Advancements in Embedded System:
My Internship Experience in Energy Department Embedded Systems

Ramy Bteich

INP-ENSEEIHT Toulouse

Two months Engineering Internship

July - August 2024

Contents

List of Figures

List of Tables

Introduction

I had the privilege of undertaking an enriching internship at the Energy Department of Indevco Phoenix, a leading company in the field of industrial and energy solutions in Lebanon. This internship provided me with hands-on experience and in-depth knowledge in various aspects of embedded systems and their applications.

During my tenure, I focused on several critical areas including embedded system design, Programmable Logic Controllers (PLCs), embedded software development, Internet of Things (IoT), Digital Signal Processing (DSP), and embedded networking. This comprehensive exposure allowed me to bridge theoretical concepts with practical applications, enhancing my technical skills and understanding of the energy sector's technological demands.
My responsibilities included conducting extensive research on embedded systems and automation, acquiring proficiency in PLCs programming languages, exploring advanced communication protocols, and adopting cutting-edge techniques. Additionally, I participated in practical testing and coding of PLCs. These activities not only reinforced my theoretical knowledge but also enabled me to apply this knowledge in practical scenarios effectively.

This report details the projects I worked on, the methodologies employed. It reflects the valuable learning experiences gained and the technical competencies developed during my internship at Indevco Phoenix.

I. Study Questions

Embedded Systems Design:

• **Microcontroller Selection:** A microcontroller is a miniature computer on a single chip, acting as the brain of embedded systems. It integrates essential components like: CPU, Memory (ROM, RAM,EEPROM), I/O Ports, Counters, ADC, DAC, Interfaces.And we have multiple types and designs of microcontrollers but the main two are:

1. *AVR Microcontrollers:* Developed by Atmel, it is widely used due to the simplicity, efficiency of the architectures.AVR microcontrollers have a exceptional architecture called "Harvard Architecture" which is separate memory spaces for program instructions and data, allowing simultaneous access to both. And AVR use RISC (Reduced Instruction Set Computing) for simplified instruction set for fast execution and efficiency.AVR can process up to 8 bits of data at a time due to his 8-bit CPU and at a high speed because he execute most of the instructions in a single clock cycle.And it is composed of 3 types of memory, flash memory which is non-volatile memory for sorting the code, SRAM a volatile memory for temporary data storage during the program execution, EEPROM also a non-volatile memory for sorting the important data that is used even when the system is powered off. And some popular models are:

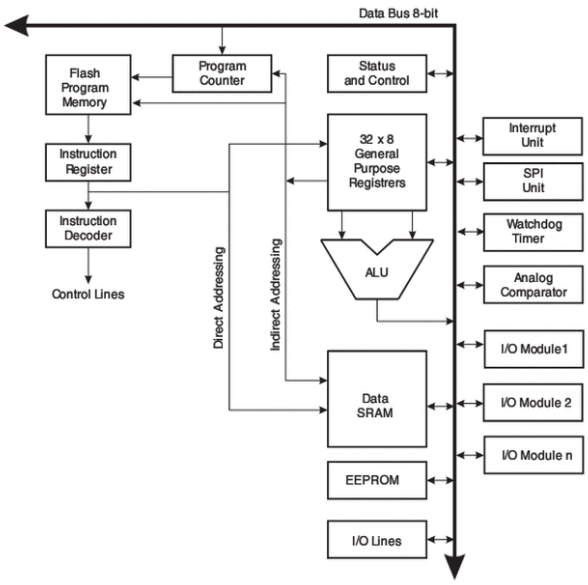

Figure 1: *AVR Architecture*

- ATmega328P, used in Arduino Uno boards.
- ATtiny85, used in small, low-power applications.
- ATmega2560, used in more advanced applications requiring more I/O pins and memory.

2. *ARMCortexMicrocontrollers* : ARM Cortex microcontrollers are based on the ARM architecture and developed by RAM Holdings. They are used due to their high performance, low power consumption.So we can say that they have the same architecture as ARM Harvard architecture and RISC. And one of the big difference is that ARM Cortex can process 32 or 64 bits because they come in two version a 32-bit CPU or a 64-bit CPU, and they can perform better and faster than before because they are capable to operate at higher clock speeds.They have the same memory types as AVR, but here we have 3 series of Cortex, Cotex -M series (M0/M0+, M3, M4, M7)

designed for low power consumption and high efficiency, Cortex-R series (R4, R5, R7) designed for real-times application and offer a high performance and reliability and finally we have Cortex-A (A7, A9, A53, A57) series designed for high performance application like smart-phones...

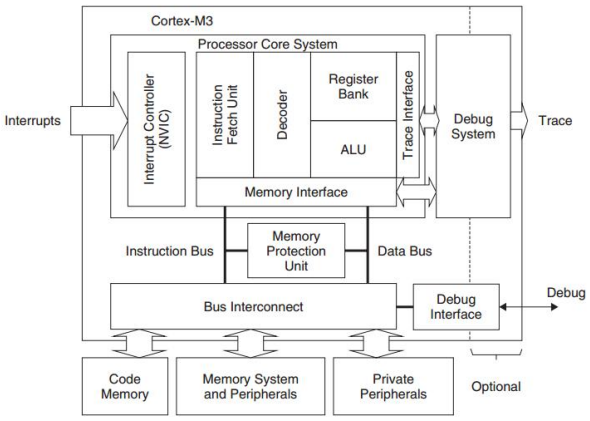

Figure 2: *ARM Cortex Architecture*

And some popular models are:
- STM32F103, for general applications.
- STM32F407, sed in more advanced applications requiring higher performance.
- NXP LPC1768, used in mbed development boards.
- SAM3X8E, used in the Arduino board.

• **RTOS (RealTime Operation Systems):** RTOS is a operating system designed to serve real-time operation that process data as it comes in without a buffering delays in another hand the typical operating systems like Windows, MacOS, etc..., are designed to handle operation within deadlines. An RTOS provides a predictable and reliable behavior because the task will be executed in a predefined time frame, and they provide mechanisms for creating, scheduling, and synchronizing tasks so they can prevent conflicts and ensure a smooth operation, they employ also a specialized scheduling algorithms to prioritize and manage tasks based on their importance and they are well known for their lightweight, designed to minimize resource usage and maximize performance.

In embedded systems when multiple processes need to run simultaneously(reading sensor, communication with devices...), RTOS can handle them by enabling the multitasking on a single processor. RTOS are very known in real-time operation due to the prioritization of tasks based on their importance and deadlines, so the urgent one are handled first and they can manage systems tasks without starving the others which is important for most of the embedded system especially with ones with limited resources. RTOS are used everywhere like in robotics, machinery, in cars (breaking systems, managing engine controls), even in the army for controlling aircraft systems, managing satellites, for medical purpose for example in pumps, monitoring patient vitals, and even we can find them in our own hand like operating smart home devices, and managing wearable devices and many others uses.

• **Low-Power Design:** Power consumption is a real problem in embedded systems, especially on operating in batteries or energy harvesting. So we need power management to extend the battery life and reduce heat dissipation to enhance the overall experience.So we have several techniques to solve this problems, so lets break them in two parts, hardware and software.

Hardware Techniques: Clock gating, by disabling clock signals to inactive modules to reduce dynamic power consumption.Power gating by turning off power to unused blocks to save leaked power.DVFS, by adjusting the frequency and voltage level based on the required load. Low-Power modes, by deep sleep modes when the system is idle.Low-Power Components selecting components designed for low power operation.

Software Techniques: Task scheduling, by optimizing task scheduling to minimize the time spent in active modes.Efficient algorithms, by designing algorithms that minimize computations and memory accesses.Code optimization, by optimizing code for size and efficiency can lead to reduced execution time.Power-Aware libraries,using libraries and frameworks designed for power optimization.Tickless Kernel: In RTOS, using a tickless kernel can save power by avoiding unnecessary wake-ups and context switches.

And embedded systems relies on methodological approach : Power profiling, to power consumption at various operational states to pinpoint components requiring optimization.Power modeling and simulation: Creating models and simulating system behavior to assess the impact of design decisions on power usage.Power Budgeting, to assign power consumption limits to individual components to maintain overall system efficiency. Co-Design, by integrating hardware and software considerations throughout the design process for power optimization.

Programmable Logic Controllers (PLCs)

A Programmable Logic Controller is a specialized industrial computer designed to control and automate machinery and processes.They are built to handle harsh environments such as temperature, vibration, and all sort of noises, and they can process inputs and outputs in real time to allow a quick responses and a precises outcome. So how a PLC work first we have the input which the PLC will receive from a signals sensors, after that the signals will be processed based on on the programmed logic, making decisions about the actions that should be taken at the time, and finally the PLC sends signals back to actuators (Motors, valves, relay ...).

Figure 3: *PLC*

• **PLC Programming Languages:** To program a PLC to execute tasks we should write and load a code on it, and it have multiple languages, Ladder Diagram, Structured Text, Function Block Diagram, Instruction List, Sequential Function Chart. But in this report we are going to talk about three of them.

	Ladder Diagram (LD)	Structured Text (ST)	Function Block Diagram (FBD)
Representation	Graphical: Network of contacts (inputs) and coils (outputs) connected by lines (power flow)	Text-based: Similar to high-level programming languages (e.g., Pascal, C) with statements, variables, and operators.	Graphical: Interconnected blocks representing functions, data flow through lines.
Programming Paradigm	Relay logic: Mimics electrical circuits with normally open/closed contacts, timers, counters.	Procedural/imperative: Execute statements sequentially, use loops and conditional statements.	Data flow: Flow of data through function blocks dictates execution.
Ideal For	Discrete (on/off) control, simple logic, sequence control, interlocking, PID control	Complex calculations, algorithms, data processing, communication protocols, string manipulation, tasks requiring flexibility.	Mix of control and calculations, complex systems, modular and reusable code, simulation, process control applications.
Strengths	Intuitive for those with electrica l background, easy to visualize and troubleshoot, good for quick development.	Powerful and expressive, supports various data types, concise and organized code, suitable for large projects.	Combines visual aspects of LD with high-level functionality of ST, supports hierarchical design, reusable code blocks.
Weaknesses	Cumbersome for complex logic, difficult to reuse code, less efficient for large programs, limited data types.	Steep learning curve, less intuitive, requires programming knowledge, challenging to debug for non-programmers.	Can become cluttered for large programs, may require more planning compared to LD, not as intuitive for simple tasks.
Best Suited For	Simple on/off control, relay replacement, beginners, maintenance personnel, applications with limited complexity.	Experienced programmers, complex algorithms, data-intensive tasks, projects requiring flexibility and code reuse.	Mix of experience levels, complex control systems, projects requiring modularity, data visualization, process engineers.
Examples of Usage	Motor start/stop control, conveyor line logic, safety interlocks, simple machine automation, basic process control.	Mathematical calculations, data logging, PID control algorithms, communication interfaces, complex logic sequencing.	Analog signal processing, PID loops, motor control systems, complex automation sequences, custom function development.

Table 1: *Comparison of different PLC languages*

• **PLC Communication Protocols:** Like we said before, PLCs are the brain of the automation systems, so they need to communicate with various devices or other PLCs to control and monitor the processes. And PLCs have multiple communication protocols and here below we will talk about them.

1. Modbus, it's the oldest version and the most used one, due to his simplicity of implementation.It is available in 2 versions , serial (Modbus RTU/ASCII) and Ethernet (Modbus TCP/IP). and its used for building automation and energy management.

2. Profibus, it's a newer version and more advanced than Modbus, which offer a higher speed and more complex functionalities, and can support various network protocols (bus,star,ring...). and it is widely used in chemical processing and complex machinery control.

Ethernet/IP, Profinet, DeviceNet, CC-Link, these are the rest of protocols we can find for PLCs. To choose the right one, we need to take in consideration the speed, the compatibility with other devices and networks, and we need to think about the future expansion and changes if it's going to be made.

• **PLC Security:** Because the PLC is the brain of all operation so it's become susceptible to cyber attacks and threats.it's easy to be hacked due PLCs outdated built-in security systems, connection to all sort of network even the internet which open doors to external attacks.If the hacker , he will be able to gain control of PLCs to disrupt operation, steal data, or even fry up the system, even if the hackers main objective isn't hacking the PLCs system he could reach all the devices in the company due the network connection of PLCs with all the other devices.To protect the system from all of that we can develop a network segmentation or the usage of VPN's to isolate PLC networks from other networks, monitoring network traffic and detecting suspicious activity by using firewalls and IDS.

Digital Systems and Data Representation

All digital electronics rely on the binary systems form, typically represented by "0" (off) and "1" (on).

• **Bits and Bytes:** What is a Bit and Byte ? A Bit is a single digit "0" or "1", and it is the smallest unit of information in the system. A Byte is a group of 8 bits, It's a fundamental unit for representing characters, numbers, and other data.

We have several number systems used in digital electronics:

1. Decimal:
   -Base-10 system (0 to 9).
   -The numbering system that we use in our daily basis.

2. Binary:
   -Base-2 system (uses only 0 and 1).
   -Each digit's position represents a power of 2.

3. Hexadecimal:
   -Base-16 system (0 to 9 ,10=A to 15=F).
   -Convenient for representing large binary numbers concisely.

4. Octal:
   -Base-8 system (0 to 7).

Conversion Between Number Systems:
-Converting binary 1101 to decimal: $1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 8 + 4 + 0 + 1 = 13$.
-Converting hexadecimal A3 to binary: A = 1010; 3 = 0011; Therefore, A3 = 10100011

• **Data Types in Programming:** In programming the kind of values a variable can hold are defined by the data types.They also determine how the data is stored and represented in the computer memory.Here are some of the most common data types:

Data Type	Memory Representation	Advantages	Disadvantages	Use Cases
Integer (int)	Fixed number of bits (e.g., 16 bits: 00000000 00001010 for decimal 10)	Exact representation of whole numbers, efficient arithmetic operations	Limited range, cannot represent fractions	Counting, indexing, calculations with whole numbers
Floating-Point (float)	Sign bit, exponent, mantissa (e.g., 0 10000010 10010010000111111011011 for 3.14159)	Wide range, can represent fractions	Approximate representation, potential for rounding errors	Scientific calculations, financial calculations, graphics
Character (char)	ASCII/Unicode code (e.g., 01001000 for 'H')	Compact representation of individual characters	Limited to representing single characters	Text manipulation, input/output operations
String (string)	Array of characters (e.g., "Hello" -> 'H', 'e', 'l', 'l', 'o', '\0')	Can represent sequences of characters	Can be inefficient for large amounts of text	Storing names, messages, user input
Boolean (bool)	Single bit (0 for false, 1 for true)	Simple representation of true/false values, used in logical operations	Limited to two possible values	Conditional statements, logical comparisons

Table 2: *Comparison of different Data Types*

• **Fixed-Point vs. Floating-Point Arithmetic:**

1. $Fixed - Point$: Represent real numbers using integers scaled by a fixed factor. The advantage of this method are :
   -Operations could be performed directly in hardware without needing a dedicated floating-point unit (FPU).
   -Fast calculation due to their simpler representation.
   -Offer predictable and consistent results, a real deal for real-time systems where timing is critical.
   And some disadvantages are:
   -Overflow and underflow due to the range of re presentable numbers is limited by the number of bits used.
   -Lower precision for very small or very large values.
   -Determining the appropriate scaling factor and managing scaling operations can be challenging.

2. $Floating - Point$: Represent real numbers using a sign bit, an exponent, and a mantissa, allowing for a wide range and varying precision.The advantage of this method are :
   -Can represent a vast range of values, from very small to very large, without requiring explicit scaling.
   -Offer higher precision than fixed-point numbers, especially for values that vary widely in magnitude.
   And some disadvantages are:
   -Requirment of a dedicated FPU, which can increase chip size and cost.
   -Calculations are generally slower than fixed-point calculations due to their complex representation and operations.
   -Slightly different results on different platforms due to rounding errors.

Fixed-point arithmetic is generally faster and can be implemented on simpler, cheaper hardware, but it offers lower precision and requires careful management of scaling to avoid overflow and underflow issues. In contrast, floating-point arithmetic provides higher precision and can handle a broader range of values, though it tends to be slower, especially on systems lacking a dedicated floating-point unit (FPU), which is often necessary for efficient floating-point operations.

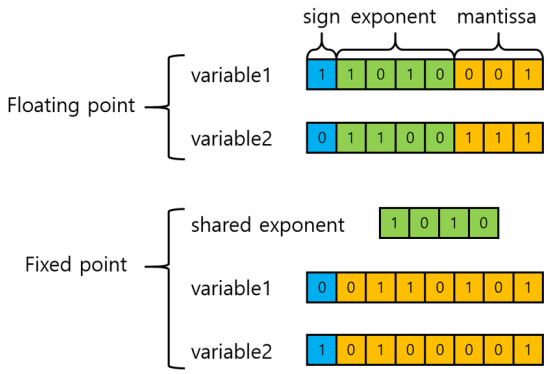

Figure 4: *Fixed-Point vs Floating-Point*

Embedded Software Development

C and C++, are used for embedded system because they offer benefits like object-oriented programming for structuring complex systems, templates for generic programming, and RAII for automatic resource management.
• **Embedded C/C++ Programming:**C/C++ are widely used in embedded coding due to:

1. Memory management
   (a) Minimize footprint by and customizing the allocated arrays and structures when possibles and using dynamic memory.
   (b) Fragmentation avoidance, by using Pool allocators, which used when we have object with similar sizes.
   (c) Leak prevention, RAII(Resource Acquisition Is Initialization), c++ idioms like smart pointers help manage resource lifetimes automatically.
2. Resource Efficiency
   (a) Optimized Algorithms and bit-wise operations, choosing the right algorithms with low space and time complexity and using bit manipulation if needed (e.g clearing flags in registers)
   (b) Power awareness, understanding the power modes techniques (sleep mode, low-power) offered by the micro controller.
3. Performance Optimizing and Robustness and Reliabilty
   (a) Profiling, by using tools to identify bottlenecks.
   (b) Compiler Optimization and Inline Functions, enabling compiler flags for size and speed optimization, considering inlining to reduce function call overhead, for small and frequently called functions.
   (c) Error Handling, by assertions liberally during development to catch the unexpected operations by choosing a clear mechanism. And implement a watchdog mechanism to recover from software freezes.

• **Devices Drivers:** We have multiple structures and architectures used in embedded device driver development and here below we will talk about them.

1. Monolithic Drivers: All driver functionality is contained within a single code file or module, which is small for small uses but it could become unwieldy and difficult to maintain for complex peripherals.
2. Layered Drivers: The driver is devided into layers:

(a) Hardware Abstraction Layer (HAL): Direct interaction with hardware register and interrupts

(b) Device Specific Layer: implements the core functionality of the devices, like sensor reading, motor control ...

3. Object-Oriented Drivers: the driver is implemented as a class or set of classes, that have data and operations related to the devices.

• **Firmware Development:**Firmware is the software that resides permanently within a micro controller or embedded system it provides the low-level instructions for how the device operates,and tasks execution.To develop a firmware we need to define the purpose and functionality of the embedded system,identify the target id it is a micro-controller or peripherals, gather some information about specifications, power consumption, and interfaces.After that we need to choose the compatible version or architecture for example bare metal, RTOS-based, design the software modules, data .And finally we need to program this system by choosing the language c or c++ or even assembly, after that we need to implements this coded software to the according design, and then we test the software and the hardware together, and if the system was slow or not like we expected we can optimize the code for less calculation and memory usage.

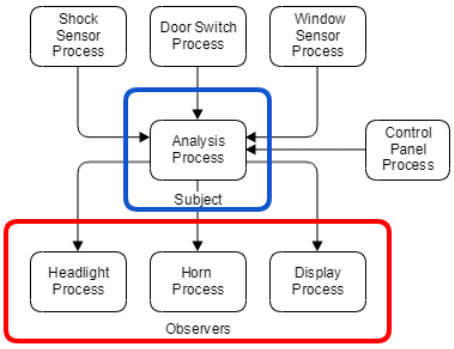

Figure 5: *Firmware Design Patterns in Embedded Systems*

Internet of Things (IoT) and Embedded Networking

• **IoT Protocols:**IoT protocols are the set of rules that control and manage how devices within IoT network interact with each other and with the internet, like how devices can find and recognize each other, the exchange of information between them and between the servers, and of course for protection from the outside or unauthorized access.

1. *Message Queuing Telemetry Transport-MQTT*, is a lightweight and efficient publish-subscribe protocol.First, we have a central server acts as the broker, managing communication,then devices like sensors that send data to the broker,after that, devices like applications receive data from the broker.And finally we should channels the communication that devices subscribe to or publish on.So we can say that MQTT is ideal for resources-constrained devices,it guarantees message delivery or at least attempts it, a flexible protocol because the publishers and the subscribers are not directly connected, and obviously it can handla a large number of connected devoces.

2. *Constrained Application Protocol-CoAP*, is a web transfer protocol designed for constrained devices and networks, its simpler and more compact than HTTP, making it suitable for environments with limited resources.CoAP is used because he uses the User Datagram Protocol(UDP), which has less overhead that TCP.And CoAP support requests and responses without maintaining a constant connection, and have a low power consumption which make it very suitable for battery-powered devices.

The best protocol for your IoT application depends on what you need. MQTT is great for situations where reliable message delivery and scalability are important, like in industrial automation and smart homes. CoAP works well for simpler uses where low power and efficient data transfer are important, like in sensor networks and wearable devices. Sometimes, combining protocols is beneficial. For example, you might use CoAP for communication between devices in a local network and MQTT to send data to a cloud server.

And they are other protocols than this last two.

1. *Advanced Message Queuing Protocol-AMQP*, a versatile messaging protocol used in various industries.

2. *Extensible Messaging and Presence Protocol-XMPP*, a protocol originally designed for instant messaging, now adapted for IoT.

3. *Zigbee*, a low-power, mesh networking protocol often used for smart home devices.

• **Wireless Communication:** WiFi typically operates in the 2.4 GHz and 5 GHz bands, whereas Bluetooth operates in the 2.4 GHz band. WiFi generally has a longer range (up to 100 meters) compared to Bluetooth (up to 10 meters). WiFi supports higher data rates (up to several Gbps with the latest standards) compared to Bluetooth (up to 3 Mbps with Bluetooth 4.0). Bluetooth is designed for low power consumption, making it suitable for battery-powered devices, while WiFi consumes more power. WiFi is commonly used for internet access and high-bandwidth applications, while Bluetooth is used for short-range communication and device pairing. Bluetooth supports more precise localization and indoor positioning systems due to its short range and the ability to measure signal strength with finer granularity. Bluetooth Low Energy (BLE) beacons are widely used for indoor positioning applications. WiFi localization typically relies on triangulation or trilateration using multiple access points, which can be less accurate due to the larger range and multipath interference. Bluetooth also uses frequency hopping, which offers several advantages compared to traditional WiFi. It enhances resistance to interference and eavesdropping by frequently changing frequencies, making it difficult for unauthorized devices to intercept communications. This approach also improves coexistence with other wireless technologies in the same frequency band, reducing the likelihood of signal clashes and improving overall communication reliability.
Wearable devices like fitness trackers and smartwatches often use Bluetooth for transmitting data, as do wireless peripherals such as keyboards and mice. On the other hand, Wi-Fi is commonly used in smart home devices like thermostats and cameras, as well as in industrial automation for remotely monitoring and controlling equipment. The choice

between Wi-Fi and Bluetooth depends on your application's requirements: use Wi-Fi for high data rates, long range, and internet connectivity, and opt for Bluetooth for low power consumption, easy pairing, and short-range communication.

• **Edge Computing:** Edge computing is a distributed computing paradigm that brings computation and data storage closer to the location where it's needed.This approach is very useful for application that require real-time processing and low latency, such as IoT, autonomous vehicles, and industrial automation. By reducing the distance data needs to travel, edge computing enhances performance and reliability.

Edge computing is used in IoT and embedded applications due to its ability to reduce latency by processing data close to its source, which is essential for real-time tasks like industrial automation and autonomous vehicles. It optimizes bandwidth by only sending relevant or summarized data to the cloud, which is beneficial in scenarios with limited bandwidth or high data transfer costs. By handling sensitive data locally, edge computing enhances security and privacy while supporting compliance with data localization regulations. It also ensures reliability and resilience by functioning even with intermittent cloud connectivity and can lead to cost savings by minimizing cloud storage and processing expenses through data filtering and aggregation.Implementing edge computing in IoT and embedded applications involves several components. *Hardware* includes edge devices ranging from simple microcontrollers to powerful single-board computers (SBCs) and specialized edge gateways, along with sensors and actuators that collect and act on data at the edge.*Software* elements involve edge runtime environments that run applications on edge devices, often using containerization technologies like Docker, along with edge analytics and machine learning for real-time data processing and decision-making, and edge orchestration for managing and deploying applications across devices. *Networking* components include local area networks (LANs) for connecting edge devices within a facility, wide area networks (WANs) for linking edge devices to the cloud or remote locations, and 5G or other cellular networks for providing high-speed, low-latency connectivity in mobile edge applications.

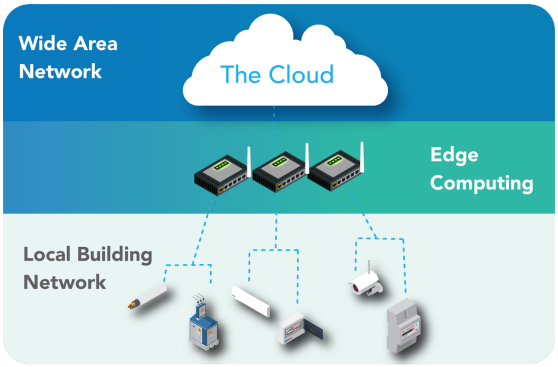

Figure 6: *What is an IoT Edge Device*

Signal Processing and Control Systems

DSP plays a role in enabling many of the technologies we use daily.It helps us to extract meaningful information from raw signals,It helps us to extract meaningful,improve the quality of the audio and video, and a smooth and efficient transmission.

• **Digital Signal Processing (DSP):** Their are multiple techniques to process digital signal, and these techniques are:

1. *Filtering*, separating signals based on frequency, removing noise, and we use Finite

Impulse Response (FIR) filters, Infinite Impulse Response (IIR) filters and Kalman filters (for estimation and tracking) Algorithms

2. *Transforms*, analysing signals in frequency,time domains for manipulations, Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT) and Wavelet Transform are the used algorithms

3. *Spectral analysis*, used to determine the frequency content of a signal, and the algorithms used are Periodogram, Welch's method and MUSIC (for direction of arrival estimation)

4. *Sampling*, to convert continuous signals to discrete representations and assigning them numerical values, and to do this we can use Nyquist-Shannon sampling theorem and Analog-to-Digital Converters (ADCs)

5. *Adaptive Filtering*, the purpose of it, is to adjust filter parameters in real-time based on changing signal characteristics, and some of the algorithms used are Least Mean Squares(LMS) and Recursive Least Squares(RLS).

6. *Modulation and Demodulation*, the modulation is to prepare the signals for transformation, and the demodulation is to recover information from received signals.To do that we should use Amplitude Modulation(AM), Frequency Modulation(FM), Phase Modulation(PM), and Quadrature Amplitude Modulation(QAM)

When working with DSP in embedded systems, key libraries and tools enhance performance and efficiency. *ARM CMSIS-DSP*, is an optimized DSP library for ARM Cortex-M processors. *MathWorks Embedded Coder*, who generates optimized C/C++ code from MATLAB/Simulink models. *TI C2000 DSP Library*, is a library for Texas Instruments DSPs.

• **Control Systems:**The Proportional-Integral-Derivative (PID) controller is a widely used control algorithm due to its simplicity, effectiveness, and versatility. It calculates control signals based on three components: Proportional (P) responds to the current error, Integral (I) considers accumulated error over time to eliminate steady-state errors, and Derivative (D) reacts to the rate of error change to prevent overshoot. To design PID controllers for embedded systems, one must model the system, tune the PID gains, implement the algorithm considering constraints like sampling rate and computational resources, and thoroughly test and validate the performance. Real-time constraints, computational limitations, sensor noise, and actuator limitations are common challenges. Advanced control techniques like Model Predictive Control (MPC), Adaptive Control, and Fuzzy Logic Control can be used for more complex systems.

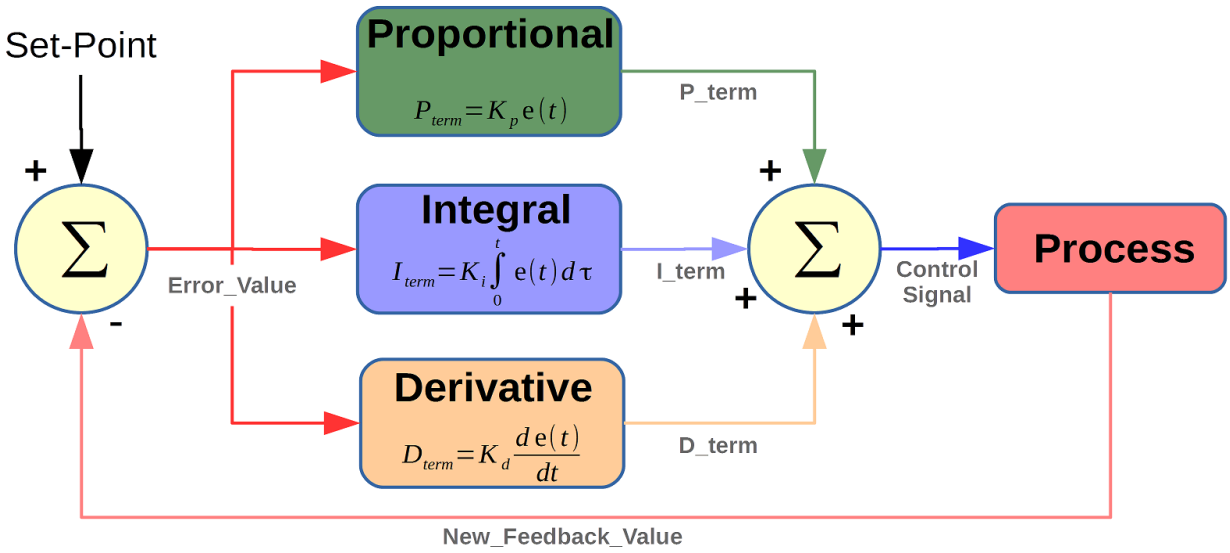

Figure 7: *Proportional-Integral-Derivative (PID)*

Emerging Technologies in Embedded Systems

• **AI in Embedded Systems:**AI is used everywhere nowadays, even in embedded systems. We can use AI for real time processing, decisions could be made locally on the device which can eliminate the latency of sending data to the cloud for processing, and the main data can be processed on the device to minimize the risk of exposure during transmission.Reducing the bandwidth, by choosing some of the main information to be transmitted which can conserve the bandwidth and reduce the cost, and AI is optimized for low-power devices and can extend the battery life which is a main thing in embedded systems.Here below are some examples:

1. *Autonomous Systems:*AI algorithms can process sensor data in real-time to make decisions about navigation, obstacle avoidance, path planning, self control, in self-driving car, drone,robots, etc.

2. *Wearable Technologies:*Most of the devices now have AI in them, for example in smartwatch we can use AI algorithms for analyzing heart rate, movement patterns, and even AI can propose some workouts, nutrition advice, or stress management based on the individual database.

3. *Smart Sensors and IoT Devices:*Embedded AI enables sensors to not only to fetch and collect data but also to process it locally, and making real-time processing to reduce the sending of the vast raw data to the cloud to be processed.For example it can be used to predict maintenance, the algorithm can analyze sensor data from the machines to predict failure before they occur to minimize the time.

• **Blockchain for IoT:**Blockchain technology provides secure and efficient solutions for managing data in IoT and embedded systems.Which can ensures that data cannot be

tampered with and remains transparent, making it ideal for handling sensitive information like sensor data, medical records, and supply chain details. Blockchain can trace the origin and history of data, ensuring it is genuine and reliable, which is for verifying products and preventing counterfeiting. Smart contracts on the blockchain automate tasks such as payments and access control, reducing the need for middlemen.

Blockchain enhances security and privacy by removing central control points, reducing the risk of attacks and data breaches, and using cryptographic techniques to protect data and transactions. It also provides secure digital identities for IoT devices, ensuring safe interactions and data exchanges. In supply chain management, blockchain ensures transparency and traceability by tracking goods in real-time and preventing fraud. It helps in energy management by facilitating peer-to-peer energy trading and managing microgrid transactions, ensuring fair compensation for all parties involved.

However, blockchain faces challenges like scalability and high energy consumption, especially in large IoT networks. Solutions such as sharding and layer-two protocols are being developed to address scalability, and more energy-efficient consensus algorithms like Proof of Stake (PoS) are being researched to reduce energy use. The lack of standard protocols can hinder interoperability, requiring industry collaboration to create common standards. Regulatory and legal issues also need to be addressed to cover data privacy and smart contract enforceability. Despite these challenges, blockchain holds great promise for improving embedded systems and IoT as research continues to overcome these obstacles.

II. PLC Exercises

First Exercise

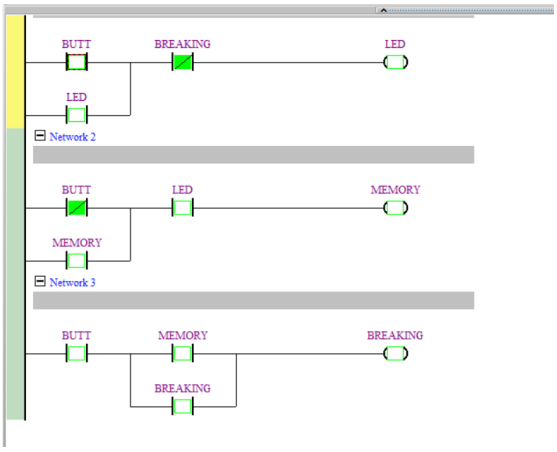

Figure 8: *Button is Off for the first time*

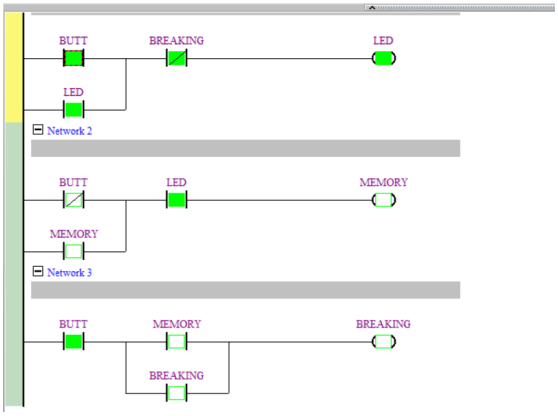

Figure 9: *Button is On for the first time*

Network 1

This network is used to create a holding circuit, when the push button is pressed, the output turn on and will stay on until the push button is pressed again

Network 2

This network is used to store the status of the output ftom the previous network.We have two input and a single output, the inputs are, an NC (Normally closed) contact and the output of the network 1.When the push button in network 1 is pressed the output will turn on and the NC contact will open, which allow the power to flow to the memory and turn it on, and this memory will stay on even after the push button is released.

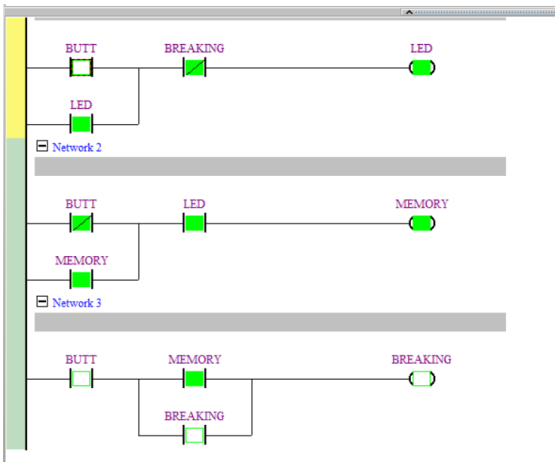

Figure 10: *Button is Off for the second time*

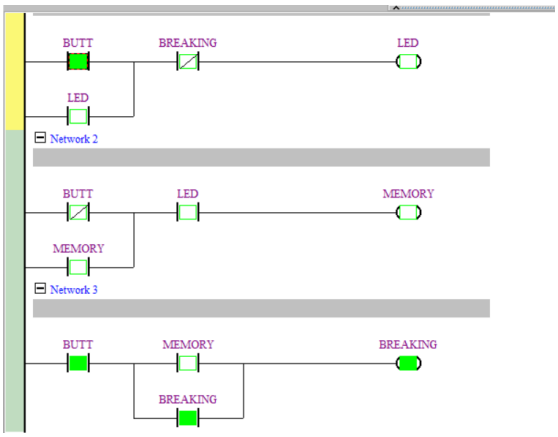

Figure 11: *Button is On for the second time*

Network 3

The objective of the network is to turn off the output from network 1 when the button is pressed again.We can say that it has three input, from the memory network 2, the push button and the third is an NC contact.And the output of this is the breaking circuit.So when the push button is presses and the memory bit is on, the NC contact will close and allow power to flow to the breaking circuit, which is going to turn off the output from network 1.

By combining these networks, the circuit can turn the output on and off using a single push button. Pressing the button once turns the output on, and pressing it again turns the output off. The memory bit helps to maintain the state of the output even after the push button is released.

Second Exercise

Network 1

In the first network a button is used to start the simulation, and we used a timer called TMR, which is set to his max limit 12 seconds.

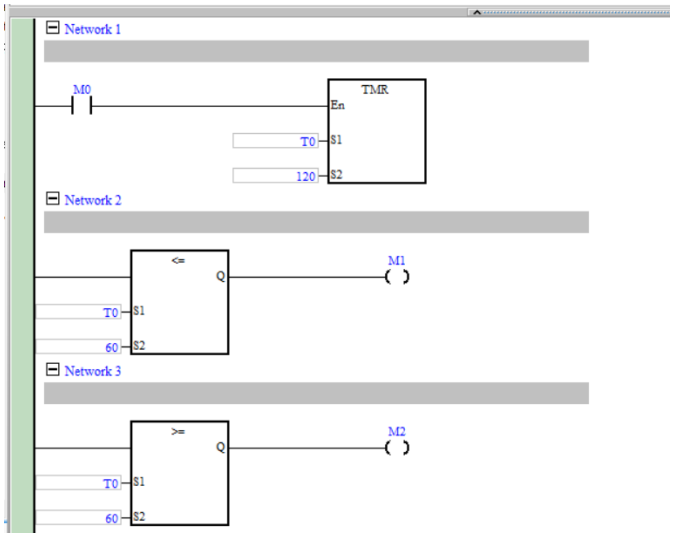

Figure 12: *Timer and Network 1 and 2*

Network 2

In network 2 we used a less than or equal to compare the time for the first execution and with the max time 12 seconds, so we can set the new time to 6 seconds, so when the initial value reach to 6 seconds this network will terminate and jump to the next one.And we will have an output named M1.

Network 3

In this network it has the same logic as the one before but here we used greater than or equal, so when timer will continue to count after the second it will keep execute the second one because the value will keep increasing greater than 6 seconds until meeting a new condition.And the output M2 will be given when this network will be terminated.

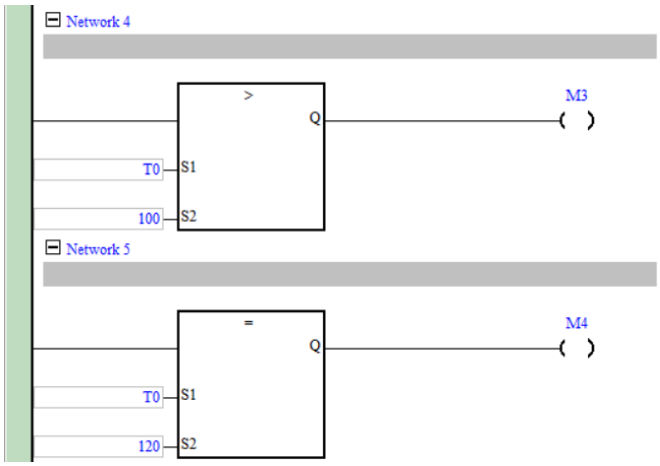

Figure 13: *Network 4 and 5*

Network 4

In this network, the timer T0 need to be compared with 10 seconds, so when its reach the 10 seconds and be greater than 10 the previous network will end and the new network will start working.and we will have the last output M3 which represent the third light.

Network 5

This network job is to give us the M4 output when the timer reach 12 seconds which is the max limit for the system.Because the first light is 6 seconds, the second one is 4 seconds, and the last one is for 2 seconds.And we will get the M4 output which is used for the reset operation later.

Network 6

This network is when the light will be turning on, So we have all the previous output except M4, so we have M1 open and M2 and M3 closed.Because Y1 is the light of M1 in the second network.

Network 7

This network is used to turn on the light Y2, which is the light of the network 3, and in this network we have the output M2 open and the output M3 and M1 closed.

Network 8

This network is used to turn on the third light Y3, which the output of the network 4 we have M3 open and M1 closed.

Network 9

This is the last network, and her job is to reset the timer to zero when the desired time is reached to the output M4 of the network 5.

At 0.7 seconds, only the first light turned on and the others still off.At 8.2 seconds the second light should be on and the first turned off and the third will remain off, so the results is approved in Figure 15.And finally in Fig 16, the third light is turned on and we can see that the second is turned off and the first is still off. And due to the last network this will always repeat continuously.

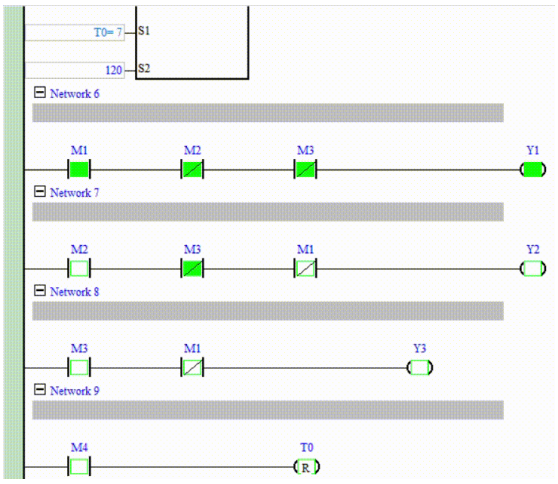

Figure 14: *Time = 0.7 seconds*

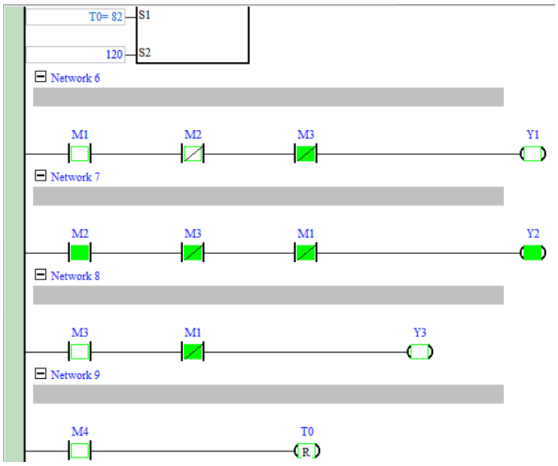

Figure 15: *Time = 8.2 seconds*

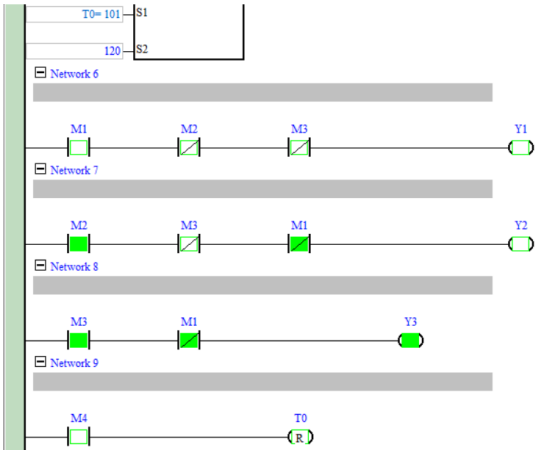

Figure 16: *Time = 10.1 seconds*

Conclusion

My internship at the Energy Department of Indevco Phoenix has been an invaluable experience, providing me with a robust platform to apply theoretical knowledge in the dynamic world of embedded systems and industrial automation.The best part was learning and working with PLC's, It's one thing to read about them, but actually programming and seeing them work was a whole other level of learning.

This internship has been more than just a learning experience; it's opened my eyes to the incredible possibilities of embedded systems and their impact on various industries. I'm leaving Indevco Phoenix with a wealth of knowledge.