INGENIERÍA
EN AUTOMATIZACIÓN

Universidad Autónoma de Querétaro

Facultad de ingeniería

Automation Engineering

SCADA Hardware Framework based on Modular FPGA

THESIS

That as part of the requirements to obtain the degree of
Bachelor in Automation Engineering

Present:
Luis Ernesto Fernández Rodríguez

Main Advisor:
Dr. Juvenal Rodríguez Reséndiz

Committee Members

Dr. Juvenal Rodríguez Reséndiz President	_____
Dr. Moisés Agustín Martínez Hernández Secretary	_____
M.C. Carlos Miguel Torres Hernández Vocal	_____
Dr. Edgar Alejandro Rivas Araiza Substitute	_____
M.C. José Luis Avendaño Juárez Substitute	_____

Centro Universitario
Querétaro, Qro.
December, 2020

Declaration of Authorship

I, Luis Ernesto Fernández Rodríguez, declare that this thesis titled, "SCADA Hardware Framework based on Modular FPGA" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Luis Ernesto Fernández Rodríguez
Querétaro, Qro. December, 2020

Dedication

I dedicate this thesis to my parents Arturo and Margarita.

I hope that this achievement will be proof
of the dream you had for me all those years ago
when you chose to give me the best education you could.

Acknowledgements

I would like to thank the following people who have helped me undertake this research:

My supervisors Dr. Juvenal Rodríguez Reséndiz and Dr. Moisés Agustín Martínez Hernández, for their enthusiasm for the project, for their support, encouragement and lots of patience. Thanks for the opportunity to start working on projects for the university.

The Autonomous University of Querétaro, for input throughout this Bachelor programme.

Professor José Luis Avendaño Juárez for his constant care since the moment he encouraged me to enter the faculty of engineering when I stumbled into the lab one day.

To my committee members for taking the time to read my thesis and their valuable comments that enriched my project.

To my dear *"kommilitone"*, fellow class mates, and friends thanks to all of you the restless nights and long days at the lab where never dull even if we were always striving to surpass each other.

To my brother who offered my invaluable support on during the good and the bad moments of my life.

To my grand parents who gave me all their support and care but were not able to see the project all the way to the end.

And to my loving parents, who set me off on the road to this Bachelors degree and in the pursuit of a doctor's degree a long time ago.

To everyone involved and who accompanied me throughout this journey, I would like to express my deepest gratitude to all those who have been side by side with me, along the long, but also short hours.

SCADA Hardware Framework based on Modular FPGA

by

Luis Ernesto Fernández Rodríguez

Abstract

This project describes the hardware design of a Supervision, Control and Monitoring System, SCADA, implemented by means of a Field Programmable Gate Array, FPGA. Implementing an FPGA allows the end user to tailor the controller to the application, with capabilities such as parallel processing, reconfiguration, and deployment of hardware accelerators. In an industrial environment this entails a system that is more economical than the commercially available alternatives (PLCs, SCADAs and distributed control systems), up to 90 % considering the cost of developing the prototype, and with a flexible, adaptable architecture and higher performance. The proposed hardware consists of four parts, analog and digital input and output cards, communications, and a base board connected to the FPGA.

As presented on the results this project serves as proof of concept and a basis to continue development of the controller with the necessary improvements and corrections to improve signal integrity on the hardware. Its performance in response time is a testament of the improvements provided by the use of a FPGA along with the modularity that allows to port the FPGA in between prototypes saving the additional cost implicated on the device.

Keywords:

- **ASIC:** Application Specific Integrated Circuits are used to implement both analog and digital functionalities in high volume or high performance. ASICs require higher development costs in order to be designed and implemented and are not reprogrammable.
- **FPGA:** Field Programmable Gate Array is a semiconductor IC where a large majority of the electrical functionality can be changed even after the equipment has been shipped out in the 'field'.
- **HDL:** A Hardware Description Language is a programming language used to describe the behavior or structure of digital circuits (ICs). HDLs are also used to stimulate the circuit and check its response. Many HDLs are available, such as VHDL and Verilog.
- **IP Core:** An Intellectual Property Core is a block of logic or data that is used in making a FPGA or ASIC design. Essentially they are reusable and portable elements that can be inserted into any vendor technology or design methodology as soft cores, hard cores or firm cores to easily implement functionality.
- **PLC:** Stands for Programmable Logic Controller. They are industrial computers used to control different electro-mechanical processes in many automation environments.
- **RLC:** Reconfigurable Logic Controller are industrial Automation controllers based on FPGA technology.

- **SCADA:** Acronym for Supervisory Control and Data Acquisition is a system of software and hardware elements that allow industrial organizations to have complete control and overview of their productive processes.
- **Soft Processor:** A Soft Core Processor is a reusable hardware module in the form of synthesizable HDL code. On FPGA devices soft cores are implemented using programmable logic resources, as opposed to hard cores baked into the silicon of the IC.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

When it comes to machine control, engineers today have more technology choices than ever: programmable logic controllers (PLCs), motion controllers, programmable automation controllers (PACs), industrial PCs, and even embedded solutions like field-programmable gate arrays (FPGAs). Before you swim through the alphabet soup to make a decision, you have to know where to start. As every controller's architecture is specifically designed with a certain application scenario in mind and continuously advance and innovate into new and more powerful solutions.

For decades, automation depended on electromechanical relays as the primary control fabric. Relays were basically designed to turn things off and on. As a result, even control of a simple task required nested layers of relays: one relay to turn the equipment on and off and a second relay to turn the power off and on to that first relay. Each had to be wired independently. Except for the most simple designs, systems quickly became expensive, complex, and power-hungry, as well as consuming a significant amount of space for the cabinet alone. Even worse, those walls of relays offered little flexibility. Making changes to the function or operating parameters of the machine typically required wholesale changes to the hardware, which was expensive, time-consuming, and difficult.

As time moved on relay based technology was eventually replaced by the programmable logic controller (PLC) as a more flexible, robust, and compact alternative. The PLC is a ruggedized control device consisting of a microprocessor and memory, along with select peripherals. Initial devices were programmed using ladder logic, to make them user-friendly to electricians accustomed to relays. More recently, in keeping with the EIC 61131 standard, they can be programmed with a variety of other languages, including structured text and function block.

PLCs are good at tasks like counting and timing, and managing I/O. However more computationally demanding applications are not properly fulfilled or accomplished with such devices. Alternatives to solve this problem like distributed control systems (DCSs), which offload certain process-control functionality to dedicated subsystems, and PC based solutions, like SCADA systems which use a collection of hardware and software tools to provide higher capabilities in terms of control and monitoring. Effective SCADA systems can result in significant savings of time and money. Numerous case studies have been published highlighting the benefits and savings of using a modern SCADA software solutions.

As new alternatives lately, embedded control systems have emerged. Devices that leverage field programmable gate arrays (FPGAs) to customize performance using hardware rather than software in order to satisfy unusual requirements that can't be satisfied by conventional control solutions. FPGAs provide a more accessible solution than the more expensive and difficult-to-develop application-specific integrated circuit (ASIC). The approach can be used to address one-off, high-performance projects that can't be addressed with a conventional solutions. Alternatively, FPGAs can be used to offload tasks like control logic and I/O triggering in highly complex systems, freeing CPU cycles to handle more complex tasks. Industrial

automation systems are in the advent of embedded single chip solutions with the capability to satisfy today's requirements and through reconfiguration provided by the use of FPGAs, those of problems to come.

1.1 Justification and Motivation

Automation systems have become very complex to achieve better performance, faster turnaround times, reduce waste, and downtime. Current equipment requires various functionalities, such as predictive maintenance, intelligent fault handling, fast response times or custom control algorithms, in addition to logical or process control.

This functionality is difficult to implement in traditional PLC-based architectures. Machines now a days need a combination of embedded systems and PLCs to deliver the expected functionality and performance.

Recently, the flexibility and benefits of FPGA technology have been demonstrated to carry out Reconfigurable Logic Controllers (RLCs) to implement parallel control strategies naturally. The implementation of SCADA and PLC systems according to the standard EIC 61131-3 [3] in FPGAs as demonstrated in works [17] and [5] allows to obtain shorter response times as well as to perform multiple tasks in parallel.

PLCs and SCADA hardware are widely used in the industrial market. This coupled with the need to continually improve your processes makes the deployment of reconfigurable hardware solutions an ideal alternative. Implementing controllers with FPGAs has advantages such as:

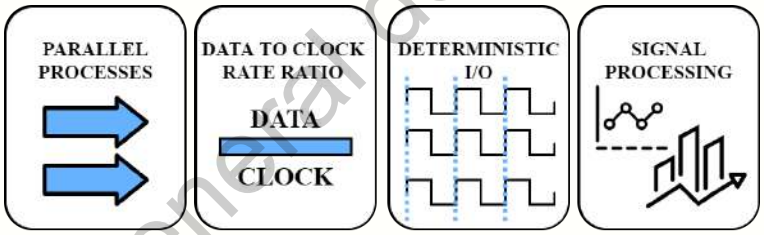


Figure 1.1: Strengths of FPGAs

- **Parallel data processing:** (such as multiple PID loops, multiple simultaneous analog-to-digital conversion channels or multiple processors as demonstrated in [9] with its multi-core PLC system).
- They are **deterministic systems** with high response speeds. (as shown in [14] with its remote terminal unit).
- **Signal processing:** (including digital filtering, demodulation, detection algorithms, frequency domain processing, image processing, or control algorithms).
- Controller **reconfiguration** according to the needs of the application (implementation of hardware accelerators)

These are all factors that make reconfigurable logic implementation a viable and highly competitive alternative.

With any significant benefit, there is often a corresponding drawback. In the case of FPGAs, the following are generally the main disadvantages of these kind of solutions.

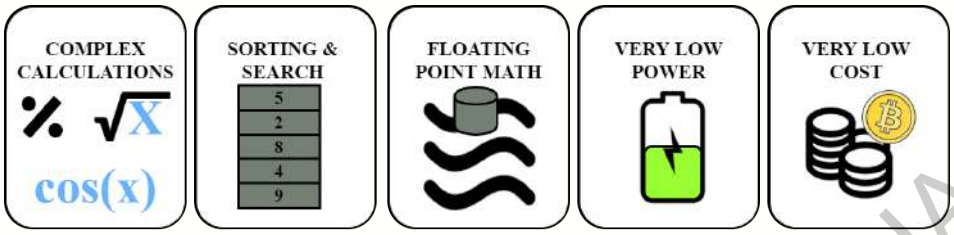


Figure 1.2: Weaknesses of FPGAs

- **Complex calculations** performed infrequently: if most of the implemented algorithms need to make computations less than 1 % of the time, logical resources are still allocated for each particular function even if they are kept idle.
- **Sort/Find:** These are processes of a sequential nature. There are algorithms that try to reduce the amount of computation involved, but in general, this is a sequential process that does not lend itself easily to the efficient use of parallel logical resources.
- **Floating point arithmetic:** The basic arithmetic elements within an FPGA are fixed point binary elements. In some cases, floating point calculations can be achieved, but will consume a lot of logical resources. There are certain exceptions, the floating point DSP blocks built into some FPGAs like the Aria Series from Altera.
- **Very low power:** some FPGAs have low power modes (hibernate and/or suspend) to help reduce current consumption. However, if power consumption is critical, generally a better option are low-power architecture microprocessors or microcontrollers.
- **Very low cost:** While FPGA costs have dropped dramatically in the past decade, they are generally more expensive than sequential processors.

1.2 Problem Statement and Context

Industrial automation consists of governing the activity and evolution of processes without the continuous intervention of a human operator. In recent years, systems called SCADA has been developed, through which the different variables that occur in a process or plant can be supervised and controlled. For this, various peripherals, application software, remote units, PLCs, communication systems, etc. must be used, which allow the operator to have full access to the process through a computer screen.

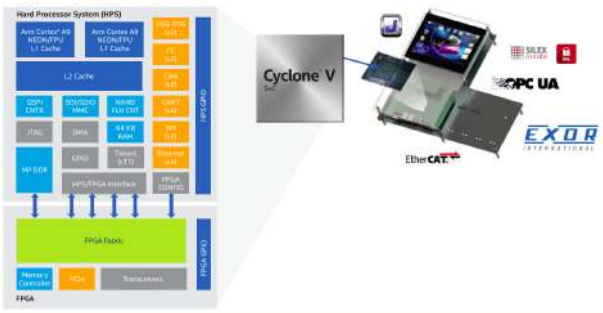


Figure 1.3: Single-Chip PLC Development Platform (ARM Architecture)

The programmable logic control systems employed range from small compact PLCs to high-end PLCs that can control thousands of nodes, their architecture must be scalable and flexible to accommodate size and performance requirements together. In order to implement hardware platforms with these characteristics and that allow the end user to adapt the controller according to the application, it is required to combine a high-performance application processor with a programmable logical network, such as those found in modern FPGAs, for customization and implementation of purpose specific hardware.

Although FPGAs are often used embedded in industrial automation control products, they do not provide access to the full functionality of the FPGA. There are products available in the market as the CompactRIO from National Instruments, controllers with a processor and FPGA, programmable by the user through the LabVIEW software suite and through modules provide connectivity and special functions. The CompactRIO architecture is made up of three main parts:

- The embedded real-time controller.
- The interconnect base containing the FPGA and communication peripherals.
- Interchangeable I/O modules.

Figure 1.4: NI RIO platform

NI RIO technology reduces the complexity of embedded hardware and of low-level and Hardware Description Languages (HDL) to provide easy, yet powerful, access to FPGAs. However, the system is closed architecture and prevents the end user from directly accessing the FPGA without using LabVIEW or developing their own expansion modules.

Hence the relevance of research and development of open architecture hardware for the implementation of SCADA systems. In this way the objective of the product resulting from the research is that it can be continued by other researchers and in the future develop a low-cost alternative commercial system open to the user.

1.3 Hypothesis

A SCADA type hardware system based on reconfigurable logic is capable of generating high performance tasks in scalable environments, while reducing the cost with respect to similar commercial architectures (PLCs, SCADAs and Distributed Control Systems), up to a 90% considering the cost of development of the prototype.

1.4 Objectives

1.4.1 General Objective

Develop a modular monitoring, control and data acquisition system (SCADA) using a reconfigurable logic device as the main controller, so that MSMEs have access to cutting-edge technology.

1.4.2 Specific Objectives

- Develop hardware modules for system expansion (communications, digital and analog inputs and outputs)
- Design the main controller, its architecture and interface with the modules, using a reconfigurable FPGA logic system with the focus on modularity and reconfiguration.
- Manufacture a functional prototype of the system, as well as its documentation for future development of expansion modules or further developments.
- Document of design guidelines for expansion modules.
- Validation of the correct operation of the hardware in an applied environment.

1.5 Thesis Overview

The present thesis is structured in the following manner.

- Chapter 2 details through references to researchers and authors the theoretical bases and past projects related to this work.
- Chapter 3 exposes the theoretical framework of the different aspects approached by the project.
- Chapter 4 describes a methodology that starts from the architecture and hardware design and continues onward with its execution on two hardware prototypes and implementation on digital devices. The main IP blocks and processor to be loaded into the FPGA are also illustrated.
- Chapter 5 and **??** present and analyze the results obtained.
- Chapter 6 establishes the conclusions of the project and if the objectives and hypotheses were achieved and verified along with future works and lines of research to develop.

Chapter 2

Literature Review

The literature review of this work has been built with a examination of the literature of different national and international authors that have had the most influence on this work and their contents are briefly described.

2.1 Previous Works at UAQ

As a part of the faculty of engineering at the Autonomous University of Querétaro the present and previous works of all alumni follow the tenet *el ingenio para crear no para destruir* which drives the academic personnel towards the continuous advancement of technology and works of knowledge at the university. The table 2.1 presents previous works developed at the university as a precedent to the proposed hardware system.

Table 2.1: Previous works at the university.

Year	Author	Title	Description
2007	R. Osornio Ríos	*Diseño de sistema de control para CNC de alta velocidad*	This project shows the relevance that programmable logic has for high speed CNC controllers. Programmable logical devices, such as FPGA, have advantages over microprocessors and DSPs; this is because their open architecture that allows the designer to specify the particular structure which is best for the application. The contribution of this work are: development of a control law and generator of polynomial profiles based on a FPGA applied to a high-speed CNC milling machine. As results demonstrate, reconfigurable logic is the most appropriate platform for developing high-speed digital controllers; it is preferable to other technologies available. [11].

2008	R. Santos Cruz	*Diseño e implementación de teclado industrial aplicado a una máquina de inyección de plástico*	This work presents the design and implementation of a communication protocol for a non-array industrial keyboard through its design in hardware. Due to the demand of automation equipment, local MSMEs (Micro Small and Medium companies) invest great amounts of money acquiring foreign machinery and devices. As effect, this investigation proposes a low cost digital module of programmable communication to be installed in an injection moulding machine. The advantage that the prototype offers is the use of a digital structure that allows simultaneous interaction with other keyboards. The implementation of this prototype was made using a Spartan 3 FPGA card, an interface and the keyboard by itself. [13].
2011	B. Muñoz Barrón	*Controlador modular y reconfigurable para máquina de inyección de plástico basado en FPGA*	Plastic injection molding is a complex process that involves many variables and several discrete input/output events, which represent a big computational load for the controller. Another problem is that commercially available controllers are close architecture, which conditions performance and efficiency. A way to solve the computational-intensive problem is to use a high performance device such a field programmable gate array (FPGA) to implement a PLC (Programmable Logic Controller). This work presents a FPGA-based PLC, fuzzy controllers, and a microprocessor network to control a plastic injection molding machine. The system was developed on a modular way and includes several control modules for continuous events, a processor for discrete events control, as well as communication modules. The result obtained showed the efficiency of the system to perform the injection process. The system developed is an excellent option for small and medium enterprises (MSMEs) dedicated to the plastic industry. [10].
2013	J. Sánchez Gómez	*Desarrollo de compilador para lenguaje escalera de controladores lógicos programables para aplicaciones industriales*	The need of more sophisticated tools has recently increased in order to improve production in industry and meet customer demand. These tools are generally fully integrated systems (Hardware and Software) which allow the implementation of more comprehensive processes. Integrated FPGA based developments (Field Programmable Gate Array) to these systems have greatly improved, due to their advantages, reprogrammability, reduced development costs and parallelism. Due to this, the present work shows the development of a compiler for a programmable logic controller (PLC) based on an FPGA, which will make the programming of the PLC faster and easier. In addition to the development of a GUI that will enable the use of ladder diagrams. Both the compiler and the GUI are based on structured and modular programming, making it upgradeable, scalable and portable. [12].

2.2 Main References in Literature

Industrial automation is one of the most important aspects of the digitization of production processes in industry. Thanks to automation systems and solutions, companies can control all their activity and manage the evolution of all processes without the continuous intervention of an engineer. SCADA technology allows exhaustive control of all devices in real time and also creates alarms and warnings and combines inputs and outputs within the same system as necessary without the need to migrate to another technology.

Given the rapid advance in the industry, areas for improvement are constantly being sought in automation systems and solutions. The table 2.2 below deals with some of the main references in literature on the development of industrial control systems related to the proposed hardware.

Table 2.2: Previous related works.

Year	Author	Title	Description
2006	A.Mililk	High Level Synthesis-Reconfigurable Hardware Implementation of Programmable Logic Controller	Implementation of a PLC (Hardware and Software) with the use of reconfigurable logic devices (based on FPGAs). Different architectures and the synthesis process of the control algorithm given in LAD are presented. The logical requirements of the controller and its performance are also compared against standard industrial solutions [8].
2006	C. Silva, C. Quintáns, J. Lago, E. Mandado	An Integrated System for Logic Controller Implementation Using FPGAs	An integrated system for the implementation of a logical controller using FPGA is presented. The hardware, in addition to performing typical automation tasks, contributes to obtaining additional features such as simulation and monitoring [15].
2007	C. Silva, C. Quintáns, M. Castro, E. Mandado	Methodology to Implement Logic Controllers with both Reconfigurable and Programmable Hardware	A hardware and software platform is described that allows the implementation of logic controllers with PLC (programmable logic controllers) and RLC (reconfigurable logic controllers). The software assists in the development of logic controllers by translating into instruction lists or hardware description language (HDL) codes. The hardware used consists of a commercial Simatic S7 and specific reconfigurable hardware consisting of a main board, which combines a Cyclone FPGA and a USB controller, and a set of analog and digital I/O boards. This hardware, in addition to performing typical automation tasks, contributes to additional features such as simulation and monitoring [16].
2010	S. Shirali, S. Ensafi, M. Naseri	RTU Hardware Design for SCADA Systems Using FPGA	This work presents a new hardware design for an RTU (remote terminal unit) that performs basic functions. This method has completely parallel computing that increases the speed of tasks, RTU without delay. By not using a processor, the risk of system crash is eliminated and RTU reliability is improved. The design consists of four main modules that include analog input, digital input, communication interface, and digital output. Being made with FPGA, there is a wide possibility of various reconfigurations so that you can generate adequate I/O using the correct number of these modules and have a suitable RTU [14].

2013	R. Czerwinski, M. Chmiel, W. Wygrabek	FPGA Implementation of Programmable Logic Controller Compliant with EN 61131-3	The document discusses the process of designing a simple programmable logic controller. The development of a PLC is presented, which is compatible with the EN 61131-3 standard. This part of the standard refers to programming languages. However, the PLC instructions are directly related to the hardware structure of the PLC. The instruction set, coding, and some elements of the design are presented (the translation of the instruction list to a hardware design implemented in an FPGA device) [6].
2014	D. Lee, E. Kim, J. Yoo	FBD to Verilog 2.0: An automatic translation of FBD into Verilog to develop FPGA	The paper proposes an automatic translation from FBD (Function Block Diagram: A PLC Software Programming Language) to HDL (Hardware Description Language). Implementing a machine translation tool, "FBD to Verilog 2.0", which helps software engineers design FPGA-based systems with their experience and knowledge. The case study using a prototype version of an FPGA-based control system showing that "FBD to Verilog 2.0" reasonably translates FBD programs for PLC to HDL [7].
2015	M. Chmiel, J. Kulisz, R. Czerwinski, A. Krzyzyk	An IEC 61131-3-based PLC implemented by means of an FPGA	The document discusses the design process of a programmable logic controller implemented by means of an FPGA device. The PLC implements at the machine language level a subset of the instruction set defined in the EN 61131-3 standard. Different aspects of the instruction list and the design of the hardware architecture are presented. All operations are fully implemented in hardware, so the solution is fast. The developed PLC is implemented using an FPGA device with the option of easily porting to an ASIC [5].
2015	M. Aamir, J. Poncela, M. Aslam, B. Chowdhry	Hardware Implementation and Testing of Reconfigurable RTU for Wireless SCADA	This document presents the implementation and testing of a remote terminal unit (RTU) design to control and monitor the industries of the oil and gas, water and energy sector. This particular implementation is based on a field programmable gate array (FPGA) that confers reliability and reconfigurability properties to the design. This results in a more powerful and optimized solution for executing monitoring data and control wirelessly. The features of the developed RTU are also compared to commercially available hardware considering its cost effectiveness.[4].
2018	A.Milik	Multiple-Core PLC CPU Implementation and Programming	The document presents a complete approach to multithreading of a control program in accordance with the IEC 61131-3 standard. The program is assigned to a multi-core CPU unit. The CPU consists of multiple independent bit and word CPUs. The computational synchronization mechanism is based on memory cells with semaphore access, which allow synchronization at the hardware level. The document presents in detail the architecture, the results of the implementation and the performance achieved. Likewise, a compiler is developed that translates the standard programming languages into an executable form with multiple threads [9].

Chapter 3

Theoretical Framework

3.1 Overview of a SCADA

According to Rodríguez [19], SCADA systems allow the management, data collection and control of any local or remote system thanks to a graphical interface that communicates the user with the system and allows him to make decisions about the operations to be carried out.

A SCADA system is a set of software applications, designed for production control, and hardware designed to act as an interface with instruments and actuators and a high-level graphical interface for the operator.

The system allows communication with field devices (programmable robots, data acquisition systems, etc.) to control the process automatically from the user interface, which is configurable and can be easily modified. In addition, it provides access to all the information generated by the process variables.

The programmable controllers and remote units used operate in accordance with the specification IEC-61131, developed by the International Electrotechnical Commission (IEC). This standard details the guidelines for the execution of the operating system, the data definitions, the programming languages and the set of instructions. For its implementation, current systems use microcontrollers, microprocessors or even computers for the implementation of control and supervision algorithms.

3.2 Characteristics of a SCADA System

Bailey and Wright [1] mention that a SCADA involves the collection of information and the transfer of data to the central site, carrying out the necessary analysis and control, and then displaying the information on a series of operator screens and in this way allow interaction, when the required control actions are transported back to the process.

According to Gómez *et al.* [20], SCADAs offer a tool that sets them apart from other industrial control systems: that of supervision. The control part is implemented by the hardware and instruments (PLC, logic controllers, etc.) applied on the plant. However, the responsibility of directing or correcting the actions that take place rests with the process supervisor; decision-making is in the hands of the operator.

This distinctly differentiates SCADA from classic automation systems, in which the control variables are distributed over the electronic plant controllers. This makes variations in the process very difficult, since once implemented, these systems do not allow optimal control in real time.

According to Gómez *et al.* [20], the main characteristics of a SCADA are the following:

- Acquisition and storage of data to collect, process and store the information received in a continuous and reliable way.

- Graphical and animated representation of process variables and their monitoring by means of alarms
- Execute control actions to modify the evolution of the process, acting either on the basic autonomous regulators (setpoints, alarms, menus, etc.) or directly on the process through the connected outputs.
- Open and flexible architecture with capacity for expansion and adaptation.
- Connectivity with other applications and databases, local or distributed in communication networks.
- Supervision, to observe from a monitor the evolution of the control variables.
- Transmission of information with field devices and other PCs.
- Database, data management with low access times.
- Presentation, graphical representation of the data. Operator interface or HMI.
- Exploitation of the data acquired for quality management, statistical control, production management and administrative and financial management.
- Alert the operator about changes detected in the plant, both those that are not considered normal (alarms) and those that occur in their daily operation (events). These changes are stored in the system for further analysis.

3.3 Benefits and Requirements of a SCADA

The SCADA package comprises a series of functions and utilities aimed at establishing the clearest possible communication between the process and the operator. According to Cerrada [2], the benefits and requirements offered by a SCADA system are as follows:

Benefits of SCADA systems

- Possibility of creating alarm panels to recognize downtime or an alarm situation, with incident log.
- Generation of historical data of the monitored variables of the plant.
- Creation of reports, notices and documentation in general.
- Execution of programs that modify the control law on the automaton (under certain conditions).
- Possibility of numerical programming, which allows high-resolution arithmetic calculations to be carried out on the computer's CPU and not on that of the automaton.

Requirements for SCADA systems

- They must be open architecture systems, capable of growing or adapting according to the changing needs of the company.
- They must communicate completely easily and transparently for the user with the plant team.
- Programs should be easy to install and use, with user-friendly interfaces.

3.4 Hardware Components

For Gómez *et al.* [20], a SCADA system needs certain inherent hardware components in its system to be able to process and manage the captured information, these are described below.

- **Central Computer or MTU *(Master Terminal Unit)*:** This is the main computer of the system, which monitors and collects the information from the rest of the substations connected to the field instruments. This computer is usually a computer that supports the HMI interface. The simplest SCADA system is made up of a single ordinate that supervises the entire station.

  The main functions of the MTU are as follows:

  - Periodically interrogates the RTUs and transmits instructions to them; usually following a master-slave scheme.
  - Acts as an operator interface, including displaying variable information in real time, managing alarms, and collecting and displaying historical information.
  - You can run specialized software that performs specific functions associated with the process supervised by SCADA.

- **Remote Computers or RTU *(Remote Terminal Unit)*:** These computers are located at the strategic nodes of the system, managing and controlling the substations; they receive the signals from the field sensors and command the final control elements by running the SCADA application software. Currently, programmable logic controllers (PLCs) are used with the ability to function as RTUs thanks to a higher level of integration and CPU with greater computing power.

- **Communication network:** This is the level that manages the information that the field instruments send to the computer network from the system. SCADA is found on standard formats such as RS-232, RS-422 and RS-485 from which, and through a TCP/IP protocol, the system can be connected to the existing network.

- **Field Instruments:** They are all those that allow both automation or control of the system (PLC, industrial process controllers and actuators in general) and those that are responsible for capturing system information (sensors and alarms).

3.5 Reconfigurable Computing

The reconfigurable computation consists of the use of hardware that can be adapted at a logical level to solve specific problems. It is based on two basic ideas: the first is that the architecture adapts to the algorithm to be implemented and not vice versa; the second is that it provides hardware support only to the algorithmic functions currently active.

High density programmable devices based on LUTs technology are used in most of today's reconfigurable computing applications, such devices are Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs). In theory, it is possible to make changes to the architecture design, even when the system is already installed and working. Another way to implement it is through processors that can be modified to include new functions and instructions designed to solve a particular problem. Reconfigurable logic can be used to implement functional units within the processor, co-processors, processing units, other processors, or a separate external unit.

3.6 FPGA

An FPGA is a general-purpose programmable multi-level device that integrates a large number of such logical devices on one chip. The size and speed of FPGAs are comparable to Application Specific Integrated Circuits (ASICs), but FPGAs are more flexible and have a shorter design cycle [18].

In technical terms, the FPGA is an array of programmable logic blocks placed in a programmable interconnect structure; In addition, it is possible to determine the functionality of the logical blocks, the interconnections between blocks and the connections between outputs and inputs. An FPGA is programmable at the hardware level, so it provides the benefits of a general-purpose processor and specialized circuit.

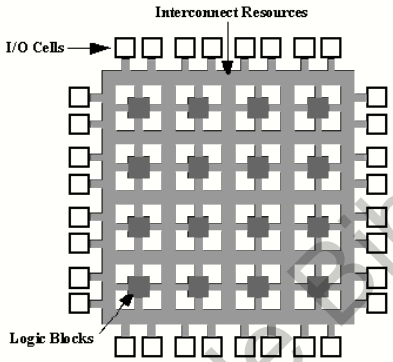


Figure 3.1: General structure of an FPGA.

The basic elements that make up an FPGA are shown in Figure 3.1 and are as follows:

- Logical blocks, whose structure and content is called architecture, in which we find a great variety, mainly in complexity, ranging from a simple gate to more complicated PLD-like structures.
- Interconnection resources, whose constitution and content is called routing structure.
- RAM memory that is loaded during the RESET to configure blocks and connect them.
- I/O Cells that allow the to connect to external interfaces.

A Programmable Logic Device (PLD) is a general-purpose device capable of integrating the logic of tens or hundreds of packets. It is programmed by the user using inexpensive programming hardware; however they are limited by their power consumption and delays. However, this allows a machine to be configured according to the needs that arise in an application, while allowing the feature to be reused at any time. These benefits have made FPGAs popular [21].

Development with FPGAs directs applications in new ways by presenting certain characteristics that are described in Table 3.1.

Table 3.1: FPGA innovation features.

Feature	**Description**
Instant implementation.	Because in-memory wiring is usually short, designs can, in effect, be implemented ”instantaneously.” This is a distinction from the conventional VLSI design, where it takes several weeks from it to manufacturing.

Dynamic Reconfiguration	With some architectures, part of the FPGA can be configured at run-time.
Design Safety	The configuration of an FPGA disappears when the chip is turned off. There is a range of applications where this additional level of design security is important; however, there is another family of FPGAs where the configuration can be programmed only once remaining fixed.

The main advantages of FPGAs compared to other devices for the design of digital systems are shown in Table 3.2.

Table 3.2: Advantages of FPGAs over ASICs.

Advantage	**Description**
Low Tooling Costs	For each design implemented in an ASIC, the cost of each one is around thousands of dollars, which must be amortized over the total number of units manufactured. The more units that are built, the impact of the development cost is reduced. FPGAs do not require this process, so they are excellent for relatively small volumes (1,000 to 10,000 units).
Rapid development	The ASICs manufacturing process takes several weeks from the completion of the design to the release of the developed parts. An FPGA can be programmed in minutes by the user. Similarly in an FPGA a modification to correct the design can be done quickly and cheaply. Immediate development, in turn, makes the appearance of new products faster and reduces the time for sale.
Effective design verification	Due to non-recurring engineering costs and manufacturing delays, ASIC users must verify their designs by extensive simulation prior to manufacturing. To test the functionality of the design in a system, it must be simulated over long periods of time. FPGAs reduce these problems since they can perform a check on the circuit. Designers can house the design and use functional parts as a prototype.
Low cost of testing	All integrated circuits must be tested to verify correct manufacture. The test is different for each design. Those implemented in an ASIC incur associated costs. Manufacturers programs verify that each FPGA can work for any possible design that is implemented in it. Users only need specific tests for their scheduled architectures.

3.7 Embedded IP

Altera's Nios II processor is the most widely used processor in the FPGA industry, since provides a great flexibility and supports RTOS. It is an Altera's 32 bit proprietary architecture, whose modules are interfaced through the Avalon bus. These modules can be from interval timers to RAM controllers. To implement the Nios II processor, Quartus provides a tool called Platform designer, wich allows us to define and customize all modules of the embedded system.

The basic modules that comprise our system are listed below:

- Nios II processor
- On-Chip Memory
- JTAG UART
- System-ID
- Clock Source and PLL

3.7.1 Nios II processor

Designer can choose among three options when selecting the processor, depending on the hardware resources he wants to use. Obviously, the more resources used, the processor provides higher performance. The three options are: economy core, standard core and fast core. The economy core uses only 600 logic elements and two M9Ks. This is the simplest core, whereas standard and fast cores are absolutely deterministic, jitter free real-time performance with unique hardware real-time features. Economy core would be enough for this application, but the fast core was used since the FPGA provides enough resources.

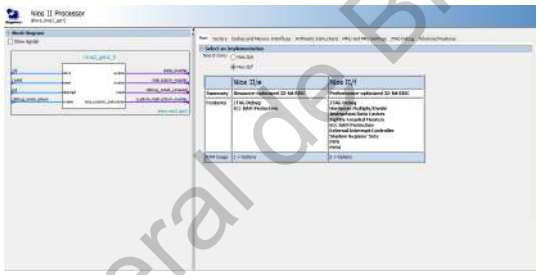


Figure 3.2: Nios II core configuration.

This processor has a RISC architecture of 32 bits, using between 1'400 and 1'800 logic elements. Besides, it works at 50 MHz, which is enough for this application (in fact the frequency of work is tuneable, but it was not needed to change it). Both the reset vector and the exception vector are located on the On-Chip memory.

3.7.2 JTAG UART

This is the controller needed to program the processor form the user's PC.

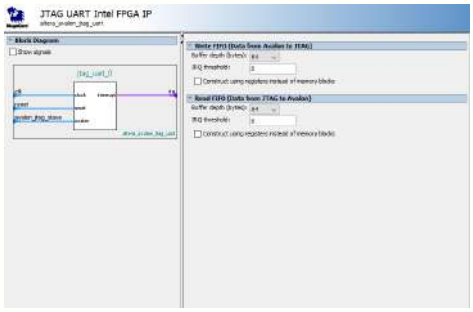


Figure 3.3: JTAG UART configuration.

3.7.3 On-Chip Memory

On-Chip memory with data width of 32 bits and a total size of 20'480 bytes was implemented. This is the RAM used by the processor and where the program is stored.

Figure 3.4: On-Chip memory configuration.

3.7.4 System-ID

The system ID core with Avalon interface is a simple read-only device that provides Platform Designer systems with a unique identifier. Nios II processor systems use the system ID core to verify that an executable program was compiled targeting the actual hardware image configured in the target FPGA.

Figure 3.5: System ID configuration.

3.7.5 Clock Source and PLL

The PLL core provides access to the dedicated on-chip PLL circuitry in the Intel Stratix and Cyclone series FPGAs. The core takes a system clock as its input and generates PLL output clocks locked to the input.

Figure 3.6: PLL configuration.

Chapter 4

Methods

This chapter documents the description and implementation of the SCADA hardware platform developed. The explanation of the system architecture and design is addressed to later analyze each of the modules and interface IP for the NIOS II processor. A list of the software and hardware resources necessary for its implementation is also added.

4.1 Specification

The research tests will be carried out in the facilities of the Autonomous University of Querétaro in the Mechatronics Laboratory in a first phase and will subsequently be carried out in the field to verify the capabilities of the system. Below are the software, instruments and equipment that are required for system implementation.

4.1.1 Software

- Autodesk Eagle Student Software 9.6.2
- Software Suite Quartus Prime Lite 18.1
- NI LabVIEW Student Edition Software Suite 2019

4.1.2 Hardware and Equipment

- Soldering iron station and material (Weller WES-51)
- Personal computer (DELL G5-5587 Intel i7-8750H, 32GB, Nvidia 1050ti)
- Measuring devices
  - Oscilloscope
  - Multimeter
  - Signal Generator
  - Logic Analyzer
- Electronic devices

- DE10-Nano FPGA Cyclone V Development Board
- Electronic components (check materials used on Appendix E)

4.2 Design

The scope development of an industrial control solution is quite large encompassing development of the hardware platform, programming software and compiler, monitoring software and interface firmware. In order to make the project feasible considering the available resources and equipment the extent of this work is currently limited to the hardware framework as a starting point to kick off the software side of development.

The design of the proposed SCADA hardware framework requires a large amount of work to be developed in various areas such as circuit board design, digital systems, programming, among others demonstrating the skills and knowledge obtained during last 5 years of undergraduate studies.

4.2.1 Architecture

Starting with the conception of the project the initial statement and requirements were established as the design guidelines to adhere as development progressed and to clarify a development milestone to be reached. As stated on the hypothesis the final product should have the following requirements:

- Low cost system (comparing cost of development against similar commercially available alternatives).
- Final product must be capable to tailor the controller to the application, with capabilities such as parallel processing, reconfiguration, and deployment of hardware accelerators.
- Implement an FPGA as the main controller along with the necessary IP blocks to interface with the hardware.
- The hardware platform must comprise of a minimum of analog and digital input and output cards, communications card and base for interconnection.
- The FPGA device must be fully isolated from the rest of the system for protection of the device.
- Communications card should include a couple of the following: CAN, RS-485, UART, Ethernet of WiFi and RS-232.
- The base for interconnection should allow to easily customize the system by adding or removing expansion cards.
- Power delivery must be through the base board (no external power necessary for each module).
- Hardware must be manufacturable (this according to the capabilities of the contracted board house).

As a target device the DE-10 Nano Cyclone V SoC development kit was selected as Intel's SoC FPGA platform integrates in a single device a dual-core ARM Cortex-A9 processor and a Cyclone V FPGA allowing development to follow two different paths if desired, using the embedded ARM processor as the main CPU along with the IP cores developed to interface with the hardware or using a NIOS II 32-bit soft-processor developed by Altera as the main CPU. The latter was selected as the development tool-chain for the NIOS II is easier to work with but still leaves the option to switch platform in the future.

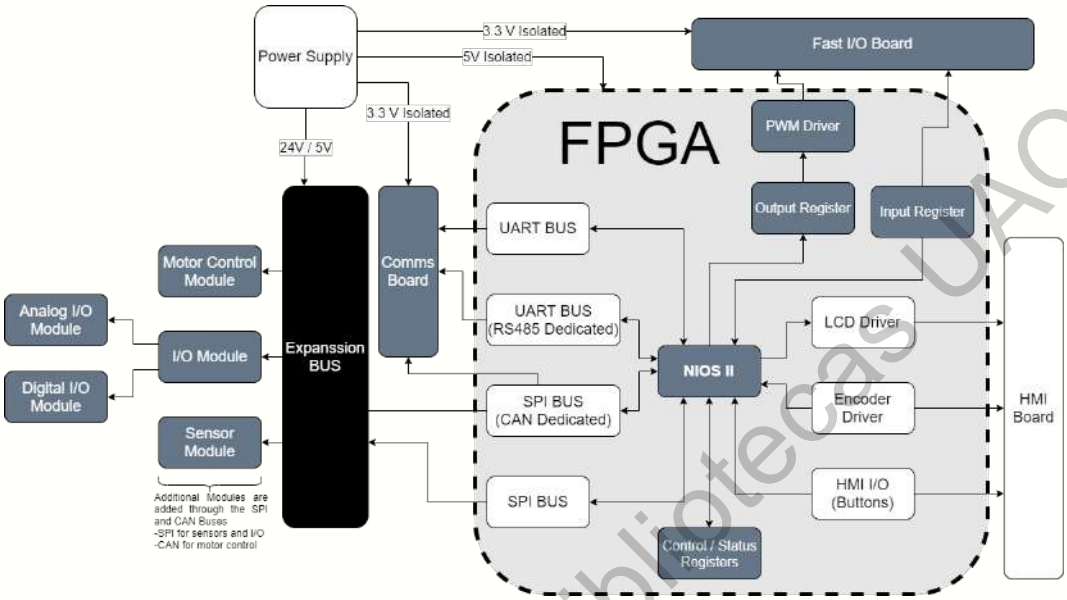


Figure 4.1: System architecture centred on NIOS II soft-processor

Complying with these guidelines and with a target in mind the following architecture was developed as presented on figure 4.1

The design revolves around the NIOS II as stated before with IP blocks to interface with the peripherals of the system. The hardware features as follows:

- LCD display as a HMI integrated on the controller.
- Communications through SPI CAN Bus controller and UARTs for RS-232, RS485 and USB.
- Digital I/O that can be directly addressed.
- SPI and CAN Buses for accessing expansion modules.
- Isolation from power and data lines for the FPGA.

4.2.2 Hardware Proposal

As part of the design of a fully integrated system a specialized hardware platform tailored to the application is required. In a manner similar to the Compact RIO from national instruments the proposed implementation consists of a base which interconnects, powers and isolates the FPGA and the interface modules.

- The main controller operates at 3.3V and is susceptible to noise from the 24V digital I/O or the 0-10V analog inputs or the 0-5 analog outputs.
- Communications done through SPI bus allow to interface with peripherals such as ADCs, DACs, I/O controllers, sensors, among many other devices as SPI and I2C are commonly used protocols for peripheral devices. On the other hand CAN bus allows fast transfer of critical data collected from an external controllers as the protocol includes CRC checks and is designed for noise suppression.
- Power distribution for the expansion modules is designed for 24V 1A and 5V 1A, additional power required must be provided externally to the module. The system is designed as to use a single 24V

power supply with the use of integrated isolated regulators for the FPGA and non-isolated regulators for the expansion bus.

- Additional communication interfaces allow to interface with the UART ports on the FPGA with the use of isolated RS-232, RS485 and USB controllers (communication protocols for RS-485 like Modbus-RTU must be implemented in software).

- Interface modules to be implemented include analog and digital I/O as they represent the basic interfaces required for industrial controllers and must comply with the IEC-61131 standard for the voltage levels used, 0-24V digital 0-10V analog I/O.

4.3 Implementation

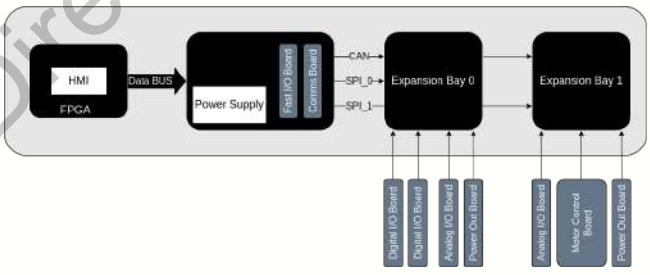
The following sections describe the development of the first run of the prototype on its final revisions, describing each of the modules that composes it.

4.3.1 First Prototype

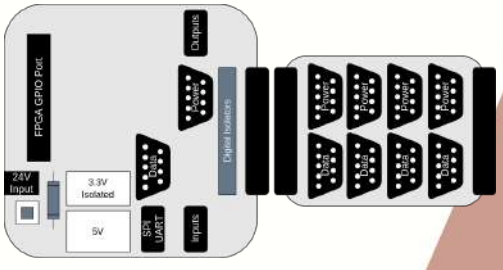
The first prototype implemented many of the characteristics just described and worked as a alpha version, proof of concept prototype, in order to better visualize how the hardware would interact and begin refining details on the devices operation. In a manner similar to the agile software development methodology implementing a complete ”build of the system” allowed to start HDL development directly on a prototype, remark necessary changes, design errors, and new features to be added on the next iteration.

Interconnection Base

The main circuit board is the interconnection base that provides data isolation and power passthrough for the expansion modules and connection to the FPGA device. Figure 4.2a provides a block diagram overview of the connection between the different parts of the system. The base board and the expansion bay design can be seen in closer detail in figure 4.2b. The main data bus between the expansion bay and the FPGA are two instances of SPI IP blocks that use ADUM3154 Spisolator from Analog devices to provide galvanic isolation and the necessary slave select lines.



(a) Proposed hardware block diagram

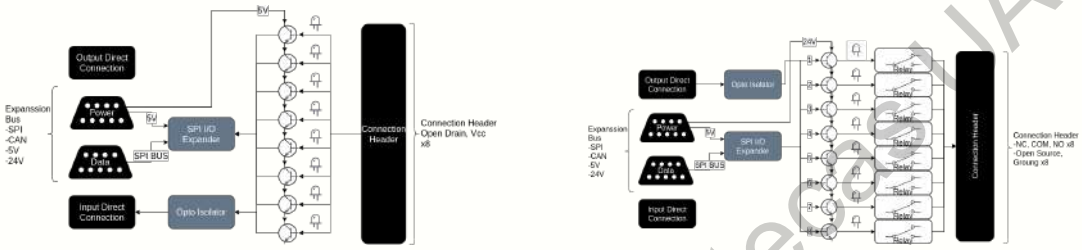


(b) Base board block diagram

Figure 4.2: Interconnection base overview

Digital I/O Module

The following block diagram illustrates the design of the digital I/O module. The module operates with the MCP23S17 a 16-bit general purpose I/O expander. The IC contains 2 banks of 8-bit I/O with bank A dedicated as 24V tolerant inputs and bank B dedicated as Relay Outputs.



(a) Inputs (b) Outputs

Figure 4.3: Digital I/O module architecture

Analog I/O Module

The following block diagram illustrates the design of the analog I/O module. The module operates with the MCP3202 a 12-bit successive approximation analog to digital converter (ADC) and a MCP4822 a 12-bit digital to analog converter (DAC). Each IC contains 2 independent channels with the DAC capable to output in the range of 0-5V and the ADC capable to input in the range of 0-5V.

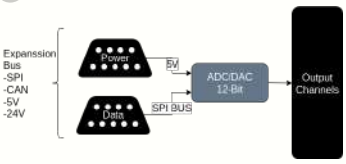


Figure 4.4: Analog I/O module architecture

Communications Module

The following block diagram illustrates the design of the communications module. The module provides CAN, RS-232, RS-485 and USB UART connectivity with their corresponding transceivers. The CAN bus is looped back into the interconnection base to provide connectivity though the expansion bay.

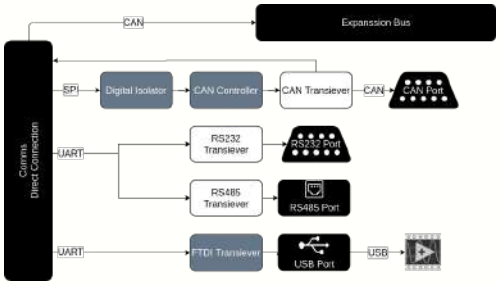


Figure 4.5: Communications module architecture

More details on the design can be observed on appendix A on the schematic diagrams and appendix B for the circuit board designs.

4.3.2 Second Prototype

Following the same line as the first prototype on the agile development, its second iteration is based on the results of the initial tests, with the implementation of additional missing features and new functionality added. Acting as a beta version its basic functionality is better defined and perfected but all changes and additions still require further testing and validation. Some of the features considered to be added on the second run of the prototype are:

- ROM information to identify on software each expansion module.
- Correct separation between modules on expansion bay.
- Motor control and sensor modules added to the system.
- Raspberry Pi based controller variant.
- Numerical displays on expansion bay to show slave number
- 0-10V tolerant inputs on analog I/O module.
- Signal filtering and additional channels on analog I/O module.
- Multiplexing of the 2 SPI buses to control allocation.
- Isolation on all communication interfaces
- Better integration of the FPGA development board on the device.
- Front panel with LCD display.

Interconnection Base

The interconnection base presents also a different design as part of the change of connectors from 2x DB9 to D25 in addition to multiplexers to control allocation of the SPI bus on the expansion ports with number displays to show the corresponding slave allocated to that port. Controlling allocation time of the SPI bus enables the user to prioritize access to certain ports over others for faster data collection.

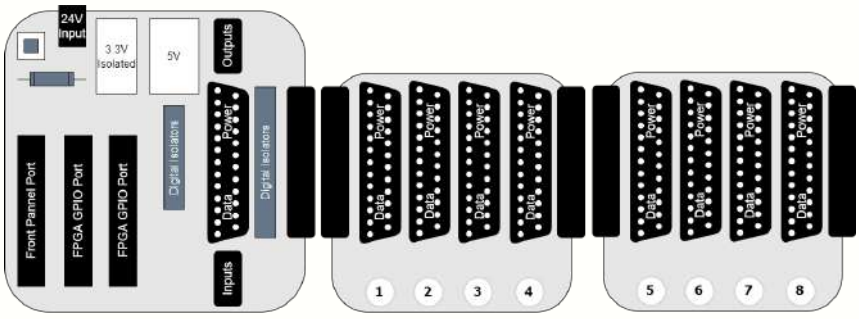


Figure 4.6: Base board version 2 block diagram

Motor Control Module

The implementation of a motor controller module allows the system to be used as a robotics platform in addition to its main purpose as a SCADA controller. The motor driver module is designed based on the CY8CKIT-037 from Cypress Semiconductor as it allows the user to configure it to operate as a stepper, BLDC or DC motor controller with encoder feedback.

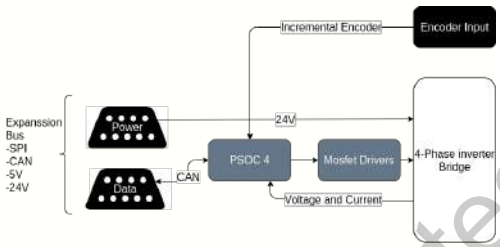


Figure 4.7: Motor control module boardarchitecture

EEPROM Adapters

Identification of the connected modules through software was not originally planned when designing the original modules, adding such functionality requires a connector with more I/O and the EEPROM memory. Such changes implicate the design of a new PCB or the use of a ”mod-board” which is cheaper and easier as the smaller PCB interfaces with the existing modules while making the necessary changes.

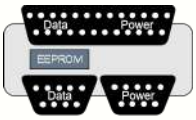


Figure 4.8: Adapter for new connector and EEPROMs

Front Panel

The front panel is designed to better integrate the communication ports of with a small user interface. Communication ports remain the same as in its prior iteration with the addition of a USB-UART bridge for direct connection with a PC if required. The main change is the interface which consists on a 8x2 character display, quadrature encoder and indicator LEDs for RUN, STOP, and ERROR status.

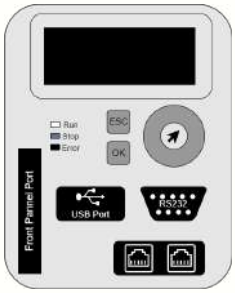


Figure 4.9: Front panel interface block diagram

4.3.3 Programming and Implementation of NIOS II Processor

The center of the HDL design is the NIOS II processor that interfaces with the different peripherals using IP blocks that interface the Avalon Memory Map interface of the processor with the Parallel, UART, SPI, etc ports as necessary. Use of the embedded processor requires a clock and PLL component to establish its operating frequency, a RAM memory used to map the peripherals registers and use as program memory and a ID to identify each of the processors used (in case more than one processor is used a mutex is required to mediate the access to memory and the peripherals or each processor must be implemented independently with the downside of having no shared resources between them).

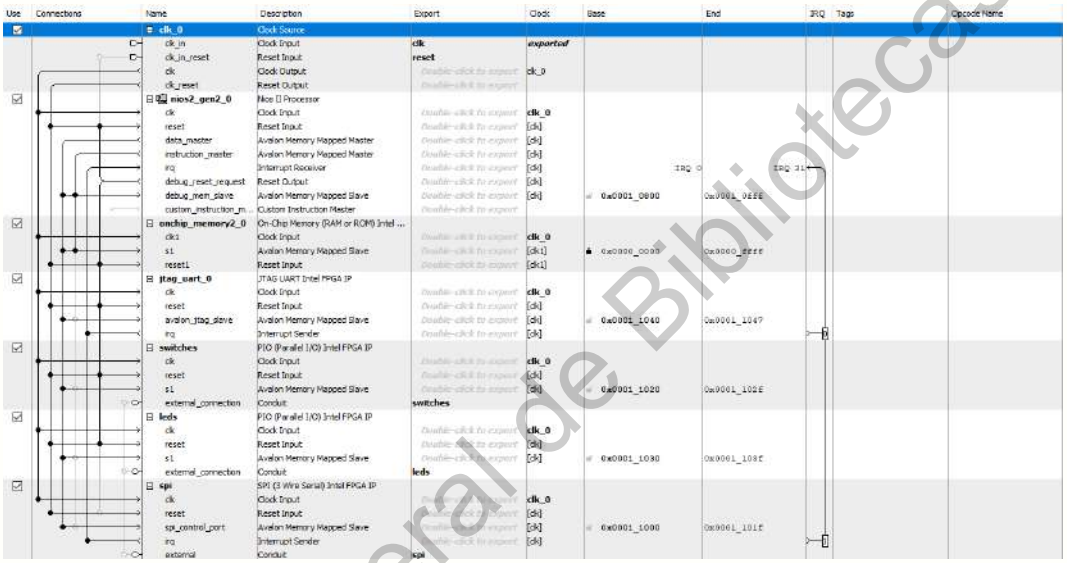


Figure 4.10: Basic Configuration required for NIOS II processor

As initially planned having a FPGA allows the use of such functionality with parallel processing of communications, control signals, etc. in addition to the use of hardware accelerators to improve performance if required.

4.3.4 IP Block Design for Peripheral Interfaces

The present implementation of the system has the required interfaces by the second iteration of the hardware. The following peripherals are used:

- 3 UART ports for communication with USB, RS-232 and RS-485.
- 1 UART port for interface with the EEPROMS on the expansion modules, the FPGA is capable of I2C communication but a small microcontroller is used as a bridge for the current implementation.
- 2 SPI and 3 Parallel I/O for the spi interface on the expansion bus, one of the PIO is used to control bus allocation with the high speed muxes.
- 2 Parallel I/O to control the direct integrated I/O.
- 1 Parallel I/O to control the character LCD, there are 2 different LCD controllers provided by Intel but only work with some models when using the old version of the softcore processor as such the LCD is controlled with port manipulation or commonly known as bit-banging.

- 2 Parallel I/O to control the LEDs, encoder and keys on the front panel.

The interconnection between components is as shown in figure 4.11. The platform designer interface acts as a patch panel to connect the instruction and data buses, clocks, resets and interrupt signals from the slave devices to the master or processor used. Interrupts are controlled by assigning priority levels from 0 to 31 with 0 being the highest and reserved for the JTAG UART used by the processor for debugging. As the memory map interface is used by the peripherals the SDRAM or on-chip memory must be assigned and mapped for the address ranges occupied, the rest is available to the processor as program memory.

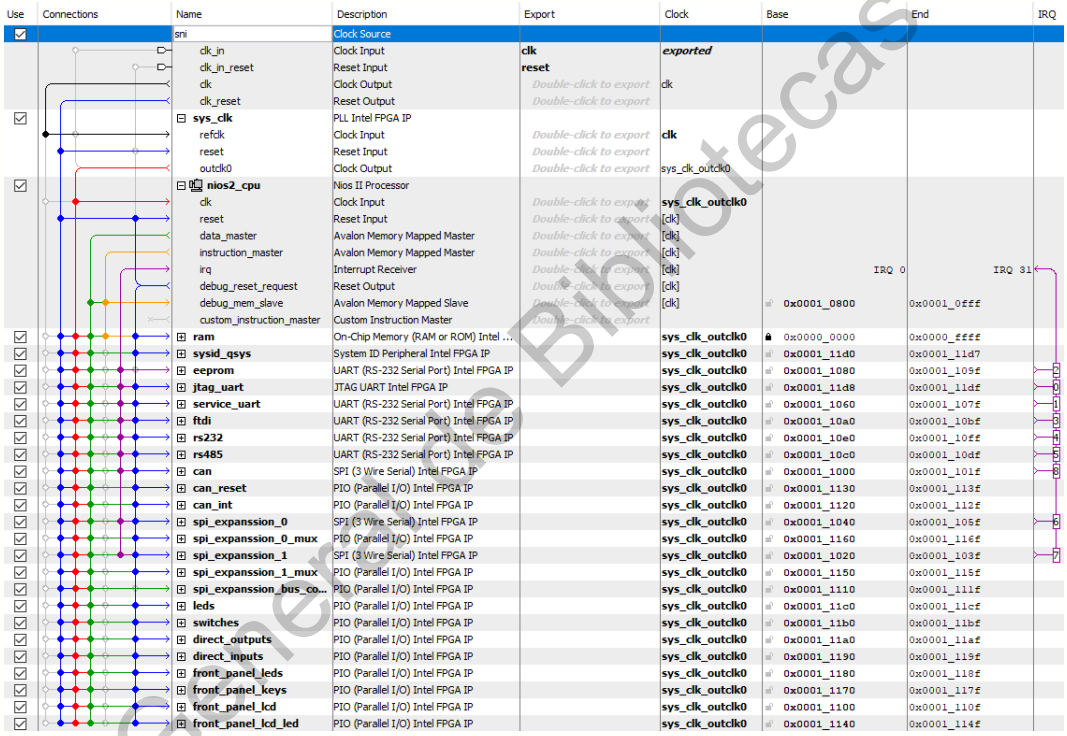


Figure 4.11: Complete configuration of the NIOS II used for the project

The peripherals are designed by Intel to use the memory map interface which maps the registers of the peripherals within RAM memory for the processor. This interface uses two 32-bit wide buses to communicate with the processor or a specialized controller in a addressable Read/Write interface between master and slave components. Other components can be designed following the Avalon Interface Specifications provided by Intel.

HDL code for the configuration is generated by platform designer and only a top level code to define the hierarchy and connect other user generated code is necessary as presented on Appendix C.

As part of the design workflow a board support package and libraries are generated for the corresponding configuration of the NIOS II. Low-level functions to access the control and data registers of the selected peripherals are generated as well a as a header file containing the configuration created on platform designer. This allows to program and configure the softcore processors operation in a manner similar to any other 32-Bit microcontroller or microprocessor. For ease of use, functions abstracting the low level access to registers were developed for the UART, SPI and Parallel I/O interfaces. Operation of this peripherals is described on the Intel Peripheral IP Handbook.

Functions developed handle:

- Interrupts

- Initialization and communications configuration
- Data input and output
- Front panel interface (inputs and status LEDs)
- Front panel LCD (commands and data for the 8-Bit interface)
- Simple testing functions

```
//-----------------------------------------------------------------------------
//Description : PIO Operations For NiosII
//-----------------------------------------------------------------------------

#ifndef PIO_H_
#define PIO_H_

/************************************************** ************************
* Public function prototypes
************************************************** ************************/

//Direct "Fast" I/O Functions
alt_u8 FAST_IO_GET_CHAR();
void FAST_IO_PUT_CHAR(unsigned char data);

//Front Panel I/O Functions
void ERROR_ENABLE();
void ERROR_DISABLE();
void STOP_ENABLE();
void STOP_DISABLE();
void RUN_ENABLE();
void RUN_DISABLE();
alt_u8 ESC_GET_CHAR();
alt_u8 OK_GET_CHAR();

//Integrated I/O Functions
alt_u8 SWITCHES_GET_CHAR();
void LEDS_PUT_CHAR(unsigned char data);

#endif /* PIO_H_ */
```

(a) Parallel I/O header file functions

```
//-----------------------------------------------------------------------------
//Description : UART Interrupt Handler & Q Buffer For NiosII
//-----------------------------------------------------------------------------

#ifndef UART_H_
#define UART_H_

/************************************************** ************************
* Public function prototypes
************************************************** ************************/
void FTDI_INIT(unsigned int BaudRate);
void FTDI_ISR();
unsigned char FTDI_EMPTY();
unsigned char FTDI_GET_CHAR(void);
unsigned char FTDI_PUT_CHAR(unsigned char in_char);
unsigned char FTDI_PUT_STRING(unsigned char in_char[]);

void RS232_INIT(unsigned int BaudRate);
void RS232_ISR();
unsigned char RS232_EMPTY();
unsigned char RS232_GET_CHAR(void);
unsigned char RS232_PUT_CHAR(unsigned char in_char);
unsigned char RS232_PUT_STRING(unsigned char in_char[]);

void RS485_INIT(unsigned int BaudRate);
void RS485_ISR();
unsigned char RS485_EMPTY();
unsigned char RS485_GET_CHAR(void);
unsigned char RS485_PUT_CHAR(unsigned char in_char);
unsigned char RS485_PUT_STRING(unsigned char in_char[]);

#endif /* UART_H_ */
```

(b) UART header file functions

```
//-----------------------------------------------------------------------------
//Description : SPI Slave Select & Buffer For NiosII
//-----------------------------------------------------------------------------

#ifndef SPI_H__
#define SPI_H__

/************************************************** ************************
* Public function prototypes
************************************************** ************************/
void SPI_ISR();
unsigned char SPI_EMPTY();
unsigned char SPI_GET_CHAR(unsigned char slave, unsigned char reg);
void SPI_PUT_CHAR(unsigned char slave, unsigned char data);

void slaveSelect(unsigned char spiChannel);
void slaveDeSelect(unsigned char spiChannel);

//External I/O Functions
void MCP23S17_INIT(unsigned char slave, unsigned char address);
alt_u8 MCP23S17_GET_CHAR(unsigned char slave, unsigned char address);
void MCP23S17_PUT_CHAR(unsigned char slave, unsigned char address, unsigned char data);
void test_outputs(unsigned char slave, unsigned char address);
void test_inputs(unsigned char slave, unsigned char address);

//External ADC Functions
alt_u16 MCP3202_GET_CHAR(unsigned char spiChannel, unsigned char adcChannel);

//External DAC Functions
void MCP4822_PUT_CHAR(unsigned char spiChannel, unsigned char dacChannel, alt_u16 value);
alt_u16 mVConvert(alt_u16 voltage);
void test_dac(unsigned char spiChannel);

#endif /* SPI_H_ */
```

(c) SPI header file functions

```
//-----------------------------------------------------------------------------
//Description : UART Interrupt Handler & Q Buffer For NiosII
//-----------------------------------------------------------------------------

#ifndef LCD_H_
#define LCD_H_

/************************************************** ************************
* Public function prototypes
************************************************** ************************/

void LCD_PUT_COMMAND(alt_u8 cmd);
void LCD_PUT_DATA(alt_u8 cmd);
void LCD_PUT_STRING(char *msg);
void LCD_INIT();
void LCD_ENTER();
void LCD_HOME();
void LCD_CURSOR(alt_u8 pos);
void LCD_TEST();

#endif /* LCD_H_ */
```

(d) LCD header file functions

Figure 4.12: Header files for abstraction functions

Chapter 5

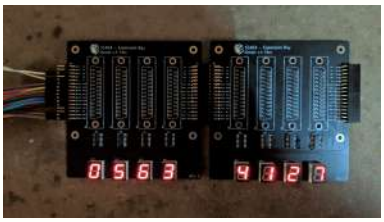
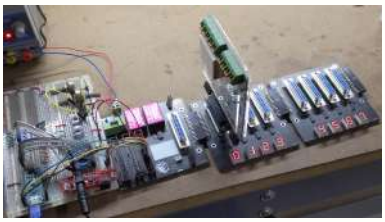
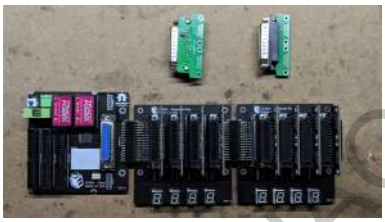
Results

Employing the manufactured hardware for the second prototype the following tests were performed to validate its proper operation and to verify its performance in terms of response time and stress handling to ensure the noise isolation barrier is operating correctly.

5.1 Functional Validation

After manufacturing and assembly of the various circuit boards, all were inspected for proper connection between data and power lines, some small errors were corrected in a couple of them as the pin-out of some parts did not match the one on its design, others were corrected with the addition of a ”Modboard” to adapt the PCB without need to re-manufacture. The following procedure was performed on each of the boards:

1. Assembly and probe testing for no open or short circuits.
2. Testing on power rails for 24V, 5V and isolated 3.3V.
3. Operational test on the various ICs with test stimulus
4. Data transmission test through the isolation barrier.
5. Full system operation testing with ATmega328p microcontroller as main processor (initial tests were performed with a microcontroller to ensure proper operation as the FPGA is a more sensitive device.)
6. Full system operation using the FPGA.
7. Response time testing on digital inputs and outputs.
8. Noise isolation (data frames integrity) during stress tests

(a) SPI multiplexer test (b) SPI isolators test (c) Assembled prototype

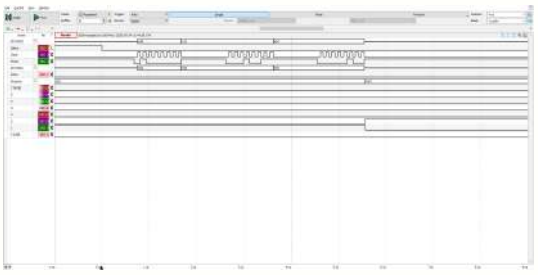
Figure 5.1: Assembly and functional validation tests

5.1.1 Modules Communication Data Frames

The expansion modules act as slave devices to modify the ports and capabilities of the main controller. After validation of the design and manufacturing inspection was performed, the base board was tested for the signal integrity due to the isolation barrier in between the controller and the modules and for the proper operation of the analog and digital I/O modules. More in depth detail on the code used for testing is available on appendix D.

Digital outputs testing

Testing on the digital I/O was performed as a turn-on cycle of each output one at a time while checking for continuity on the terminals of the relays with a multimeter. On figure below the SPI command string, denoted by the 0x40 value (frame start for a write operation), can me seen along with the outputs cycling one a time.

(a) Output cycle. (b) SPI Command string

Figure 5.2: Digital outputs test data

Digital inputs testing

Testing on the digital I/O was performed mirroring the inputs on the outputs of the same module. Below are presented two read operations denoted by the 0x41 command (frame start for a read operation) at the beginning, next to the register address and the corresponding values.

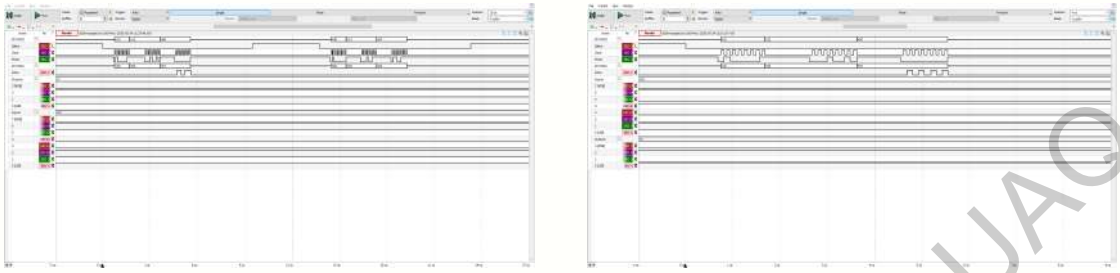


Figure 5.3: Digital inputs test data

DAC testing

Testing on the digital to analog converter was performed by ranging the outputs from 100mV-2000mV for output A and from 100mV-4000mV for output B correspondingly. On the figure below a SPI string for 2.32V on output A can be seen with the first 4 bits for the command and the succeeding 12 bits for the DAC value.

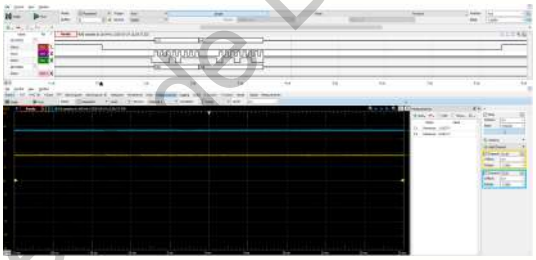


Figure 5.4: DAC test data

ADC testing

Testing on the analog to digital converter was performed by ranging the input voltage with a potentiometer while monitoring the obtained value with a oscilloscope and a serial monitor. On input A a value of 1.55V and on input B a value of 2.53V can be measured which is close to the 1.58V applied on input A and the 2.27V applied on input B.

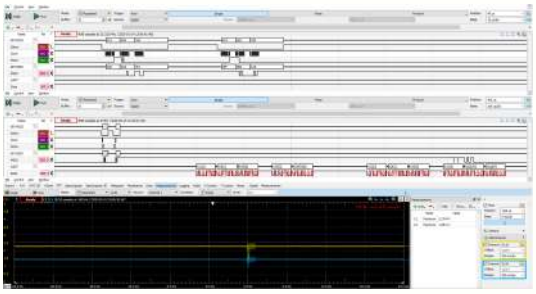


Figure 5.5: ADC test data

5.1.2 NIOS II and Peripheral controller Test

In a similar manner to the tests performed using a microcontroller for the functional validation. The modules were put through their paces using the NIOS II on the FPGA. Using the function for the Parallel I/O and SPI described on chapter 4. A necessary 10ms delay was added in order to not saturate the storage memory from the oscilloscopes serial decoder and to more easily verify the data frames.

The SPI isolators installed on the prototype have an integrated CS multiplexer so an additional function is necessary to control which slave is selected and deselected on the 2 SPI buses.

```
void slaveSelect(unsigned char spiChannel){
  alt_u16 controlByte;
  if(0 <= spiChannel && spiChannel < 4){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_0_BASE, 1<<0);
    controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE, (controlByte|
   ALTERA_AVALON_SPI_CONTROL_SSO_MSK));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_0_MUX_BASE, spiChannel);
  }else if(4 <= spiChannel && spiChannel < 8){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_1_BASE, 1<<0);
    controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE, (controlByte|
   ALTERA_AVALON_SPI_CONTROL_SSO_MSK));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_1_MUX_BASE, spiChannel);
  }
}

void slaveDeSelect(unsigned char spiChannel){
  if(0 <= spiChannel && spiChannel < 4){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_0_BASE, 1<<0); /* no need
   to setup slave select register as only one slave but just in case*/
    //controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE, 0);//(controlByte|(~
   ALTERA_AVALON_SPI_CONTROL_SSO_MSK)));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_0_MUX_BASE, 0);
  }else if(4 <= spiChannel && spiChannel < 8){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_1_BASE, 1<<0); /* no need
   to setup slave select register as only one slave but just in case*/
    //controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE, 0);//(controlByte|(~
   ALTERA_AVALON_SPI_CONTROL_SSO_MSK)));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_1_MUX_BASE, 0);
  }
}
```

The following function performs a turn-on cycle of each of the outputs one at a time. Sending the necessary command through SPI to the module of the specified slave number and address (the address number is part of the data frame and allows to have 8 different modules on the same bus).

```
void test_outputs(unsigned char slave, unsigned char address){
  MCP23S17_PUT_CHAR(slave, address, GPIOB_value);
  GPIOB_value = GPIOB_value<<1;
  if (!GPIOB_value) GPIOB_value = 0x01;
}
```

Figure 5.6: Digital outputs test data

The following function verifies the state of the digital inputs on the module and loops it back to the outputs of the same module.

```
void test_inputs(unsigned char slave, unsigned char address){
  GPIOA_value = MCP23S17_GET_CHAR(slave, address);
  MCP23S17_PUT_CHAR(slave, address, GPIOA_value);
}
```

Figure 5.7: Digital input test data

The following code tests the digital to analog converter by ranging the outputs from 100mV-2000mV on output A and from 100mV-4000mV on output B.

```
void test_dac(unsigned char spiChannel){
  MCP4822_PUT_CHAR(spiChannel, 0, mVConvert(DAC_value));
  MCP4822_PUT_CHAR(spiChannel, 1, mVConvert(DAC_value*2));
  DAC_value = DAC_value + 100;
  if(DAC_value*2 > SUPPLY_VOLTAGE){
    DAC_value = 100;
  }
}
```

Figure 5.8: DAC test data

The following code obtains the data on the analog to digital converter and prints it to the NIOS console.

```
void test_adc(unsigned char spiChannel){
    alt_u16 dataADC;
    dataADC = MCP3202_GET_CHAR(spiChannel, 1);
  alt_printf("Data from ADC...%4d\n", dataADC);
  IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, dataADC & 0x00FF);
}
```

Figure 5.9: ADC test data

5.2 Performance Verification

5.2.1 Response Time

In order to quantify the performance of the system, its response time was measured using the digital I/O module. As the system has a isolation barrier, measurement of the time required to obtain an input or assert an output was not possible without the use a sync signal to time the measurement that would itself have an additional delay. As such the *"two way response time"* was measured as to obtain an approximation

of the time required to obtain an input or assert an output considering the delay and processing time is approximately the same for both directions.

For the measurement a signal generator was used to send a pulse of 30us every 300us on one of the digital inputs and the signal was looped back by the controller to one of the outputs (the relay output was disabled as the switching frequency would be too high for it withstand, relays usually operate up to 50Hz).

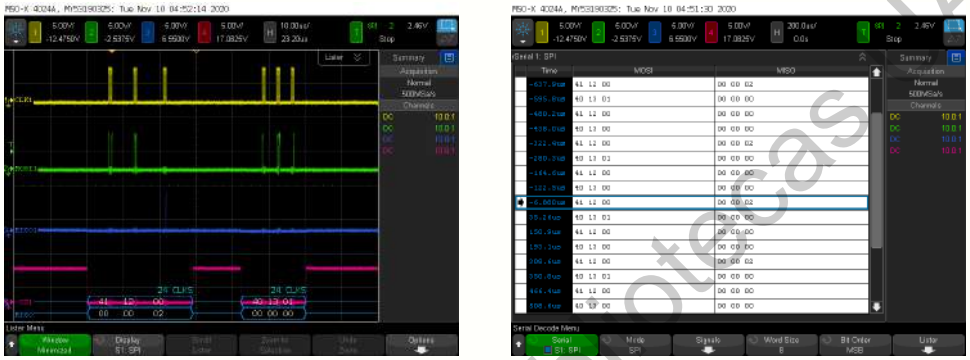
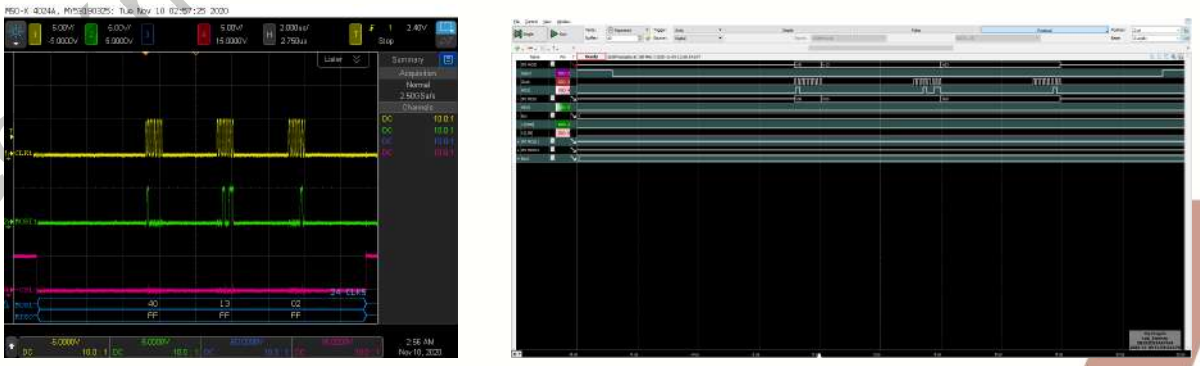


Figure 5.10: Response time test data

On figure 5.10 the data frames for the output input and output commands are shown, *40 13 XX* for the output command where *XX* are the states for the outputs and *41 12 XX* for the input command where *XX* are the states for the inputs.The time required for the system to loop back the signal from the generator is approximately 42us measuring it with the help of the timestamps. Approximating that the response time is equal in both directions give us a value of 21us.

5.2.2 Stress Tests

Following performance tests, the integrity of the communication signal lines were tested with the use of a switching 12V 1.68A DC load from a server fan motor. Setting the relay output to switch at a frequency of 50Hz caused certain issues to arise on the data and power lines of the expansion modules.



(a) Load side (b) Controller side

Figure 5.11: Stress test data frames

As seen on figure 5.11 the data is transmitted by the controller and received on the module making it switch normally, as expected with no missed data packets. But a dip on the 5V power line of 400mV can

be seen whenever the load is turned on and off with only 1 module and 1 load active it's not a problem but since the system can have up to 56 outputs connected at once with the correct configuration of modules (it was not possible to test such scenario since only 2 such circuit boards were manufactured) this could be a potential problem that could cause data packets to be lost, the controllers on the modules to continuously reset and to cause a problem with the equipment it's controlling.

The other issue identified was when ever the load was switched off, the data lines were affected by the return current causing spikes of voltage on the data lines. As can be clearly seen on figure 5.12 the load experiences a spike from -33.2V of 17V but isn't affected by it and keeps operating. On the other hand, this introduces noise on the data lines as a pulse of 2.1us that oscillates between -6.5V and 15V. Data packets don't seem to be lost at a significant percentage as the speed difference between the data transmission rates and how often the *"glitch"* is presented is orders of magnitude higher. The possible causes for the this problem are the ground pour used on the top and bottom layers of the PCBs that is not grounded and acts as an antenna making the data lines susceptible to noise and due to the non continuous return path for the data lines which is bad to signal integrity.

Figure 5.12: Voltage spikes on the data lines

5.3 Results Analysis

It's important to observe that the system operates properly with its digital and analog I/O modules as seen on the results with fast response times of approximately 21us as proper data and power isolation between the controller and load sides of the device. Tests performed for functional validation demonstrates the system works as designed and the same goes for the IP cores used on the FPGA, the NIOS II allows a high level of customization that would be difficult to reproduce with a microcontroller or microprocessor, need more UART lines, need more parallel I/O, with a softcore processor you got it. The cost to pay is the use of low level access functions for the processor, requiring the user to abstract them to perform more difficult operations.

Nevertheless the device requires a third iteration with better layout considering the return paths of the signals as well as better use of reference planes and decoupling capacitors to route power and reduce voltage dips on the power lines. Further testing and validation is required for the device to be robust and reliable while remaining low cost as it was intended from the beginning.

5.4 Future Works

This work is considered as a first delivery on the development of a the control and monitoring of systems using FPGAs as its core. As shown on the results it's still a work in progress with a pending third iteration designed to address many of the concerns of the first 2 prototypes used at the time of writing.

Many of the changes made were already discussed but following is a series of developments required for the prototype to perform as expected of a industrial grade controller.

- PC interface for the device, to properly be a data acquisition system the user must have a interface that displays the data from the device in real time.
- I/O controllers, ADC, DAC and Communication ICs must be Industrial grade components.
- Network controller implemented for remote access.
- Use of a SOM FPGA instead of development board or design of proprietary board for better device integration.
- Testing using higher loads, was not possible at the time of writing due to difficulties accessing such equipment.
- Real time clock integration for scheduling events and time stamps on data acquisition.
- Design specification and documentation for user developed modules.
- Development of 0-20mA/4-20mA sensor interface modules.

One of the objectives of this work is to open and facilitate research on the following topics:

- Motor control modules and Sensors for the system to act as a possible robotics platform
- Development of lesser equipped versions with a microprocessor like a raspberry pi or similar.
- Make use of this thesis to address development of IP cores for application specific controllers using the platform.
- Make use of the prototype and hardware for digital systems courses as a development platform.
- Development of a ladder or FBD to C compiler for its use with the hardware platform.

Chapter 6

Conclusions

On this work a hardware framework to implement a Supervisory Control And Data Acquisition system by means of an FPGA to develop a new and economic industrial controller compared to existing commercial alternatives like the Compact RIO from National Instruments. As presented on Appendix E the total cost of both runs of the prototype are about $1/5$ of the cost of the embedded base of Compact RIO alone with most of it spent toward the FPGA development board currently in use. Better integration of the IC onto the Base board would greatly reduce its cost. During the development of the objectives the challenge of designing a complete hardware platform and complementary HDL code was proposed with basic functions such as digital and analog I/O and UART communications as well as the necessary documentation for further developments. Certain milestones have been achieved, mainly on documentation, design and development of the first prototypes and testing the capabilities of the NIOS II processor as a customizable controller. As demonstrated on chapter 5 the system operates as expected with performance issues to be resolved in terms of signal integrity and protection, certain additional features are still not implemented and will be passed on to the third run of the prototype.

As presented the scope of this project requires further *R&D* with development necessary on the software side to be comparable to the Hardware and Software suites provided by large companies producing this kind of control systems. This work contributes toward existing development of FPGA based industrial controller platforms by providing a flexible and open source hardware framework and to make cutting-edge industrial control technology available at a low cost. Further developments on the project will be presented in due course.

Appendix A

Circuit Diagrams

A.1 First Prototype

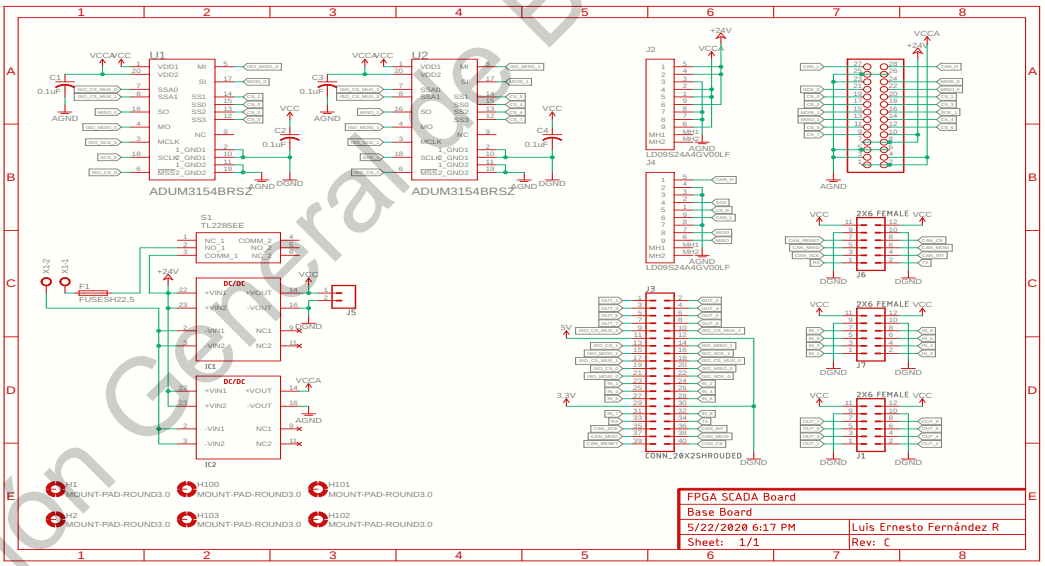


Figure A.1: Base Board Schematic Diagram

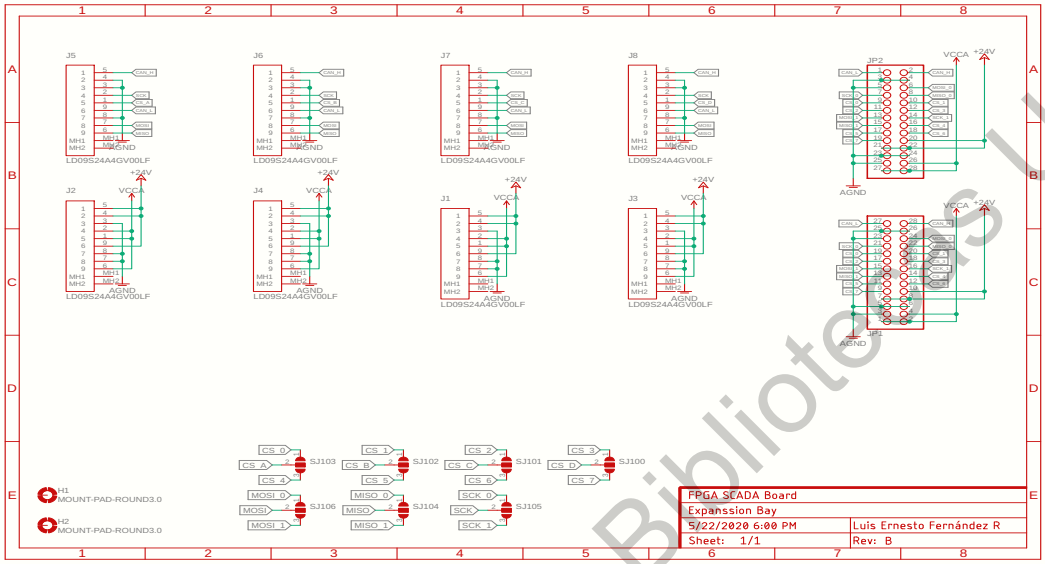


Figure A.2: Expansion Bay Schematic Diagram

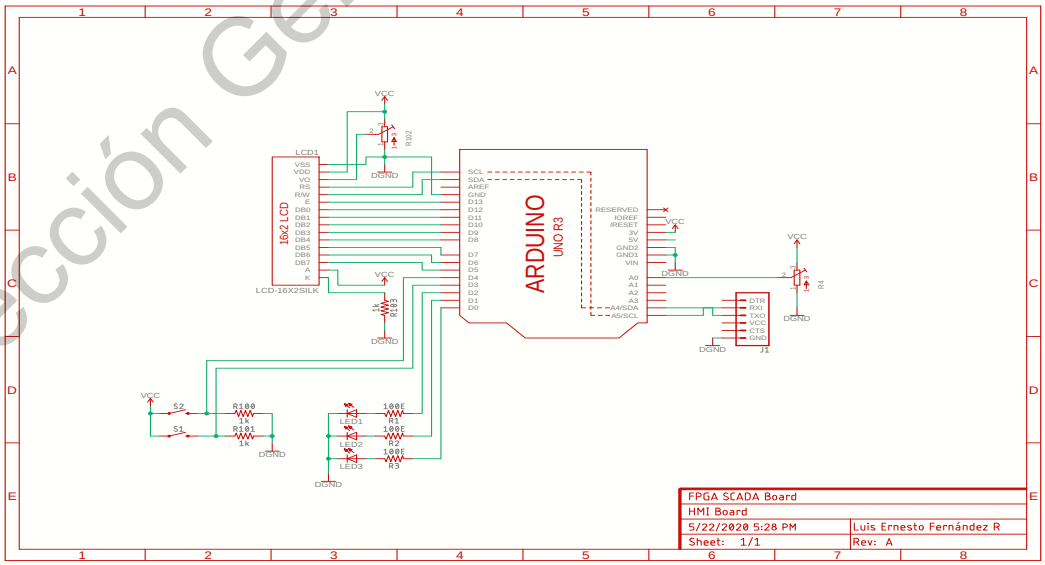
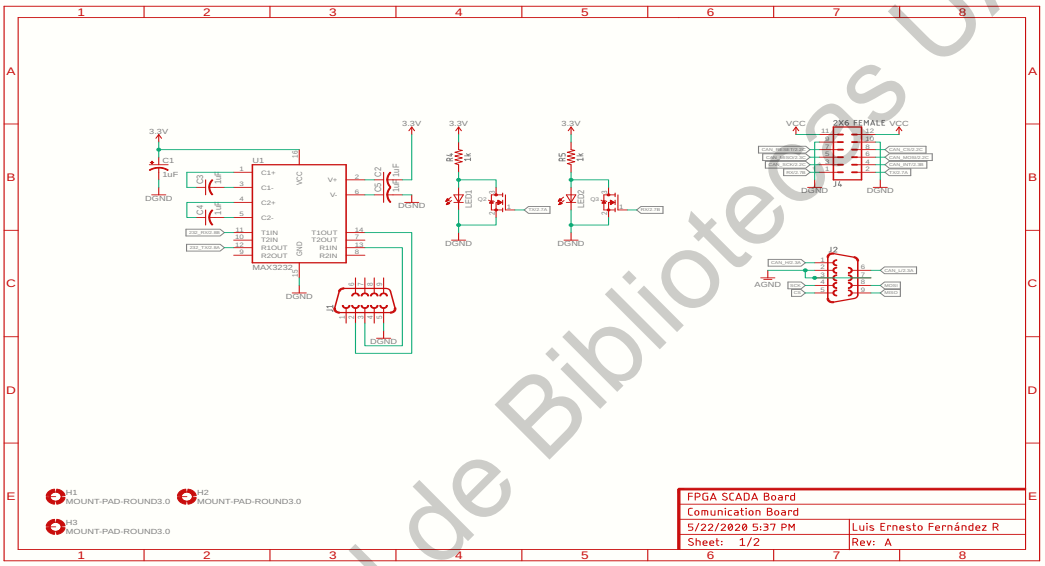


Figure A.3: HMI Add-on Schematic Diagram



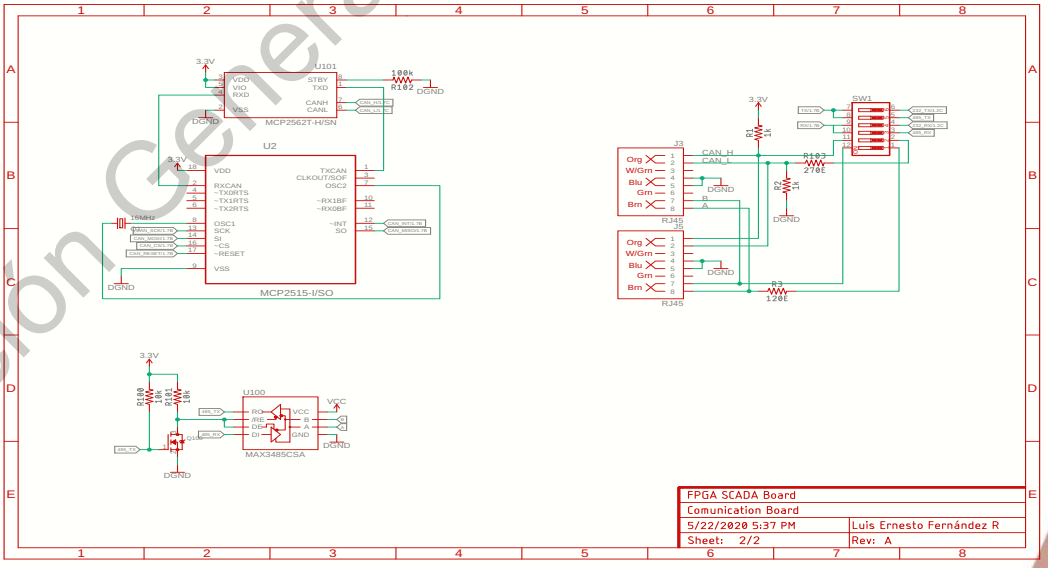
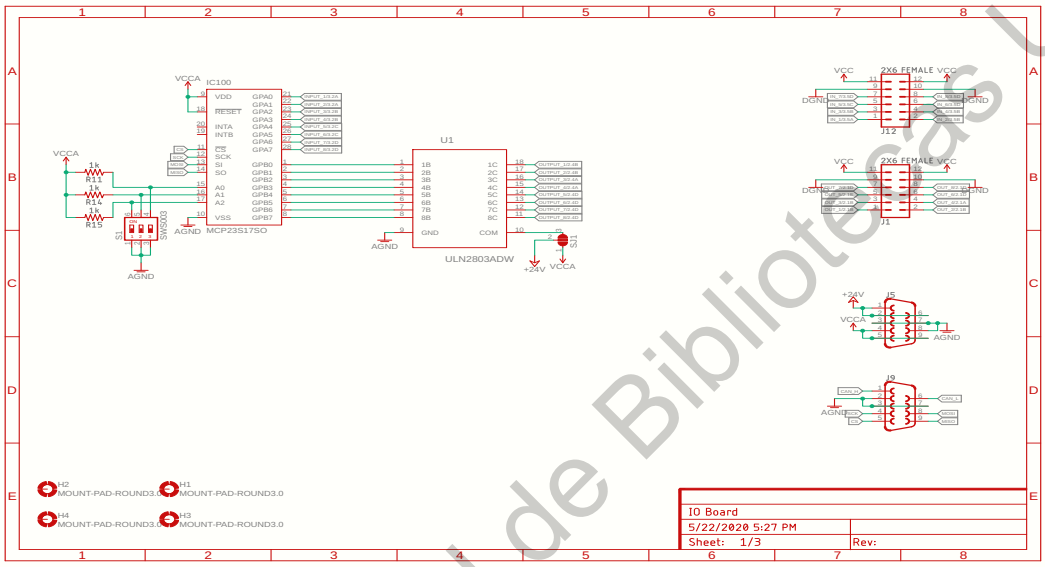


Figure A.4: Communication Module Schematic Diagram



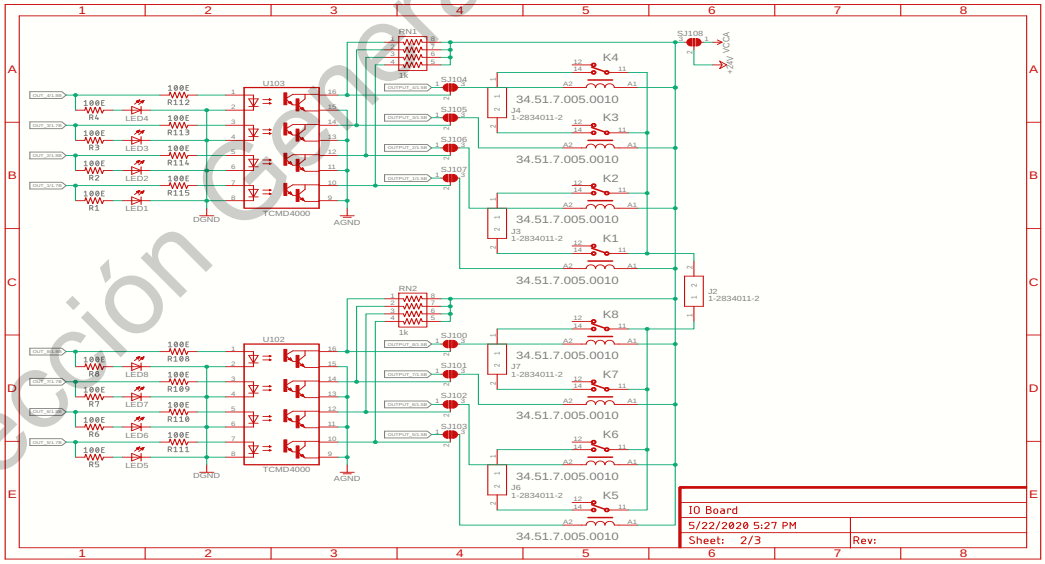


Figure A.5: Digital I/O Module Schematic Diagram

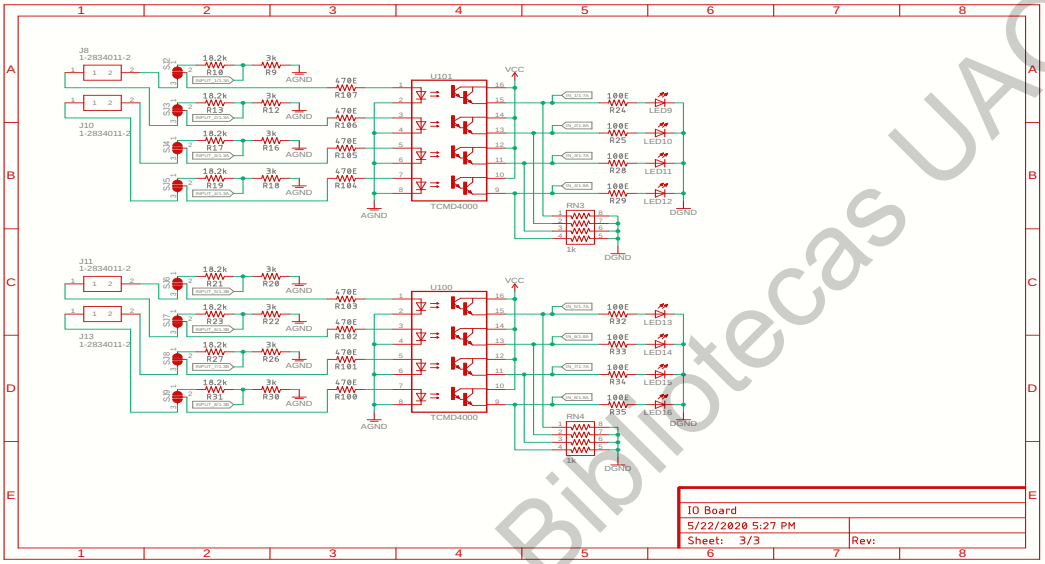


Figure A.5: Digital I/O Module Schematic Diagram

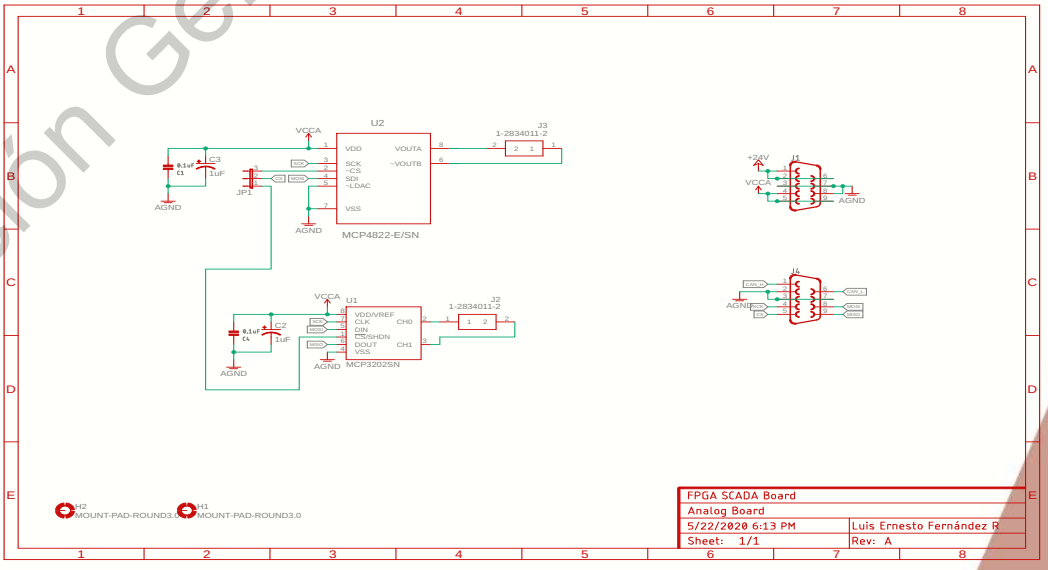


Figure A.6: Analog I/O Module Schematic Diagram

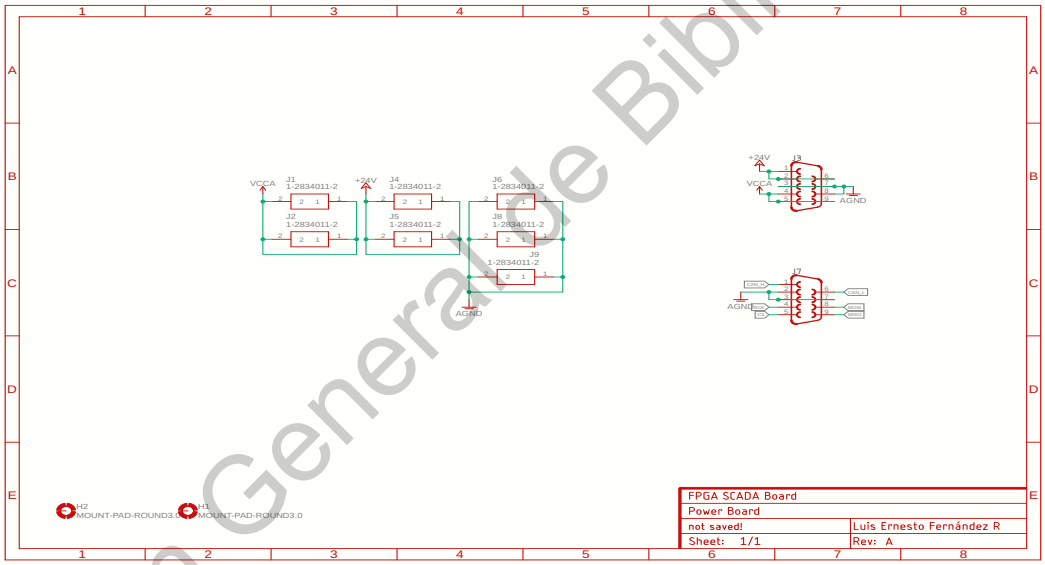
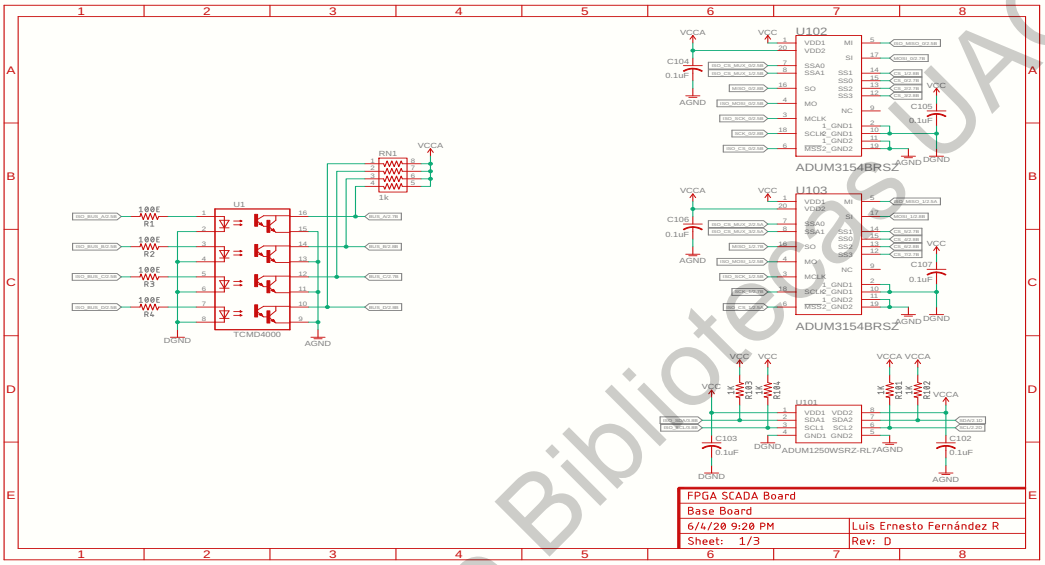


Figure A.7: Power Delivery Module Schematic Diagram

A.2 Second Prototype



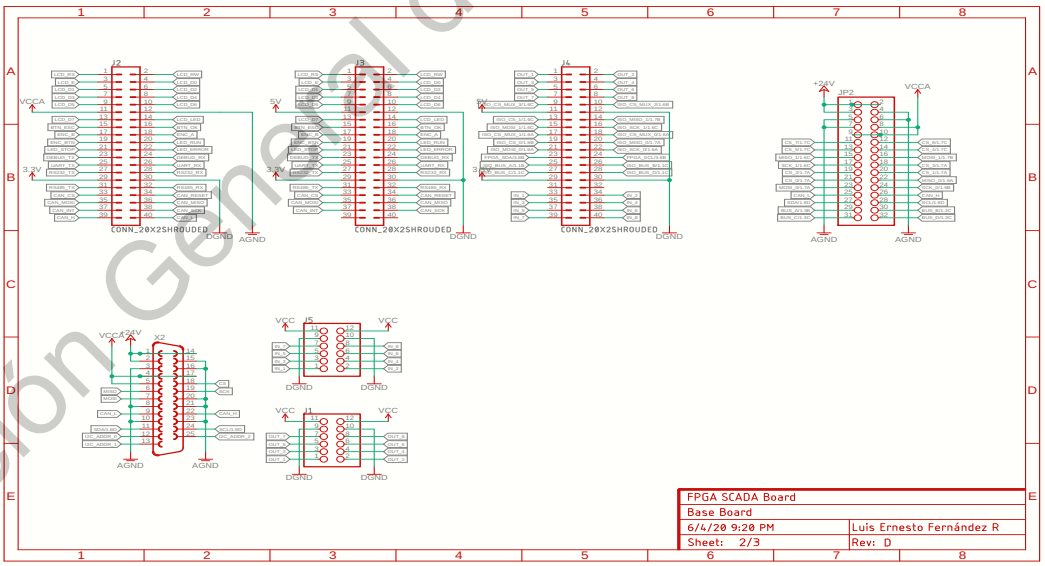


Figure A.8: Base Board Schematic Diagram

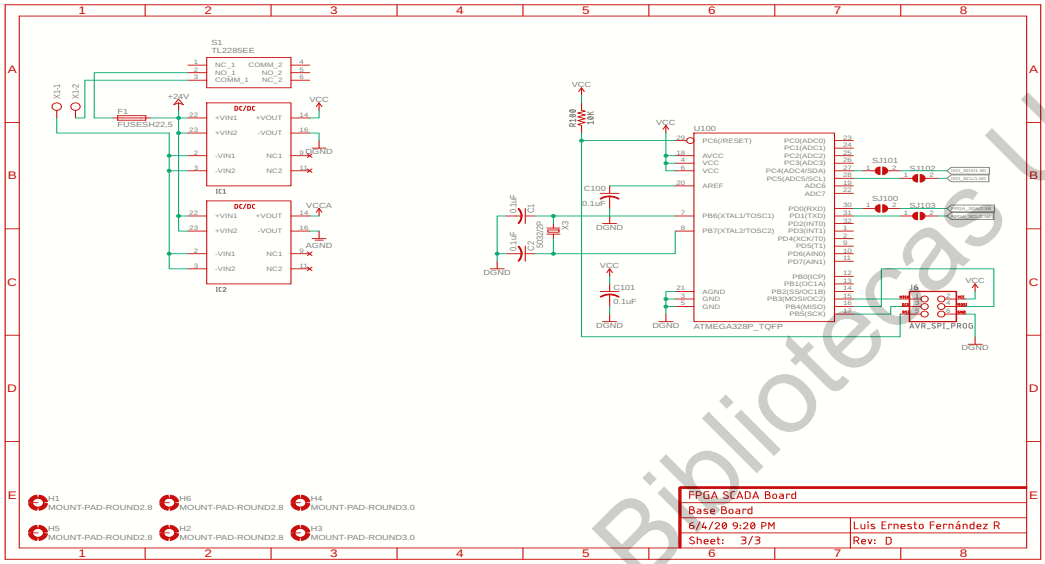


Figure A.8: Base Board Schematic Diagram

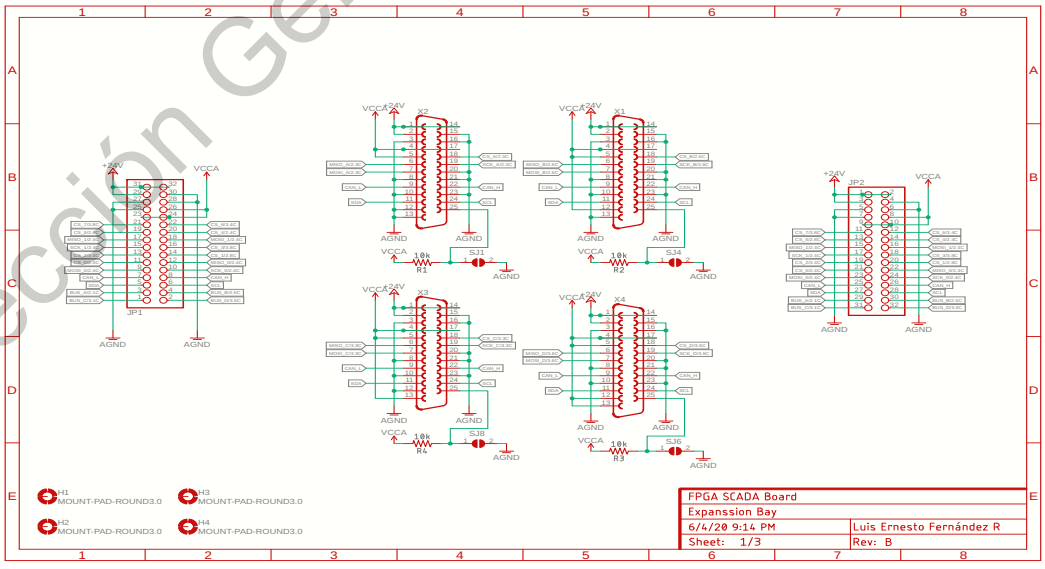
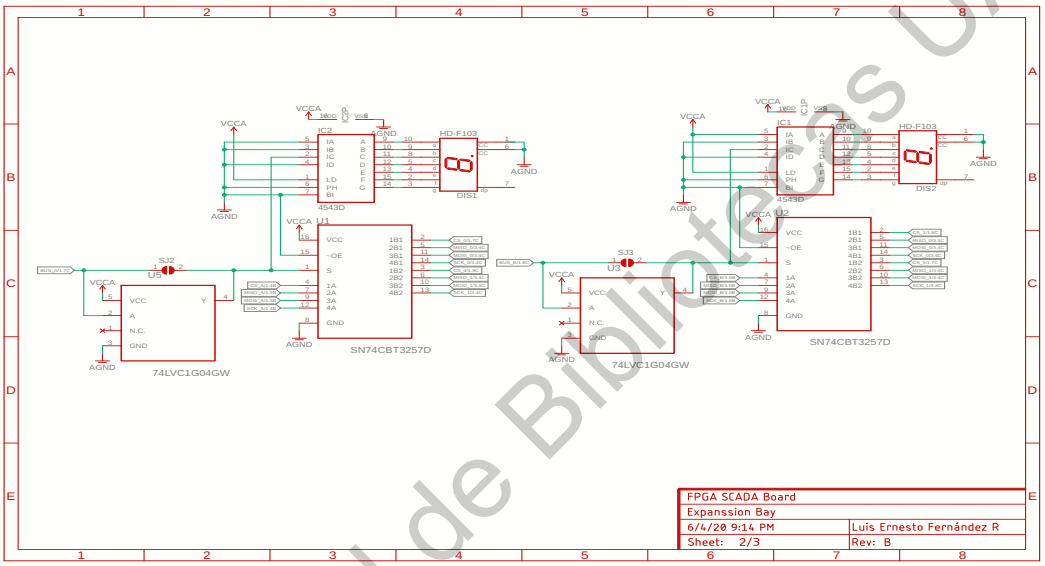


Figure A.9: Expansion Bay Schematic Diagram



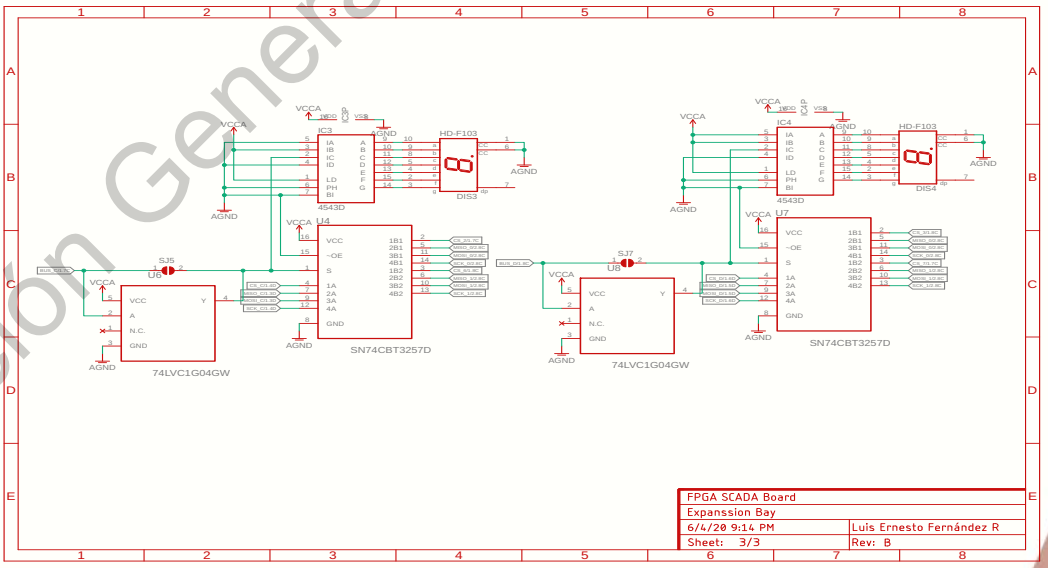
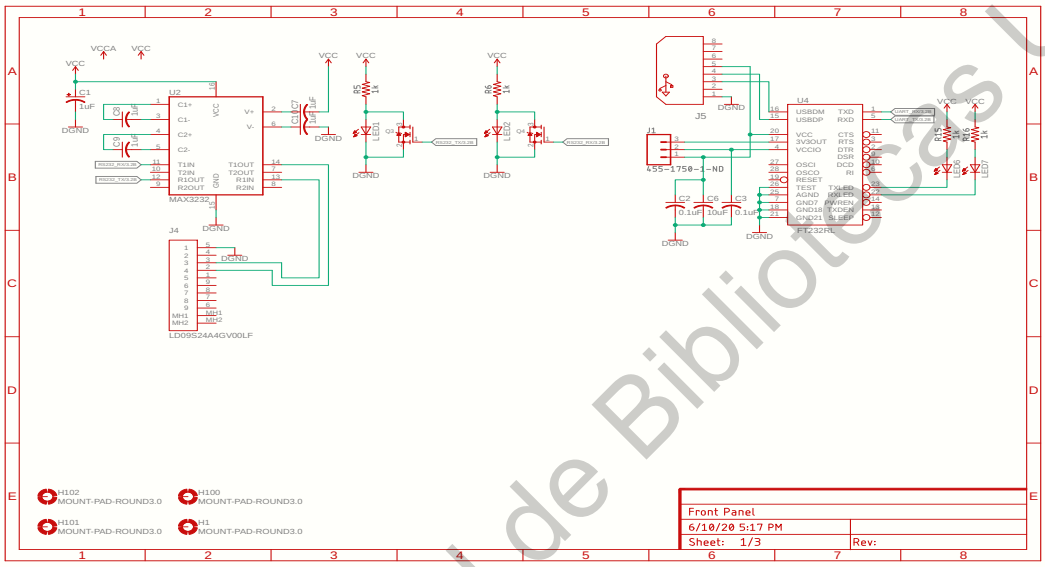


Figure A.9: Expansion Bay Schematic Diagram



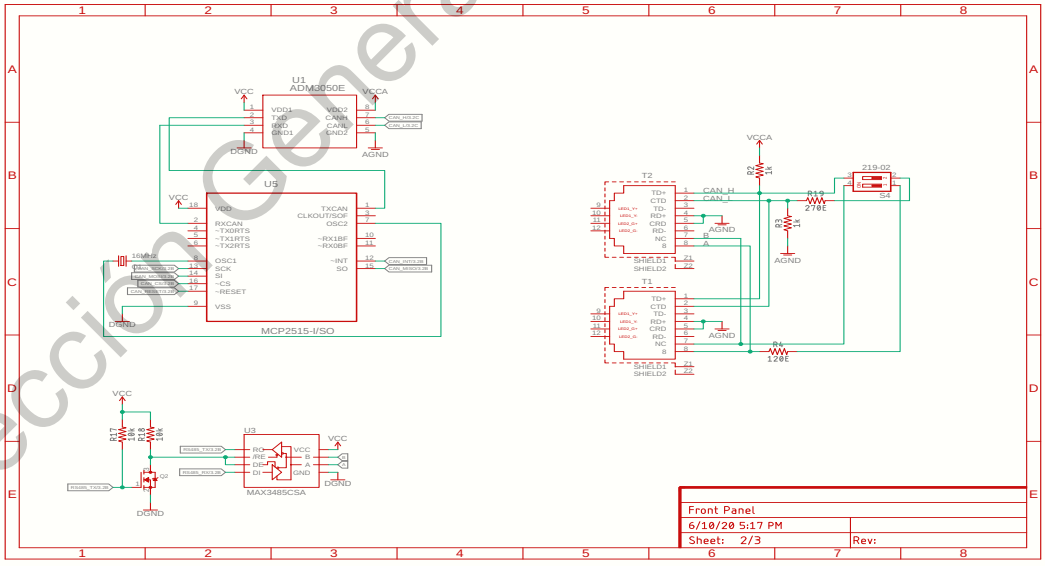


Figure A.10: Front Panel Schematic Diagram

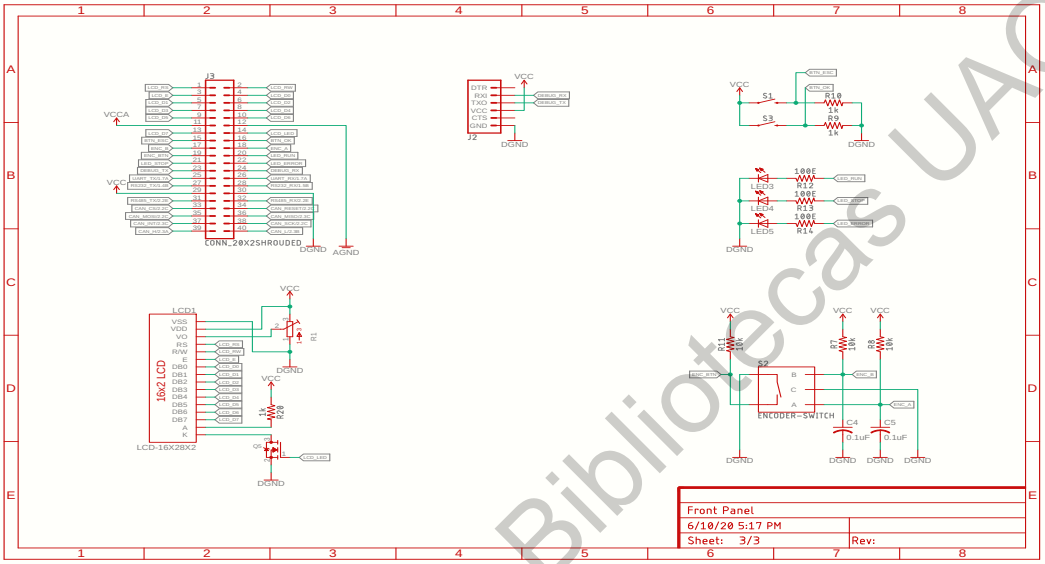


Figure A.10: Front Panel Schematic Diagram

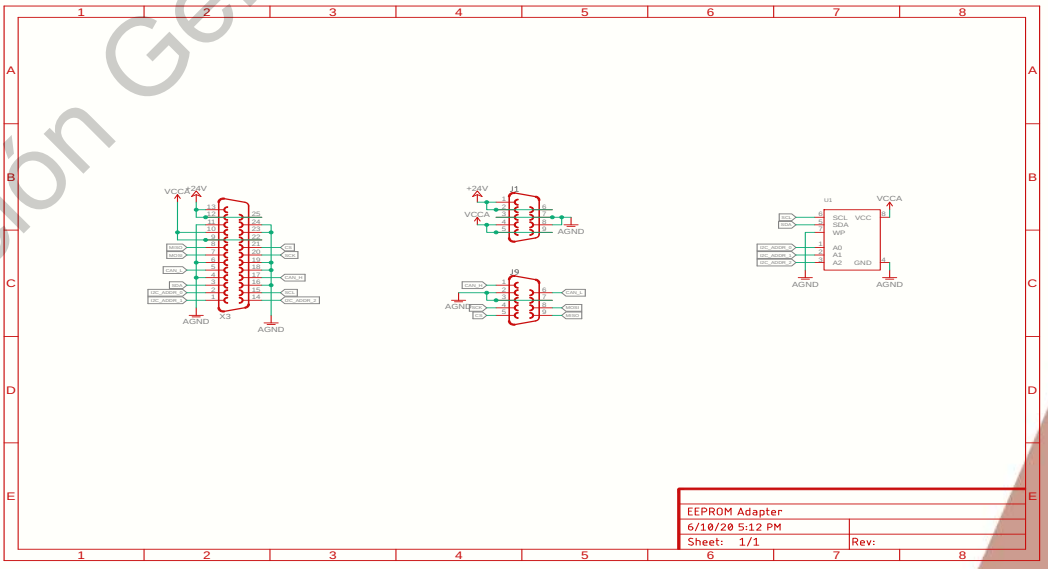


Figure A.11: EEPROM Adapter Schematic Diagram

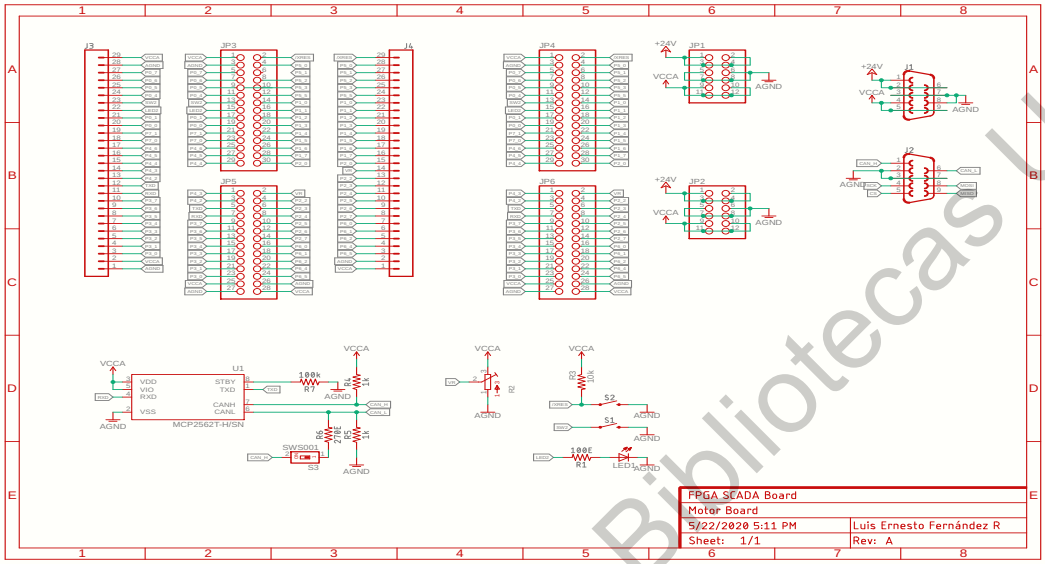


Figure A.12: Motor Controller Module Schematic Diagram

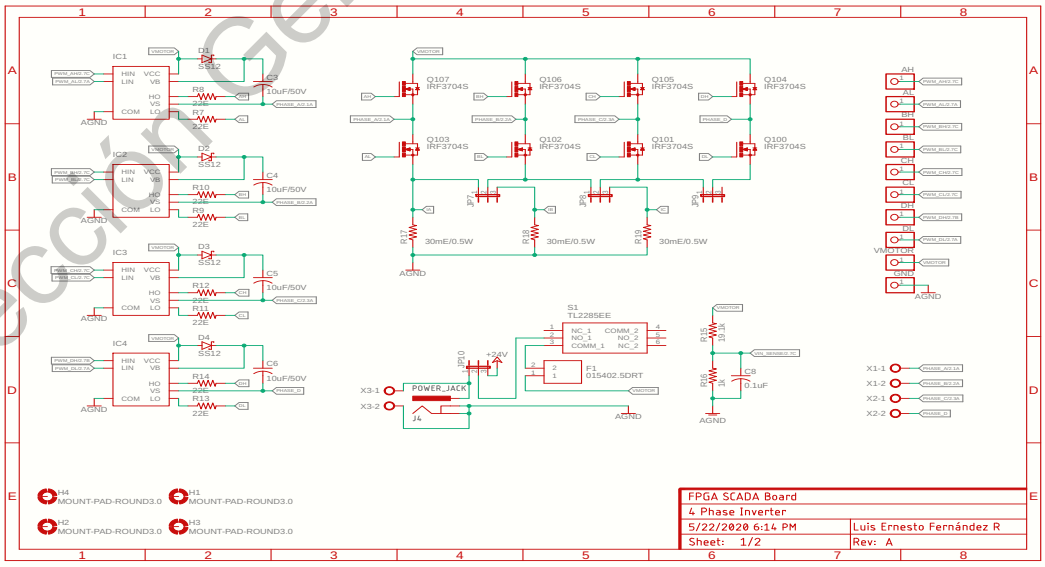
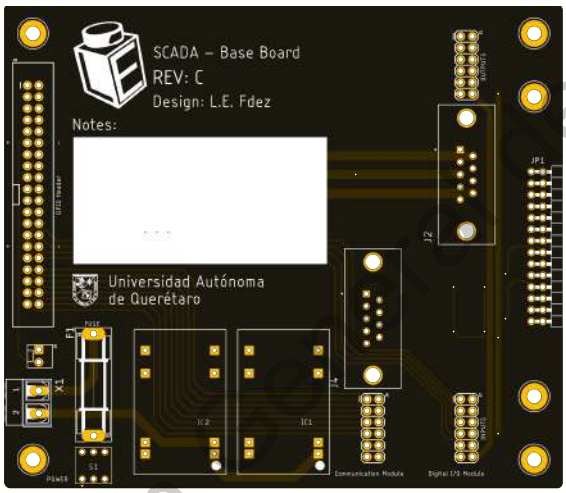


Figure A.13: 4 Phase Inverter Schematic Diagram

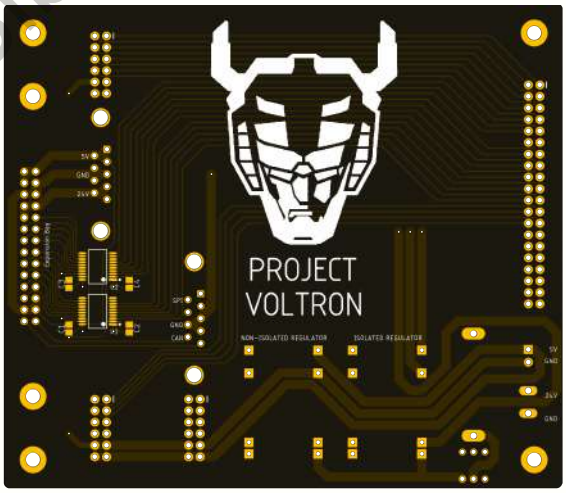
Appendix B

PCB Design

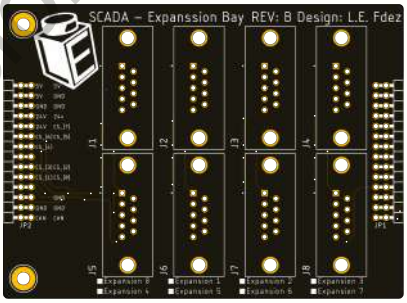
B.1 First Prototype



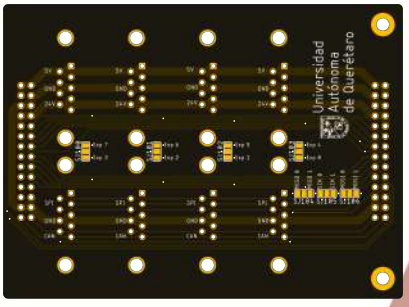
(a) Base Board Top



(b) Base Board Bottom

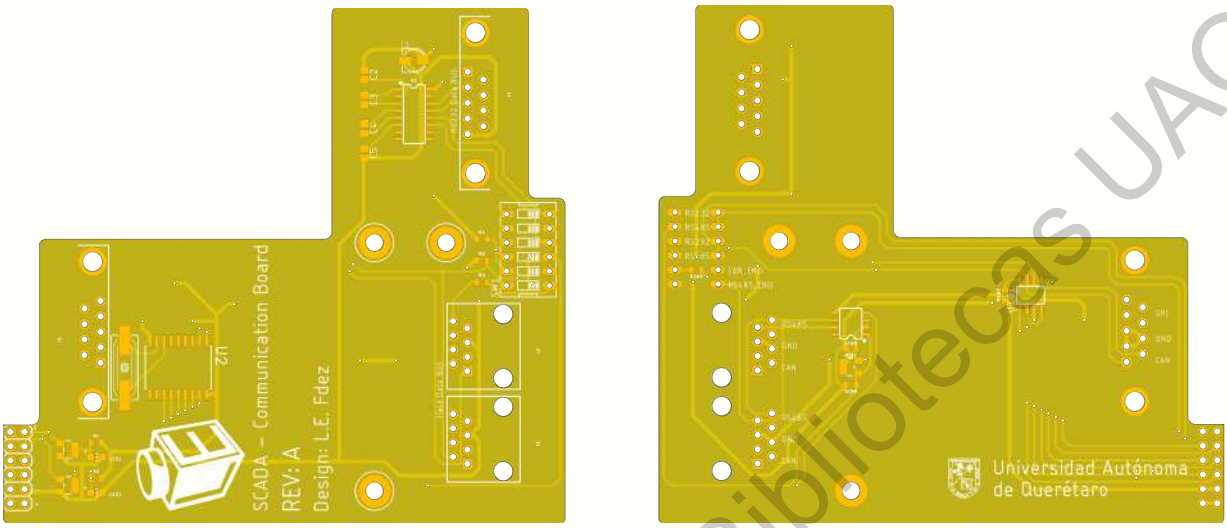


(c) Expansion Bay Top



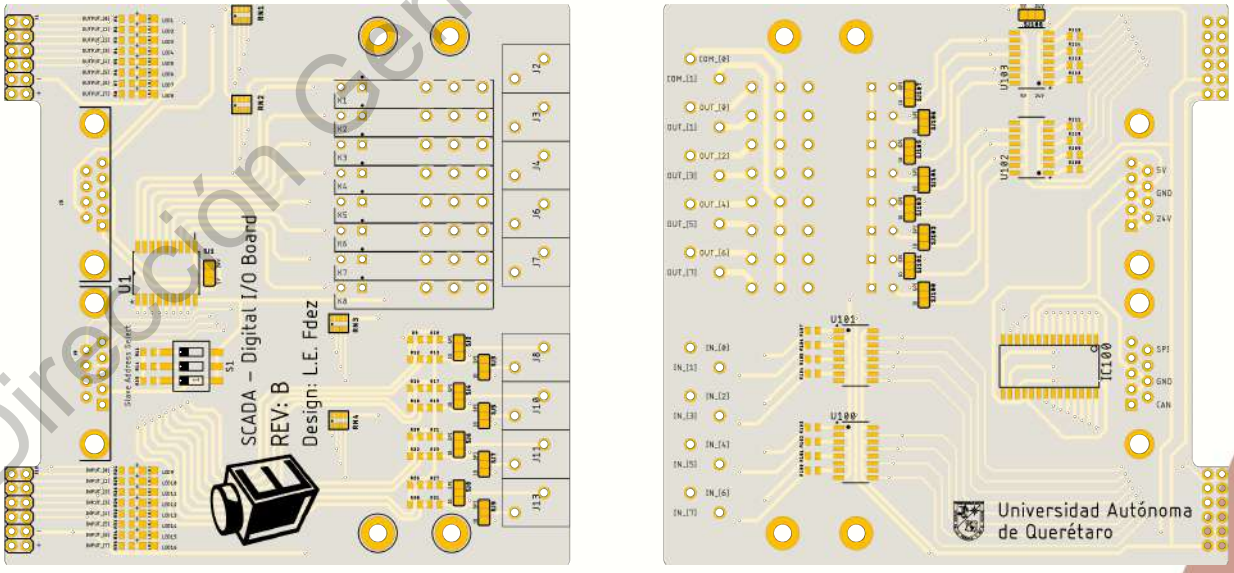
(d) Expansion Bay Bottom

Figure B.1: Interconnection Base Board PCB



(a) Top (b) Bottom

Figure B.2: Communication Module PCB

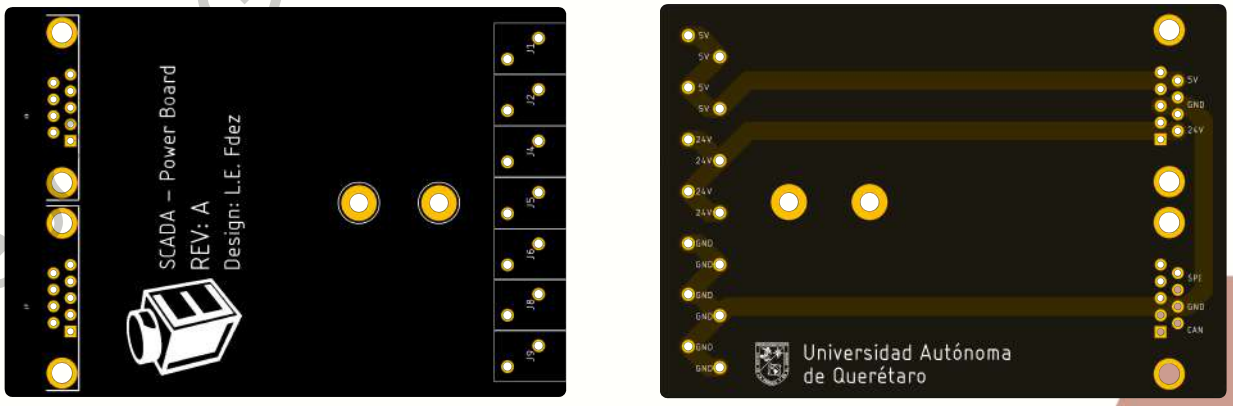


(a) Top (b) Bottom

Figure B.3: Digital I/O Module PCB

(a) Top (b) Bottom

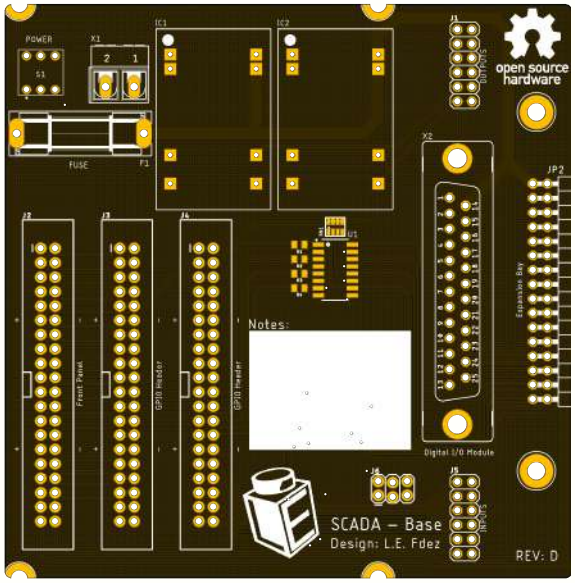
Figure B.4: Analog I/O Module PCB



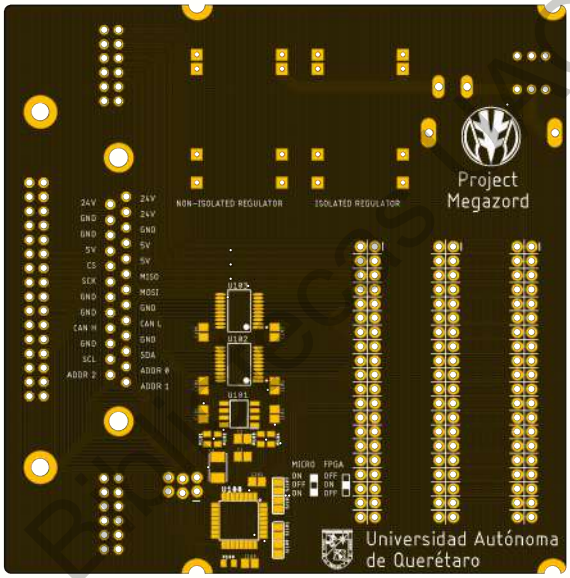
(a) Top (b) Bottom

Figure B.5: Power delivery module PCB

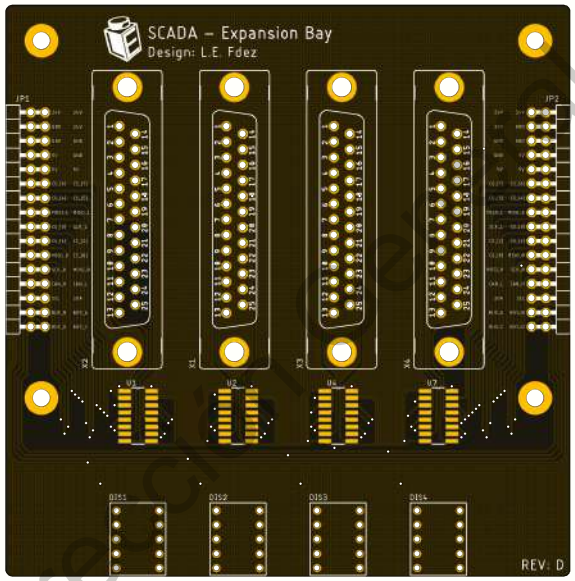
B.2 Second Prototype



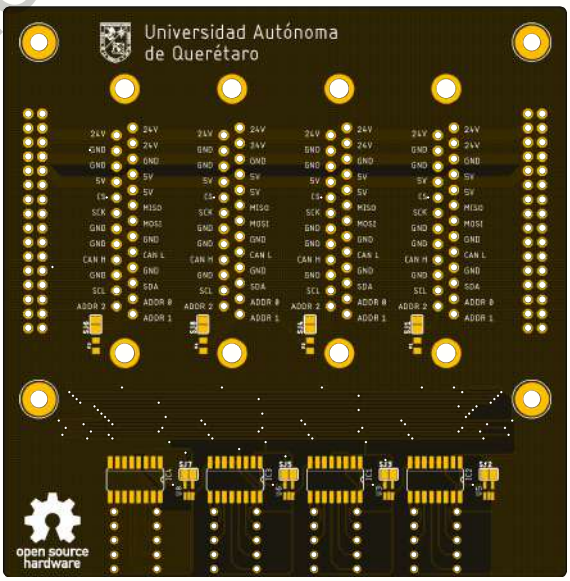
(a) Base Board Top



(b) Base Board Bottom

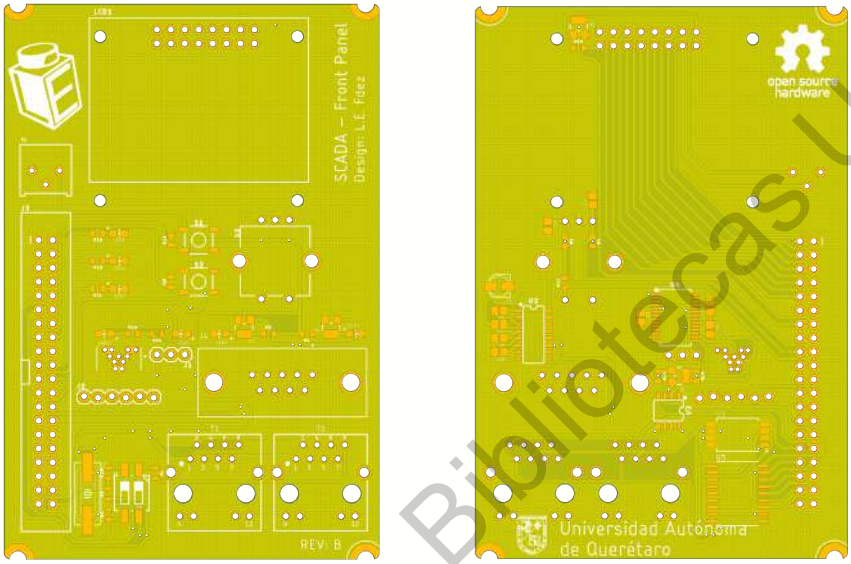


(c) Expansion Bay Top



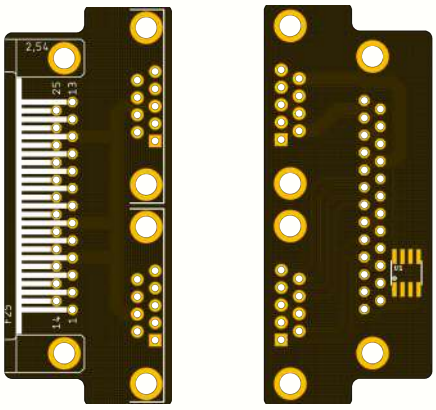
(d) Expansion Bay Bottom

Figure B.6: Interconnection Base Board PCB



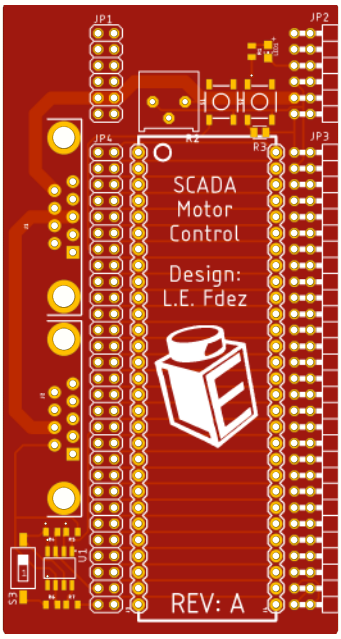
(a) Top (b) Bottom

Figure B.7: Front Panel PCB

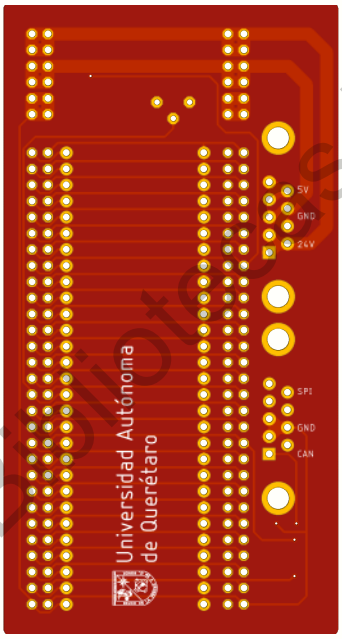


(a) Top (b) Bottom

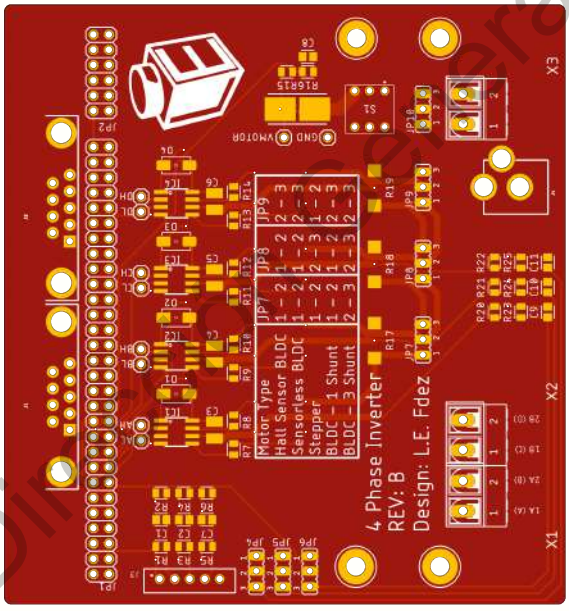
Figure B.8: EEPROM Adapter PCB



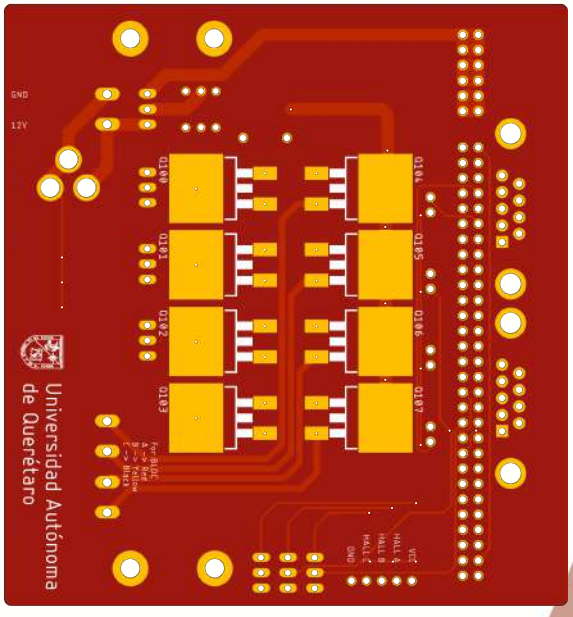
(a) Control Top



(b) Control Bottom



(c) 4-Phase Inverter Top



(d) 4-Phase Inverter Bottom PCB

Figure B.9: Motor control module PCB

Appendix C

Platform Designer Code

The following section contains the main codes and configuration for the NIOS II tests performed initially for development on the FPGA platform prior to the IP core design stage.

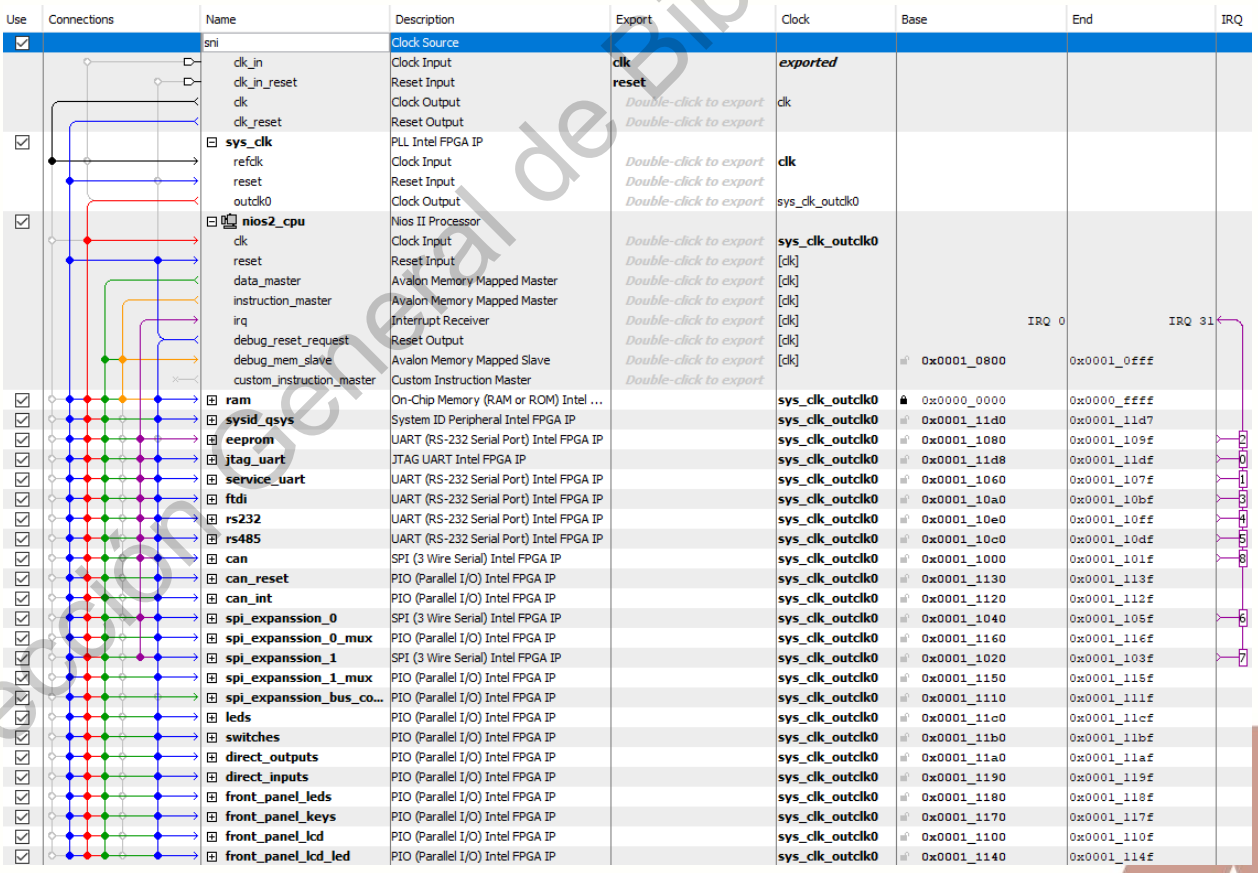


Figure C.1: Platform Designer Configuration

C.1 HDL top module for NIOS II processor test

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module nios2(
  ////////////// CLOCK //////////
  input                       CLOCK2_50,
  input                       CLOCK3_50,
  input                       CLOCK4_50,
  input                       CLOCK_50,
  ////////////// SDRAM //////////
  output         [12:0]     DRAM_ADDR,
  output          [1:0]     DRAM_BA,
  output                     DRAM_CAS_N,
  output                     DRAM_CKE,
  output                     DRAM_CLK,
  output                     DRAM_CS_N,
  inout          [15:0]     DRAM_DQ,
  output                     DRAM_LDQM,
  output                     DRAM_RAS_N,
  output                     DRAM_UDQM,
  output                     DRAM_WE_N,
  ////////////// KEY //////////
  input           [3:0]     KEY,
  ////////////// LED //////////
  output          [9:0]     LEDR,
  ////////////// SW //////////
  input           [9:0]     SW,
  ////////////// GPIO_0, GPIO_0 connect to GPIO Default //////////
  inout          [35:0]     GPIO_0,
  ////////////// GPIO_1, GPIO_1 connect to GPIO Default //////////
  inout          [35:0]     GPIO_1
);

//=======================================================
//  Structural coding
//=======================================================

    niosII u0 (
        .clk_clk          (CLOCK_50),          //      clk.clk
        .reset_reset_n    (KEY[0]),   //    reset.reset_n
        .switches_export  (SW[7:0]), // switches.export
        .leds_export      (LEDR[7:0]),       //     leds.export
        .spi_MISO         (GPIO_0[0]),         //      spi.MISO
        .spi_MOSI         (GPIO_0[1]),         //          .MOSI
        .spi_SCLK         (GPIO_0[2]),         //          .SCLK
        .spi_SS_n         (GPIO_0[3])          //          .SS_n
    );
endmodule
```

C.2 Example test code for NIOS II processor

```
/*
 * "Small Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It requires a STDOUT  device in your system's hardware.
 *
 * The purpose of this example is to demonstrate the smallest possible Hello
 * World application, using the Nios II HAL library.  The memory footprint
 * of this hosted application is ~332 bytes by default using the standard
 * reference design.  For a more fully featured Hello World application
 * example, see the example titled "Hello World".
 *
 * The memory footprint of this example has been reduced by making the
 * following changes to the normal "Hello World" example.
 * Check in the Nios II Software Developers Manual for a more complete
 * description.
 */

#include "sys/alt_stdio.h"
#include "system.h"
#include "altera_avalon_pio_regs.h"

void delay(int time) {
  volatile int i =0;
  while(i<time*50000){
    i++;
  }
}

int main() {
  alt_putstr("Hello from Nios II!\n");
  char count = 0;
  char select;
  char led_on = 0;

  /* Event loop never exits. */
  while (1) {
      select = IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE)&0xFF;
      IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, select);
      /*if(select == 1) {
        led_on ^= 0x0F;
      }else if(select == 2) {
        led_on ^= (0x0F<<4);
      }*/
      count++;
      delay(10);
  }
  return 0;
}
```

Appendix D

ATmega 328p test Code

D.1 Digital I/O Test

```
//Test for the MCP23S17 16-Bit I/O Expander
#include <SPI.h>

SPISettings portExpanderSettings(16000000, MSBFIRST, SPI_MODE0);

const int PORT_EXPANDER_SS_PIN = 7;
const uint8_t PORT_EXPANDER_ADDRESS = 0;
const uint8_t SLAVE_CONTROL_BYTE = 0b1000000 | (PORT_EXPANDER_ADDRESS << 1);

#define     IOCON     (0x0A)
#define     IODIRA    (0x00)
#define     IODIRB    (0x01)
#define     IOPOLA    (0x02)
#define     IOPOLB    (0x03)
#define     GPIOA     (0x12)
#define     GPIOB     (0x13)

uint8_t INPUT_PIN_1 = 1; // GPA1
uint8_t INPUT_PIN_2 = 2; // GPA2
uint8_t INPUT_PIN_3 = 3; // GPA3
uint8_t INPUT_PIN_4 = 4; // GPA4
uint8_t INPUT_PIN_5 = 5; // GPA5
uint8_t INPUT_PIN_6 = 6; // GPA6
uint8_t INPUT_PIN_7 = 7; // GPA7
uint8_t INPUT_PIN_8 = 8; // GPA8

uint8_t OUTPUT_PIN_1 = 1; // GPB1
uint8_t OUTPUT_PIN_2 = 2; // GPB2
uint8_t OUTPUT_PIN_3 = 3; // GPB3
uint8_t OUTPUT_PIN_4 = 4; // GPB4
uint8_t OUTPUT_PIN_5 = 5; // GPB5
uint8_t OUTPUT_PIN_6 = 6; // GPB6
uint8_t OUTPUT_PIN_7 = 7; // GPB7
uint8_t OUTPUT_PIN_8 = 8; // GPB8

```

```
uint8_t GPIOB_value = 0x00;
uint8_t GPIOA_value = 0x00;


//Command: setup SPI, ports and interrupts.
void setup() {
  pinMode(PORT_EXPANDER_SS_PIN, OUTPUT);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  SPI.begin();
  SPI.beginTransaction(portExpanderSettings);

  writeByte(IOCON,  0b00001000); // enable hardware address pins; bank=0
    addressing
  writeByte(IODIRA, 0xFF);// set input ports
  writeByte(IODIRB, 0x00);// set output ports
}


void loop() {
  //test_outputs();
  test_inputs();
  delay(500);
}


void test_inputs(){
  GPIOA_value = readByte(GPIOA);
  writeByte(GPIOB, GPIOA_value);
}


void test_outputs(){
  writeByte(GPIOB, GPIOB_value);
  GPIOB_value = GPIOB_value<<1;
  if (!GPIOB_value) GPIOB_value = 0x01;
}


//Command: write a single byte to the specified register
void writeByte(uint8_t reg, uint8_t data) {
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  SPI.transfer(SLAVE_CONTROL_BYTE);
  SPI.transfer(reg);
  SPI.transfer(data);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
}


//Returns: byte read from specified register
uint8_t readByte(uint8_t reg) {
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  SPI.transfer(SLAVE_CONTROL_BYTE | 1);
  SPI.transfer(reg);
  uint8_t data = SPI.transfer(0);
```

```
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  return data;
}


//Command: write two bytes to specified register
//demonstrates sequential write and transfer16 alternate SPI method.
void writeSequentialBytes(uint8_t reg, uint8_t first, uint8_t last) {
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  SPI.transfer16((uint16_t)SLAVE_CONTROL_BYTE << 8 | reg);
  SPI.transfer16((uint16_t)first << 8 | last);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
}
```

D.2 DAC Test

```
//Test for the MCP4822 12-Bit ADC

#include <SPI.h>

SPISettings portExpanderSettings(16000000, MSBFIRST, SPI_MODE0);

const int PORT_EXPANDER_SS_PIN = 10;
const float SUPPLY_VOLTAGE = 4570;
int voltage = 100;

void setup() {
  pinMode(PORT_EXPANDER_SS_PIN, OUTPUT);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  SPI.begin();
  SPI.beginTransaction(portExpanderSettings);
}

void loop() {
  writeDAC(0, voltageConvert(voltage));
  writeDAC(1, voltageConvert(voltage*2));
  voltage = voltage + 100;
  if(voltage*2 > SUPPLY_VOLTAGE){
    voltage = 100;
  }
}

uint16_t voltageConvert(float voltage){
  return voltage*4095/SUPPLY_VOLTAGE;
}

//Command: write dac value to selected output
void writeDAC(uint8_t channel, uint16_t value){
  uint8_t dataOut = 0;
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  dataOut = (channel == 0) ? 0x10 : 0x90;
  dataOut = dataOut ^ (value >> 8);
```

```
  SPI.transfer(dataOut);
  SPI.transfer(value & 0x0FF);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
}
```

D.3 ADC Test

```
//Test for the MCP3202 12-Bit ADC

#include <SPI.h>

SPISettings portExpanderSettings(16000000, MSBFIRST, SPI_MODE0);

const int PORT_EXPANDER_SS_PIN = 10;
float channel1 = 0;
float channel2 = 0;

void setup() {
  pinMode(PORT_EXPANDER_SS_PIN, OUTPUT);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  SPI.begin();
  SPI.beginTransaction(portExpanderSettings);
  Serial.begin(115200);
}

void loop() {
  channel1 = readADC(0);
  channel2 = readADC(1);
  Serial.print(channel1);
  Serial.print("\t");
  Serial.println(channel2);
}

//Command: request ADC data from selected channel
uint16_t readADC(uint8_t channel){
  uint8_t dataIn = 0;
  uint8_t result = 0;
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  uint8_t dataOut = 0x01;
  dataIn = SPI.transfer(dataOut);
  dataOut = (channel == 0) ? 0xA0 : 0xE0;
  dataIn = SPI.transfer(dataOut);
  result = dataIn & 0x0F;
  dataIn = SPI.transfer(0x00);
  result = result << 8;
  result = result | dataIn;
  //input = input << 1;
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  return result;
}
```

Appendix E

Financial Statement

The following table E.1 includes an estimate of the budget invested on the project up to date.

Table E.1: Financial estimate.

Qty	Description	Subtotal (USD)	Total (USD)
1	DE10-Nano Board	$ 110.00	$ 110.00
1	CY8CKIT-037 DEV KIT MOTOR CONTROL PSOC 4	$ 186.56	$ 186.56
1	CY8CKIT-042 PIONEER CY8C4245AXI EVAL BRD	$ 28.13	$ 28.13
2	CANDIY-SHIELD V2 - CAN-BUS Shield	$ 14.14	$ 28.28
1	SMT ADAPTERS 3 PACK 28SOIC/TSSOP	$ 4.95	$ 4.95
1	SMT ADAPTERS 3 PACK 20SOIC/TSSOP	$ 4.50	$ 4.50
1	SMT ADAP 6 PACK 8SOIC/MSOP/TSSOP	$ 2.95	$ 2.95
6	MCP23S17T-E/SOCT-ND	$ 1.30	$ 7.80
3	ADUM3154BRSZ-ND	$ 6.75	$ 20.25
1	Traco Power / TEL 5-2411 5V 5W	$ 18.30	$ 18.30
1	Traco Power / TEL 5-2410 3.3V 4W	$ 18.30	$ 18.30
3	MCP3202-CI/SN-ND	$ 2.61	$ 7.83
4	MCP4822-E/SN-ND	$ 3.09	$ 12.36
2	MCP2515T-I/SOCT-ND	$ 1.87	$ 3.74
4	MCP2562T-E/SNCT-ND	$ 0.93	$ 3.72
2	TC7MBL3253CFT	$ 0.625	$ 1.25
20	IRFS3607TRLPBF	$ 0.49697	$ 9.94
10	IR2102SPBF	$ 1.786364	$ 17.86
12	ULN2803A	$ 0.132114	$ 1.59
5	MAX3232CSE+T	$ 1.1077	$ 5.54
30	2N7002LT1G	$ 0.0592	$ 1.78
5	MAX3485ESA+T	$ 1.632	$ 8.16
10	TCMT4600	$ 1.0299	$ 10.3
16	HF41F/5-ZS	$ 1.069697	$ 17.12
8	K8-8081D-L1	$ 0.15303	$ 1.23
4	015402.5DRT	$ 1.219697	$ 4.88
2	DP-06BP	$ 0.3985	$ 0.8
10	HAD-03LWA-R	$0.2197	$ 2.20

50	TS5215A 250gf	$ 0.0138	$ 0.69
2	DSIC01LSGET	$ 0.386364	$ 0.77
30	DS1037-09FNAKT74-0CC	$ 0.166667	$ 5.00
40	DS1034-09MUNSi44	$ 0.209091	$ 8.36
20	DS1027-2LBF1	$ 0.02245	$ 0.45
10	DC-005-20A	$ 0.081013	$ 0.81
20	Test Ring 5001	$ 0.145455	$ 2.91
2	B5B-EH(LF)(SN)	$ 0.12368	$ 0.25
20	JL2EDGR-50802G01	$ 0.028367	$ 0.57
20	JL2EDGK-50802G01	$ 0.095451	$ 1.91
10	C35445	$ 0.0479	$ 0.48
16	C124415	$ 0.0663	$ 1.07
20	C124416	$ 0.0902	$ 1.81
5	C132435	$ 0.0845	$ 0.43
10	A2541HWR-2x7P	$ 0.2045	$ 2.05
2	FH1-200CK-G	$ 0.1939	$ 0.39
10	DS1128-04-S8B8P-X	$ 0.3907	$ 3.91
10	C132439	$ 0.2029	$ 2.03
52	1-2834011-2	$ 0.381	$ 19.81
100	0805W8F1822T5E	$ 0.002797	$ 0.28
100	0805W8F1912T5E	$ 0.001684	$ 0.17
100	0805W8F2200T5E	$ 0.001524	$ 0.15
100	0805W8F4700T5E	$ 0.001681	$ 0.17
20	0805X106K250NT	$ 0.035928	$ 0.72
20	0805F104M500	$ 0.029385	$ 0.59
60	D-G080508G1-KS2	$ 0.016163	$ 0.97
40	D-B080508B1-KS2	$ 0.016163	$ 0.65
100	WR08X1002FTL	$ 0.005438	$ 0.54
100	AC0805FR-072KL	$ 0.002988	$ 0.30
20	D-W080508W1-KS2	$ 0.026702	$ 0.53
100	RS-05K3901FT	$ 0.001693	$ 0.17
50	CRA034RF1K00P05Z	$ 0.008652	$ 0.43
100	CR0805F3K00P05Z	$ 0.001936	$ 0.19
100	WR08X22R0FTL	$0.001554	$ 0.16
50	0805N331J500CT	$ 0.010931	$ 0.55
100	ERJP06F1001V	$ 0.023739	$ 2.37
10	RTT25R030FTE	$ 0.057	$ 0.57
20	US1AF	$ 0.0147	$ 0.3
50	RC1210JR-0722RL	$ 0.0067	$ 0.34
10	9C16000132	$ 0.1131	$ 1.14
50	D-R080508L3-KS2	$ 0.0135	$ 0.68
40	D-W080508W1-KS2	$ 0.0242	$ 0.97
50	AF0805JR-07100KL	$ 0.0051	$ 0.26
50	AF0805JR-07100RL	$ 0.0049	$ 0.25
20	0805B104J500NT	$ 0.0258	$ 0.52

20	0805F105M500NT	$ 0.0254	$0.51
10	0805B475K250NT	$ 0.0657	$ 0.66
10	0805F106M250NT	$ 0.0609	$ 0.61
50	RC0805JR-071KL	$ 0.0027	$ 0.14
60	D-G080508G1-KS2	$ 0.0203	$ 1.22
60	D-B080508B1-KS2	$ 0.022	$ 1.32
5	3386P-1-103TLF	$ 1.2498	$ 6.25
50	AC0805JR-0710KL	$ 0.0131	$ 0.66
100	AF0805JR-07100RL	$ 0.0104	$ 1.04
20	CRA064RJ1KE04	$ 0.0602	$ 1.21
50	AC0805JR-07270RL	$ 0.0047	$ 0.24
5	Motor Board PCB Production	$ 4.00	$ 4.00
5	IO Board PCB Production	$ 4.00	$ 4.00
5	4 Phase Inverter PCB Production	$ 4.00	$ 4.00
5	Power Board PCB Production	$ 2.00	$ 2.00
5	Base Board PCB Production	$ 8.50	8.50
5	Expansion Bay PCB Production	$ 4.00	$ 4.00
5	FPGA Shield Adapter PCB Production	$ 6.70	$ 6.70
5	HMI Board PCB Production	$ 2.00	$ 2.00
5	Comunication Board PCB Production	$ 2.00	$ 2.00
5	Analog Board PCB Production	$ 2.00	$ 2.00
5	Pi Adapter Board PCB Production	$ 4.00	$ 4.00
		Total	**$ 664.72**

Bibliography

[1] BEng, D. B. *Practical SCADA for Industry (IDC Technology)*. Newnes, sep 2003.

[2] Cerrada, Mariela Cardillo, J. P. A. Diagnóstico de fallas basado en modelos: Una solución factible para el desarrollo de aplicaciones SCADA en tiempo real. *Ciencia e Ingeniería* (2011).

[3] Eldijk, V. IEC 61131: General information, Apr 2018.

[4] Muhammad Aamir and Muhammad Aslam Uqaili and Nishat Ahmad Khan and Javier Poncela and B. S. Chowdhry. Hardware Implementation and Testing of Reconfigurable RTU for Wireless SCADA. *Wireless Personal Communications 85*, 2 (May 2015), 511–528.

[5] M. Chmiel and R. Czerwinski and P. Smolarek. IEC 61131-3-based PLC Implemented by means of FPGA. *IFAC-PapersOnLine 48*, 4 (2015), 374–379.

[6] Robert Czerwinski and Miroslaw Chmiel and Wojciech Wygrabek. FPGA Implementation of Programmable Logic Controller Compliant with EN 61131-3. *IFAC Proceedings Volumes 46*, 28 (2013), 138–143.

[7] Dong-Ah Lee and Eui-sub Kim and Junbeom Yoo and Jang-Soo Lee and Jong Gyun Choi. FBDtoVerilog 2.0: An Automatic Translation of FBD into Verilog to Develop FPGA. In *2014 International Conference on Information Science & Applications (ICISA)* (May 2014), IEEE.

[8] Adam Milik. High level synthesis - reconfigurable hardware implementation of programmable logic controller. *IFAC Proceedings Volumes 39*, 21 (Feb. 2006), 138–143.

[9] Adam Milik. Multiple-Core PLC CPU Implementation and Programming. *Journal of Circuits, Systems and Computers 27*, 10 (May 2018), 1850162.

[10] Muñoz Barrón, Benigno. Controlador modular y reconfigurable para máquina de inyección de plástico basado en FPGA, Jun 2011.

[11] Osornio Rios, Roque Alfredo. Diseño de sistema de control para CNC de alta velocidad, Aug 2007.

[12] Sánchez Gómez, Jesús Iván. Desarrollo de compilador para lenguaje escalera de controladores lógicos programables para aplicaciones industriales, Jun 2013.

[13] Santos Cruz, Rafael. Diseño e implementación de teclado industrial aplicado a una máquina de inyección de plástico, Aug 2008.

[14] Soroush Shirali and Shahab Ensafi and Mahsa Naseri. RTU hardware design for SCADA systems using FPGA. In *2010 International Conference on Computer Applications and Industrial Electronics* (Dec. 2010), IEEE.

[15] CELSO F. SILVA AND CAMILO QUINTANS AND JOSE M. LAGO AND ENRIQUE MANDADO. An Integrated System for Logic Controller Implementation Using FPGAs. In *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics* (Nov. 2006), IEEE.

[16] C. F. SILVA AND C. QUINTANS AND E. MANDADO AND M. A. CASTRO. Methodology to Implement Logic Controllers with both Reconfigurable and Programmable Hardware. In *2007 IEEE International Symposium on Industrial Electronics* (June 2007), IEEE.

[17] MILIK, A., AND PULKA, A. On FPGA dedicated SFC synthesis and implementation according to IEC61131. In *2014 International Conference on Signals and Electronic Systems (ICSES)* (Sept. 2014), IEEE.

[18] OLDFIELD, J. V. *Field-Programmable Gate Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems*. Wiley-Interscience, jan 1995.

[19] PENIN, A. *Sistemas SCADA*. Alfaomega Marcombo, Mexico, 2013.

[20] SARDUY, J. *Temas especiales de instrumentacion y control*. Editorial Felix Varela, La Habana, 2005.

[21] TRIMBERGER, S. M. *Field-Programmable Gate Array Technology*. Springer, jan 1994.