
  - I.4.1 Compact Design 7
  - I.4.2 Modular Design 7
  - I.4.3 PLC hardware 7
- I.5 PLC Internal architecture 8
  - I.5.1 Power supply 9
  - I.5.2 Central Processing Unit (CPU) 10
  - I.5.3 Memory 10
  - I.5.4 Input/Output interface 11
  - I.5.5 Sourcing and Sinking 11
- I.6 PLC function cycle 12
- I.7 Things to consider before buying a PLC 14
- I.8 Comparison between the leading PLC brands 15
  - I.8.1 Programmable Logic Relay 15
  - I.8.2 Programmable logic controllers 15
- I.9 Conclusion 18

II PLC's programming languages **19**
- II.1 Introduction 20
- II.2 IEC 61131-3 standard 20
  - II.2.1 Goals and benefits of the standard 20

II.2.2 Manufacturers (PLC hardware and software) . . . 20
II.2.3 Users . . . 21
II.3 The Program Organisation Unit (POU) . . . 21
II.4 The Programming Languages of IEC 61131-3 . . . 22
II.4.1 Ladder Diagram LD . . . 23
II.4.2 Sequential Function Chart SFC . . . 28
II.4.3 Function Block Diagram FBD . . . 32
II.4.4 Structured Text ST . . . 33
II.4.5 Instruction List IL . . . 36
II.5 Conclusion . . . 39

III Architecture design . . . 40
III.1 Introduction . . . 41
III.2 PLC Hardware architecture . . . 41
III.2.1 CPU choice . . . 42
III.2.2 Serial communication protocols . . . 45
III.2.3 Isolation methods . . . 47
III.2.4 Power supply . . . 49
III.2.5 Communication module . . . 49
III.2.6 PLC software environment . . . 51
III.2.7 PLC programs . . . 52
III.2.8 User program . . . 52
III.2.9 Integrated Development environment IDE . . . 53
III.3 Conclusion . . . 59

IV Results and discussion . . . 60
IV.1 Introduction . . . 61
IV.2 First version prototyping . . . 61
IV.2.1 4 Channel 5V Optical Isolated Relay Module . . . 61
IV.2.2 Digital I/Os . . . 63
IV.2.3 Analog I/Os . . . 67
IV.2.4 Pulse-Width Modulation PWM . . . 72
IV.2.5 Real-Time Clock (RTC) . . . 74
IV.2.6 Serial communication . . . 78
IV.3 PLC final version . . . 80
IV.4 Conclusion . . . 81

General conclusion . . . 82

Appendix . . . 88

List of Figures

I.1 Compact PLCs. 7
I.2 Modular PLCs. 7
I.3 PLC hardware.[21] 8
I.4 PLC internal architecture.[19] 9
I.5 CPU architecture.[13] 10
I.6 Sink logic.[39] 12
I.7 Source logic.[39] 12
I.8 Inputs Acquisition[9] 13
I.9 Processing[9] 13
I.10 outputs assignment[9] 14

II.1 Evolution from previous block types (e.g. German DIN 19239) to the POUs of IEC 61131-3[47] 21
II.2 Call of the standard FB CTU in an LD network[47] 28
II.3 Instruction in IL[47] 37

III.1 PLC hardware architecture 42
III.2 STM32F407 Discovery board[50] 43
III.3 STM32F407 header[43] 45
III.4 I2C communication protocol[53] 46
III.5 SPI communication protocol[10] 47
III.6 Isolation components[38][15][6] 47
III.7 SSR DC input circuit[2] 48
III.8 SSR AC input circuit[8] 48
III.9 SSR output circuit 49
III.10 SWD module[45] 50
III.11 STlink module[49] 51
III.12 PLC software architecture 51
III.13 The LDmicro architecture and deployment process 53
III.14 LDmicro interface 54
III.15 STM32CubeIDE interface 55
III.16 Putting LDmicro and STM32CubeIDE together 55
III.17 Mcutable files 56
III.18 Generating build files 57
III.19 Build LDmicro application files 57
III.20 Building our new version using MS VS2013 58
III.21 LDmicro updated version 58

III.22 New MCU added successfully 59

IV.1 4 relays module[3] 61
IV.2 4 relay module schematic.[4] 62
IV.3 PLC first version prototyping. 63
IV.4 Digital I/Os test LADDER program. 64
IV.5 LED test result case 1. 64
IV.6 LED test result case 2. 65
IV.7 20x4 LCD screen with integrated I2C interface.[17] 65
IV.8 LCD test LADDER program. 66
IV.9 LCD test result. 67
IV.10 LM35 pinout.[20] 67
IV.11 LM35 circuit configuration.[20] 68
IV.12 LM35 temperature reading LADDER program. 69
IV.13 YwRobot motherboard v2 power supply module.[23] 69
IV.14 LM35 temperature reading result. 70
IV.15 Timer interruption test LADDER program. 71
IV.16 Timer/Counter test result. 72
IV.17 Ideal PWM signal.[18] 72
IV.18 PWM application setup. 73
IV.19 PWM application LADDER program. 73
IV.20 PWM application result. 74
IV.21 RTC Pin Assignment.[5] 74
IV.22 RTC typical operating circuit.[5] 75
IV.23 RTC application setup. 76
IV.24 Date/Time adjusting code (start). 76
IV.25 Date/Time adjusting code (end). 77
IV.26 Month/Year adjusting. 77
IV.27 2 STM32 serial communication. 78
IV.28 Serial communication LADDER program. 78
IV.29 Serial communication result. 79
IV.30 PLC final prototype. 80
31 MOVE function. 89
32 COMPARE function. 89
33 CIRCULAR COUNTER function. 90
34 LOOK-UP TABLE function. 90
35 A/D function. 91
36 UART (SERIAL) RECEIVE function. 91
37 UART (SERIAL) SEND function. 92
38 UART (SERIAL) SENDn function. 92
39 I2C READ. 92
40 I2C WRITE. 93
41 Compile into a C code. 93
42 Generating the .hex file. 94

43 .hex file generated. 94
44 Program implementation. 95
45 General MCU info. 96
46 MCU I/O pins infos. 96
47 SPI/I2C pins infos. 97
48 PWM pin info. 97
49 ADC pins/channels info. 98
50 New MCU structure. 98
51 I2C protocol-based 8-bit I/O port expander.[17] 99
52 I2C interface module address. 99
53 relevent Ldmicro I2C LCD commands. 100
54 cursor commands for an 20x4 LCD screen in hexadecimal. 100

List of Tables

I.1 Programmable Logic Relays[33][29] 15
I.2 Siemens's PLC characteristics 16
I.3 Siemens's PLC characteristics continued[34][36][37][35] 17
I.4 Schneider Electric's PLC characteristics 17
I.5 Schneider Electric's PLC characteristics continued[26][27][28] 18

II.1 Connections in Ladder Diagram[47] 24
II.2 Ladder Diagram: contacts[47] 25
II.3 Ladder Diagram: coils[47] 26
II.4 LD: Graphical objects for execution control[47] 27
II.5 Possible SFC sequences 1[47] 29
II.6 Possible SFC sequences 2[47] 30
II.7 Possible SFC sequences 3[47] 31
II.8 Connections in FBD[47] 32
II.9 Graphical elements for execution control in FBD[47] 33
II.10 ST statements[47] 35
II.11 Operators of ST as elements of expressions incl. precedence rule[47] 36
II.12 Modification of the CR by different groups of operators[47] 37
II.13 Operators with Boolean and Bit String operands (Type ANY BIT)[47] 38
II.14 Operators for operands of generic numeric data type (type ANY NUM)[47] . . 39

III.2 Commonly used variables on LDmicro. 54

List of Acronyms

AC	*Alternative Current*
ALU	*Arithmetic–Logic Unit*
CPU	*Central Processing Unit*
CAN	*Controller Area Network*
DC	*Direct Current*
DB	*Data Block*
DFU	*Device Firmware Update*
DSP	*Digital Signal Processor*
ERP	*Enterprise Resource Planning*
FB	*Function Block*
HMI	*Human-Machine Interface*
HAL	*Hardware Abstraction Layer*
IDE	*Integrated Development environment*
I/O	*Input/Output*
I2C	*Inter Integrated Circuits*
IEC	*International Electrotechnical Commission*
LCD	*Liquid Crystal Display*
LED	*Light Emitting Diode*

MCU *Microcontroller Unit*

MES *Manufacturing Execution System*

OB *Organisation Block*

PWM *Pulse Width Modulation*

PLC *Programmable Logic Controller*

PC *Personal Computer*

POU *Program Organisation Unit*

PB *Program Block*

RAM *Random Access Memory*

SB *Sequence Block*

SPI *Serial Peripheral Interface*

SSR *Solid State Relay*

SWD *Serial Wire Debug*

SCADA *Supervisory Control and Data Acquisition*

UART *Universal Asynchronous Receiver-Transmitter*

USART *Universal Synchronous/Asynchronous Receiver/Transmitter*

USB *Universal Serial Bus*

General introduction

General introduction

In the industrial field, where competition is fierce, companies are faced with a requirement to improve the quality and quantity of their products while minimizing prices and expenses. To do this, they must have flexible and efficient production lines with less human intervention. This is how the idea of automated processes in the industry has immersed. To achieve this task, a Programmable Logic Controller was introduced.

The programmable logic controllers (PLC) appeared in the USA around 1969 in the automobile assembly line. After it becomes a worldwide industrial revolution. A PLC is a programmable electronic machine, able to pilot a system with a simple and easy programming language with a very precise response time and in real time.

Therefore, the communication is closed and restricted to the devices belonging to the same manufacturer. Within the same producing environment, after having controllers from totally different manufacturers, communication becomes a problem. The issue in connecting the devices interferes directly with the performance of the activity to be controlled since, without correct communication, the system loses the power to spot the tasks to be performed on the assembly chain.

The development of a platform that gives open communication is fundamentally necessary for the advancement in keeping with the new technological reality. Enable any device integrated with the network to communicate seamlessly clear by proprietary protocols is crucial to realize an extremely integrated management surroundings that may be enforced unimpeded from the communicative perspective of the system. Starting native PLC production in Algeria is a strategic approach to extend productivity and adaptability of automation field corporations whereas decreasing prices and enabling quicker and better quality prognostic maintenance. Furthermore, Algerian companies will afford a lot of productive time and cash by eliminating waiting time for the shipping and maintenance team costs.

Our work consists in realizing a prototype of an open source hardware and software ecosystem in order to facilitate and accelerate the development process and eventually the realization of a PLC using an STM32 board that will be compliant with the requirements of Industry 4.0. This is referred to as a programmable logic controller designed on the basis of programmed logic. The latter will be able to carry out most of the applications which will be asked to him by simple modification of the software and by keeping the same material structure. Our PLC will answer globally to the same requirements expected from a classical PLC, in particular the possibility of being programmed and reprogrammed with a non structural language via a compiler.

Our work is organized into four chapters:

Chapter I: represents a global view on the programmable logic controllers and their operating method, with a brief comparison between the famous ones.

Chapter II: will be about PLC's programming languages and the IEC 61131-3 standard.

Chapter III: to determine the design of the important parts of our PLC and the software

study of the latter (representation of the programming compiler of the PLC).

Chapter IV: will be about results and discussions.

Finally, we will end this thesis with a general conclusion and perspectives.

Chapter I

PLC’s Fundamentals

I.1 Introduction

Industrial systems are becoming more and more complex, and this development is accompanied by an evolution of the automation process.

Indeed, between 1950-1970, thanks to the progress of electronics and the efforts of electronics engineers, the first major technological revolution in the world broke out in the industrial field. This technology brought an industrial revolution in the way of organizing the control of a process. Its design increases the reliability of systems and allows them to be more adapted to environment and the demands of the present time.

The PLC[1] is the basic automation device and the heart of every industrial automation system. This chapter aims to describe and define the PLC, first we define the PLC and the different types known, then we give the structure and the internal architecture of a PLC. We define the principle of operation of a PLC and also talk about programming languages and the advantages that express its importance in the industry.

I.2 PLCs Brief History

Over the past decades, new technological advancements have continued to emerge to improve manufacturing processes and bring automation to factories. Over time, computers began to play a vital role in the industry. Innovations emerged that we're able to revolutionize manufacturing. One of these is the programmable logic controller, an industrial computer capable of controlling manufacturing processes, from assembly lines to fault diagnosis.

As the complexity of the system increased, so did the need for a more advanced control system. Wired relays and switches were among the components of this system. This was fine for a machine that was never modified or updated, but as manufacturing techniques advanced and the plant's transition to new products became more interesting and essential, a more flexible method of controlling this equipment was needed. Changing the wired relays and transfer logic was impractical and time consuming. To accommodate the new control system, the wiring had to be stripped and replaced. This modification was difficult and time consuming to design and install.

The first PLC was developed for the automotive industry, that pioneered this technology. the development of PLCs began within the late sixties, around 1968. it absolutely was developed at a call for participation from the yank manufacturer General Motors. The goal was to interchange relay management systems and hard-wired timers, which were changing into too tough to manage, with programmable controllers that offered rather more flexibility. Since then, PLCs are used wide as extremely reliable automation controllers thanks to their ability to more harsh environments. Before the appearance of programmable logic controllers, factories used cam timers, relays, closed-loop controllers, and sequencers as control logic for producing. If this system worked, it absolutely was extraordinarily rigid associate degreed tough to change. These systems were hardwired, that created it terribly difficult to form any changes to the man-

[1] *Programmable Logic Controller*

ufacturing process. If a retardant arose on an assembly line, for example, or if changes required to be made, it'd involve rewiring and change the documentation. Worst of all, it was enough to place a wire within the wrong place for the system to fail, leading to typically pricey production delays. employees had to waste hours comparison existing wiring to schematics in an effort to seek out the fault, which was counterproductive.

With inexpensive microprocessors becoming available in the late 1970s, PLC manufacturers began developing graphical programming devices. These devices enabled PLCs to be digitally programmed with graphical representations for normally open and closed relay contacts, push-buttons, switches, relay coils, motor starters, valves, timers, and more. Unfortunately, these programming devices were prohibitively expensive, often costing more than the PLC they were designed to program. The programming devices were also only compatible with a single manufacturer's controllers, further reducing their utility. Third-party software packages that worked with multiple PLC hardware platforms came available in the 1980s to alleviate the high cost of early programming devices. The 1980s also saw the introduction of the IEC[2] 61131-3 Standard for Programmable Controllers, which is the standard all PLC software is held against. The IEC brings consistency to PLC software programs by defining various standards for data types, naming conventions, program flow, and other programming elements.

In the 1990s, manufacturers began requesting for machinery to include interactive terminals that allowed them to monitor their PLC software. In addition, the terminals would allow operators to quickly see what was wrong with their machines preventing them from having to spend hours troubleshooting. This idea led to the development of the HMI[3].

By the end of the 1990s, PLCs were becoming smaller, faster, more powerful, and equipped with significantly more memory. As the world moved into the twenty-first century, these advancements in PLC technology would help pave the way for manufacturers to implement a new era of internet connectivity, big data, and even forms of artificial intelligence on the factory floor.[7]

I.3 PLC: Programmable Logic Controller

A PLC is an electronic machine of a particular form of microprocessor control, programmable by a non-computer user. It uses a memory to store the instructions composing the various functions of automation intended to control, in real time and in an industrial environment, sequential and combinational logic processes.

[2] *International Electrotechnical Commission*
[3] *Human-Machine Interface*

I.4 PLCs Architecture

I.4.1 Compact Design

These PLCs integrate the CPU[4], power supply, inputs and outputs in a single box (rack). Depending on the model and manufacturer, it can perform certain additional functions (counting, analog I/O[5] ...) and receive a limited number of extensions. They are generally intended for the control of small automation processes.



Figure I.1: Compact PLCs.

I.4.2 Modular Design

The PLC is of the modular type contains a rack, a power supply module, a CPU, I/O modules, communication and counting modules. This modular organization allows a great flexibility of configuration for user's needs, as well as easy diagnosis and maintenance, and is designed for complex automation applications where power, processing capacity and flexibility are required.



Figure I.2: Modular PLCs.

I.4.3 PLC hardware

The Hardware components of a PLC include the following:

Processor, Power Supply, Input/Output Modules, and a Programming Device.

[4] *Central Processing Unit*
[5] *Input/Output*

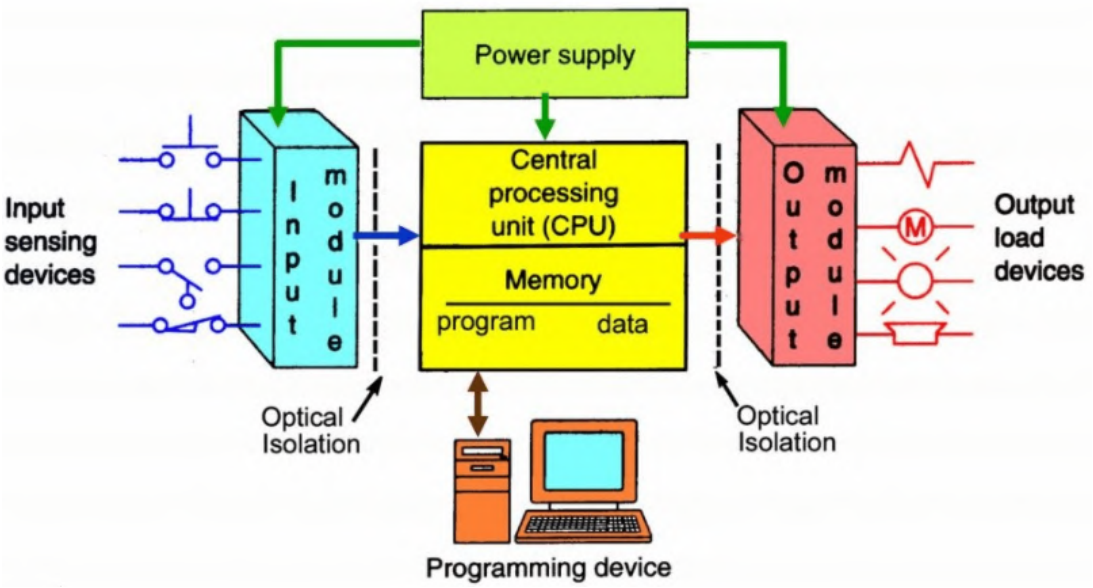


Figure I.3: PLC hardware.[21]

- The Power Supply is connected to AC mains for the supply voltage. The Power Supply output is a DC voltage used to power all of the other modules associated with the PLC. The Power supply DOES NOT provide power for field devices.
- The input/output modules are connected to digital or analog field devices. Input field devices include switches, encoders, and transmitters for example. Typical output field devices are relays, lamps, and proportional valves.
- The Processor consists of the CPU or central processing unit and memory. The processor section makes decisions needed to observe and operate the field devices connected to the Input/Output modules. The decisions are based upon a user-created program saved in the memory. The memory also stores data representing the condition of all input field devices and contains the data telling the output field devices what to do.
- The Programming Device in today's industrial applications is usually a laptop or a desktop computer that facilitates the creation of decision-making programs destined for the PLC.[24]

I.5 PLC Internal architecture

Machine Microprocessor, memory and input/output circuits are located in the central processing unit (CPU). All PLC operations are managed and processed by the CPU. It comes with a clock that runs between 1 and 8 MHz. PLC specifies this frequency Speed of operation and timing and synchronization of all equipment components. Digital signals are used to transmit information within the PLC.

Buses are the internal paths from which digital signals pass. A bus is simply a set of conductors through which electrical signals can travel.

The Data bus is used by the CPU to transmit data between the constituent components. The address bus is used to send location addresses for accessing stored data. The control bus is used to send signals relating to internal control actions. Finally, the device bus is used for communication between the input/output ports and the input/output unit.[19]

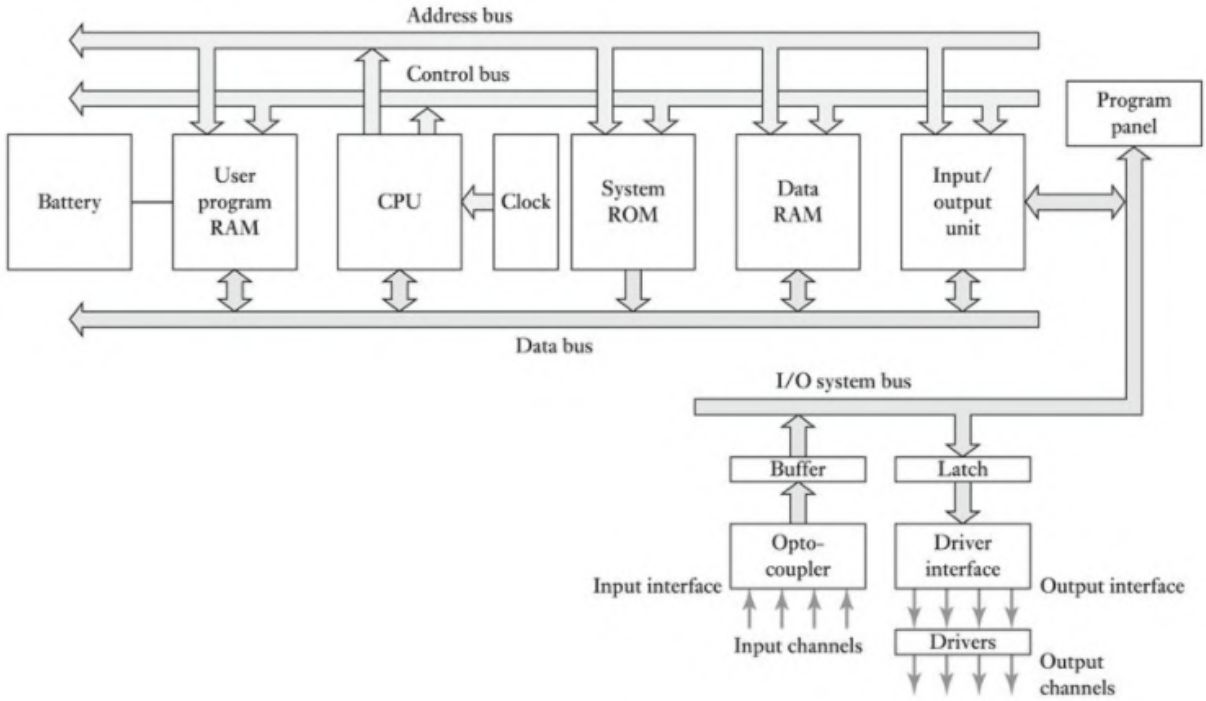


Figure I.4: PLC internal architecture.[19]

I.5.1 Power supply

It provides the energy necessary for the proper functioning of the PLC. The main voltages used in a PLC are +12v and ± 5v adapted to the operation of the internal electronic cards. To be able to act directly on the OP in case of failure, some power supply modules are equipped with a contact and an indicator. The power supply module has a reserved place in the main rack.[9] We distinguish:

- the AC[6] power module which supplies the necessary energy from the 220V mains.
- the DC[7] power supply module which provides the necessary energy from external power supplies (24 or 48V).

The choice of the power supply module for a PLC is based on its configuration and an energy configuration and an energy balance of the consumption of the installed couplers.

[6] *Alternative Current*
[7] *Direct Current*

I.5.2 Central Processing Unit (CPU)

A central processing unit (CPU), also called a central processor, main processor or just processor, is the electronic circuitry that executes instructions comprising a computer program. The CPU performs basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions in the program. This contrasts with external components such as main memory and I/O circuitry.

Principal components of a CPU include the ALU[8] that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that orchestrates the fetching (from memory), decoding and execution of instructions by directing the coordinated operations of the ALU, registers and other components.[9]

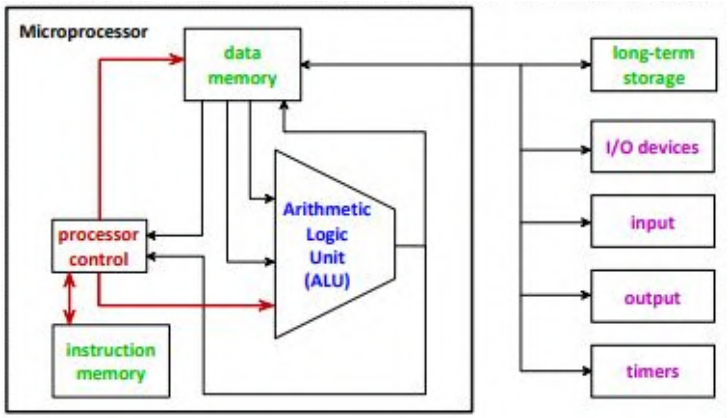


Figure I.5: CPU architecture.[13]

I.5.3 Memory

Program and data memory in a PLC is contained in the RAM[9]. This type of memory may be volatile or non-volatile, and it can (and will) be overwritten often. The program itself is in one area of RAM and must be kept in memory even when the PLC is powered off. PLC Memory consists of the operating system and firmware of the processor and modules, and the program and data that is used by the programmer. In the previous hardware section, it was explained that there are volatile and non-volatile areas of memory, and that the volatile part of memory needs a battery, "super capacitor" or other rechargeable energy storage module to hold its program and/or data. The RAM part of memory in a PLC can be separated into two general areas: **Program Memory** and **Data Memory**. [9]

- **Program memory** consists all of the lists of Instructions and Program Code. This is what is sent to the processor. The act of sending the program instructions to the PLC is called "downloading" on most brands of PLC, however this may differ on some platforms.

[8] *Arithmetic–Logic Unit*
[9] *Random Access Memory*

- **Data memory** includes the Input and Output Image Tables as well as Numerical and Boolean Data. Most of the data used in the PLC program is internal memory and not directly related to I/O.

I.5.4 Input/Output interface

PLC I/O is the part of the PLC that connects the brain of the PLC, the CPU, to the outside world, the machines.

In a PLC system there will usually be dedicated modules for inputs and dedicated modules for outputs. An input module detects the status of input signals such as push-buttons, switches, temperature sensors, etc... An output module controls devices such as relays, motor starters, lights, etc.

- **Discrete I/O:** The most common type of PLC I/O is discrete I/O. Sometimes discrete I/O is referred to as digital I/O. The concept is simple, discrete I/O are signals that are either on or off. Typical system designs utilize 5 V, 12 V or 24 VDC because the PLC in generale can interface any DC voltage signal between 3 and 30 VDC. Any connection to ground (or voltage below 1.5 VDC) or an open connection is sensed as '0' and any affiliation to DC voltage between 1.5 and 30 VDC is sensed as a '1'.

  For digital inputs that are AC signals, the ports can be connected to Velocio Optocoupled Input Terminal Block modules that convert the AC signals to the right DC levels.

- **Analog I/O:** The other common form of PLC I/O is analog I/O. Analog I/O refers to signals that have a range of values much greater than just 1 or 0. For instance, an analog signal could produce a voltage anywhere in the range of 0 – 10 VDC. The signal could be 2 V, 3 V, 8.5 V, etc. In the PLC world, analog input modules usually measure analog inputs in one of the following forms: -10 to 10 VDC, 0 – 10 VDC, 1 to 5 VDC, 0 to 1 mA, or 4 – 20 mA. Basically, the analog input module either measures voltage or current from the input device. There are other types of analog signals but these are definitely the most common. Similarly, the analog output module can supply voltage or current signals in one of the ranges I mentioned previously.[22]

I.5.5 Sourcing and Sinking

1. **Sink logic:** For a sink logic, a **NPN transistor** provides a path to ground for the electrical load. For an NPN transistor circuit to work, it must be connected to a PNP transistor circuit. In other words, a sink logic circuit needs to be connected to a source logic circuit.

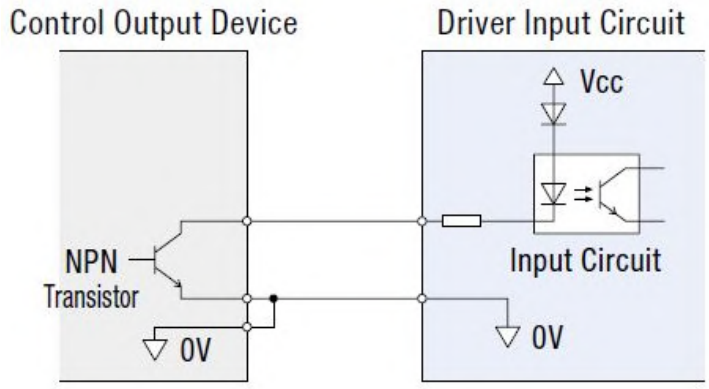


Figure I.6: Sink logic.[39]

2. **Source logic:** For a source logic, a **PNP transistor** provides a path to voltage for the electrical load. For a PNP transistor circuit to work, it must be connected to a NPN transistor circuit. In other words, a source logic circuit needs to be connected to a sink logic circuit.[39]

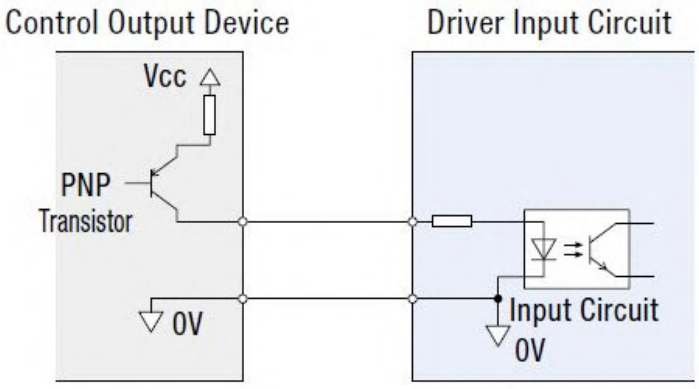


Figure I.7: Source logic.[39]

I.6 PLC function cycle

The PLC cycle can be broken down as follows below. Such a **polling cycle** will allow elementary programming. Although limited in power, this method avoids a good number of hazards, and the writing of the program in synchronous form will ensure a conforming implementation of the GRAFCET. It should be noted that the assignment of internal variables is carried out as they are processed in the program.

The various PLCs on the market do not all have the same type of cycle. The essential difference comes from the system's input/output acquisition policy. The distribution of the intelligence in the multiprocessor structure also intervenes.[9]

1. **Inputs aquisition :** Read all inputs from all input's couplers and, via the I/O manager, copy them to data memory in the input image area.

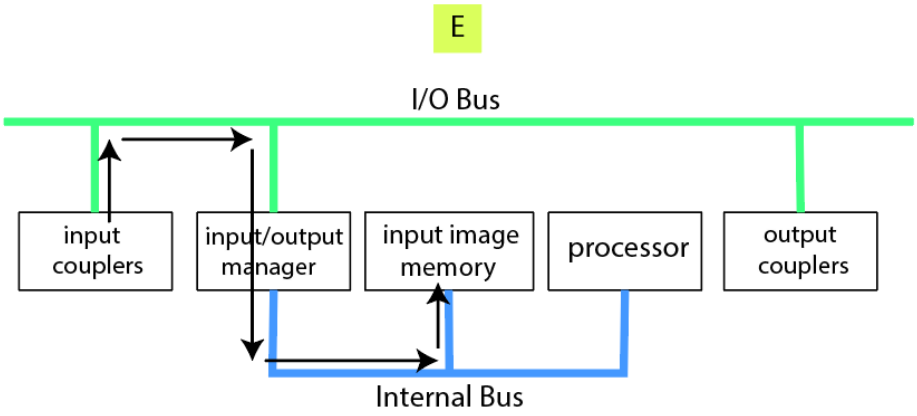


Figure I.8: Inputs Acquisition[9]

2. **Processing:** Process the equations of the application program from the stored inputs and:
   - For the output, write the values in the image area of the outputs
   - For the internal variables, write the values in the corresponding zone.

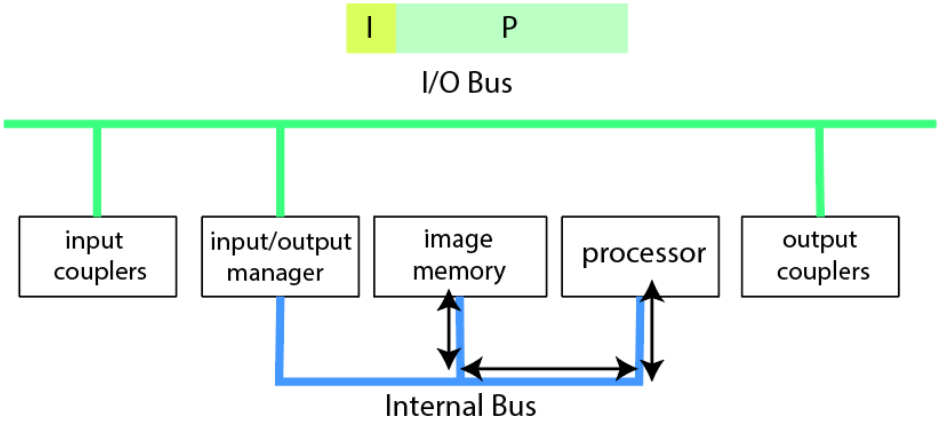


Figure I.9: Processing[9]

3. **Outputs assignment:** Read all outputs values in the output image area and, via the I/O manager, copy all outputs to all output's couplers.

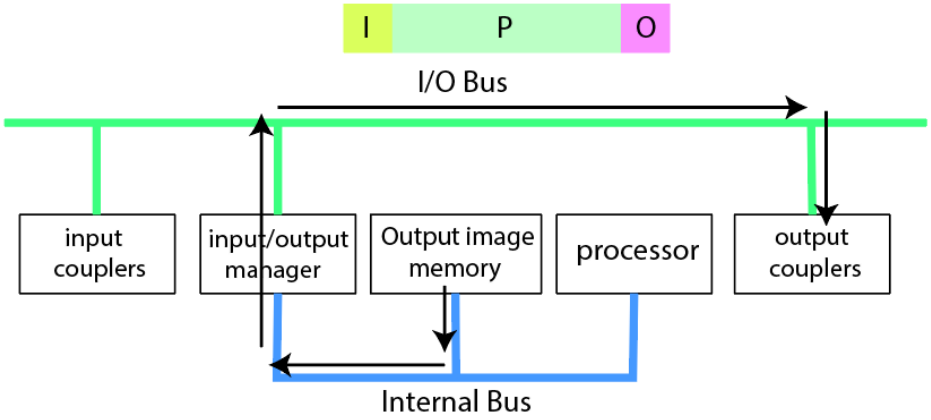


Figure I.10: outputs assignment[9]

I.7 Things to consider before buying a PLC

There are many considerations that should be taken into account to help narrow down the options for a particular application [46]:

- **CPU speed** – how big is the system and what response rate does the process require?
- **Memory capacity** – how much memory is enough? Will an external memory card be required?
- **Redundancy** – is any level of redundancy required?
- **I/O** – how many devices does it need to control or monitor? Is there a requirement to work with external or remote I/O interfaces?
- **Communication protocols** – what devices will it need to communicate with and what protocols are used?
- **Compatibility** – is the system being built from scratch, or does it need to interface with existing hardware and software?
- **Scalability** - can the system be scaled up or down to accommodate changes with the system?
- **Environment** what type of environmental conditions will it be subjected to? Will it require any special considerations such as conformal coated components, or a more robust option for harsh environments?
- **Actions performed** does it need to be able to perform any unique actions, such as motion control, robotics or safety?
- **Reliability** is the system expected to run with little or no downtime for extended periods of time, sometimes years?

- **Cost** what is the budget? Have things been considered collectively as a 'system', such as the purchase of hardware, software licenses, programming, installation, ease of interface with existing plant/systems, maintenance, and back-up support along with any future upgrade considerations?

I.8 Comparison between the leading PLC brands

We must emphasize that comparing all of the CPUs (Central Processing Units) of each PLC model is not possible. This is due to the fact that its utilization is essentially dependent on the application for which it is required, and there are several CPU variants available for each model. As a result, we will concentrate on exposing the special characteristics and needs of PLC models equipped with a certain CPU from the industrial range.

I.8.1 Programmable Logic Relay

We shall draw a link between the simplest programmable logic devices: the LOGO and Zelio Programmable Logic Relays produced by the firms under consideration.

Characteristics	**Model LOGO230RCE**	**Model Zelio Logic SR3BB101FU**
Display	Yes	Yes
Micro SD cards	Yes	No
Ethernet interface	Yes	No
Integrated web server	Yes	No
Supply voltage	**Nominal value (DC):** 115VDC and 230VDC **Nominal value (AC):** 115VAC and 230VAC	**Nominal value (DC):** 115VDC and 230VDC **Nominal value (AC):** 115VAC and 230VAC
Digital inputs	8	6
Digital outputs	4-Relay	4-Relay
Index Protection	IP20	IP20
Ambient conditions	-20°C a 55°C	-20°C a 40°C

Table I.1: Programmable Logic Relays[33][29]

I.8.2 Programmable logic controllers

Before choosing a PLC, it is critical to understand several key elements such as software, the programming language we will use, and the number of signals we will utilize to properly size the input-output and communications modules.

Siemens PLCs

we will provide the SIEMENS PLC models in increasing order depending on power and production goal.

Characteristics	SIMATIC S7-1200 Basic Controller	SIMATIC S7-300 Advanced Controller	SIMATIC S7-400 Advanced Controller	SIMATIC S7-1500 Advanced Controller
Technique reference	CPU 1211C AC/DC/Relay	CPU 314	CPU 412-2 PN	CPU 1516-3 PN/DP
Programming package	STEP 7 V14 or superior	STEP 7 V5.5 or higher + SP1 or STEP 7 V5.2 and higher + SP1 with HSP 218	STEP 7 V5.5 or higher with HSP 262	TIA PORTAL STEP 7 Professional V13 SP1 update 4 or higher
Supply voltage	120 VAC or 230 VAC	24 VDC	24 VDC	24 VDC
Work memory	50 Kbyte	128 Kbyte (Non-expandable)	1 Mbyte	5 Mbyte
Load memory	1 Mbyte (Integrated)	8 Mbyte (MMC)	512 Kbyte	32 Gbyte (Only with memory card)
CPU run times (Depending on the type of operation	0.08µs to 2.3µs	0.1µs a 1.1µs	31.25 ns to 62.5ns	10 ns a 64 ns
Digital inputs	6(3 usable for High-Speed Counting)			
Digital outputs	4-Relay			
Analog inputs	2			
Number of Ports	1xPROFINET /Ethernet		1xMPI/ PROFIBUS_DP 2xPROFINET	X1:2(Integrated Switch y RJ45) X2: 1(RJ45) X3: 1(RS485)
web server	Yes	Yes	Yes	Yes
Ambient conditions	0°C a 60°C	0°C a 60°C	0°C a 60°C	0°C a 60°C

Table I.2: Siemens's PLC characteristics

Inteface	X1:RJ45/ PROFINET	RS485	X1:RS485/ PROFIBUS+ MPI X2:Ethernet RJ45/ PROFINET	
Communication protocol	TCP/IP, PROFINET IO-Controller, PROFINET IO-Device, SIMATIC Communication.	MPI		IPv4, PROFINET IO-Controller, PROFINET IO-Device, SIMATIC Communication, PROFIBUS DP
Supported Languages	KOP, FUP, SCL.	KOP, FUO, AWL, SCL, CFC, GRAPH, HiGRAPH.		LAD, FBD, STL, SCL, GRAPH

Table I.3: Siemens's PLC characteristics continued[34][36][37][35]

As shown in the table above, we have broken down the primary characteristics and operational objectives of each of the industrial models manufactured by SIEMENS.

Schneider Electric PLCs

We will proceed to perform the same procedure of comparison with Schneider Electric equipment.

Characteristics	**MODICON M221 Logic Controller**	**MODICON M241 Logic Controller**	**MODICON M251 Logic Controller**
Technique reference	TM221CE40R	TM241CE40R	TM251MESE
Programming package	MachineStruxure, SoMachine Basic	MachineStruxure, SoMachine Basic	MachineStruxure, SoMachine Basic
Supply voltage	100 VAC or 240 VAC	100 VAC or 240 VAC	100 VAC or 240 VAC
Work memory	64 Mbyte	64 Mbyte	64 Mbyte
Load memory	128 Mbyte(integrated) Expandable to 2 GB with SD card	128 Mbyte(integrated) Expandable to 2 GB with SD card	128 Mbyte(integrated) Expandable to 2 GB with SD card

Table I.4: Schneider Electric's PLC characteristics

Characteristics	MODICON M221 Logic Controller	MODICON M241 Logic Controller	MODICON M251 Logic Controller
CPU run times	5µs to 10ms	5µs to 10ms	0.3ms to 0.7ms
Digital inputs	24	24	
Digital outputs	16-Relay	16-Relay	
Analog inputs	2 (0-10V)	0	
Number of Ports	1x USB 2.0 mini port, 1x Ethernet RJ45, 1x Serial Port RJ45	1x USB 2.0 mini port, 1x Ethernet RJ45, 1x Serial Port RJ45	1x USB 2.0 mini, 1x Serial RJ45, 1x Dual Ethernet 1 RJ45, 1x Ethernet 2 RJ45.
Inteface	Serial Port: RS232/RS485	1xSerial Port: RS232/RS485 1xSerial Port: RS485	Serial Port: RS232/RS485 X2:Ethernet RJ45/ PROFINET
Communication protocol	TCP/IP, Modbus RTU, SoMachine Network, USB, Ethernet	Ethernet/IP, Modbus TCP, SoMachine Network, USB	Ethernet/IP, Modbus TCP, CANopen, USB
web server	Yes	Yes	Yes
Ambient conditions	-10°C to 55°C	-10°C to 55°C	-10°C to 55°C
Supported Languages	IL, LD, SFC, ST, FBD	IL, LD, SFC, ST, FBD	IL, LD, SFC, ST, FBD

Table I.5: Schneider Electric's PLC characteristics continued[26][27][28]

As seen in the table above, Schneider Electric PLCs are mostly dedicated to basic and medium industrial operations, with little deployment in sophisticated manufacturing processes.

All of this implies that the German manufacturer continues to favor the balance, but it does not rule out Schneider as a viable option for tiny and basic applications with a limited budget. However, this is always a subject for the designer to consider.

I.9 Conclusion

In this chapter, we have introduced the fundamentals of programmable Logic Controllers and elaborate a brief comparison between the leading ones.

As a conclusion, ideally, We must remember that, in addition to the PLC, we will need power supply, connection modules, input-output extension modules, and other components to complete our project. The selection and the use of these will always be determined by the individual demands of our process sizing.

In the next chapter, we will introduce the PLC's programming languages according to the IEC 61131-3 standard.

Chapter II

PLC’s programming languages

II.1 Introduction

The term PLC is used to sit down with the technology as a whole, both hardware and software, and not just to the hardware architecture. The IEC 61131 programming languages are often used for programming classical PLCs, embedded controllers, industrial PCs and even standard PCs. the standard was established by working group SC65B WG7 (originally: SC65A WG6) of the international standardisation organisation IEC (International Electrotechnical Commission) that consists of representatives of various PLC manufacturers, software houses and users. [47]

II.2 IEC 61131-3 standard

The IEC 61131 standard summarizes the PLC requirements into several parts system. These requirements relate to PLC hardware and programming system. The standard contains two common concepts already used in PLCs programming and other new programming methods. IEC 61131-3 sees itself as a guide to PLC programming rather than a rigid set the rule of. Lots of defined details mean programming A system can be expected to implement only some, but not all, of the standards. PLC Manufacturers must record this quantity: if they want to comply standards, they must demonstrate in which parts they meet the standards.

This has the advantage that it's accepted as a guideline by most PLC manufacturers. Thus, IEC 61131-3 has made its approach to become the sole worldwide standard for PLC programming in recent years, it not only describes the PLC programming languages themselves, but also offers comprehensive concepts and guidelines for creating PLC projects.[47]

II.2.1 Goals and benefits of the standard

Due to the increasing complexity of PLC systems, the following costs:

- Training of application programmers.
- Create larger and larger programs.
- Implementation of increasingly complex programming systems.

The PLC programming system gradually keeps up with the trend of the mass software market PC[1] world. Here, in particular, cost pressures can be reduced Standardization, interoperability and re-usability.[47]

II.2.2 Manufacturers (PLC hardware and software)

Multiple manufacturers can collectively invest in multi-million-dollar software to provide the functionality required in today's marketplace. Basic software such as editors, excluding certain

[1] *Personal Computer*

parts such as market segment differentiation results, and from PLC hardware. Reduced significantly by purchasing finished products. Using already tested software can greatly reduce newly developed software, and the development cost of modern programming tools increases significantly due to the required functionality. Software components (provided only by some programming system manufacturers) or complete systems, "time to market" can be significantly.[47]

II.2.3 Users

Users often work simultaneously with PLC systems from different manufacturers. Up to now this has meant that employees have needed to take several different training courses in programming, whereas with IEC 61131-3-compliant systems training is limited to the finer points of using the individual programming systems and additional special features of the PLCs. This cuts down on the need for system specialists and training personnel, and PLC programmers are more flexible. The requirements of the standard ease the selection of suitable programming exchanged between different PLC systems in the foreseeable future, language elements and program structure are nevertheless similar among the different IEC systems.[47]

II.3 The Program Organisation Unit (POU)

IEC 61131-3 calls the blocks from which programs and projects are engineered Program Organisation Units (POUs). POU[2]s correspond to the program blocks, organisation blocks, sequence blocks and perform blocks of the traditional PLC programming world. One vital goal of the standard is to limit the variability and infrequently implicit meanings of block varieties and to unify and simplify their usage.

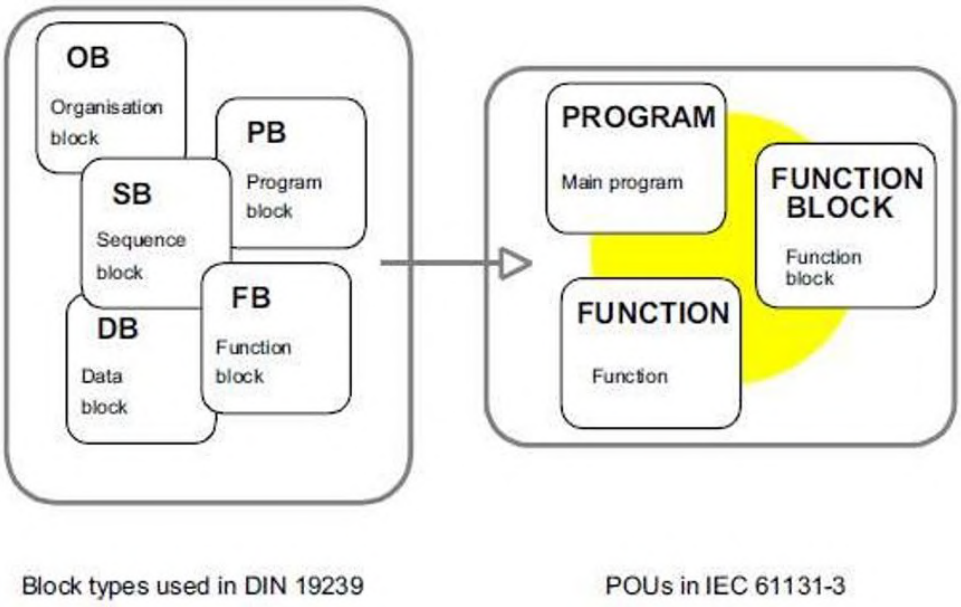


Figure II.1: Evolution from previous block types (e.g. German DIN 19239) to the POUs of IEC 61131-3[47]

[2] *Program Organisation Unit*

IEC 61131-3 reduces the different block types of PLC manufacturers to three unified basic types. Data blocks are replaced by FB data memories or global multi-element variables.

These three POU types differ from each other in certain features:

- **Function (FUN):** POU that can be assigned parameters, but has no static variables (without memory), which, when invoked with the same input parameters, always yields the same result as its function value (output).
- **Function block (FB):** POU that can be assigned parameters and has static variables (with memory). An FB (for example a counter or timer block), when invoked with the same input parameters, will yield values which also depend on the state of its internal (VAR) and external (VAR-EXTERNAL) variables, which are retained from one execution of the function block to the next.
- **Program (PROG):** This type of POU represents the "main program". All variables of the whole program that are assigned to physical addresses (for example PLC inputs and outputs) must be declared in this POU or above it (Resource, Configuration). In all other respects it behaves like an FB.

A POU is an encapsulated unit, that are often compiled severally of other program parts. However, the compiler needs information regarding the job interfaces of the other POUs that are known as within the POU ("prototypes"). Compiled POUs will be linked along later in order to create a complete program.[47]

II.4 The Programming Languages of IEC 61131-3

IEC 61131-3 provides three textual languages and three graphical languages for writing application programs. The textual languages are:

- Instruction list IL.
- Structured Text ST.
- Sequential Function Chart (textual version).

The graphical languages are :

- Ladder Diagram LD.
- Function Block Diagram FBD.
- Sequential Function Chart SFC (graphical version).

The code part of the textual language **ST!**[3] consists of a sequence of **statements** and the code part of the textual language IL of a sequence of **instructions**.[25] [47]

[3] **ST!**

II.4.1 Ladder Diagram LD

Ladder Diagram was originally modeled from relay-logic which used physical devices, such as switches and mechanical relays to control processes. Ladder Diagram utilizes internal logic to replace all, except the physical devices that need an electrical signal to activate them.

Ladder Diagram is built in the form of horizontal rungs with two vertical rails that represent the electrical connection on relay-logic schematics. We can program all the necessary input conditions to affect the output conditions, whether logical or physical.[25] [47]

Ladder Diagram Advantages

The main advantages of the Ladder Diagram language are:

1. The rungs allow it to be organized and easy to follow.
2. It also lets you document comments that are readily visible.
3. It supports online editing very successfully.

Ladder Diagram Disadvantages

The main disadvantage is that there are some instructions that are not available, which might make it more difficult for programming such as motion or batching.

Graphical objects in LD

A LD network consists of the following graphical objects:

- Connections
- Contacts and coils
- Graphical elements to control the execution sequence (jumps).
- Graphical elements to call a function or FB[4] (standard or user POU).
- Connectors

1. **Connections:** LD has horizontal and vertical lines to connect elements, and also includes crossing points.

[4] *Function Block*

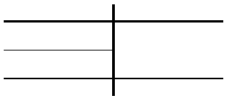
Graphical object	Name	Explanation
	Horizontal connection	The horizontal connection copies the value written on the left-hand side (FALSE or TRUE) to the right-hand side.
	Vertical connection including horizontal lines	A vertical connection evaluates all incoming lines (from left) with OR and copies this value to all connections on the right-hand side.

Table II.1: Connections in Ladder Diagram[47]

2. **Contacts and coils:** A contact performs a logic operation on the value of an incoming line and the value of the associated variable. The kind of logic operation depends on the type of the contact. The calculated value flows to the right connecting line.

   Contacts (open and closed) control the passing on of values in a network. The values passed depend on the value of the contact variables as well as their position in the network (the values of the contacts themselves are not changed!). Results are assigned to output variables by means of coils (values are assigned to the coil variables).

   There are two kinds of contacts: Normally open and Normally closed. A Normally open contact is like an electromechanical contact: when it is not activated (FALSE), it is open, and when it is activated (TRUE) it closes and power can flow. A Normally closed contact is the exact opposite: power flows when it is not activated (FALSE). It opens and interrupts the power flow when it is activated (TRUE)

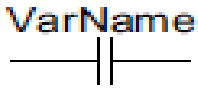
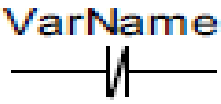
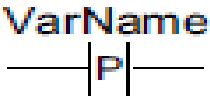
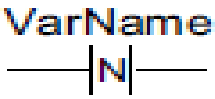
Graphical object	Name	Explanation
	Normally open contact	RightLink := LeftLink AND VarName, (Copy the status of the left link to the right link if the variable evaluates to TRUE, else FALSE).
	Normally closed contact	RightLink := LeftLink AND NOT VarName, (Copy the status of the left link to the right link if the variable evaluates to FALSE, else FALSE).
	Positive transition-sensing contact	RightLink := TRUE, if * the variable had the value FALSE during the last evaluation (cycle) and * the variable now has the value TRUE (current cycle) and * LeftLink has the status TRUE := FALSE otherwise. (Copy the value of the left link to the right link when the variable has changed from FALSE → TRUE. For all other combinations: copy FALSE to the right link).
	Negative transition-sensing contact	RightLink := TRUE, if * the variable had the value TRUE during the last evaluation (cycle) and * the variable now has the value FALSE (current cycle) and * LeftLink has the status TRUE := FALSE otherwise. (Copy the value of the left link to the right link when the variable has changed from TRUE → FALSE. For all other combinations: copy FALSE to the right link).

Table II.2: Ladder Diagram: contacts[47]

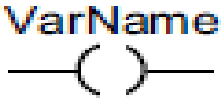
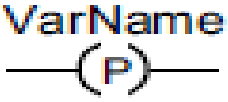
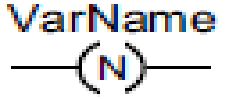
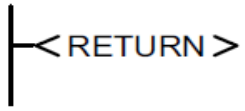
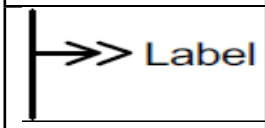
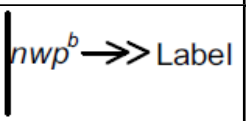
Graphical object	Name	Explanation
	Coil	Variable := LeftLink. (Copy the value of the left link to the variable).
	Negated coil	Variable := NOT LeftLink. (Copy the negated value (change FALSE <-> TRUE) of the left link to the variable).
	Set (latch) coil	Variable := TRUE, if LeftLink has the state TRUE, Else: do not change the variable.
	Reset (unlatch) coil	Variable := FALSE, if LeftLink has the state TRUE, Else: do not change the variable.
	Positive transition-sensing coil	Variable := TRUE, if * LeftLink had the status FALSE during the last evaluation (cycle) and * LeftLink now has the status TRUE (current cycle) Else: do not change the variable. (Copy the value of the left link to the variable when the left link has changed from FALSE →TRUE. Otherwise: do not change the variable.)
	Negative transition-sensing coil	Variable := TRUE, if * LeftLink had the status TRUE during the last evaluation (cycle) and * LeftLink now has the state FALSE (current cycle) Else: do not change the variable. (Write TRUE to the variable if the left link has changed from TRUE → FALSE. Otherwise: do not change the variable.)

Table II.3: Ladder Diagram: coils[47]

3. **Execution control:** To define the sequence of execution of POUs and their networks, LD offers two types of execution control: one to leave a POU and the second to jump to a specified network.

Graphical object	Name	Explanation
	(Unconditional) return	Leave the POU and return to the calling POU. Execution control returns automatically to the calling program when the end of a POU is reached.
	Conditional return	LeftLink of the RETURN element == 1: Stop the POU and return to the calling POU, Else: Ignore.
	(Unconditional) jump	Jump to the network labelled with the name "Label".
	Conditional jump	LeftLink of the jump element == 1: Jump to the network labelled with the name "Label", Else: Ignore.

b nwp represents a part of a network which supplies a Boolean value.

Table II.4: LD: Graphical objects for execution control[47]

4. **Call of functions and function blocks:** LD supports calls to POUs of type Function or FB. The called POU is represented by a rectangular box. FBs and functions can have several input and output parameters. Functions can have one or more return parameters. The names of the formal input and output parameters appear inside the box; extensible/overloadable functions do not have any formal parameter names. The appropriate actual parameters (variables, constants) are written on the connecting line outside the box adjacent to the formal parameter name on the inside. It is also possible to connect inputs directly with input or output parameters of other boxes (equivalent to FBD).

   The parameters of **function blocks** can be of any data type, but at least one input and one output parameter must be of type Boolean and these must have a direct or indirect connection to the left and right power rails. Standard FBs have the output Q to meet this requirement. The variable SetValue is assigned to the parameter PV. The value of ResVar is copied to the input R. Output CV is not connected. The Boolean result of the left nwp controls the transitionsensing input CU of Count. The output parameter Q is the start value of the right nwp (a Boolean part of an LD network).[47]

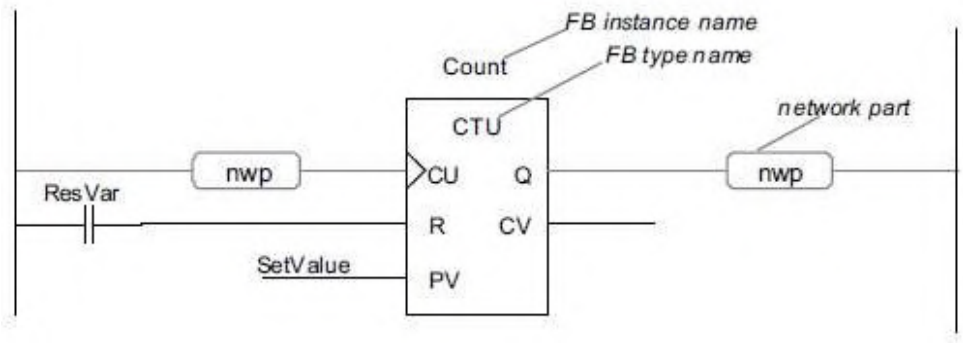


Figure II.2: Call of the standard FB CTU in an LD network[47]

II.4.2 Sequential Function Chart SFC

The IEC 61131-3 language SFC was defined to break down a complex program into smaller manageable units and to describe the control flow between these units. Using SFC, it is possible to design sequential and parallel processes. The timing of the execution of these units depends on both static conditions (defined by the program) and dynamic conditions (behaviour of the I/Os).

The units themselves are programmed in one of the other four IEC 61131-3 languages or in a further SFC structured description. The methodology of SFC has been derived from wellknown techniques like Petri-net or sequence (cascade) methodology.

Sequential Function Charts is similar to flowcharts, however in SFCs we use steps and transitions to achieve your end results. Steps act as a major function in your program. These steps house the actions that occur when you program them to happen. This decision can be based on timing, a certain phase of the process, or a physical state of an equipment. Transitions are the instructions that you use to move from one step to another step by setting conditions of true or false.

Unlike traditional flowcharts, the Sequential Function Charts can have multiple paths. we can use branches to initiate multiple steps at one time.[47]

Sequential Function Charts Advantages

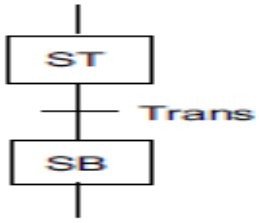
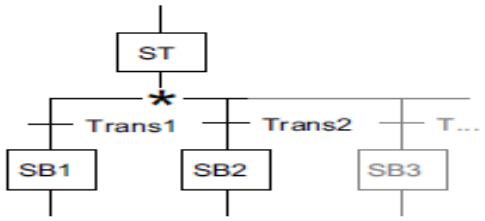
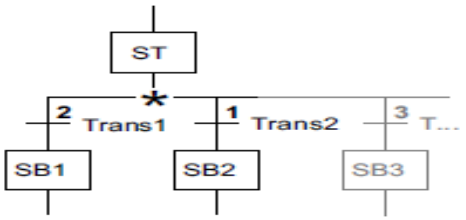
A couple of the advantages of Sequential Function Charts are:

- Processes can be broken into major steps that can make troubleshooting faster and easier.
- has direct access in the logic to see where a piece of equipment faulted.
- It can be faster to design and write the logic due to the ability to use repeated executions of individual pieces of logic.

Sequential Function Charts Disadvantages

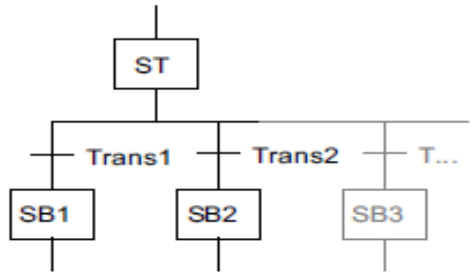
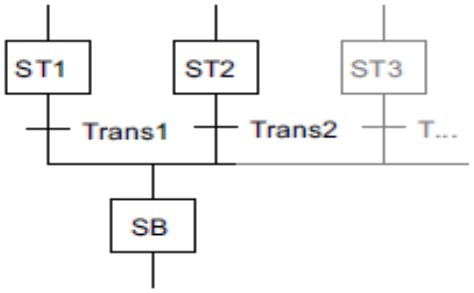
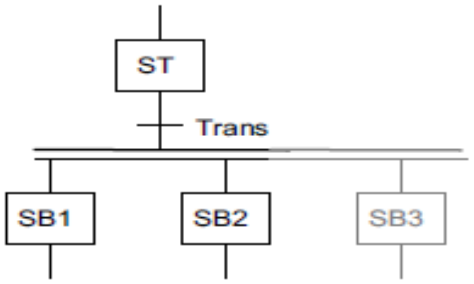
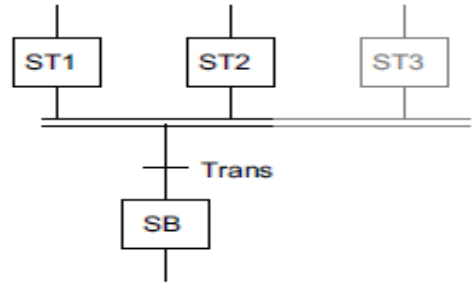
Even when you consider the advantages of the Sequential Function Charts, this PLC Programming Language does not always fit every application.[47]

- **Step - transition sequence :** Steps and transitions must always alternate in a network. It is not permitted to link two elements of the same type. A transition can have several predecessor and successor steps. A simple combination of steps and transitions is called a "Sequence". After executing a step, it is possible either to select one of several subsequent sequences (alternative sequences) or to activate several sequences at the same time (simultaneous sequences). The rules governing these sequences are described in the figure down below

Graphical object	Name and Explanation
	**Single sequence:** Alternation of step → transition → step in series. ST is deactivated, as soon as Trans evaluates to TRUE. SB becomes active, as soon as Trans evaluates to TRUE.
	**Divergent path:** Selection of exactly one sequence, evaluating from left to right. As long as ST is active, the transitions are evaluated from left to right (indicated by the use of the asterisk). The first transition with value TRUE stops the further execution of ST and activates the associated subsequent step.
	**Divergent path with user-defined priority:** This corresponds to the standard divergent path with the exception that the user can control the precedence of the transition evaluations. This is done by writing a number next to each transition; the lowest number has the highest priority and the priority decreases as the number increases.

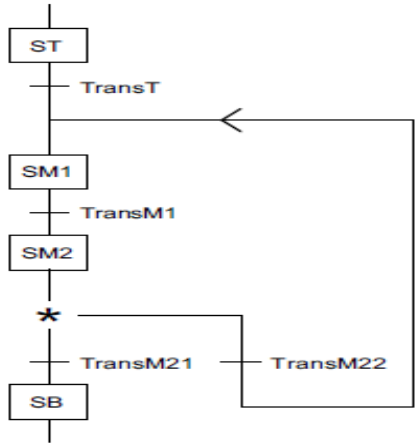
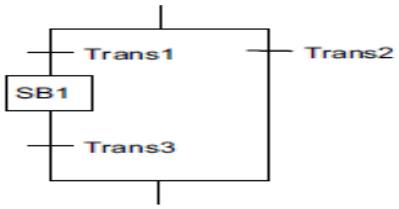
ST: Step at the Top of the transition
SB: Step Below the transition

Table II.5: Possible SFC sequences 1[47]

	**Divergent path under user control:** This corresponds to the standard divergent path with the exception that the transitions are evaluated without any precedence. The user must ensure that the transitions are mutually exclusive, i.e. that only one transition may become true in any given execution cycle. In this way, the program behaves deterministically.
	**Convergence of sequences:** The paths of divergence are combined. When one of STn is active and the corresponding successor transition condition becomes TRUE, the deactivation of step STn and the activation of SB are initiated.
	**Simultaneous sequences:** Simultaneous activation of all connected steps. ST is deactivated when Trans evaluates to TRUE and all subsequent steps connected via Trans become active. These newly activated sequences run simultaneously.
	**Convergence of simultaneous sequences:** The paths of simultaneous sequences are reunited. When all of the STn are active and the corresponding transition condition, Trans, evaluates to TRUE, all STn steps are deactivated and SB is activated.

ST: Step at the Top of the transition
SB: Step Below the transition

Table II.6: Possible SFC sequences 2[47]

	**Sequence loop (branch back):** A transition branches back to a predecessor step (see Example TransM22). The sequence selection is the same as described for divergent paths. In addition to the example shown on the left using the standard "*" mode, IEC 61131-3 defines two additional modes "user-defined priority" and "mutually exclusive" to indicate evaluation precedence. To improve readability, IEC 61131-3 allows the information flow to be indicated by an arrow. (See TransM22 to SM1).
	**Sequence skip:** This can be used with all three divergent path modes. It consists of no step and exactly one transition; this path is called a sequence skip (right path Trans2).

ST: Step at the Top of the transition
SB: Step Below the transition
SM: Step in the Middle

Table II.7: Possible SFC sequences 3[47]

II.4.3 Function Block Diagram FBD

The Function Block Diagram which is also a graphical type of language comes originally from the field of signal processing. The Function Block Diagram describes a function between inputs and outputs that are connected in blocks by connection lines.

Function Blocks were originally developed to create a system that you could set up many of the common, repeatable tasks, such as counters, timers, PID Loops, etc.

We program the blocks onto sheets and then the PLC constantly scans the sheets in numerical order, or is determined by connections which you program between the blocks.[47]

Function Block Diagram Advantages

FBD has the following graphical objects:

- Connections.
- Graphical elements for execution control (like JUMP).
- Graphical elements to call a function or function block (standard or userdefined), 4) Connectors **Connections**.

1. **Connections:** FBD uses horizontal and vertical connecting lines. It is possible to split a line into several connections.

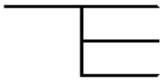
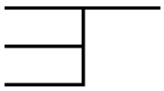
Graphical object	Name	Name Explanation
	Horizontal connection	The horizontal connection copies the value written on the left-hand side to the right-hand side.
	Vertical connection with horizontal connections.	The horizontal connection copies the value written on the left-hand side to all places on the right-hand side (splitting).
	forbidden connection ("wired OR")	

Table II.8: Connections in FBD[47]

2. **Execution control (jumps):** In order to influence program execution, there are calls for leaving the POU (returns) and for changing the sequence of processing networks (jumps to network labels).

Graphical object	Name	Name Explanation
1—⟨RETURN⟩ (equivalent representation)	(Unconditional) return	The standard does not define any specific graphical element. The user can program an unconditional return to the calling POU by using a conditional return with input parameter TRUE (equivalent "1"). Control is also returned to the calling POU when the end of a POU is reached.
nwp[b]—⟨RETURN⟩	Conditional return	Left link of the RETURN element == 1: Stop POU and return to the calling POU, Else: Ignore.
1 →> Lab	(Unconditional) jump	Jump to the network labelled with the name "lab"
nwp[b]—>> lab	Conditional jump	Left link of the RETURN element == 1: Jump to the network labelled with the name "lab", Else: Ignore.

b nwp defines a part of a network that supplies a Boolean output

Table II.9: Graphical elements for execution control in FBD[47]

3. **Call of functions and function blocks:** The graphical representations for calling POUs of types FB and function are similar. FB and functions can have several input and output parameters. The formal parameters are written inside the box; extensible/overloadable functions do not have any formal parameter names. An actual parameter is a variable name or constant written next to (outside) its formal parameter. The second method of passing information is to connect the input parameter with the output of another box.

   In FBD, it is **not** mandatory for FBs to have at least one Boolean input and output parameter. This requirement exists in Ladder Diagram. The return value of a function is determined by a value assignment to the function name. If several assignments to the function name are made, the last value assigned before leaving the function is valid.[47]

II.4.4 Structured Text ST

This language is a textual based language. Structured Text is a high-level language that is like Basic, Pascal and "C", because it does not use low level, machine-oriented operators but offers a large range of abstract statements describing complex functionality in a very compressed way. It is a very powerful tool that can execute complex tasks utilizing algorithms and mathematical functions along with repetitive tasks. The code uses statements that are separated by semicolons and then either inputs, outputs, or variables are changed by these statements. We must write out each line of code and it uses functions such as FOR, WHILE, IF, ELSE, ELSEIF AND CASE.[47]

Structured Text Advantages

Some of the advantages of Structured Text are:

1. It is very organized and good at computing large mathematical calculations.
2. It will enable you to cover some instructions that are not available in some other languages like the Ladder Diagram.

Structured Text Disadvantages

The disadvantages of the Structured Text PLC programming language are:

1. The syntax can be difficult.
2. It is hard to debug.
3. It is difficult to edit online.[47]

ST statements

A ST program consists of a number of statements. Statements are separated by semicolons (;). Unlike IL, the "end of line", line feed, is interpreted syntactically as a space. This means that a statement can be extended over several lines or several statements can be written on a single line.[47]

Graphical object	Description	Name Explanation
:=	Assignment	Assignment of a calculated value on the right to the identifier on the left.
	Call of an FB "a"	Call of another POU of type FB including its parameters. ":=" for input parameters, "=>" for output parameters.
RETURN	Return	Leave the current POU and return to the calling POU
IF	Selection	Selection of alternatives by means of Boolean expressions.
CASE	Multi-Selection	Selection of a statement block, depending on the value of the expression "f".
FOR	Iteration (1)	Multiple loop of a statement block with a start and end condition and an increment value.
WHILE	Iteration (2)	Multiple loop of a statement block with end condition at the beginning.
REPEAT	Iteration (3)	Multiple loop of a statement block with end condition at the end.
EXIT	End of loop	Premature termination of an iteration statement.
;	Dummy statement	

a Function calls are not statements in themselves but can be used within an expression as operands

Table II.10: ST statements[47]

Operands: Operands may consist of the following:

- **Literal** (numeric, alphanumeric characters, time) Example : 10, 'abcdef'.
- **Variables** (single-/ multi-element variables) Examples: Var1, Var[1,2], Var1.Substruc,
- **Function call:** more precisely: **the function return value** Example : FunName(Par1, Par2, Par3),

Operators: For logic operations on the operands within expressions, the following operators are defined in ST:

Graphical object	Description	Example and value of the expressions
()	Parenthesis	(2*3) +(4*5), Value : 26.
	Function Call	CONCAT ('AB','CD'), Value 'ABCD'
-	Negation	-10, Value : -10
NOT	Complement	NOT TRUE Value: FALSE
**	Exponentiation	3 ** 4 Value : 81.
*	Multiplication	10*20 Value: 200.
/	Division	20/10 Value : 2
MOD	Modulo	17 MOD 10 Value : 7
+	Addition	1.4 + 2.5 Value : 0
-	Subtraction	3 - 2 Value 1
<, >, <=, >=	Comparison	10 > 20 Value : FALSE
=	Equality	T#26h = T#1d2h Value : TRUE
<>	Inequality	8#15 <> 13 Value : FALSE
AND	Boolean AND	TRUE AND FALSE, Value : FALSE
XOR	Boolean Exclusive OR	TRUE XOR FALSE, Value : TRUE
OR	Boolean OR	TRUE OR FALSE, Value : TRUE

Table II.11: Operators of ST as elements of expressions incl. precedence rule[47]

II.4.5 Instruction List IL

The Instruction List is also a textual based language. The Instruction List language resembles Assembly Language. When you use this PLC Programming Language, you will use mnemonic codes such as LD (Load), AND, OR, etc. The Instruction List contains instructions with each instruction on a new line with any comments you might want to annotate at the end of each line.[47]

Instruction List Advantages

The Instruction List language is valuable for applications that need code that is compact and time critical.[47]

Instruction List Disadvantages

The main disadvantages of this PLC Programming Language are:

1. There are few structuring possibilities with the "Goto" command being one of them.
2. There can also be many errors that are more difficult to deal with in comparison to many of the other languages that I have previously reviewed.

Instruction in IL

IL is a line-oriented language. An instruction, which is an executable command for the PLC, is described in exactly one line. Empty instructions in the form of blank lines are also allowed. A statement in IL consists of the elements represented in Figure below

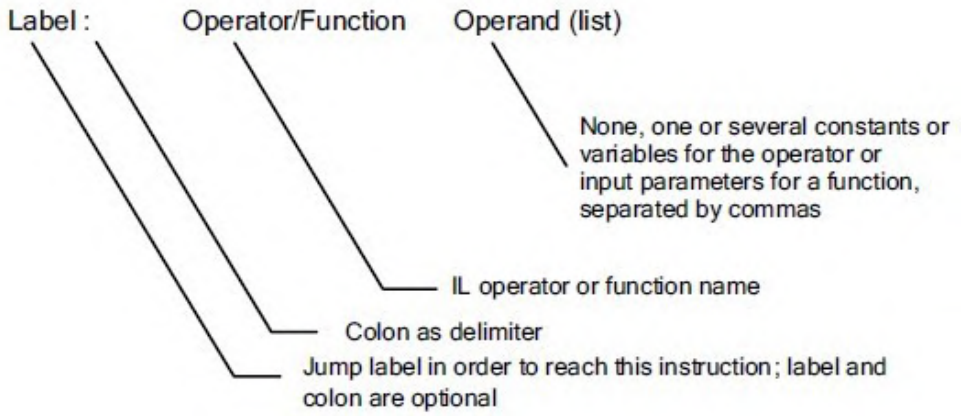


Figure II.3: Instruction in IL[47]

at least one blank space is required between operator and operand to distinguish these parts. Comments with (* ... *) brackets are allowed wherever a blank space may be inserted.

The universal accumulator (Current Result)

Standard assemblers are usually based on a real (hardware) accumulator of a processor, i.e., a value is loaded into the accumulator, further values are added, subtracted, and the result of the accumulator may finally be stored in a memory location. IL also offers an accumulator called the "Current Result" (CR) . However, the CR does not have a fixed number of bits like real hardware accumulators. The IL compiler ensures that a virtual accumulator (including accumulator stack) of any storage width is always available. The number of bits depends on the data type of the operand being processed. The data type associated with the CR also changes to match the data type of the most recent operand. Some operators change the type of the CR.[47]

Influence of the group of operators on the CR	Abbrev.	Example operators
Create	C	LD.
Process	P	GT
Leave unchanged	U	ST; JMPC
Set to undefined	-	CAL = Unconditional call of a function block. (The following instruction has to load the CR anew, because on return from the FB the CR has no defined value.)

Table II.12: Modification of the CR by different groups of operators[47]

Operators This section describes all the operators defined for IL. Some operators have extensions called modifiers. An operator combined with a modifier like N, (or C has an extended meaning. Modifiers in detail:

- **N** Negation of operand.
- **(** Nesting levels by parenthesis.
- **C** Conditional execution of operator (if CR=TRUE).

IEC 61131-3 demands Boolean operands or Bit String (ANY-BIT) for the operators of Table illustrated in the figure below

BOOL & Bit String:

Operators	Group of operators	Description
LD LDN	C	Load operand (negated) to CR.
AND ANDN	P	Boolean AND of the (negated) operand with the CR
OR ORN	P	Boolean OR of the (negated) operand with the CR
XOR XORN	P	Boolean Exclusive-OR of the (negated) operand with the CR
NOT	P	Bit-wise Boolean negation (unit complement)
ST STN	U	Store CR to the operand
S	U	Set operand TRUE, if CR=1
R	U	Set operand FALSE if CR=1
)	P	Closing parenthesis; evaluate deferred operation

Table II.13: Operators with Boolean and Bit String operands (Type ANY BIT)[47]

Operators	Group of operators	Description
LD	C	Load operand to CR.
ST	U	Store CR to the operand
ADD ADD(	P	Add operand to CR
SUB SUB(	P	Subtract operand from CR
MUL MUL(	P	Multiply operand with CR
DIV DIV(	P	Divide CR by operand
MOD MOD(	P	Modulo division
GT GT (	P	CR > operand (greater than)
GE GE (	P	CR >= operand (greater than or equal)
EQ EQ (	P	CR = operand (is equal)
NE NE (	P	CR <> operand (is not equal)
LE LE (	P	CR <= operand (less than or equal)
LT LT (	P	CR < operand (less than)
)	P	Closing parenthesis level

Table II.14: Operators for operands of generic numeric data type (type ANY NUM)[47]

Operators comparing two values (CR and operand) generate TRUE in the CR when the condition is TRUE, otherwise FALSE.

II.5 Conclusion

In this chapter, we have introduced a brief definition of PLC's programming languages developed in IEC 61131-3 Standard. As a conclusion, and after reading many reviews and opinions and with our personal experiences, the Ladder Diagram is by far the most popular PLC programming language. The main reason for this is that the Ladder Diagram language naturally followed the technology advancement from a physical relay logic to a digital and logical one. This allowed the engineers and skilled workers to follow and troubleshoot and make that transition.

In summary, there is certainly a place for all the PLC Programming Languages that we have reviewed. In the next chapter, we will take a look on the trending PLC's on today's market and elaborate a brief comparison between the leading ones.

Chapter III

Architecture design

III.1 Introduction

A Programmable Logic Controller (PLC) is an industrial microprocessor-based controller with programmable memory that stores software commands and several functions. It is composed of:

- A Central Processor Unit (CPU) that translates inputs, executes the control program saved in memory, and sends output signals.
- A power supply unit that converts AC voltage to DC.
- A memory unit that stores information from inputs and the program to be executed with the aid of using the processor.
- An input and output interface wherein the controller gets and sends data from/to external devices.
- A communication interface to acquire and transmit data on communication networks from/to remote PLCs.

In order to communicate with the PLC and order it to achieve tasks, a Development Environment containing the user program is required.

In this chapter, we will elaborate our PLC's software environment and its hardware architecture.

III.2 PLC Hardware architecture

The modular PLC is a form of PLC that allows for various expansions of the PLC system by the use of modules, hence the word "modular".

Modules provide extra functionality to the programmable logic controller, such as an expanded number of I/O units, and they are often easier to operate because each component is independent of the others.

Starting from that, our PLC hardware architecture is shown below:

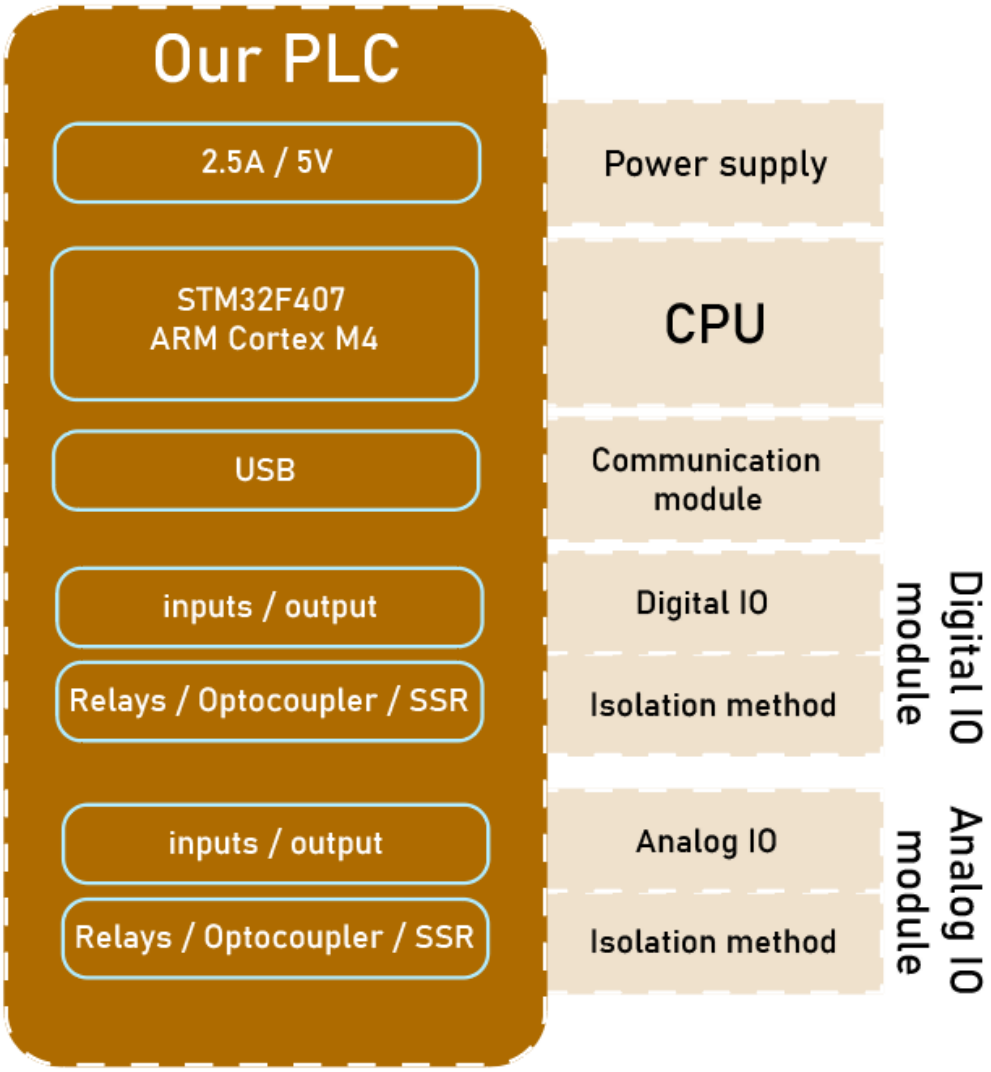


Figure III.1: PLC hardware architecture

III.2.1 CPU choice

The CPU's processing speed must be fast enough to handle many I/Os and the various forms of data collecting required by the program. Memory is also important, and how much you need depends on how many devices you have. Another factor to consider is the scan time, which is the time it takes the CPU to complete one cycle of gathering inputs, running the PLC program, and updating the outputs. In addition, the type of program and instructions you want to employ will decide the CPU's program memory.

That's why, in order to complete our PLC, we used the STM32f407 discovery board from STMicroelectronics due to its CPU: ARM Cortex-M4 processor series' capacity to do high-speed tasks is the primary reason. Because it is a RISC processor, it can run at a faster rate and execute millions of instructions per second. This is accomplished by reducing unnecessary instructions and enhancing paths. The rate is about one clock per instruction cycle.

Furthermore, STM32 MCUs are inexpensive and C/C++ programmable. On the development side, a large developer community guarantees enough troubleshooting and debugging help.

Furthermore, STMicroelectronics, the creators of the STM32, has enhanced unconditional support for its STM line of microcontrollers.[12]



Figure III.2: STM32F407 Discovery board[50]

Main features

- **CPU:** ARM 32-bit Cortex M4 @ 180MHz.
- **RAM:** 192KB.
- **FLASH:** 1MB.
- **Performance:** At 168 MHz, the STM32F407deliver 210 DMIPS/566 CoreMark performance executing from Flash memory, with 0-wait states using ST's ART Accelerator. The DSP[1] instructions and the floating-point unit enlarge the range of addressable applications.
- **Rich connectivity:** the STM32F407 features Ethernet MAC10/100 with IEEE 1588 v2 support and an 8- to 14-bit parallel camera interface to connect a CMOS camera sensor.
- **Power efficiency:** ST's 90 nm process, ART Accelerator and the dynamic power scaling enables the current consumption in run mode and executing from Flash memory to be as low as 238 µA/MHz at 168 MHz.

[1] *Digital Signal Processor*

- **Operation voltage range:** 5V. [44]

They also feature standard and advanced communication interfaces.

- Up to three I2Cs.
- Three SPIs, two I2Ss full duplex to achieve audio class accuracy.
- Four USART[2]s plus two UART[3]s.
- An USB OTG full-speed and a USB OTG. high-speed with full-speed capability (with the ULPI).
- Two CANs.
- An SDIO/MMC interface.

These features make the STM32F407xx microcontroller family suitable for a wide range of applications:

- Motor drive and application control.
- Medical equipment.
- Industrial applications: PLC, inverters, circuit breakers.
- Printers, and scanners.
- Alarm systems, video intercom, and HVAC.
- Home audio appliance.

[2] *Universal Synchronous/Asynchronous Receiver/Transmitter*
[3] *Universal Asynchronous Receiver-Transmitter*

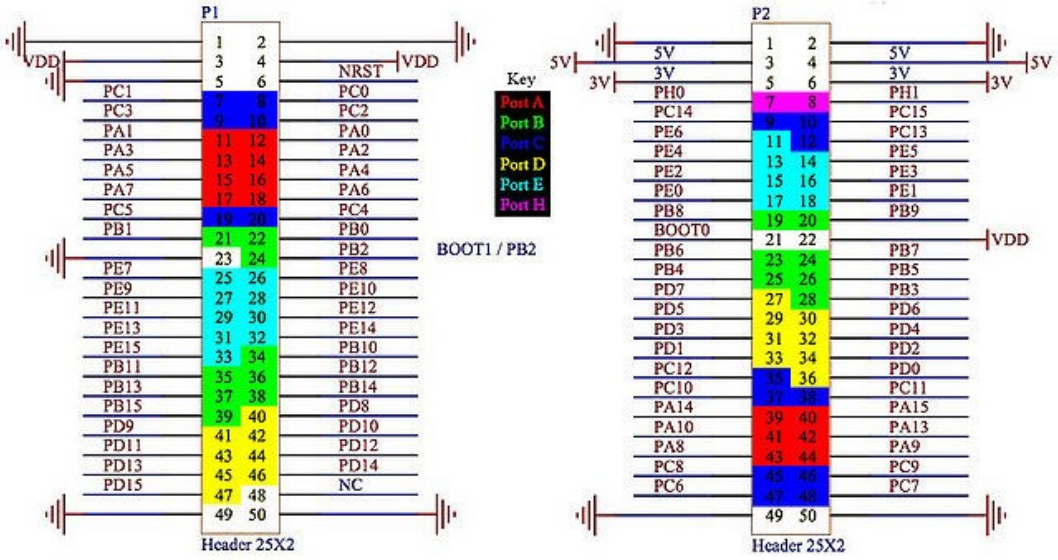


Figure III.3: STM32F407 header[43]

III.2.2 Serial communication protocols

Serial communication is a communication method in which data is continually delivered and received one bit at a time via one or two transmission lines. Because it allows for connections with minimal signal lines, one of its advantages is the potential to save money on wiring material and relaying equipment.

The most common serial communication protocols in stm32f407 are I2C[4] and SPI[5], these protocols allow data serial transmission over short distances. [31]

I2C protocol

[30] I2C is an acronym that stands for Inter Integrated Circuits protocol. It is also known as the IIC protocol. It facilitates the linking of many slave devices to one or more master devices. This serial connection is used for short-distance communication between two Integrated Circuits on the same basis (PCB). It transfers data via two bidirectional transmission lines. It can support up to 3.4 Mbps, which is a very fast data rate.

The I2C protocol has 2 wires: one for the serial clock (SCK), and one for the serial data (SDA).

[4] *Inter Integrated Circuits*
[5] *Serial Peripheral Interface*

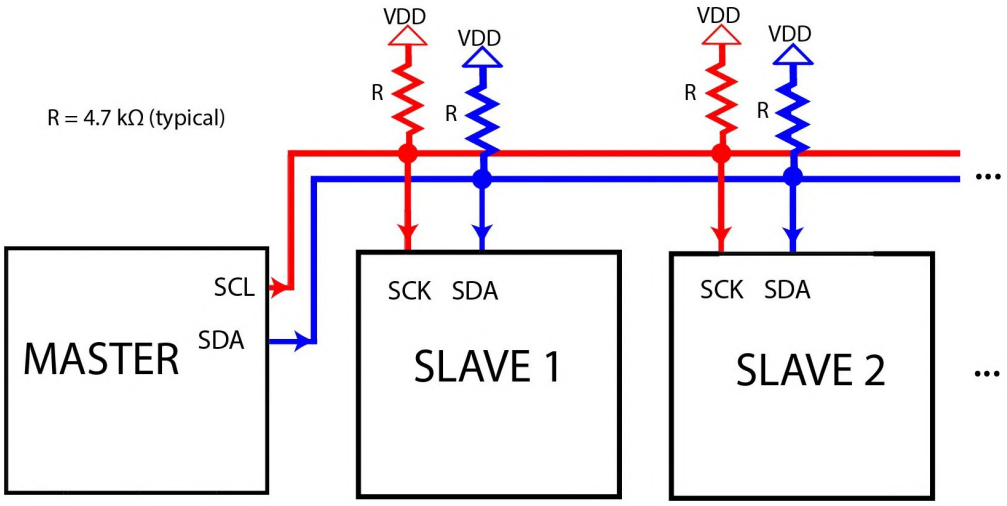


Figure III.4: I2C communication protocol[53]

SPI protocol

SPI is an abbreviation for Serial Peripheral Interface. SPI is an all-purpose synchronous serial interface. During an SPI transmission, transmit and receive data are serially moved out and in. Motorola developed it in the mid-1980s for inter-chip communication. SPI allows a microcontroller to interface with external devices such as E2PROMs.

Four logic signals are specified by the SPI bus:

- SCLK stands for Serial Clock (output from master).
- MOSI stands for Master Out Slave In (data output from master).
- MISO stands for Master In Slave Out (data output from slave).
- Chip/Slave Select (CS/SS): (often active low, output from master to indicate that data is being sent).

SPI devices communicate in a master-slave relationship. SPI requires more work and hardware resources than I2C when more than one slave is involved due to its absence of built-in device addressing. However, in point-to-point (single master, single slave) applications, SPI is easier and more efficient than I2C for a similar reason: the lack of device addressing implies less overhead.[51]

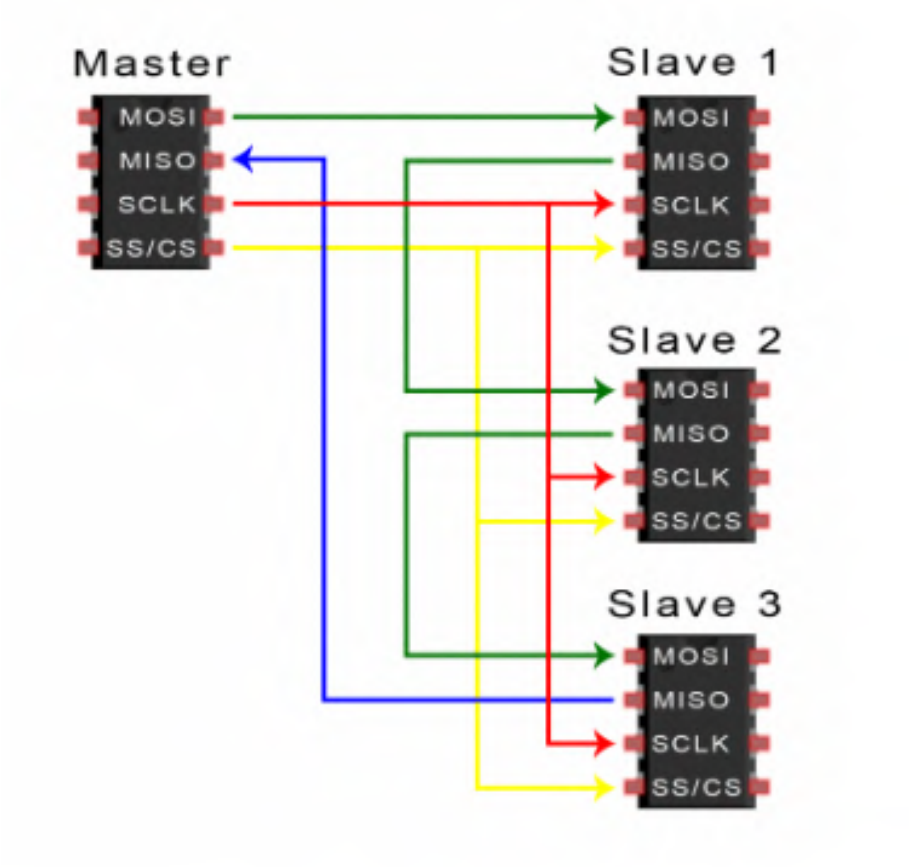


Figure III.5: SPI communication protocol[10]

III.2.3 Isolation methods

In order to protect our inputs and outputs, and to increase our PLC's durability in the long term, optocouplers and Electromechanical Relays (EMR) are used to achieve this task.

Another isolation method and the most effective one is by using a Solid-State Relay (SSR), this later contains an integrated optocoupler which made it the best isolation method.

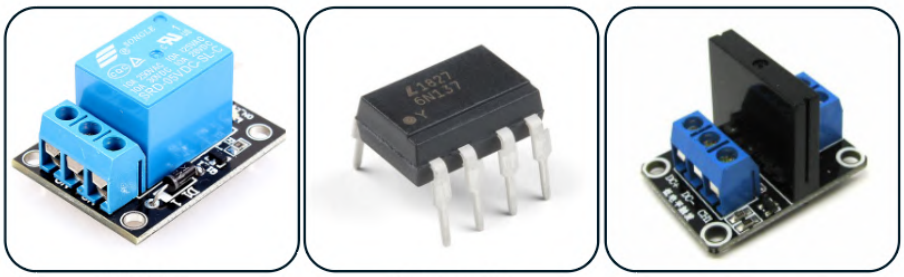


Figure III.6: Isolation components[38][15][6]

Solid-State Relay (SSR)

An SSR is an electrical switching device that turns on or off when an external voltage (AC or DC) is applied across its control terminals. They perform the same job as an electromechanical relay, but because they include no moving components, they have a longer operating lifetime.

SSRs are made up of three pieces: a sensor that reacts to a suitable input (control signal), an electronic switching device that switches power to the load circuitry, and a coupling mechanism that allows the control signal to trigger this switch without the need of mechanical parts.

They can be configured to switch AC or DC loads. Power semiconductor components such as thyristors and transistors are used in packaged SSRs to switch currents up to several hundred amperes.[11]

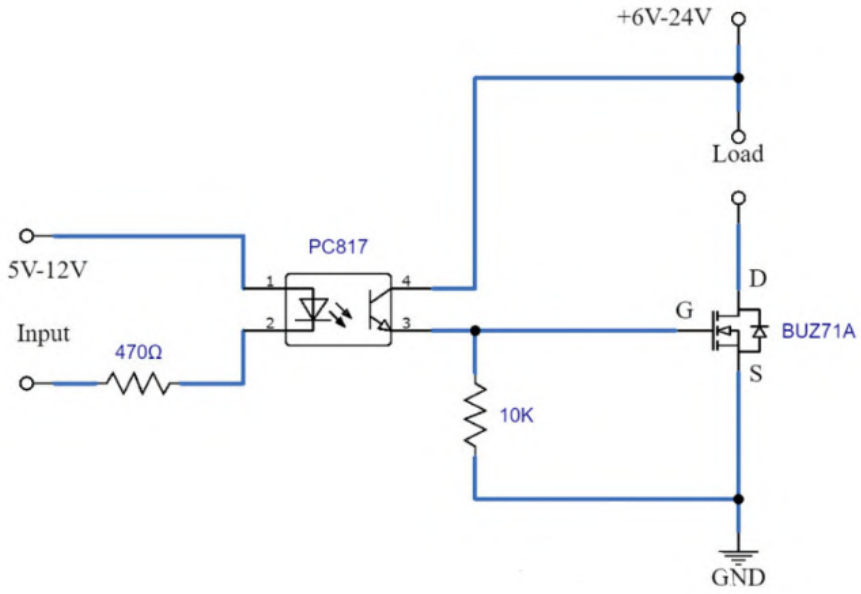


Figure III.7: SSR DC input circuit[2]

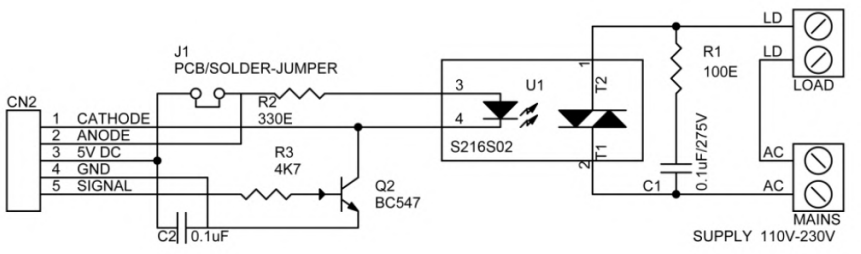


Figure III.8: SSR AC input circuit[8]

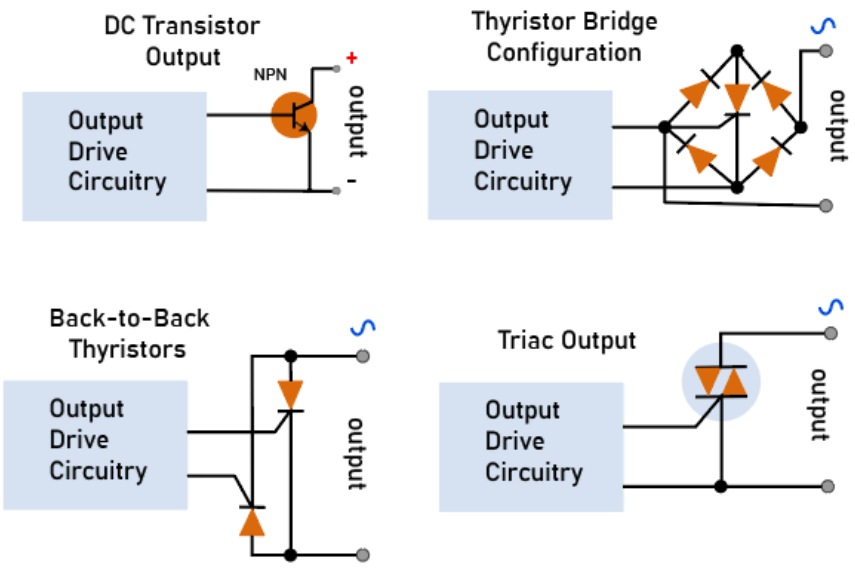


Figure III.9: SSR output circuit

III.2.4 Power supply

The power supply is provided either by the host PC through the USB cable, or by an external 5 V power supply. The D1 and D2 diodes protect the 5V and 3V pins from external power supplies: [14]

- 5V and 3V can be used as output power supplies when another application board is connected to pins P1 and P2.

In this case, the 5V and 3V pins deliver a 5 V or 3 V power supply and power consumption must be lower than 100 mA.

- 5V can also be used as input power supplies, for instance when the USB connector is not connected to the PC. In this case, the STM32F4DISCOVERY board must be powered by a power supply unit or by auxiliary equipment.

III.2.5 Communication module

There are three protocols to debug and communicate with our PLC and allow it to communicate with its internal and external slave devices:

Serial Wire Debug (SWD)

Serial Wire Debug (SWD) is an electrical alternative JTAG interface with two pins (SWDIO/SWCLK) that uses the same JTAG protocol. SWD employs a bi-directional wire protocol described in the ARM Debug Interface v5 that is an ARM CPU standard.[32]

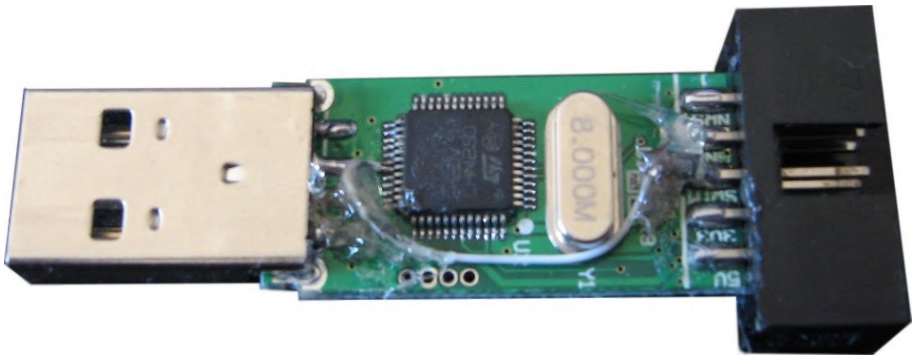


Figure III.10: SWD module[45]

Universal Serial Port (USB)

The most successful serial interface was USB, which had the following characteristics:

- Simplicity and adaptability (plug-and-play)
- Bi-directionality
- Accelerating speeds at a reasonable cost

The STM32 USB hardware and software are compliant with USB1.1 and USB2.0 specifications.

The flash memory of the microcontroller can be programmed through the mini USB port by the STlink interface and through the micro USB port using Device Firmware Update (DFU)

STlink

The ST-LINK/V2 is an in-circuit debugger and programmer for the STM8 and STM32 microcontrollers. The ST-LINK/V2-ISOL features digital isolation between the PC and the target application board. It also withstands voltages of up to 1000 Vrms.

In our case the STlink debugger is integrated in our PLC microcontroller board.[40]

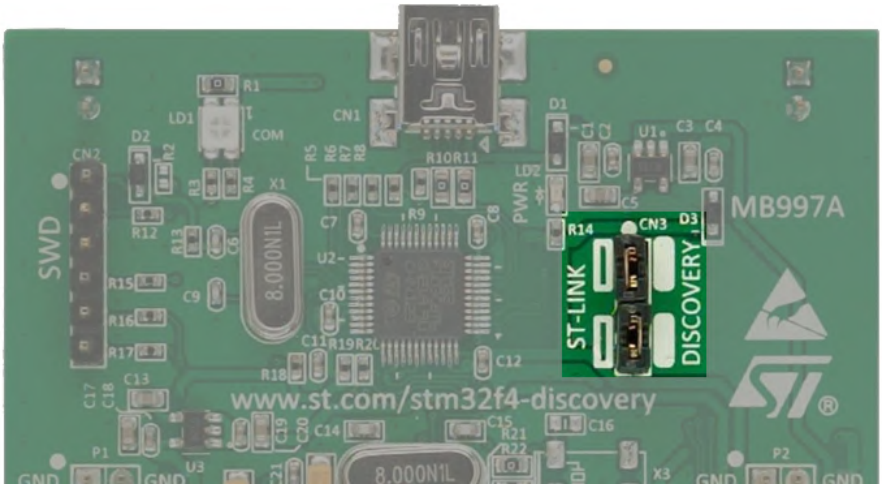


Figure III.11: STlink module[49]

Device Firmware Update (DFU)

A firmware update will provide your gadget with advanced operational instructions without requiring any hardware upgrades. By updating the firmware, you will be able to explore new features introduced to the device as well as have a better user experience while interacting with it. A firmware update will improve the performance of the CPU by optimizing the performance of the firmware or device driver.[48]

III.2.6 PLC software environment

Our project software architecture is illustrated below:

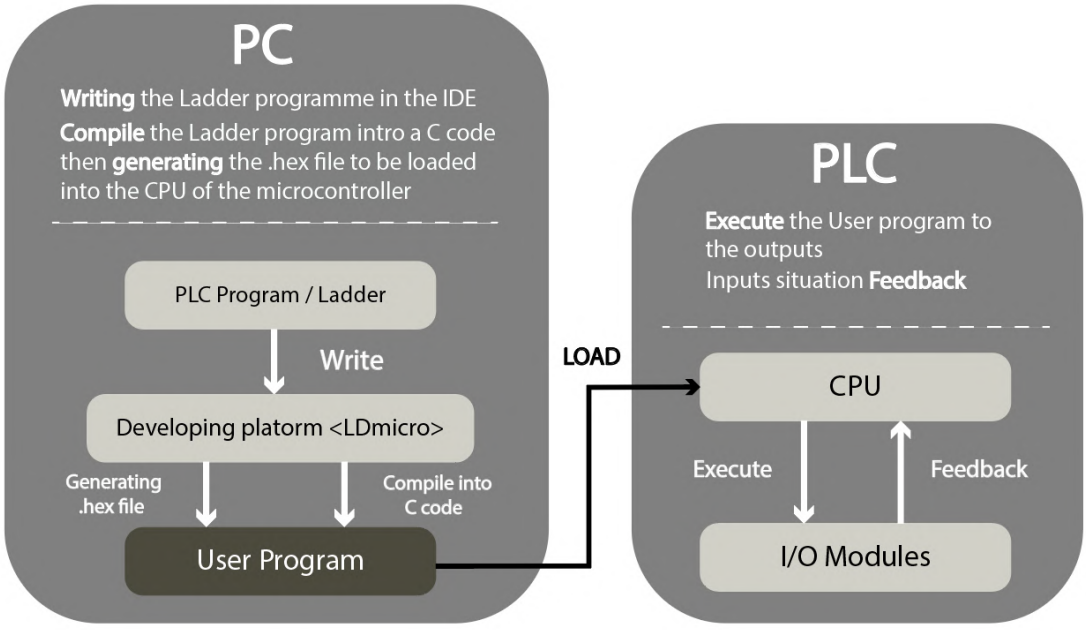


Figure III.12: PLC software architecture

III.2.7 PLC programs

PLC programming is a key part of creating and executing control applications based on the needs of the client. A PLC program is a collection of written or graphical instructions that reflect the logic to be applied for specific industrial real-time applications.

1. Textual language
    - Instructions List (IL)
    - Structured Text (ST)
    - Sequential Function Chart (textual version)
2. Graphical language
    - Ladder Diagrams (LD)
    - Function Block Diagram (FBD)
    - Sequential Function Chart (graphical version)

Many users prefer graphical languages to text-based languages for programming a PLC owing to their easy and convenient programming capabilities. Each PLC software's standard library contains all of the required functions and functional blocks. Timers, counters, strings, comparators, numeric, arithmetic, bit-shifting, calling functions, and so on are examples of function blocks. [41]

III.2.8 User program

To begin with, the program is written in LADDER language. This later can be saved as LDmicro file (.ld) compatible with other computers containing LDmicro. The LADDER program is compiled into a C code file (.c) and built into a native hexadecimal file (.hex) to be deployed to our PLC's processing unit (CPU).

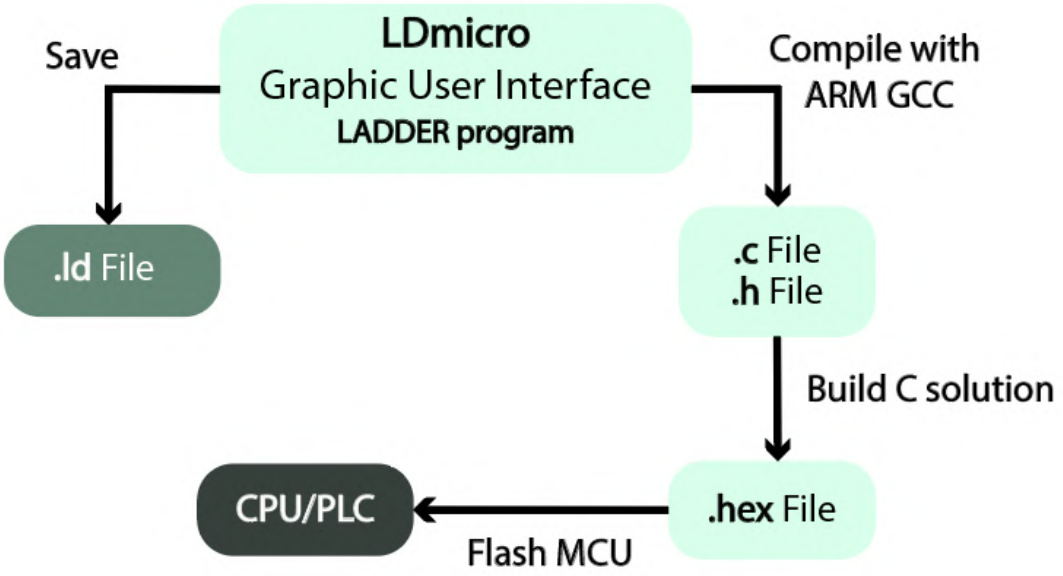


Figure III.13: The LDmicro architecture and deployment process

III.2.9 Integrated Development environment IDE

In this project we are mainly going to be using the LDmicro developing software.

LDmicro software

Typically, we utilize software that generates programs in C, BASIC, or assembly language to generate code for a microcontroller. The code in this sort of programming is textual (list of instructions).

However, because this approach is problematic with PLCs, LDmicro provides a programming interface in Ladder language.

LDmicro is an application that allows you to create, build, and modify ladder diagrams, as well as simulate and compile ladder diagrams into native hexadecimal firmware code (.hex files) for ARM, Atmel AVR, and Microchip PIC microcontrollers. The program is represented graphically, using relay contacts (input) and coils (output).

The software has its own simulator that operates in real time, allowing us to evaluate the logic of our program before compilation.

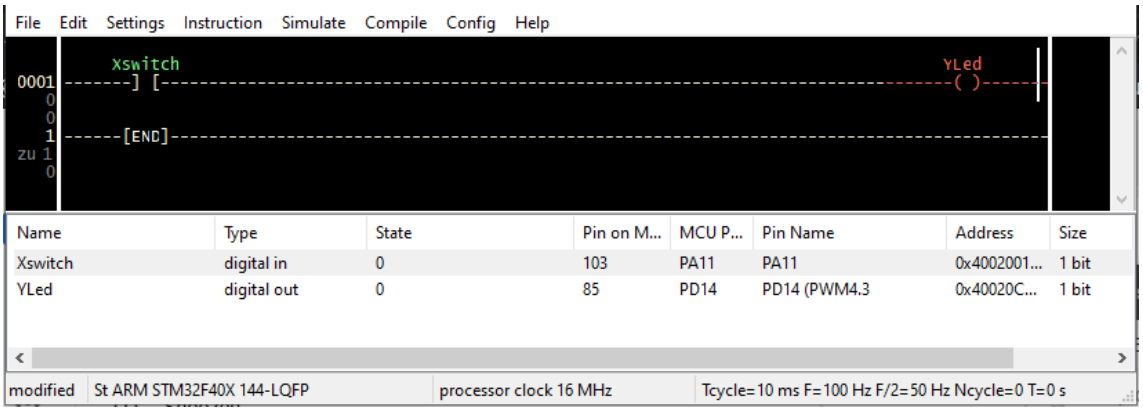


Figure III.14: LDmicro interface

Variable name	Signification
X name	Relay to a microcontroller input pin
Y name	Relay to a microcontroller output pin
R name	Internal relay: one bit in memory
T name	Timer; work time, rest time, or totalizer
C name	Counter or discounter
name	Generic variable (Integer)
A name	An integer read on an A/D converter

Table III.2: Commonly used variables on LDmicro.

STM32CubeIDE

STM32CubeIDE is a professional C/C++ development platform for STM32 microcontrollers and microprocessors that includes peripheral configuration, code creation, code compilation, and debug tools.

Features

- Integration of services from STM32CubeMX:STM32 microcontroller, microprocessor, development platform and example project selectionPinout, clock, peripheral, and middleware. configurationProject creation and generation of the initialization codeSoftware and middleware completed with enhanced STM32Cube Expansion Packages.
- Based on Eclipse/CDT, with support for Eclipse add-ons, GNU C/C++ for Arm® toolchain and GDB debugger.
- STM32MP1 Series: Support for OpenSTLinux projects: LinuxSupport for Linux.
- Additional advanced debug features including: CPU core, peripheral register, and memory viewsLive variable watch viewSystem analysis and real-time tracing (SWV)CPU fault analysis toolRTOS-aware debug suport including Azure.

- Support for ST-LINK (STMicroelectronics) and J-Link (SEGGER) debug probes.
- Import project from Atollic TrueSTUDIO and AC6 System Workbench for STM32 (SW4STM32).
- Multi-OS support: Windows, Linux, and macOS, 64-bit versions only.

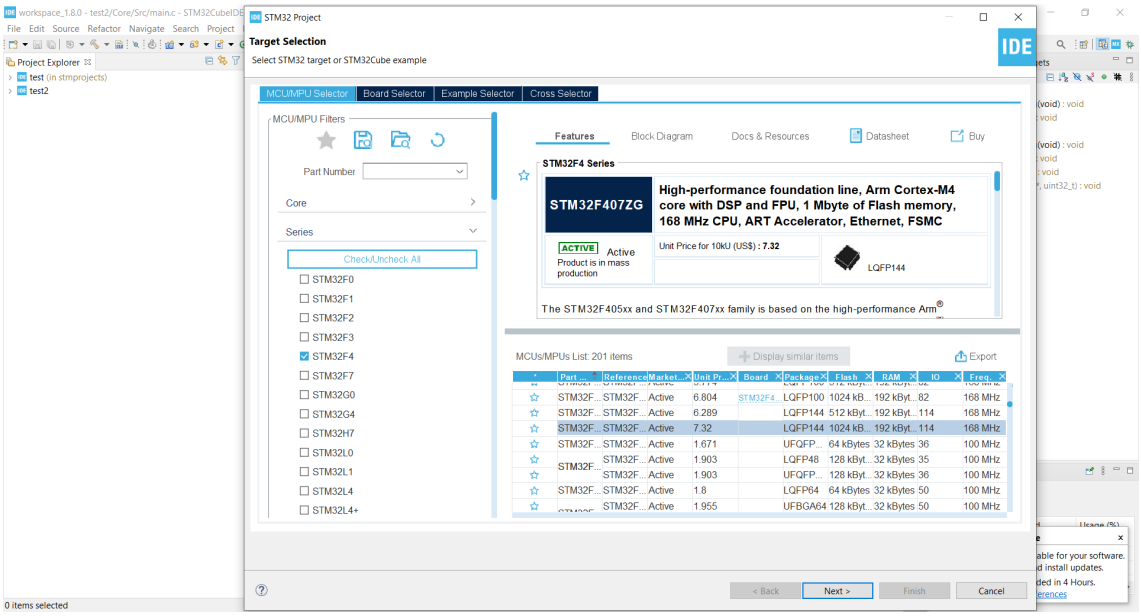


Figure III.15: STM32CubeIDE interface

After exploring both softwares files, we found out that LDmicro uses the standard STM32 libraries and its (main.c) file generated has a static structure, on the other hand STM32CubeIDE uses the STM32 HAL libraries and does not have a static structure.

We want to get these softwares to work together by establishing a link between them so LDmicro will be able to use the HAL libraries instead of the standard ones.

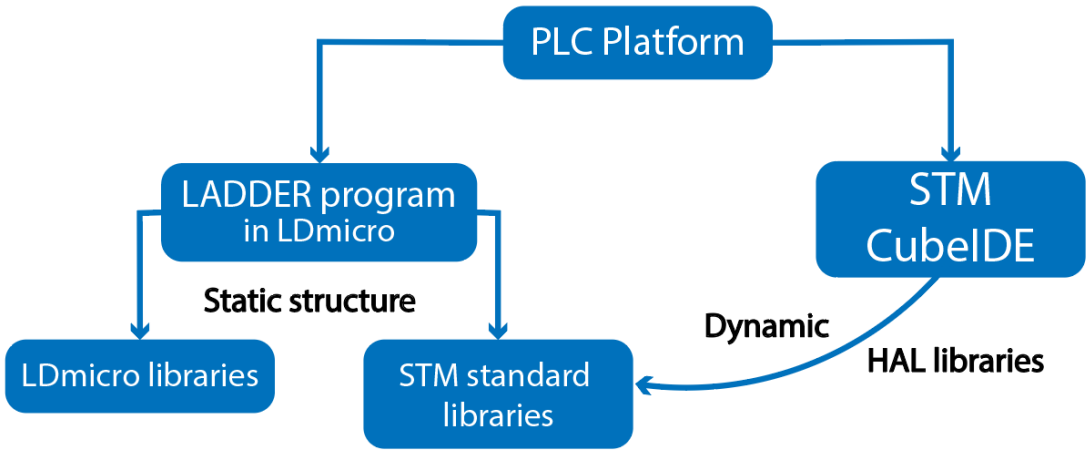


Figure III.16: Putting LDmicro and STM32CubeIDE together

The hardware abstraction layer (HAL) provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). The HAL driver APIs are generic APIs, which provide common and generic functions for all the STM32 series and extension APIs.

Adding a new STM32 microcontroller to LDmicro

By establishing the link between these two softwares, all stm32 microcontrollers can be used in order to create a PLC using LDmicro. Unfortunately, due to programming difficulties we couldn't make it happen.

Another way to use other stm32 microcontrollers in LDmicro is by modifying the source code files as it is shown below:

First, we start of by defining a new Microcontroller Unit (MCU) in the "mcutable.h" file and initializing all of the IO, ADC, PWM, UART, SPI and I2C pin information from the board's Datasheet (for example STM32F401VE). Then, in order to get our new MCU board to show up in LDmicro Graphic User Interface (GUI), all we have to do is to define it in the mcutable.cpp respecting the structure elaborated by LDmicro developer.

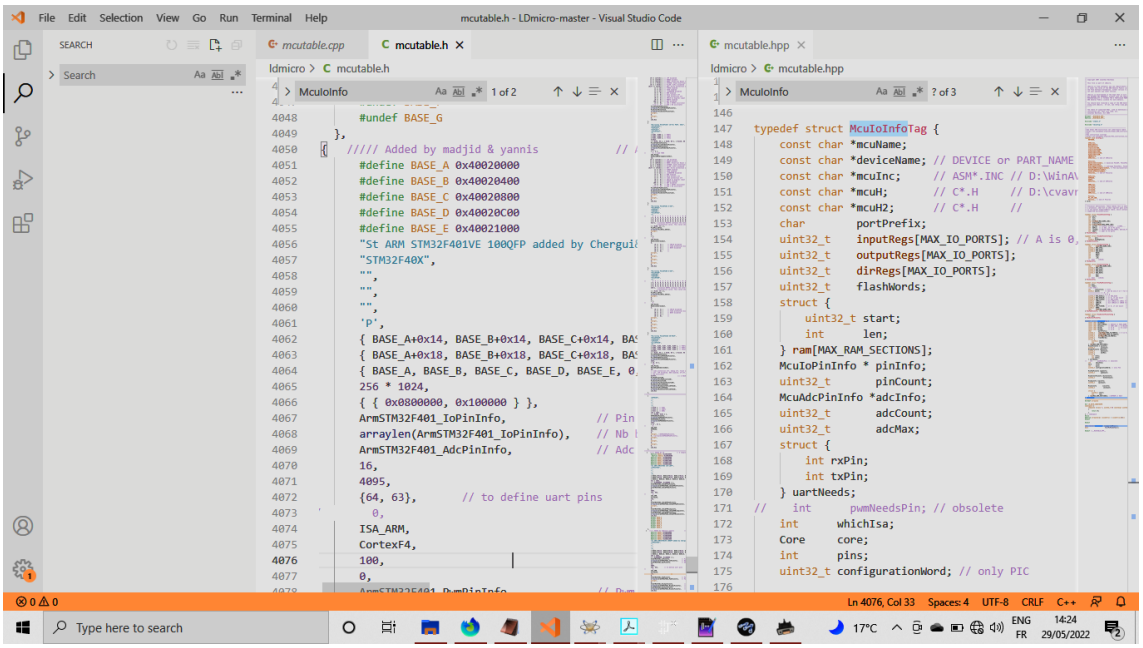


Figure III.17: Mcutable files

In the next step, build files of the modified source code are generated using CMake and Strawberry Perl in the Command Prompt (CMD).

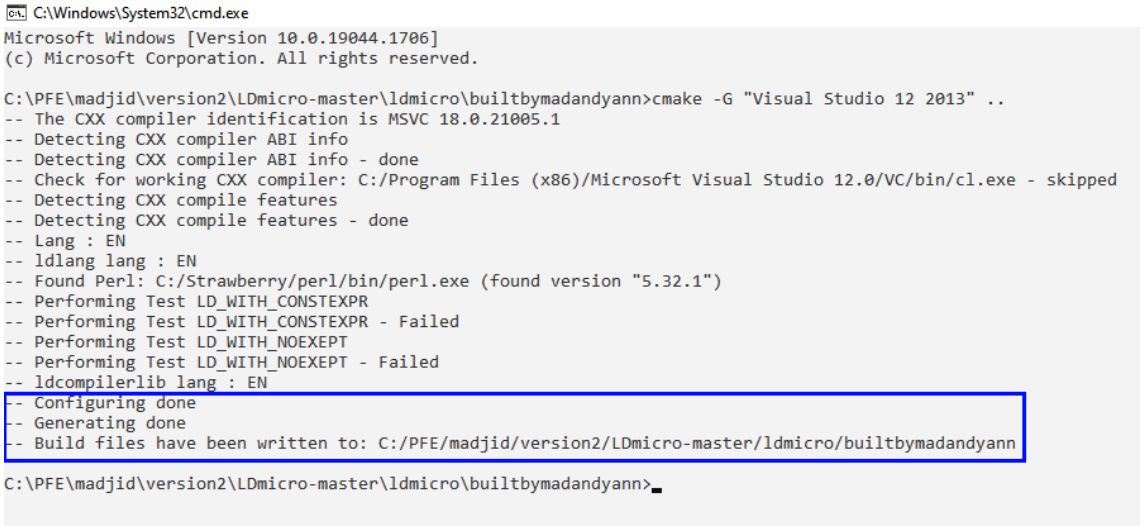


Figure III.18: Generating build files

CMake is an extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner.

Strawberry Perl is a perl environment for MS Windows containing all you need to run and develop perl applications. It includes perl binaries, compiler (gcc) + related tools and much more.

Note: we used the command '**cmake –G "Visual studio 12 2013" . .**' because Microsoft Visual Studio 2013 IDE is going to be used to rebuild our updated LDmicro version.



Figure III.19: Build LDmicro application files

The final step consists of rebuilding the build files generated in the previous step using MS Visual Studio 2013, open the generated solution file 'build/ldmicro.sln' in Visual Studio and build the target 'ALL-BUILD'. As a consequence, we obtain our new LDmicro version containing the new board we added.

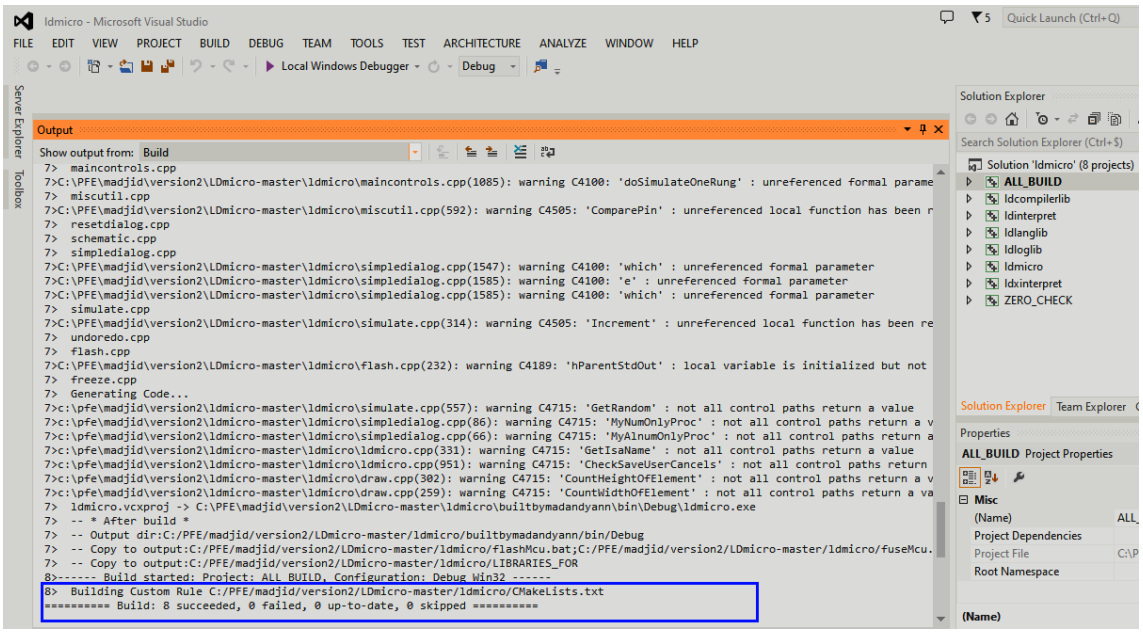


Figure III.20: Building our new version using MS VS2013

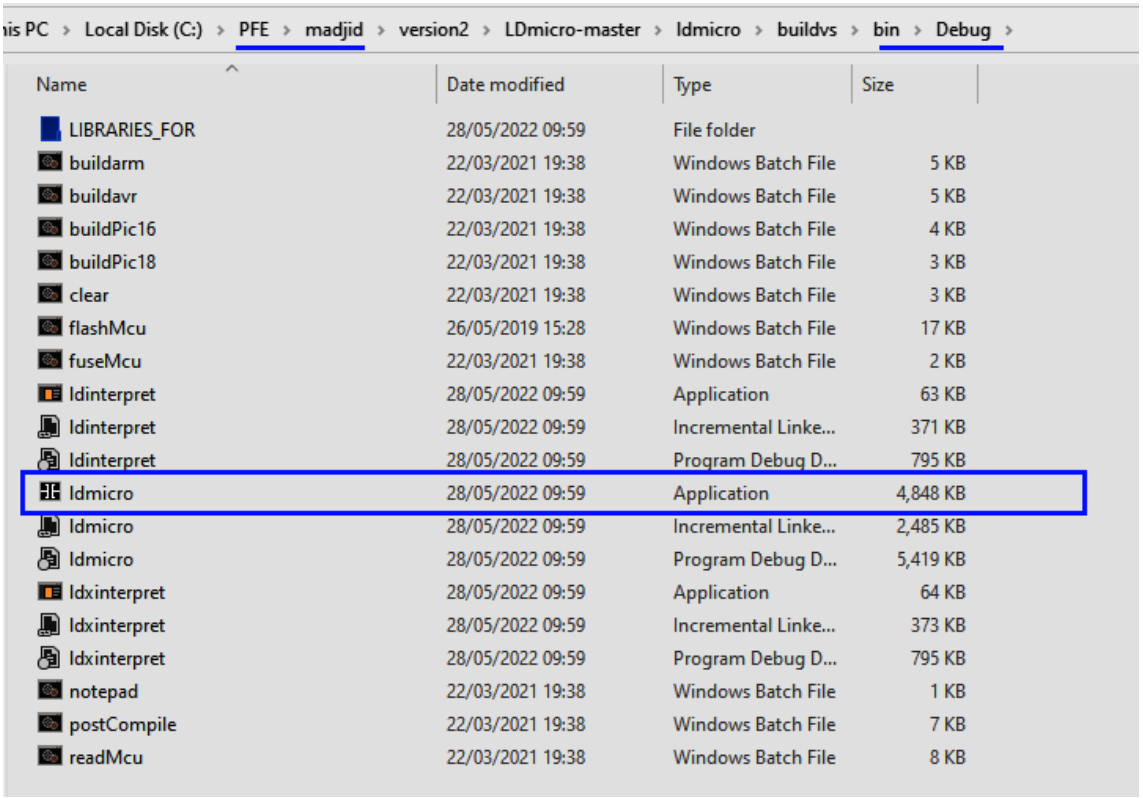


Figure III.21: LDmicro updated version

As we can see, a new MCU[6] appears in the ARM MCUs settings

[6] *Microcontroller Unit*

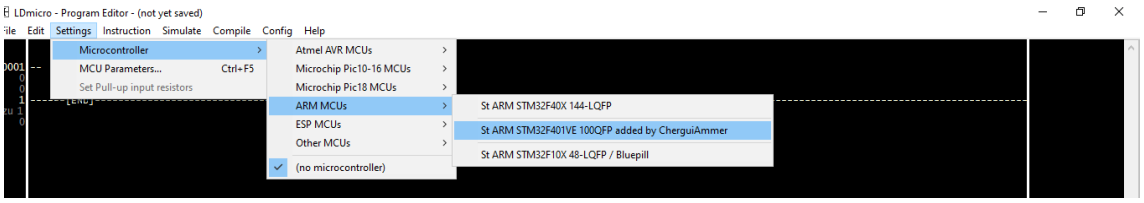


Figure III.22: New MCU added successfully

III.3 Conclusion

The goal of this chapter was to elaborate our PLC's hardware architecture and software environment in which the latter will accord with the requirements of the industry 4.0.

In the next chapter, we will be moving on to finalizing our PLC prototype and putting it to the test.

Chapter IV

Results and discussion

IV.1 Introduction

This chapter consists of evaluating our PLC prototype's efficiency and functioning behavior. We begin by verifying the digital circuit and the addressing flow to each I/O pin. Then we move on to the analog circuit. Finally, we will test the communication protocols.

IV.2 First version prototyping

Our initial prototype shown in the figure III.3 represents the exact pins of our PLC in which they'll be addressed and initialized by the LDmicro software.

IV.2.1 4 Channel 5V Optical Isolated Relay Module

This is a LOW Level 5V 4-channel relay interface board with a 15-20mA driver current requirement for each channel. It can be used to control a variety of high-current appliances and equipment.

It has high-current relays that operate at AC250V 10A or DC30V 10A. It possesses a standard interface that can be directly controlled by a microcontroller This module is called optically isolated from the high voltage side for safety and ground loop prevention when interacting with a microcontroller. [3]

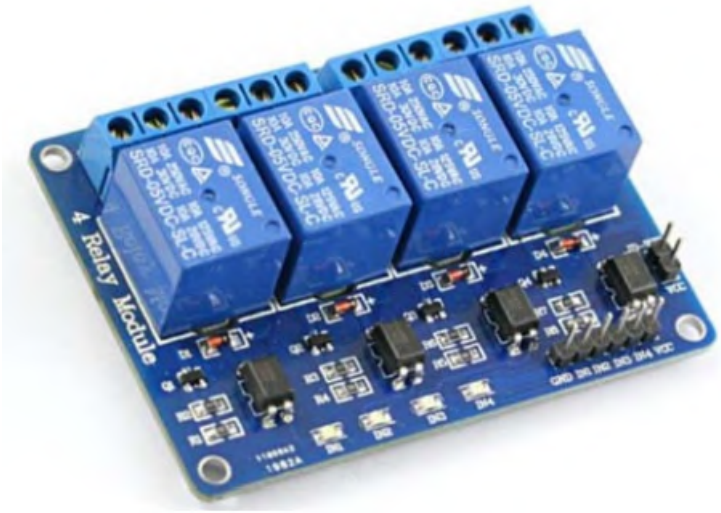


Figure IV.1: 4 relays module[3]

Brief data

- Relay Maximum output: DC 30V/10A, AC 250V/10A.
- 4 Channel Relay Module with Opto-coupler. LOW Level Trigger expansion board, which is compatible with Arduino control board.

- Standard interface that can be controlled directly by microcontroller (8051, AVR, *PIC, DSP, ARM, MSP430, TTL logic).

- Relay of high quality low noise relays SPDT. A common terminal, a normally open, one normally closed terminal.

- Opto-Coupler isolation, for high voltage safety and prevent ground loop with microcontroller.

Schematic

VCC and RY-VCC are also the relay module's power supplies. When you need to drive a high-power load, remove the jumper cap and connect an extra power supply to RY-VCC to power the relay; connect VCC to the MCU board's 5V to supply input signals.[3]

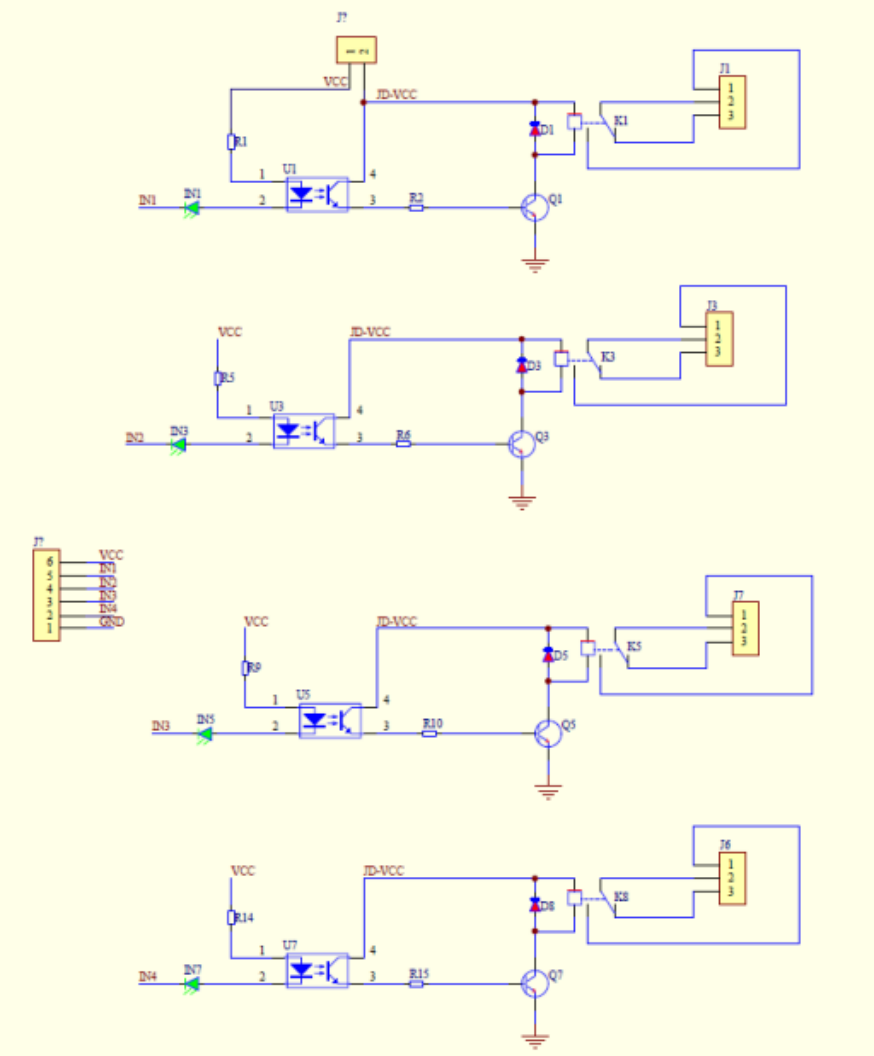


Figure IV.2: 4 relay module schematic.[4]

IV.2.2 Digital I/Os

We are going to be testing our PLC digital IOs by using 4 push buttons and 3 LED[1] lights connected to a 4 relay module as it is shown below:

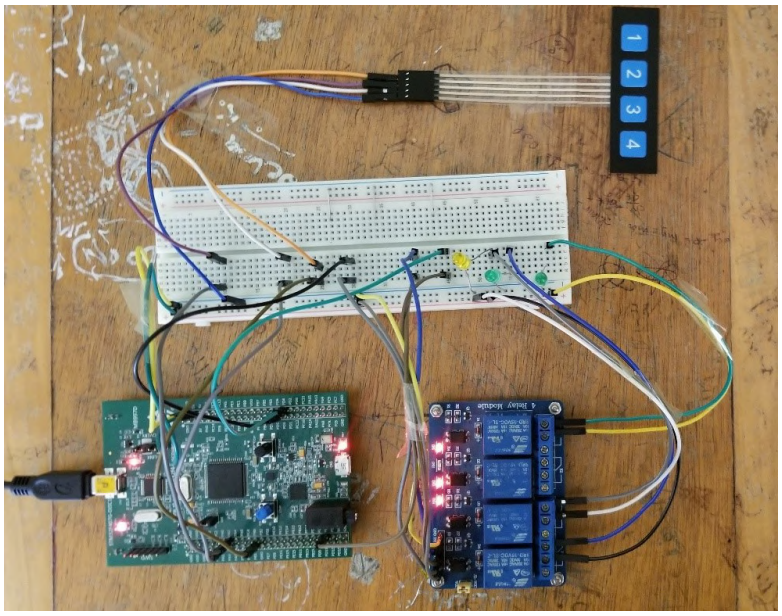


Figure IV.3: PLC first version prototyping.

LADDER program

In order to test our PLC functionality and confirm our hardware and software structure validity, we need to write a LADDER program and load it into our microcontroller CPU.

We start of by creating a new LDmicro project by selecting file > new in the upper toolbox. Once the project is created, we have to select the microcontroller in which we will be using by clicking on Settings > microcontroller > ARM MCUs (in our case the stm32f407 is used). After writing LADDER program, every input/output must be assigned to any of the microcontroller pins.

In our example, we will be using the pins PB1, PA1, PA2, PA3 as digital inputs and the pins PD12, PD4 as digital outputs. The final step consists of compiling our LADDER program and generate the .hex file to be loaded into our PLCs CPU.

[1] *Light Emitting Diode*

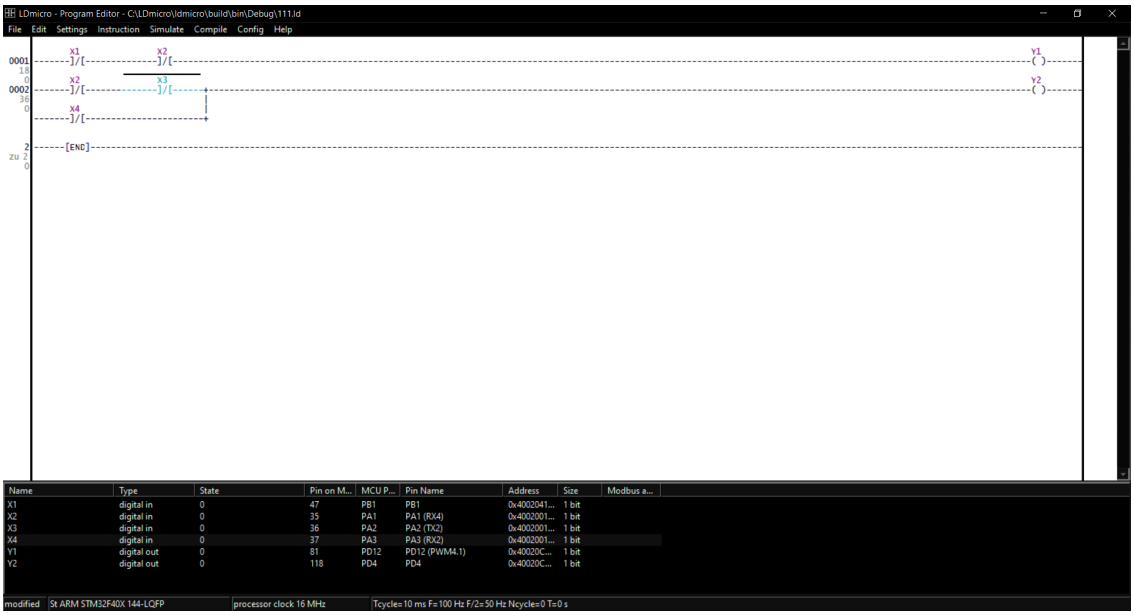


Figure IV.4: Digital I/Os test LADDER program.

As a result, if we push the button 1(X1) and 3(X2) at the same time, the LED 1(Y1) will light up and if we push buttons 3(X2) and 4(X3) together or button 2(X4), the LED 2(Y2) will light up.

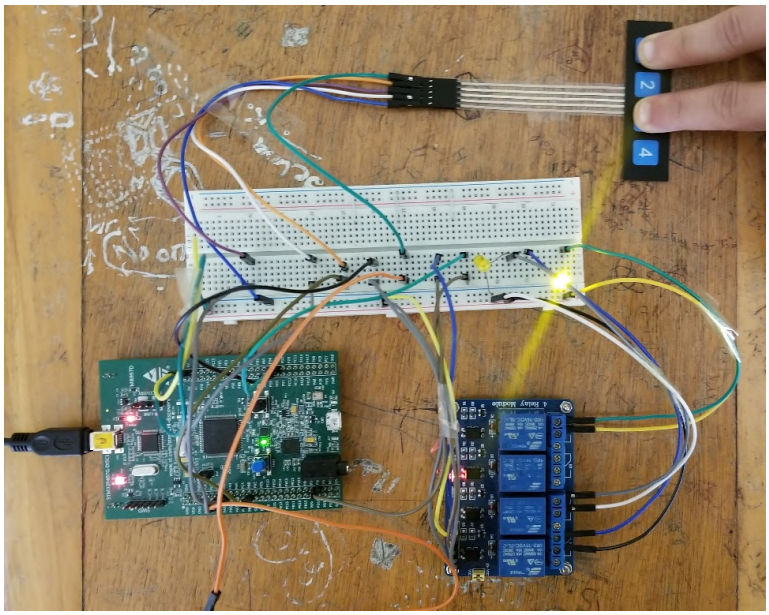


Figure IV.5: LED test result case 1.

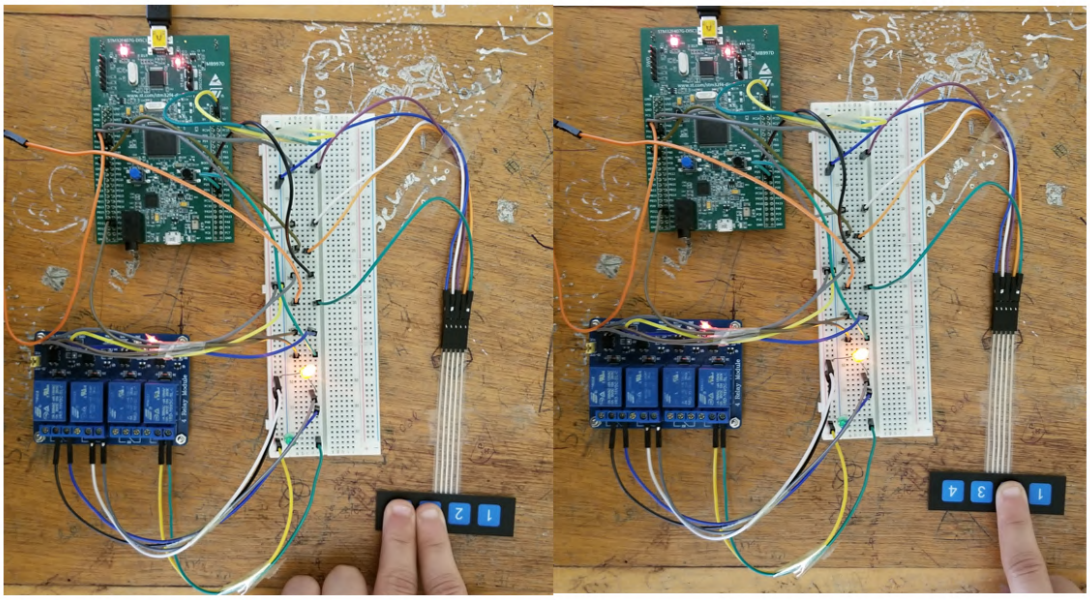


Figure IV.6: LED test result case 2.

I2C Liquid Crystal Display 20x4 (LCD)

This is a new high-quality 4 line 20 character LCD display module with on-board contrast control adjustment, backlight, and I2C communication interface. The real value of this I2C Serial LCD[2] module is that it simplifies the circuit connection and saves some I/O pins on our STM32 board.[17]

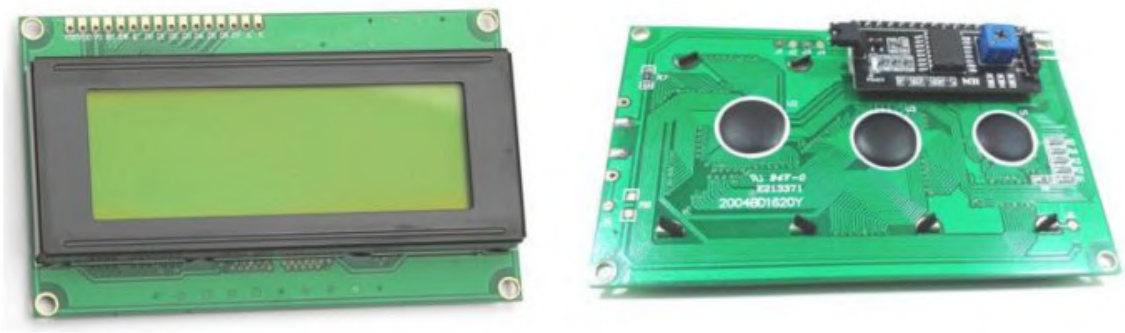


Figure IV.7: 20x4 LCD screen with integrated I2C interface.[17]

Brief data

- Compatible with STM32 Boards or other controller boards with I2C bus.
- Display Type: Black on yellow green backlight.
- I2C Address:0x38-0x3F (0x3F default)
- Supply voltage: 5V

[2] *Liquid Crystal Display*

- Interface: I2C to 4bits LCD data and control lines.
- Contrast Adjustment : built-in Potentiometer.
- Backlight Control: Firmware or jumper wire.
- Board Size: 98x60 mm.

We are going to write a LADDER program in which ‘Testing our I2C protocol to our PLC Prototype’ will appear on the LCD screen:

LADDER program

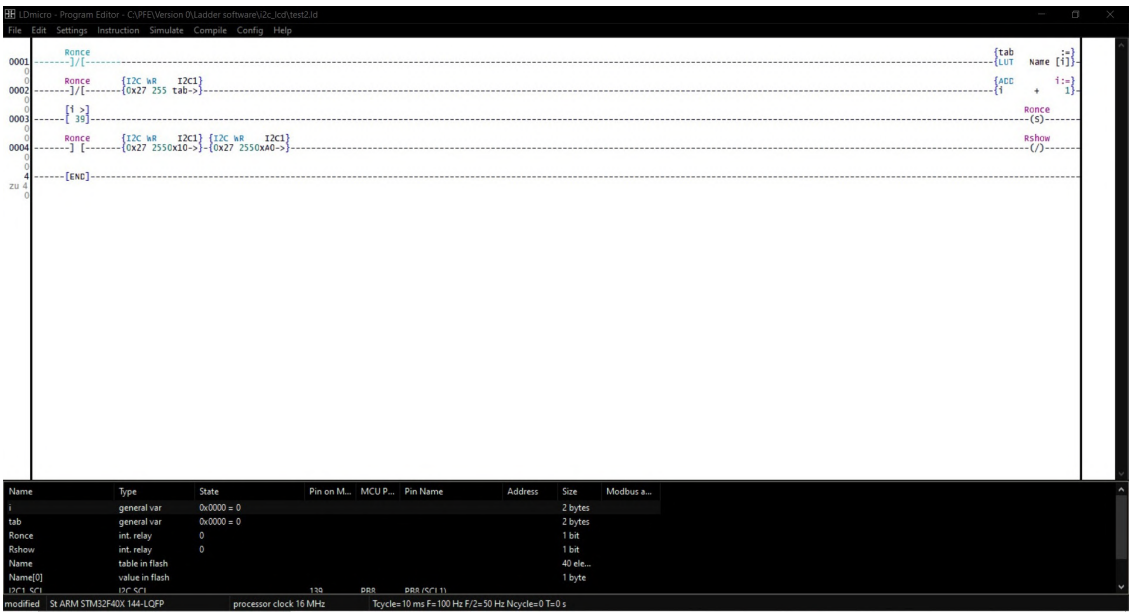


Figure IV.8: LCD test LADDER program.

After compiling the program and loading it into the PLC’s CPU we obtain what it is shown in the Figure IV.9:

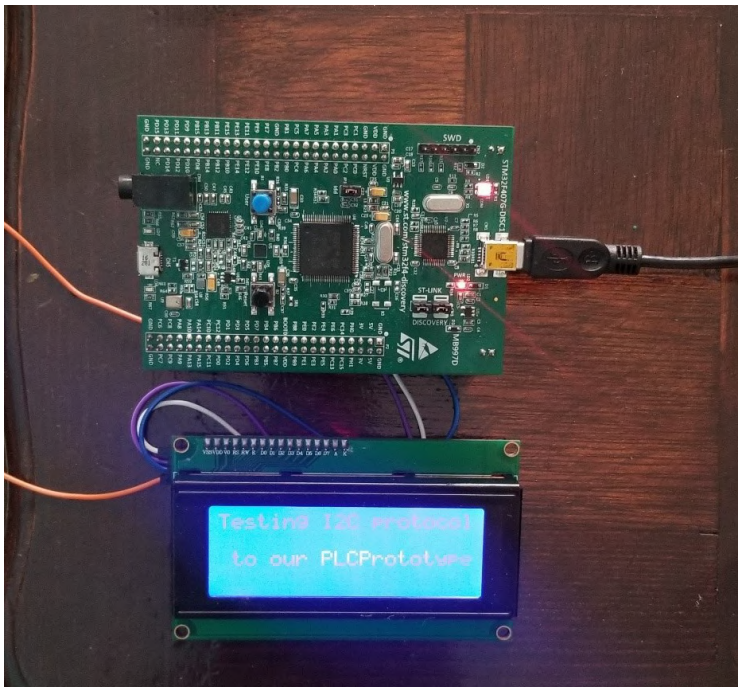


Figure IV.9: LCD test result.

IV.2.3 Analog I/Os

Temperature sensor LM35

A temperature sensor is a type of electronic device that monitors the temperature of its surroundings and turns the measured data into electronic data in order to record, monitor, or communicate temperature changes. Temperature sensors come in a variety of shapes and sizes. Some temperature sensors need direct touch with the physical thing being monitored (contact temperature sensors), whilst others detect an object's temperature indirectly (non-contact temperature sensors).

in our Analog I/Os test, we will be using the LM35 which is a temperature sensor that produces an analog signal proportional to the immediate temperature. The output voltage may be simply interpreted to produce a temperature value in Celsius. The benefit of the lm35 over the thermistor is that it does not require external calibration. The covering also protects it from self-heating. [52]

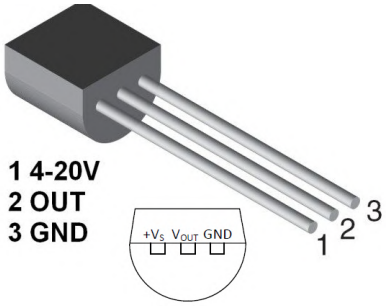


Figure IV.10: LM35 pinout.[20]

Features

- Calibrated Directly in Celsius (Centigrade)
- Linear + 10-mV/°C Scale Factor
- 0.5°C Ensured Accuracy (at 25°C)
- Rated for Full −55°C to 150°C Range
- Suitable for Remote Applications
- Operates from 4 V to 30 V
- Less than 60-µA Current Drain
- Low Self-Heating, 0.08°C in Still Air
- Non-Linearity Only ±¼°C Typical
- Low-Impedance Output, 0.1 Ω for 1-mA Load

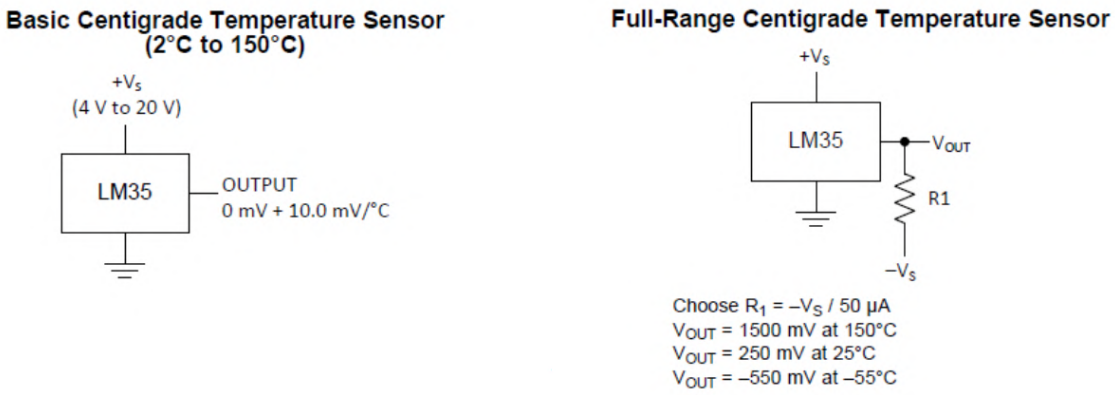


Figure IV.11: LM35 circuit configuration.[20]

LADDER program

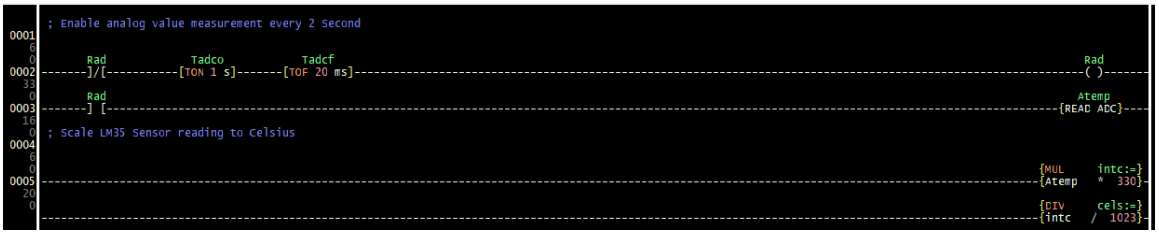


Figure IV.12: LM35 temperature reading LADDER program.

Note: A YwRobot power motherboard v2 will be used as an external power supply in order to power up our STM32 microcontroller.

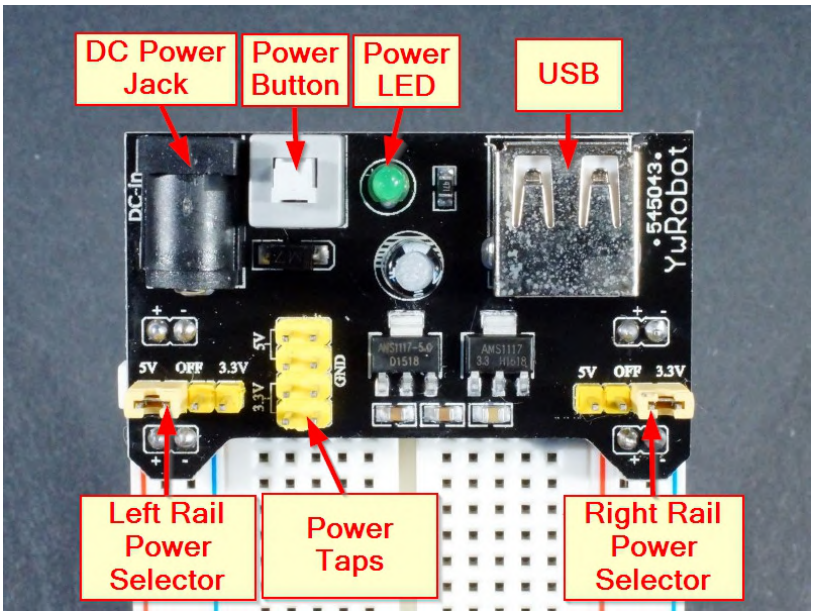


Figure IV.13: YwRobot motherboard v2 power supply module.[23]

this application consists of reading the environment's temperature using a L35 temperature sensor then display it on a LCD screen via I2C communication protocol.

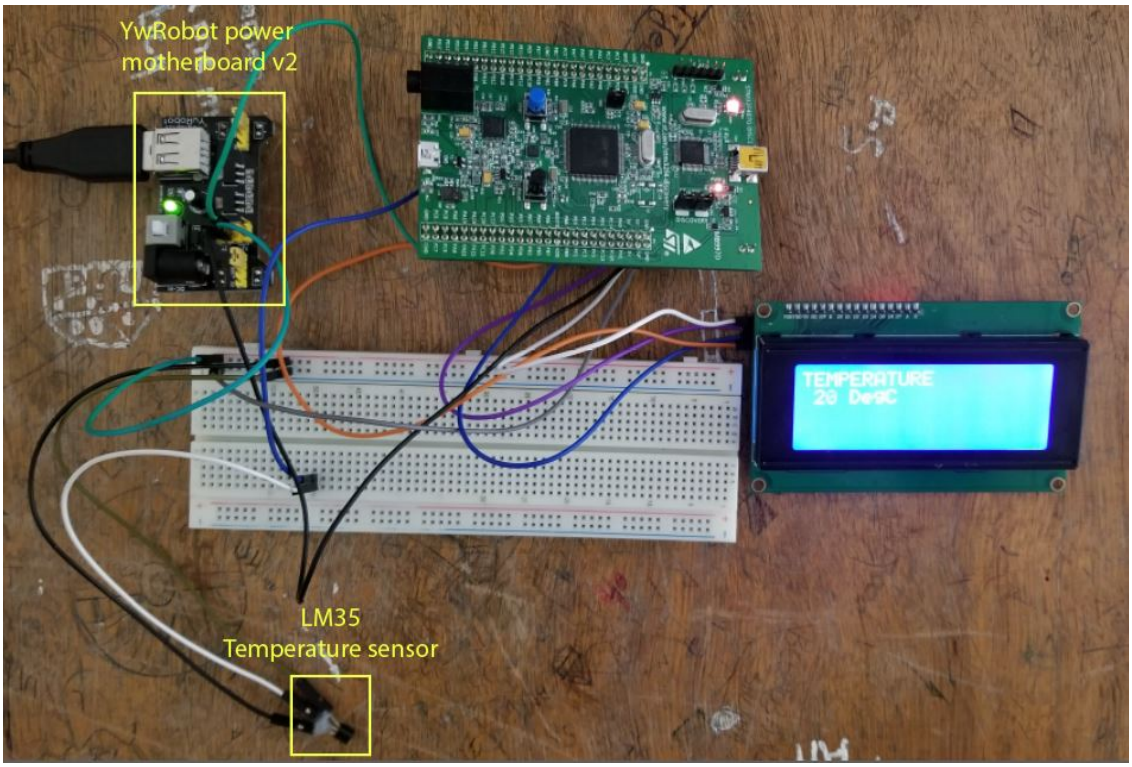


Figure IV.14: LM35 temperature reading result.

Timers

A timer (sometimes known as a counter) is a piece of hardware included in many microcontrollers. Their job is straightforward: they count (up or down, depending on the configuration—we'll assume up for the time being). An 8-bit timer, for example, will count from 0 to 255. When a timer reaches its maximum value, it will "roll over." As a result, once our 8-bit timer hits 255, it will reset to 0.

Most timers may be configured in a number of ways to modify how they work. These options are often applied through various specific function registers within the microcontroller. For example, instead of counting up to 255, you may instruct the timer to stop at 100. Furthermore, you may frequently attach other hardware or peripherals inside the microcontroller to the timer, such as automatically switching a certain pin when the timer rolls over.[16]

Here are some of the common hardware functions you'll see with timers:

- Output compare (OC): toggle a pin when a timer reaches a certain value.
- Input capture (IC): measure the number of counts of a timer between events on a pin.
- Pulse width modulation (PWM): toggle a pin when a timer reaches a certain value and on rollover. By adjusting the on versus off time (duty cycle), you can effectively control the amount of electrical power going to another device.

Prescaler

How fast do timers operate? It depends on how quickly you order them to flee. A clock of some kind is required for all timers. The majority will be linked to the microcontroller's primary CPU clock (others, like real time clocks, have their own clock sources). When a clock pulse is received, a timer will tick (increase by one).

Longer events need the employment of a prescaler, which is a piece of hardware that splits the clock source. A prescaler of 80, for example, would convert an 80 MHz clock to a 1 MHz frequency.

In order to test the timer's function, we will use an STM32F407VGT6 with an 8 MHz default main clock (HCLK) and use its timer as a counter.

LADDER program

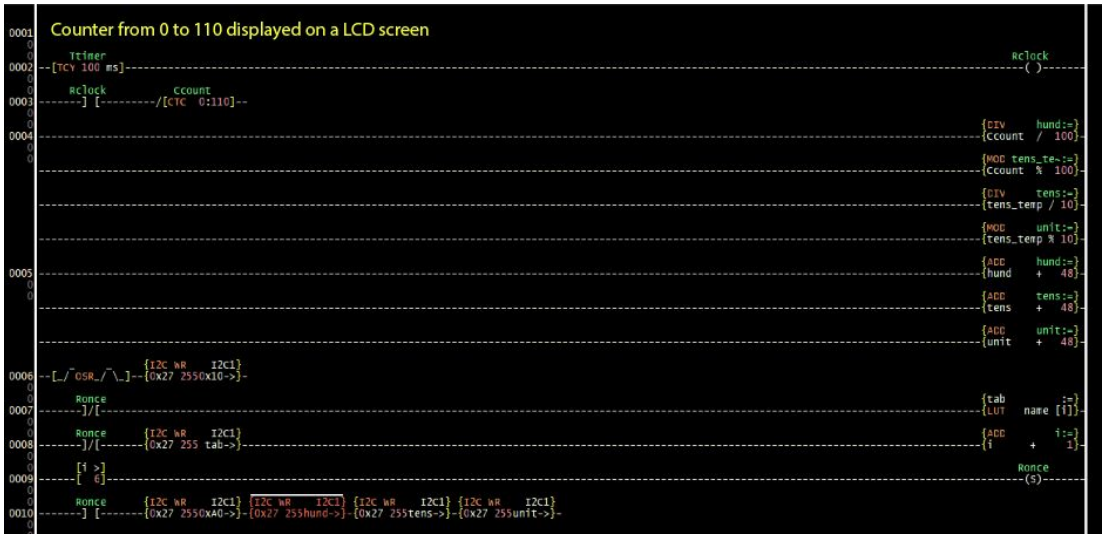


Figure IV.15: Timer interruption test LADDER program.

This program illustrated in the figure above uses our stm32 microcontroller's timer to count from 0 to 110.Once the counting reaches 110, it rolls over 0 and restarts counting in an infinite loop.

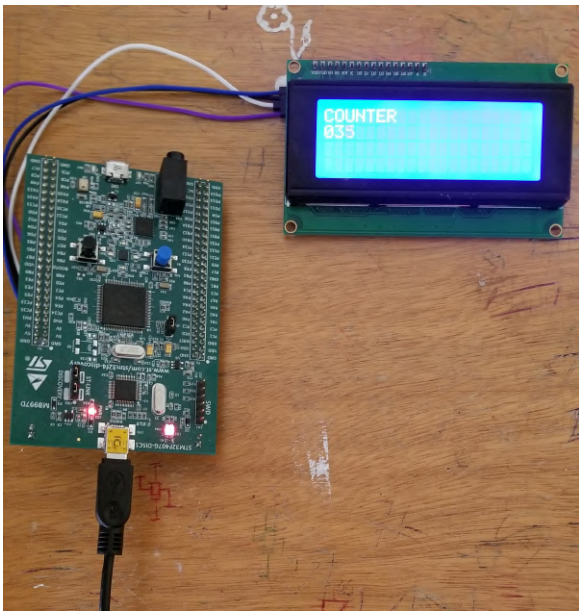


Figure IV.16: Timer/Counter test result.

IV.2.4 Pulse-Width Modulation PWM

PWM (pulse-width modulation) or PDM (pulse-duration modulation) is a technique for decreasing the average power produced by an electrical signal by basically slicing it up into discrete portions. The average value of voltage (and current) provided to the load is regulated by rapidly switching the supply and load switches on and off. The higher the overall power provided to the load, the longer the switch is on relative to the off times. It is one of the principal ways, along with maximum power point tracking (MPPT), of decreasing the output of solar panels to that which can be used by a battery.

PWM is particularly well suited for running inertial loads such as motors, which are less easily influenced by this discrete switching due to their inertia. The PWM switching frequency must be high enough such that it does not damage the load, which means that the resulting waveform perceived by the load must be as smooth as possible.[1]

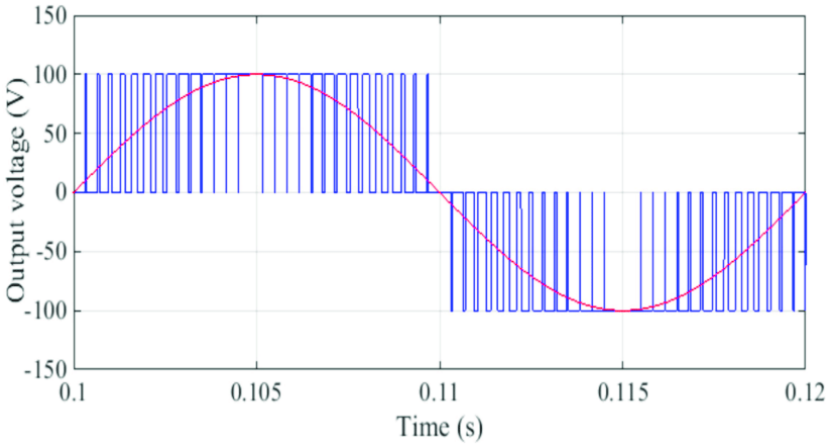


Figure IV.17: Ideal PWM signal.[18]

In order to test our PLC's PWM function, we will be using a potentiometer and a LED light as it is shown in the figure below:

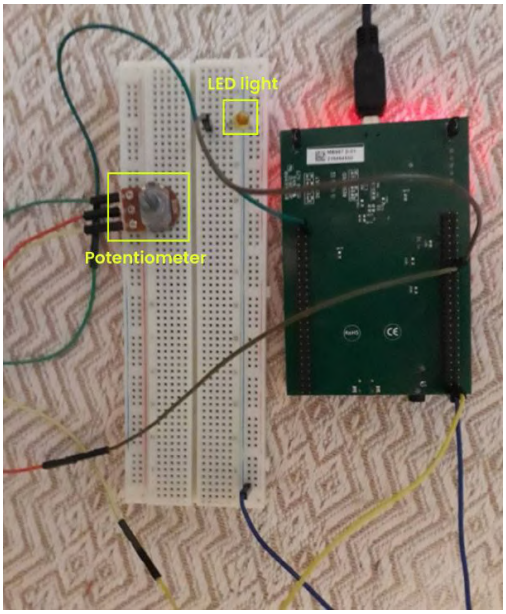


Figure IV.18: PWM application setup.

this application consists of controlling a LED lighting intensity using a potentiometer, the LADDER program is shown in the figure bellow:

LADDER program

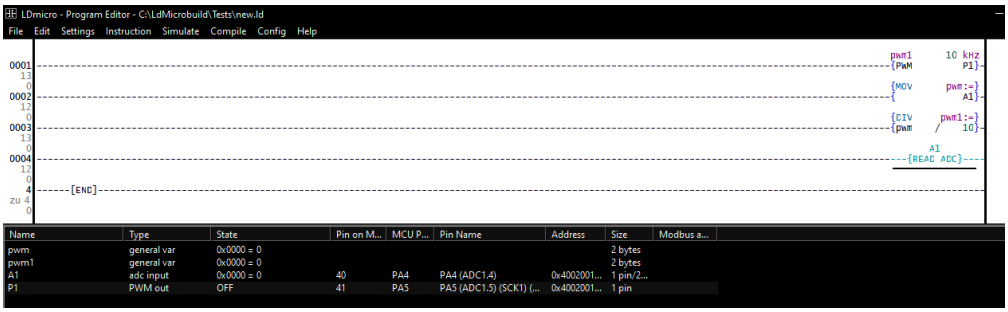


Figure IV.19: PWM application LADDER program.

As a result, decreasing the potentiometer's resistance will cause the LED light to shine even brighter. Increasing the potentiometer's resistance causes to lower the LED light brightness until its off.

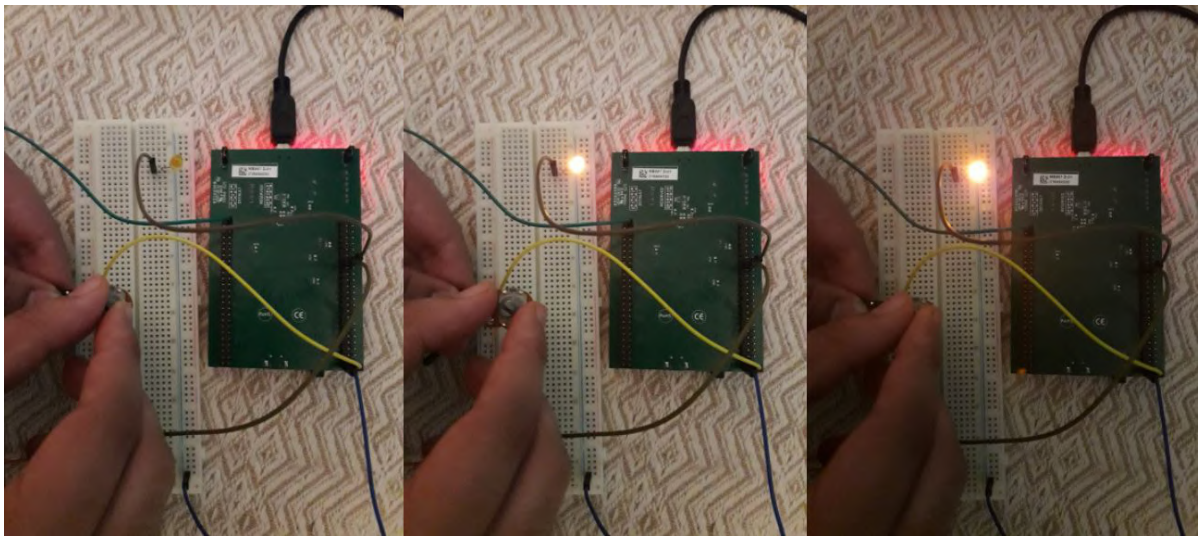


Figure IV.20: PWM application result.

IV.2.5 Real-Time Clock (RTC)

The Time Clock Module is a module that measures the time, dependently or independently of his microcontroller card through of his cell.

The DS1307 Serial Real-Time Clock is a low-power clock/calendar with complete binary-coded decimal (BCD) support and 56 bytes of NV SRAM. A 2-wire, bi-directional bus is used to transport address and data serially. The clock/calendar displays seconds, minutes, hours, days, dates, months, and years. For months with fewer than 31 days, the end of the month date is automatically modified, including leap year corrections. The clock has a 24-hour or 12-hour format and an AM/PM indication. The DS1307 includes a built-in power sensor circuit that detects power outages and switches to the battery supply.[5]

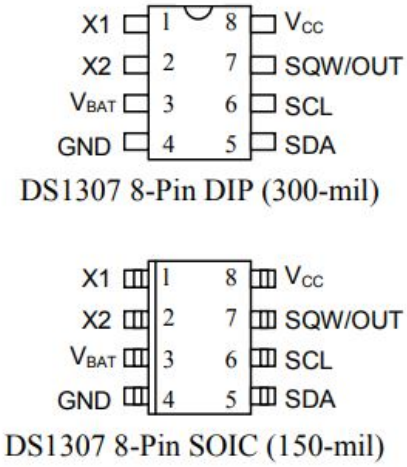


Figure IV.21: RTC Pin Assignment.[5]

Features

- Real-time clock (RTC) counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap-year compensation valid up to 2100.
- 56-byte, battery-backed, nonvolatile (NV) RAM for data storage.
- Two-wire serial interface.
- Programmable squarewave output signal.
- Automatic power-fail detect and switch circuitry.
- Consumes less than 500nA in battery backup mode with oscillator running
- Optional industrial temperature range: -40°C to +85°C.
- Available in 8-pin DIP or SOIC.
- Underwriters Laboratory (UL) recognized.

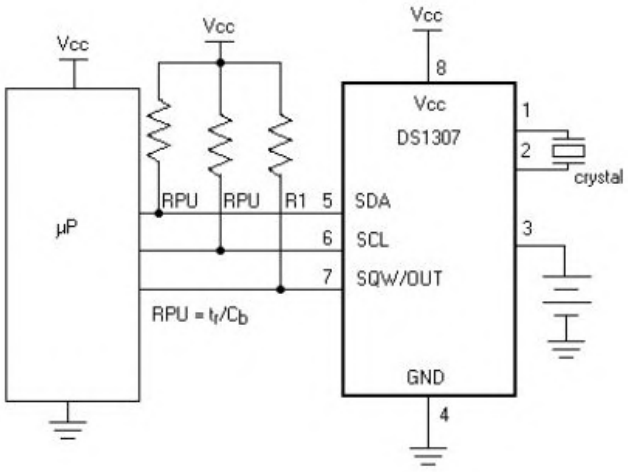


Figure IV.22: RTC typical operating circuit.[5]

In this application we will be using the RTC module to adjust time (hours/minutes/seconds), date (day/month/year) in order to confirm that our PLC's BCD2Binary function works just fine. The application setup is shown in the figure below

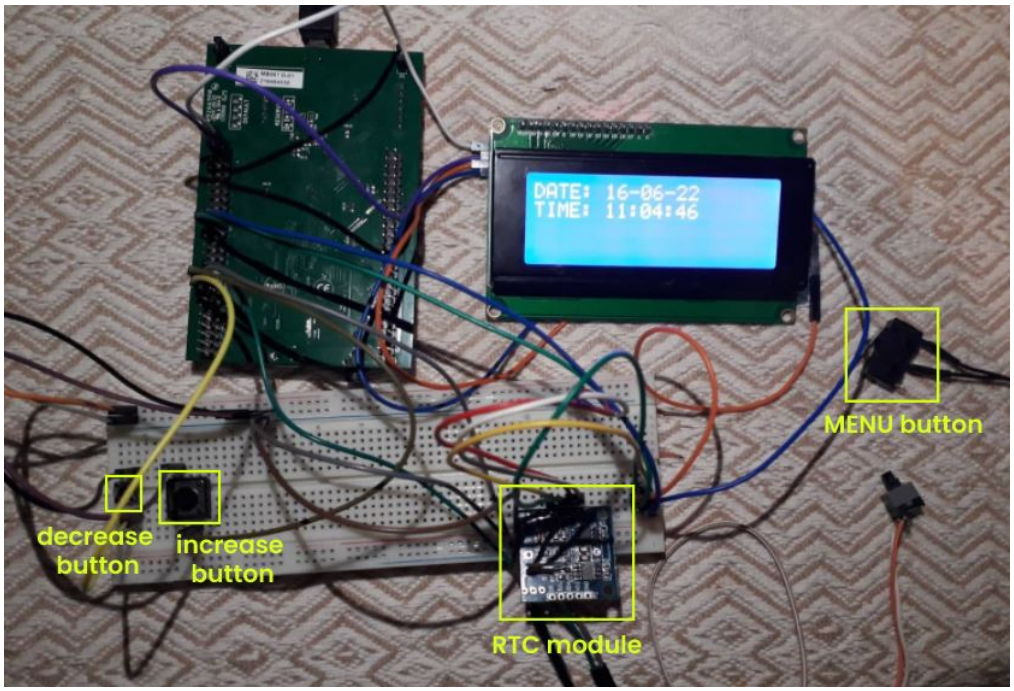


Figure IV.23: RTC application setup.

LADDER program

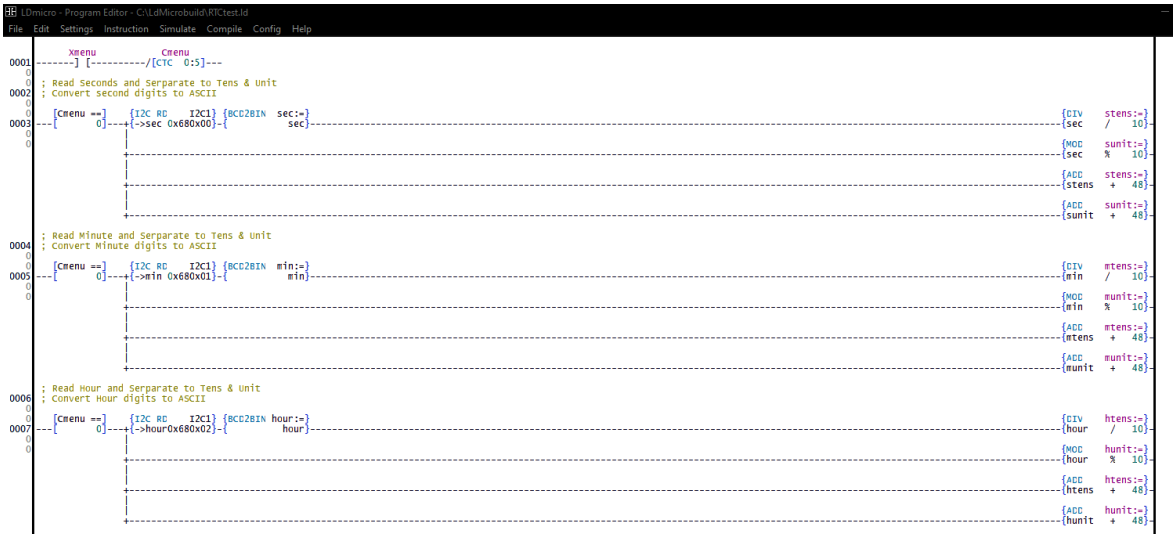


Figure IV.24: Date/Time adjusting code (start).

Create a program in LDmicro to display,the seconds, minutes, hours, dates,months, and years. These can be set with the four push-button switches menu option:

Rung 1 has the menu contact that increases a circular counter through 5 options to set the date, month, year, hour, and minute of the real-time clock. The default counter value of zero select the option to acquire the values of the date and time from the DS3231 module.

In rung 3, when Cmenu is zero, the I2C read instruction, reads the second value at register zero from the module at device address 68 hexadecimal and stores it into variable “sec”. The read value is then converted from BinaryCoded Decimal to Binary by BCD2BIN instruction. The second value is then separated into the two digits, converted to ASCII format, and stored in

variables “stens” and “sunit”.

Similar to rung 3 are rungs 5, 7, 9, 11 and 13.

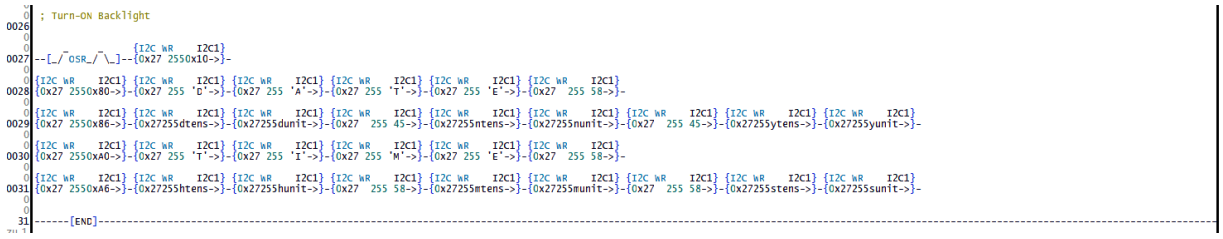


Figure IV.25: Date/Time adjusting code (end).

Rung 27 turns ON the backlight.

Rung 28 will display the text “DATE” starting from column 1, row 1 of the LCD.

Rung 29 displays the acquired values of the date,month and year separated by a dash, starting from position 86 hexadecimal.

Similarly, rung 30 will display the text “TIME”in row two of the LCD. Finally, the Hours, minutes and seconds separated by colons will be displayed on the LCD, starting from position A6 hexadecimal. Configure the push button switch input pins as theSTM32F407 board pins of D0 for Xdec, D1 for Xenter,D2 for Xinc, D3 for Xmenu, B8 for I2C SCL and B9 for I2C SDA. Compile the program as before,build C solution and call flashMCU.

Note: the rest of LADDER program can be found in the appendix the a proper explanation to every rung.

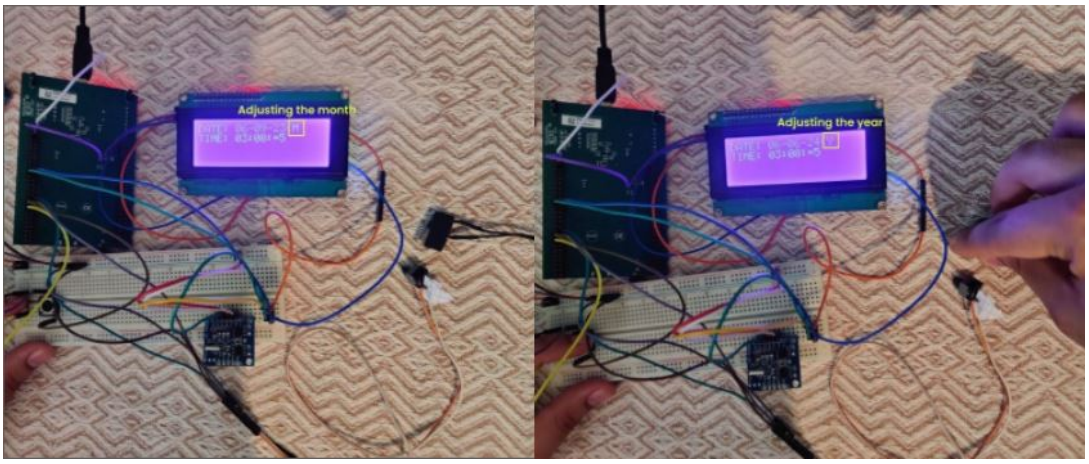


Figure IV.26: Month/Year adjusting.

Here is the display of the calendar and time:

To adjust the calendar and time, press the menu button once. We observe a less than D at the top right of the LCD. Repeatedly press the increase or decrease button to adjust to today’s date.

Then press the enter button to save. Press the menu button again to adjust the month, repeating the previous steps. Press the menu button to set the year, hour, and minute, observing the indication of the parameter to be set on the right side of the LCD, until the current date and time is being displayed on the LCD.

IV.2.6 Serial communication

In this application, we will use two stm32f407 microcontrollers and establish a Master/Slave communication using the UART protocol as it is shown in the figure below:

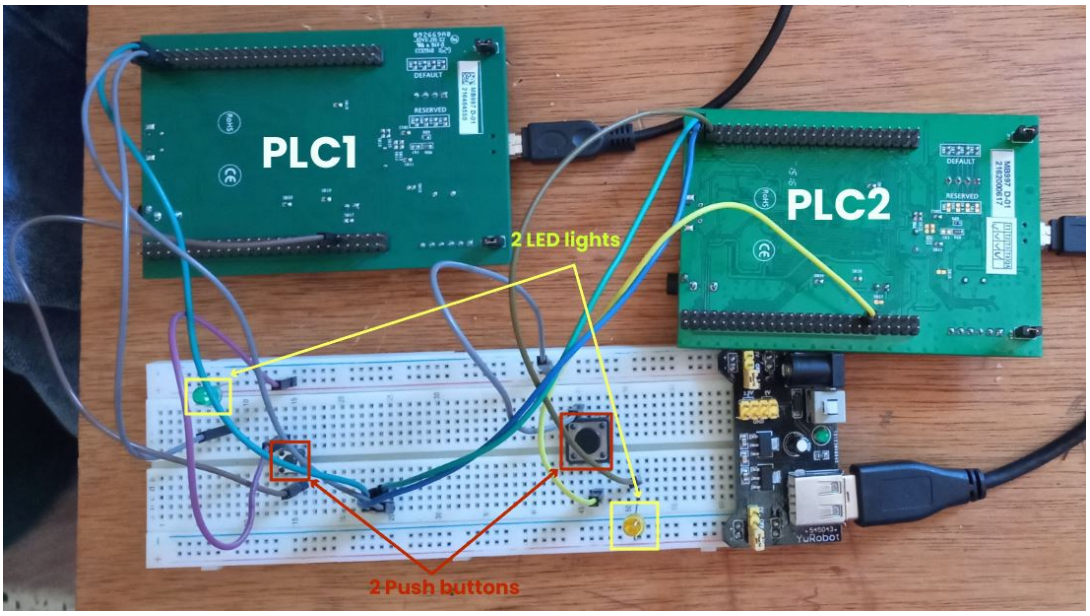


Figure IV.27: 2 STM32 serial communication.

LADDER program

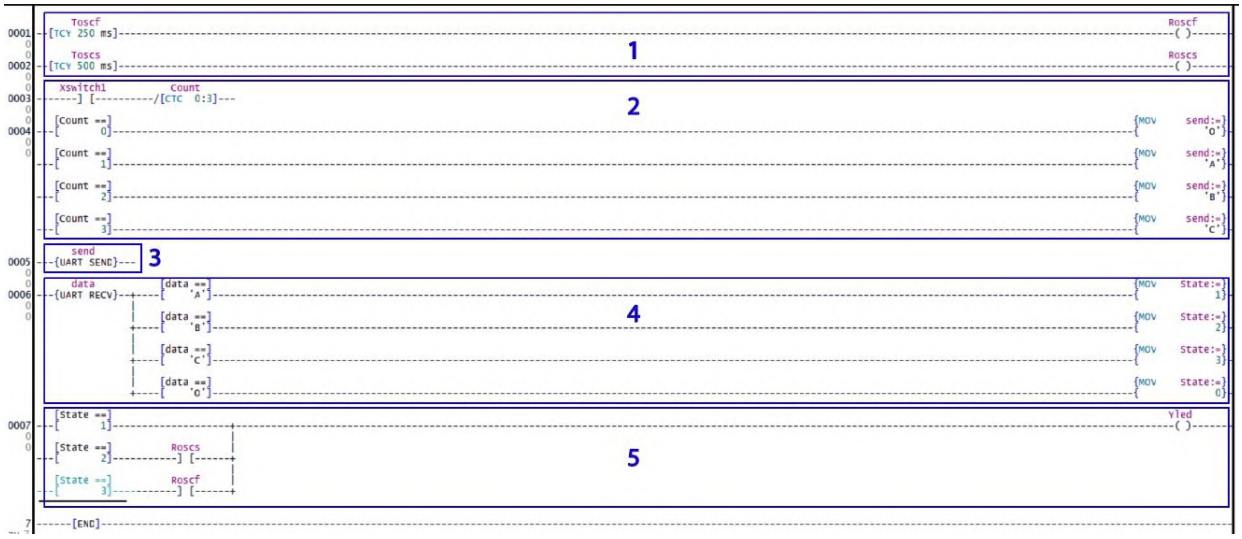


Figure IV.28: Serial communication LADDER program.

1. We start of by inserting two **Cyclic Timers** in the first two rungs linked to an internal relay coil for each one of them and with the duration of 500ms for slow blinking and 250ms for fast blinking.

2. Initiate a ’0 to 3’ counter and insert a **Compare for Equal** instructions, each one of them gets a ’0, 1, 2 and 3’ respectively in line with the four **MOV** instructions with their destination values as “send” and the source values as ’O, A, B and C’.

3. Insert a **UART send** instruction with the source variable of “send”. This instruction transmits earlier stored characters in the “send” variable by the **MOV** instructions.

4. Insert a **UART receive** instruction with the destination variable of data then insert four **Compare for Equal** instructions in line with four **MOV** instructions after **UART RECEIVE**. The “data” variable stores the received UART data which is then compared to determine the required state of the LED.

5. Insert three **Compare for Equal** instructions in which each one of them will trigger the LED state based on the received information from the other STM32 microcontroller.

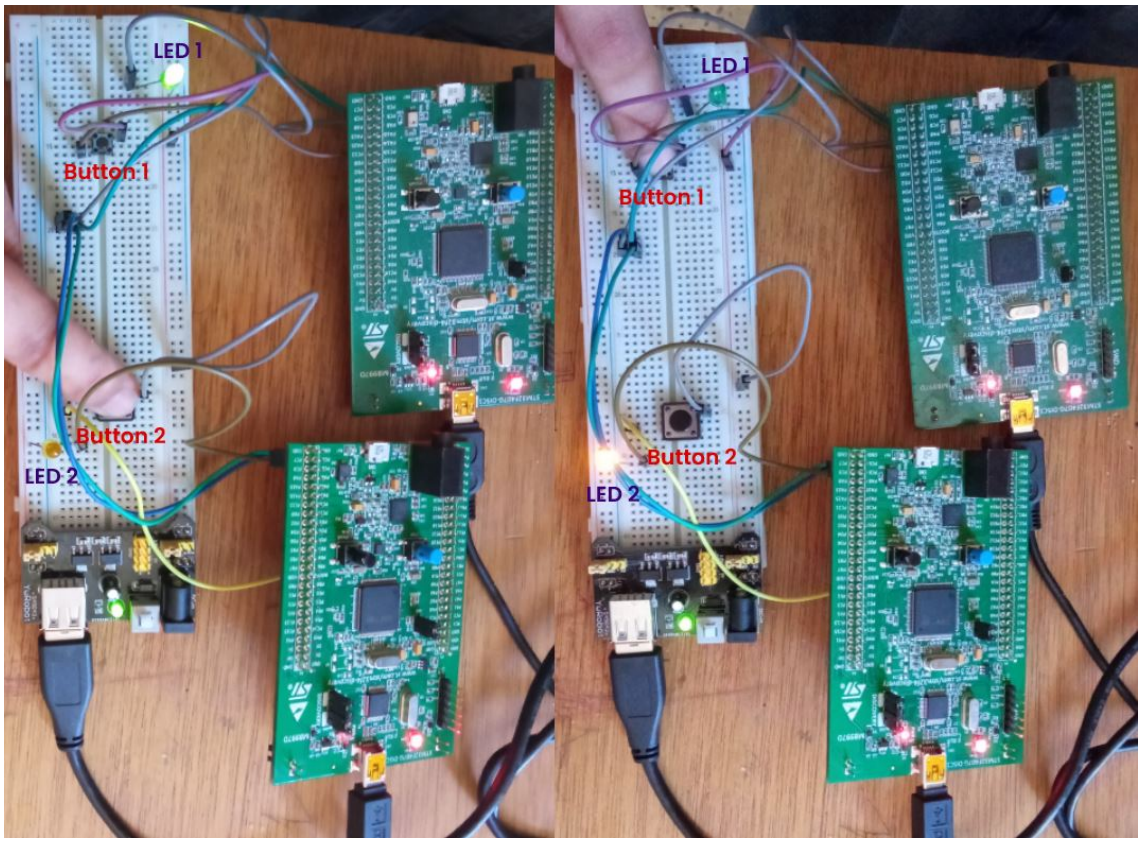


Figure IV.29: Serial communication result.

So as a result, if we push on a Button of a PLC, this will send a ’A state’ which will cause the LED of the other PLC to turn on. If we push a second time, the LED will start blinking slowly (State ’B’). Finally, if we push the Button for a third time, the LED will start blinking rapidly (State ’C’). A Fourth push on the button will turn off the LED light.

IV.3 PLC final version

Our final PLC model prototype and its main components are shown in the figure down below:

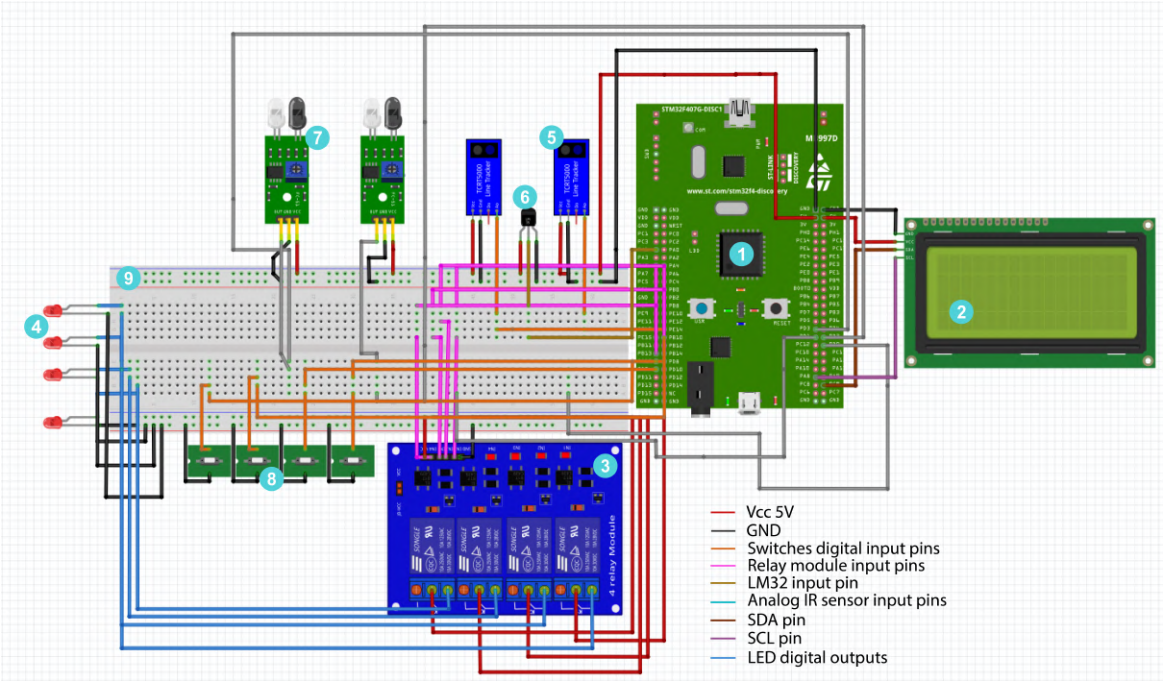


Figure IV.30: PLC final prototype.

1. STM32F407 microcontroller discovery board.
2. 20x4 LCD screen with integrated I2C interface.
3. 4 relays module 5V.
4. 4x LED lights.
5. 2x Analog line tracker TCRT5000.
6. Temperature sensor LM35.
7. 2x Proximity sensor FC-51.
8. 4x Digital inputs (switches).
9. Bread board.

IV.4 Conclusion

In this chapter, we made sure that our final PLC prototype is able to handle and monitor most of the applications needed in a automated process such as digital and analog I/Os, timers, RTC, PWM, communication protocols and the serial communication with other PLCs alongside with verifying our software architecture and elaborating a method on how to add a new microcontroller to the software architecture if needed.

STM32F407 had a major role into making a powerful and efficient processor due to its processing power, flash memory, connectivity and reliability.

As a result, our PLC prototype is efficient and has an acceptable performance in real-time control processes and with that, we can say that our PLC is compliant with the requirements of the industry 4.0 and to answer globally to the same requirements expected from a classical PLC, in particular the possibility of being programmed and reprogrammed with a non structural language via a compiler which was our main goal.

General conclusion

General conclusion

As technology progresses, technical investment must continue to be made for the ongoing advancement of these vital approaches that lift the automation control field to ever more intriguing levels of efficient and economical automation.

Even with Industry 4.0 and other ”Smart Factory” projects, the PLC is still regarded as the primary control center for production machinery. The PLC may be thought of as the central processor for all real-time production decisions. Input sensors will immediately send their status to the PLC, allowing the PLC to create what is needed at that time.

The goal of this work was to elaborate a PLC model able to control an automation process: We introduced the main fundamentals of a PLC, its hardware architecture, its programming languages. In order to achieve our goal, we relied on Ldmicro application in which played a huge part in order to program our PLC using LADDER language and convert it into native hexadecimal file to build a custom and efficient PLC capable of handling most of the automation processes in the industry.

The architecture of our PLC is mainly designed for the small automation processes, the number of inputs/outputs is limited.

The use of STM32F407 microcontroller allowed us to upgrade our PLC performances such as power efficiency, rapidity in collecting data, scan time and thus meet industry 4.0 requirements.

We can state that we have attained our goal and that our prototype can be modified to produce a commercial industrial product.

STM32F407 plays the major role as the low-cost and efficient processor of the PLC prototype providing huge processing power, flash memory and reliability which make it an excellent choice for our controller.

Some of the difficulties we encountered in our work are the instabilities in the STM32 standard libraries and its complexity using Ldmicro compared to the other microcontrollers.

The PLC elaborated in this work confirmed to be effective and fantastically appropriate alternative in terms of low-cost Programmable Logic Controllers that offer suitable conduct in the essential scan time.

Bibliography

[3] *4 Channel 5V Optical Isolated Relay Module user guide*. Handson Technology.

[4] *4 relay module data sheet*.

[5] *64 x 8 Serial Real-Time Clock*. DALLAS SEMICONDUCTOR.

[11] Anthony Bishop. *Solid-State Relay Handbook with Applications*. Sams Technical Publishing, 1986.

[14] *Discovery kit with STM32F407VG MCU User manual*. STmicroelectronics.

[17] *I2C Serial Interface 20x4 LCD Module*. Handson Technology.

[26] *Schneider Electric MODICON M221 Logic Controller Datasheet*. Schneider Electric.

[27] *Schneider Electric MODICON M241 Logic Controller Datasheet*. Schneider Electric.

[28] *Schneider Electric MODICON M241 Logic Controller Datasheet*. Schneider Electric.

[29] *Schneider Electric Zelio Logic SR3BB101FU Datasheet*. Schneider Electric.

[33] *Siemens LOGO230RCE Datasheet*. Siemens.

[34] *Siemens SIMATIC S7-1200 Basic Controller Datasheet*. Siemens.

[35] *Siemens SIMATIC S7-1500 Advanced Controller Datasheet*. Siemens.

[36] *Siemens SIMATIC S7-300 Advanced Controller Datasheet*. Siemens.

[37] *Siemens SIMATIC S7-400 Advanced Controller Datasheet*. Siemens.

[42] *STM32F401xD STM32F401xE Data Sheet*. STMicroelectronics.

[47] Michael Tiegelkamp and Karl-Heinz John. *IEC 61131-3: Programming industrial automation systems*. Vol. 166. Springer, 2010.

Webography

[1] *"Sizing a Grid-Tied PV System ...with Battery Backup"*. en-US. URL: https://www.homepower.com/home (visited on 2022).

[2] *12 Volt DC Solid State Relay using BUZ71A*. en-US. URL: https://circuits-diy.com/12-volt-dc-solid-state-relay-using-buz71a/ (visited on 2022).

[6] *6N137 Optocoupler*. en-US. URL: https://ardustore.dk/produkt/6n137-optocoupler (visited on 2022).

[7] *a-brief-history-of-programmable-logic-controllers-plcs*. en-US. URL: https://processsolutions.com/a-brief-history-of-programmable-logic-controllers-plcs/ (visited on 2022).

[8] *AC SOLID STATE RELAY*. en-US. URL: https://www.electronics-lab.com/project/ac-solid-state-relay/ (visited on 2022).

[9] *Automate Programmable Industriel TS CRSA*. fr-FR. URL: http://bts.crsa.rascol.free.fr/Automatismes/cours/Les%5C%20Automates%5C%20Programmables%5C%20Industriels.pdf (visited on 2022).

[10] *BASICS OF THE SPI COMMUNICATION PROTOCOL*. en-US. URL: https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/ (visited on 2022).

[12] *Choosing the Right PLC: 6 Things to Consider ,Mike Jeffries | February 15, 2021*. en-US. URL: https://www.maderelectricinc.com/blog/choosing-the-right-plc-6-things-to-consider (visited on 2022).

[13] *CPU Architecture*. en-US. URL: https://eeeproject.com/microprocessor-vs-microcontroller/ (visited on 2022).

[15] *Electromecanical Relay*. en-US. URL: https://www.fasttech.com/category/1005-listing-removed (visited on 2022).

[16] *Getting Started with STM32 - Timers and Timer Interrupts*. en-US. URL: https://www.digikey.com/en/maker/projects/getting-started-with-stm32-timers-and-timer-interrupts/d08e6493cefa486fb1e79c43c0b08cc6 (visited on 2022).

[18] *Ideal PWM signal*. en-US. URL: https://www.researchgate.net/figure/Ideal-pulse-width-modulation-PWM-inverter-output-voltage-instantaneous-component-blue_fig4_319071311 (visited on 2022).

[19] *INTERNAL ARCHITECTURE - PROGRAMMABLE LOGIC CONTROLLERS*. en-US. URL: https://www.wisdomjobs.com/e-university/programmable-logic-controllers-tutorial-523/internal-architecture-14628.html (visited on 2022).

[20] *LM35 Temperature Sensor Pin out, Interfacing guide, Circuit Construction and Working Principals*. en-US. URL: https://www.engineersgarage.com/lm35-description-and-working-principal/ (visited on 2022).

[21] *PLC Programmable Logic-Controller Hardware-Components PLC Hardware-Basics*. en-US. URL: https://electricala2z.com/motors-control/plc-programmable-logic-controller-hardware-components-plc-hardware-basics/ (visited on 2022).

[22] *PLC-Inputs-and-Outputs-(I/O)*. en-US. URL: https://www.automation.com/en-us/articles/2018/a-beginners-plc-overview-part-3-of-4-plc-inputs-an (visited on 2022).

[23] *Power Supply Module*. en-US. URL: https://yduf.github.io/arduino-power/ (visited on 2022).

[24] RealPars. *PLC-hardware*. en-US. URL: https://realpars.com/plc-hardware/ (visited on 2022).

[25] RealPars. *WHAT ARE THE MOST POPULAR PLC PROGRAMMING LANGUAGES?* en-US. URL: https://realpars.com/plc-programming-languages/ (visited on 2022).

[30] *Serial Communication – Protocols, Standards and Advantages*. en-US. URL: https://electricalfundablog.com/serial-communication-protocols-standards/ (visited on 2022).

[31] *Serial communication Basic Knowledge*. en-US. URL: https://www.contec.com/support/basic-knowledge/daq-control/serial-communicatin/ (visited on 2022).

[32] *Serial Wire Debug (SWD)*. en-US. URL: https://community.silabs.com/s/article/serial-wire-debug-swd-x?language=en_US (visited on 2022).

[38] *Solid State Relay*. en-US. URL: https://sktechworks.ca/product/solid-state-relay-module/ (visited on 2022).

[39] *Sourcing and Sinking*. en-US. URL: https://blog.orientalmotor.com/what-is-the-difference-between-sink-and-source-logic (visited on 2022).

[40] *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32*. en-US. URL: https://www.st.com/en/development-tools/st-link-v2.html (visited on 2022).

[41] *Step by Step Procedure of PLC Programming in Industries*. en-US. URL: https://www.watelectronics.com/how-to-program-the-programmable-logic-controllers/ (visited on 2022).

[43] *STM32F407 Microcontroller*. en-US. URL: https://sciencezero.4hv.org/index.php?title=STM32F407_Microcontroller (visited on 2022).

[44] *stm32f407/417 main features*. en-US. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f407-417.html (visited on 2022).

[45] *SWO with CubeMX using ST-Link clones*. en-US. URL: http://eeblog.co.uk/tag/stm32/ (visited on 2022).

[46] *Things to consider before buying a PLC*. en-US. URL: https://www.automationit.com/blog/80-choosing-a-plc-things-to-consider (visited on 2022).

[48] *Understanding Firmware Updates: The Whats, Whys, and Hows*. en-US. URL: https://www.einfochips.com/blog/understanding-firmware-updates-the-whats-whys-and-hows/ (visited on 2022).

[49] *Using Your STM32F4 Discovery Board as a Programmer and Debugger*. en-US. URL: https://geekilyinteresting.wordpress.com/2014/05/04/using-your-stm32f4-discovery-board-as-a-programmer-and-debugger/ (visited on 2022).

[50] *Waveshare STM32 Discovery Kit for STM32F4 with STM32F407 MCU STM32F4DISCOVERY On-board ST-LINK/V2 Cortex-M4F STM32 Development Board*. en-US. URL: https://www.amazon.co.uk/Waveshare-Discovery-STM32F4DISCOVERY-Cortex-M4F-Development/dp/B01DNYSTJ8 (visited on 2022).

[51] *What is a SPI Protocol in the Computer Network?* en-US. URL: https://www.tutorialspoint.com/what-is-a-spi-protocol-in-the-computer-network (visited on 2022).

[52] *What is a temperature sensor?* en-US. URL: https://www.fierceelectronics.com/sensors/what-a-temperature-sensor (visited on 2022).

[53] *What is I2C?* en-US. URL: https://www.teachmemicro.com/i2c-primer/ (visited on 2022).

Appendix

.1 Ldmicro functions

.1.1 MOVE

When the input to this instruction is true, it sets the given destination variable equal to the given source variable or constant. When the input to this instruction is false nothing happens. You can assign to any variable with the move instruction; this includes timer and counter state variables, which can be distinguished by the leading ‘T’ or ‘C’. For example, an instruction moving 0 into ‘Tretentive’ is equivalent to a reset (RES) instruction for that timer. This instruction must be the rightmost instruction in its rung.



Figure 31: MOVE function.

.1.2 COMPARE

If the input to this instruction is false then the output is false. If the input is true then the output is true if and only if the given condition is true. This instruction can be used to compare (equals, is greater than, is greater than or equal to, does not equal, is less than, is less than or equal to) a variable to a variable, or to compare a variable to a 16-bit signed constant.

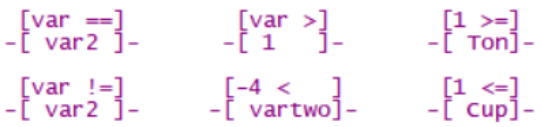


Figure 32: COMPARE function.

.1.3 CIRCULAR COUNTER

A circular counter works like a normal CTU counter, except that after reaching its upper limit, it resets its counter variable back to 0. For example, the counter shown above would count 0, 1, 2, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, ... This is useful in combination with conditional statements on the variable ‘Cname’;

you can use this like a sequencer. CTC counters clock on the rising edge of the rung input condition condition. Obsolete: This instruction must be the rightmost instruction in its rung.

CTR is reversive circular counter. It like CTC but with decrement counting inside. For example, the counter shown above would count 7, 6, 5, 4, 3, 2, 1, 0, 7, 6, 5, 4, 3, 2, 1, 0, 7, ...

Added ’Start’ value in CTU,CTD,CTC,CTR counters. CTU,CTC counter start from ’Start’ and count up to ’Max’ value. CTC generate overfill(carry) impulse when Counter==Max. For the proper work ’Start’ value should be less than ’Max’ value.

CTD,CTR counter start from ’Start’ and count down to ’Min’ value. CTR generate overfill(borrow) impulse when Counter==Min. For the proper work ’Start’ value should be more than ’Min’ value.



Figure 33: CIRCULAR COUNTER function.

.1.4 LOOK-UP TABLE

A look-up table is an ordered set of n values. When the rung-in condition is true, the integer variable ‘dest’ is set equal to the entry in the lookup table corresponding to the integer variable ‘i’. The index starts from zero, so ‘i’ must be between 0 and (n-1). The behaviour of this instruction is not defined if the index is outside this range. This instruction must be the rightmost instruction in its rung.

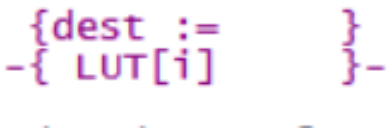


Figure 34: LOOK-UP TABLE function.

.1.5 A/D CONVERTER READ

LDmicro can generate code to use the A/D converters built in to certain microcontrollers. If the input condition to this instruction is true, then a single sample from the A/D converter is acquired and stored in the variable ‘Aname’. This variable can subsequently be manipulated with general variable operations (less than, greater than, arithmetic, and so on). Assign a pin to the ‘Axxx’ variable in the same way that you would assign a pin to a digital input or output, by double-clicking it in the list at the bottom of the screen. If the input condition to this rung is false then the variable ‘Aname’ is left unchanged.

For all currently-supported devices, 0 volts input corresponds to an ADC reading of 0, and an input equal to Vdd (the supply voltage) corresponds to an ADC reading of 1023. If you are using an AVR, then connect AREF to Vdd. You can use arithmetic operations to scale the reading to more convenient units afterwards, but remember that you are using integer math. In general not all pins will be available for use with the A/D converter. The software will not allow you to

assign non-A/D pins to an analog input. This instruction must be the rightmost instruction in its rung.

Updated: ADC Voltage Reference. The reference voltage for the ADC (Vref) indicates the conversion range for the ADC. Single-ended channels that exceed Vref will result in codes close to 0x3FF(1023). Vref can be selected as either AVCC, internal 2.56V reference, internal 1.1V reference, or external Aref pin. The parameter REFS=0 is backward compatible with the previous LDmicro releases.

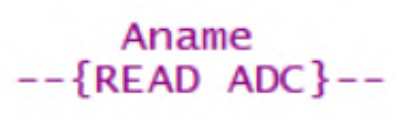


Figure 35: A/D function.

.1.6 UART (SERIAL) RECEIVE

LDmicro can generate code to use the UART built in to certain microcontrollers. On AVRs with multiple UARTs only UART1 (not UART0) is supported. Configure the baud rate using Settings -> MCU Parameters. Certain baud rates may not be achievable with certain crystal frequencies; LDmicro will warn you if this is the case.

If the input condition to this instruction is false, then nothing happens. If the input condition is true then this instruction tries to receive a single character from the UART. If no character is read then the output condition is false. If a character is read then its ASCII value is stored in ‘var’, and the output condition is true for a single PLC cycle.

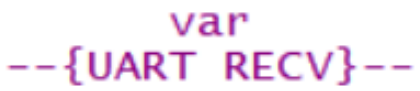


Figure 36: UART (SERIAL) RECEIVE function.

.1.7 UART (SERIAL) SEND

LDmicro can generate code to use the UARTs built in to certain microcontrollers. On AVRS with multiple UARTs only UART1 (not UART0) is supported. Configure the baud rate using Settings -> MCU Parameters. Certain baud rates may not be achievable with certain crystal frequencies; LDmicro will warn you if this is the case.

If the input condition to this instruction is false, then nothing happens. If the input condition is true then this instruction writes a single character to the UART. The ASCII value of the character to send must previously have been stored in ‘var’. The output condition of the rung is true if the UART is busy (currently transmitting a character), and false otherwise.

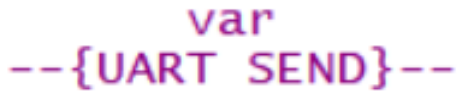


Figure 37: UART (SERIAL) SEND function.

.1.8 UART (SERIAL) SENDn

If the input condition to this instruction is false, then nothing happens. If the input condition is true then this instruction writes a variable to the UART in raw binary format. One byte sends per cycle of PLC, like 'FORMATTED STR OVER UART' element. The output condition of the rung is true if the UART is busy (currently transmitting a variable), and false otherwise.

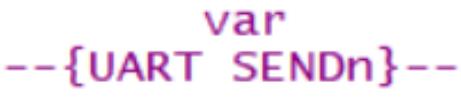


Figure 38: UART (SERIAL) SENDn function.

.1.9 I2C READ

I2C functions are only available in 'C' mode, that is via C compiling, only for 32-bit ARM and AVR of type AtMega. Only one I2C bus can be used in a same ladder, called I2C, or I2C1, I2C2, I2C3 when there are several available on the micro. I2C works only in master mode, and can access any register (0 based) at any address (1-127) on the I2C bus. I2C frequency is set in the configuration panel of the micro. I2C READ function reads a byte in a register on the slave at given address.

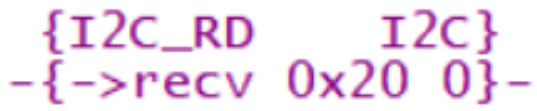


Figure 39: I2C READ.

.1.10 I2C WRITE

I2C WRITE function writes a byte in a register on the slave at given address.

NB: I2C functions use external C libraries (furnished) which must be compiled with the main C program generated by ldmicro when compiling via ldmicro menus < Compile Atmel AVR-GCC > or < Compile ARM-GCC >, followed by a < FlashMCU > command to build the .hex executable, and, if well configured, load this executable into the AVR or ARM target.

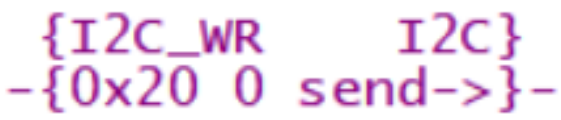


Figure 40: I2C WRITE.

.2 How to generate the .hex file and load it into the microcontroller?

1. We start of by compiling our LADDER program into a C code file using the ARM GCC compiler by going to 'compile > compile ARM-GCC for 32-bit ARM'.

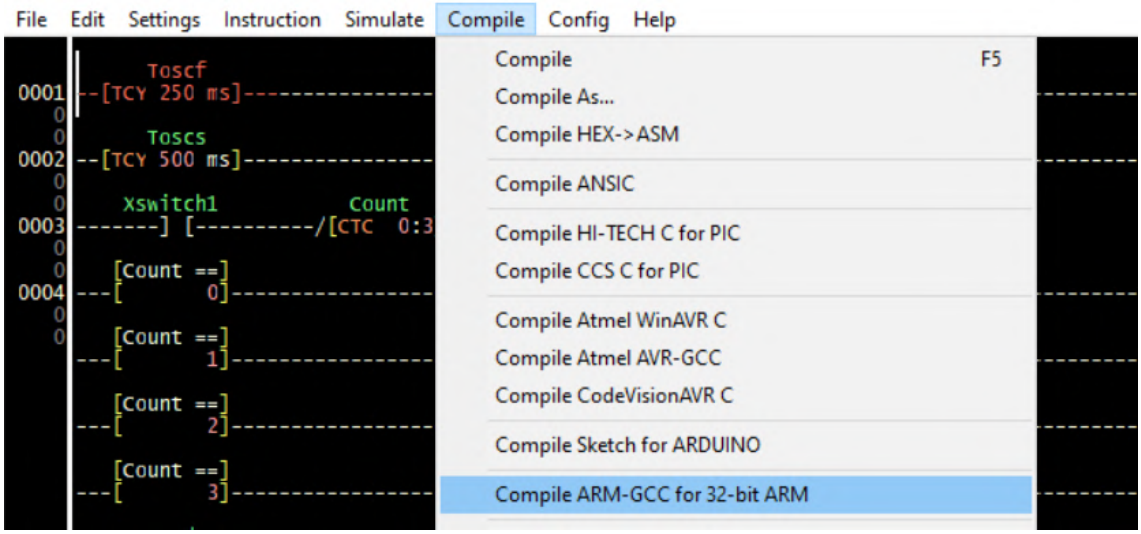


Figure 41: Compile into a C code.

2. Then we generate the .hex file by going into 'compile > Build C solution'.

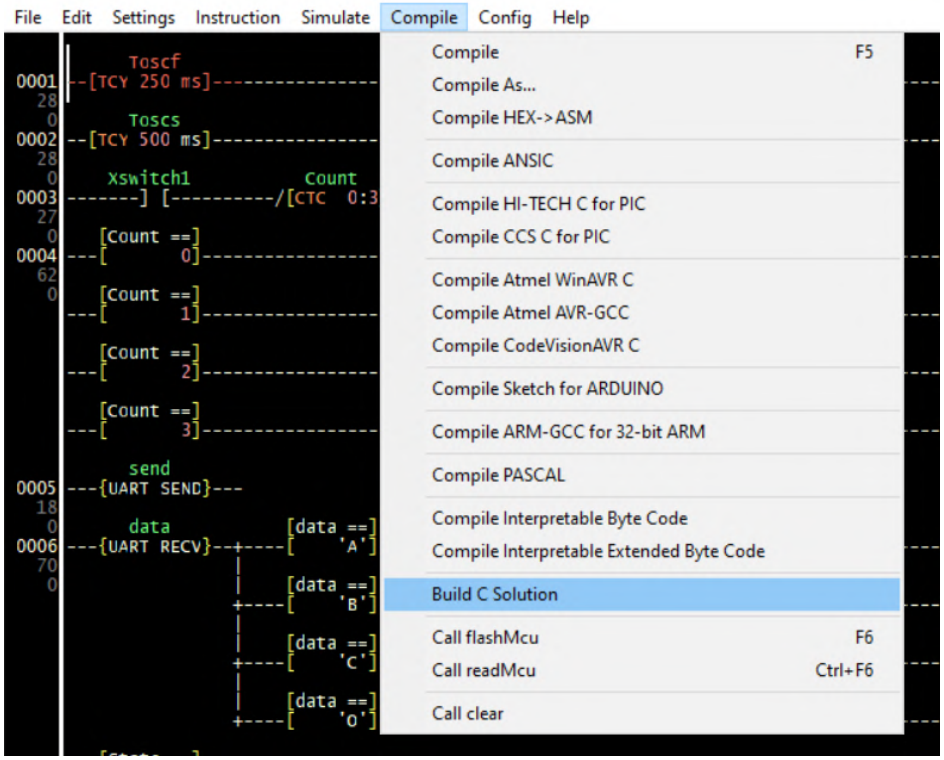


Figure 42: Generating the .hex file.



Figure 43: .hex file generated.

3. the last step consists of loading the .hex file into the microcontroller's CPU by calling FLASH MCU in 'compile > CallFlash MCU'.



Figure 44: Program implementation.

the program is now loaded into our microcontroller's CPU without errors and ready to be executed.

.3 Adding a new microcontroller to Ldmicro (more details)

1. We start of by defining the general MCU informations such as PORTS, PORT addresses, the flash memory size and declare pins info libraries, CPU type, and all the communication protocols (UART, I2C, SPI).

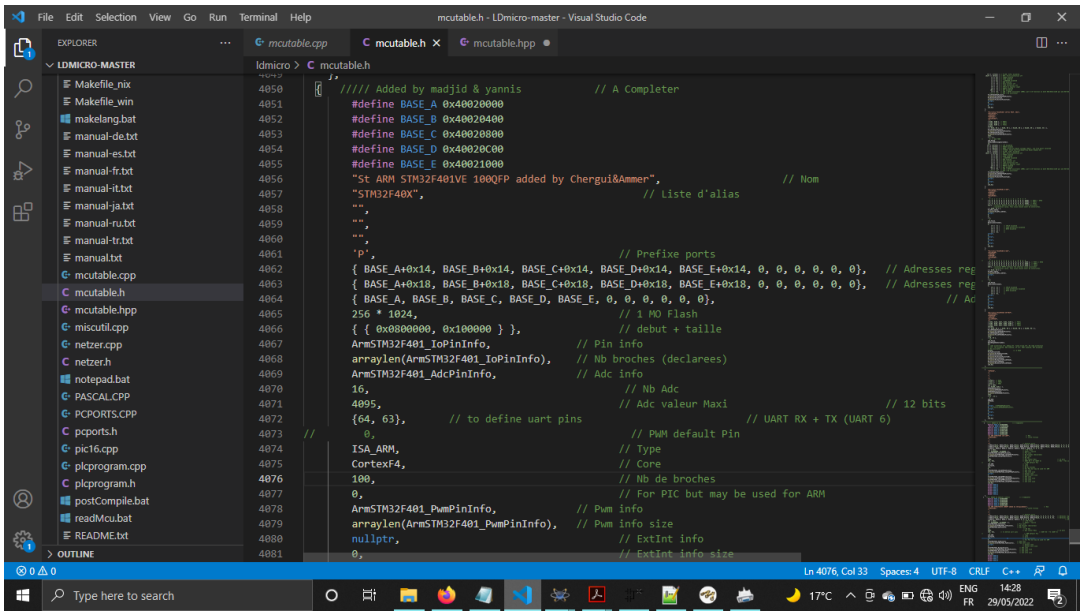


Figure 45: General MCU info.

2. Define all the card I/O pins and their functions.

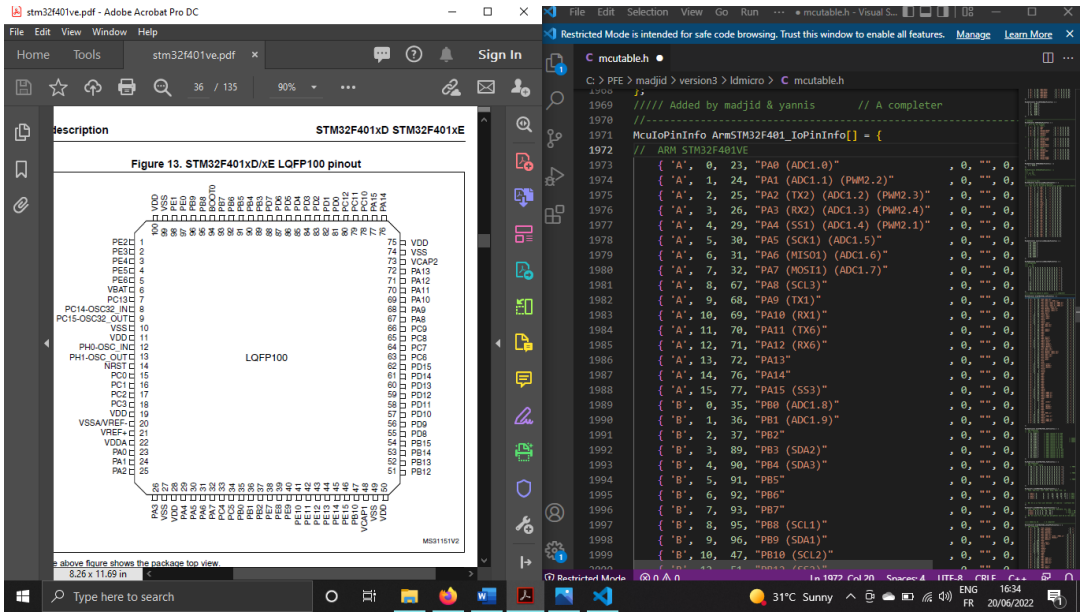


Figure 46: MCU I/O pins infos.

3. Define the communication protocols pins infos.

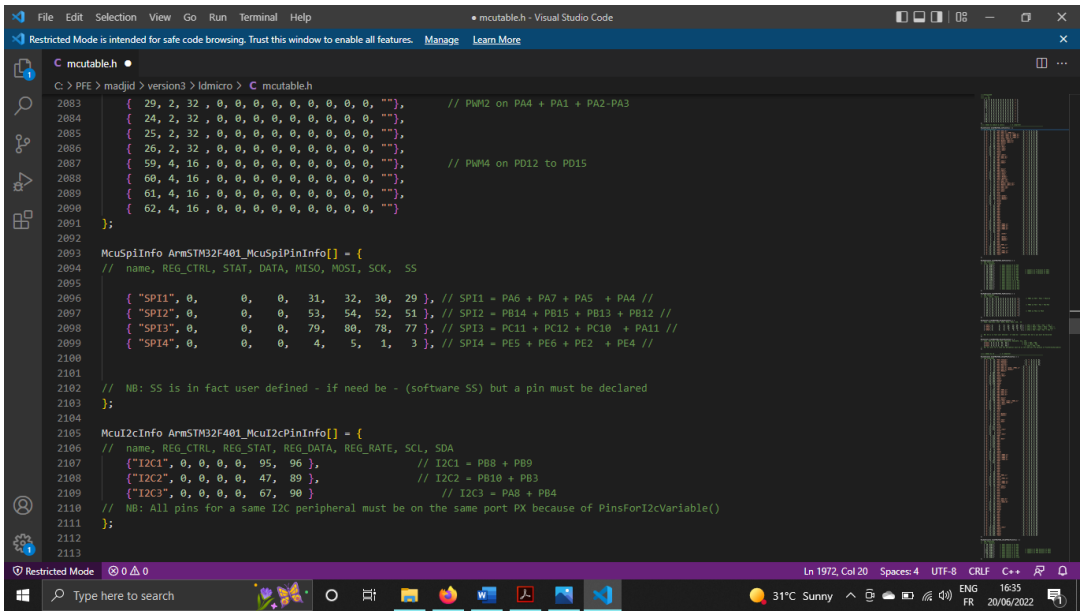


Figure 47: SPI/I2C pins infos.

4. Define the PWM pin infos.

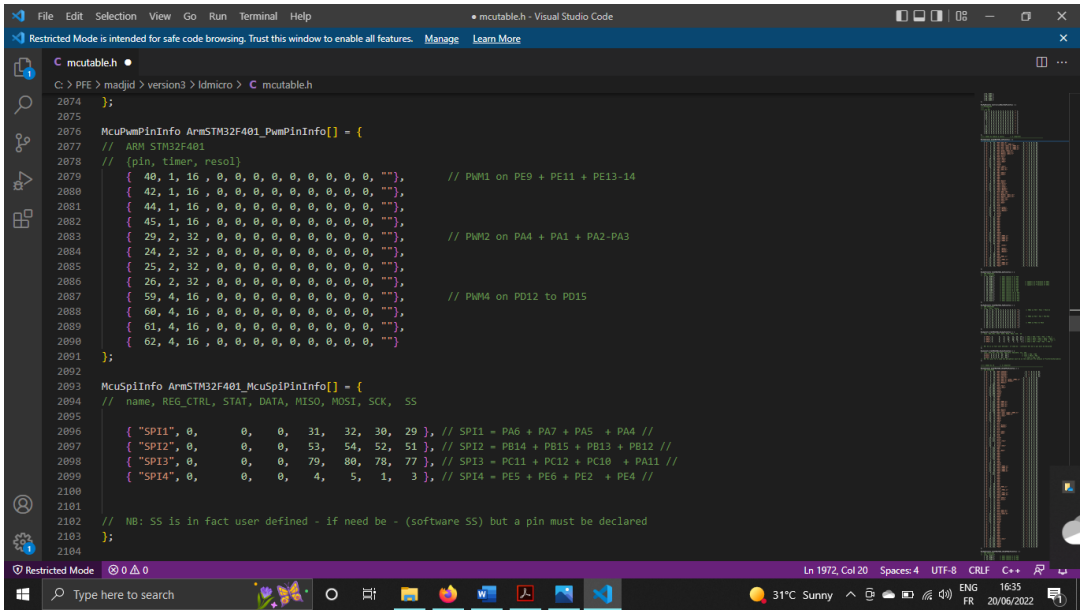


Figure 48: PWM pin info.

5. Define the ADC pins and channels info.

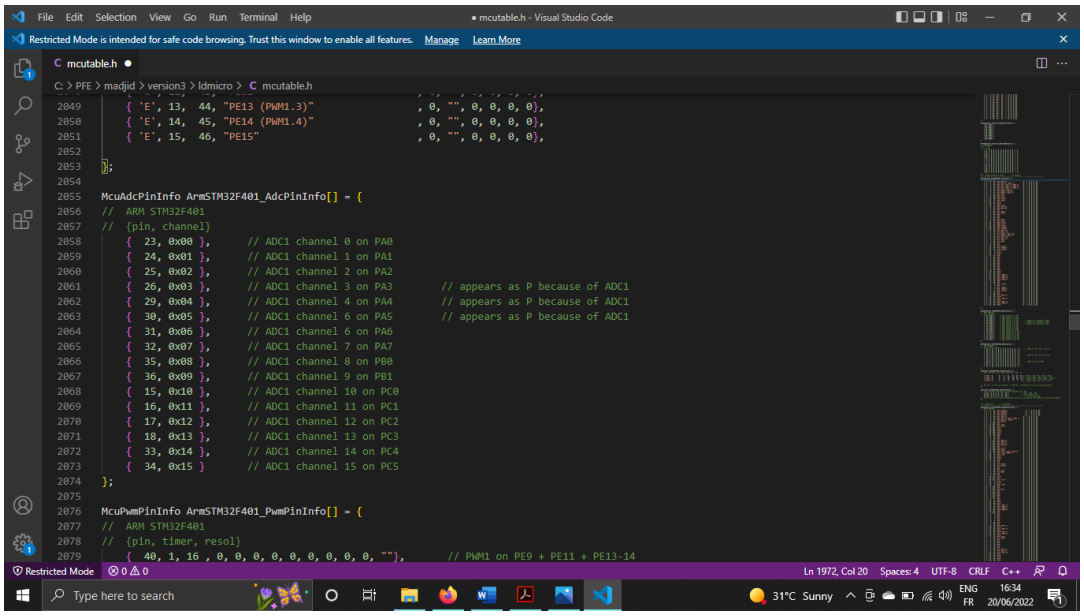


Figure 49: ADC pins/channels info.

Note 1: All these steps are declared according to a structure elaborated by Ldmicro's first developer in the mcutable.hpp header file.

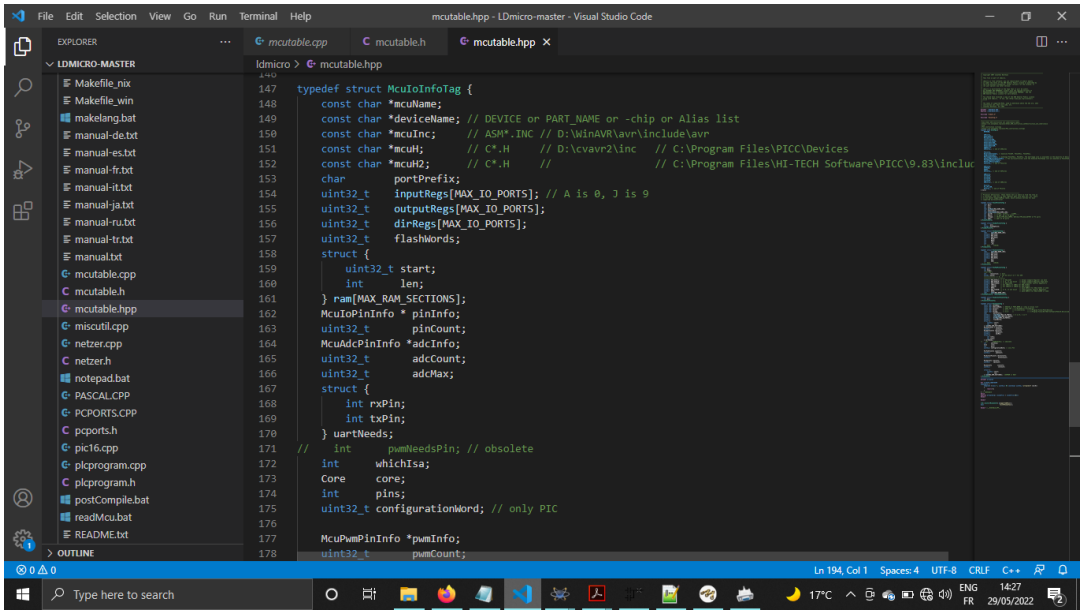


Figure 50: New MCU structure.

Note 2: all the pins and channels infos were taken from the STM32F401 board's datasheet.[42]

.4 I2C interface module

Hitachi's HD44780-based character LCD is a low-cost and widely accessible component that is important for any project that displays information. The I2C bus may be used to show desired data on the LCD using the LCD piggy-back board. In general, such backpacks are designed around the NXP PCF8574, which is a general purpose bidirectional 8 bit I/O port expander that

uses the I2C protocol. The PCF8574 is a two-line bidirectional silicon CMOS circuit that enables general purpose remote I/O extension (an 8-bit quasi-bidirectional) for most microcontroller families (I2C-bus).



Figure 51: I2C protocol-based 8-bit I/O port expander.[17]

It should be noted that the majority of piggy-back modules are focused around the PCF8574T (SO16 package of PCF8574 in DIP16 package) with a default slave address of 0x27. If you have a PCF8574AT chip on your piggy-back board, the default slave address will be 0x3F.

In summary, if the piggy-back board is based on PCF8574T and the address connectors (A0-A1-A2) are not soldered together, the slave address 0x27 will be used.[17]

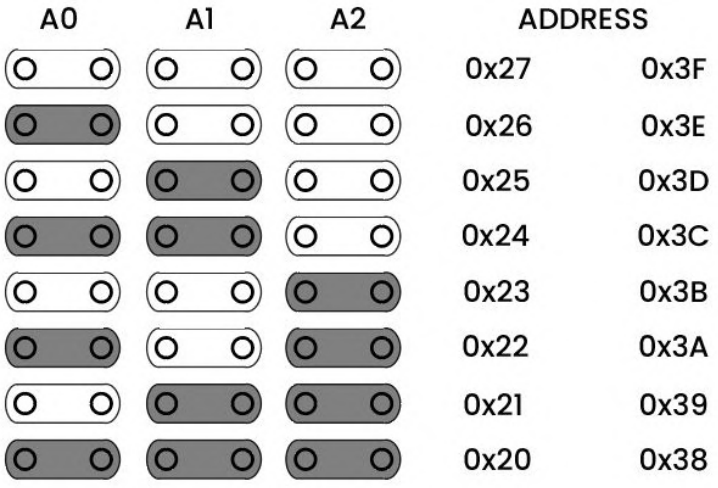


Figure 52: I2C interface module address.

In the figure below, you'll find some of the relevent Ldmicro I2C LCD commands :

Function	Address
Erase display	0x01
Cursor home	0x02
Blinking display	0x03
Normal diplay	0x04
Backlight OFF	0x09
Backlight ON	0x10

Figure 53: relevent Ldmicro I2C LCD commands.

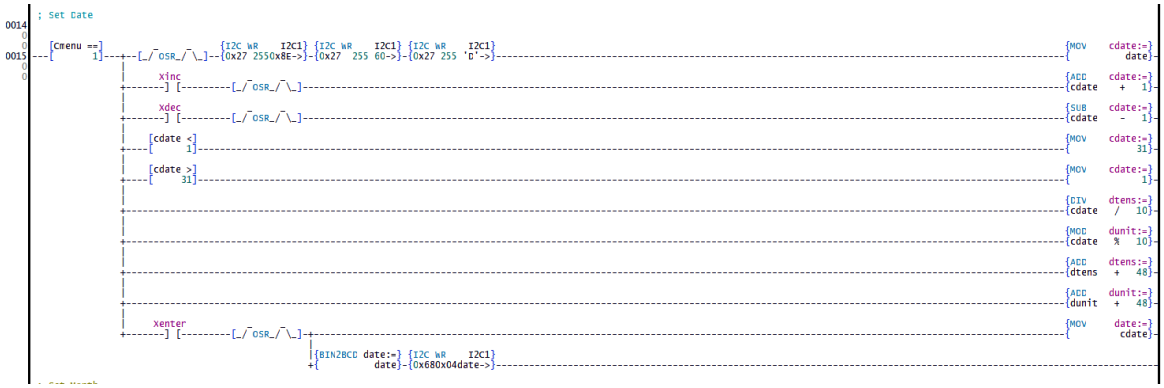
As well as the cursor commands for an 20x4 LCD screen in hexadecimal:

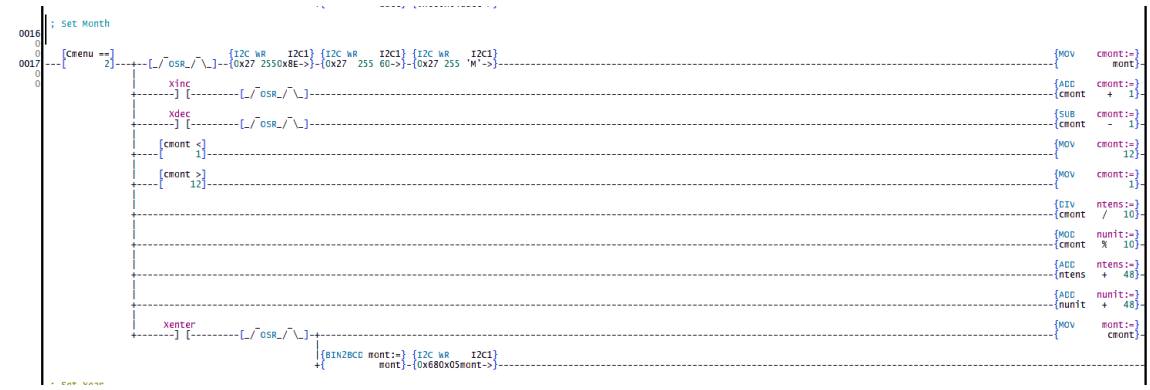
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	F0	F1	F2	F3

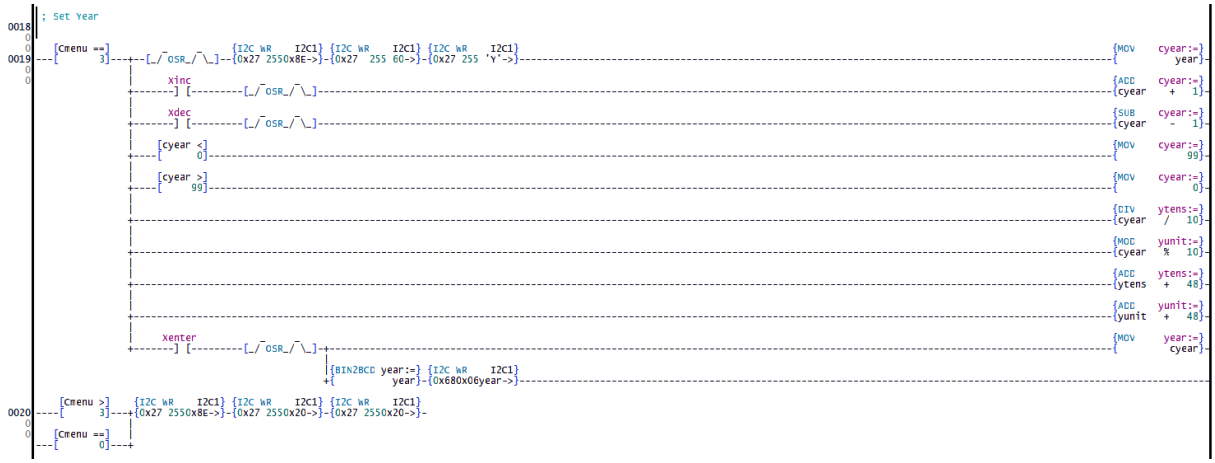
Figure 54: cursor commands for an 20x4 LCD screen in hexadecimal.

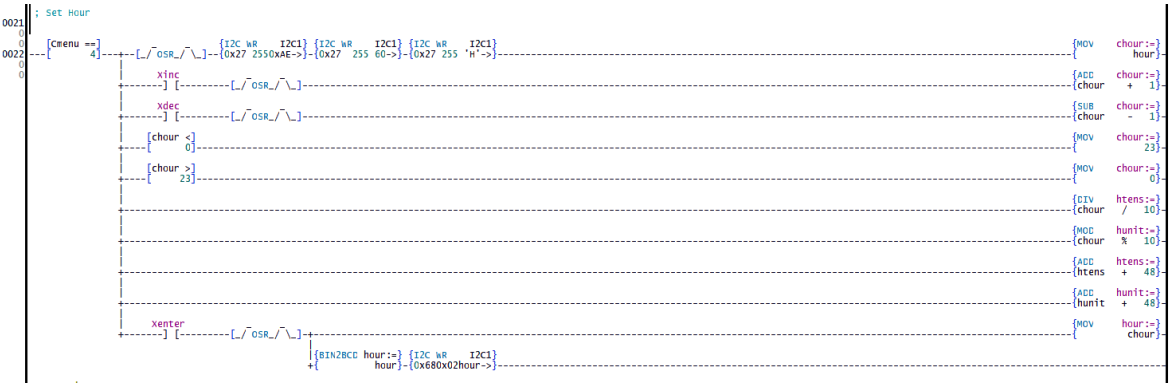
.5 RTC LADDER program (the following)

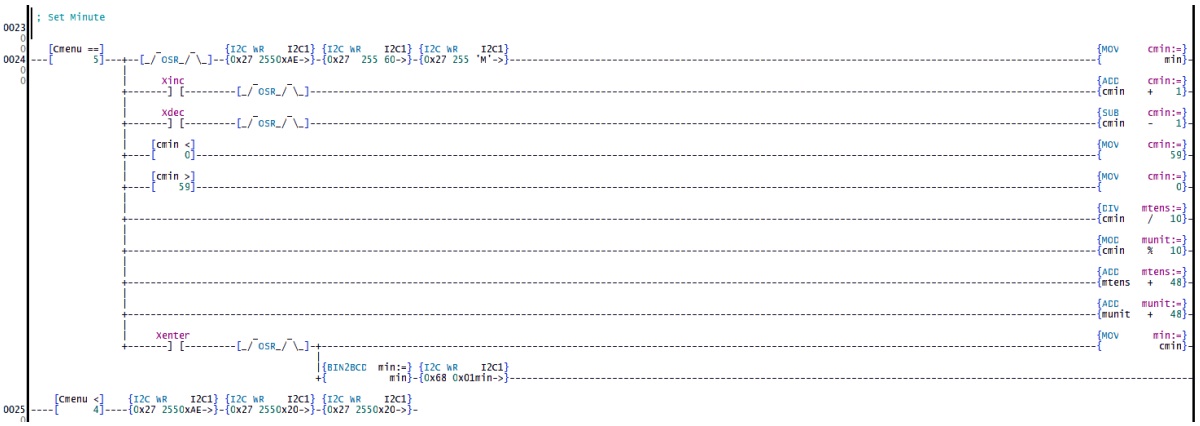



; Set Date
[Cmenu ==]
[1]
{I2C WR I2C1} {0x27 2550x8E->}
{I2C WR I2C1} {0x27 255 60->}
{I2C WR I2C1} {0x27 255 'D'->}
xinc
xdec
[cdate <]
[1]
[cdate >]
[31]
xenter
{MOV cdate:=} {date}
{ADD cdate:=} {cdate + 1}
{SUB cdate:=} {cdate - 1}
{MOV cdate:=} {31}
{MOV cdate:=} {1}
{DIV dtens:=} {cdate / 10}
{MOD dunit:=} {cdate % 10}
{ADD dtens:=} {dtens + 48}
{ADD dunit:=} {dunit + 48}
{MOV date:=} {cdate}
{BIN2BCD date:=} {I2C WR I2C1} {0x680x04date->}


; Set Month
[Cmenu ==]
[2]
{I2C WR I2C1} {0x27 2550x8E->}
{I2C WR I2C1} {0x27 255 60->}
{I2C WR I2C1} {0x27 255 'M'->}
xinc
xdec
[cmont <]
[1]
[cmont >]
[12]
xenter
{MOV cmont:=} {mont}
{ADD cmont:=} {cmont + 1}
{SUB cmont:=} {cmont - 1}
{MOV cmont:=} {12}
{MOV cmont:=} {1}
{DIV mtens:=} {cmont / 10}
{MOD munit:=} {cmont % 10}
{ADD mtens:=} {mtens + 48}
{ADD munit:=} {munit + 48}
{MOV mont:=} {cmont}
{BIN2BCD mont:=} {I2C WR I2C1} {0x680x05mont->}


; Set Year
[Cmenu ==]
[3]
{I2C WR I2C1} {0x27 2550x8E->}
{I2C WR I2C1} {0x27 255 60->}
{I2C WR I2C1} {0x27 255 'Y'->}
xinc
xdec
[cyear <]
[0]
[cyear >]
[99]
xenter
{MOV cyear:=} {year}
{ADD cyear:=} {cyear + 1}
{SUB cyear:=} {cyear - 1}
{MOV cyear:=} {99}
{MOV cyear:=} {0}
{DIV ytens:=} {cyear / 10}
{MOD yunit:=} {cyear % 10}
{ADD ytens:=} {ytens + 48}
{ADD yunit:=} {yunit + 48}
{MOV year:=} {cyear}
{BIN2BCD year:=} {I2C WR I2C1} {0x680x06year->}
[Cmenu >]
[3]
[Cmenu ==]
[0]
{I2C WR I2C1} {0x27 2550x8E->}
{I2C WR I2C1} {0x27 2550x20->}
{I2C WR I2C1} {0x27 2550x20->}


; Set Hour
[Cmenu ==]
[4]
{I2C WR I2C1} {0x27 2550xAE->}
{I2C WR I2C1} {0x27 255 60->}
{I2C WR I2C1} {0x27 255 'H'->}
xinc
xdec
[chour <]
[0]
[chour >]
[23]
xenter
{MOV chour:=} {hour}
{ADD chour:=} {chour + 1}
{SUB chour:=} {chour - 1}
{MOV chour:=} {23}
{MOV chour:=} {0}
{DIV htens:=} {chour / 10}
{MOD hunit:=} {chour % 10}
{ADD htens:=} {htens + 48}
{ADD hunit:=} {hunit + 48}
{MOV hour:=} {chour}
{BIN2BCD hour:=} {I2C WR I2C1} {0x680x02hour->}



Rung 5 acquires the minute value from register address one and stores the separated ASCII digits at “mtens” and “munit”.

Down to rung 13 that acquires the year value from register address six and stores the separated ASCII digits at “ytens” and “yunit”.

When the “Xmenu” contact sets “Cmenu” to one, rung 15 is enabled to set the date in the RTC module.

The I2C write instructions set the cursor at column 15, row 1 of the LCD to display a LESS Than symbol (<) and character “D” that indicates the setting of the date.

And then move the current value of the date to variable “cdate”. “cdate” variable is increased or decreased by contacts “Xinc” and “Xdec”. With the insurance that the selected date is within 1 and 31. The date is separated into two digits and converted into their ASCII values.

When “Xenter" is activated, “cdate” value is moved back to the “date” variable, converted from binary to binary-coded-decimal using the BIN2BCD instruction and sent to the RTC module register address of 04 to set the date.

Rungs 17 and 19 are similar to rung 15 but sets the month and year respectively.

Rung 20 clears the less than symbol and the menu option indicator from columns 15 & 16, row 1 of the LCD.

Rungs 22 and 24 are also similar to rung 15, but they set the hour and minute respectively.

Then, rung 25 clears the less than symbol and the menu indication from the LCD as before.

.6 Reading LM35 value

STM32F407VG6T microcontroller ADC supports 6-bit, 8-bit, 10-bit, and 12-bit configurable resolution. Furthermore, it supports three modes of A/D conversion such as single, continuous, scan, or discontinuous mode. The result of the ADC is stored in left or right-aligned 16-bit data register. The ADC resolution can be defined as the smallest input voltage at the analog pin that an ADC can identify and increments its digital value. The maximum and minimum digital output value of ADC depends on the number of bits of the ADC. For example, for a 8-bit ADC, the digital output value will be between 0-255, for a 10-bit ADC, the digital output value will be between 0-1023 and for a 12-bit ADC, the digital output value will be between 0-4095.

ADC resolution can be defined as:

Resolution = (Operating voltage of ADC) / 2 ^ (number of bits ADC)

We note that the LM35 puts out a voltage proportional (with no offset) to the temperature T in degrees Celsius. Since the MCU's A/D can't convert negative values, this means the circuit will work only for temperatures above freezing (should be translated with an offset if Temp <= 0 and may be amplified before because Vout = T/100 is low).

If we connect directly a 10 bits ADC (0 –> 1023) with 3.3V reference we can compute temperature (T >= 0 °C only) simply in LDmicro with:

As mentioned, the scale is 10 mV/degree. So Vout = 0.010*T

the operating voltage of STM32F4 series microcontroller is 3.3V

Then for a 10-bit A/D converter with a 3.3 V reference,

code = 1023*V/3.3= 1023*0.010*T/3.3 =1023*T/330

so T = 330*(code/1023)

the same for a 10-bit A/D converter with a 5 V reference,

code = 1023*V/5 1023*0.010*T/5 = 2.046*T

so T = code/2.046

If we want to evaluate this using only integer math, then we might write T = (code*22)/45. That doesn't overflow with 16-bit math, since 22*1023 = 22507 < 32768. We chose that approximation by truncating the continued fraction, so there's a sense in which it's optimal. This doesn't really matter, as long as it's much better than other sources of error (for example, the tolerance on the voltage reference, often around 1

ملخص

تحتاج صناعة الأتمتة إلى الابتكار في كل من البرامج والأجهزة. هناك ندرة في الأجهزة التي تتيح إنشاء منطق تحكم خالٍ من قيود منصات الاتصالات الخاصة بالمجال. الهدف من هذا المشروع هو إنشاء وحدة تحكم صناعية منخفضة التكلفة يمكنها تشغيل برامج مكتوبة بلغة البرمجة الصناعية. نتيجة لذلك، من المفترض تصميم وبناء والتحكم في وحدة تحكم صناعية قوية تعتمد على متحكم .STM32F407

أولاً، يتعلق الأمر بإجراء حالة هذا الفن لتحديد بنية عامة تلبي متطلبات السوق وتقديم دراسة حول حلول البرامج الحالية التي تسمح باختيار أفضل لتنفيذ منصة النماذج الأولية ، والتي سوف يتم إعدادها بعد الدراسة واختيار بنية ومكونات الأجهزة / البرامج.

كلمات مفتاحية:

نموذج أولي، وحدة تحكم منطقية قابلة للبرمجة، متحكم دقيق، حافلات صناعية، إنترنت الأشياء

Abstract

The automation industry need innovation in both software and hardware. There is a scarcity of devices that enable the construction of control logic that is free of the constraints of domain-specific communication platforms . The goal of this project is to create a low-cost Industrial Controller that can run programs written in industrial programming language. As a result, it is meant to design, construct, and control a strong Industrial Controller based on a stm32f407 microcontroller.

First, it is a matter of making a state of the art on existing technologies to define a generic architecture that meets the market requirements and to provide a study on existing software solutions allowing for a better selection for the implementation of the prototyping platform, which will be set up after study and choice of the hardware/software architecture and components.

Keywords :

Prototyping, PLC, Microcontroller, Industrial buses, IoT.

Résumé

Le secteur de l'automatisation a besoin d'innovation tant au niveau des logiciels que du matériel. Il y a une pénurie de dispositifs qui permettent la construction d'une logique de contrôle qui est libre des contraintes des plates-formes de communication spécifiques au domaine . L'objectif de ce projet est de créer un contrôleur industriel à faible coût qui peut exécuter des programmes écrits en langage de programmation industrielle. Par conséquent, il s'agit de concevoir, construire et contrôler un contrôleur industriel solide basé sur le microcontrôleur stm32f407.

Dans un premier temps, il s'agit de faire un état de l'art sur les technologies existantes afin de définir une architecture générique répondant aux exigences du marché et de fournir une étude sur les solutions logicielles existantes permettant une meilleure sélection pour la mise en place de la plateforme de prototypage, qui sera mise en place après étude et choix de l'architecture et des composants matériels/logiciels.

Mots clés :

Prototypage, PLC, Microcontrôleur, bus industriels, IoT.