Development of CompactRIO based PID Temperature Controller

Thesis submitted in partial fulfillment of the requirement for the award of degree of

Master of Engineering

in

Electronics Instrumentation and Control

By:
Navneet Kaur
(80751016)

Under the supervision of:
Dr. Mandeep Singh
Assistant Professor
E&I Deptt.

JUNE 2009

ELECTRICAL AND INSTRUMENTATION DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

Declaration

*I hereby declare that the report entitled "**Development of CompactRIO based PID Temperature Controller**" is an authentic record of my own work carried out as requirements for the award of degree of M.E. (Electronic Instrumentation & Control) at Thapar University, Patiala, under the guidance of Dr. Mandeep Singh (AP, EIED) during January to June 2009.*

Navneet

NAVNEET KAUR

Roll. No. 80751016

Date: 18/06/09

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

(----------)

Dr. Mandeep Singh
Assistant Professor, EIED
Thapar University, Patiala.

S. Ghosh 07/07/09

(----------)

Dr. Smarajit Ghosh
Assistant Professor & Head EIED
Thapar University, Patiala.

(----------)

Dr. R. K. Sharma
Dean of Academic Affairs
Thapar University, Patiala.

Acknowledgment

The real spirit of achieving a goal is through the way of excellence and austere discipline. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various personalities.

With deep sense of gratitude I express my sincere thanks to my esteemed and worthy supervisor, Dr. Mandeep Singh, Assistant Professor, Department of Electrical & Instrumentation Engineering, Thapar University, Patiala, for his valuable guidance in carrying out this work under his effective supervision, encouragement, enlightenment and cooperation.

I shall be failing in my duties if I do not express my deep sense of gratitude towards Dr. Smarajit Ghosh, A.P. & Head of the Department of Electrical & Instrumentation Engineering, Thapar University, Patiala who has been a constant source of inspiration for me throughout this work.

I am also thankful to all the staff members of the Department for their full cooperation and help.

Acknowledgement is due, to UGC Assistance for Strengthening Virtual Instrumentation as Special Paper in M.E. at Department of Electrical and Instrumentation, Thapar University, Patiala under innovative program of teaching and research in inter-disciplinary and emerging area, for providing the financial help for purchase of the necessary equipment and software.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the ALMIGHTY who bestowed self-confidence, ability and strength in me to complete this work.

Navneet

Navneet Kaur

Place: Thapar University, Patiala

Date: 18/06/09

Abstract

During the last 10 years, most of the technological advancements have occurred in the field of high volume products like mobile phones, DVD players, microwave ovens etc. New York Times stated that “On average an American comes in contact with sixty microprocessors in a day”. All this is possible because of the progress made in the field of embedded systems.

Field Programmable Gate Array (FPGA), being the latest venture in the field of embedded electronics and the most successful configuration for embedded systems, is in great demand these days. CompactRIO, a powerful control and acquisition system introduced by National Instruments, is based upon reconfigurable FPGA technology. Apart from FPGA, it also incorporates a real-time processor for reliable stand alone applications. Now instrumentation engineers can also benefit from the FPGA technology without any knowledge of hardware description languages, because the CompactRIO systems can be developed using LabVIEW development platform.

In this work, we have developed a temperature controlled chamber using the NI CompactRIO. The chamber has been fabricated using the acrylic sheets. Two 200W bulbs have been used for heating purpose and a thermocouple for temperature sensing. A small CPU fan has been used for cooling and only comes into action during emergency situation. A thermocouple input module (NI 9211) has been used to acquire data from thermocouple and a digital output module (NI 9474) has been used to drive the bulbs (via a relay card) and the fan. PID control algorithm has been used for our application and is implemented in LabVIEW.

Organization of Thesis

Apart from introduction about embedded systems, the first chapter provides a detailed look into Field Programmable Gate Array (FPGA) technology as well as a comparative study of FPGA with microcontrollers and Application Specific Integrated Circuit (ASIC). In addition, a brief overview of National Instruments CompactRIO (an FPGA based control and acquisition system) is given.

In the second chapter, various practical implementations of NI CompactRIO, which has been carried out till date, are given.

The third chapter is dedicated to problem definition and a brief overview of the recommended solution.

The fourth chapter includes the configuration of the NI CompactRIO system. Various I/O modules have also been discussed in this chapter.

In the fifth chapter, complete solution including the hardware setup and the software development, is discussed. In addition, results and various problems that we encountered during our project work are also included in this chapter.

Lastly, in the sixth chapter we have concluded the whole work and also provided some suggestions for future betterment.

Table of Contents

List of figures

List of Tables

List of abbreviations

PC	Personal Computer
GPS	Global Positioning System
DSP	Digital Signal Processing
LED	Light Emitting Diode
NRE	Non-Recurring Engineering
CPU	Central Processing Unit
IC	Integrated Circuit
RAM	Random Access Memory
ROM	Read Only Memory
I/O	Input/Output
PDA	Personal Digital Assistant
DMA	Direct Memory Access
ASIC	Application Specific Integrated Circuit
HDL	Hardware Description Language
FPGA	Field Programmable Gate Array
LUT	Look up Table
SRAM	Static RAM
EEPROM	Electrically Erasable Programmable ROM
PROM	Programmable ROM
NI	National Instruments
RIO	Reconfigurable I/O
cRIO	CompactRIO

LabVIEW	Laboratory Virtual Instrument Engineering Workbench
PCI	Peripheral Component Interconnect
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
PID	Proportional-Integral-Derivative
ECU	Engine Control Unit
SHM	Structural Health Monitoring
PI	Proportional-Integral
PWM	Pulse Width Modulation
POST	Power-On Self Test
VI	Virtual Instrument
RT	Real Time
SMB	Subminiature Version-B
IP	Internet Protocol
MAX	Measurement and Automation Explorer
DSUB	D-Subminiature
DIO	Digital I/O
AO	Analog Output
AI	Analog Input
DO	Digital Output

Chapter 1

Embedded Systems: Technologies and hardware

1.1 Introduction

When the era of computers started, they were sometimes referred to as special purpose machines i.e. dedicated to only single task, but were far too large and expensive for most kinds of tasks performed by embedded computers of today's era. However, over the time, the concept of programmable controllers evolved from traditional electromechanical sequencers to the use of computer technology.

Over the time, embedded systems have come down in price and experienced a significant rise in processing power and functionality. For example, Intel 4004, the first microprocessor, was designed for calculators and other small systems but required a large amount of external memory and support chips.

Due to the reduced cost of microprocessors and microcontrollers, it seemed feasible to replace expensive analog components such as potentiometers with up-down buttons or knobs that a microprocessor could read out. By mid 80s, most of the common external system components had been integrated into the same chip as the processor and this modified form of microcontroller resulted in more widespread use, which by the end of the decade did not remain an exception.

This integration of microcontrollers also resulted in increase in application areas for embedded systems; where earlier a computer was not into consideration. A general purpose and a low-cost microcontroller is often programmed to perform the same task as performed by a large number of separate components. Though in this scenario, an embedded system is more complex than a traditional solution, most of the complexity is within the microcontroller only. Only few additional components may be required and most of the design effort is in the software.

1.2 Embedded systems

1.2.1 Definition

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions [1], often with real-time computing constraints. The system is usually *embedded* as a part rather than a complete device, including hardware and mechanical parts. On the other hand, a general-purpose computer, such as a Personal Computer (PC), can do many different tasks depending on the programming. Embedded systems control many of the common devices used in today's era.

Because the embedded system is special purpose, design engineers can always optimize it, thereby, reducing the size as well as cost of the product, and increasing the reliability and performance.

Physically, embedded systems find their application in portable devices such as digital watches and MP3 players, as well as in large installations like traffic lights, factory controllers etc. Complexity varies from low i.e. a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis.

Since many systems have some element of programmability in them, an embedded system hence is not an exactly defined term. For example, handheld computers are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected.

1.2.2 Examples of embedded systems

- Telecommunication systems uses vast number of embedded systems from telephone switches for the network to mobile phones at the user end. Computer networking uses embedded systems in the form of dedicated routers and network bridges to route data.
- Consumer electronics like Personal Digital Assistants (PDAs), mobile phones, videogame consoles, digital cameras, Digitized Video Discs (DVD) players, Global Positioning System (GPS) receivers, and printers. Household appliances, such as microwave ovens, washing machines and dishwashers, are employing embedded systems

to provide flexibility and efficiency. Home automation systems used to control lights, climate, security etc., makes use of embedded devices for sensing and controlling.

- Transportation systems from airplanes to automobiles numerously use embedded systems. Advanced avionics such as inertial guidance systems and GPS receivers are in use. A wide range of automotive safety systems such as Anti-lock Braking System (ABS) and Electronic Stability Control (ESP) also makes use of embedded systems.
- Medical equipment is continuing to advance due to the advancements made in the field of embedded systems. Embedded systems proved to be beneficial in vital signs monitoring, electronic stethoscopes, and various medical imaging techniques like Positron Emission Tomography (PET), Computed Tomography (CT), Magnetic Resonance Imaging (MRI).

1.2.3 Characteristics

- Embedded systems are not always standalone devices. Many embedded systems consist of various other peripherals within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar [2] features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music.
- **Reactive and real time operation:** Real time system operation means that the correctness of a computation depends, on the time in which it is delivered. Reactive computation means that the software executes in response to external or internal events. They continually reacts to changes in the system's environment.
- They are tightly constrained i.e. they provide fast and low cost solutions. They also consume less power and space.
- A wide array of processors like microcontrollers, microprocessors, Digital Signal Processing (DSP) processors supports the embedded systems.
- The program instructions written for an embedded system are known as firmware.
- **User interfaces:** Embedded systems range from no user interface i.e. dedicated to a single task to complex graphical user interfaces resembling modern computer operating systems.
- **Simple systems:** Simple embedded systems make use of buttons, LEDs, and small character or digit-only displays, with a simple menu system**.**

- **More complex systems:** A full graphical screen, with touch sensing panel provides flexibility while reducing the space used and the meaning of the buttons changes as the screen changes.

  An example – Fig.1.1 gives an inside view of a **Digital camera**

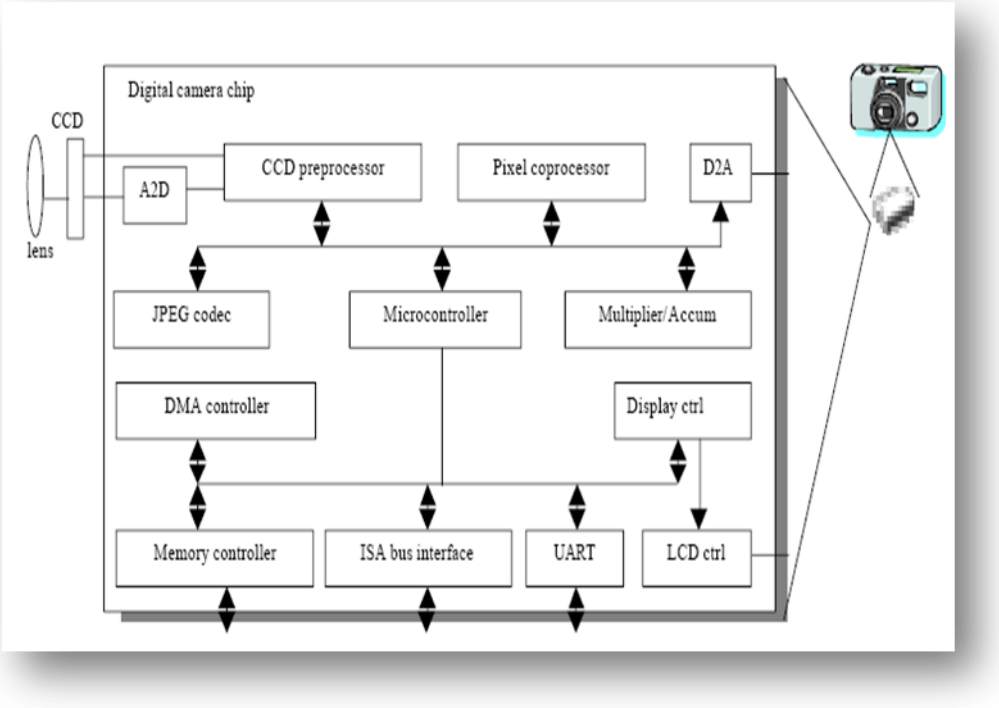


Figure 1.1 - Embedded system example - a digital camera

- Single functioned – will always work as a camera
- Small size and fast
- Real time system

1.3 Challenges and technologies

1.3.1 Key challenges

Any embedded application of integrated circuits seeks to minimize, simultaneously, four factors that are given below:

- The transistor count remains an important metric of system efficiency as it has a great impact on die and package size, unit cost and power consumption. Due to the advancements in process technology, transistor area is continuously shrinking. However, both static and dynamic power consumption depend on the transistor count.
- The number of clock cycles required is also an important factor to consider because performance and power consumption greatly rely on this factor. If clock frequencies are increased for smaller process geometries then there will be more clock cycles in a given time interval, which results in, increased power consumption. Less is the number of clock cycles, lesser will be the power consumption.
- The time taken by the application to develop strongly influences its market acceptance. The development efforts get wasted if a product misses the market window. In most of the scenarios, software development takes more time and costs more as compared to its hardware counterpart.
- Nonrecurring engineering (NRE) costs such as mask manufacturing is one of the most important design metric. Due to the increased NRE costs, most of the leading process technologies are out of reach.

1.3.2 Existing technologies

1.3.2.1 Microprocessor

A microprocessor is an Integrated Circuit (IC) that incorporates most or all of the functions of a Central Processing Unit (CPU) [3]. The first microprocessors invented in the early 70s and were used for simple electronic calculators. Further, embedded uses of 4 and 8-bit microprocessors, such as terminals, printers, various kinds of automation etc, followed quickly. In the mid-1970s 8-bit microprocessors with 16-bit addressing led to the first general purpose microcomputers.

For a long interval of time, CPUs were constructed out of small and medium-scale ICs. The integration of the whole CPU onto a single chip therefore resulted in reduced cost of processing capacity. As the microprocessor capacity continued to increase, the other forms of computers have been rendered almost completely obsolete, with one or more microprocessor as processing element in everything from the smallest embedded systems to the largest supercomputers.

Characteristics:

- It is a general-purpose processor like CPU.
- There are no on chip RAM, ROM and I/O ports.
- Microprocessor based embedded systems are bulkier and expensive.
- They have the advantage of being versatile i.e. the designer can decide the amount of RAM, ROM and I/O ports needed for a particular system.
- They are normally used where cost and space are not the primary requirements e.g. a desktop PC

1.3.2.2 Microcontrollers

A microcontroller (μC) is generally referred to as a functional computer system-on-a-chip. It contains a processor core, memory, and programmable input/output peripherals. It is highly integrated, in contrast to a microprocessor, which only contains a CPU. In addition to the arithmetic and logic elements of a general-purpose microprocessor, the microcontroller incorporates additional elements such as read-write memory, read-only memory, flash memory, peripherals, and input/output interfaces.

They consume relatively little power (milliwatts or even microwatts), and generally have the ability to retain its functionality while waiting for an event such as an interrupt. Power consumption while sleeping (of the order of nanowatts), makes them ideal for low power and long lasting battery applications.

Microcontrollers are widely used in automatically controlled products and devices, such as automobile engine control systems, remote controls, office machines, appliances etc. By reducing the size, cost, and power consumption in contrast to a design using a separate microprocessor, memory, and input/output devices, microcontrollers prove to be more economical for electronically controlling many more processes.

Characteristics:

- They are special-purpose computers i.e. they do one thing well.

- They are small and provide low cost solutions.
- There is no versatility i.e. designer cannot decide the amount of RAM, ROM or I/O ports required by the system.
- They are normally used where cost and space are the primary considerations e.g. a TV remote control.

1.3.2.3 DSP processors

A **digital signal processor** is a specialized microprocessor designed specifically for digital signal processing, generally in real-time computing [4].

In DSP algorithms, a large number of mathematical operations require to be performed quickly on a set of data. First signals are converted from analog to digital form, and then manipulated, and are then again converted to analog form, as shown in Fig1.2. Many DSP applications have constraints on response time; that is, for the system to work, the DSP operation must be completed within some time constraint.

Figure 1.2 - Digital signal processing scheme

Although most general-purpose microprocessors and operating systems can execute DSP algorithms, but these microprocessors are not suitable for applications like cellular telephone and pocket PDA systems etc. because of power supply and space limit. A specialized digital signal processor, however, tends to provide a low-cost solution, with better performance and lower latency (response time).

The architecture of a digital signal processor is optimized for digital signal processing work. Some useful features of an optimized DSP processor are outlined below.

Features:

- A memory architecture designed for streaming data, using Direct Memory Access (DMA) extensively.
- Separate program and data memories (Harvard architecture).
- Instructions to increase parallelism: Single Instruction Multiple Data (SIMD), Very Long Instruction Word (VLIW), superscalar architecture.
- Special arithmetic operations, such as fast multiply-accumulates (MACs). Many fundamental DSP algorithms, such as Finite Impulse Response (FIR) filters or the Fast Fourier transform (FFT) depend on multiply-accumulates.
- Highly parallel multiply–accumulates (MAC units).

1.3.2.4 Application Specific Integrated Circuit (ASIC)

An application-specific integrated circuit (ASIC) is an integrated circuit (IC) which is customized for a specific application, rather than general-purpose. For example, a chip designed to run a cell phone is an ASIC.

Over the time, as feature sizes reduced and design tools improved, the maximum complexity (and hence functionality) possible in an ASIC has grown from 5,000 gates to over 100 million. Apart from peripherals like memory blocks and other large building blocks, modern ASICs often include entire 32-bit processors. Designers of ASIC use hardware description language (HDL), to describe the functionality of ASICs.

Generally, an ASIC configuration is employed for a product that will have a large production run. As can be imagined, ASIC design costs high, and therefore they are often reserved for high volume products.

Despite the fact that their costs are high, ASICs can be very cost effective for many high volume applications. It is possible to design an ASIC as per the exact requirement for the product and hence much of the overall design is contained in one IC. As a result, they are widely used in high volume products like cell phones or other similar applications.

The development and manufacturing process of an ASIC design including the ASIC layout is expensive one. In order to reduce the costs, different levels of customization are used. Costs are reduced for designs where large level of customization of the ASIC is not required. Essentially, three levels of ASIC that may be used are:

- **Full custom design:** This type of ASIC is the most flexible because it involves the design of the ASIC down to transistor level. The complete layout can be designed as per the exact requirements of the circuit. As it provides the highest degree of flexibility, the costs are very much higher and it takes longer periods to develop. The disadvantage is that the risks are higher as the whole design is untested and not built up from standard library components that have been used before.

- **Standard cell:** This type of ASIC is semi-customized. It contains some standard components like simple gates and flip-flops. This also gives a high degree of flexibility, if standard functions are able to meet the requirements.

- **Gate array:** This type of ASIC is the least customizable. Here the silicon layers are standard but the interconnections between different areas on the chip are customizable. This type of ASIC is ideal where a large number of standard functions are required that can be connected in a manner in order to meet the specific requirements.

1.3.2.5 Field Programmable Gate Array (FPGA)

A field-programmable gate array (FPGA) is a reprogrammable and reconfigurable semiconductor device i.e. the customer or designer can configure it after manufacturing—hence the name "field-programmable". A single FPGA chip can replace thousands of components by incorporating millions of logic gates in a single integrated circuit (IC). FPGAs support parallel processing; hence, different operations do not have to compete for the same resources. Consequently, the performance of one part of the application remains unaffected when additional processing is added.

To program an FPGA, one must specify that how you want the chip to work with the help of a logic circuit diagram or a source code in a hardware description language (HDL). FPGAs can be used to implement any logical function that an ASIC could perform, but the ability to update the functionality after shipping proves to be advantageous.

FPGAs contain programmable logic components called "logic blocks", also referred to as "configurable logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together resembling a one-chip programmable breadboard. The logic blocks can implement simple logic gates like AND and XOR or can also be configured to perform complex combinational functions. In most FPGAs, the logic block also incorporates memory elements, which may be simple flip-flops or more complete memory blocks.

1.4 FPGA – A leading edge technology

1.4.1 Greener and faster computing

Due to the emergence of a range of disruptive technologies and developments in the market for high-performance embedded systems, field programmable gate arrays (FPGAs) have become the need of the hour. This greatly supports the move towards greener computing, which uses less power.

Field-programmable gate array (FPGA) technology continues to gain momentum, and the worldwide FPGA market is expected to grow from $1.9 billion in 2005 to $2.75 billion by 2010 [5]. Since its invention by Xilinx in 1984, FPGAs have come a long way and are actually replacing custom ASICs and processors for a variety of applications. In the upcoming sections, we will be discussing the FPGA technology, as well as its benefits, which make them unique and successful.

1.4.2 Introduction to FPGAs

As already defined, FPGA is a reprogrammable and reconfigurable silicon chip i.e. it can be configured in the field. It contains programmable logic components that can be configured to emulate the functionality of basic logic gates or other complex functions. FPGA also have a

hierarchy of programmable interconnects which provides a high degree of flexibility while interconnecting the logic blocks. The basic FPGA structure is shown in Fig1.3.

We can configure these chips to implement custom hardware functionality without even touching a breadboard or soldering iron. For the system to work, one simply develops the computing task in software and compiles it down to a configuration file that contains information on how the components should be interconnected. Earlier, FPGA technology was only available to those that had a deep understanding of digital hardware design. The developments in the field of design tools, however, has changed the rules for FPGA programming, with new technologies that convert graphical block diagrams or even C code into digital hardware circuitry.

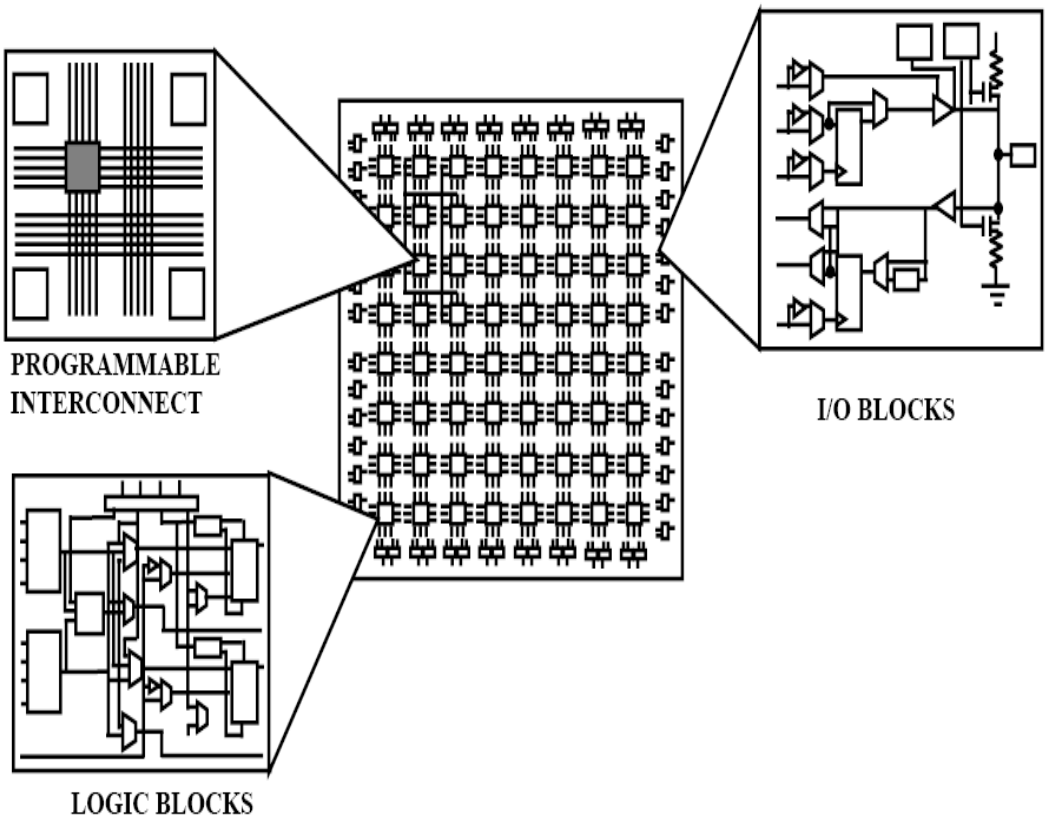


Figure 1.3 - FPGA structure

FPGA provides its user a way to configure:

- The interconnections between the logic blocks and
- The function of each logic block

1.4.3 Logic blocks

Logic block of an FPGA can be configured according to the required functionality. It can work as a simple gate or a complex microprocessor. Logic blocks of an FPGA can be implemented by any of the following:

- **Transistor pair:** In this transistor pairs run in parallel lines as shown in Fig1.4.

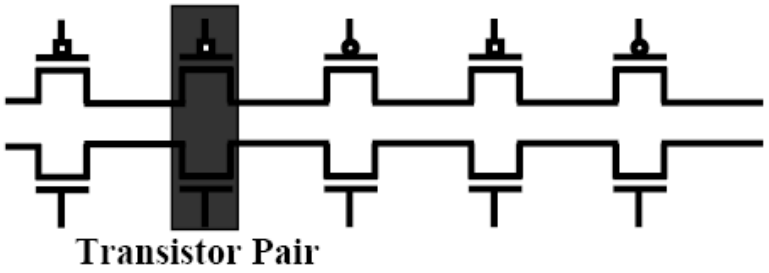


Figure 1.4 - Logic block having transistor pair tiles

- **Plessey logic block:** Basic building block here is 2-input NAND gate which is connected to each other to implement desired function. The layout is shown in Fig1.5.

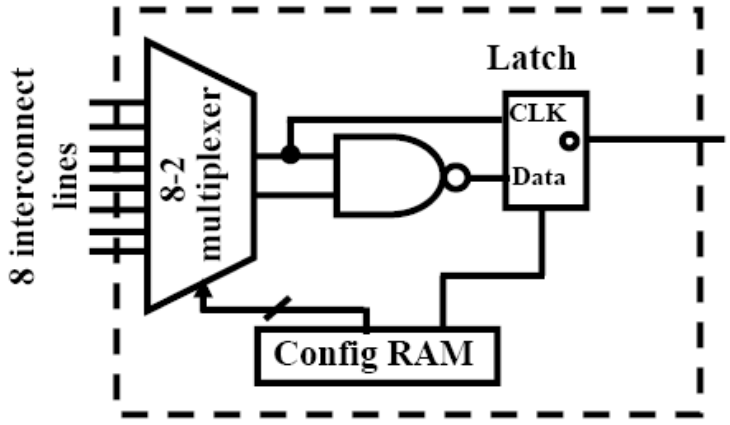


Figure 1.5 – Plessey logic block

- **Actel logic block:** Typically, an Actel logic block consists of multiple numbers of multiplexers and logic gates. Fig1.6 shows a typical Actel logic block.

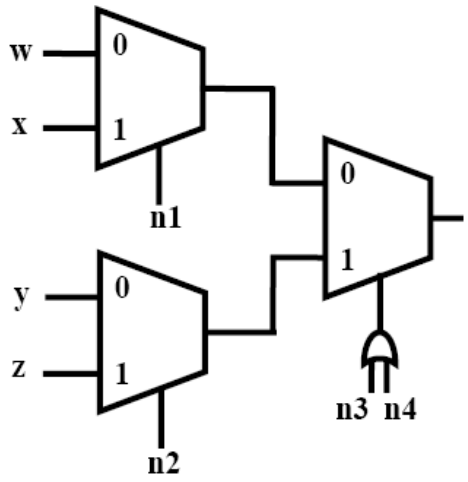


Figure 1.6 - Actel logic block

- **Xilinx logic block:** Xilinx logic blocks are based on the concept of look up tables that can implement any number of different functionality. A typical Xilinx logic block is shown in Fig1.7. The input lines are fed to the input and enable pin of lookup table. The output of the Lookup Table (LUT) gives the result of the function that it implements.

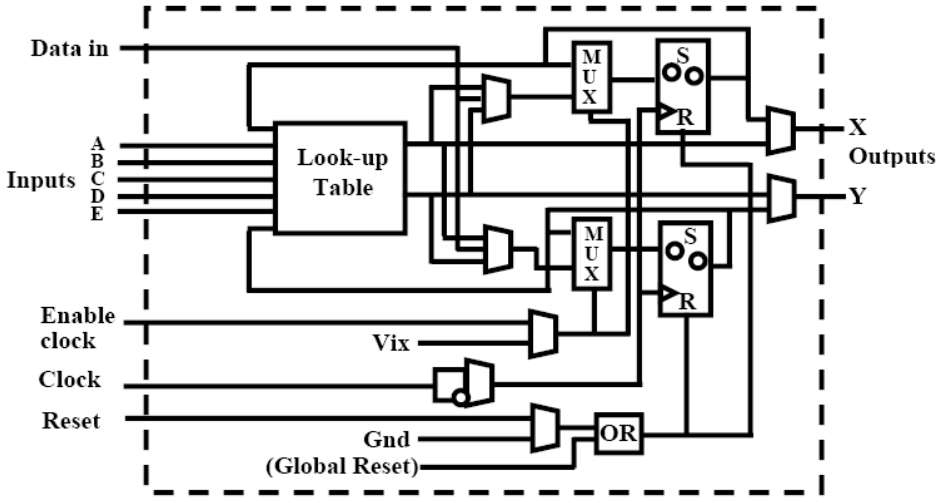


Figure 1.7 - Xilinx logic block

1.4.4 FPGA classification based on user programmable switch technologies

FPGAs are based on an array of logic components and a supply of wires to route signals. The user connects these wires and therefore an electronic device must be used to connect them. Three types of devices that have been commonly used to do this are given below:

- Pass transistors controlled by an Static Ram (SRAM) cell
- A direct connect using antifuses
- A flash or Electrically Erasable Programmable ROM (EEPROM) cell to pass the signal

Each of these interconnect devices have their own advantages and disadvantages which affects the design, architecture, and performance of the FPGA.

- **SRAM based FPGA:** The FPGA's configuration pattern is stored in the SRAM cell. The major advantage of SRAM based devices are that they are infinitely re-programmable and can be embedded into the system. Their functionality can be quickly changed by simply changing the contents of the Programmable ROM (PROM). The disadvantage is that they need to be reprogrammed each time when power is applied because SRAM is a volatile form of memory. Hence, SRAM based FPGAs needs an external memory to store program and hence require large area.

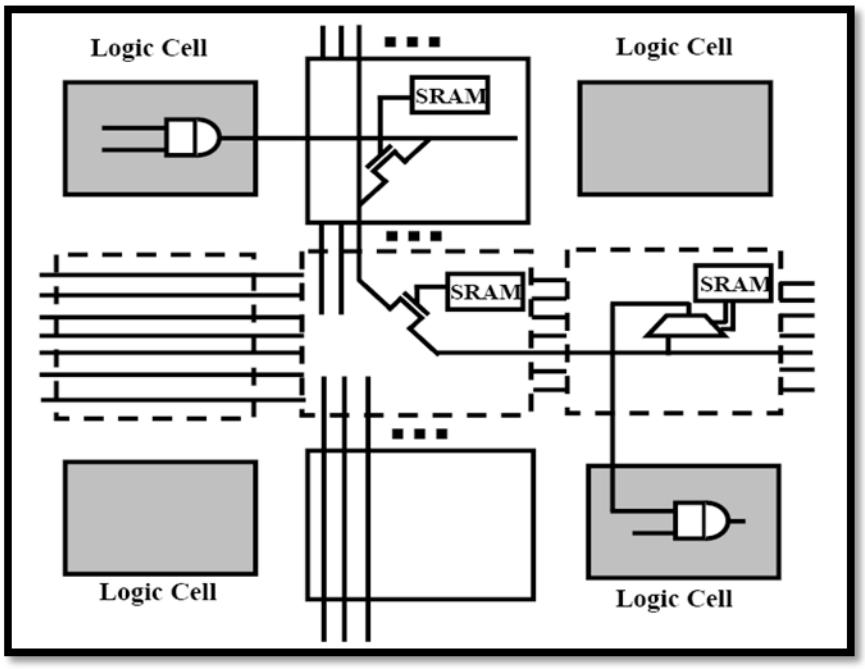


Figure 1.8 - SRAM controlled programmable switches

Fig1.8 depicts the connection of one logic block (represented by the AND-gate in the upper left corner) to another through two pass-transistor switches, and then a multiplexer, all controlled by SRAM cells. Depending upon the product requirement, the FPGA can use pass-transistors or multiplexers, or both. [6]

- **Antifuse based FPGA:** An antifuse is the opposite of a regular fuse i.e. an antifuse is normally an open circuit until you apply a programming current to it. These devices however are only one-time programmable and therefore have to be replaced every time the design is modified. Physically, it consists of an insulator layer between two conductors. When current is applied, the insulator changes to a low resistance link, which is permanent. The disadvantage of the antifuse is the requirement to integrate the fabrication of the antifuses into the IC process.

  As an example, Actel's antifuse structure is depicted in Fig1.9. The figure shows an antifuse positioned between two interconnect wires and physically consisting of three layers: the top and bottom layers are conductors, and the middle layer is an insulator. When not programmed, the insulator isolates the top and bottom layers, but after programming, the insulator changes to become a low-resistance link. It uses Poly-Si and n+ diffusion as conductors and Oxide-Nitride-Oxide (ONO) as an insulator, but other antifuses rely on metal for conductors, with amorphous silicon as the middle layer [6].

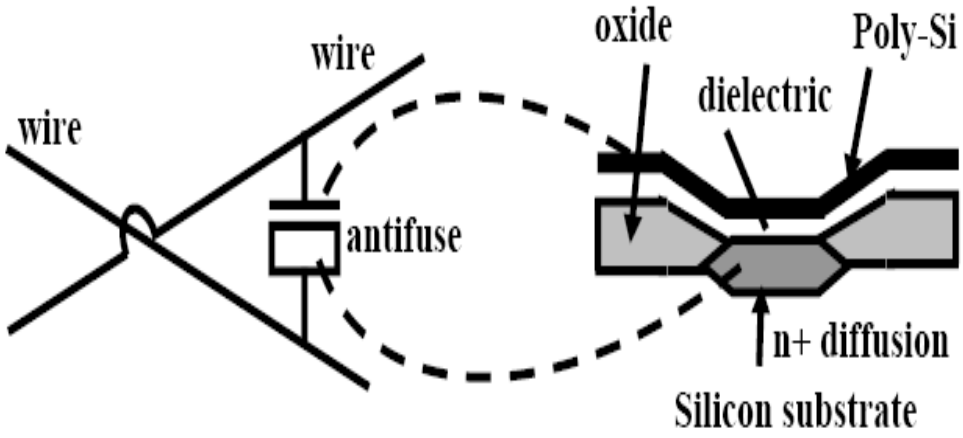


Figure 1.9 - Actel antifuse structure

- **Flash/EEPROM based FPGA:** These FPGAs store their configuration patterns in the reprogrammable storage cells. They are non-volatile so they do not require an extra PROM for loading. They are most efficient as an interconnect.

1.4.5 FPGA Vs Microcontroller

- Normally microcontroller is embedded in an FPGA.
- The main advantage of an FPGA is that it is reconfigurable.
- FPGA supports parallelism i.e. multiple instructions can execute in parallel while in microcontroller, instructions execute sequentially.
- High performance is provided by FPGA
- FPGA provide low cost solutions

FPGA disadvantages

- It is difficult to design and debug.
- It takes higher time to market as compared to microcontroller.
- It is harder to change the design functionality.

1.4.6 FPGA Vs ASIC

- The major advantage of FPGA over ASIC is that the interconnects between various logic blocks of an FPGA are reprogrammable (except in case of antiufse based FPGAs).
- The system development life cycle of an ASIC is much longer i.e. design is frozen into final state before much of silicon layout and circuit design work.
- In FPGAs, design can be rapidly loaded bypassing the time consuming and expensive Custom IC development process.
- FPGAs take lesser time to market as compared to an ASIC.
- NRE cost of an ASIC is high.

FPGA disadvantages

- Power consumption is high because FPGAs contain much longer routing tracks and the switching activity on these long routing tracks causes significant power dissipation.
- Per unit cost of FPGA is high because the programmable logic and interconnections found in an FPGA results in very large die sizes thereby reducing yield and significantly increasing manufacturing cost. External non-volatile programming devices also add to the cost.

- **Interconnect delay**: The reason behind the performance gap between ASIC and FPGA is the interconnect delay. In contrast to ASIC, where length of the metal routing is optimized, FPGA routing is a combination of different fixed wire lengths connected via pass transistors. These active routing elements add significant delay to signal paths.
- FPGAs also waste many internal LUT resources.

1.5 CompactRIO: FPGA based control & acquisition system

1.5.1 Introduction

National Instruments (NI) CompactRIO is a small rugged industrial control and acquisition system powered by Reconfigurable I/O (RIO) FPGA technology for ultrahigh performance and customization. NI CompactRIO incorporates a real-time processor and reconfigurable FPGA for reliable stand-alone embedded or distributed applications, and hot-swappable industrial I/O modules with built-in signal conditioning for direct connection to sensors and actuators. With NI CompactRIO, one can rapidly build embedded control or acquisition systems that rival the performance and optimization of custom-designed hardware circuitry. CompactRIO embedded systems are developed using LabVIEW, the LabVIEW Real-Time (RT) Module and the LabVIEW FPGA Module. [7]

1.5.2 CompactRIO Embedded System

A CompactRIO embedded system features a real-time embedded processor, 4 or 8-slot reconfigurable chassis containing a user-programmable FPGA, and hot-swappable industrial I/O modules. This low-cost embedded architecture delivers open access to low-level hardware resources for rapid development of custom stand-alone or distributed control and acquisition systems. Fig.1.10 displays a CompactRIO embedded system.

Figure 1.10: CompactRIO embedded system

1.5.3 Low-Cost Open Architecture

CompactRIO combines a low-power-consumption real-time embedded processor with a high-performance RIO FPGA chip. The RIO core has built-in data transfer mechanisms to pass data to the embedded processor for real-time analysis, post-processing, data logging, or communication to a networked host computer. Each I/O module includes built-in connectivity, signal conditioning, conversion circuitry (such as ADC or DAC), and an optional isolation barrier. This design represents a low-cost architecture, as shown in fig.1.11, with open access to low-level hardware resources.

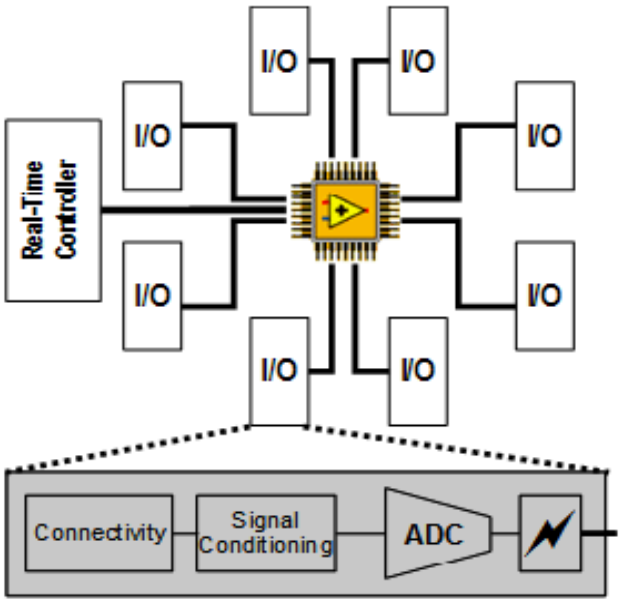


Figure 1.11: CompactRIO architecture

1.5.4 I/O Modules

Each CompactRIO I/O module contains built-in signal conditioning and screw terminal, BNC or D-Sub connectors. By integrating the connector junction box into the modules, the CompactRIO system significantly reduces the space requirements and cost of field wiring. A variety of I/O types are available including ±80 mV thermocouple inputs, ±10 V simultaneous sampling analog inputs/outputs, 24 V industrial digital I/O with up to 1 A current drive, differential/TTL digital inputs with 5 V regulated supply output for encoders, and 250 Vrms universal digital inputs. Because the modules contain built-in signal conditioning for extended voltage ranges or

industrial signal types, you can usually make your wiring connections directly from the CompactRIO module to your sensors/actuators [7]. Such an I/O module is shown in fig.1.12.

Figure 1.12: An I/O module

1.5.5 Real-Time Processor

The CompactRIO embedded system features an industrial 200 MHz Pentium class processor that reliably and deterministically executes the LabVIEW Real-Time applications. One can choose from thousands of built-in LabVIEW functions to build a multithreaded embedded system for real-time control, analysis, data logging, and communication. The controller also features a 10/100 Mb/s Ethernet port for programmatic communication over the network (including e-mail) and built in Web (HTTP) and file (FTP) servers. Using the remote panel Web server, you can automatically publish the front-panel graphical user interface of your embedded application for multi-client remote monitoring or control. The real-time processor also features dual 11 to 30 VDC supply inputs, a user DIP switch, LED status indicators, a real-time clock, watchdog timers, and other high-reliability features. Fig.1.13 depicts a cRIO real-time processor.

Figure 1.13: Real time processor

1.5.6 Reconfigurable Chassis

The reconfigurable chassis, as shown in fig.1.14, is the heart of NI CompactRIO embedded systems, containing the RIO FPGA core. This user-defined RIO FPGA is a custom hardware implementation of your control logic, input/output, timing, triggering, and synchronization design. The RIO FPGA chip is connected to the I/O modules in a star topology, for direct access to each module for precise control and unlimited flexibility in timing, triggering, and synchronization. A local PCI bus connection provides a high-performance interface between the RIO FPGA and the real-time processor. The reconfigurable chassis features the same rugged metal construction that characterizes the entire CompactRIO platform.

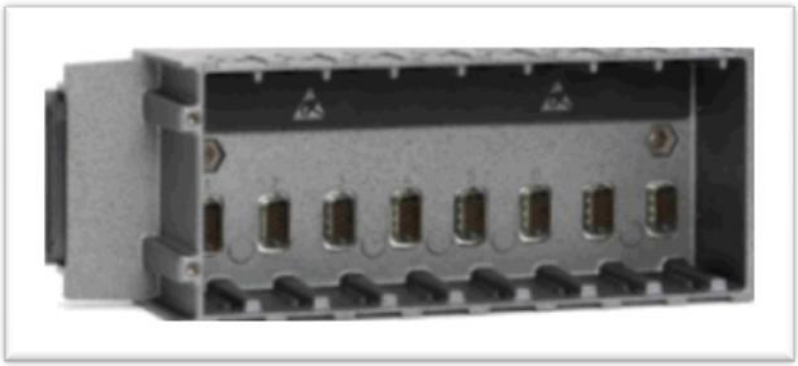


Figure 1.14: The reconfigurable chassis

1.5.7 Reconfigurable I/O (RIO) Technology

With NI RIO technology, one can define their own custom measurement hardware circuitry using reconfigurable FPGA chips and LabVIEW graphical development tools. Reconfigurable FPGA technology proves advantageous to automatically synthesize a highly optimized electrical circuit implementation of our input/output, communication, or control application. [7]

FPGA devices are widely used by control and acquisition system vendors because of their performance, reconfigurability, small size, and low engineering development costs. One can take advantage of user-programmable FPGAs to create highly optimized reconfigurable control and acquisition systems with no knowledge of specialized hardware design languages such as VHDL. With CompactRIO, one can design their own custom control or acquisition circuitry in silicon with 25 ns timing/triggering resolution.

1.5.8 Performance, Size, and Weight

Using LabVIEW FPGA software and reconfigurable hardware technology, one can create ultrahigh performance control and acquisition systems with CompactRIO. The FPGA circuitry is a parallel processing reconfigurable computing engine that executes the LabVIEW application in silicon circuitry on a chip. LabVIEW FPGA features built-in functions for analog closed-loop PID control, fifth-order FIR filters, 1D LUTs, linear interpolation, zero-crossing detection, and direct digital synthesis of sine waves.

With the embedded RIO FPGA hardware, you can implement multiloop analog PID control systems at loop rates exceeding 100 kS/s. Digital control systems can be implemented at loop rates up to 1 MS/s, and it is possible to evaluate multiple rungs of Boolean logic using single-cycle while loops at 40 MHz (25 ns). Due to the parallel nature of the RIO core, adding additional computation does not necessarily reduce the speed of the FPGA application.

CompactRIO is designed for applications characterized by harsh environments and small places. Size, weight, and I/O channel density are critical design requirements in many such embedded applications. By taking advantage of the extreme performance and small size of FPGA devices, CompactRIO is able to deliver unprecedented control and acquisition capabilities in a compact, rugged package. For example, a 4-slot reconfigurable embedded system measures 179.6 by 88.1 by 88.1 mm (7.07 by 3.47 by 3.47 in.) and weighs just 1.58 kg (3.47 lb).

1.5.9 Extreme Industrial Certifications and Ratings

CompactRIO combines reliable stand-alone embedded capability with extreme industrial certifications and ratings for operation in harsh industrial environments. Some of its striking features are listed below:

- –40 to 70 ℃ (–40 to 158 °F) operating temperature
- Up to 2,300 Vrms isolation (withstand)
- 50 g shock rating
- International safety, Electromagnetic Compatibility (EMC), and environmental certifications
- Class I, Division 2 rating for hazardous locations
- Dual 11 to 30 VDC supply inputs, low power consumption (7 to 10 W typical)

1.5.10 Applications

Because of its low cost, reliability, and suitability for high-volume embedded measurement and control applications, CompactRIO can be adapted to solve the needs of a wide variety of industries and applications. Examples include:

- Batch control
- Discrete control
- Motion control
- In-vehicle data acquisition
- Machine condition monitoring
- Rapid control prototyping (RCP)
- Industrial control and acquisition
- Distributed data acquisition and control
- Mobile/portable noise, vibration, and harshness analysis

CompactRIO is designed for advanced developers who will use LabVIEW graphical development tools to adapt the reconfigurable hardware for a wide variety of industries and applications. Customers such as MTS, Roush, Göpel, Process Automation and Virginia Tech have already successfully developed CompactRIO embedded systems for heavy machine control, in-vehicle data acquisition, acoustics and vibration analysis and electric motor drive characterization.

Chapter 2

Literature Review

2.1 CompactRIO – Practical implementations

In the previous chapter, we discussed about NI CompactRIO, an FPGA based control and acquisition system. Using the power of the FPGA core, the brains behind the CompactRIO, engineers can design 1 MHz digital control loops with no performance reductions with an increased number of logic computations and running 100 kHz analog PID control loops using 32-bit, integer-based calculations in the FPGA. Machine builders are using features like the speed and customization to integrate ultra high speed motion control for servo and stepper motors.

CompactRIO also allows engineers to easily and quickly redesign and upgrade their embedded systems as per the requirements of the application. As the programming of FPGA is done in LabVIEW, which is a graphical programming language, the system functionality can be changed by simply changing the LabVIEW code and download a new bit stream configuration to the FPGA hardware.

Because of its low cost, reliability, and suitability for high-volume embedded measurement and control applications, CompactRIO is employed to a wide variety of industries and applications. Brief overviews of some of its practical implementations are discussed in this chapter.

2.1.1 Motorcycle control prototyping

Carroll Dase, Jeannie Sullivan Falcon and Brian Maccleery developed a research Engine Control Unit (ECU) that would be suitable for prototyping engine control algorithms as well as sensor and actuator design. For this, there was a need for reliable, high-performance hardware and custom software.

This project involved the development of a research ECU for the 2004 Yamaha YZF-R6 motorcycle. National Instruments (NI) CompactRIO embedded control system was used because of its flexibility, small size, and rugged form factor. With this system, one can add sensors and actuators while readily visualizing the data. In addition, the controller can be mounted in the limited space available in a motorcycle. With the combined architecture of FPGA and a real-time processor, multiple control approaches and algorithms can be quickly designed and tested on the motorcycle.

CompactRIO completely replaced the original ECU located under the seat. Because the size of CompactRIO is larger than the factory ECU, the CompactRIO system is placed in the tail of the motorcycle. Once an engine control system has been tested with the research ECU, a modified factory ECU can be designed and optimized to meet size and cost constraints.

The modified motorcycle was test driven extensively. Experienced drivers could not detect significant differences between the factory ECU and CompactRIO control system. Thus, the CompactRIO system was shown to be an effective research ECU for the motorcycle. In the future, new control algorithms can be designed and tested using the CompactRIO system. [8]

2.1.2 Wind tunnel control

The faculty of Aerospace Engineering of the ***Delft University of Technology*** deployed a supersonic wind tunnel. It reaches speeds up to Mach 4.2. The tunnel is a "blowdown" type in which compressed air from an air accumulator is guided into the tunnel. There is a valve for controlling the airflow. Controlling this valve results in the desired pressure. Tunnel control is done by a CompactRIO and FPGA chassis. This CompactRIO is controlled and monitored by a PC and this PC is called the Host PC.

The tunnel controller consists of two sub controllers:

- Position control of the valve
- Pressure control of the settling chamber

Fig.2.1 represents the basic architecture for the wind tunnel control using NI CompactRIO.

Figure 2.1: Wind tunnel control

Both controllers are implemented as "hardware controllers" using the FPGA chassis of the CompactRIO. The embedded controller of the CompactRIO acts the interconnection between FPGA and Host PC. The Host PC initializes, monitors and configures the controllers online.
This results in a fast overall system response. The desired pressure and the wind speed can be realized within 2 seconds. Optimization of the controllers could result in even faster system response. [9]

2.1.3 Adaptive control braking system

Somil Gautam and Nishith Verma developed an automated adaptive braking system for automobiles with an effort to increase driver's comfort level and enhancing braking efficiency. The researchers tried to monitor automatically the braking force required to maximize the comfort while applying brakes.

Firstly, the data acquisition has been done using the ultrasonic proximity sensors, put around the vehicle, to calculate the distance of vehicles in its vicinity. The output is fed into one of the channels of the analog input module. Simultaneously the piston movement in the cylinder of the braking system is measured by a LVDT whose output is fed into another channel of the analog input module. The LabVIEW analyses the data from both these sensors and controls the valve and pump of the brake lines using the analog output module. The pump gets synchronized to this signal and pumps out the redundant fluid from the system, thereby introducing an adaptive controlled comfort braking in vehicles. The schematic architecture for the system is shown in fig.2.2.

This enables the researchers to develop a real-time data acquisition system that is highly deterministic and possess standalone characteristics. CompactRIO as a hardware allowed the researchers to do custom measurement with highly accurate loop rates. The cRIO modules used

were not only hot swappable with built in signal conditioning but has a high operating temperature range. [10]

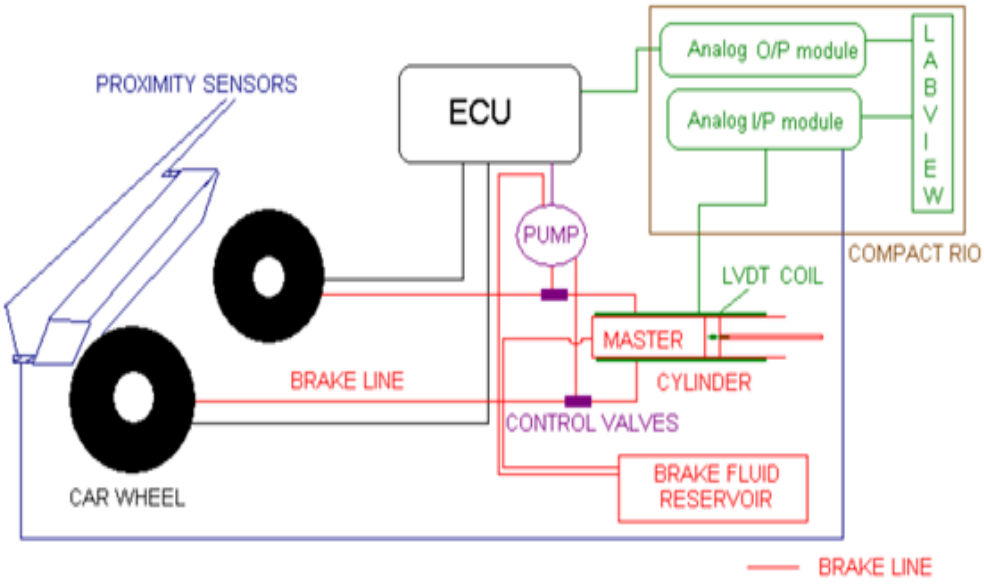


Figure 2.2: Adaptive control braking system

2.1.4 IT-based bridge health monitoring

Structural health monitoring (SHM) is a standard engineering practice today for a variety of structures in many sectors.

For bridges, SHM is of central importance. The major objective of bridge health monitoring is to identify damages or deterioration. Bridge health monitoring provides quantitative data of the bridge, and the data can be used for additional purposes as well. For example, the data can be used for accessing extent of damages/deterioration, evaluating the structural performance, responding to unexpected accidents, performing repair or strengthening and managing the bridge's normal operations. The data can also be employed for research purposes to improve bridge design and construction technologies.

Bridge health monitoring has some specific characteristics, such as the following:

- A bridge is a geographically distributed system, which is usually several to tens of kilometers in length.
- Bridges are usually working under rough environments and the data are gathered while the bridge is working.
- The monitoring is a long-term process and certainly a remote process, preferred with minimized on-site maintenance because of both the geographical distance and the environmental hostility.
- The monitoring is a continuous, real-time process, in which huge amounts of data are gathered. Smart techniques need to be employed to obtain characterizing information for evaluation from these data.

These characteristics establish a set of challenges that need to be addressed in the development of the bridge health monitoring system.

A paper published by ***Ayaho Miyamoto*** not only introduces an IT based bridge health monitoring system incorporated with latest information technologies but also a data collecting system for bridge health monitoring

Bridge health monitoring via information technology and sensors is capable of providing more accurate knowledge of bridge performance characteristics in contrast to the traditional methods. Monitoring is an important technique for evaluating the soundness and diagnosing bridge performance based on the real time measurements of strain, displacement, vibration and other parameters. The data collection has been done with the help of CompactRIO.

Commercial data loggers/recorders are big, heavy, expensive and specialized. In contrast to these, the cRIO is compact and flexible and all the sensors can be attached to the cRIO. To transfer the sensed data from the bridge site to the central office, the traditional data collecting systems need to consist of some data communication device like a web/FTP server, which they do not have. In contrast the cRIO can work as a web/FTP server and it is easy to interface cRIO with data communication devices such as Ethernet.

In this application, two cRIO based data collecting systems are used:

- The temperature data collecting system consisting one thermocouple.

- The vibration data collecting system consisting of an accelerometer and a laser displacement meter. [11]

2.1.5 Power quality analysis

A power quality analyzer is basically a meter which is plugged into a power system to record, analyze and display advance electrical parameters in real time.

Electrical measurement department of VSB Technical University (Czech Republic) has been involved for more than 14 years in research and development of Power Quality Analyzer built on Virtual Instrumentation Technology. PC-based power quality analyzer with National Instruments DAQ board was designed and developed in this period.

After long time development and improvements of PC based Power Quality Analyzer, ***Petr Bilik, Ludvik Koval and Jiri Hajduk*** decided to port the proved PC code to CompactRIO. The original code of PC based instrument was divided into three layers:

- FPGA (data acquisition process)
- real-time processor, (data analysis functions)
- host PC (user interfaces)

All three described parts were implemented in the same development environment: National Instruments LabVIEW. [12]

2.1.6 Water quality sensor calibration system

This paper by, ***Helena Geirinhas Ramos, Octavian Postolache***, presents an application of LabVIEW graphical programming and FPGA technology in the area of sensor calibration. A water quality sensor calibrator based on compact reconfigurable FPGA core that works under a real-time controller is described. The system provides automatic calibration of standalone sensors such as turbidity, pH or conductivity using different calibration solutions that are injected in the calibration vessels using a set of pumps and electro valves. The control of the system actuators, data acquisition, primary filtering of the acquired signals and sensor modeling is implemented using the FPGA core included in the compact reconfigurable I/O system. To

program the FPGA, the LabVIEW FPGA Module that extends LabVIEW graphical development to reconfigurable FPGAs on National Instruments (NI) Reconfigurable I/O (RIO) hardware, is used. Parts of data processing, data logging, and data communication are embedded in the real time controller that is connected to the FPGA core through a PCI interface.

Indications of the pumps or electro valves state or the level sensor output state are acquired using an 8-channel digital input module (cRIO-9423) that also works under the programmed FPGA core control.

The sensor calibration system presents a user-friendly interfaced based on a HMI touch screen that is connected to the Ethernet port of real time controller. FPGA based implementation permits the different tasks associated with the calibration system to be performed in parallel mode, which implies shorter processing times and better accuracy. The FPGA core added to a real time controller permits the implementation of advanced processing techniques and data communication tasks not possible with conventional processing devices. [13]

2.1.7 Remotely transporting and launching an unmanned helicopter

Jason Collins, Michael Perreca and Caitlyn Worthington-Kirsch, in this project, built an enclosed trailer system that can be used to transport and aide in the launching of Unmanned Aerial Vehicles (UAVs).

Their goal was to design and build a trailer capable of safely carrying a UAV (Unmanned Aerial Vehicle) over rough terrain to the designated launch point, and then safely releasing the helicopter for launch. This would allow workers to effectively use a UAV helicopter without endangering themselves.

Compact RIO was used to control and monitor various functions and sensors of the trailer system. This system allows for on-board data manipulation from the sensors as well as the ability to actuate the motors and linear actuators that are required for latching, braking and enclosure sub-systems. The two modules used are the NI 9205 and the NI 9476. The NI 9205 is a 32 channel analog input module which allows for an analog signal input of ± 10V DC with 16-bit resolution. These inputs allows for potentiometers, reed switches and momentary switches to be imported into the LabVIEW VI to determine the status of the system. The NI 9476 is a 32

channel digital output. With the ability to output 6 – 36V DC as “Hi” and ground to be “Low”, the 9476 allows for control of status LEDs, relays, and motor controllers. Each channel has a maximum current output of 250mA but outputs can be paralleled to increase the current output for devices that require it. [14]

2.1.8 Wheelchair Direction Control

The paper by ***Chia-Hua Hsu, Huai-Yuan Hsu, Hsin-Yi Wang and Tzu-Chien Hsiao***, discusses about the development of a vocal cords vibration control wheelchair system, which proves to be beneficial for the patients wanting to move around on their own will. By attaching the microphone on the left side of the throat, the vibration of the vocal cords can be determined. The basic pronunciation can be distinguished by the difference of the resulting analog waveform (signals), and further instructions (commands) can be carried out by the vocal cords vibration control wheelchair system. For example, if the word “forward” is pronounced in Chinese, the wheelchair will then move forward automatically.

To instruct the movement of wheelchair by determining the variation of the output signal generated by vocal cords is not an easy task. Hence, a rapid and real-time processor with FPGA system, NI CompactRIO (cRIO-9014) is used as the development platform. When the wheelchair moves, CompactRIO can perform normally with impact vibration. The most important thing is that the CompactRIO has real-time processor, it can analyze the analog waveform (signal) that is generated by user rapidly and reactively response the operations that wheelchair should do. [15]

Chapter 3

Problem Definition and Proposed Solution

3.1 Problem definition

The main objective of our thesis is to configure the NI CompactRIO (cRIO) control and acquisition system, based on reconfigurable FPGA technology, and to build a simple control application around it.

As already mentioned in previous chapters, NI cRIO, a high performance programmable automation controller, has proved successful for a wide array of industrial and control applications. FPGA technology, the brains behind the cRIO, is in great demand for control and acquisition systems because of their performance, parallel processing, reconfigurability, small size and low development costs.

During our thesis work, we have developed a PID based temperature-controlled chamber using NI cRIO. Temperature-controlled chamber supports a large number of applications like incubators for microbial growth, for egg hatching and many more. Though there are many other factors like humidity, ventilation etc. which affects the operation of an incubator, accurate temperature control is the most vital one.

3.2 Proposed Solution

3.2.1 Hardware requirements

For constructing the chamber, we have used acrylic sheets of dimension 1.5×1.5 ft. and thickness 2mm. Holes and slits were cut at various places in order to introduce heating bulbs, fan and sensor. Then we built the chamber using a wooden frame, and fixed the acrylic sheets to the outer side of the frame. After this, we fixed the bulb holders and fan. The number of bulbs and their wattage depends upon the temperature requirements in the chamber. Here we have used two

bulbs of 200W each and a simple CPU fan for cooling purpose. For temperature sensing, we used a K-type thermocouple.

Fig.3.1 depicts some images of the chamber.

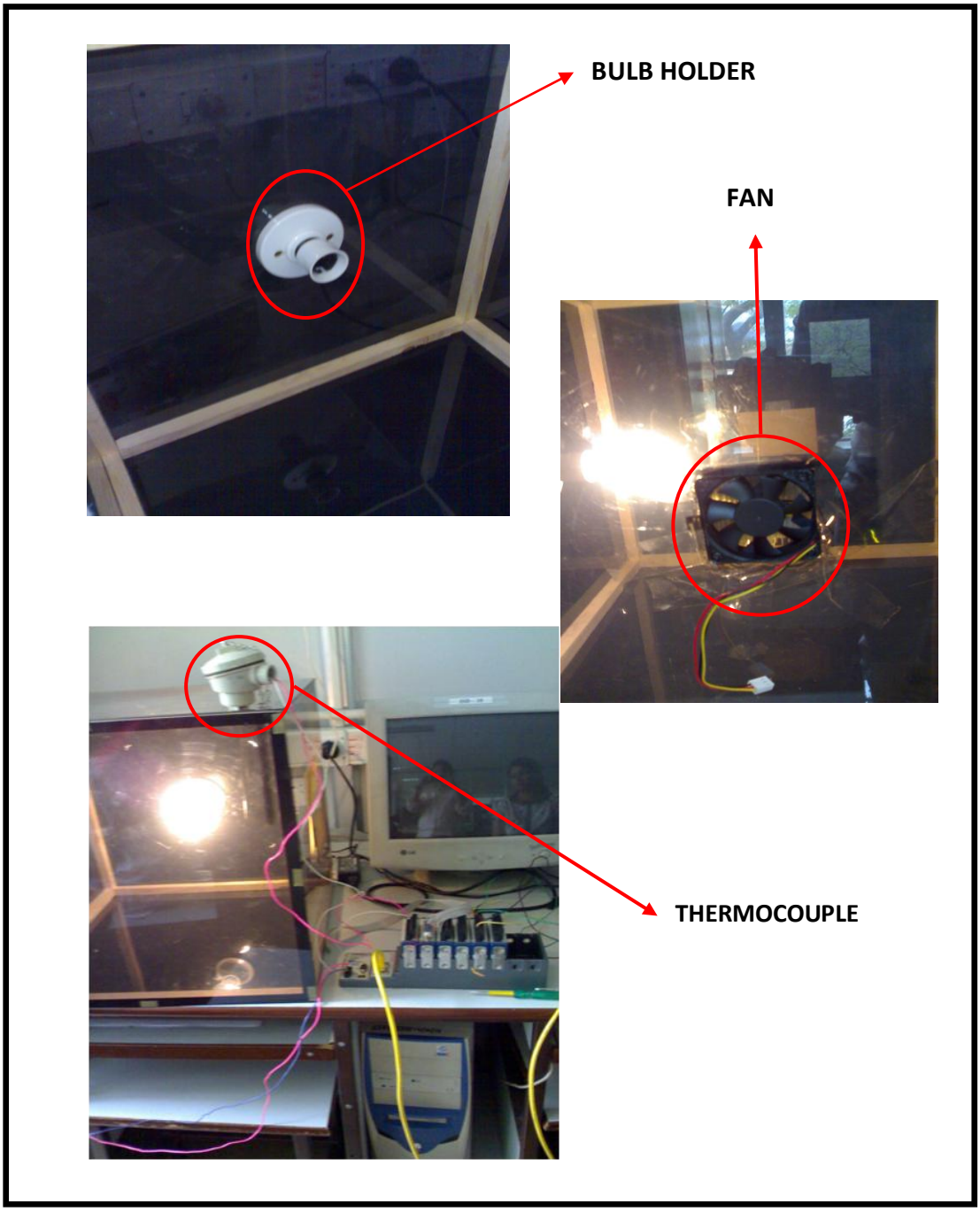


Figure 3.1: Chamber images

3.2.2 PID control algorithm

PID algorithm is the most widely used control loop feedback mechanism and has been universally accepted in industrial control. It attempts to minimize the error between the measured process variable and the desired setpoint by performing some calculations and generating a corrective action to adjust the process accordingly.

The calculation involves three parameters: Proportional (P), Integral (I) and Derivative (D). The proportional value reacts to the current error, the integral value reacts to the sum of recent errors and the derivative value reacts to the rate at which error has been changing. The weighted sum of these three actions is used to adjust the process. The response of the controller is described in terms of controller's response to the error, the degree of overshoot and the degree of oscillations.

Proportional term, Integral term and Derivative term are given as:

$$P_{\text{out}} = K_p\, e(t)$$

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

$$D_{\text{out}} = K_d \frac{de}{dt}(t)$$

where, P_{out} = output's proportional term

I_{out} = output's integral term

D_{out} = output's derivative term

K_p = proportional gain

K_i = Integral gain

K_d = Derivative gain

e = error (SP-PV)

t = instantaneous time

τ = integration variable

A large value of proportional gain produces large change in output for a given error. If the value is too high then the system will become unstable and if we keep the value too small then the system will be less responsive.

The integral term speeds up the process toward setpoint and also eliminates the steady state error that occurs with P-only control. However, overshoot of the present value can occur because it responds to the accumulated errors from past.

The derivative term slows down the rate of change of controller output and hence it is used to reduce the magnitude of overshoot, which occur with integral action. However, differentiation of a signal amplifies the noise, hence; this term is highly sensitive to noise in error term. The process will become unstable if derivative gain is sufficiently large. Fig.3.2 given below depicts the PID control algorithm:

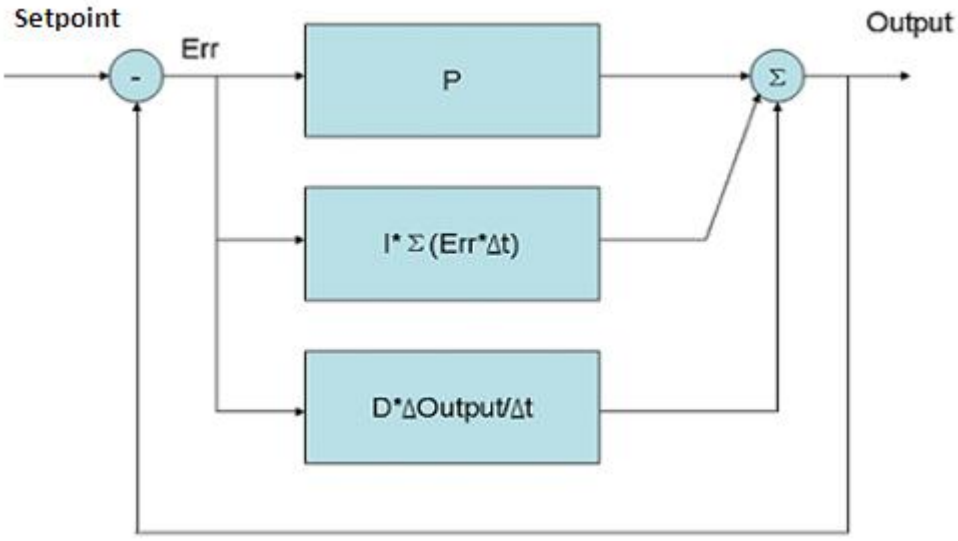


Figure 3.2: PID controller algorithm

It is not always necessary that PID algorithm will provide optimal system stability. Some applications may require only one or two modes for appropriate system control. This can be done by simply setting the values of undesired control outputs to zero. For example, if we set the derivative gain to zero then the controller is called as Proportional-Integral (PI) controller. PI controllers are most common as the derivative action is very difficult to tune because of their sensitivity to measurement noise and if we remove the integral action, then the system will not

be able to reach the target value. In our application also, we got good results using PI control only.

3.2.3 Control scheme for our application

The control scheme that we have used in our application is shown in fig.3.3. The main idea was to control the duty cycle for both the bulbs depending upon the PID controller output. The fan comes into action only when temperature rises above a threshold value.

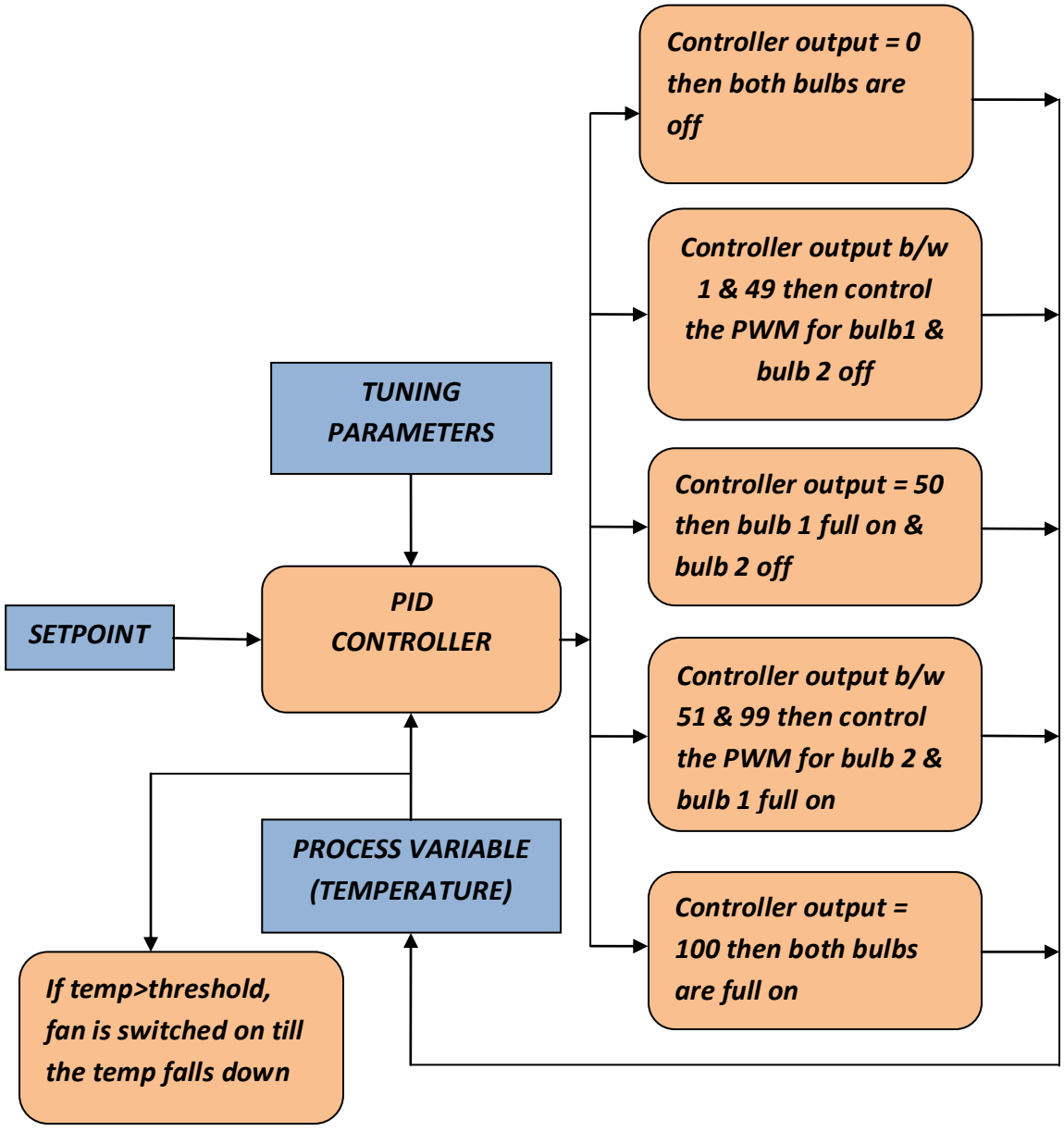


Figure 3.3: Block diagram representation of the control scheme

3.2.4 Using NI CompactRIO

During our work, we used two NI I/O modules- NI 9211 (Thermocouple analog input module) and NI 9474 (digital output module).

Thermocouple input module is used to acquire the temperature of the chamber. The module interacts with the FPGA core, which further communicates with the PC or the RT target.

The digital output module used here operates the bulbs and the fan. The fan is connected to the module directly while the bulbs are connected via relays. The module is programmed to generate Pulse Width Modulation (PWM) signal. The time period (Ton + Toff) for the fan is kept 200µs and for bulbs it is 10s. This module also needs an external 24V power supply, which provides current to the devices that are connected to it (in this case, relay and fan). The LEDs provided on the module indicates the state of each channel. Here we are using first three channels for the two bulbs and the fan. Therefore, the first three LEDs indicate the state of bulbs and fan.

Chapter 4

Configuring NI CompactRIO

4.1 NI cRIO-9074 overview

For our project, we have used NI cRIO- 9074 configurable chassis and controller, which is shown in fig.4.1.

Figure 4.1: CompactRIO-9074

Some of its striking features are listed below:

- Rugged, embedded control and monitoring system
- 400 MHz industrial real-time processor for control, data logging, and analysis
- 2M gate, 8-slot FPGA chassis for custom I/O timing, control, and processing
- Two 10/100BASE-T Ethernet ports; RS232 serial port for connection to peripherals
- -20 to 55 ℃ operating temperature range; single 19 to 30 VDC power supply input

4.1.1 Hardware overview

The controller Front panel, shown in fig.4.2, consists of

1) LEDs

2) RS-232 Serial Port

3) RJ-45 Ethernet port 2

4) RJ-45 Ethernet port 1

5) SMB Connector

6) Power Connector

7) Reset button

8) DIP Switches

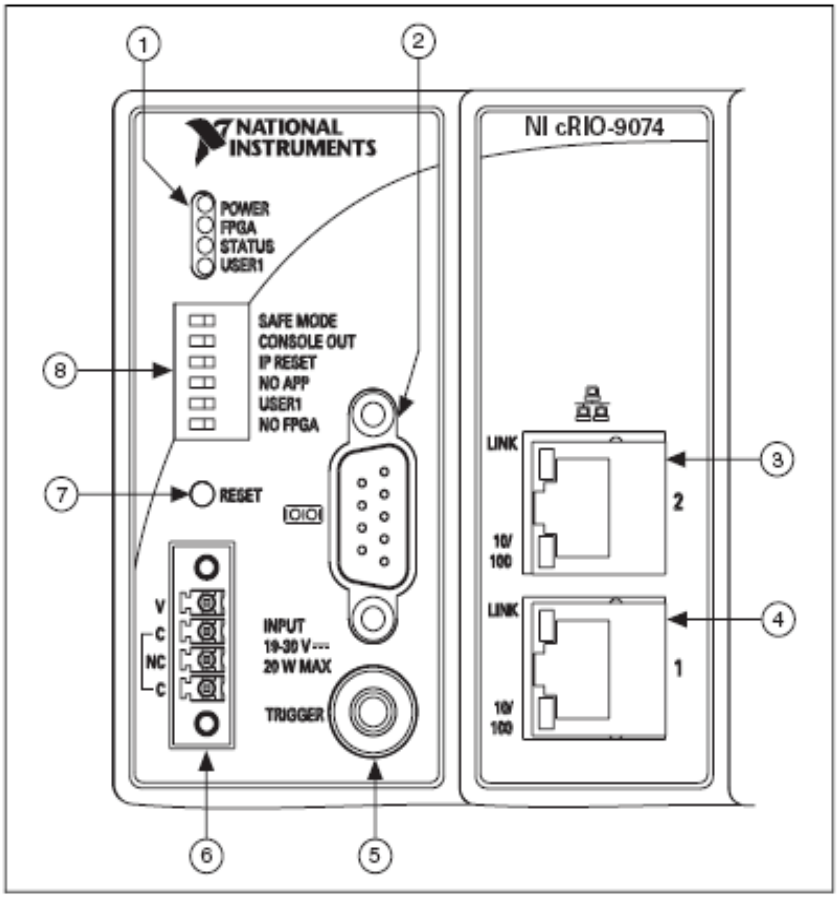


Figure 4.2: CompactRIO 9074 controller

Fig.4.3 given below shows the controller and the embedded chassis:

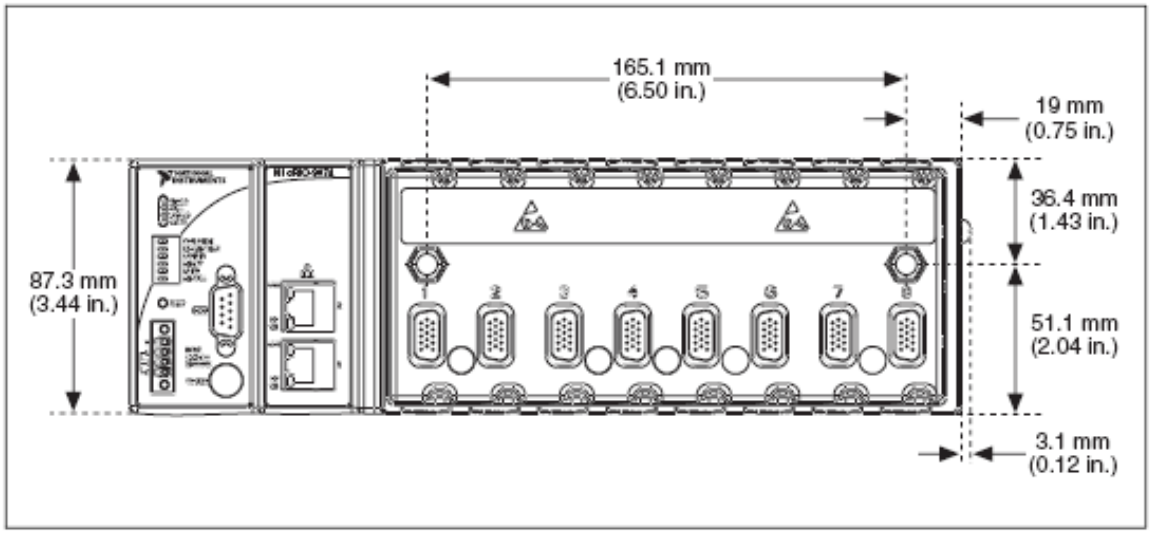
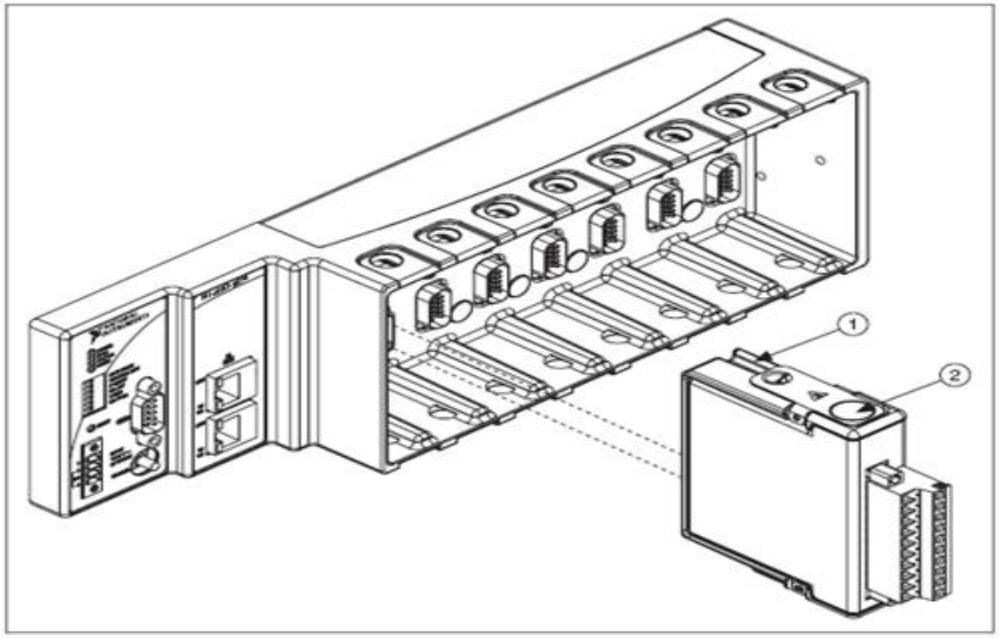


Figure 4.3: cRIO-9074 front view of controller and embedded chassis

4.1.2 Installing the I/O modules in the chassis

Fig.4.4 given below depicts the installation of an I/O module in the chassis:



1. Insertion groove	2. Latch

Figure 4.4: Installing an I/O module in the chassis

Complete the following steps to install a C Series I/O module in the chassis.

1) Make sure that no I/O-side power is connected to the I/O module. If the system is in a nonhazardous location, the chassis power can be on when you install I/O modules.

2) Align the I/O module with an I/O module slot in the chassis as shown in Figure. The module slots are labeled 1 to 8, left to right

3) Squeeze the latches and insert the I/O module into the module slot.

4) Press firmly on the connector side of the I/O module until the latches lock the I/O module into place.

5) Repeat these steps to install additional I/O modules.

4.1.3 Connecting the chassis to a network

The chassis is connected to an Ethernet network using RJ-45 Ethernet port 1 on the controller front panel. A standard Category 5 (CAT-5) or better Ethernet cable can be used to connect the chassis to a network.

If the host computer is on a network, the chassis must be configured on the same subnet as the host computer. If neither the host computer nor the chassis is connected to a network, one can connect the two directly using a crossover cable. We connected the chassis and the host PC directly to each other.

4.1.4 Wiring power to the chassis

The cRIO-9074 requires an external power supply of 48 W, 24V. It filters and regulates the supplied power and provides power for all of the I/O modules.

The chassis are powered by simply connecting the positive lead of the power supply to the V terminal of the power connector and the negative lead to the C terminal. When the power is applied to it, the controller runs a Power-On Self Test (POST). During the POST, the Power and Status LEDs turn on. The status LED turns off, indicating the POST is complete. If the LEDs do not behave in this way when the system powers on, one must refer to the *Understanding LED Indications* section in the instruction manual of cRIO-9074. [16]

4.1.5 Connecting serial devices to cRIO-9074

The cRIO-9072/9074 has an RS-232 serial port to which you can connect devices such as displays or input devices. Use the Serial VIs to read from and write to the serial port from a LabVIEW RT application. Fig.4.5 shows the controller serial port.

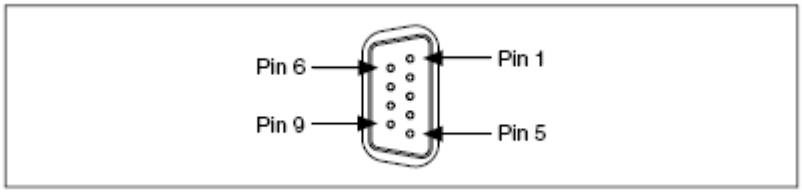


Figure 4.5: Controller serial port

4.1.6 Using the SMB connector for digital I/O

One can use the SMB connector of the cRIO-9074 to connect digital devices to the controller. For example, if you connect the pulse-per-second output of a GPS device to the SMB connector of the cRIO-9074, you can use the GPS device to correct for drift of the system clock.

4.1.7 Configuring the DIP switches

Fig.4.6 given below shows various DIP switches available on cRIO controller:

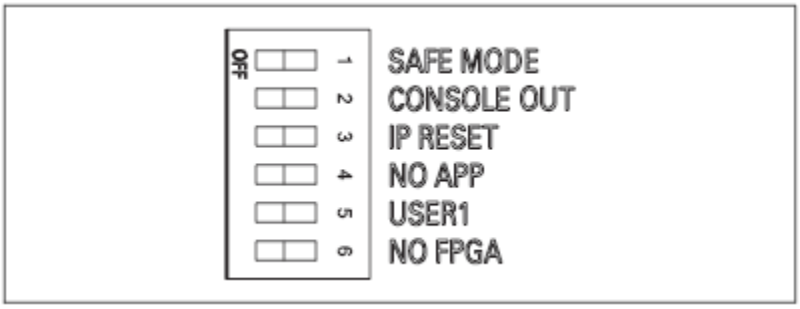


Figure 4.6: The DIP switches

Safe mode switch

The position of the SAFE MODE switch determines whether the embedded LabVIEW Real-Time engine launches at startup. If the switch is in the OFF position, the LabVIEW Real-Time

engine launches. This switch is in the OFF position during normal operation. If the switch is in the ON position at startup, the cRIO-9074 launches only the essential services required for updating its configuration and installing software. The LabVIEW Real-Time engine does not launch.

Console out switch

With a serial-port terminal program, you can use the CONSOLE OUT switch to read the IP address and firmware version of the controller. This switch is in the OFF position during normal operation.

IP reset switch

Push the IP RESET switch to the ON position and reboot the chassis to reset the Internet Protocol (IP) address to 0.0.0.0. If the chassis is on the local subnet and the IP RESET switch is in the ON position, the chassis appears in Measurement and Automation Explorer (MAX) with IP address 0.0.0.0. One can configure a new IP address for the chassis in MAX. More detail on this will be given in later on stage.

No app switch

Push the NO APP switch to the ON position to prevent a LabVIEW RT application from running at startup. If one wishes to permanently disable a LabVIEW RT application from running at startup, it must be disabled in LabVIEW. To run an application at startup, push the NO APP switch to the OFF position, create an application using the LabVIEW Application Builder, and configure the application in LabVIEW to launch at startup. If already an application is configured to launch at startup and the NO APP switch is pushed from ON to OFF, the startup application is automatically enabled.

User1 switch

The user for his application can define this switch. To define the purpose of this switch in the embedded application, use the RT Read Switch VI in your LabVIEW RT embedded VI.

No FPGA switch

This switch is kept in the ON position to prevent a LabVIEW FPGA application from loading at startup.

4.1.8 Understanding the LED indications and the Reset switch

Fig.4.7 shows various LEDs available on cRIO controller

Figure 4.7: cRIO-9074 LEDs

Power LED

The POWER LED remains in ON state while the cRIO-9072/9074 is powered on. This LED indicates that the power supply connected to the chassis is adequate.

FPGA LED

FPGA LED can be used to help in debugging ones application or easily retrieve application status. Using the LabVIEW FPGA Module and NI-RIO software, we can define the FPGA LED to meet the needs of our application.

Status LED

The STATUS LED is off during normal operation. The cRIO-9072/9074 indicates specific error conditions by flashing the STATUS LED a certain number of times as shown in table 1 given below:

No. of flashes	Indication
1	The chassis is unconfigured. Use MAX to configure the chassis.
2	The chassis has detected an error in its software. This usually occurs when an attempt to upgrade the software is interrupted. Reinstall software on the chassis.
3	The chassis is in safe mode because the Safe Mode DIP switch is in the ON position.
4	The software has crashed twice without rebooting or cycling power between crashes. This usually occurs when the chassis runs out of memory. Review the RT VI and check the memory usage. Modify the VI as necessary to solve the memory usage issue.
Continuous flashing or solid	The chassis has detected an unrecoverable error. In this case one should contact National Instruments.

Table 1: Indications by STATUS LED

User1 LED

USER1 LED can be defined to meet the needs of the application. To define the LED, use the RT LEDs VI in LabVIEW.

Using the RESET switch

Pressing the Reset switch resets the processor in the same manner as cycling power.

4.1.9 Resetting the Network Configuration of cRIO-9074

If the cRIO-9072/9074 is not able to communicate with the network, use the IP RESET switch to manually restore the chassis to the factory network settings. When we restore the chassis to the factory network settings, the IP address, subnet mask, DNS address, gateway, and Time Server IP are set to 0.0.0.0.
Perform the following steps to reset the chassis:

1) Move the IP RESET DIP switch to the ON position.
2) Push the RESET button to cycle power to the chassis. The STATUS LED flashes once, indicating that the IP address is unconfigured
3) Move the IP RESET switch to the OFF position.

The factory network settings are restored and now we can reconfigure the settings in MAX from a computer on same subnet, which will be discussed later on.

4.2 Required software

- LabVIEW
- LabVIEW Real-time
- LabVIEW FPGA
- Measurement and Automation Explorer (MAX)

Driver software

- NI RIO for reconfigurable embedded systems

During our thesis work, we have worked on LabVIEW Real-time 8.6, LabVIEW FPGA 8.6 and NI RIO 3.0.0.

4.2.1 LabVIEW

LabVIEW is a graphical programming environment used by millions of engineers and researchers to develop test, measurement and control systems using graphical icons and wires, resembling a flow-chart. It offers integration with thousands of hardware devices and provides hundreds of built-in libraries for advanced analysis and data visualization.

4.2.2 LabVIEW Real-time

The National Instruments LabVIEW Real-Time Module is an add-on component for the LabVIEW Development System. When installed, this software compiles NI LabVIEW graphical code and optimizes it for the selected real-time target. Using the LabVIEW Real-Time Module, we can develop and deploy applications to all NI real-time hardware targets like CompactRIO.

4.2.3 LabVIEW FPGA

National Instruments LabVIEW and the LabVIEW FPGA Module deliver graphical development for field-programmable gate array (FPGA) chips on NI Reconfigurable I/O (RIO) hardware targets. With an NI LabVIEW FPGA Module, we can develop FPGA VIs on a host computer running Windows, and LabVIEW compiles and implements the code in hardware. We can create embedded FPGA VIs that combine direct access to I/O with user-defined LabVIEW logic to define custom hardware for different applications.

4.2.4 Measurement and Automation Explorer (MAX)

Measurement and Automation Explorer (MAX) provides access to the National Instruments devices and systems. With the help of MAX, one can:

- Configure the NI hardware and software
- View devices and instruments connected to your system
- Update the NI software
- Execute system diagnostics

4.2.5 NI RIO driver software

NI-RIO 3.0.0 installs support for all R Series and CompactRIO embedded targets. NI-RIO 3.0.0 and higher versions support Windows Vista, the 32-bit and 64-bit version.

It provides support for LabVIEW 8.6, LabVIEW Real-time 8.6 and LabVIEW FPGA 8.6.

4.3 Configuring chassis and installing software

4.3.1 Configuring network settings

Complete the following steps to configure the network settings on the CompactRIO target:

1) Launch Measurement and Automation Explorer (MAX).

2) From MAX window, as shown in fig.4.8, expand **Remote Systems** in the configuration tree and select the remote system.

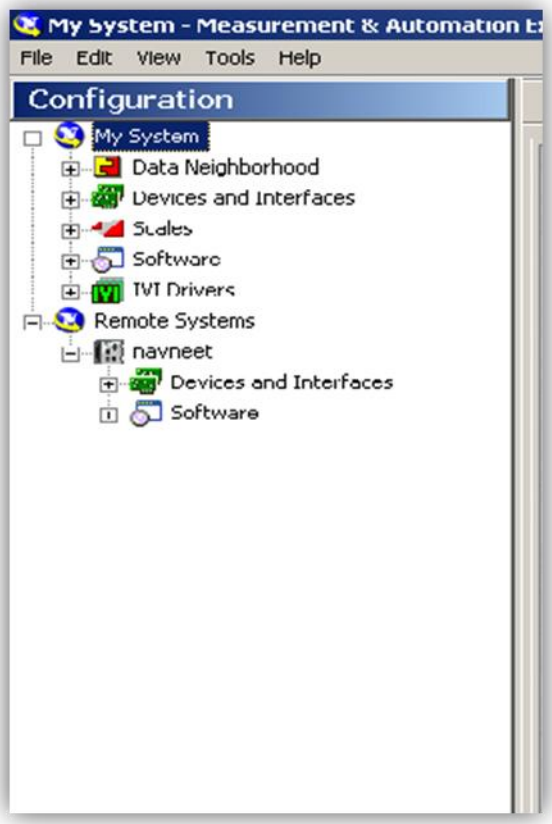


Figure 4.8: MAX window displaying Remote systems

3) Select the Network Settings tab. If this is the first time the controller is started, the settings could show either a network address, an automatic private IP address (169.254.*x*.*x*), or 0.0.0.0, depending on how the target acquires network addresses.

4) Enter a host name in the **Name** text box. During our work, we named it "navneet". If already a name is assigned to it, then one can change it or keep it.

5) Specify the IP address manually by selecting the Use the following IP address radio button and after that click the Suggest values button. You will be prompted for an IP address. Choose one that is not already in use on the network and click **OK**. If a Gateway and DNS are not available on the network, set these to the default value of 0.0.0.0.

6) Click **Apply** in the Network Settings tab.

7) Click **Yes** when prompted to restart the remote system.

4.3.2 Installing software

We can install the LabVIEW Real-Time Module or other driver software on the remote target after an IP address is assigned to it. If LabVIEW Real-Time is already installed, one may still need to download additional driver software or update existing driver software. Complete the following steps to download software using the LabVIEW Real-Time Software Wizard:

1) From MAX, as shown in fig.4.9, expand **Remote systems** in the configuration tree and then expand the RT target (in this case, CompactRIO named as "navneet").

2) Select **Software** and click the **Add/Remove Software** icon on the toolbar to launch the LabVIEW Real-Time Software Wizard.

3) Now, one can either install **Recommended software sets** or go for **Custom software selection**.

4) Click **Next** to move to the confirmation page after you make the appropriate selections. This page lists all the changes that are to be made to the remote target.

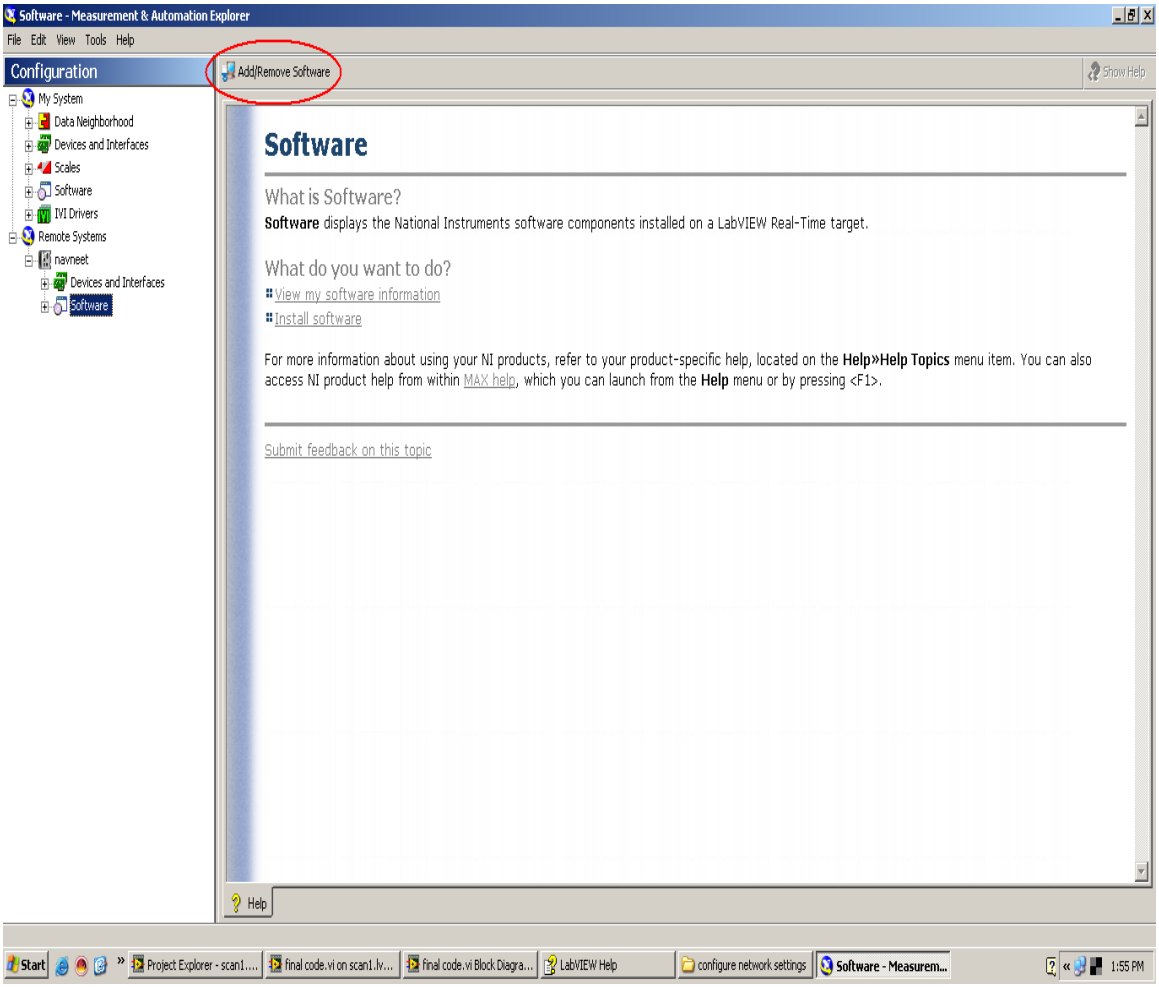


Figure 4.9: MAX window for adding software

5) Click **Next** to continue. The target first boots into install mode, if necessary, and then begins installation. When installation completes, the wizard reboots your remote target.

6) Click **Finish** to close the wizard.

7) The installed software should now be running on the remote target.

4.4 Various I/O modules

A wide array of I/O modules is available which are compatible with reconfigurable cRIO-9074 chassis. We have been working on six of them which will be discussed in this section

4.4.1 NI 9211: A 4-channel Thermocouple Input Module

The NI 9211 has a 10-terminal, detachable screw-terminal connector that provides connections for four thermocouple input channels. Each channel has a terminal to which you can connect the positive lead of the thermocouple, TC+, and a terminal to which you can connect the negative lead of the thermocouple, TC–. The NI 9211 also has a common terminal, COM, that is internally connected to the isolated ground reference of the module. [17]

Connect the positive lead of the thermocouple to the TC+ terminal and the negative lead to the TC– terminal. If you are unsure which of the thermocouple leads is positive and which is negative, check the thermocouple documentation or the thermocouple wire spool.

Table 2 given below shows the terminal assignments for each channel:

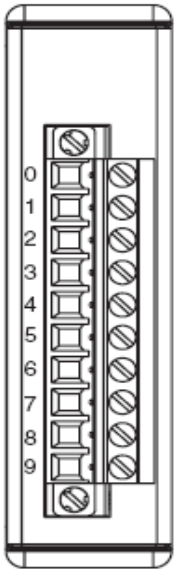
Module	Terminal	Signal
	0	TC0+
	1	TC0–
	2	TC1+
	3	TC1–
	4	TC2+
	5	TC2–
	6	TC3+
	7	TC3–
	8	No connection
	9	Common (COM)

Table 2: Terminal assignments for NI 9211

Some of its striking features are listed below:

- 4 channels, ±80 mV analog thermocouple inputs
- -40 to 70°C operating range

- 24-bit resolution
- Hot swappable operation

4.4.2 NI 9401: An 8-channel TTL Digital I/O Module

The NI 9401 shown in fig.4.10 has a DSUB connector that provides connections for the eight digital channels. Each channel has a DIO pin to which we can connect a digital input or output device. The DIO channels of the NI 9401 are grouped in two ports, one containing channels 0, 1, 2, and 3, and one containing channels 4, 5, 6, and 7. We can configure each port in software for input or output. All the four channels in a port must be configured in same direction. [18]

Figure 4.10: NI 9401 digital I/O module

Fig.4.11 given below shows the pin assignment for this module:

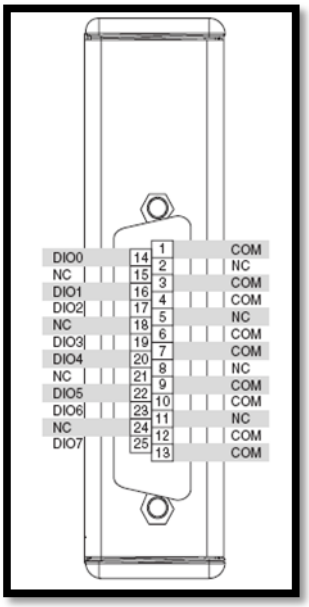


Figure 4.11: Pin assignment for NI 9401

Some of its striking features are given below:

- 8-channel, 100 ns ultrahigh-speed digital I/O
- 5 V/TTL, sinking/sourcing digital I/O
- -40 to 70 ℃ operating range
- Industry-standard 25-pin D-Sub connector
- Hot-swappable operation
- Bidirectional pins

4.4.3 NI 9265: A 4-channel, 16-bit Analog Current Output Module

The NI 9265 has a 10-terminal, detachable screw-terminal connector that provides connections for 4 analog output channels. The NI 9265 has four analog output channels, AO. Each channel has a common terminal, COM, that is internally connected to the isolated ground reference of the module. The 9265 also has a terminal for an external power supply, Vsup, and an external power supply common terminal, Power Supply COM. [19]

To connect a load to the NI 9265, the positive lead of the lead is connected to the AO terminal and the ground is connected to the corresponding COM terminal. Each channel has a digital-to-analog converter that produces a current signal.
Fig.4.12 showing the terminal assignments of this module is given below:

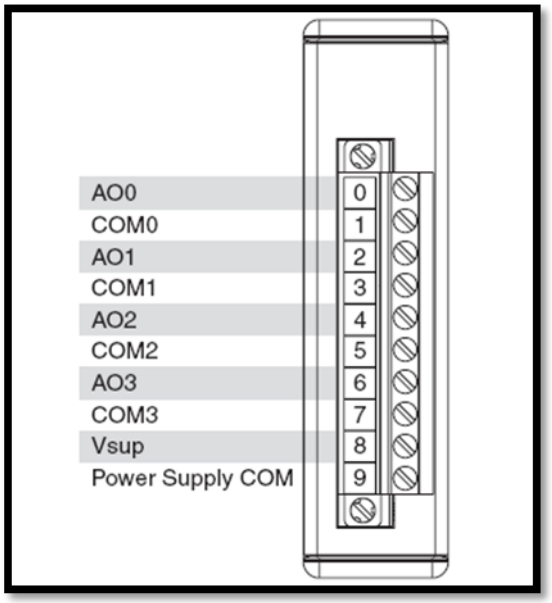


Figure 4.12: NI 9265 terminal assignment

Some of its striking features are mentioned below:

- 0 to 20 mA output range, 16-bit resolution
- Hot-swappable operation
- 4 analog outputs, 100 kS/s, simultaneously updated
- -40 to 70 °C operating range

Fig.4.13 given below illustrates how to connect a load to NI 9265:

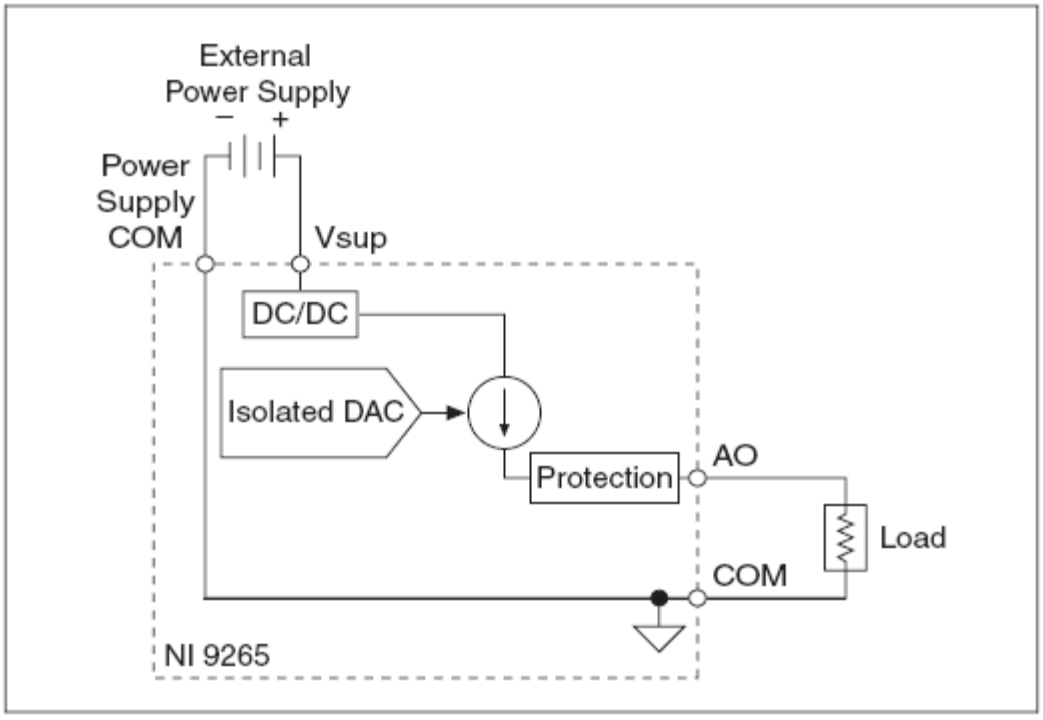


Figure 4.13: Connecting a load to NI 9265

4.4.4 NI 9201: An 8-channel Analog Input Module

The NI 9201/9221 with screw terminal has a 10-terminal, detachable screw-terminal connector. Each channel has an AI terminal or pin to which you can connect a voltage signal. COM, the common terminal or pin, is internally connected to the isolated ground reference of the module.[20]

Some of its features are listed below:

- 8 analog inputs, ±10 V input range

- 500 kS/s aggregate sampling rate
- 12-bit resolution, single-ended inputs, screw terminal or D-Sub connectors
- Hot-swappable operation; overvoltage protection; isolation
- -40 to 70 °C operating range

Table 3 given below shows the terminal assignment of NI 9201:

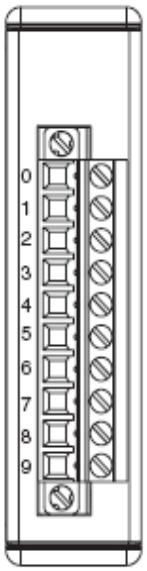
Module	Terminal	Signal
	0	AI0
	1	AI1
	2	AI2
	3	AI3
	4	AI4
	5	AI5
	6	AI6
	7	AI7
	8	No Connection
	9	COM

Table 3: Terminal assignments for NI 9201

To connect a voltage signal to NI 9201, connect the positive lead of the voltage signal to AI terminal and ground signal to the COM terminal as shown in fig.4.14 given below:

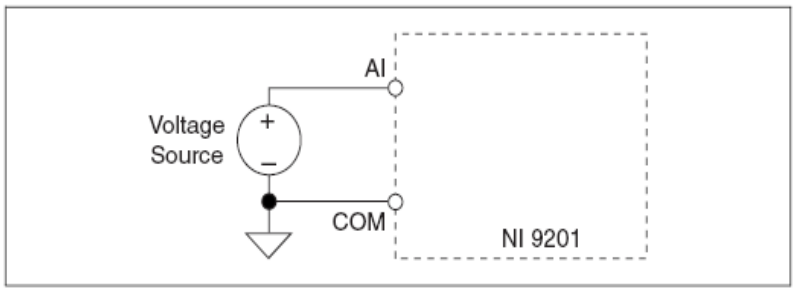


Figure 4.14: Connecting a voltage source to NI 9201

4.4.5 NI 9474: An 8-channel Digital Output Module

The NI 9472/9474 provides connections for eight digital output channels. Each channel of the NI 9474 has a terminal or pin, DO, to which you can connect a device. The eight digital output channels are internally referenced to the common terminal or pin, COM.

An external power supply must be connected to the NI 9474. This power supply provides the current for the devices we connect to the module. Connect the positive lead of the power supply to Vsup terminal and the negative lead of the power supply to COM terminal. The NI 9474 has current sourcing outputs therefore the DO terminal is driven to Vsup when the channel is turned on. [21]

Fig.4.15 given below shows the NI 9474 terminal assignment:

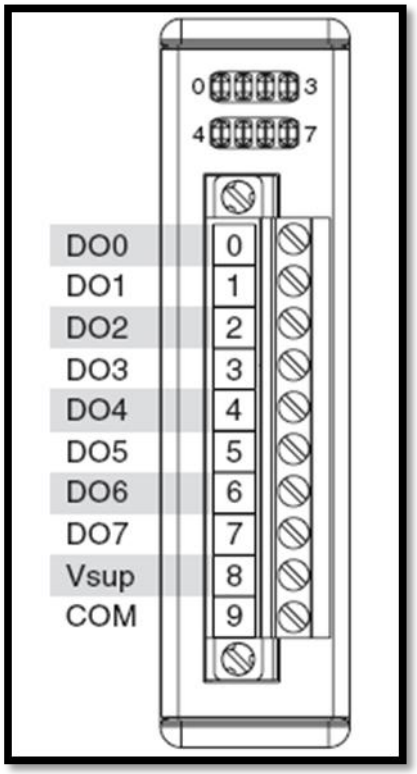


Figure 4.15: NI 9474 terminal assignment

Each channel has an LED that indicates the state of the channel. When a channel LED is glowing, the channel is on and when the LED is dark, the channel is off. We can directly connect the NI 9474 to a variety of industrial devices such as solenoids, motors, actuators, relays etc. The

devices we connect to NI 9474 should be compatible with the output specifications of the module.

Fig.4.16 given below shows how to connect an external device to NI 9474. Connect the device to the DO pin and the common of the device to the COM pin.

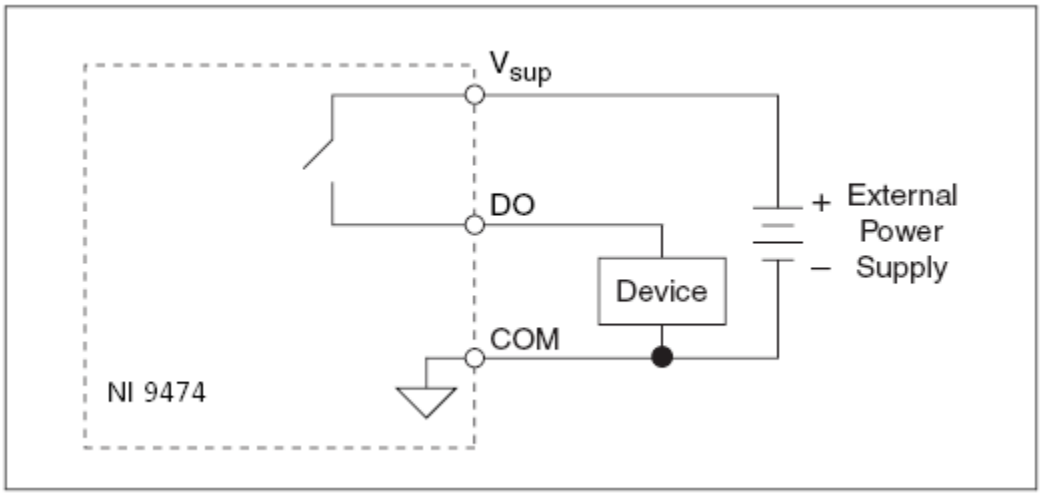


Figure 4.16: Connecting a device to NI 9474

Some of its features are:

- 8-channel, 1 µs high-speed digital output
- 5 to 30 V, sourcing digital output
- -40 to 70 °C operating range
- Extreme industrial certifications/ratings
- Hot-swappable operation

4.4.6 NI 9263: A 4-channel Analog Voltage Output Module

The NI 9263 has connections for four analog output channels and each channel has a terminal, AO, to which we can connect the positive lead of a voltage signal. The NI 9263 also has common terminals, COM, that are internally connected to the isolated ground reference of the module. [22]

Some of its characteristics are mentioned below:

- 4 simultaneously updated analog outputs, 100 kS/s
- 16-bit resolution

- Hot-swappable operation
- -40 to 70 °C operating range

Table 4 given below provides the terminal description of NI 9263:

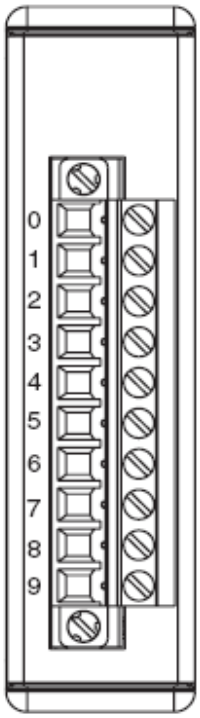
Module	Terminal	Signal
	0	AO0
	1	Common (COM)
	2	AO1
	3	Common (COM)
	4	AO2
	5	Common (COM)
	6	AO3
	7	Common (COM)
	8	No Connection
	9	Common (COM)

Table 4: Terminal description of NI 9263

In order to connect a load to NI 9263, connect the positive lead of the load to the AO terminal and the negative lead to the COM terminal as shown in fig.4.17 given below:

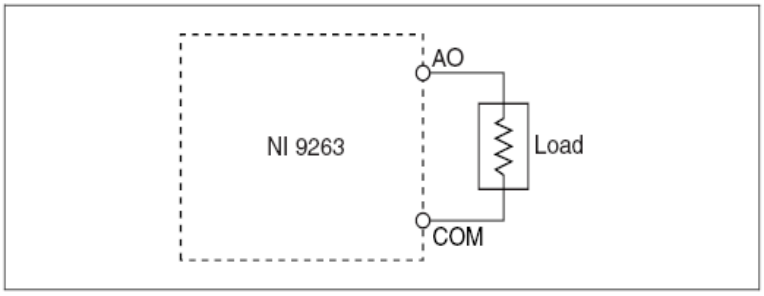


Figure 4.17: Connecting a load to NI 9263

Chapter 5

Hardware Setup and Programming

5.1 Setting up the Hardware

After configuring the chassis and installing the required software, we can now use the CompactRIO for application development. Given below are the steps for hardware setup for the temperature control application we have developed:

1) Connect the positive terminal of the 24V power supply to the V terminal of the power connector and the common to the C terminal.

Figure 5.1: NI cRIO front panel

2) With the help of Ethernet cable, connect the chassis to the Ethernet port of the PC using any of the Ethernet port on the controller front panel. Fig.5.1 given above shows the Ethernet port and power connector provided on cRIO front panel.
3) From the same 24V power supply, provide connection to the NI 9474 Digital output module as this module needs an external power supply. Connect the positive terminal of the power supply to the Vsup terminal of the module and the common to the COM terminal.

4) Now, connect the thermocouple to the NI 9211 Thermocouple Input module as shown in the fig.5.2. The figure also depicts that the module interacts with the FPGA core.

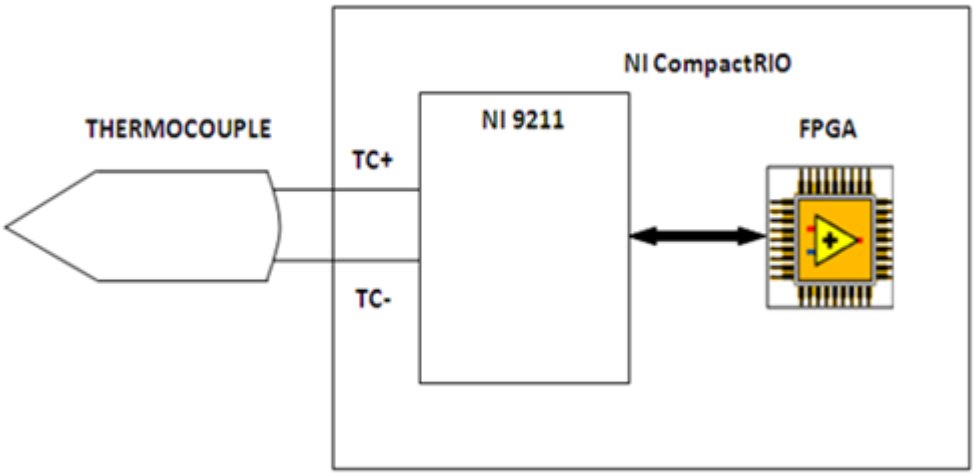


Figure 5.2: Connecting the thermocouple to NI 9211

5) Now connect the two bulbs to the NI 9474 Digital output module via relay card. Fig.5.3 and fig.5.4 shows the relay card we used and the connection diagram, respectively. One bulb is connected to channel DO and the other to D1 of the NI 9474 module.

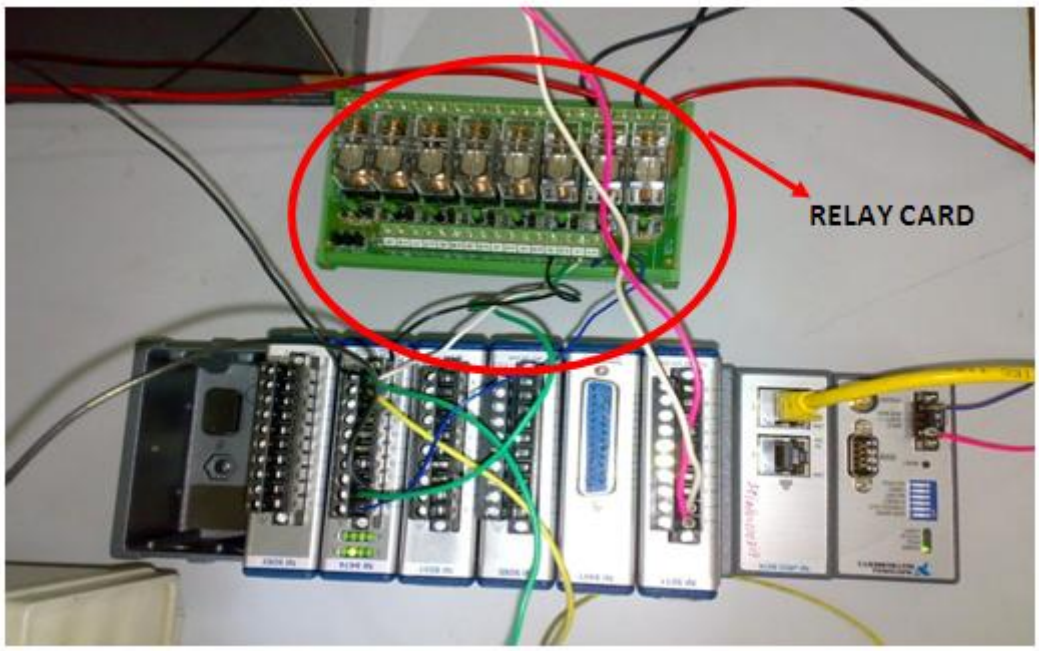


Figure 5.3: Relay card connected to cRIO

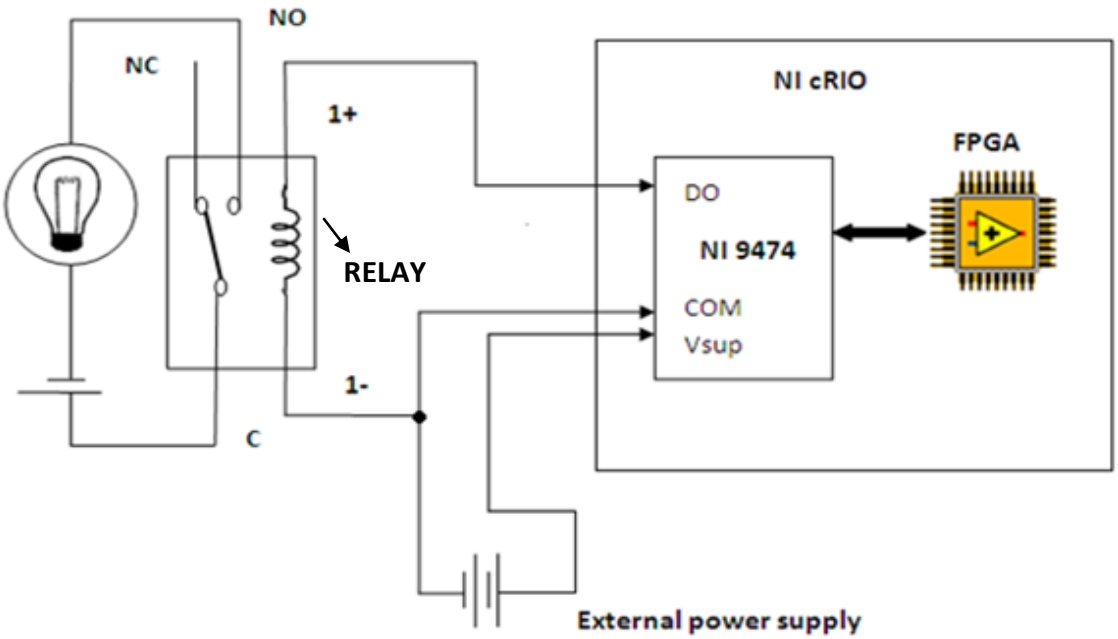


Figure 5.4: Bulb connection to NI 9474 via relay

6) The small CPU fan we are using here can operate at a voltage between 3-12V.Therefore, the fan is directly connected to the NI 9474 module. The positive terminal of the fan is connected to the D2 channel and the common terminal to the COM terminal of the module. The complete hardware setup is shown in fig.5.5 given below:

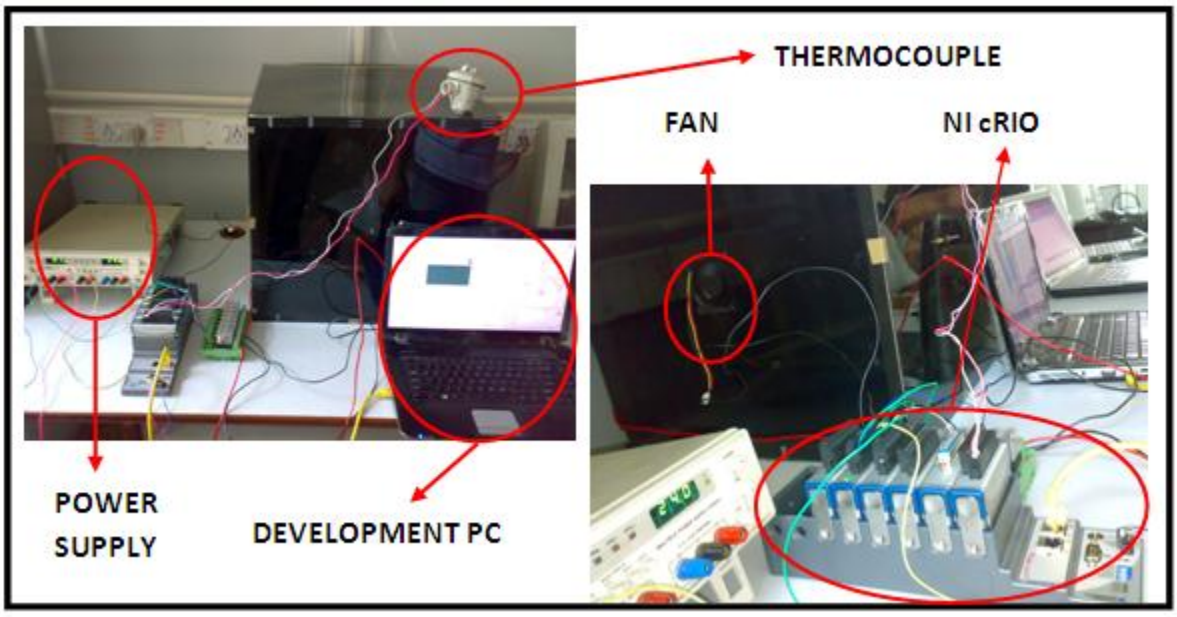


Figure 5.5: Complete Hardware setup

5.2 Getting Started with the Programming

5.2.1 Selecting the programming mode for the application

LabVIEW Real-time 8.6 introduces many powerful new features, one of which is programming the CompactRIO programmable automation controller in Scan Interface programming mode, thereby reducing development time and complexity.

Prior to the introduction of LabVIEW 8.6, the CompactRIO I/O was first accessed by programming the FPGA, and then in LabVIEW Real-Time using the LabVIEW FPGA Interface VIs. Now the user can choose among the **Scan interface mode** or the **LabVIEW FPGA interface** mode, depending upon the application.

During our thesis work, we developed our application in both the programming modes.

5.2.2 LabVIEW FPGA interface programming mode

5.2.2.1 Introduction

LabVIEW FPGA interface mode enables us to use the C series modules (I/O modules) from LabVIEW FPGA VIs. Modules that we use in LabVIEW FPGA interface mode appear directly under the FPGA target item in the **Project Explorer** window (discussed later) and the I/O channels appear as FPGA I/O items under the FPGA target. In order to access the I/O channels, we need to configure the FPGA I/O nodes in the FPGA VIs. [23]

In LabVIEW FPGA interface mode, we can use LabVIEW FPGA programming to add more flexibility and customization to our applications. To use the CompactRIO system in this mode, one must have LabVIEW FPGA module installed on the host computer.

5.2.2.2 Configuring a project for CompactRIO system

After the hardware setup, power on the CompactRIO controller. We use projects in LabVIEW to group together LabVIEW files and files not specific to LabVIEW, create build specifications, and deploy or download files to targets. When we save a project, LabVIEW creates a project file (.lvproj), which includes references to files in the project, configuration information, build

information, deployment information, and so on. We should also use a project to work with an RT, FPGA, Touch Panel, DSP, or embedded target.

Complete the following steps to configure a project:

1) Launch LabVIEW.
2) On the **Getting Started** window, under **Targets**, select **FPGA Project**. We can also select **Tools» FPGA Module» FPGA Project.**
3) Click **Go**.
4) On the **Create New LabVIEW FPGA Project** page that appears, select **CompactRIO Reconfigurable Embedded System**. Click **Next**.
5) Make sure **Discover existing system** is selected and click **Next**.
6) On the **Create New CompactRIO FPGA Project** page, select the controller you are using. Click **Next**.
7) Uncheck **Launch FPGA Wizard when finished** and click **Finish**.
8) Select **File» Save Project** and save the project as **final.lvproj**.

The Project Explorer window should look like the fig.5.6 given below:

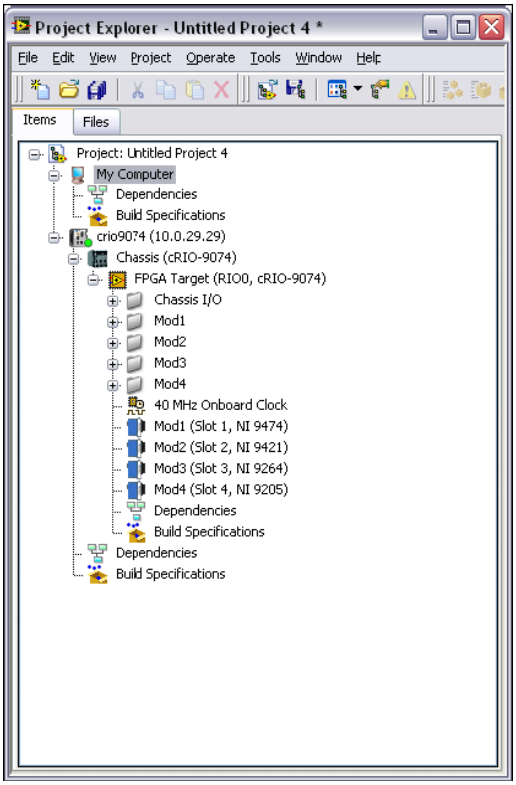


Figure 5.6: Project Explorer window in LabVIEW FPGA interface

5.2.2.3 Building our temperature control application

- **Creating the FPGA VI and adding the thermocouple module to it**

  The FPGA VI is the VI that we download to the FPGA target, which, in this application, is the CompactRIO chassis. We use the FPGA VI to read from and write to the I/O channels of C Series modules (I/O modules). Complete the following steps to create the FPGA VI and add various modules used in the application to it:

  1) Right-click the **FPGA Target** item in the **Project Explorer** window and select **New» VI** from the shortcut menu to open a blank VI.
  2) Place a while loop on the block diagram of the VI.
  3) In the **Project Explorer** window, expand the **MOD1** folder item.
  4) Drag and drop the **Mod1/TC0** item from the **Project Explorer** window to inside the While Loop on the block diagram to create an FPGA I/O Node for that item.
  5) On the block diagram, drag down the bottom edge of the **Mod1/TC0** FPGA I/O Node to add two more elements.
  6) Click the second new element and select **Mod1» Mod1/Autozero** from the shortcut menu to associate the element with the Mod1/Autozero item.
  7) Click the third element and select **Mod1» Mod1/CJC** from the shortcut menu to associate the element with the Mod1/CJC item.
  8) One by one, right-click the three elements and select **Create» Control** from the shortcut menu to create controls on the front panel.
  9) Right-click the conditional terminal at the bottom right of the While Loop and select **Create» Control** from the shortcut menu to create a **stop** control.
  10) Rename the control **Stop temp acq**.
  11) Click the **clean up diagram** button on the toolbar.
  12) Save the VI as **tempmeasure.vi**.

The block diagram should exactly look like the one given below in fig.5.7:

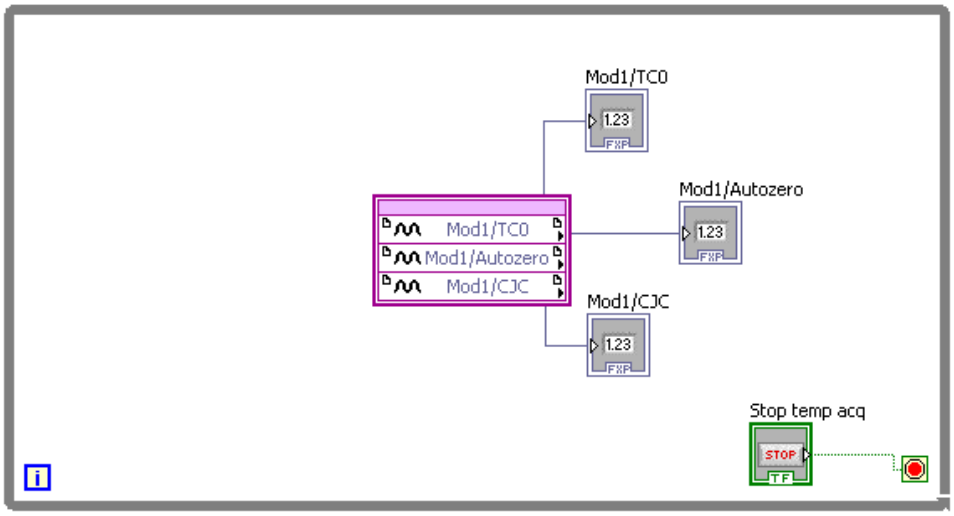


Figure 5.7: FPGA VI showing thermocouple module

- **Creating the PWM loop for Digital output module in FPGA VI**

  Pulse-width modulation varies the duty cycle of a digital voltage output to produce an analog signal range for control applications. We can use pulse-width modulation to provide analog control for digital devices such as DC motors, heaters, and lights. Complete the following steps to add pulse-width modulation to the FPGA VI:

  1) Place an additional While Loop on the block diagram below the first While Loop.
  2) In the **Project Explorer** window, expand the **Mod5** folder item.
  3) Drag and drop the **Mod5/DO0** item from the **Project Explorer** window to inside the While Loop on the block diagram.
  4) Place a Flat Sequence structure around the **Mod5/DO0** FPGA I/O Node on the block diagram.
  5) Right-click the left border of the Flat Sequence structure and select **Add Frame Before** from the shortcut menu.
  6) Place a Loop Timer VI in the new frame.
  7) On the **Configure Loop Timer** dialog box that appears, make sure **ms** and **32 Bit** are selected, then click **OK**.
  8) Place two numeric controls on the front panel and label the controls **L Pulse lamp1** and **H Pulse lamp1**.

9) Place a **Select** function to the left of the Flat Sequence structure on the block diagram.
10) Move the **L Pulse lamp1** and **H Pulse lamp1** controls to inside the While Loop, to the left of the Select function.
11) Wire the output terminal of the **L Pulse lamp1** control to the **t** input of the Select function.
12) Wire the output terminal of the **H Pulse lamp1** control to the **f** input of the Select function.
13) Wire the output terminal of the **Select** function to the input terminal of the Loop Timer VI.
14) Right-click the left border of the While Loop and select **Add Shift Register** from the shortcut menu.
15) Place a **False** Boolean constant outside the While Loop and wire it to the shift register terminal on the left side of the While Loop.
16) Wire the shift register terminal on the left to the **s** input of the **Select** function.
17) Place a **Not** function inside the While Loop, below the frame of the Flat Sequence structure that contains the **Mod5/DO0** FPGA I/O Node.
18) Wire the shift register terminal on the left side of the While Loop to the **Mod5/DO0** input of the **Mod5/DO0** FPGA I/O Node.
19) Wire the shift register terminal on the left side of the While Loop to the input terminal of the **Not** function.
20) Wire the output terminal of the **Not** function to the shift register terminal on the right side of the While Loop.
21) Right-click the conditional terminal at the bottom right of the While Loop and select **Create» Control** from the shortcut menu to create a **stop** control.
22) Name the new **stop** control **Stop lamp1 pwm**.
23) Similarly create PWM loops for the other bulb and the fan. Name the stop control for second bulb **Stop lamp2 pwm** and for the fan **Stop fan pwm**. Also on the **Configure Loop Timer** dialog box that appears for the fan PWM loop, select **µs** and **32-bit**, then click **OK**. Also change the name of numeric controls for loop timer.

24) Click the **Clean up diagram** button on the toolbar and save the VI.

25) Save the project.

26) In the **Project Explorer** window, right-click **tempmeasure.vi** and select **Compile** from the shortcut menu to compile the FPGA VI. Compiling can take from a few minutes to a few hours. In our application, it took about 6-7 minutes.

A PWM loop will look like the one given below in fig.5.8. Complete FPGA VI is given in Appendix A.

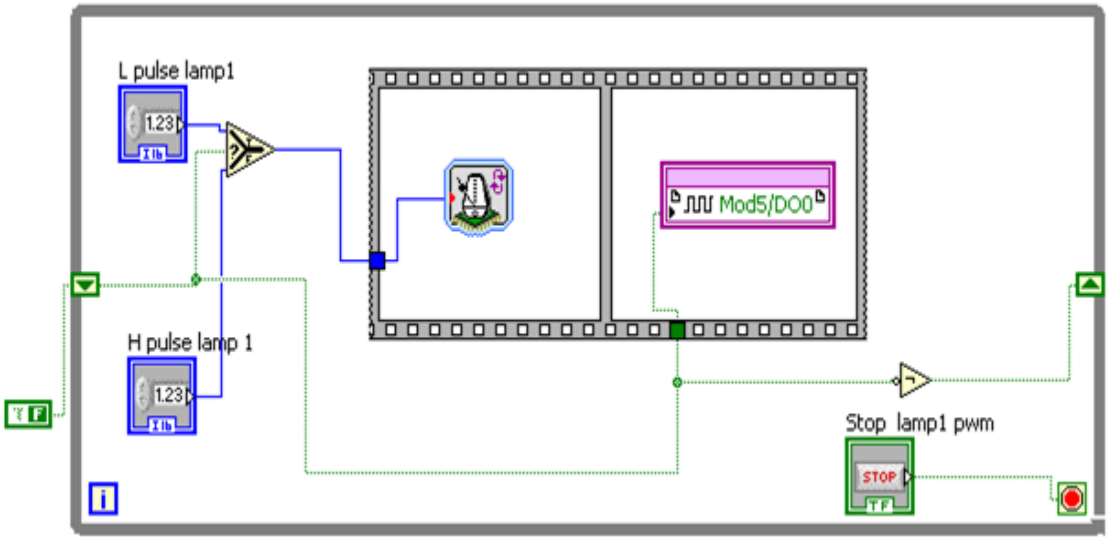


Figure 5.8: FPGA VI showing PWM loop

- **Creating the HOST VI to communicate with FPGA VI**

The host VI communicates with the FPGA VI. We can run the host VI on a Real-Time (RT) target, such as a CompactRIO controller, or on a Windows PC. The host VI uses controls and indicators on the FPGA VI front panel to transfer data between the FPGA on the RIO device and the host-processing engine. These front panel objects are represented as data registers within the FPGA. We created the Host VI on the development PC. Complete the following steps to build the host VI:

1) In the **Project Explorer** window, right-click **My Computer** and select **New» VI**.
2) Place a While Loop on the block diagram of the new VI.

3) Place an **Open FPGA VI Reference** function on the block diagram to the left of the While Loop.
4) Double-click the **Open FPGA VI Reference** function.
5) On the **Configure Open FPGA VI Reference** dialog box that appears, select the **VI** radio button.
6) On the Select VI dialog box that appears, select **tempmeasure.vi** and click **OK**.
7) Click **OK** on the **Configure Open FPGA VI Reference** dialog box.
8) Place a **Read/Write Control** function inside the While Loop on the block diagram.
9) Wire the **FPGA VI Reference Out** output of the **Open FPGA VI Reference** function to the **FPGA VI Reference In** input of the **Read/Write Control** function.
10) Right-click the **FPGA VI Reference Out** output of the **Read/Write Control** function and select **FPGA Interface Palette» Close FPGA VI Reference**.
11) Place the **Close FPGA VI Reference** function to the right of the While Loop.
12) Wire the **FPGA VI Reference Out** output of the **Read/Write Control** to the **FPGA Interface In** input of the **Close FPGA VI Reference** function.
13) Wire the **error out** output of the **Open FPGA VI Reference** function to the **error in** input of the **Read/Write Control** function.
14) Wire the **error out** output of the **Read/Write Control** function to the **error in** input of the **Close FPGA VI Reference** function.
15) Click the **Unselected** element of the **Read/Write Control** function and select **Mod1/TC0**.
16) Expand the **Read/Write Contro**l function downward until it shows all of the controls and indicators in **tempmeasure.vi**: **Mod1/TC0**, **Mod1/CJC**, **Mod1/Autozero, Stop AI/AO**, **L Pulse lamp1**, **H Pulse lamp1**, **L Pulse lamp2**, **H Pulse lamp2, L Pulse fan, H Pulse fan, Stop temp acq, Stop lamp1 pwm, Stop lamp2 pwm, Stop fan pwm**.
17) Create controls for the **L Pulse lamp1**, **H Pulse lamp1**, **L Pulse lamp2**, **H Pulse lamp2, L Pulse fan** and **H Pulse fan** elements.
18) Wire the input terminals of the **Stop temp acq, Stop lamp1 pwm, Stop lamp2 pwm** and **Stop fan pwm** element to the output terminal of the **Stop** control of the While Loop.

The block diagram of the Host VI up to this step is shown below in fig.5.9:

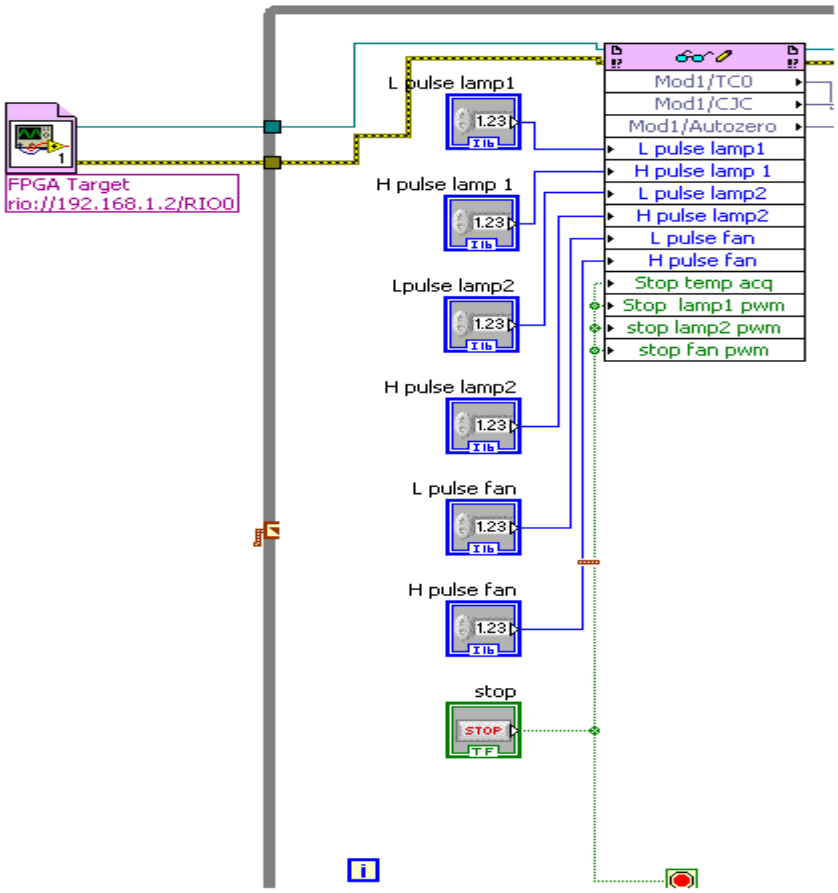


Figure 5.9: Host VI showing Read/Write Control function

19) Now we need to implement the PID control loop in the Host VI, where the temperature reading from the thermocouple will be the process variable.

20) The thermocouple module in the FPGA VI provides the data in terms of voltage. Therefore, this needs to be converted into temperature units.

21) The conversion of volts into temperature is done in the Host VI and is achieved using the **NI 9211 Convert to Temperature (Calibrated).vi**. To use this VI, Select **Select a VI** from the **Functions** palette. The **Select VI to Open** dialog box appears. Browse to Program Files\National Instruments\LabVIEW 8.6\examples\CompactRIO\Module Specific\NI 9211 and select **NI 9211 Support Files.llb**. The **File Dialog** dialog box appears. Select **NI 9211 Convert to Temperature (Calibrated).vi** and click the **OK** button.

22) Connect the output terminals of **Mod1/TC0**, **Mod1/CJC** and **Mod1/Autozero** to the inputs of the conversion VI. The other inputs of the conversion VI are **Type of thermocouple** (in this case, K-type) and **Temperature units** (we used Celsius). The output of this VI provides the linearized temperature that we can now use as process variable for the PID controller.

23) The PID control toolkit is installed with the LabVIEW and we can use the PID VIs to develop various control applications. We can combine these VIs with LabVIEW math and logic functions to create block diagrams for real control strategies.

The fig.5.10 given below shows the implementation of the PID control loop in Host VI:

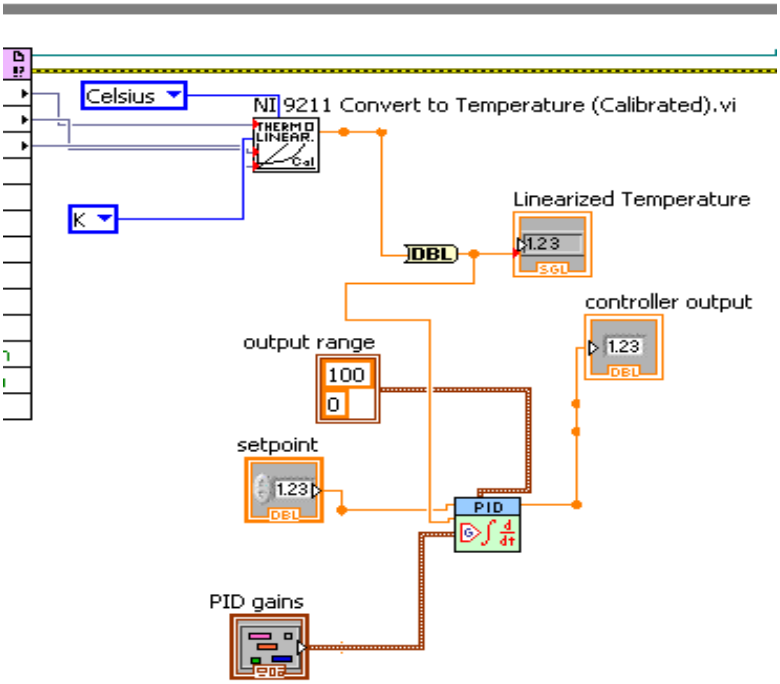


Figure 5.10: Host VI showing the PID.vi

24) The VI used here, from the PID control toolkit is called as **PID.vi**. Apart from process variable, the other inputs of this VI are **setpoint**, **PID gains**, **output range** and **dt** (control-loop cycle time). **PID gains** is a cluster of three values- proportional gain, integral time and derivative time.

25) By default, the PID controller output range is -100 to 100. We used the output range of 0 to 100, depending upon which the process is controlled.

26) After making all the necessary connections in the block diagram, click the **clean up diagram** button on the toolbar and save the VI as **tempmeasurehost.vi**. Save the project.

27) We can use this Host VI to control our application. For running and testing the Host VI, first set the values for setpoint and PID gains. Now run the VI. We can notice the response on Process variable Vs time waveform chart. The complete Host VI is given in Appendix B and various results are discussed later in this chapter.

- **Communication between FPGA VI and Host VI**

  With Programmatic FPGA Interface Communication, we programmatically monitor and control an FPGA VI with a Host VI running on host computer. When we use Programmatic FPGA Interface Communication, the FPGA VI runs on the FPGA target and the host VI runs on the host PC. We make use of FPGA Interface Functions, which are available for a host VI on Windows or on RT target, to communicate with the FPGA VI and for performing the required functions. If the host VI is running on the RT target, then using the Programmatic FPGA Interface Communication, the host VI can communicate with the FPGA VI and we can use a Windows PC to communicate with the RT target.

  Another method of data transfer between the Host VI and the FPGA VI is Direct Memory Access (DMA). This method is used mainly when we do not want to transfer each data point immediately. This is mainly used for data logging purposes. Programmatic FPGA Interface Communication is ideal for frequent data transfers and hence commonly used for control and simulation purposes.

5.2.3 Scan interface programming mode

5.2.3.1 Introduction

Scan interface mode, a new feature introduced by LabVIEW 8.6, enables us to use C series modules directly from LabVIEW Real-time. Modules that we use in this mode appear directly under the chassis item in the **Project Explorer** window and the I/O channels appear as I/O variables under respected modules. To use these I/O variables, we can simply drag and drop them to LabVIEW Real-time VIs. [23]

In Scan interface mode, there is no need to do any LabVIEW FPGA development or program communication between FPGA and Host VIs. Also there is no need to wait for VIs to be compiled to the FPGA before running them. In this mode, LabVIEW programs the FPGA on the CompactRIO target to work with the variables. Fig.5.11 given below depicts the CompactRIO Scan mode architecture:

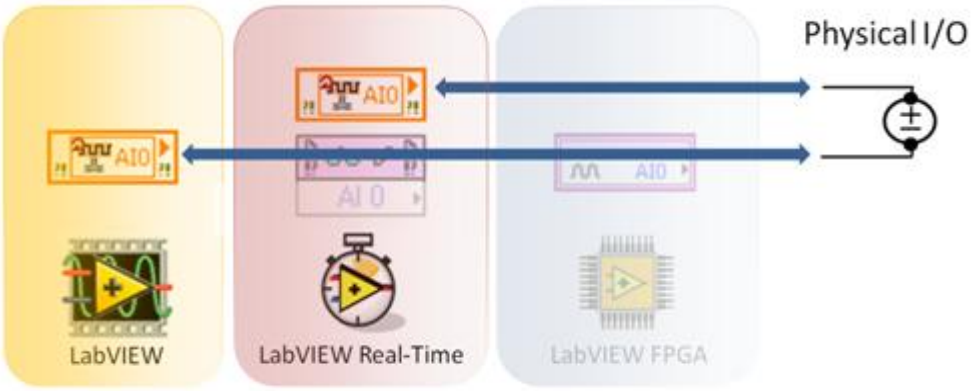


Figure 5.11: CompactRIO Scan mode architecture

5.2.3.2 Configuring a project for CompactRIO system

Complete the following steps to configure a project in Scan Interface mode:

1) Launch LabVIEW.
2) Click the **Empty Project** link in the **Getting Started** window to display the **Project Explorer** window.
3) Right-click the top-level project item in the **Project Explorer** window and select **New» Targets and Devices** from the shortcut menu to display the **Add Targets and Devices** dialog box.
4) Select the **Existing target or Device** radio button.
5) Expand **Real-Time CompactRIO**.
6) Select the CompactRIO controller to add to the project and click **OK**.
7) The **Select Programming Mode** dialog box appears. Select **Scan interface** to put the system into Scan interface mode.
8) Click **Continue**.

9) Click **Discover** in the **Discover C Series Modules?** dialog box that appears. LabVIEW adds items for the controller, the chassis, and all installed C series modules to the project. LabVIEW also adds I/O variables to the project for all installed C series module I/O channels.

10) When LabVIEW finishes discovering hardware, select **File» Save Project** and save the project as **scan1.lvproj**.

The Project explorer window should look like the one given below in fig.5.12:

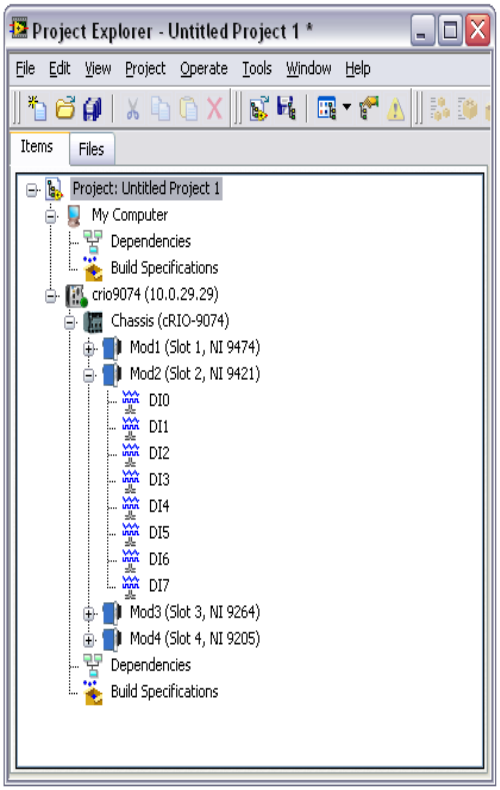


Figure 5.12: Project Explorer window in Scan Interface mode

5.2.3.3 Building our temperature control application

- **Creating the VI**

1) Right-click the controller item in the **Project Explorer** window and select **New» VI** from the shortcut menu to open a blank VI.
2) Place a While Loop on the block diagram of the VI. We can also use Timed loop instead of While loop which will execute the sub diagrams at the rate we specify. But it will not work if

the time required to execute the sub diagram is more than the period we specified for the timed loop.

- **Configuring the Digital Output module for PWM**

Complete the following steps to configure the DO module for PWM in scan interface mode:

1) Right-click the DO module item (**Mod5** in our project) in the **Project Explorer** window and select **Properties** from the shortcut menu to display the **C Series Module Properties** dialog box.
2) Select **Specialty Digital Configuration** in the Category list.
3) Click **Pulse-Width Modulation** in the **Specialty Mode** listbox.
4) Under Channels, make sure that PWM0 is highlighted.
5) Select the required value for frequency. The **C Series Module Properties** dialog box should look like the fig.5.13 given below:

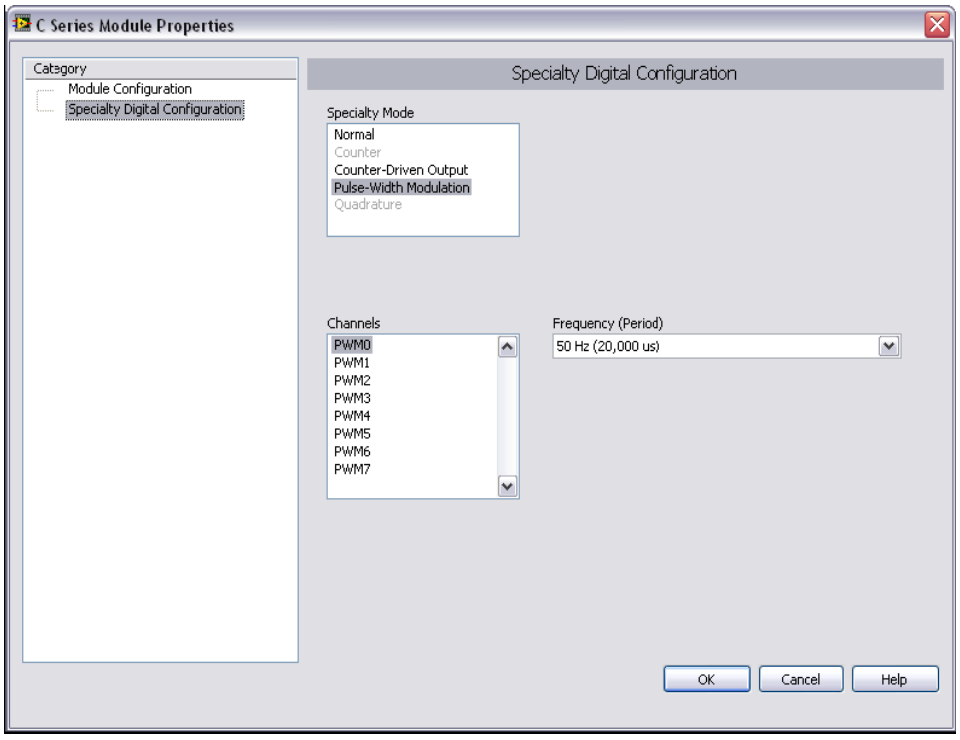


Figure 5.13: C Series Module Properties dialog box for NI 9474

6) Click **OK**.

7) Expand the DO module item in the **Project Explorer** window to see the output variable items for the module channels. We can see that LabVIEW has changed all the DO output variables to PWM output variables.

- **Adding the PWM to VI**

1) To add this module to the VI, simply drag and drop the **PWM0** output variable from the **Project Explorer** window to inside the While loop on the block diagram.
2) By right clicking the PWM0 input of PWM output variable on the block diagram, we can create either a control or a constant depending upon the application requirements.

- **Configuring the Thermocouple input module**

1) Right-click the Thermocouple module item (**Mod1** in our project) in the **Project Explorer** window and select **Properties** from the shortcut menu to display the **C Series Module Properties** dialog box.
2) Module Configuration opens up. In the **Thermocouple Type** listbox, change the type of thermocouple to type K for Channel 0.
3) In the **Measurement Units** listbox, select the units as Degree Celsius. The **C Series Module Properties** dialog box should look like the fig.5.14 given below:

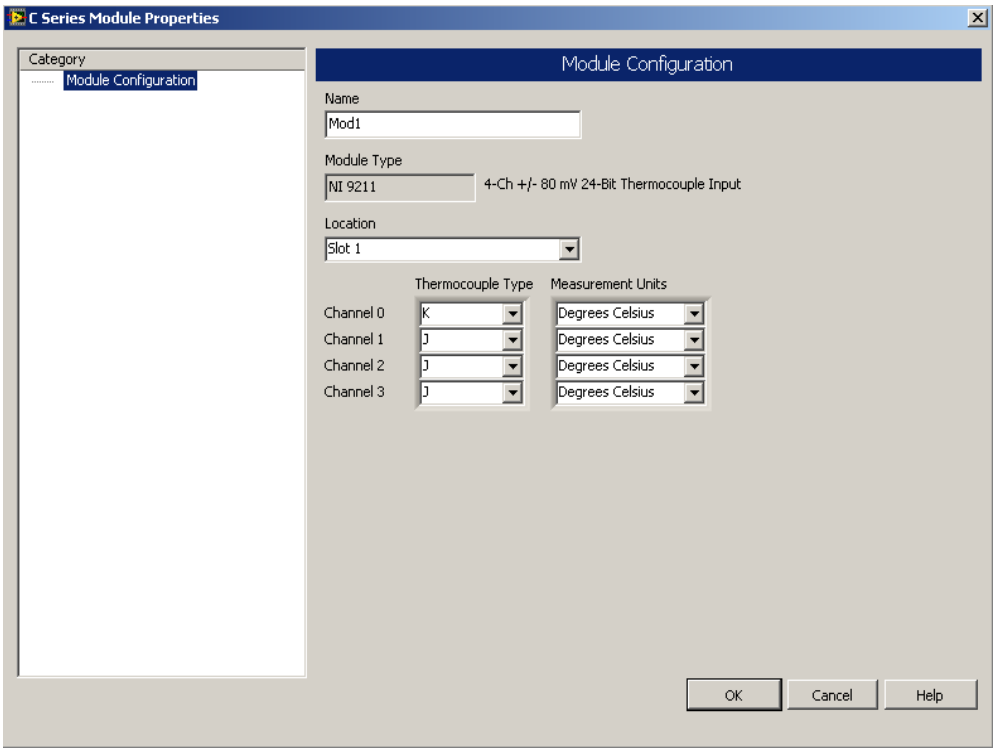


Figure 5.14: C Series Module Properties dialog box for NI 9211

4) Click **OK**.

- **Adding the thermocouple input variable to the VI**

  1) Expand the thermocouple module item in the Project Explorer window and simply drag and drop the AI0 input of thermocouple input variable.
  2) By right clicking the AI0 output of thermocouple input variable on the block diagram, select Create» Indicator from the shortcut menu to create an indicator on the front panel which displays the data in temperature units.

- **Completing the VI by adding the PID control loop**

  1) Right click on the block diagram, and select the Real-time module.
  2) Go to the **Function Blocks** palette and select **Control** palette. Select the **PID** function block and it will appear on the block diagram of the VI.
  3) Apply all the required inputs like setpoint, process variable (temperature from thermocouple input module), PID gains.
  4) Double click on the **PID** function block to open the **PID Function Block Properties** dialog box. Here we can select the terminals we want to include in the PID function block. We can also change the values of the terminals.
  5) Change the default value of **output low** terminal to 0. The box should look like the one given below in fig.5.15:

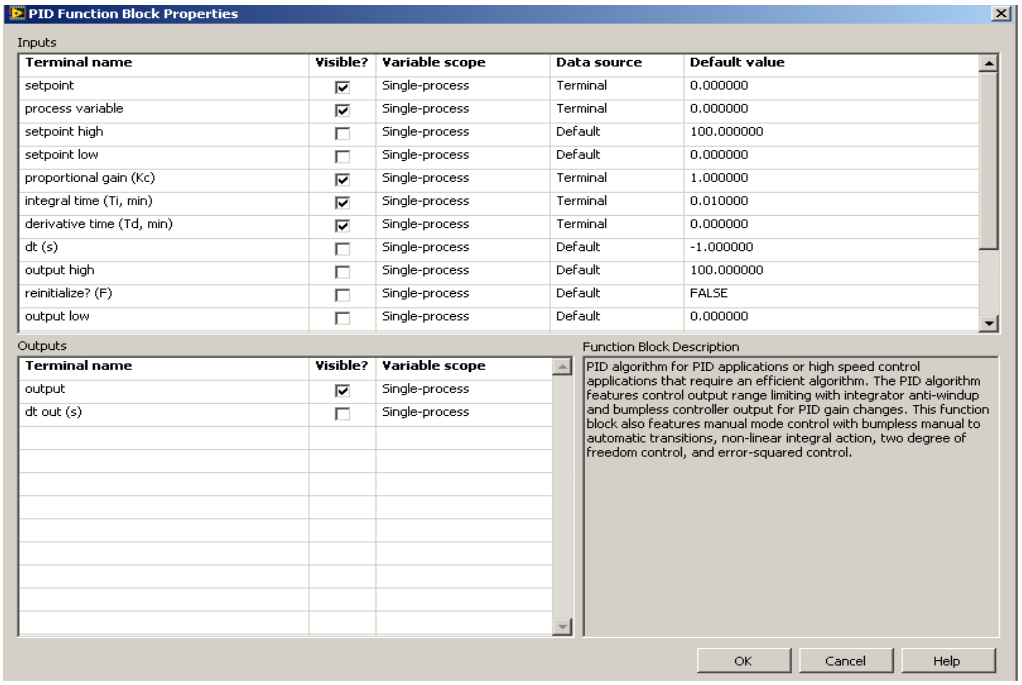


Figure 5.15: PID Function Block Properties dialog box

6) Click **OK**. This will display the controller output in the range 0-100.
7) Right click on the output terminal of the **PID** function block and select **Create» Indicator** from the shortcut menu to create an indicator on the front panel which displays the PID controller output. This output has been used to control our process. The VI showing various I/O variables and PID function block is shown in Fig. 5.16. Complete VI developed in Scan interface mode is given in Appendix C.
8) After making all the necessary connections in the block diagram, click the **clean up diagram** button on the toolbar and save the VI as **final code.vi**. Save the project.
9) Run the VI after setting the values for setpoint and PID gains. LabVIEW deploys the VI, and all modules and I/O variables the VI uses, to the controller. We can observe the response on a Process variable Vs Time waveform chart. Various results have been discussed in next section.

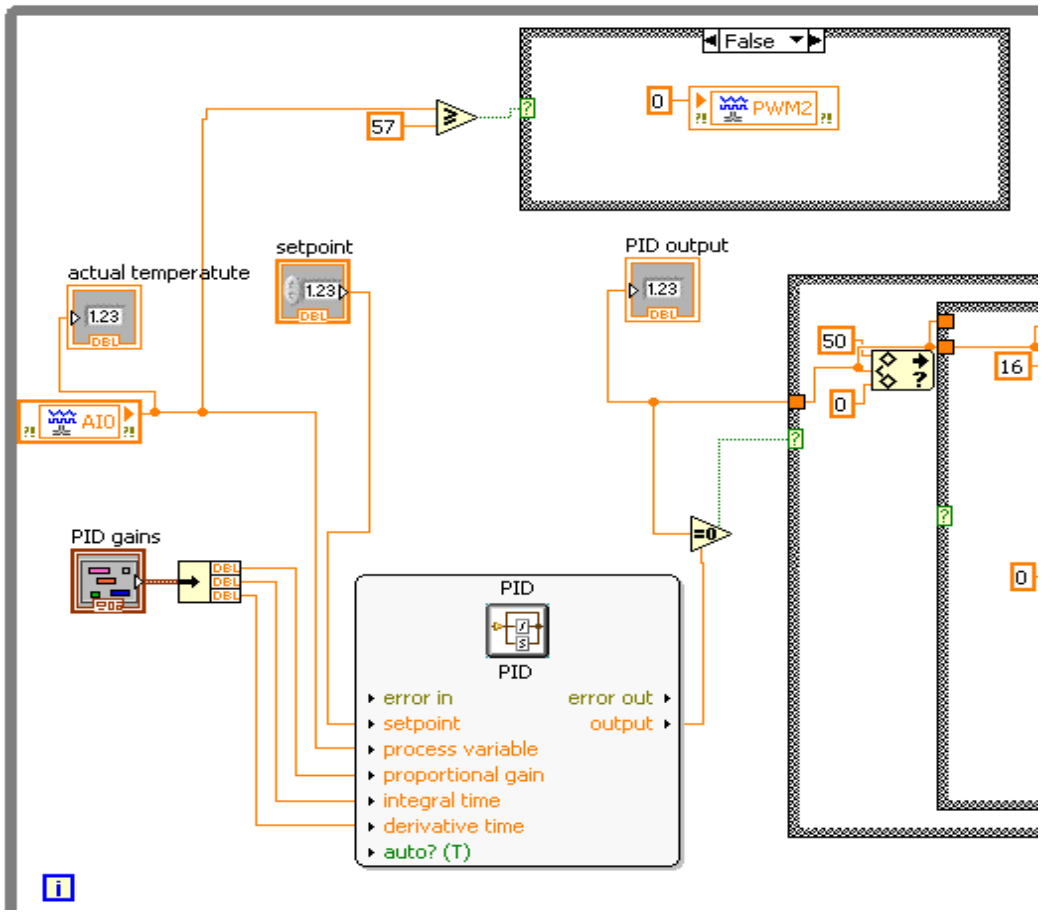


Figure 5.16: VI showing I/O variables and PID block in Scan interface mode

5.3 Results and Discussion

The requirement was to control the temperature of a chamber; hence, we fabricated one using the acrylic sheets. To increase the temperature upto the desired level, two 200W bulbs are used and temperature control is achieved by switching the bulbs ON & OFF periodically, where the controller decides the duty cycle. A simple fan has been used for cooling in emergencies. A K-type thermocouple has been used as a temperature sensor. FPGA based NI CompactRIO, along with two I/O modules has been used as a control and acquisition system. The thermocouple input module (NI 9211) has been used to acquire the temperature of the chamber and the digital output module (NI 9474) has been used to drive the bulbs and fan. Since the NI 9474 gives an output of 24V, the bulbs have been connected to it via a relay card.

PID control algorithm has been implemented using the PID control toolkit available with the LabVIEW 8.6 full development system. The PID controller parameters i.e. Proportional gain (K_c), Integral time (T_i) and the Derivative time (T_d) have been manually tuned for optimal control.

LabVIEW Real-time 8.6 supports two programming modes for developing an application around the RIO hardware – LabVIEW FPGA interface mode and Scan interface mode. We have developed our application using both the programming modes and obtained the results using Temperature Vs Time waveform chart.

The results obtained with both the programming modes are not appreciably different and are shown in the next sections. The results have been achieved for a setpoint of 52°C.

5.3.1 Results with LabVIEW FPGA interface mode

Firstly, we have obtained the results in LabVIEW FPGA interface mode. PID parameters have been tuned manually and given below are the results at various values.

Fig.5.17depicts the response at the following values of tuning parameters:

- Proportional gain (K_c) = 10.0
  Integral time (T_i) = 4.0 min (240 s) or Integral gain (K_i) = 0.041
  Derivative time (T_d) = 0 or Derivative gain (K_d) = 0

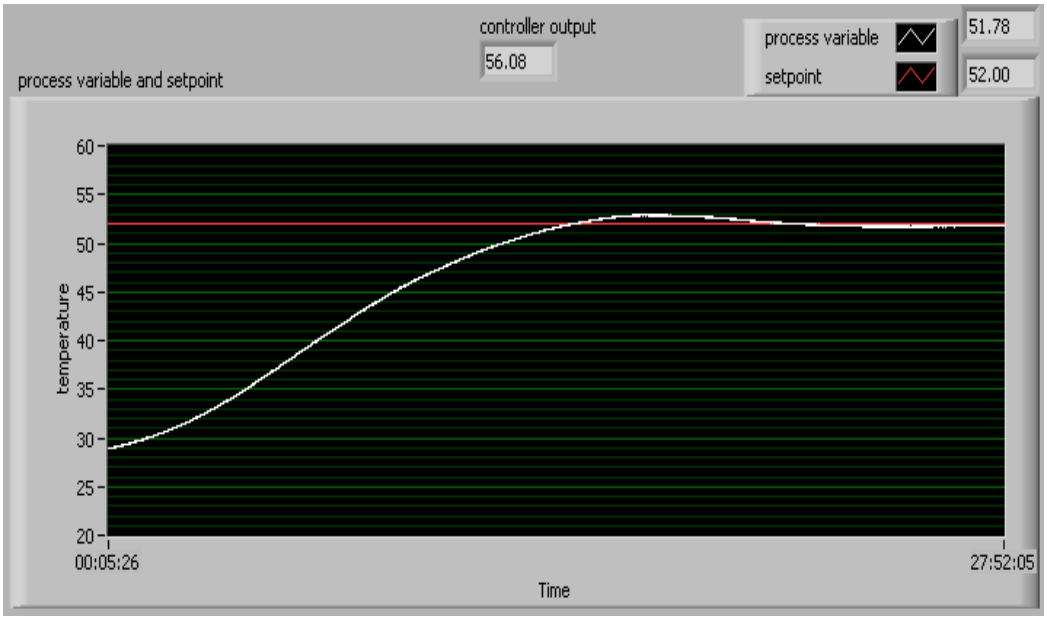


Figure 5.17: Response in LabVIEW FPGA interface mode for $K_c = 10$

Now for the same values of the parameters given above, we applied a step change and got the response given in fig.5.18, fig.5.19 and fig.5.20 given below:

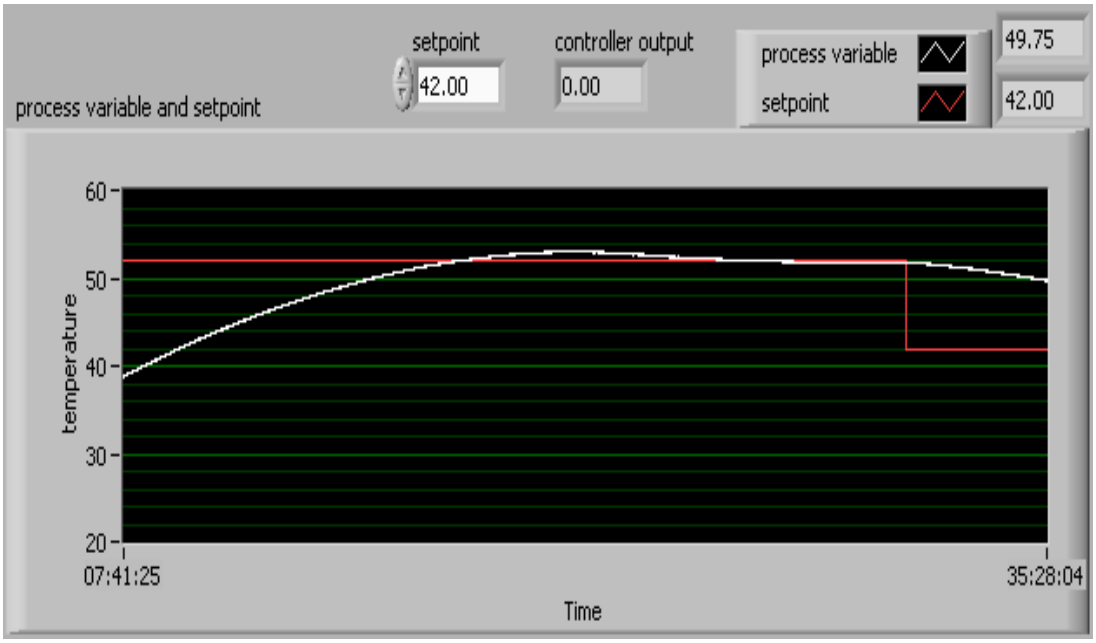


Figure 5.18: Response for a step change (part1)

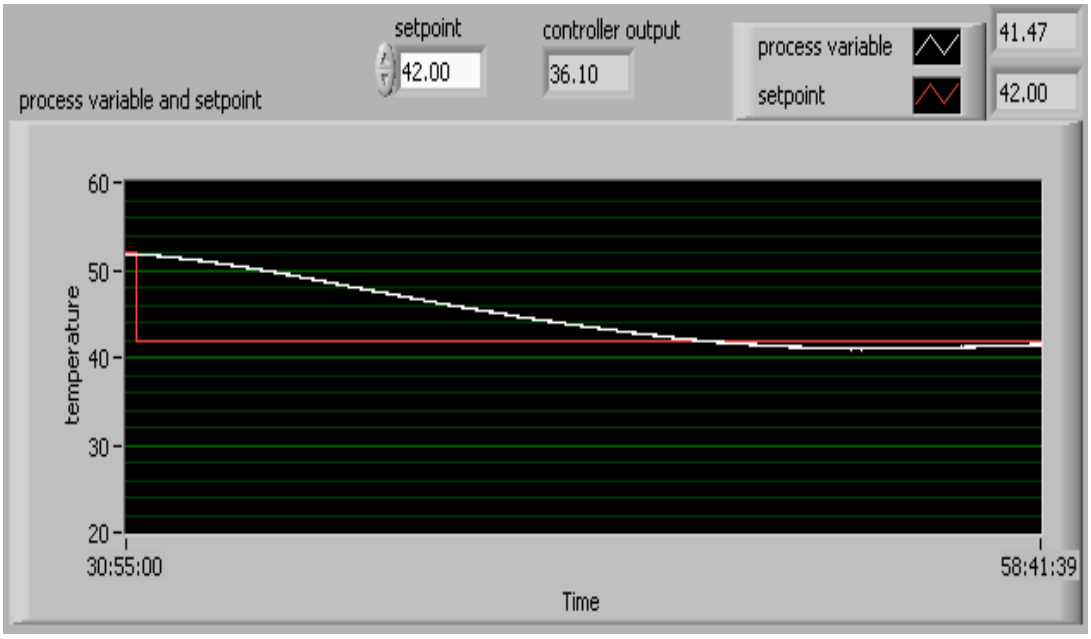


Figure 5.19: Response for a step change (part2)

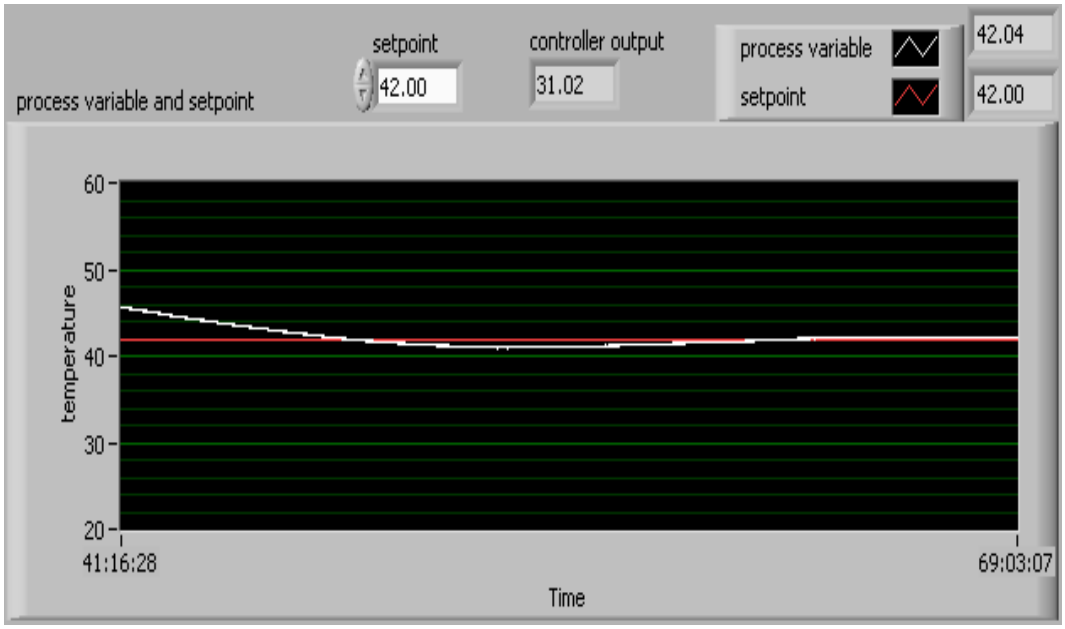


Figure 5.20: Response for a step change (part3)

5.3.2 With Scan Interface mode

Fig.5.21 and fig.5.22 depicts the response obtained with Scan interface programming mode at the following values of PID tuning parameters:

- Proportional gain (K_c) = 10.0

  Integral time (T_i) = 4.0 min (240 s) or Integral gain (K_i) = 0.041

  Derivative time (T_d) = 0 or Derivative gain (K_d) = 0

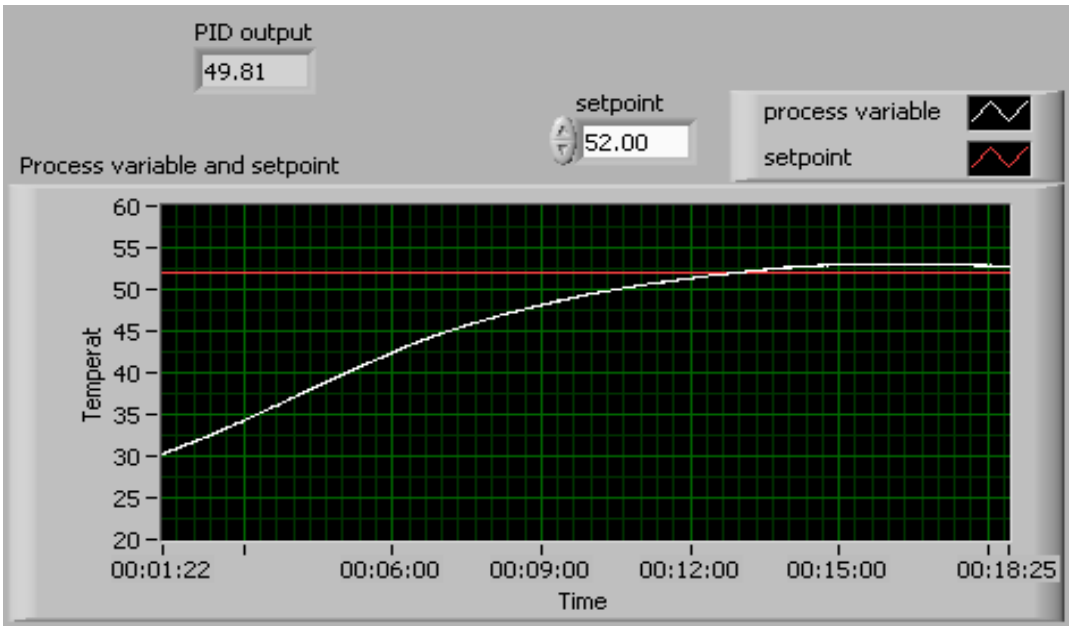


Figure 5.21: Response in scan interface mode at K_c = 10 (part1)

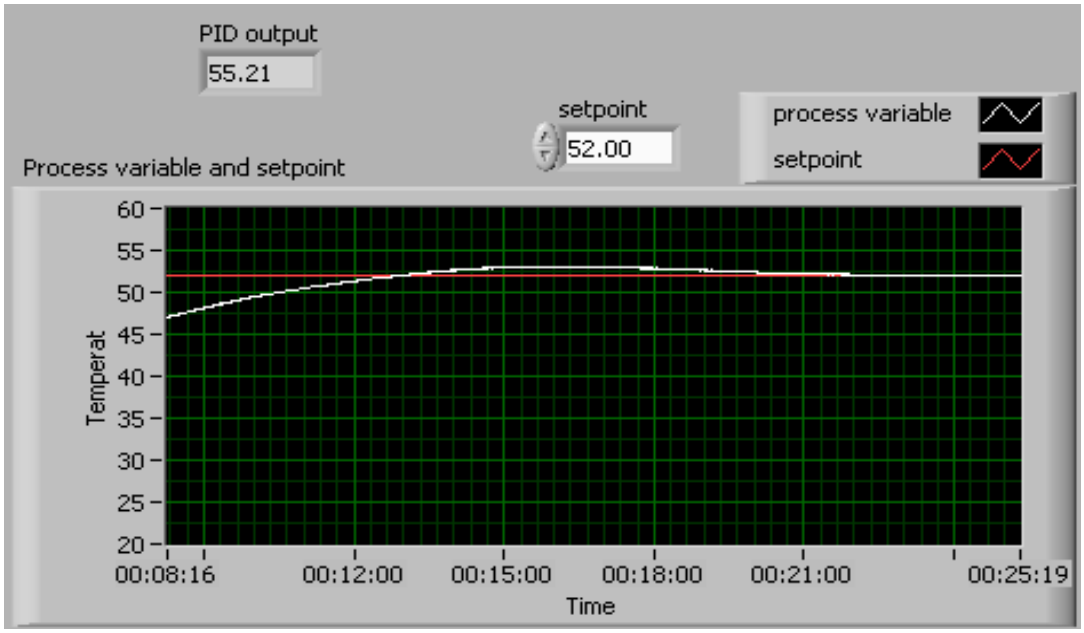


Figure 5.22: Response in scan interface mode (part2)

5.4 Encountered problems

- Communication between LabVIEW and the FPGA target is not an easy task as it seems to a new user. One is forced to explore all the function available in the FPGA module, which proves to be a time consuming task. To connect the FPGA VI to the Host VI, three functions in the LabVIEW have been used. The Open FPGA VI Reference function opens the specified VI, which means that the VI is activated to be used in Host VI. The Read/Write Control function enables to read from and write to the FPGA VI i.e. transferring data between the FPGA VI and the Host VI. The final function, Close FPGA VI Reference, is required to stop the FPGA VI from running whenever we want.
- Programming in LabVIEW FPGA interface is a tedious task to perform. The thermocouple module provides data in terms of voltage in the FPGA VI. This needs to be converted to temperature units in order to use the temperature as process variable in our application. The conversion has been performed in the Host VI using the **NI 9211 Convert to Temperature (Calibrated).vi** that we discussed in section 5.2.2.3. We also have to program the digital output module to generate the PWM, which also proves to be a time consuming task especially if one needs to provide different duty cycles to different channels of the same module. In that case, one needs to develop different PWM loops for all channels.
- To use PID function blocks, one needs to go through the PID control toolkit user manual to understand their functionality. We tried to use the **PID Autotuning.vi**, available with PID control toolkit, using which; we need not perform manual tuning of PID parameters. It employs Ziegler-Nicolas method to tune the PID parameters. However, it does not provide good results with slow processes like the one we have developed. Therefore, we switched to general **PID.vi** and performed manual tuning.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

The construction of the programs in both the programming modes has been a troublesome road. The main reason was the lack of knowledge in LabVIEW FPGA module and the NI CompactRIO. As the project went on and the understanding grew larger, the pieces fell into place. It must be mentioned that the FPGA device has worked with no problems whatsoever. It is very reliable and the only thing that can go wrong seems to be mistakes done by the programmer himself.

We successfully implemented the application in both the programming modes, but since our application does not require much of customization and also the results obtained in both the modes are not appreciably different, it is better to develop the application in Scan Interface programming mode. In scan interface mode, there is no need to wait for VIs to be compiled to the FPGA before running them. LabVIEW FPGA interface is used for high performance requirements, such as high-speed PID control loops (more than 1 KHz) or analog streaming at nearly 1 MHz.

6.2 Future scope

The process developed in this project is working satisfactorily; still some things could make it work even better.

Replacing the heating element

If the process is slow, then the problem does not lie in the FPGA, but the sensors and actuators connected to the I/O modules. Our process is slow mainly because of the heating element we have used. The system can be improved by simply using some other heating element instead of normal bulbs like infrared bulb as it heats up fast.

Adding a humidity sensor

We can also modify the system by simply adding a humidity sensor and controlling the humidity inside the chamber along with the temperature. The modified chamber can then be used for variety of applications like egg incubator, incubator for microbial growth etc.

Following a temperature profile

We can also modify our program to follow a temperature profile rather than a fixed value. PID control toolkit also contains a Gain scheduling VI that can be used in such cases. Using this VI, we can apply different sets of PID parameters to different regions of temperature profile.

References

1) Michael Barr. "*Embedded Systems Glossary*". Netrino Technical Library.

2) Embedded.com - Under the Hood: Robot Guitar embeds auto tuning

3) Adam Osborne, "*An Introduction to Microcomputers*" Volume 1, Basic Concepts, 2nd Edition, Osborne-McGraw Hill, Berkely California, 1980.

4) A. John Anderson (1994). *Foundations of Computer Technology*

5) "*Introduction to FPGA technology: Top five benefits*" available at http://zone.ni.com/devzone/cda/tut/p/id/6984

6) "*Module 4: Design of embedded processors, Lesson-20, Field programmable gate arrays and applications*", Version 2, IIT, Kharagpur available at nptel.iitm.ac.in/courses/ Webcoursecontents/ IIT%20Kharagpur/ Embedded%20systems/Pdf/Lesson-20.pdf.

7) NI CompactRIO concepts manual

8) C.Dase, J.S.Falcon, B.MacCleery "*Motorcycle control prototyping using an FPGA-based embedded control system*", Control systems magazine, IEEE Volume 26, Oct 2006, Pg: 17-21.

9) R van Zanten "*Wind tunnel control by a CompactRIO*" available at http://www.citengineering.com/downloads/pdf/TUDelft_Eng.pdf

10) S Gautam, N Verma, S Arora, K Krishan, S Asgotraa "*Adaptive control braking system using LabVIEW*" available at http://digital.ni.com/worldwide/india.nsf/87e62f4c89ea9df9862564250075e6e4/e58b63047ebab6a8862573700050c030/$FILE/Adaptive%20Braking%20System%20using%20Labview.pdf.

11) Ayaho Miyamoto "*IT-based Bridge Health Monitoring"*, The 23rd International Technical Conference on Circuits/Systems, Computers and Communications, July 2008, Pg: SP-9 – SP-14

12) Petr Bilik, Ludvik Koval, Jiri Hajduk "*CompactRIO Embedded System in Power Quality Analysis*", Proceedings of the International Multiconference on Computer Science and Information Technology, Oct 2008, Pg: 577-580

13) H.G.Ramos, O.Postolache "*Virtual Instrumentation and Reconfigurable Technology on Water Quality Sensor Calibration System*", Proceedings of The International Workshop on Marine Technology- Martech, Spain, Nov 2007

14) "*System to Remotely Transport and Deploy an Unmanned Helicopter*" available at http://www.pages.drexel.edu/~mgp27/deliverables/final%20report.pdf

15) Chia-Hua Hsu, Huai-Yuan Hsu, Hsin-Yi Wang, Tzu-Chien Hsiao "*Wheelchair Direction Control by Acquiring Vocal Cords Vibration with Multivariate Analysis*", 4th European Conference of the International Federation for Medical and Biological Engineering, Vol-22, Nov 2008, Pg: 1839-1842

16) NI CompactRIO 9072/9074 Operating Instructions and Specifications manual

17) NI 9211 Operating Instructions manual

18) NI 9401 Operating Instructions manual

19) NI 9265 Operating Instructions manual

20) NI 9201 Operating Instructions manual

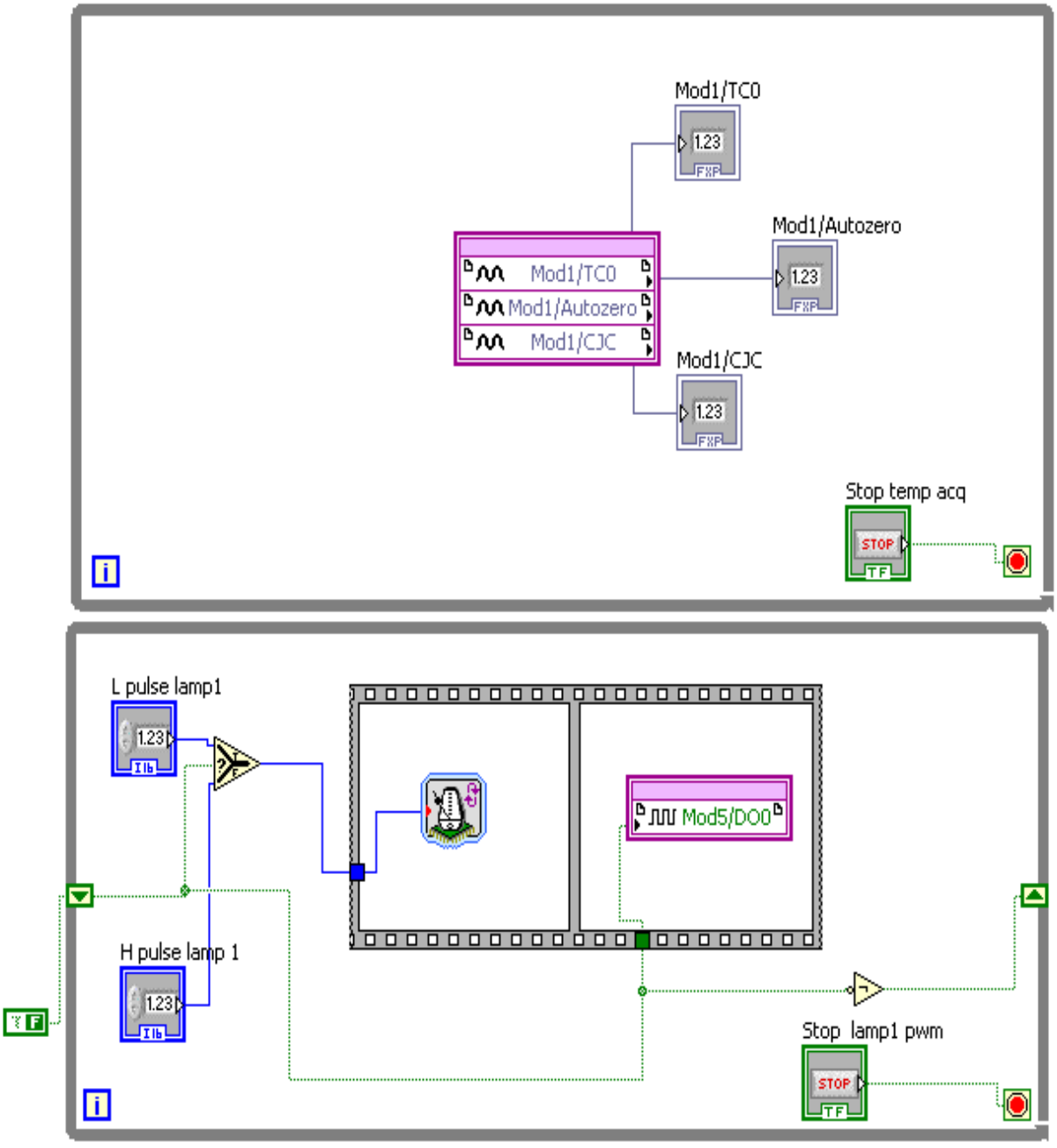
21) NI 9474 Operating Instructions manual

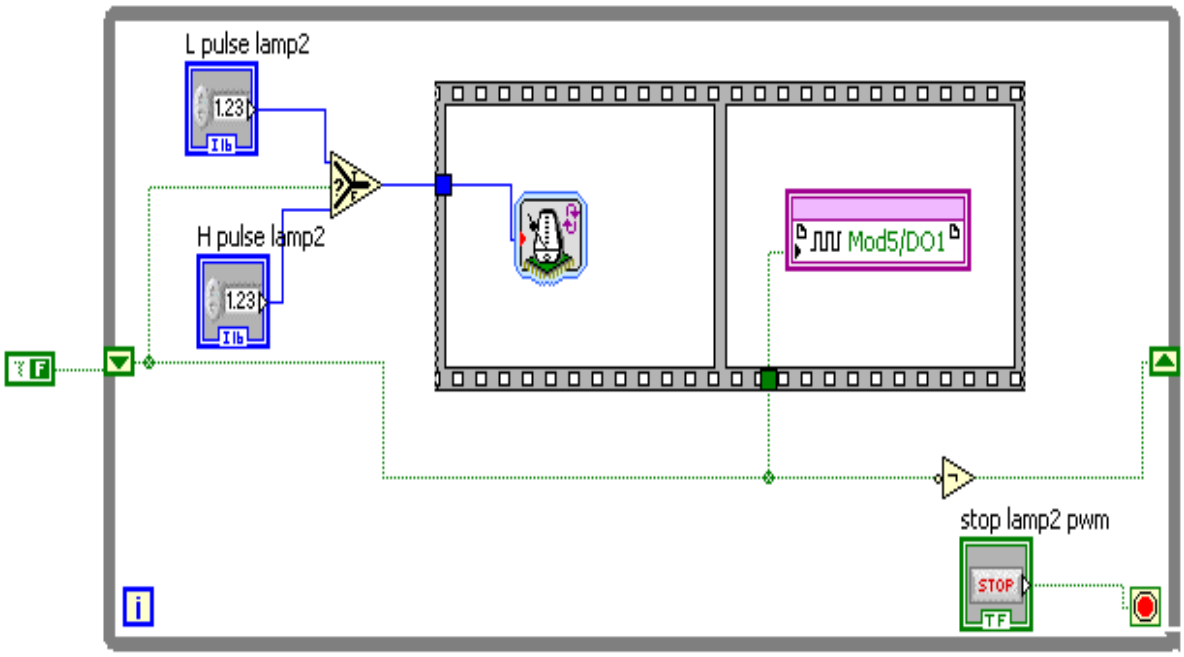
22) NI 9263 Operating Instructions manual

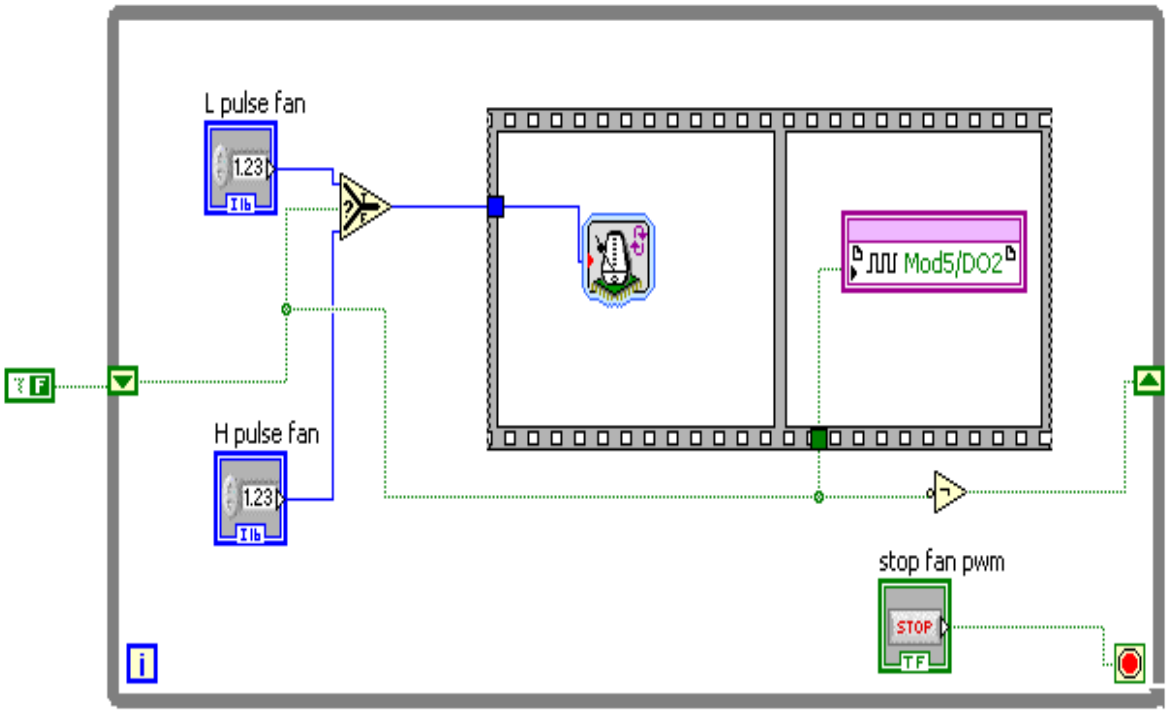
23) NI Getting Started with CompactRIO and LabVIEW manual

Appendix A – FPGA VI

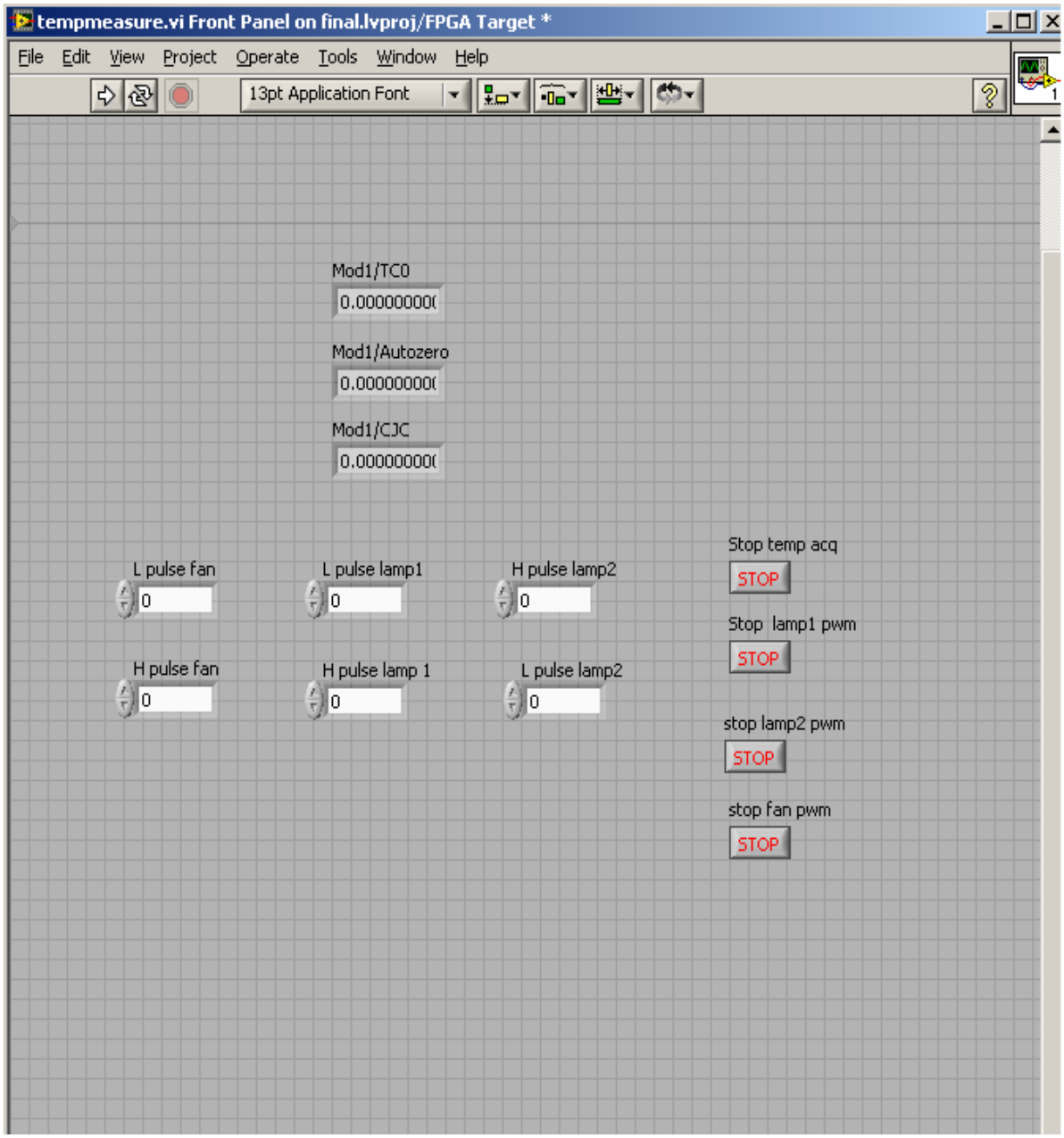
- **Block Diagram**




L pulse lamp2
1.23
I16
H pulse lamp2
1.23
I16
Mod5/DO1
stop lamp2 pwm
STOP
TF

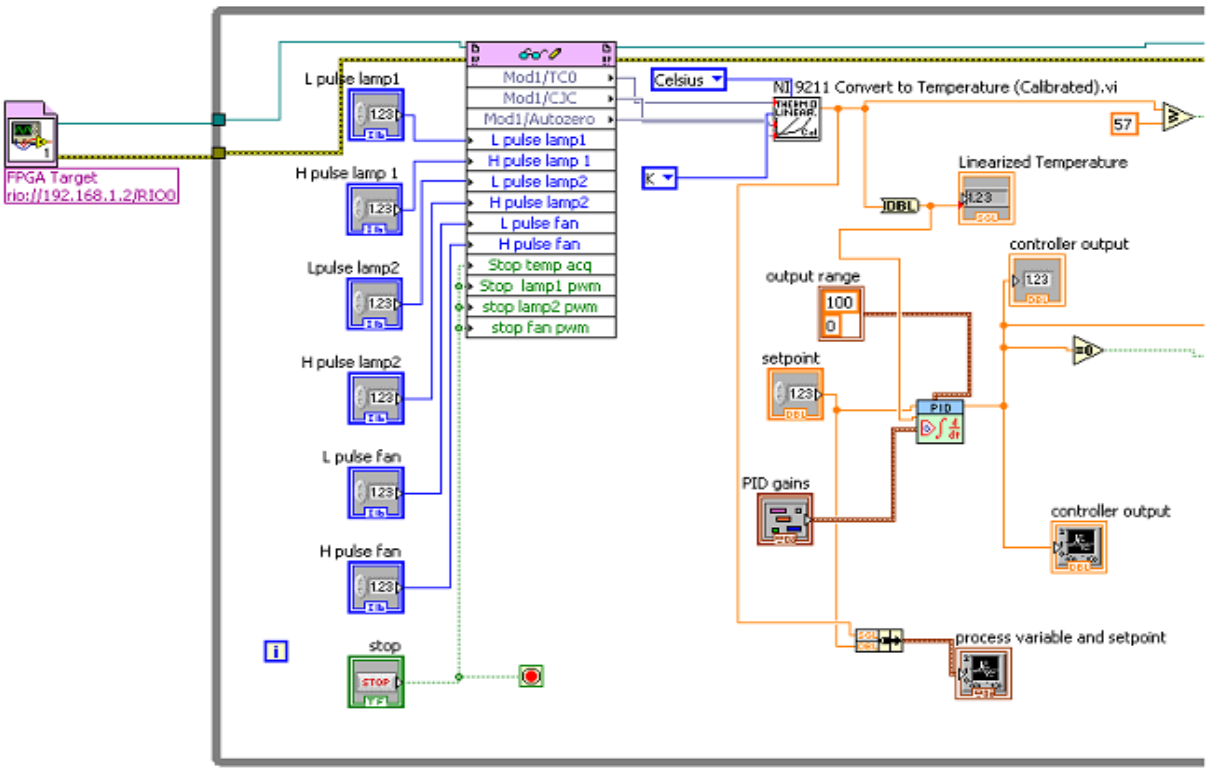

L pulse fan
1.23
I16
H pulse fan
1.23
I16
Mod5/DO2
stop fan pwm
STOP
TF

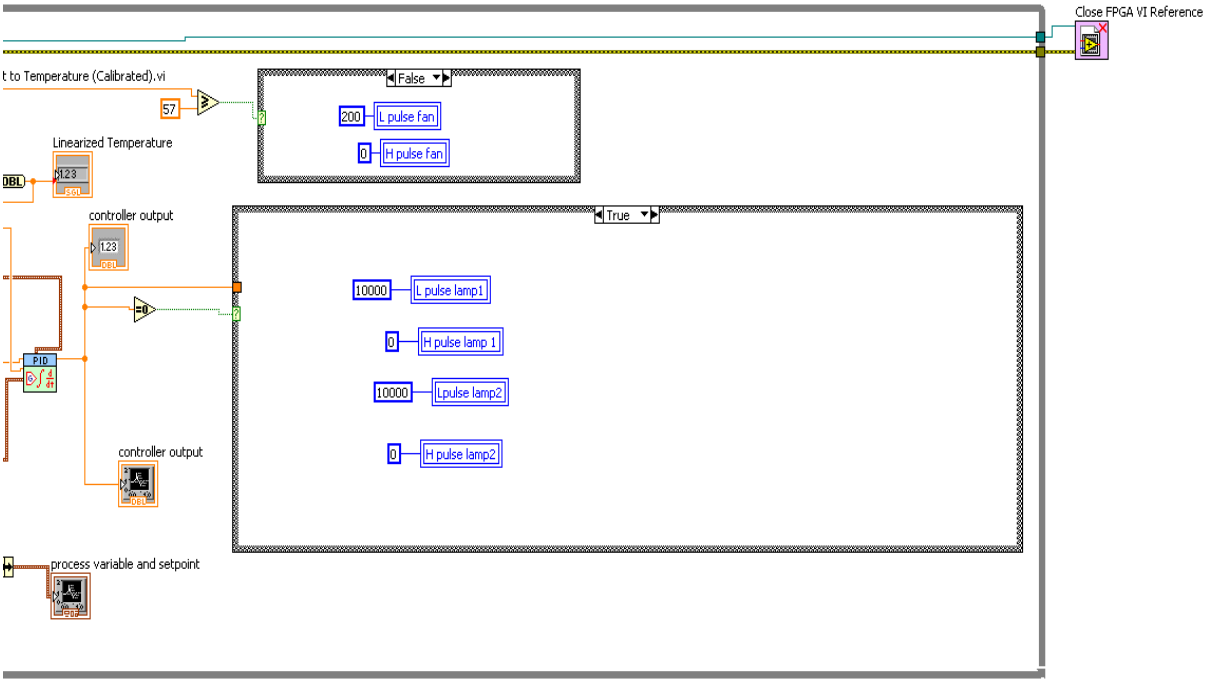
➢ Front Panel

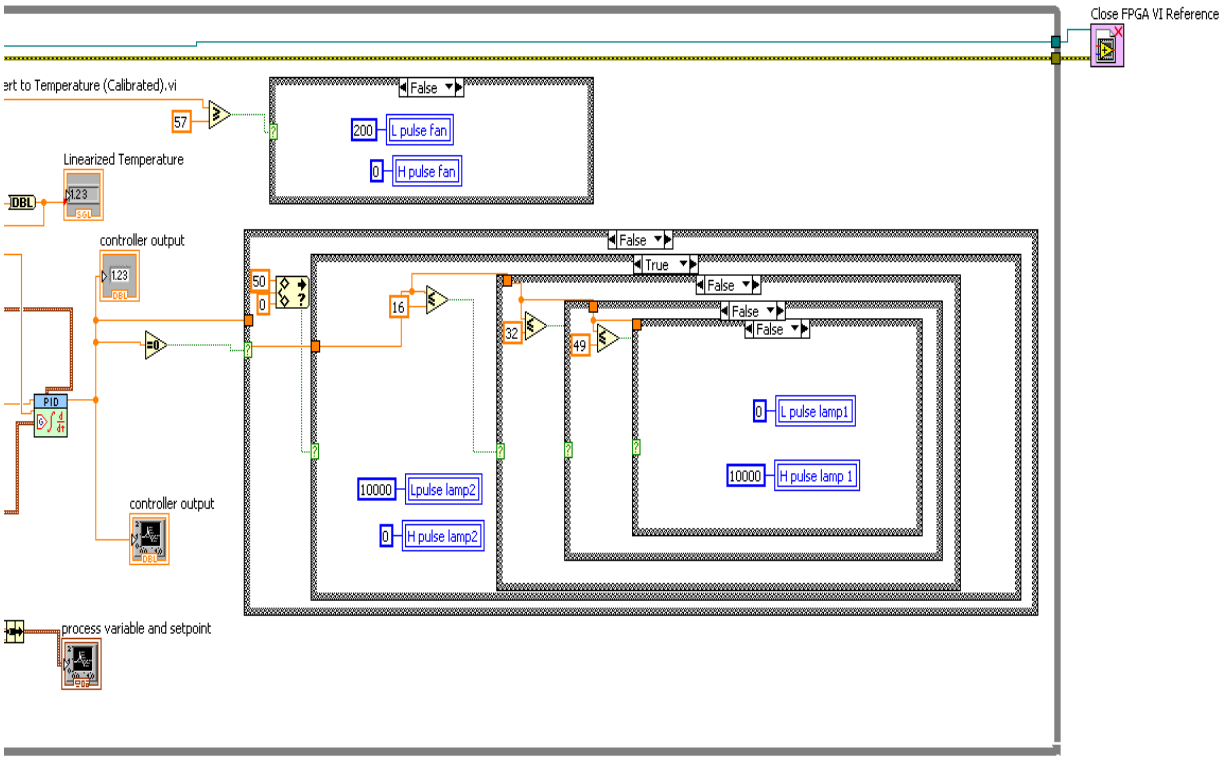

tempmeasure.vi Front Panel on final.lvproj/FPGA Target *
File Edit View Project Operate Tools Window Help
13pt Application Font
Mod1/TC0
0.000000000
Mod1/Autozero
0.000000000
Mod1/CJC
0.000000000
L pulse fan
0
L pulse lamp1
0
H pulse lamp2
0
H pulse fan
0
H pulse lamp 1
0
L pulse lamp2
0
Stop temp acq
STOP
Stop lamp1 pwm
STOP
stop lamp2 pwm
STOP
stop fan pwm
STOP

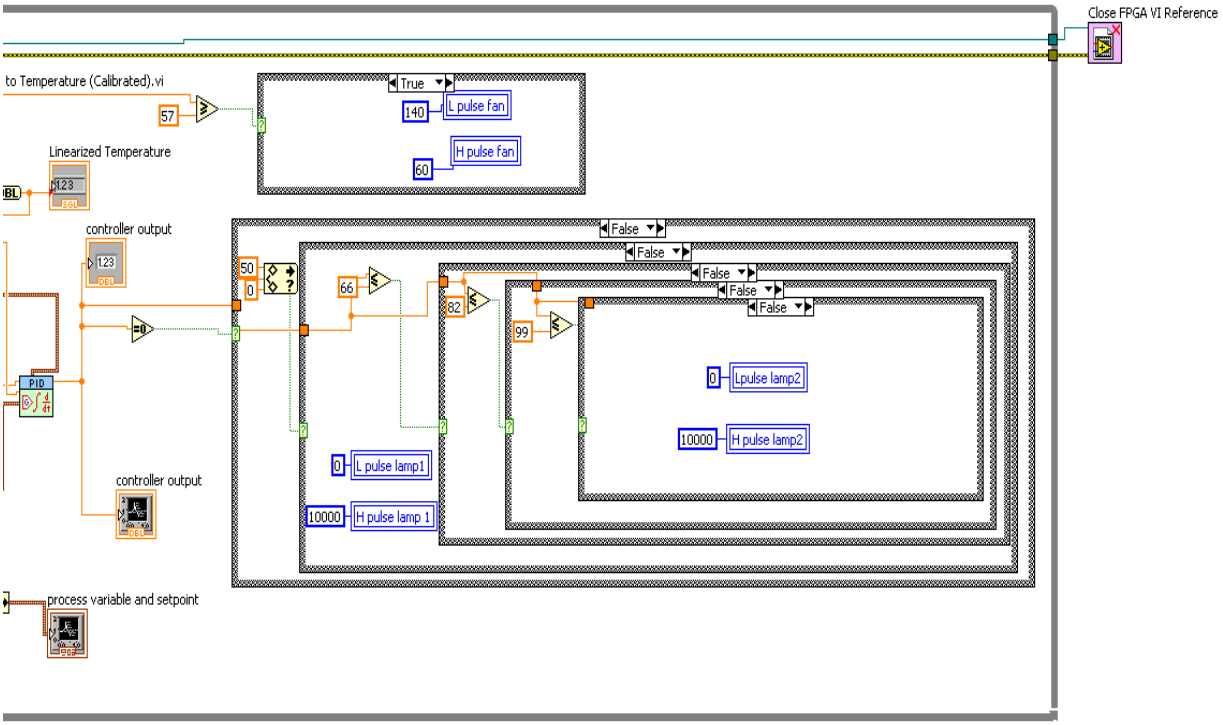
Appendix B – Host VI

➢ Block Diagram

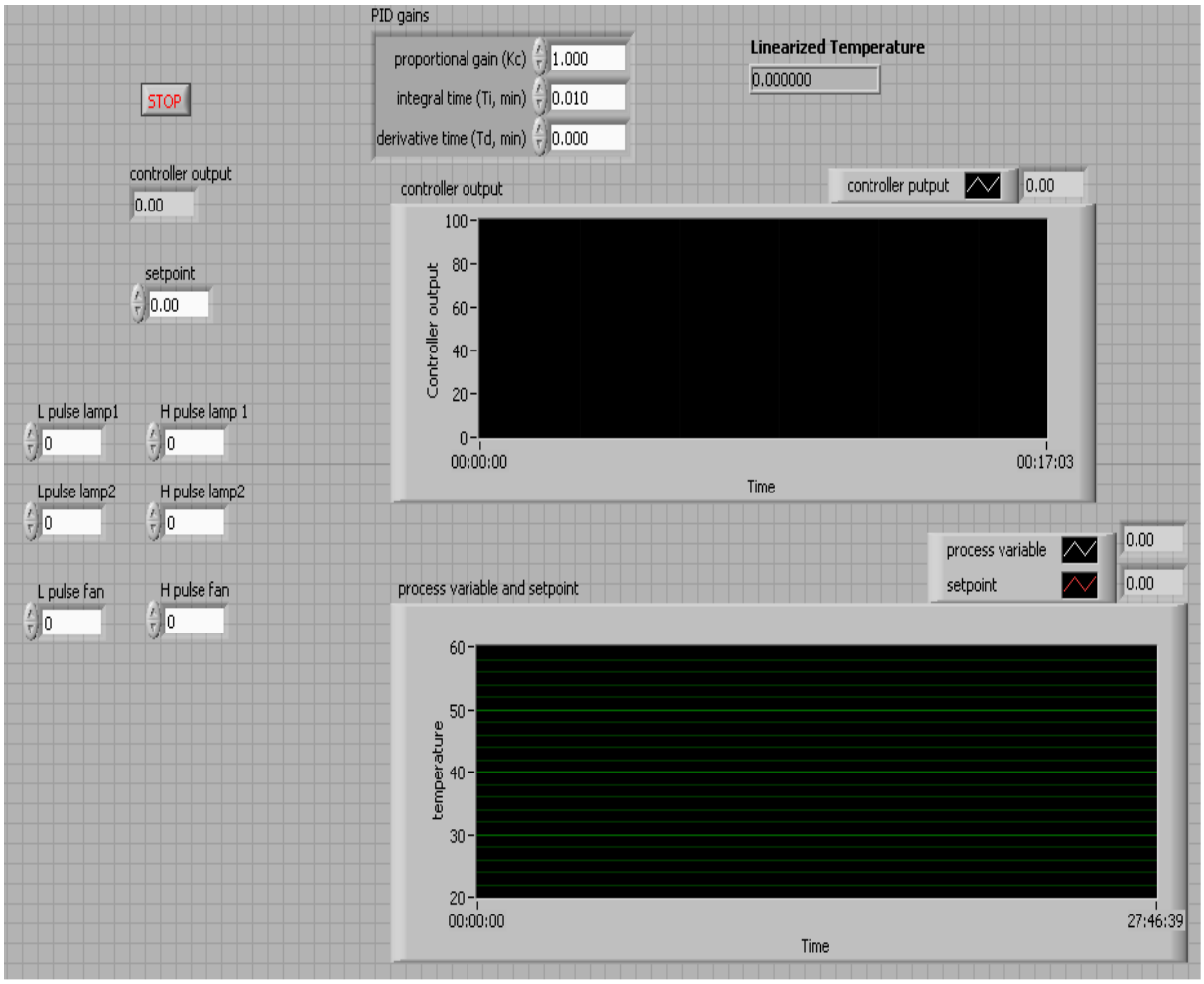





Close FPGA VI Reference
ert to Temperature (Calibrated).vi
57
Linearized Temperature
DBL
controller output
123
DBL
PID
controller output
process variable and setpoint
False
200
L pulse fan
0
H pulse fan
50
0
16
32
49
True
False
False
False
0
L pulse lamp1
10000
H pulse lamp 1
10000
Lpulse lamp2
0
H pulse lamp2

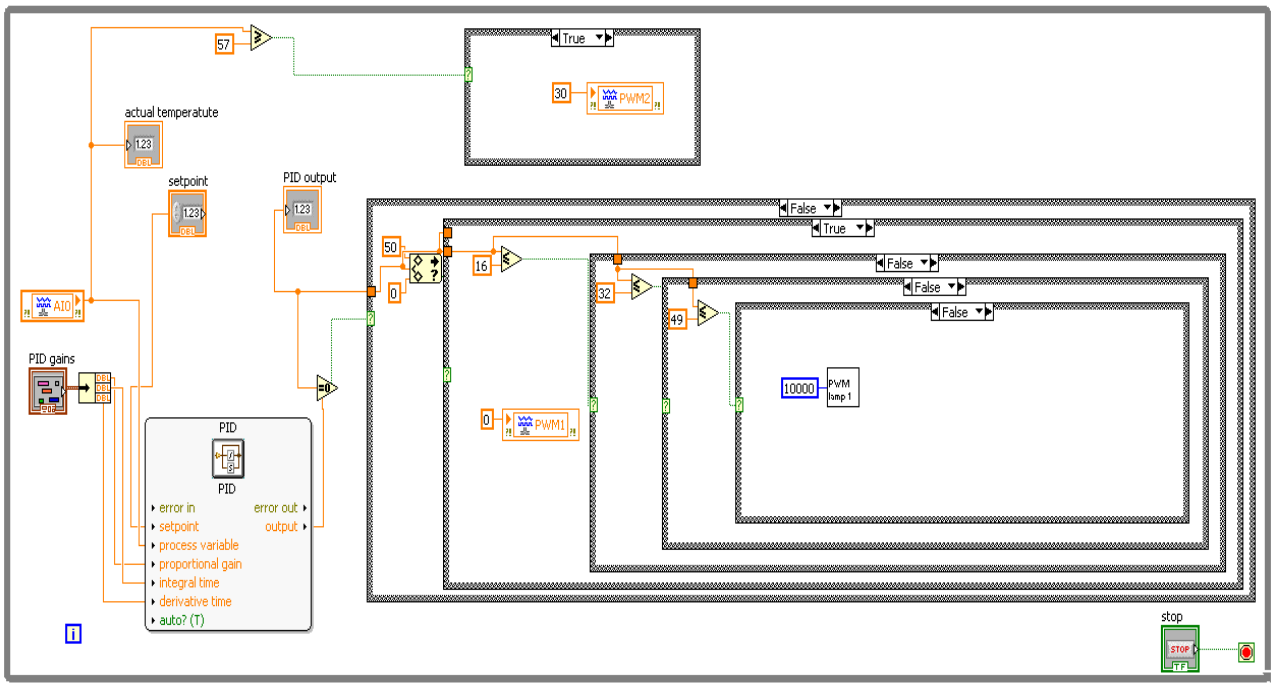

Close FPGA VI Reference
to Temperature (Calibrated).vi
57
Linearized Temperature
DBL
controller output
123
DBL
PID
controller output
process variable and setpoint
True
140
L pulse fan
60
H pulse fan
50
0
66
82
99
False
False
False
False
False
0
Lpulse lamp2
10000
H pulse lamp2
0
L pulse lamp1
10000
H pulse lamp 1

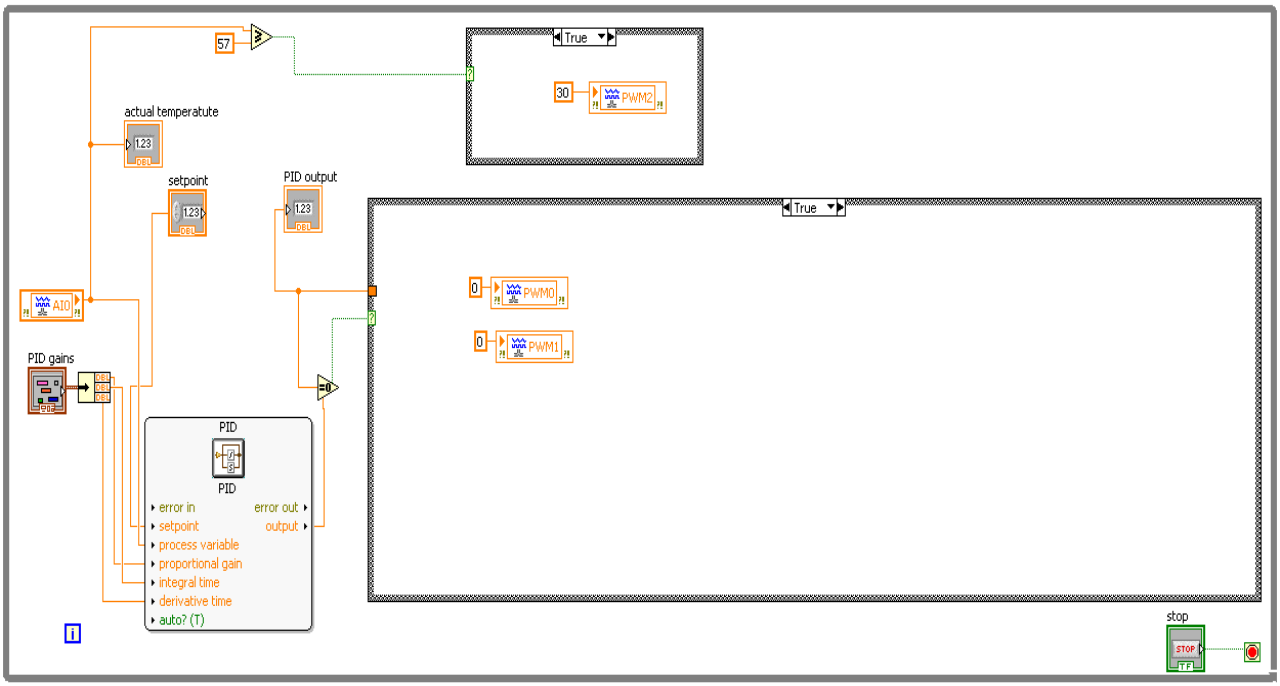
➢ Front Panel

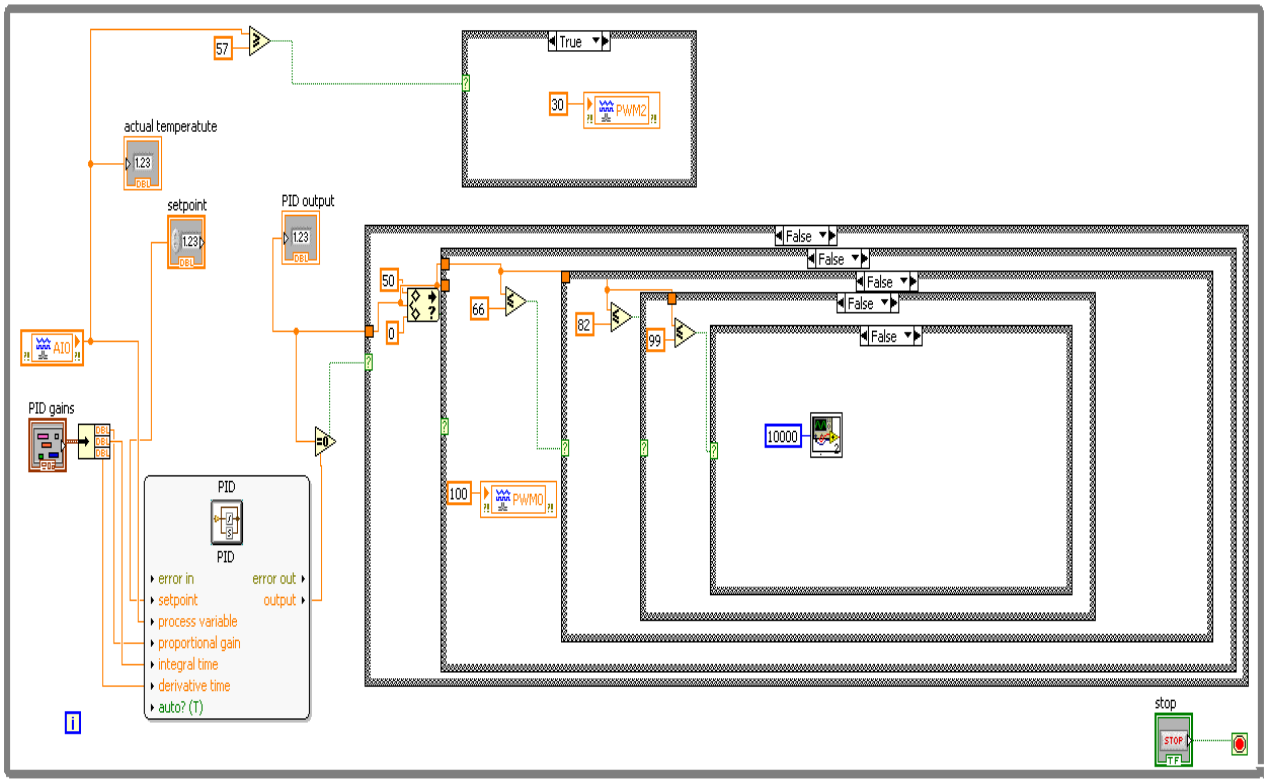

PID gains
proportional gain (Kc)
1.000
integral time (Ti, min)
0.010
derivative time (Td, min)
0.000
Linearized Temperature
0.000000
STOP
controller output
0.00
setpoint
0.00
controller output
controller putput
0.00
Controller output
Time
00:00:00
00:17:03
L pulse lamp1
H pulse lamp 1
Lpulse lamp2
H pulse lamp2
L pulse fan
H pulse fan
process variable and setpoint
process variable
setpoint
0.00
0.00
temperature
Time
00:00:00
27:46:39

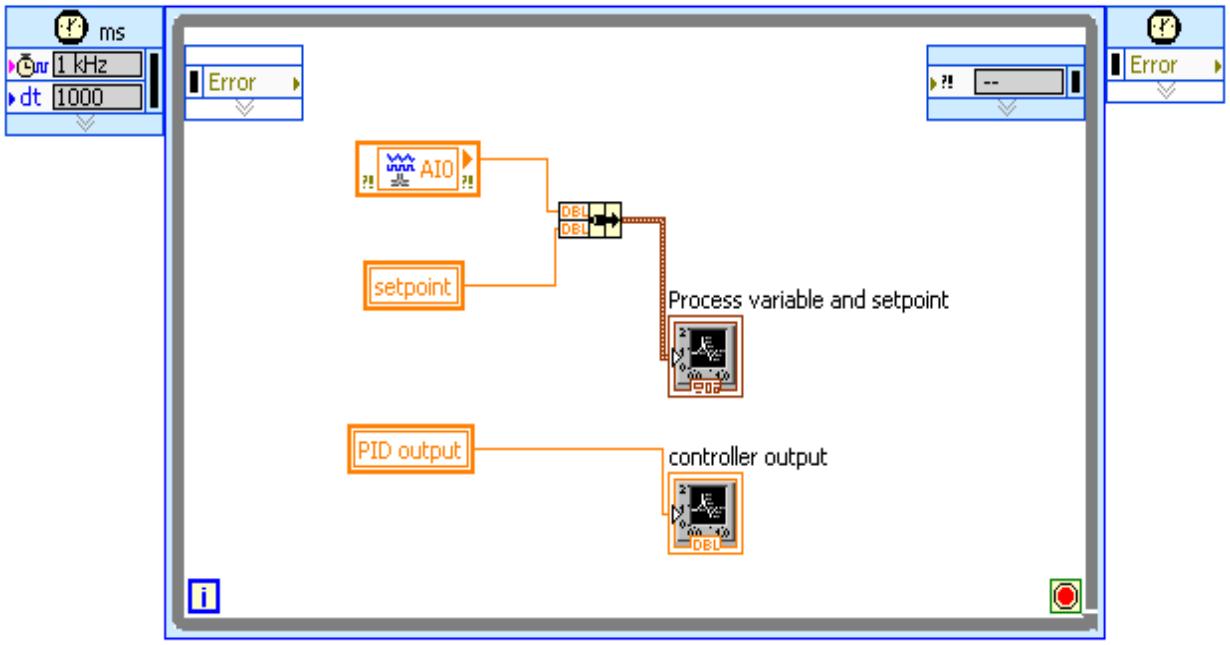
Appendix C – Scan Interface mode VI

➢ Block Diagram

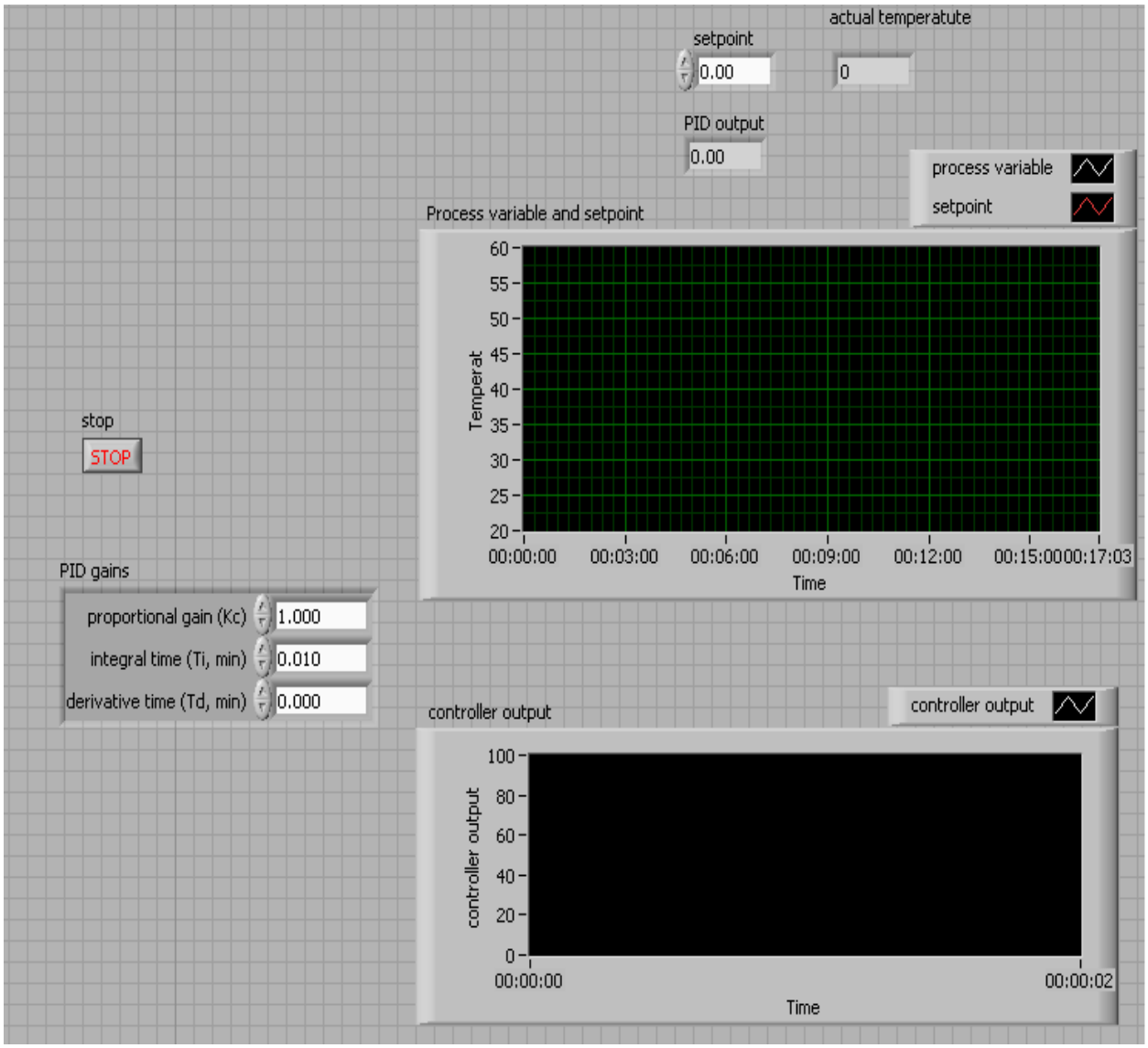





57
True
30
PWM2
actual temperatute
setpoint
PID output
AIO
PID gains
50
0
66
82
99
False
10000
100
PWM0
PID
error in
setpoint
process variable
proportional gain
integral time
derivative time
auto? (T)
error out
output
stop
STOP


ms
1 kHz
dt
1000
Error
AIO
setpoint
Process variable and setpoint
PID output
controller output
DBL

➢ Front Panel


actual temperatute
setpoint
0.00
0
PID output
0.00
process variable
setpoint
Process variable and setpoint
60
55
50
45
40
35
30
25
20
Temperat
00:00:00
00:03:00
00:06:00
00:09:00
00:12:00
00:15:0000:17:03
Time
stop
STOP
PID gains
proportional gain (Kc)
1.000
integral time (Ti, min)
0.010
derivative time (Td, min)
0.000
controller output
controller output
100
80
60
40
20
0
controller output
00:00:00
00:00:02
Time