

Zynq-7000 SOC based device for Remote Data Monitoring and Processing

By

SHUJA UL MULK
01-133162-068

ABDULLAH ABDUL KHALIQ
01-133162-002

HAROON HAIDER
01-133162-080

Supervised by
DR ATIF RAZA JAFRI

{Session 2016-20}

A Report is submitted to the Department of Electrical Engineering,
Bahria University, Islamabad.
In partial fulfillment of requirement for the degree of BS (EE)



Certificate

We accept the work contained in this report as a confirmation to the required standard for the partial fulfillment of the degree of BS (EE).

Head of Department

Supervisor

Internal Examiner

External Examiner

Dedication

We would like to dedicate this project to God All-powerful who is our maker likewise our solid column, our wellspring of astuteness, information and comprehension. God All-powerful has been the wellspring of our quality all through this project and under the shadow of His wings just have we taken off. We would likewise devote this work to our supervisor; Dr Atif Raza Jafri who has been extremely strong from the beginning and empowered every one of us from the earliest starting point and without his help and support we could always have been unable to complete what we had begun. To our family and friends who may have been influenced in any capacity conceivable by this mission. Much thanks to you. Our adoration for all of you can never be evaluated.

Acknowledgements

In the Name of Allah, the Most Merciful, the Most Compassionate all praise is to Allah, the Lord of the worlds; and prayers and peace be upon Mohammad His messenger. First and foremost, we must acknowledge our limitless thanks to Allah, the Ever-Magnificent; the Ever-Thankful, for His help and bless. We are totally sure that this work would have never become truth, without His guidance. We owe a deep debt of gratitude to our university for giving us an opportunity to complete this work. We are grateful to some people, who worked hard with me from the beginning till the completion of the present research particularly my supervisor Dr. Atif Raza Jafri, who has been always generous during all phases of the project, and we highly appreciate the efforts expended Mr. Roman Shah .We would like to take this opportunity to say warm thanks to all our beloved friends, who have been so supportive along the way of doing our thesis. We also would like to express our wholehearted thanks to our family for their generous support they provided us throughout our entire life and particularly through the process of pursuing our education.

Abstract

Field programmable gate array (FPGA) based system on chip provides flexibility, fast processing and compatibility whereas systems like 8051 microcontroller are time inefficient and increase cost of the system. Monitoring remote sensor data and processing the data using FPGA is used in time critical applications where fast processing and compactness is required. FPGA can be used numerous fields, for example, digital image processing and audio, aerospace and defense, automotive, medical and consumer electronics. FPGA can perform continuous signal processing quicker than universally useful programmable processors just as they offer rapid interfaces to other framework segments. During the framework configuration phase of FPGA, the various tasks are doled out to usage in either Programmable Logic or Processing System which is called task partitioning. This stage is significant in light of the fact that the presentation of the general framework will rely upon tasks being doled out for usage in the most fitting innovation which is equipment or programming. This proposed project offers real time data sensing, analog to digital conversion, processing the digital data, and displaying the data into a graphical user interface.

Table of Contents

Certificate	2
Dedication	3
Acknowledgements	3
Abstract	4
Table of Contents	5
List of Figures	8
List of Tables	9
Introduction	10
Project Background/Overview.....	11
Problem Description:.....	11
Project Objectives:.....	12
Project Scope:.....	12
Literature Review	13
Requirement Specifications	15
Existing System:.....	16
Proposed System:.....	16
Requirement Specifications:.....	16
Use cases:.....	17
System Design	19
System Architecture:.....	20
Design Constraints:.....	21
Design Methodology:.....	22
High Level Design:.....	23
Low Level Design:.....	25
Database Design:.....	27
GUI Design:.....	27
System Implementation	28
System Architecture:.....	29
Tools and Technology Used:.....	30
Development Environment/Languages Used:.....	30
Processing Logic/Algorithms:.....	31
1. Machine Learning.....	31
2. Algorithm for compass.....	32
Application Access Security:.....	34
Database Security:.....	34
System Testing and Evaluation	36
Components testing:.....	38
1. Temperature LM35 sensor.....	38
2. Compass Pmod CMPS2.....	41
3. PL and PS Interfacing.....	43
Software performance testing:.....	44
Usability testing:.....	44
Installation testing:.....	45
Exception handling.....	45
Compatibility testing:.....	46
Graphical user interface testing:.....	46

Limitations:.....	46
Evaluation and Results:.....	47
Conclusion:	50
References:	52
Appendices:	54

List of Figures

Figure 3. 1 Use case methodology.....	17
Figure 4. 1 FPGA architectural Design flow.....	21
Figure 4. 2 Architectural Overview.....	23
Figure 4. 3 Architectural Module Based Overview.....	25
Figure 4. 4 Internal structure of LM35.....	26
Figure 5. 1 Slave master configuration I2C Protocol.....	29
Figure 5. 2 Timing Diagram.....	33
Figure 5. 3 Access control for Data Security.....	35
Figure 6. 1 Types of system testing.....	37
Figure 6. 2 Temperature sensor LM35.....	38
Figure 6. 3 Accuracy vs. Temperature.....	39
Figure 6. 4 Connection of XADC with LM35.....	40
Figure 6. 5 CMPS2 pin Configuration.....	41
Figure 6. 6 CMPS2 Schematic.....	41
Figure 6. 7 Connection of FPGA (port JC) with CMPS2.....	43
Figure 6. 8 PL and PS Interfacing.....	43
Figure 6. 9 Xilinx-SDK-Features-Including-the-System-Performance-Analysis- Toolbox.....	44
Figure 6. 10 Room Temperature.....	47
Figure 6.11 Rise in Temperature.....	47
Figure 6. 12 Variation in compass direction.....	48
Figure 6. 13 Graphical User Interface.....	49
Figure 6. 14 Compass Output on RealTerm.....	49

List of Tables

Table 3. 1 Use case methodology.....	18
Table 6. 1 Design parameters.....	38
Table 6. 2 Specifications for temperatures: $-55^{\circ}\text{C} \leq T \leq 150^{\circ}\text{C}$	39
Table 6. 3 Specification of CMPS2.....	42

Chapter # 1

Introduction

Project Background/Overview

Field-Programmable Gate Array (FPGAs) have become one of the key computerized circuit execution media in the course of the most recent decade. A vital piece of their creation lies in their engineering, which administers the idea of their programmable rationale usefulness and their programmable interconnect. FPGA architecture dramatically affects the gadget's speed execution, efficiency, and power utilization. As of late, FPGA's have gotten progressively significant and have discovered their way into system design. Along these lines, the longing rises for an implies that permits data processing along with monitoring.

Processing the data using FPGA is used in time critical applications where fast processing and compactness is required. FPGA can be used numerous fields, for example, digital image processing, audio, aerospace and defense, automotive, medical and consumer electronics. FPGA can perform real time signal processing quicker than universally useful programmable processors just as they offer rapid interfaces to other framework segments. FPGA can perform real time signal handling quicker than universally useful programmable processors just as they offer fast interfaces to other framework parts. End-showcase applications that require multi-usefulness, rapid signal processing and real time response drive the requirement for more astute frameworks with more elevated levels of embedded system performance. Applications, for example, high frequency trading, aviation, video and broadcast, surveillance have a few basic necessities, for instance;

- Advanced decision and control handling.
- Complex client or control framework interfaces.
- High-performance, low-latency signal processing.
- Compact and highly efficient.

The concept of this project is, analog real time data sensing to digital conversion, processing the digital data, and displaying the data into a graphical user interface.

Problem Description:

Microcontrollers have a specific instruction set, certain operations you can perform, like adds, subtracts, binary arithmetic, and in higher-end ones, floating point math. If you want to do something very involved with a microcontroller, you can, but it needs to be built out of those basic instructions. That's what compilers do. As you might expect, when you want to do something complicated in a microcontroller, and it needs to be achieved by doing many lower-level operations, this is slow. Slow compared to an FPGA, that is, where the logic can be expressly tailored to your application. So an FPGA can do a huge FFT, for instance, an operation that requires a ton of multiply operations, much faster than a microcontroller can. 400 million samples per second was achievable 10 years ago. FPGAs are also good at using a processor internally (many "soft" processors are available with known instruction sets) and offloading the toughest computation to hardware accelerators that also reside on-chip. As we usually don't have to go across board traces, these systems can be much faster (and use less power) than the equivalent multi-chip solution. Many operations are much faster than what a microcontroller can do like a 1024-bit data path.

As microcontrollers run at a slower clock speed, and possibly use more power, the FPGA's provide a gateway to solution of all these problems which is being faster and compact.

FPGAs are much more flexible than microcontrollers. FPGA allows you to process parallel task in same time. A FPGA is great at doing precisely the same task, again and again. For example, processing video, audio, RF signals, routing Ethernet packets or simulating fluid flow. Any circumstance where you have a ton of a similar sort of information being tossed at

you truly quick and you need to manage it all similarly or you need to run a similar calculation more than once, FPGA starts to lead the pack in all angles. The FPGA doesn't generally have 'undertakings' that start and stop, its whole occupation is to do something very similar to whatever information it gets, for whatever length of time that it is on. It doesn't switch gears, it doesn't do whatever else. It will do something very similar more than once, as quickly as possible, until the end of time.

Algorithm used in existing systems for data processing and transmission works in a way that sensors collect the data and this data gets transmitted to a processing hub where this data is processed and meaningful data is extracted. A new algorithm is proposed in which the data is processed right after the sensors and then gets transmitted to user using ethernet or wi-fi module where live data can be seen on GUI, increasing effectiveness of system.

Project Objectives:

- To explore the resources of Zynq 7000 programmable System on Chip for using its processing system (PS) and programmable logic parts (PL)
- Interfacing different Analog, Discrete and Digital Sensors I2C, SPI, UART Serial and Analog interfaces.
- Integrating different signal processing cores to process acquired data from the sensors.
- Transmitting the useful data to a remote terminal thorough Ethernet (UDP and/or TCP/IP Protocol) or through UART (for direct connectivity)
- Making a GUI to display useful information.

Project Scope:

The main scope of this project is to implement different sensors like compass and temperature at a remote location and take real time useful data only, along with a camera that would initially have the capability of detecting vehicles where in later stage it can be used to get information like the registration plates of the vehicle. This data is processed at the remote location on the device and the useful data would be displayed on a graphical user interface. The transmission of data takes place using Ethernet protocol for remote locations and through UART for direct connection. The architecture of the FPGA device allows it to work in parallel where it is possible to use the programmable logic and programmable system side by side.

Chapter#2

Literature Review

In this chapter, review of work carried out is presented in the form of literature survey. This chapter also covers the comprehensive general and specific literature survey on FPGA based designs for processing and monitoring on data using FPGAs.

Karen Pamell and Roger Bryner [1] of Xilinx have given a white paper Comparing and Contrasting FPGA and Microprocessor System Design and Development. Programmable Logic Devices offer a cost effective alternative to custom microprocessors due to their generic nature with the added benefits of short time-to-market, no NRE costs, off-the shelf availability, ability to control inventory in peak and trough times, and ability to reduce total cost of ownership over the lifetime of an end product.

Rene Mueller and Jens Teubner [2] addresses the potential of FPGAs as co-processor for data intensive operations in the context of multi-core systems. The type of data processing operations are illustrated where FPGAs have performance advantages (through parallelism and low latency) several ways are discussed to embed the FPGA into a larger system so that the performance advantages are maximized.

Color space conversion is very important in many types of image processing applications including video compression. This operation consumes up to 40% of the entire processing power of a highly optimized decoder. Therefore, techniques which efficiently implement this conversion are desired. F. Bensaali and A. Amira [3] describe four different scalable architectures for efficient implementation of two such color space converters using an FPGA based system.

S. Velusamy et al. [4] have presented the design of a system that monitors the temperatures at various locations on the FPGA. This system is composed of a controller interfacing to an array of temperature sensors that are implemented on the FPGA fabric. Such a system can be used to implement dynamic thermal management.

Atibi mohamed, Benrabh Mohamed, Atouf Issam, Boussaa Mohamed, Bennis Abdellatif [5] presents the idea of implementation of a vehicle detection system in the FPGA platform. This system is based on two algorithms, an image processing algorithm that combines an algorithm for detecting areas of interest through the shadow of the vehicle, and an image descriptor like features type, And another classification algorithm named artificial neural network which aims to detect the presence of the vehicles in these zones. To evaluate the results obtained, which showed that the proposed system is a fast and robust vehicle detector, a hardware implementation was performed in the FPGA embedded platform.

Sanjay Singh, Anil K Saini and Ravi Saini [6] describes the design and implementation of camera interface module required for connecting analog camera with Xilinx ML510 (Virtex-5 FXT) FPGA board having no video input port. Digilent VDEC1 video daughter card is used for digitizing the analog video into digital form. The necessary control logics for video acquisition and video display are designed using VHDL and Verilog, simulated in ModelSim, and synthesized using Xilinx.

Chapter # 3

Requirement Specifications

Innovation is the aggregate of aptitudes, strategies, and procedures utilized in the creation of merchandise or benefits or in the achievement of destinations, for example, logical examination. And it's an entity which always keeps evolving with the passage of time using effective methods and algorithms. The following comparison features the correlation of existing and proposed frameworks just as clarifies the useful and non-practical necessities of the proposed system;

Existing System:

The pattern toward computerized transmission is demonstrated to be an after effect of expanding prerequisites for exactness, dependability, high data rates, and long ways. In this way, information handling and transmission procedures seem, by all accounts, to be converging quickly, both in view of the benefits of digital communication systems and in light of the anticipated weight and volume decreases for PC hardware. Numerous enhancements in transmission effectiveness will result from this converging of methods, strikingly in the zones of information compaction and PC controlled versatile interchanges. Meanwhile most systems tend to extract the information using sensors or other information-collecting devices and then transfer all the information to a processing hub where the collected data is processing using specific algorithms as per requirement. But transmission of all the raw data from sensors to processing hub doesn't seem to be an effective option in some scenarios where the processing, power and cost effectiveness is the main priority.

Proposed System:

General patterns in the methods intended to improve this transmission viability are accounted for and a few promising advancements are underscored. One of the proposed techniques is to filter out the raw data before transmission. So, the data will be collected on the sensors end and a processing device will be connected and provided with certain algorithm as per the sensors and requirement of application. The collected raw data will be processed and only the filtered and meaningful data will be transmitted to the GUI. This system provides multiple advantages over the existing system since you no longer have a need to send the collected data to processing hub rather your data is getting processed and only filtered data is getting transmitted, increasing effectiveness of system.

Requirement Specifications:

To implement the proposed system and its algorithm, four different sensors were chosen which includes sensors for temperature, humidity, compass and camera to detect a certain phenomenon in the video captured by the camera. So, there are four main function requirements which are as fallow;

- Detection of temperature
- Detection of humidity in air
- Detection of direction in which device moves
- Detection of number plate in the video captured by camera

Project is not just limited to these certain functions above mentioned rather it's vastly devised, and any sensor can be implemented, and an algorithm can be designed accordingly as per the user's requirements to filter out the meaningful data. This meaningful data can later be sent to graphical user interface where user can monitor live feed of this meaningful data using ethernet or wi-fi as per user's requirement or data can be transferred directly through UART.

In this chapter, non-functional requirements are of great importance and need to be highlighted since these are the main improvements in proposed system which increases the effectiveness of system.

- For the compass, local magnetic field strength can be calculated in a ± 16 Gauss range where the heading accuracy is of 1° and up to 0.5 mG of the resolution.
- The relative humidity of the environment can be determined with till 14 bits of resolution and result in an accuracy of $\pm 2\%$.
- The temperature can be detected with an accuracy of 0.5°C definite accuracy (at 25°C) within temperature range of -55°C to 150°C .
- Camera is fed with algorithm to detect the number plates of a vehicle in live time without having any significant delay.

Use cases:

Since it's a one platform based device so every sensor will be connected in one place. Once the user described functional requirements are met and fed onto the platform, user needs to do the following;

- Place the device as per the requirement of user and the meaningful place for sensors to work.
- User connects the device to internet and goes to relevant Graphical user interface.
- Live data will be shown on graphical user interface.

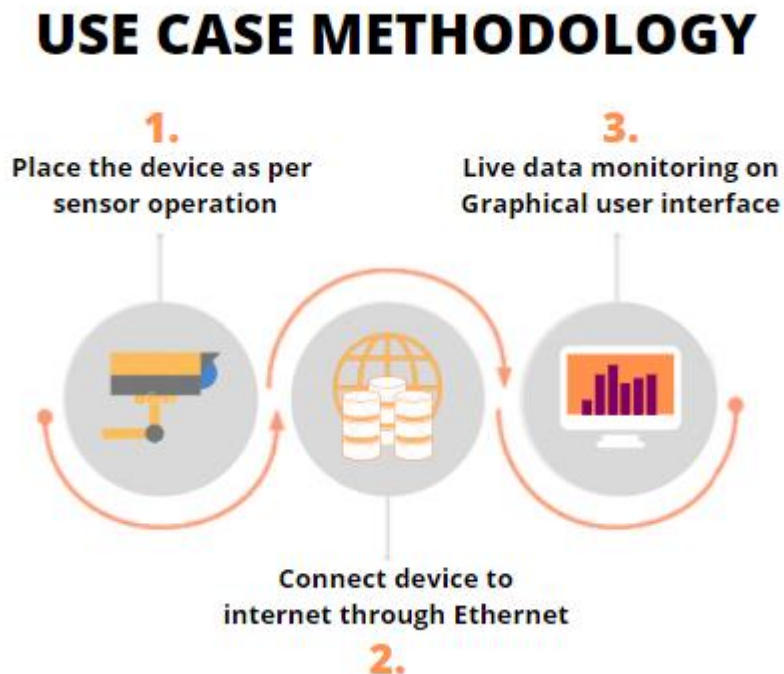


Figure 3. 1 Use case methodology

Title	Zynq-7000 SOC based device for Remote Data Monitoring and Processing	
Version No.	1.0	
Actors	Administrator	
Description	This is the Use Case used for monitoring the data on GUI	
Trigger	User enters the credentials	
Main Success Scenario	Step	Action
	1.	Place device as per requirement of user and meaningful place for sensors to work.
	2.	User connects the device to internet and goes to relevant Graphical user interface.
	3.	Live data will be shown on graphical user interface.
Special Requirements	Requires credentials to monitor the data	
Assumptions	None	
Pre-conditions	Needs proper synchronization among sensors and stable internet	
Post-conditions	N/A	
User interface	Project FYP	
Business Rules	N/A	
	N//A	
Issues	N/A	

Table 3. 1 Use case methodology

Chapter # 4

System Design

System Architecture:

The proposed framework, as shown in Fig. 1, is composed of three subsystems, which are as follow;

- Data from temperature sensor (LM35) and conversion from analog to digital
- Data from compass sensor (CMPS2) and processing it to extract direction
- Graphical user interface

As the proposed framework above mentioned has both, the analog signals and digital signals, so there is a need to map the system in such a way that Field programmable gate array device supports mixed analog and digital signal. Lately, modern series of devices of Field programmable gate array like Zybo Z7-10 series are released which supports both analog and digital signals data collection and processing. A special internal analog to digital connector is provided which is capable of converting analog data to digital in real time and can be processed accordingly.

Additionally, architecture of FPGA being used is also of significant importance as it provides multiple flexibilities to user for the operation of modules connected to device. As of standard architecture of FPGA, Zybo Z7-10 comes comprising of 3 parts;

- Configurable Logic Blocks
- Programmable Interconnects
- Programmable I/O Blocks

Above mentioned terms allow FPGA to operate not only with the internally connected modules as well as the externally connected modules. Configurable logic blocks allow FPGA to implement logic functions required to perform an operation and it consists of flip-flops, transistors, look up tables and multiplexers whereas Programmable interconnects implements the routing between different configurable logic blocks since each logic block that is configurable is connected to a switch matrix in order to access the main routing structure. For sake of connectivity with externally connected modules, multiple input/output connectors are provided allowing user to connect multiple connectors to Zybo Z7. For the case of FYP, we have used 2 externally connected modules which are as follow;

- Analog temperature sensor (LM 35)
- Digital compass (CMPS2)

Data extracted from these two sensors is processed with different mechanism based on if the data is analog or digital and if the application is time critical or not. The usage of programmable logic or the processing system on choice in the Zybo Z7 provides the flexibility to process different operations depending on user's requirements, we used these sensors and processed their extracted data accordingly which fits best to our needs. Finally, the FPGA building configuration stream involves

- design entry
- logic synthesis
- design implementation
- device programming
- design verification

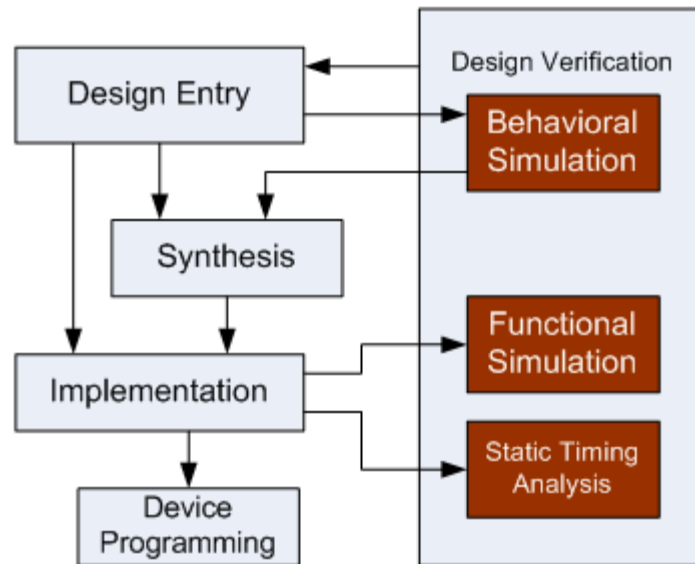


Figure 4. 1 FPGA architectural Design flow

Design Constraints:

Despite the fact that limitations are there to impact the FPGA configuration tools including the synthesizer, place & route tools however they likewise permit a person to determine the execution plan prerequisites and guide the tools toward meeting these necessities. The tools organize their activities dependent on the advancement levels of synthesis, timing, all out quantities of pins utilized, and rationale gave to the tools by an individual or a group. The four sorts of requirements are as decrepit;

- Synthesis
- Inputs/Outputs
- Timing
- Area

The constraints of the synthesis impact even the micro details of the synthesis of Hardware Descriptive Language (HDL) code to Register Transfer Level (RTL). There is a scope of synthesis constraints and their specific circumstance, configuration and use normally fluctuate between various apparatuses.

Input/output limitations which are otherwise called pin task limitations are utilized to dole out a sign to a particular connector on the input/output end.

Timing requirements are alluded for the timing attributes of the structure. Timing imperatives influences all inward timing related interconnections, delays for rationales, look-up tables and between flip-flops. Timing limitations can be worldwide or way explicit relying upon modules and their activity.

Area limitations are utilized to outline necessary hardware within various assets inside the FPGA. It comes extremely convenient when a manual enhancement of gadget is required since programmed directing and placing may utilize a greater number of assets than required. Furthermore, comparatively, area limitations indicate the area either alluding to another

relative structure component or to a particular fixed asset required for the logic inside the FPGA.

Since Zybo Z7-10 has more than enough resources on Programmable Logic part and on the other hand there is dual core ARM processor attached to the device, there is no as such limitation of resources for the processing of extracted data from sensors which can change as per the application of device. Secondly, dealing with a real time acquisition of data from sensors and operating it has pros and cons. This provides a lot of user convenience. The challenge with real time acquisition of data from sensors is the requirement to use different input/output formats like UART, Analog to digital and I2C. The synchronization between the interface that is designed and both (processing unit and memory) must be deliberately considered. In conclusion, the GUI configuration is viewed as the center of the observing and showing of data handling. The presentation module utilized in the proposed project relies upon the Red-Green-Blue (RGB) mode. In this way, so as to display a shading to a spot in the main presentation module, three clock pulse are required from the FPGA. In like manner, there is a requirement for exact synchronization.

Design Methodology:

This project offers a complete hardware system that is reconfigurable and is capable of acquiring data which is obtained through sensors and presents through graphical user interface (GUI). The FPGA does a significant work in this project. The FPGA will in the long run have, in our last form, every single basic segment required for the data acquiring and data processing in one chip. The FPGA not simply controls the entire process that a client needs yet in addition also process the data that is taken from sensors. Playing out specific algorithms or undertakings that an individual need or some other sign preparing activity upon the gathered information from sensors would be possible utilizing the FPGA. Additionally, the input and the output drivers of device, for example, the showcase controller (module) can be implanted in the main FPGA design. The proposed framework will accomplish minimal cost, quick processing and compact size by the reconciliation of major practical units on a chip. It will accomplish fast processing by effective usage of FPGA resources and upon need it can also use the Processing System (PS) of Zybo board in case there are non-time critical operations.

Two terms being used in above needs to be highlighted here. FPGA mainly for processing can use 2 types of resources which are as follow;

- Processing System (PS)
- Programmable Logic (PL)

Programming system will be backed up by the processor where as Programmable logic will be backed up by the resources of FPGA. Even though processor performs a task quite faster than FPGA but the real difference is the Pipelining and use of resources in parallel. Consider it like if we want to execute 5 tasks then processor will execute each task in sequential manner and will execute each task one by one while on the other hand FPGA will execute these tasks in parallel manner and all 5 tasks will get executed at the same time in parallel. Now this phenomena and resources of Zybo Z7-10 provides the flexibility to an individual to choose if an operation should be performed by FPGA resources or Processor. It also provides the flexibility to optimize a system so if there is a time critical application then the task can be performed using FPGA resources otherwise processor can execute the task as well depending upon the need of individual. Language used in case of FPGA is Verilog which is a Hardware Descriptive Language while for processor C++ is used in our case.

High Level Design:

System design is divided into 3 main parts which are as follow;

- Extracting data from Analog sensors and processing them
- Extracting data from Digital sensors and processing them
- Monitoring data on GUI.

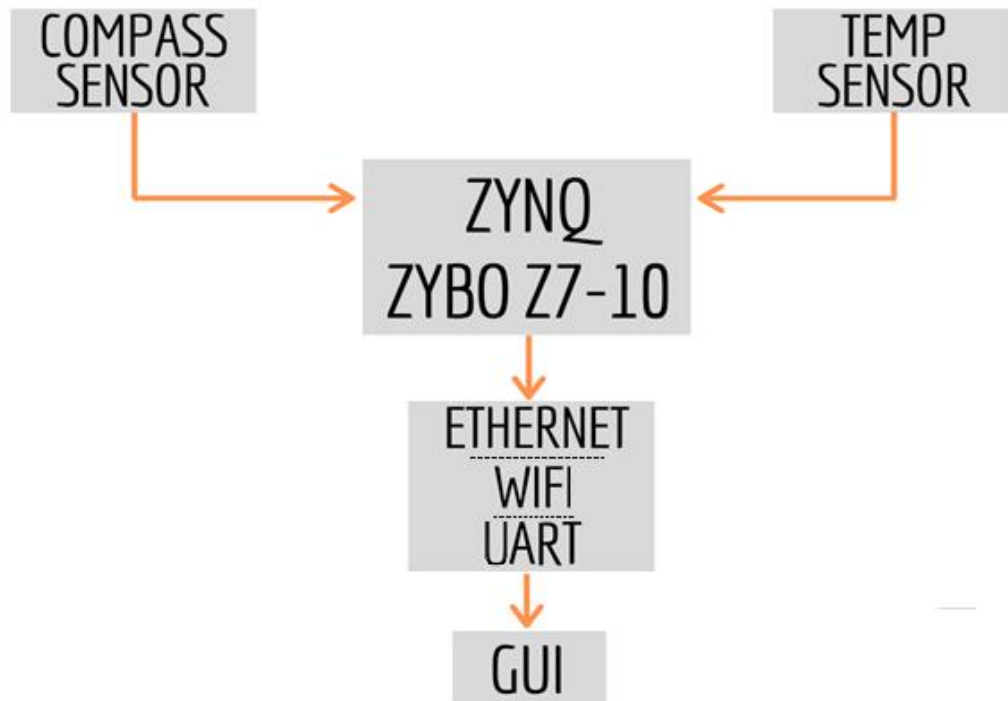


Figure 4. 2 Architectural Overview

Logical View:

In this case, 2 sensors are being used. Temperature (LM35) sensor is an analog sensor so there is a need to extract that information in analog form and then converting it into digital so it can be processed under required algorithms. On the other hand, we have used Compass sensor (CMPS2) which is a digital sensor and its pre-built to return the value in digital. Now after the algorithms are applied and useful and meaningful data is extracted which in this case is direction from CMPS2 and temperature from LM35 and the data is sent to the Graphical User Interface.

Process View:

Procedure of framework is direct and straightforward. Sensors are put which get the information according to their usefulness and the information is changed accordingly from analog to the digital form. The data is processed under certain algorithms depending upon the sensor and then useful information is extracted. Finally this data is transmitted to a GUI.

Performance:

Performance is going to be very important for this project. For everything to run smoothly for the project, the gateways will have to be able to synchronize the extraction and processing of data in live time. Additionally, if there is any problem and data doesn't synchronize then it can cause delay in extraction of data. Similarly, if internet module doesn't work properly then it can delay problems as well.

Module:

Coding is done separately for both sensors and depending upon the application if its time critical or not, the Programmable logic or Processing system resources were chosen. We will discuss both sensors separately. In case of temperature sensor only internal Analog to digital converter was used and backed by programmable logic, Verilog coding was done accordingly and depending upon the reference voltage and temperature sensed voltage, a value was generated showing the temperature surrounding the sensor. On the other hand, for Compass both Verilog and C++ were used. It was taken in the sense that all the required hardware should be described with the Verilog and what to do with this hardware was defined in C++ in the software development kit.

Security:

Since security isn't the essential focal point of the project, just the basic security tools can be applied like in the graphical user interface where a username and password can be added as a feature when accessing data. Additionally, comparing the Zybo against other microcontrollers, FPGA provides a dedicated module built in for encryption of data which can come handy dealing with a defense related application.

Portability:

This system ought to have the capacity that once it is together, the whole system ought to have the option to genuinely be moved starting with one area then onto the next. That is actually the whole point of system on chip (SOC) so every feature should be available on one chip. This is the reason why it can be useful in the remote areas where the system can be taken and using the ethernet the data can be transmitted and monitored by a concerned department.

Reusability:

Characteristic of system to be 1 chip based device provides many advantages. One of them is reusability, since the system is just plug_and_play so it can be used on multiple sites without having any problem or issues.

Resource utilization:

Having resources of FPGA as well as processing power of processor, multiple modules can be used at the same time on device and resources of Zybo can be put to use. Even if the system is overloaded, further optimization can be done which will free a few more resources to be used.

Low Level Design:

There are a number of internally and externally connected modules in the project but for digital and analog data processing we will discuss 2 modules mainly which are as follow;

- CMPS2 with built-in ADC
- LM35 with internal ADC

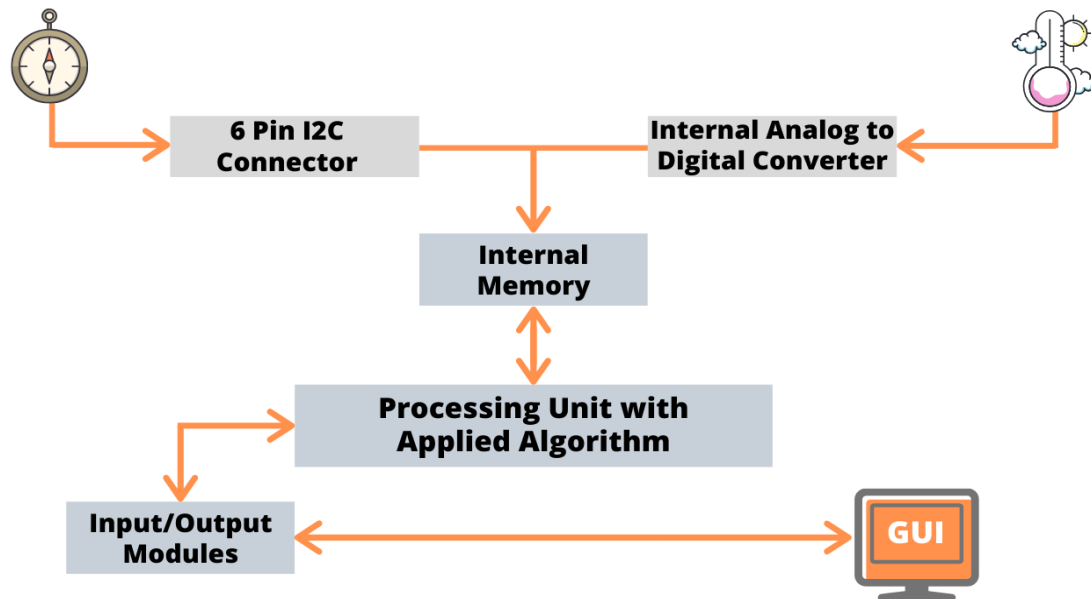


Figure 4. 3 Architectural Module Based Overview

The Digilent Pmod CMPS2 is a 3-axis anisotropic magneto-resistive sensor. With Memsic's MMC34160PJ, the local magnetic field strength in a ± 16 Gauss range with a heading accuracy of 1° and up to 0.5 mG of resolution. It features low noise and data communication protocol used for it is I2C with fast mode. So it will be allowing data transfer of about 400kHz. For the case being, we will be using C++ language and below mentioned steps will be implemented using C++. Here is the series of commands to acquire a set of magnetometer data from the Pmod CMPS2 via pseudo I²C code:

1. Power on the Pmod CMPS2 and wait for 10 mS before further operation.
2. Provide a START condition and call the device ID with a write bit I2CBegin(0xA0) because the device ID is 0x30.
3. Wait to receive an ACK from the Pmod CMPS2.
4. Send the Internal Control Register 0 (address 0x07) as the register to communicate with I2CWrite(0x07).
5. Wait to receive an ACK from the Pmod CMPS2.
6. Write the command to take a measurement by setting bit 0 high followed by a STOP bit I2CWrite(0x01).
7. Delay at least 7.92 mS by default to allow the Pmod CMPS2 to finish collecting data.
8. Provide a START condition and call the device ID with a write bit I2CBegin(0xA0).
9. Wait to receive an ACK from the Pmod CMPS2.
10. Send the Status Register (0x03) as the register to read I2CWrite(0x03).
11. Provide a START condition and call the device ID with a read bit I2CBegin(0xA1).
12. Wait to receive an ACK from the Pmod CMPS2.
13. Cycle the SCL line to receive the Status Register data on the SDA line.

14. Provide a START condition and call the device ID with a write bit I2CBegin(0xA0);
15. Wait to receive an ACK from the Pmod CMPS2.
16. Send the first register address corresponding to Xout LSB (0x00) as the register to be read I2CWrite(0x00).
17. Provide a START condition and call the device ID with a read bit I2CBegin(0xA1).
18. Wait to receive an ACK from the Pmod CMPS2.
19. Convert the readings into usable data.
20. Wait 1/3 of the acquisition time (by default 2.64 ms) before performing another measurement

If the above procedure is analyzed then its nothing but following I2C protocol and sending bits and receiving bits as per the rules defined for I2C.

On the other hand, LM35 is different from the compass application. This sensor doesn't have a built-in converter that can convert analog data to digital data so we have to manually convert analog output to digital. In the case of analog sensor we have put a use to built in module of Zybo Z-10 also known as internal ADC. We have multiple channels available for ADC module and a reference can be provided to ADC and depending upon that voltage difference LM35 will be giving us temperature values accordingly. Out of 4 channels we have used AD14 channel of internal ADC and connected it with LM35 along with Vcc and Ground. Now for every 10mV voltage difference, it's going to count as 1 degree centigrade, and voltage reference is provided on the second row of pins of ADC of Ja connector so the voltage difference can be calculated. Internal structure of LM35 is as follow;

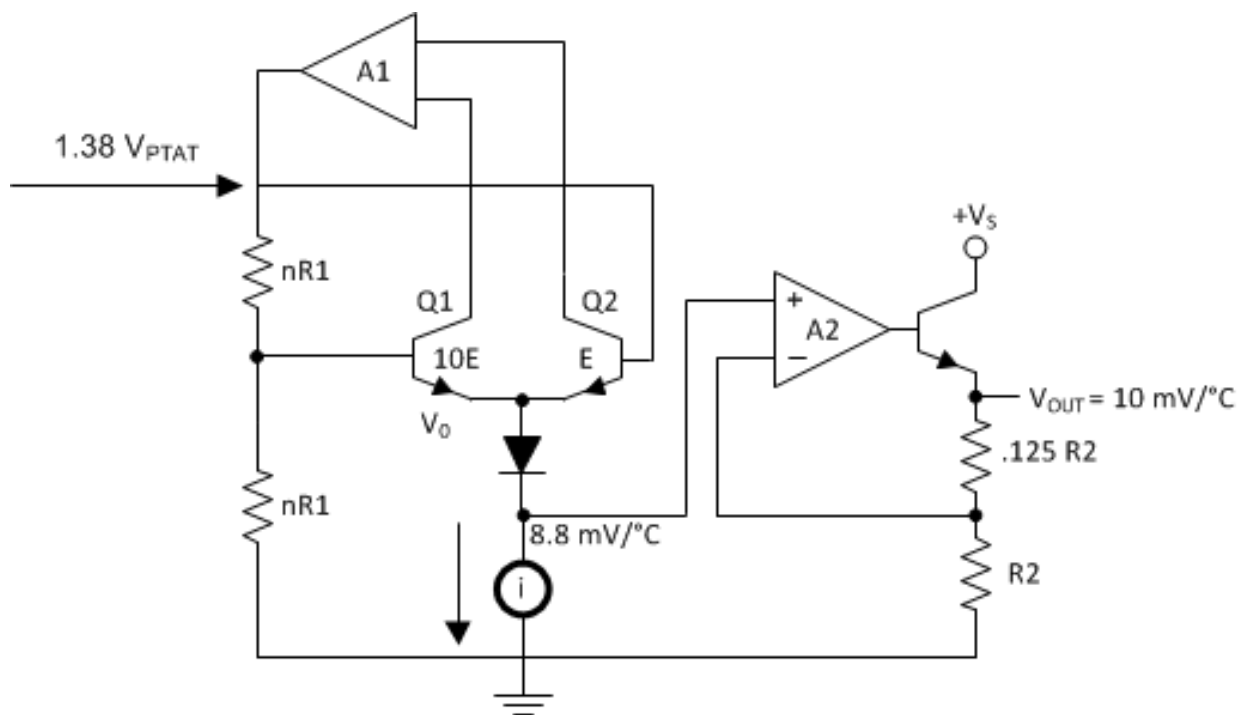


Figure 4. 4 Internal structure of LM35

For the coding part, internal ADC channels can be used using the following commands

```
// .vauxn6      (ja [7]),  
// .vauxp6      (ja [3]),  
  
// .vauxn7      (ja [5]),  
// .vauxp7      (ja [1]),  
  
// .vauxn15     (ja [6]),  
// .vauxp15     (ja [2]),  
  
// .vauxn14     (ja [4]),  
// .vauxp14     (ja [0]),  
  
// .vauxn14     (ja [1]),  
// .vauxp14     (ja [0])
```

Database Design:

The files are individually held by its own format and managed by the software vivado itself as per their operation. Verilog files are stored in “.v” format and once a user completes the design flow for FPGA, user can access these different files in the folder specified by the user itself. Additionally, for the block designing, once each block is places and properly routed and HDL wrapper is done creating then a bit file is generated which later is launched for software development kit and C++ code is fed in as per the functionality required.

GUI Design:

Graphic user interface is designed in order to transmit the date for a user to understand. The platform that we have used for the GUI design is PyQt5. It is a python 3 module that allows for rapid development of GUI applications using its built in program Qt-Designer. There are multiple basic options to start off with like adding labels, buttons and checkboxes and the interface is totally customizable. For our project we initially installed all the necessary libraries required and then imported the installed libraries in python followed by the coding. The Qt-Designer tool is used to graphically design the interface which creates a “.ui” file and this file can then be converted into a python code using command prompt and generate a “.py” file which can then edited to add logic to the buttons and import data from the FGPA board in our case. The data obtained from temperature sensor is passed to the GUI through UART Protocol whereas for the compass, I2C Protocol is used. The data obtained from compass CMPS2 is already directly stored in PS whereas the temperature sensor LM35 data is in PL which first needs to be transferred to PS and then it is passed onto the GUI.

Both the temperature and compass data can be displayed using PyQt5 but the limitation with PyQt5 is that it can only show the data in real time so we used another app called “RealTerm” to display the date of compass which allows us to have a backlog of the entire dataset. RealTerm is designed for the purpose of monitoring and capturing the date while storing a history of previous values obtained. RealTime also allows us to add a time stamp which can be used if a user wants to view the recorded data from a previous particular point in time. This program could have also been used to display the temperature data but we used transmitted the temperature data in real time and the compass data is transmitted through this program so both the functionalities are illustrated and could be used according to the need.

Chapter # 5

System Implementation

System Architecture:

The Zynq Zybo Z7-10 is a feature-rich, ready-to-use embedded software and digital circuit development board. The Zynq family is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, that tightly integrates a dual-core ARM Cortex-A9 processor along with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. The Zybo Z7 surrounds the Zynq with a large set of multimedia and connectivity peripherals in order to create a formidable single-board computer, even before taking into consideration the flexibility and power added by the FPGA. Attaching additional hardware is made easy by the Zybo Z7's Pmod connectors, allowing access to Digilent's catalog of over 70 Pmod peripheral boards, including motor controllers, sensors, displays, and six Pmod ports are available to put any design on an easy growth path.

Connecting these different Pmod's require a protocol in order to transmit the data like in order to connect Pmod CMPS2: 3-Axis Compass, I2C protocol was used. Initially we'll look into the detail as how i2c protocol works in order to communicate the sensor with the board. I2C requires a mere two wires, like the asynchronous serial, but those two wires can support up to 1008 slave devices. Also, unlike SPI, I2C can support a multi-master system, that means allowing more than one master to communicate with all devices on the bus although the master devices can't talk to each other over the bus and they must take turns using the bus lines. Data rates fall between asynchronous serial and SPI; most I2C devices can communicate at 100 kHz or 400 kHz.

The Pmod CMPS2: 3-Axis Compass works as a slave in this project while the Zynq Zybo Z710 board works as a master. For every 8 bits of data to be sent, one extra bit of acknowledgement must be transmitted.

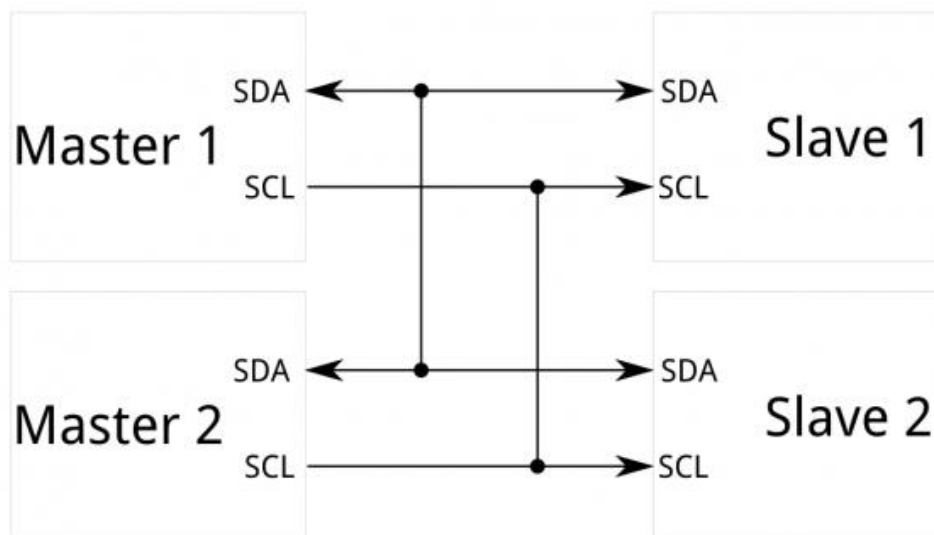


Figure 5. 1 Slave master configuration I2C Protocol

The temperature sensor LM35 is connected to the internal XADC port of the board. The on-board Pmod expansion connector labeled “JA” is wired to the auxiliary analog input pins of the PL. Depending on the configuration, this connector can be used to input differential analog signals to the analog-to-digital converter inside the Zynq (XADC). Any or all pairs in the connector can be configured either as analog input or digital input-output. In analog input mode, the voltage on these pins must be limited to 1V peak-to-peak. In digital mode, the regular VCCO-dependent limits apply. The Dual Analog/Digital Pmod on the ZYBO differs from the rest in the routing of its traces. The eight data signals are grouped into four pairs, with the pairs routed closely coupled for better analog noise immunity. Pins 1 and 7, pins 2 and 8, pins 3 and 9, and pins 4 and 10 are paired up. Furthermore, each pair has a partially loaded anti-alias filter laid out on the PCB. The filter does not have capacitors C94-C97. In designs where such filters are desired, the capacitors can be manually loaded by the user. The coupled routing and the anti-alias filters might limit the data speeds when used for digital signals. The XADC core within the Zynq is a dual channel 12-bit analog-to-digital converter capable of operating at 1 MSPS. Either channel can be driven by any of the auxiliary analog input pairs connected to the JXADC header. The XADC core is controlled and accessed from the PL via the Dynamic Reconfiguration Port (DRP). The DRP also provides access to voltage monitors that are present on each of the FPGA’s power rails. The LM35 temperature sensor when connected to the XADC port works by every 10 millivolt change in voltage brings about a change of 1 degree Celsius that is detected by the sensor.

Tools and Technology Used:

1. Technology for Connecting Sensors:
 - Vivado Hlx Edition
 - Ise Design Suite
 - Software Development Kit
2. Tools for Connecting Sensors:
 - Verilog
 - C++
3. Technology for Graphical User Interface:
 - Python
4. Tools for Graphical User Interface:
 - PyQt5

Development Environment/Languages Used:

1. For Block design (Development Environment)
 - Vivado Hlx Edition
2. For communicating Zybo Z7-10 board with sensors (Development Environment)
 - Vivado Hlx Edition
3. For communicating Zybo Z7-10 board with sensors (Languages Used)
 - Verilog,
 - C Language
4. For Graphical User Interface (Development Environment)
 - PyQt5

5. For Graphical User Interface (Languages Used)

Processing Logic/Algorithms:

- **Machine Learning:**

1. Background:

Machine learning can be classified into two main categories that are supervised and unsupervised. Supervised machine learning mainly relies on the labeled input data in order to learn a function that will give a correct output with unlabeled data whereas an unsupervised machine learning will use input data without labels, for instance without any supervisor(label) that would tell when it is correct or when it is supposed to self correct. In this project supervised machine learning algorithm is adopted for the purpose of training the data in Matlab for a disease in order to learn the principles of machine learning and how it can be used for further implementations depending on the need. Supervised machine learning in simple words can be understood with an example of a father (label) who shows different animals to his son (computer) everyday and by some time his son learns to differentiate between different species of animals, however at certain instances where two objects are alike the computer might not correctly recognize them but this could be corrected with more intensive training on a larger set of data.

2. K Nearest Neighbor Algorithm in Matlab :

The K nearest neighbour (Knn) algorithm was implemented in Matlab. Knn mainly assumes that things that are similar depending on certain features are always in close proximity. This means that similar things will always be near each other and this is what this algorithm relies upon for being true. This refers to calculating distance between points and creating a class of things that are near to each other based on there features. The Eucilidean distance approach was adopted for calculating the distance. As our approach was to get data for classification, therefore considering the current pandemic (covid-19), we opted for a similar data with feautres like body temperature, age and then apply machine learning which could be later used by cameras to detect possible positive carriers, we used data provided by “kaggle”, a website for data science community, for diabetes disease since our aim on the software part was to apply the algorithm and see the results for accuracy however different data sets can be added depending on the type of model needed. In our approach we divided the given data in ratio of 70% for training data and 30% for testing data. As odd values are preffered for selecting the value of k nearest neighbour therefore we selected K=3 although recommended value for K for better accuracy is 10. As value of K when it decreases to 1, the system would not make a lot of computations therefore the system would be less stable but if the value of K increases from 1 towards 10 the system would be more stable but there would be a lot of computations. The standard formula for accuracy in Knn is:

$$(TP + TN)/(TP + TN + FP + FN).$$

where TP, FN, FP and TN represent the number of true positives, false negatives, false positives and true negatives, respectively

In true positive where the actual class and the predicted class is also the same, for instance the actual class of patient was positive and the predicted class was also detected as positive, this will be true positive and opposite for true negative where actual was negative and the prediction was also negative. The sum of these two will be divided by total sum of true

positive and true negative along with false positive and negative. False positive means actual sample was positive and it was detected as negative, the opposite would apply to false negative. In results the accuracy for our algorithm came out to be as 69.73%/.

The algorithm for Pmod CMPS2: 3-Axis Compass works in the following way:

1. Power on the Pmod CMPS2 and wait for 10 mS before further operation.
2. Provide a START condition and call the device ID with a write bit
I2CBegin (0xA0); //device ID 0x30 with a write (0) bit
3. Wait to receive an ACK from the Pmod CMPS2.
4. Send the Internal Control Register 0 (address 0x07) as the register to communicate with
I2CWrite (0x07); //address 0x07 corresponds to Control Register 0
5. Wait to receive an ACK from the Pmod CMPS2.
6. Write the command to take a measurement by setting bit 0 high followed by a STOP bit.
I2CWrite (0x01); //0x01 initiates a data acquisition
7. Delay at least 7.92 mS by default to allow the Pmod CMPS2 to finish collecting data.
8. Provide a START condition and call the device ID with a write bit
I2CBegin(0xA0); //device ID 0x30 with a write (0) bit
9. Wait to receive an ACK from the Pmod CMPS2.
10. Send the Status Register (0x03) as the register to read

11. Provide a START condition and call the device ID with a read bit
I2CBegin (0xA1); //device ID 0x30 with a read (1) bit
12. Wait to receive an ACK from the Pmod CMPS2.
13. Cycle the SCL line to receive the Status Register data on the SDA line. Keep reading the Status Register by repeating steps 8 through 13 until bit 0 is set to '1', indicating that the data on all 3 axes as available to be read.
14. Provide a START condition and call the device ID with a write bit
I2CBegin (0xA0); //device ID 0x30 with a write (0) bit
15. Wait to receive an ACK from the Pmod CMPS2.
16. Send the first register address corresponding to Xout LSB (0x00) as the register to be read.
I2CWrite(0x00); //address 0x00 as the first register to be read

17. Provide a START condition and call the device ID with a read bit
I2CBegin(0xA1); //device ID 0x30 with a read (1) bit
18. Wait to receive an ACK from the Pmod CMPS2.
19. Cycle the SCL line to receive the data bits from the X, Y, and Z registers in the SDA line, providing an ACK between each data byte. The Pmod CMPS2 address pointer automatically moves to each consecutive byte. End the communication by sending a NACK followed by a STOP command.
I2CReadMultiple (6); //read six bytes, sending an ACK to the slave device between each byte received and a NACK after the last byte
20. Convert the readings into usable data.

Data Conversion:

- i). Calculate the real Gauss value for the X and Y axes from the amount of LSBs returned where the LSB value by default is 0.48828125 mG, resulting in 2048 LSBs per Gauss.

$$xGaussData = xDataLSB * 0.48828125mG$$

$$xGaussData = yDataLSB * 0.48828125mG$$

- ii). Calculate the direction D by first checking to see if the X Gauss data is equal to 0 to prevent divide by 0 zero errors in the future calculations. If the X Gauss data is 0, check to see if the Y Gauss data is less than 0. If Y is less than 0 Gauss, the direction D is 90 degrees; if Y is greater than or equal to 0 Gauss, the direction D is 0 degrees.
- iii). If the X Gauss data is not zero, calculate the arctangent of the Y Gauss and X Gauss data and convert from polar coordinates to degrees.

$$D = \arctan (\frac{yGaussData}{xGaussData}) * 180 \pi$$

- iv). If the direction D is greater than 360 degrees, subtract 360 degrees from that value.
- v). If the direction D is less than 0 degrees, add 360 degrees to that value.
- vi). The compass heading can then be determined by the direction value D:
 - If D is greater than 337.25 degrees or less than 22.5 degrees – North
 - If D is between 292.5 degrees and 337.25 degrees – North-West
 - If D is between 247.5 degrees and 292.5 degrees – West
 - If D is between 202.5 degrees and 247.5 degrees – South-West
 - If D is between 157.5 degrees and 202.5 degrees – South
 - If D is between 112.5 degrees and 157.5 degrees – South-East
 - If D is between 67.5 degrees and 112.5 degrees – East
 - If D is between 0 degrees and 67.5 degrees – North-East

Wait 1/3 of the acquisition time (by default 2.64 ms) before performing another measurement.

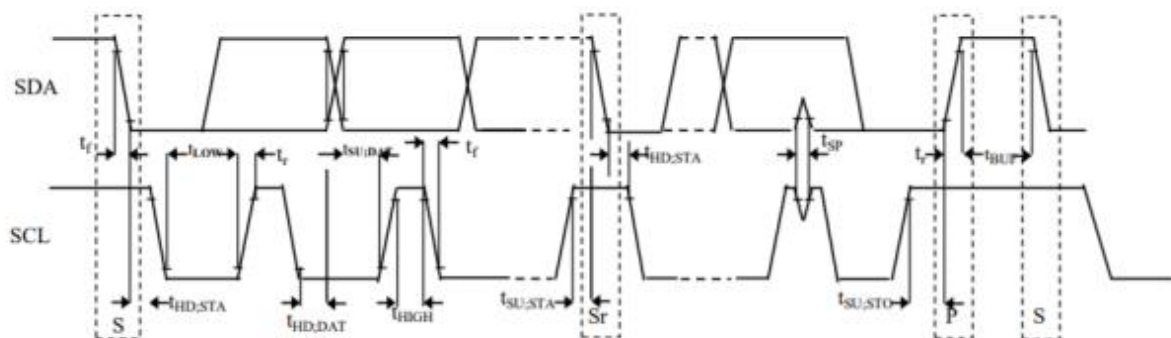


Figure 5. 2 Timing Diagram

Application Access Security:

There are two types of application access security: physical and logical. Physical application access security means that the system is to be protected physically from someone trying to destroy it in any way. Logical application access security protects the system by the help of firewalls from different viruses and third party intruders who try to gain access of the system remotely.

This system is to be placed in a remote location in order for the data to be retrieved and processed hence for the physical protection of the system it would be placed in a glass box with a keypad lock that would secure the system from different situations to some extent. Since the logic is implemented on vivado and the language used is verilog mainly so there is

no logical security implemented, which can be considered one of the constraints of the system.

Database Security:

Access control is one of the essential administrations that any data management should have. Its mainly the data that protected data from unapproved read and the write tasks. Access control characterize ensure that all correspondence to the database and other framework objects are as per the policies defined. Result in an error can be as significant which can make issue in a frameworks activity and at some point might bring it to stop. Through controlling access rights may likewise helps in decreasing the dangers that may decisively affect the security of the database on the fundamental servers. For instance, if any table is erased or gets to is altered inadvertently the outcomes can be roll backed or for some specific rules, however by applying the access control their deletion can be restricted.

A fundamental security prerequisite is that you should know your clients. You should distinguish them before you can decide their benefits and access rights, thus that you can review their activities upon the information. Client can be validated from numerous points of view before they are permitted to make database. Database validation incorporates both ID and confirmation of clients. This is the essential prerequisite to guarantee security since the ID procedure characterizes a lot of individuals that are permitted to get to information. To guarantee security, the person is validated and it keeps the information (sensitive) secure and from being changed by unapproved client. Hacker can adopt various strategies like detour confirmation, default password when they intend to bargain client data and validation. In this undertaking the graphical UI can be utilized as safety effort where it can require a username and secret phrase from the client so as to get to which can be advantageous such that they would have no entrance to the realtime information regardless of whether they have effectively introduced a malware in the framework. This would hold back any client who isn't confirmed from the chance of review the information. Anyway this is each other requirement of the undertaking as it was past the targets of the task to include an element of security in the graphical UI.

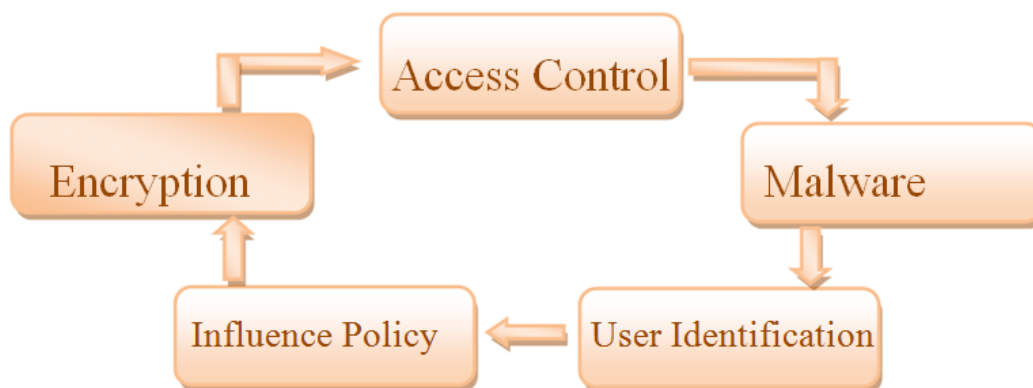


Figure 5. 3 Access control for Data Security

Chapter # 6

System Testing and Evaluation

System testing and Evaluation

System testing is the process of assessment which checks the original or current status of a system in comparison with the expected results to chart its future direction. The outcomes are assessed to evaluate progress of plan, execution, legitimacy, and so on.

Test and Assessment includes assessing an item from the part level, to independent framework, integrated system, and, if it is appropriate, system-of-system and enterprise. Figure 1 features the various sorts of testing necessitated that should be assessed so as to assist us with checking that the framework meets its predefined prerequisites.

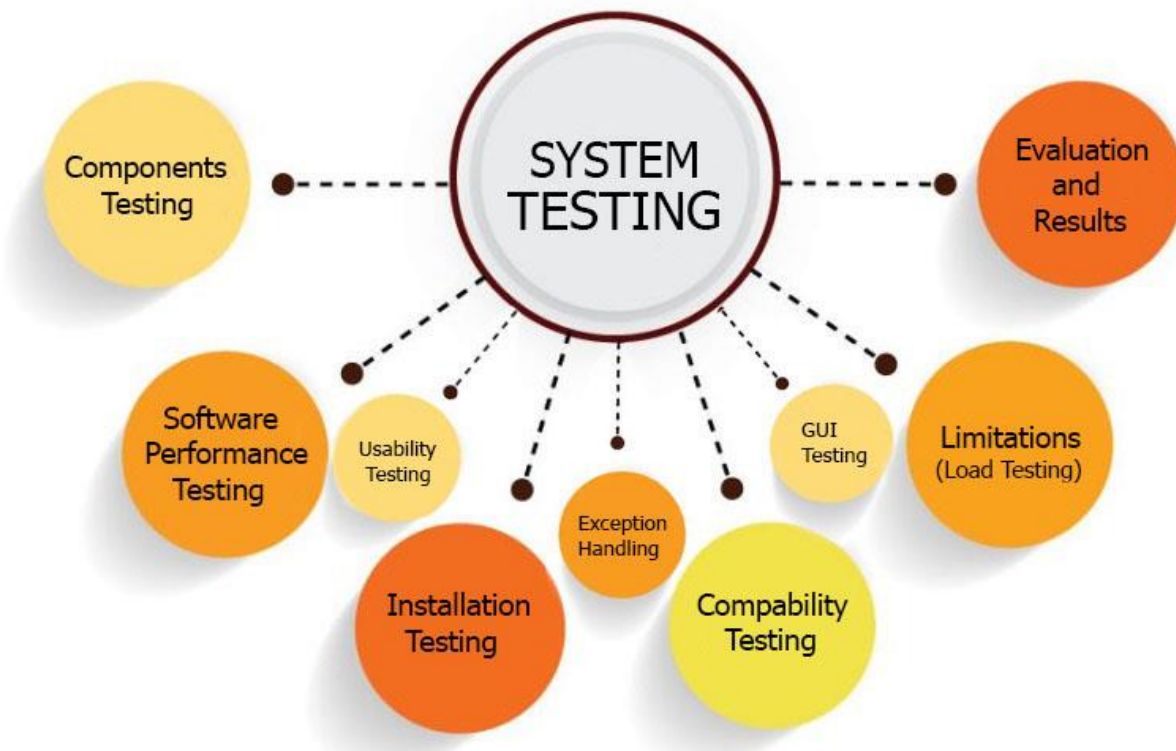


Figure 6. 1 Types of system testing

Components testing:

Part testing is otherwise called module testing. It mainly finds the bugs and errors in the module and confirms the working of programming. The hardware implementation has been discussed in Chapter 4 and Chapter 5 showing the complete design of the system and integration of the different sensors but before the integration of a sensor It is important to perform tests on individual components separately in order to understand the working and limitations. Testing of the sensors are done separately demonstrating the working principle of each sensor, their capabilities and limitations. Two sensors are being integrated with the FPGA which are as follow:

- Temperature LM35 sensor.
- Compass manufactured by Digilent called Pmod CMPS2.
- PL and PS intergration

Temperature LM35 sensor:

Temperature LM35 sensor is a coordinated simple temperature sensor whose electrical output is relative to Degree Centigrade. The sensitivity of the sensor LM35 is 10 mV/degree Celsius. As temperature starts increasing then the output voltage also increases.

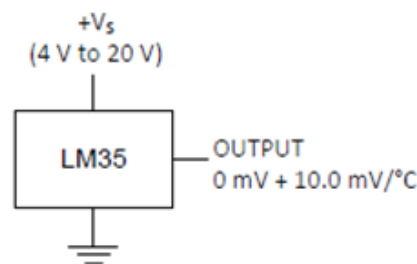


Figure 6. 2 Temperature LM35 sensor.

PARAMETER	VALUE
Accuracy at 25°C	±0.5°C
Accuracy from -55 °C to 150°C	±1°C
Temperature Slope	10 mV/°C

Table 6. 1 Design parameters

1) Accuracy of LM35

Precision is characterized as the error that is between the output voltage and 10 mV/°C times the case temperature of the gadget, at indicated states of voltage, current, and temperature (communicated in °C). The accuracy determinations of the LM35 are given as for a basic transfer function (linear) which implies that the voltage (output) is linear to temperature. There will be an increase of 10mV (0.01V) for each 1°C increase in temperature.

PARAMETER	TEST CONDITIONS	LM35A			UNIT
		TYP	TESTED LIMIT ⁽¹⁾	DESIGN LIMIT ⁽²⁾	
Accuracy	T = 25°C	±0.2	±0.5		°C
	T = -10°C	±0.3			
	T = T _{MAX}	±0.4	±1		
	T = T _{MIN}	±0.4	±1		

Table 6. 2 Specifications for temperatures: $-55^{\circ}\text{C} \leq T \leq 150^{\circ}\text{C}$

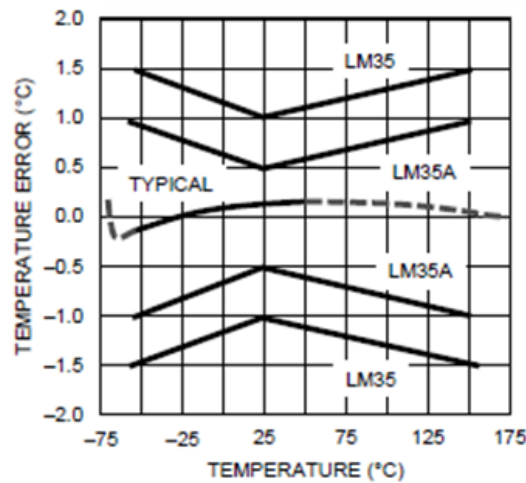


Figure 6. 3 Accuracy vs. Temperature

2) Data Conversion :

LM35 is a precision Integrated circuit Temperature sensor whose output voltage varies based on the temperature around it. It can easily be interfaced with any Microcontroller that has ADC function or any development platform like FPGA in our case

If the temperature is 0°C , then the output voltage will also be 0V . There will be rise of 0.01V (10mV) for every degree Celsius rise in temperature. The voltage can be converted into temperature using the below formulae.

$$V_{\text{OUT}} = 10 \text{ mV}/^{\circ}\text{C} \times T$$

Where:

- V_{OUT} is the LM35 output voltage
- T is the temperature in $^{\circ}\text{C}$

3) LM35 and FPGA Interfacing

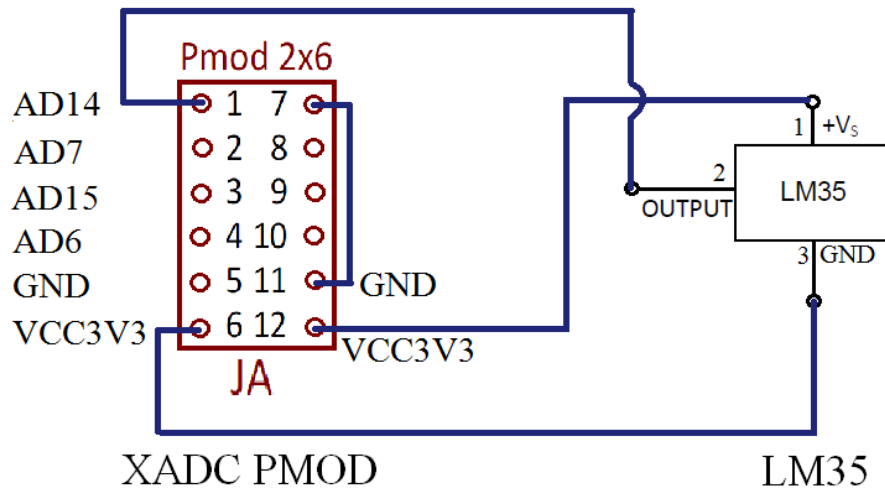


Figure 6. 4 Connection of XADC with LM35

Connect LM35 to FPGA's Internal ADC as shown in circuit diagram. The +Vs for LM35 can be taken from the VCC3V3 pin of XADC. Also the ground pin of LM35 can be connected to GND pin of XADC. Connect OUTPUT (the analog out of LM35) to AD14 input pin of ADC.

The internal ADC of the ZYBO Z7 is a 16 bit ADC which contains 4 channels so a total of 65536 (2^{16}) values can be measured. The AD14 pin of the ADC get an analog value from the LM35 and the converts the voltage level by dividing the analog value by 65536 and a voltage level is measured. Since LM35 detects a change of 1°C with 0.01V (10mV), the answer obtained is then multiplied by 1000 to convert the voltage level from Volts to millivolts. Finally to observe a change for degree Celsius, the answer in millivolts is divided by 10 to get the corresponding temperature as shown in the following portion of the Verilog code.

```

:
:
    wire [9:0] Temp_C;
    assign Temp_C = pwm_duty0 * 'd1000 /'d65536;

ila_0 your_instance_name (
    .clk(clk), // input wire clk

    .probe0(pwm_duty0), // input wire [15:0] probe0
    .probe1(Temp_C) // input wire [9:0] probe1
);
:
:

```

Compass Pmod CMPS2:

The 3-axis compass accompanies a 6-pin Pmod connector with I2C interface and a go through Pmod have port for the chaining. Regular applications incorporate an electronic compass, GPS route, and position detecting.

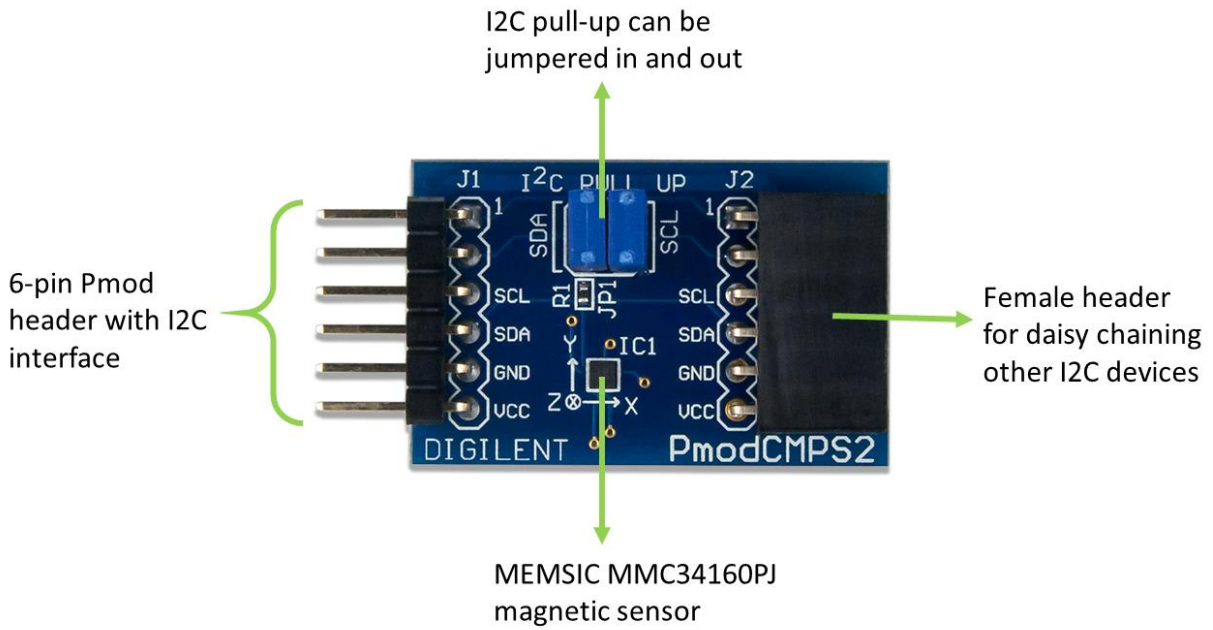


Figure 6. 5 CMPS2 pin Configuration

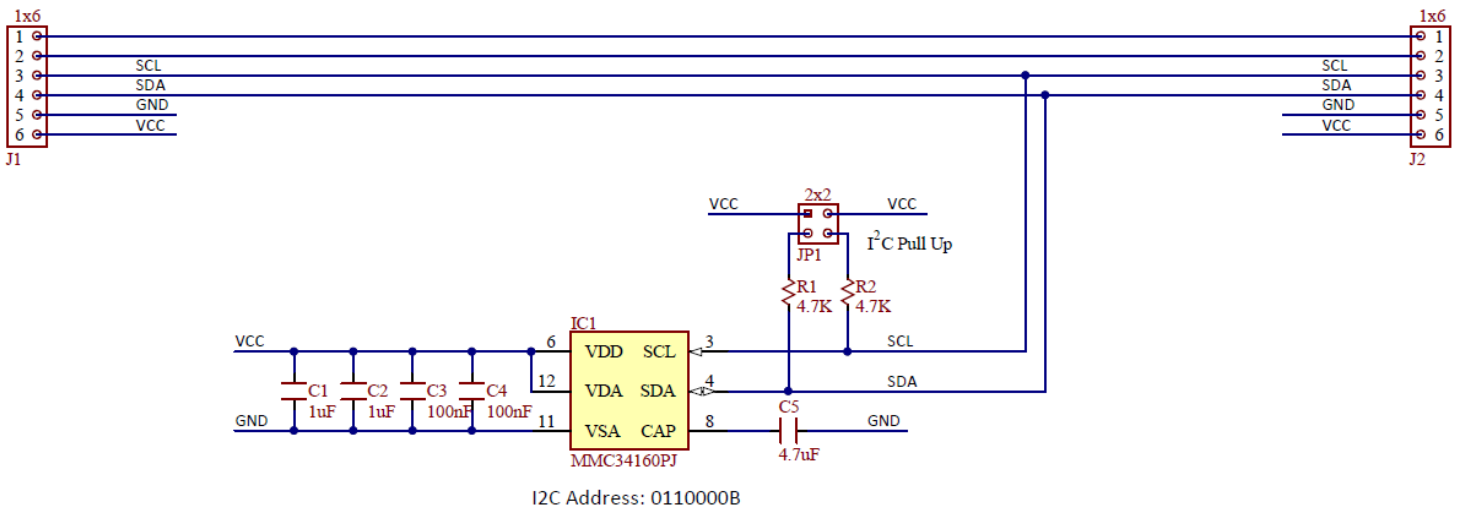


Figure 6. 6 CMPS2 Schematic

1) Accuracy

The magnetic field strength around can be calculated in a ± 16 Gauss range with a heading exactness of 1° and up to 0.5 mG of resolution. The figures appeared underneath depicts the determinations of the compass.

Parameter	Condition	Value	Units
Total RMS Noise	16 bits at 7.92 ms/S	1.5	mG
Total RMS Noise	16 bits at 4.08 ms/S	2.0	mG
Total RMS Noise	14 bits at 2.16 ms/S	4.0	mG
Total RMS Noise	12 bits at 1.20 ms/S	6.0	mG
Max Output Data Rate	16 bits at 7.92 ms/S	125	Hz
Max Output Data Rate	16 bits at 4.08/S	250	Hz
Max Output Data Rate	14 bits at 2.16 ms/S	450	Hz
Max Output Data Rate	12 bits at 1.20 ms/S	800	Hz

Parameter	Value	Units
Field Range for Each Axis	± 16	G

Parameter	Min	Typical	Max	Units
Power Supply Voltage	1.62	1.8	3.6	V
Output Resolution	12	14	16	bits
Alignment Error	-3	± 1	+3	Degrees

Table 6. 3 Specification of CMPS2

2) Data Conversion

The conversion of data is discussed in detail in Chapter 5 which shows the techniques used to determine the direction D. The direction is declared as per the calculation of D which is shown below:

- If D is greater than 337.25 degrees or less than 22.5 degrees – North
- If D is between 292.5 degrees and 337.25 degrees – North-West
- If D is between 247.5 degrees and 292.5 degrees – West
- If D is between 202.5 degrees and 247.5 degrees – South-West
- If D is between 157.5 degrees and 202.5 degrees – South
- If D is between 112.5 degrees and 157.5 degrees – South-East
- If D is between 67.5 degrees and 112.5 degrees – East
- If D is between 0 degrees and 67.5 degrees – North-East

3) CMPS2 and FPGA interfacing

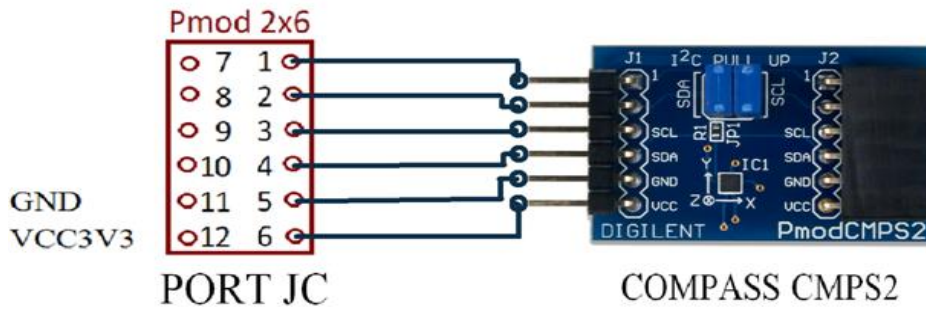
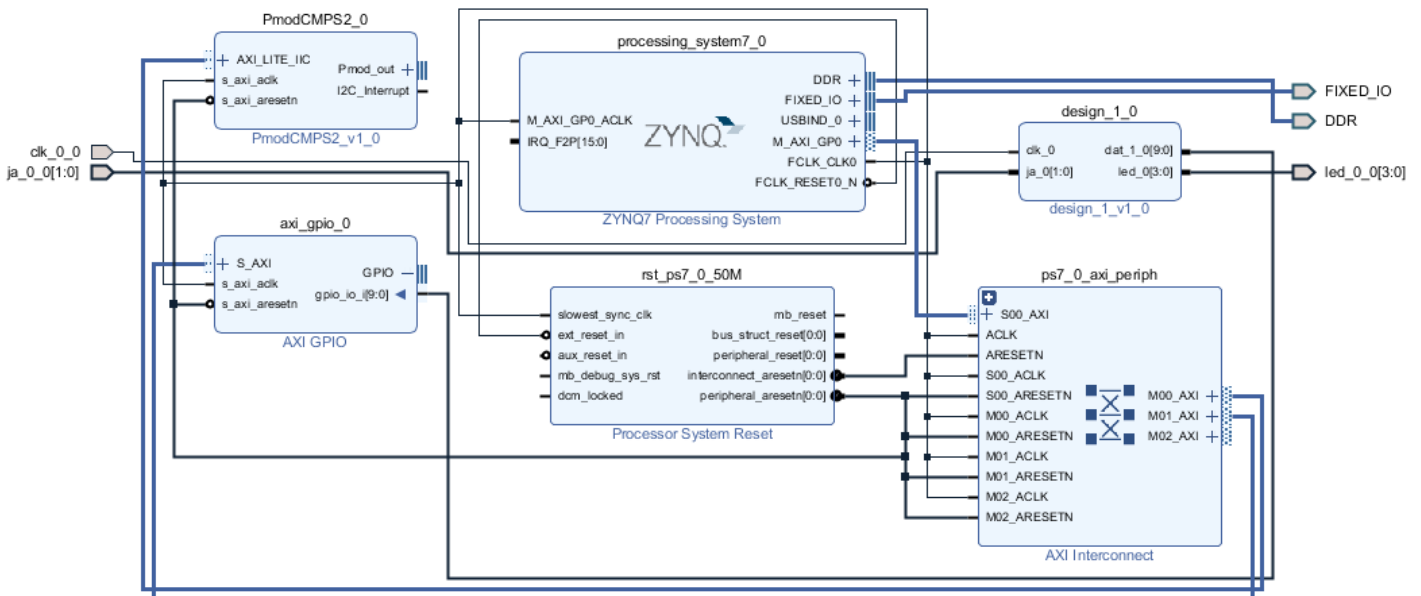


Figure 6. 7 Connection of FPGA (port JC) with CMPS2

PL and PS interfacing:

As the sensors were interfaced with the board, in case of compass the data was already in Ps part of board while for temperature it was in PL. In order to transmit the data using any protocol we needed to bring the processed data first to PS part of the system and then it would be able to transmit successfully. For the said condition, in our initial block design we added a custom XADC ip which is basically an analog to digital converter that transmits 10 bits of data through AXI GPIO to the processing system in case of temperature. Meanwhile for compass the Pmod_cmps2 ip was attached with AXI I2C to the processing system that reads the data and then writes it to any computer application. The baud rate selected from transmission was 115200 and transmission to the gui was done using UART. The following block design shows the integration of both modules together.



Software performance testing:

Performance testing is the process of determining the speed, responsiveness and stability of a computer, network, software program or device under a workload. To address the need for performance analysis and benchmarking techniques, the Xilinx Software Development Kit (SDK) has been enhanced with a System Performance Analysis (SPA) toolbox to provide early exploration of hardware and software systems. Specifically, a Zynq-7000 SoC designer is presented with insights into both the PS and the PL to understand the interactions across such a complex, heterogeneous system. You can observe system performance at critical stages of a design flow, enabling you to refine the performance of your system.

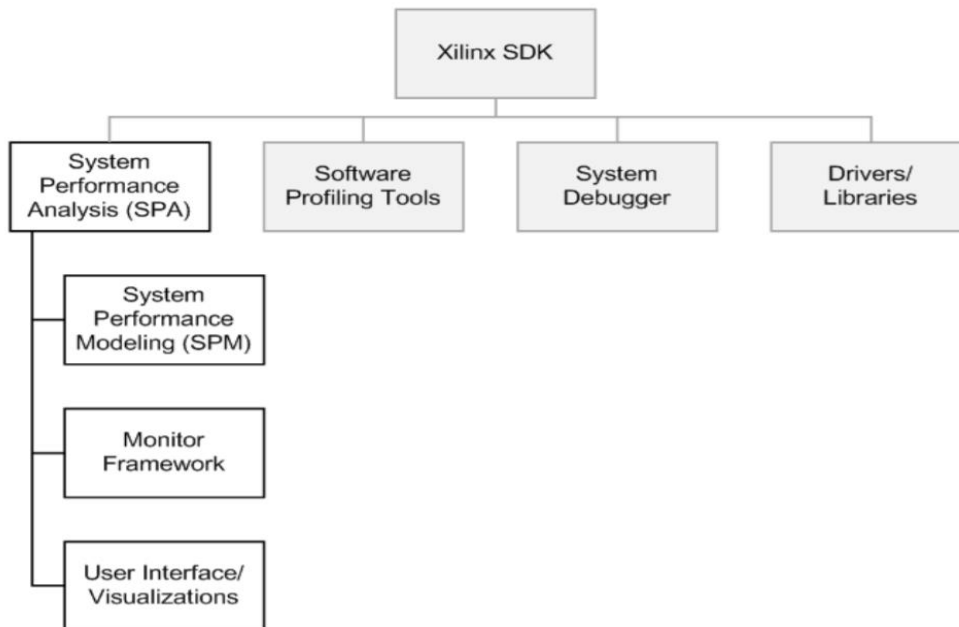


Figure 6. 8 Xilinx SDK Features Including the System Performance Analysis Toolbox

Usability testing:

Usability testing shows how simple is it to utilize and user friendly a product framework is for genuine clients. This testing for the most part centers around the client's simplicity to utilize the application, adaptability in taking care of controls and the capacity of the framework to meet its destinations. Usability testing decides if are applications are valuable, findable, open, usable and attractive. The point of this testing is to fulfill clients and it predominantly focuses on the accompanying boundaries:

- The system effectiveness
- Efficiency
- Accuracy
- Friendliness

Before one performs a usability test, it must be clear who their target audience is and the testing procedure varies accordingly. Our project targets a very specific group of people like Aerospace and Defense sector, large scale industries, Security systems and Medical Electronics who demand long term availability, a fast and efficient system with massively parallel data processing. The programming used in FPGA is not as simple as C/C++ programming used in processor based hardware and FPGA's are not the first option for the usage of general public since FPGA's are expensive and not easy to use, it makes them less user-friendly and to be used by professionals. FPGA's come into play where a very fast data processing system is required and some specific applications of an FPGA include digital signal processing, bioinformatics, medical imaging, voice recognition and many more.

Installation testing:

Installation testing is done to look at if the software has been effectively installed with all the characteristic highlights and that the item is filling in according to desires. Installation testing helps in the distinguishing even the smallest errors. It is otherwise called Execution testing. Installation testing is done to guarantee the given points underneath:

- To make sure that the software would perform well as desired after the installation is done.
- To make sure the security properties of the software are not lost after the installation.
- To make sure that the software does not consume the hardware resources abnormally high making the system slower, after the installation.
- To check whether the software is able to create its own directory in the primary drive of the system, if not customized by the user.
- To check whether the user is allowed to see the progress of the installation is visible through the GUI.
- The installer should have a specified option which should uninstall the software the same way it was installed.
- To make sure that not only the software but the packages and libraries should also work after the installation

Exception handling

In safety critical systems, nothing is allowed to go unknown or unacknowledged, every possible condition (behavior) must be accounted for, and the hardware and software must deal with every possible behavior. The Zynq-7000 All Programmable SoC with its double ARM Cortex A9 processors, and double Neon gliding point units have a table of addresses where the execution is coordinated when something occurs, and interrupts the currently running process on one or the other CPU, and then goes to the interrupt handling code segment on that CPU.² When serviced (finished handling the condition), the CPU returns to where it had been when it was interrupted and continues from there.

For exceptions, one needs to deal with all seven possible exceptions in a manner that allows the system to continue or recover, or take specific actions to behave properly. The exceptions are as follow:

- Reset.
- Undefined Instruction.
- Undefined Software Interrupt.
- Execution from an Undefined Address
- Operating on Data from an Undefined Address.
- IRQ Interrupt.
- Fast Interrupt Exception.

Compatibility testing:

Compatibility testing is a part of non-functional testing conducted on application software to ensure the application's compatibility with different computing environment. The platform that we have used is Vivado 2018.2 by Xilinx. Sensors designed by digilent are all compatible with the ZYNQ Zybo Z7 board which is connected to a PC and a hardware descriptive language is used to run the hardware through the software. The FPGA board is connected via a USB to the computer and Vivado 2018.2 is supported by Windows XP Pro and higher versions.

Graphical user interface testing:

The Graphic user interface is designed in Python for displaying the useful information. The tool used for making the GUI is PyQt5 which is a cross-platform GUI toolkit. The data is collected from the sensors and is processed on the FPGA board which can either be transmitted through an Ethernet connection or the date can directly be displayed on the GUI using UART protocols.

Limitations:

FPGAs are fast and much more flexible than microcontrollers and because to this high processing ability the power consumption of FPGA is more and programmers do not have any control on power optimization in FPGA whereas no such issues are in faced in ASIC. If you have something very simple implemented in a microcontroller, and you want to do the same in an FPGA, it will likely take you more time to get it running in an FPGA, will likely need to run at a slower clock speed, and will possibly use more power. The temperature sensor integrated with the FPGA board is a cheap and easy to use component but it cannot operate in liquid which is a limitation and the Compass sensor runs electrically, so if electricity fails the gyro will also fail thus making the system vulnerable in some conditions along with that saving images directly to block ram for processing can be done only by adding compressed images as the block ram in Zybo Z7-10 is 256kbytes, therefore it would be difficult to see the images later due to compression however if camera is integrated for real time video capture it would be able to work at peak performance.

Evaluation and Results:

Temperature sensor LM35:

The figures shows below demonstrates the results obtained after implementing the verilog coding in Vivado and tested in real time. As soon as the temperature sensor LM35 starts to detect a rise in temperature the change in temperature is measured through the PMOD XADC and the data is processed in the Programmable Logic (PL) part of the FPGA and the results are obtained as shown below:

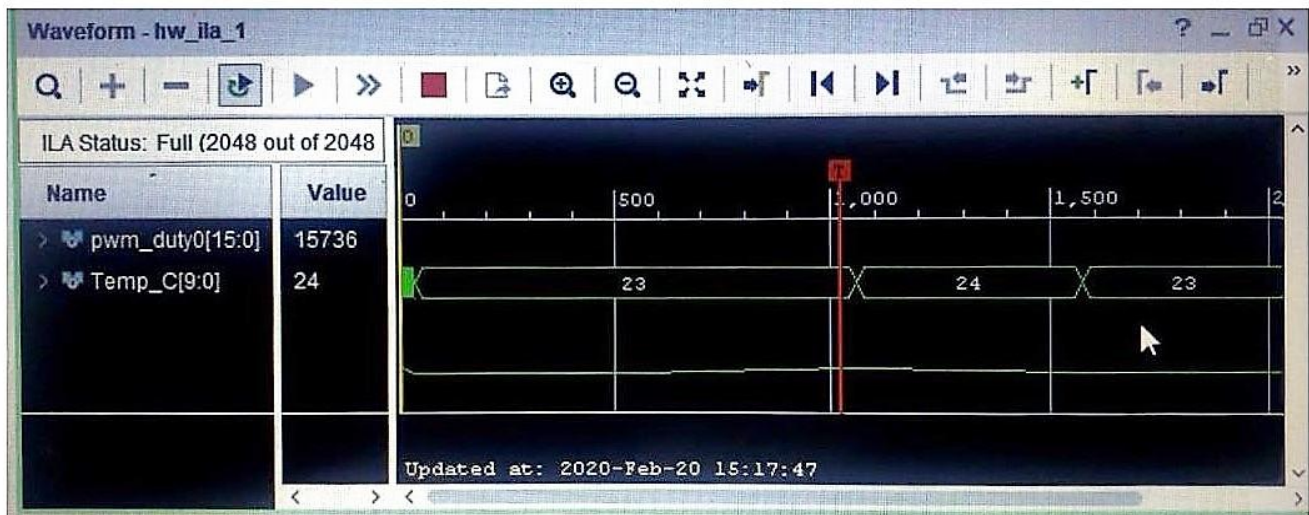


Figure 6.9 Room Temperature

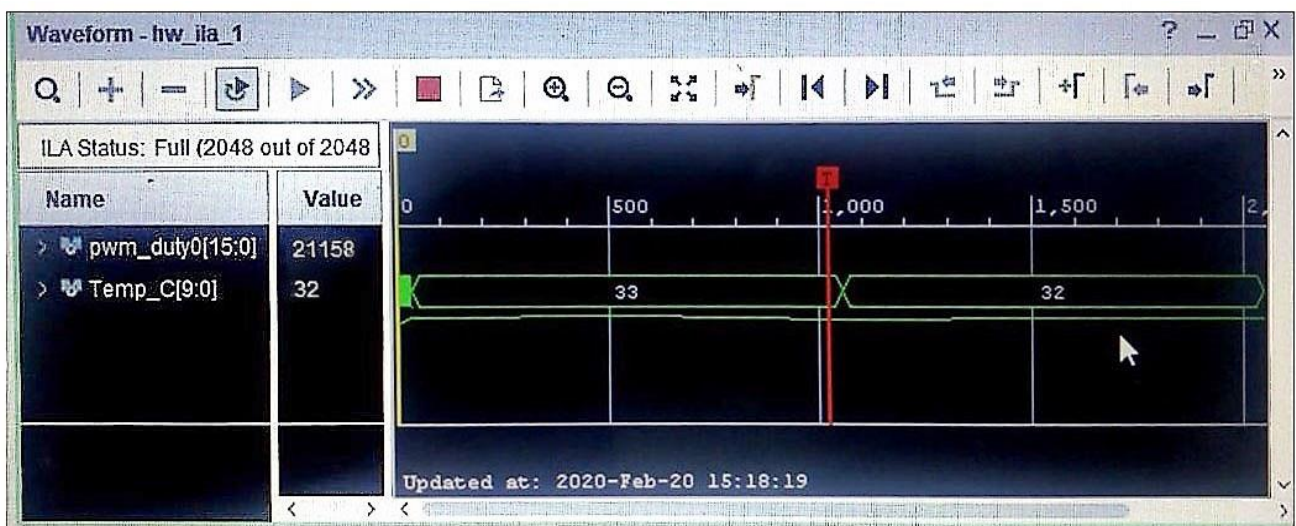
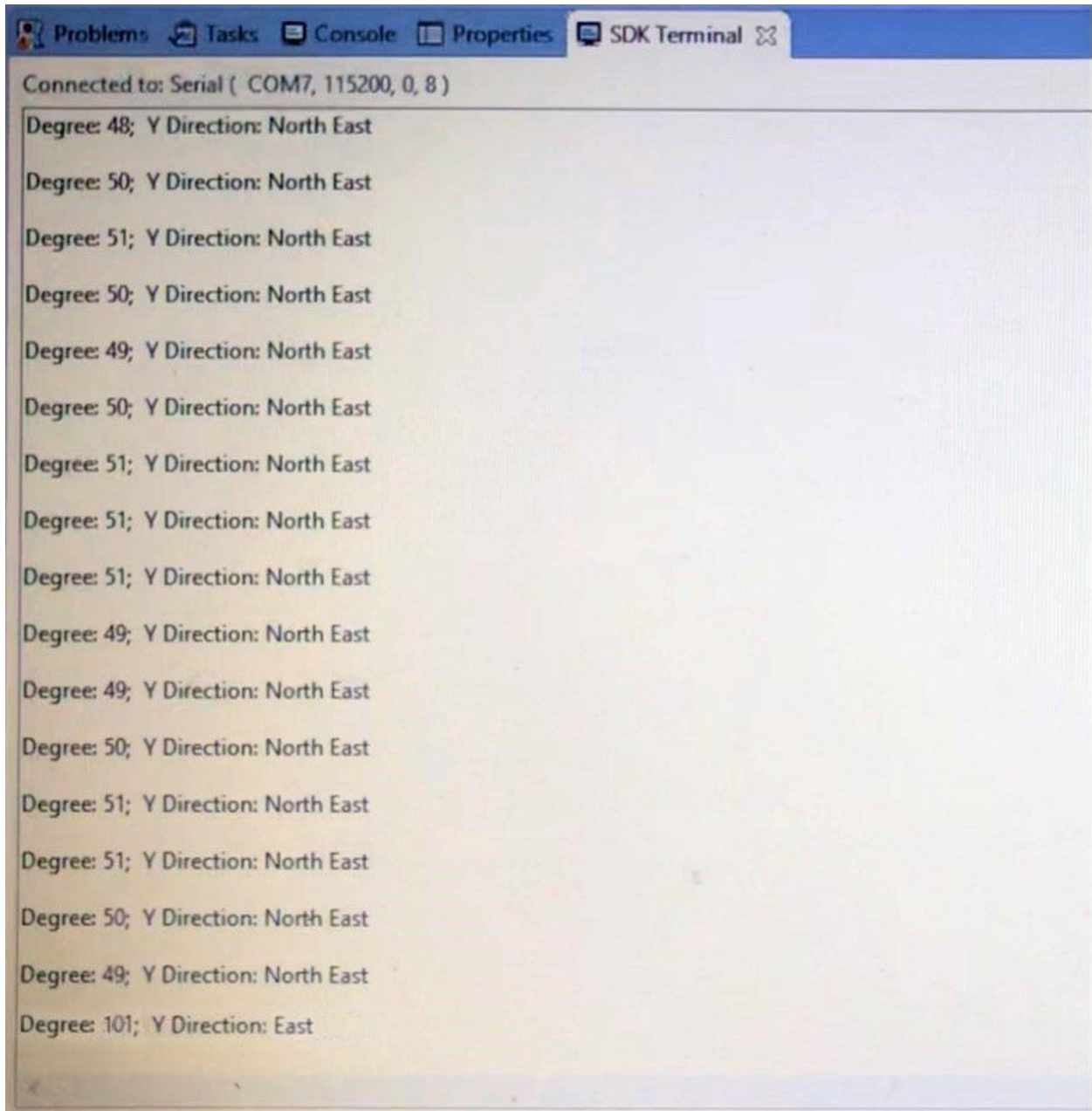


Figure 6.10 Rise in Temperature

Compass Pmod CMPS2:

When the connection is maintained and the compass is integrated with the FPGA. The SDK tool in Vivado is used to display the Output of the compass. The conversion of data is done as stated in “Data conversion” under Components testing to obtain the results. As the magnetic sensor detects a change in motion, the data is processed and measured the change and it displays the Degree with the direction as shown in the figure below:

The image shows a screenshot of an SDK Terminal window. The title bar includes tabs for 'Problems', 'Tasks', 'Console', 'Properties', and 'SDK Terminal'. The terminal content shows a connection to a serial port: 'Connected to: Serial (COM7, 115200, 0, 8)'. Below this, there is a list of 16 lines of output, each representing a compass reading. The first 15 lines all show a 'Y Direction: North East' with degrees ranging from 48 to 51. The final line shows a 'Y Direction: East' with a degree of 101.

```
Connected to: Serial ( COM7, 115200, 0, 8 )
Degree: 48; Y Direction: North East
Degree: 50; Y Direction: North East
Degree: 51; Y Direction: North East
Degree: 50; Y Direction: North East
Degree: 49; Y Direction: North East
Degree: 50; Y Direction: North East
Degree: 51; Y Direction: North East
Degree: 51; Y Direction: North East
Degree: 51; Y Direction: North East
Degree: 49; Y Direction: North East
Degree: 49; Y Direction: North East
Degree: 50; Y Direction: North East
Degree: 51; Y Direction: North East
Degree: 51; Y Direction: North East
Degree: 50; Y Direction: North East
Degree: 49; Y Direction: North East
Degree: 101; Y Direction: East
```

Figure 6. 11 Variation in compass direction

Results shown on GUI:

The figure below shows the final outputs when all the sensors have been integrated together and the processed data is transferred to the GUI whereas in figure 6.10 and 6.11, the outputs shown were for 2 different individual projects.



Figure 6.12 Graphical User Interface

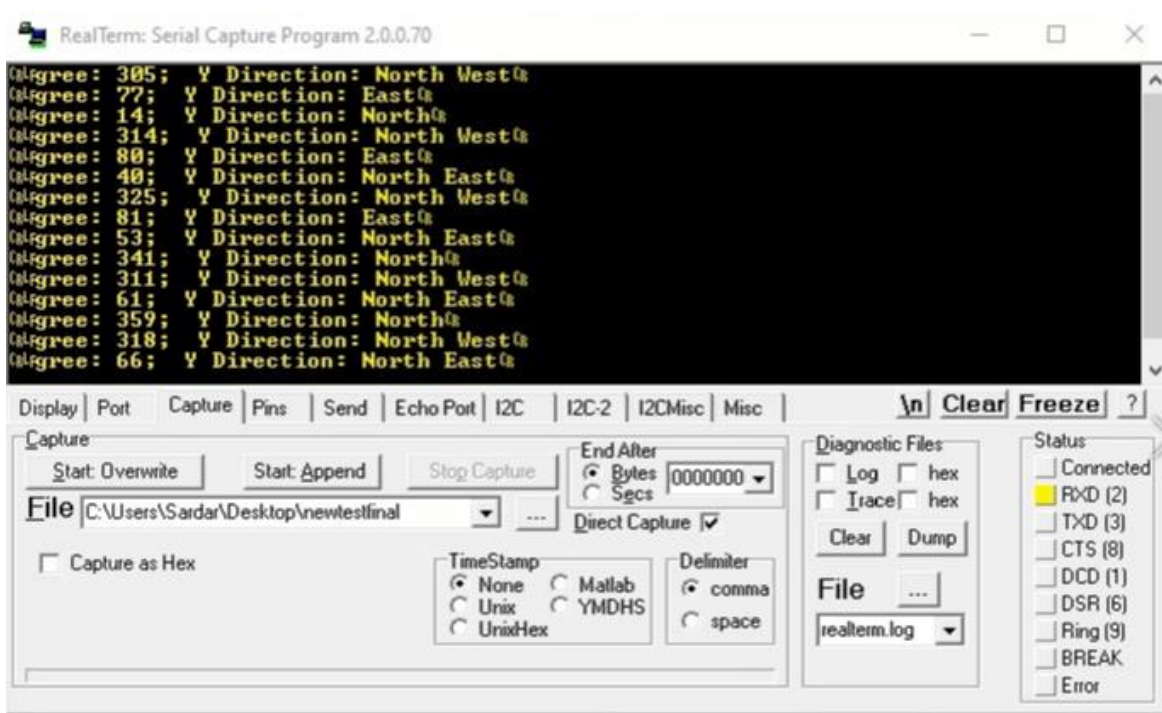


Figure 6.13 Compass Output on RealTerm

Chapter # 7

Conclusion

This work included planning and building up a system that furnishes clients valuable information (useful data) just with a FPGA board Zynq Z7-10 as equipment that enables clients to design and connect with different FPGA boards as well as cores. We showed that the system is adaptable, in that it could suit different application necessities, for example, various sources of inputs and coordination of various sensors on one board. We likewise indicated that this system is versatile, in that it could oblige various equipment cores and boards of FPGA. Utilizing an algorithm that was created to take information from sensors and transmitting just the useful data, we examined different system designs to watch the impacts of joining extra equipment segments (sensors) on execution times and furthermore decide the size of the general system with these increases, which came out to be a compact system. FPGAs separate themselves in profoundly parallelized tasks.. While present day microprocessors execute operations on various cores with sequential and faulty directions, not all capacities are appropriate to be worked by them, like digital signal processing and time critical applications. As a bonus, FPGAs now allow System-on-chip based devices where processor can be interconnected with FPGA resources allowing users to put right use of these resources depending on different scenarios like if the application is time critical or not. Additionally, FPGA offer more reliability, power and performance efficiency, security as well as the feature of putting all the modules on a single chip provides multiple advantages over microcontrollers. Keeping all this in mind, Zybo Z-10 was chosen which contains enough resources for the application we desired.

For the purpose of learning, multiple modules were selected so a range of features on Zybo board can be explored. Analog sensors as well as digital sensors were implemented using different communication protocols including I2C. Additionally, sensors were implemented and were operated using Programmable logic and processing system so the FPGA resources and Processor resources can be put to use separately depending upon application requirement.

So, all in all, the proposed system would work in a set of principles. Firstly, data from sensors will be extracted depending upon if the data is analog or digital. Secondly, applying specific algorithms to extract useful information from the extracted data. And finally, meaningful data will be transmitted and fed into Graphic user interface. This proposed system offers more effectiveness since there is no need to send all the extracted data from sensors to be sent to a processing hub where it can be processes and then fed into the Graphic user interface rather the proposed system allows the user to filter the data right after the sensor so only the meaningful data can be sent offering more efficiency as well as taking less time since resources of FPGA were used.

As mentioned already in earlier chapters, the application of proposed system doesn't just limit to certain sensors rather the sensors used in project are more of a learning purpose subjects. Any application can be made possible on the device which can be achieved by developing a corresponding algorithm for that application and following the set of principles proposed in the system. However the limitation of integrating different Pmods to the Zynq Z7-10 board allows only six external hardware components to work at a given instance together hence in order to incorporate more than six hardware components (sensors) an additional FPGA board would be required.

References

[1] Karen Parnell, Roger Bryner. “Comparing and Contrasting FPGA and Microprocessor System Design and Development”. Xilinx, WP213 (v1.1) July 21, 2004

[2] Rene Mueller and Jens Teubner. “Data Processing on FPGAs”. Systems Group, Department of Computer Science, ETH Zurich, Switzerland, 2013

[3] F. Bensaali and A. Amira. “Design and Implementation of Efficient Architectures for Color Space Conversion”. ICGST-GVIP Journal, 1(5), 2004, 37-47

[4] S.Velusamy, Wei Huang, J. Lach, M. Stan and K. Skadron. “Monitoring temperature in FPGA based SoCs”. IEEE International Conference on Computer Design, 2005, 634 - 37.

[5] Atibi mohamed, Benrabh Mohamed, Atouf Issam, Boussaa Mohamed, Bennis Abdellatif. “Implementation of a vehicle detection system in the FPGA embedded platform”. Journal of Theoretical and Applied Information Technology”, 2017.

[6] Ms.Rachna Singh and Dr.Arvind Rajawat. “Interfacing the Analog Camera with FPGA Board for Real-time Video Acquisition”. MECS, 2014, 32-38

[7] Ms.Rachna Singh and Dr.Arvind Rajawat. “ A Review of FPGA-based design methodologies for efficient hardware Area estimation”. IOSR Journal of Computer Engineering (IOSR-JCE),2013

Appendices

APPENDIX A: Verilog Implementation

A1. Design code for Compass

```
`timescale 1 ps / 1 ps
module PmodCMPS2
  (AXI_LITE_IIC_araddr,
   AXI_LITE_IIC_arready,
   AXI_LITE_IIC_arvalid,
   AXI_LITE_IIC_awaddr,
   AXI_LITE_IIC_awready,
   AXI_LITE_IIC_awvalid,
   AXI_LITE_IIC_bready,
   AXI_LITE_IIC_bresp,
   AXI_LITE_IIC_bvalid,
   AXI_LITE_IIC_rdata,
   AXI_LITE_IIC_rready,
   AXI_LITE_IIC_rresp,
   AXI_LITE_IIC_rvalid,
   AXI_LITE_IIC_wdata,
   AXI_LITE_IIC_wready,
   AXI_LITE_IIC_wstrb,
   AXI_LITE_IIC_wvalid,
   I2C_Interrupt,
   Pmod_out_pin10_i,
   Pmod_out_pin10_o,
   Pmod_out_pin10_t,
   Pmod_out_pin1_i,
   Pmod_out_pin1_o,
   Pmod_out_pin1_t,
   Pmod_out_pin2_i,
   Pmod_out_pin2_o,
   Pmod_out_pin2_t,
   Pmod_out_pin3_i,
   Pmod_out_pin3_o,
   Pmod_out_pin3_t,
   Pmod_out_pin4_i,
   Pmod_out_pin4_o,
   Pmod_out_pin4_t,
   Pmod_out_pin7_i,
   Pmod_out_pin7_o,
   Pmod_out_pin7_t,
   Pmod_out_pin8_i,
   Pmod_out_pin8_o,
   Pmod_out_pin8_t,
   Pmod_out_pin9_i,
   Pmod_out_pin9_o,
   Pmod_out_pin9_t,
   s_axi_aclk,
   s_axi_aresetn);
input [8:0]AXI_LITE_IIC_araddr;
output AXI_LITE_IIC_arready;
```

```

input AXI_LITE_IIC_arvalid;
input [8:0]AXI_LITE_IIC_awaddr;
output AXI_LITE_IIC_awready;
input AXI_LITE_IIC_awvalid;
input AXI_LITE_IIC_bready;
output [1:0]AXI_LITE_IIC_bresp;
output AXI_LITE_IIC_bvalid;
output [31:0]AXI_LITE_IIC_rdata;
input AXI_LITE_IIC_rready;
output [1:0]AXI_LITE_IIC_rresp;
output AXI_LITE_IIC_rvalid;
input [31:0]AXI_LITE_IIC_wdata;
output AXI_LITE_IIC_wready;
input [3:0]AXI_LITE_IIC_wstrb;
input AXI_LITE_IIC_wvalid;
output I2C_Interrupt;
input Pmod_out_pin10_i;
output Pmod_out_pin10_o;
output Pmod_out_pin10_t;
input Pmod_out_pin1_i;
output Pmod_out_pin1_o;
output Pmod_out_pin1_t;
input Pmod_out_pin2_i;
output Pmod_out_pin2_o;
output Pmod_out_pin2_t;
input Pmod_out_pin3_i;
output Pmod_out_pin3_o;
output Pmod_out_pin3_t;
input Pmod_out_pin4_i;
output Pmod_out_pin4_o;
output Pmod_out_pin4_t;
input Pmod_out_pin7_i;

output Pmod_out_pin7_o;
output Pmod_out_pin7_t;
input Pmod_out_pin8_i;
output Pmod_out_pin8_o;
output Pmod_out_pin8_t;
input Pmod_out_pin9_i;
output Pmod_out_pin9_o;
output Pmod_out_pin9_t;
input s_axi_aclk;
input s_axi_aresetn;

```

```

wire [31:0]S_AXI_1_RDATA;
wire S_AXI_1_RREADY;
wire [1:0]S_AXI_1_RRESP;
wire S_AXI_1_RVALID;
wire [31:0]S_AXI_1_WDATA;
wire S_AXI_1_WREADY;
wire [3:0]S_AXI_1_WSTRB;
wire S_AXI_1_WVALID;

```

```

wire axi_iic_0_IIC_SCL_I;
wire axi_iic_0_IIC_SCL_O;
wire axi_iic_0_IIC_SCL_T;
wire axi_iic_0_IIC_SDA_I;
wire axi_iic_0_IIC_SDA_O;
wire axi_iic_0_IIC_SDA_T;
wire [1:0]axi_iic_0_gpo;
wire axi_iic_0_iic2intc_irpt;
wire pmod_bridge_0_Pmod_out_PIN10_I;
wire pmod_bridge_0_Pmod_out_PIN10_O;
wire pmod_bridge_0_Pmod_out_PIN10_T;
wire pmod_bridge_0_Pmod_out_PIN1_I;
wire pmod_bridge_0_Pmod_out_PIN1_O;
wire pmod_bridge_0_Pmod_out_PIN1_T;
wire pmod_bridge_0_Pmod_out_PIN2_I;
wire pmod_bridge_0_Pmod_out_PIN2_O;
wire pmod_bridge_0_Pmod_out_PIN2_T;
wire pmod_bridge_0_Pmod_out_PIN3_I;
wire pmod_bridge_0_Pmod_out_PIN3_O;
wire pmod_bridge_0_Pmod_out_PIN3_T;
wire pmod_bridge_0_Pmod_out_PIN4_I;
wire pmod_bridge_0_Pmod_out_PIN4_O;
wire pmod_bridge_0_Pmod_out_PIN4_T;
wire pmod_bridge_0_Pmod_out_PIN7_I;
wire pmod_bridge_0_Pmod_out_PIN7_O;
wire pmod_bridge_0_Pmod_out_PIN7_T;
wire pmod_bridge_0_Pmod_out_PIN8_I;
wire pmod_bridge_0_Pmod_out_PIN8_O;
wire pmod_bridge_0_Pmod_out_PIN8_T;
wire pmod_bridge_0_Pmod_out_PIN9_I;
wire pmod_bridge_0_Pmod_out_PIN9_O;
wire pmod_bridge_0_Pmod_out_PIN9_T;
wire s_axi_aclk_1;
wire s_axi_aresetn_1;
wire [1:0]xlconstant_0_dout;

```

```

assign Pmod_out_pin3_o = pmod_bridge_0_Pmod_out_PIN3_O;
assign Pmod_out_pin3_t = pmod_bridge_0_Pmod_out_PIN3_T;
assign Pmod_out_pin4_o = pmod_bridge_0_Pmod_out_PIN4_O;
assign Pmod_out_pin4_t = pmod_bridge_0_Pmod_out_PIN4_T;
assign Pmod_out_pin7_o = pmod_bridge_0_Pmod_out_PIN7_O;
assign Pmod_out_pin7_t = pmod_bridge_0_Pmod_out_PIN7_T;
assign Pmod_out_pin8_o = pmod_bridge_0_Pmod_out_PIN8_O;
assign Pmod_out_pin8_t = pmod_bridge_0_Pmod_out_PIN8_T;
assign Pmod_out_pin9_o = pmod_bridge_0_Pmod_out_PIN9_O;
assign Pmod_out_pin9_t = pmod_bridge_0_Pmod_out_PIN9_T;
assign S_AXI_1_ARADDR = AXI_LITE_IIC_araddr[8:0];
assign S_AXI_1_ARVALID = AXI_LITE_IIC_arvalid;
assign S_AXI_1_AWADDR = AXI_LITE_IIC_awaddr[8:0];
assign S_AXI_1_AWVALID = AXI_LITE_IIC_awvalid;

```



```

assign S_AXI_1_BREADY = AXI_LITE_IIC_bready;
assign S_AXI_1_RREADY = AXI_LITE_IIC_rready;
assign S_AXI_1_WDATA = AXI_LITE_IIC_wdata[31:0];
assign S_AXI_1_WSTRB = AXI_LITE_IIC_wstrb[3:0];
assign S_AXI_1_WVALID = AXI_LITE_IIC_wvalid;
assign pmod_bridge_0_Pmod_out_PIN10_I = Pmod_out_pin10_i;
assign pmod_bridge_0_Pmod_out_PIN1_O = Pmod_out_pin1_o;
assign pmod_bridge_0_Pmod_out_PIN2_I = Pmod_out_pin2_i;
assign pmod_bridge_0_Pmod_out_PIN3_I = Pmod_out_pin3_i;
assign pmod_bridge_0_Pmod_out_PIN4_I = Pmod_out_pin4_i;
assign pmod_bridge_0_Pmod_out_PIN7_I = Pmod_out_pin7_i;
assign pmod_bridge_0_Pmod_out_PIN8_I = Pmod_out_pin8_i;
assign pmod_bridge_0_Pmod_out_PIN9_I = Pmod_out_pin9_i;
assign s_axi_aclk_1 = s_axi_aclk;
assign s_axi_aresetn_1 = s_axi_aresetn;
PmodCMPS2_axi_iic_0_0_axi_iic_0
(.gpo(axi_iic_0_gpo),
.iic2intc_irpt(axi_iic_0_iic2intc_irpt),
.s_axi_aclk(s_axi_aclk_1),
.s_axi_araddr(S_AXI_1_ARADDR),
.s_axi_aresetn(s_axi_aresetn_1),
.s_axi_arready(S_AXI_1_ARREADY),
.s_axi_arvalid(S_AXI_1_ARVALID),
.s_axi_awaddr(S_AXI_1_AWADDR),
.s_axi_awready(S_AXI_1_AWREADY),
.s_axi_awvalid(S_AXI_1_AWVALID),
.s_axi_bready(S_AXI_1_BREADY),
.s_axi_bresp(S_AXI_1_BRESP),
.s_axi_bvalid(S_AXI_1_BVALID),
.s_axi_rdata(S_AXI_1_RDATA),
.s_axi_rready(S_AXI_1_RREADY),
.s_axi_rresp(S_AXI_1_RRESP),
.s_axi_rvalid(S_AXI_1_RVALID),
.s_axi_wdata(S_AXI_1_WDATA),
.s_axi_wready(S_AXI_1_WREADY),
.s_axi_wstrb(S_AXI_1_WSTRB),
.s_axi_wvalid(S_AXI_1_WVALID),
.scl_i(axi_iic_0_IIC_SCL_I),
.scl_o(axi_iic_0_IIC_SCL_O),
.scl_t(axi_iic_0_IIC_SCL_T),
.sda_i(axi_iic_0_IIC_SDA_I),
.sda_o(axi_iic_0_IIC_SDA_O),
.sda_t(axi_iic_0_IIC_SDA_T));
PmodCMPS2_pmod_bridge_0_0_pmod_bridge_0
(.in2_I(axi_iic_0_IIC_SCL_I),
.in2_O(axi_iic_0_IIC_SCL_O),
.in2_T(axi_iic_0_IIC_SCL_T),
.in3_I(axi_iic_0_IIC_SDA_I),
.in3_O(axi_iic_0_IIC_SDA_O),
.in3_T(axi_iic_0_IIC_SDA_T),
.in_top_i2c_gpio_bus_O(axi_iic_0_gpo),
.in_top_i2c_gpio_bus_T(xlconstant_0_dout),
.out0_I(pmod_bridge_0_Pmod_out_PIN1_I),
.out0_O(pmod_bridge_0_Pmod_out_PIN1_O),
.out0_T(pmod_bridge_0_Pmod_out_PIN1_T),

```

```

.out1_I(pmod_bridge_0_Pmod_out_PIN2_I),
.out1_O(pmod_bridge_0_Pmod_out_PIN2_O),
.out1_T(pmod_bridge_0_Pmod_out_PIN2_T),
.out2_I(pmod_bridge_0_Pmod_out_PIN3_I),
.out2_O(pmod_bridge_0_Pmod_out_PIN3_O),
.out2_T(pmod_bridge_0_Pmod_out_PIN3_T),
.out3_I(pmod_bridge_0_Pmod_out_PIN4_I),
.out3_O(pmod_bridge_0_Pmod_out_PIN4_O),
.out3_T(pmod_bridge_0_Pmod_out_PIN4_T),
.out4_I(pmod_bridge_0_Pmod_out_PIN7_I),
.out4_O(pmod_bridge_0_Pmod_out_PIN7_O),
.out4_T(pmod_bridge_0_Pmod_out_PIN7_T),
.out5_I(pmod_bridge_0_Pmod_out_PIN8_I),
.out5_O(pmod_bridge_0_Pmod_out_PIN8_O),
.out5_T(pmod_bridge_0_Pmod_out_PIN8_T),
.out6_I(pmod_bridge_0_Pmod_out_PIN9_I),
.out6_O(pmod_bridge_0_Pmod_out_PIN9_O),
.out6_T(pmod_bridge_0_Pmod_out_PIN9_T),
.out7_I(pmod_bridge_0_Pmod_out_PIN10_I),
.out7_O(pmod_bridge_0_Pmod_out_PIN10_O),
.out7_T(pmod_bridge_0_Pmod_out_PIN10_T));
PmodCMPS2_xlconstant_0_0 xlconstant_0
(.dout(xlconstant_0_dout));
endmodule

```

A2. Design code for LM35 (Temperature Sensor)

```

`timescale 1ns / 1ps

module top(
    input clk,
    input [1:0] ja,
    output [3:0] led
);
    reg [6:0] daddr = 0; // address of channel to be read
    reg [1:0] ledidx = 0; // index of the led to capture data for

    wire eoc; // xadc end of conversion flag
    wire [15:0] dout; // xadc data out bus
    wire drdy;

    reg [1:0] _drdy = 0; // delayed data ready signal for edge detection

    reg [15:0] data0 = 0, // stored XADC data, only the uppermost byte
        data1 = 0,
        data2 = 0,
        data3 = 0;

    reg [7:0] pwm_count; // shared pwm counter

```

```

reg [15:0] pwm_duty0; // duty cycles for the 4 pwm led brightness controllers
reg [15:0] pwm_duty1;
reg [15:0] pwm_duty2;
reg [15:0] pwm_duty3;

xadc_wiz_0 myxadc (
    .clk_in      (clk),
    .den_in      (eoc), // drp enable, start a new conversion whenever the last one has ended
    .dwe_in      (0),
    .daddr_in    (daddr), // channel address
    .di_in       (0),
    .do_out      (dout), // data out
    .drdy_out    (drdy), // data ready
    .eoc_out     (eoc), // end of conversion

    // .vauxn6     (ja[7]),
    // .vauxp6     (ja[3]),

    // .vauxn7     (ja[5]),
    // .vauxp7     (ja[1]),

    // .vauxn15    (ja[6]),
    // .vauxp15    (ja[2]),

    // .vauxn14    (ja[4]),
    // .vauxp14    (ja[0])

    .vauxn14     (ja[1]),
    .vauxp14     (ja[0])
);

always@(posedge clk)
    _drdy <= {_drdy[0], drdy};

always@(*)
    case (ledidx)
    0: daddr = 7'h1E;
    1: daddr = 7'h17;
    2: daddr = 7'h1F;
    3: daddr = 7'h16;
    default: daddr = 7'h1E;
    endcase

always@(posedge clk) begin
    if (_drdy == 2'b10) begin // on negative edge
        ledidx <= ledidx + 1;
        case (ledidx)
        0: data0 <= dout;
        1: data1 <= dout;
        2: data2 <= dout;
        3: data3 <= dout;
        endcase
    end
end

```

```

        endcase
    end
end

always@(posedge clk)
    pwm_count <= pwm_count + 1;

always@(posedge clk)
    if (pwm_count == 0) begin
        pwm_duty0 <= data0;
        pwm_duty1 <= data1;
        pwm_duty2 <= data2;
        pwm_duty3 <= data3;
    end

assign led[0] = (pwm_count <= pwm_duty0) ? 1 : 0;
assign led[1] = (pwm_count <= pwm_duty1) ? 1 : 0;
assign led[2] = (pwm_count <= pwm_duty2) ? 1 : 0;
assign led[3] = (pwm_count <= pwm_duty3) ? 1 : 0;

wire [9:0] Temp_C;
assign Temp_C = pwm_duty0 * 'd100 /'d65536;

ila_0 your_instance_name (
    .clk(clk), // input wire clk

    .probe0(pwm_duty0), // input wire [15:0] probe0
    .probe1(Temp_C) // input wire [9:0] probe1
);

Endmodule

```

APPENDIX B: C Language Implementation

B1. Design code for Compass in C

```

#include <stdio.h>
#include "math.h"
#include "PmodCMPS2.h"
#include "sleep.h"
#include "xil_cache.h"
#include "xparameters.h"

// Calibration data struct, track minimum, maximum, and average sample seen for
// each x/y/z channel
typedef struct {
    CMPS2_DataPacket max, min, mid;
} CMPS2_CalibrationData;

void DemoInitialize();

```

```

void DemoRun();
int DemoConvertDegree(PmodCMPS2 *InstancePtr, CMPS2_CalibrationData calib,
    CMPS2_DataPacket data, int declination);
void DemoClearCalibration(CMPS2_CalibrationData *calib);
void DemoCalibrate(PmodCMPS2 *InstancePtr, CMPS2_CalibrationData *calib,
    CMPS2_DataPacket data);
char *DemoGetCardinalDirectionString(int deg, char *cardinal_table[]);
void DemoCleanup();
void EnableCaches();
void DisableCaches();

PmodCMPS2 myDevice;
CMPS2_CalibrationData myCalibrationData;

const int myDeclination = 15; // Magnetic declination for Seattle, WA
const u8 chip_address = 0x30; // I2C chip address

int main(void) {
    DemoInitialize();
    DemoRun();
    DemoCleanup();
    return 0;
}

void DemoInitialize() {
    EnableCaches();
    CMPS2_begin(&myDevice, XPAR_PMODCMPS2_0_AXI_LITE_IIC_BASEADDR,
chip_address);
    usleep(10000);
    CMPS2_SetSensor(&myDevice);
    usleep(10000);
    CMPS2_SetOutputResolution(&myDevice, 0b00);
}

void DemoRun() {
    // FIXME: data becomes invalid when the board is not face up and flat.

    char *cardinal_table[] = {"North", "North East", "East", "South East",
        "South", "South West", "West", "North West"};
    char *cardinal;
    int deg;
    CMPS2_DataPacket data;

    DemoClearCalibration(&myCalibrationData);

    while (1) {
        data = CMPS2_GetData(&myDevice);

        DemoCalibrate(&myDevice, &myCalibrationData, data);

        deg = DemoConvertDegree(&myDevice, myCalibrationData, data,

```

```

    myDeclination);

    cardinal = DemoGetCardinalDirectionString(deg, cardinal_table);

    printf("Degree: %d; Y Direction: %s\r\n", deg, cardinal);

    usleep(100000);
}
}

int DemoConvertDegree(PmodCMPS2 *InstancePtr, CMPS2_CalibrationData calib,
    CMPS2_DataPacket data, int declination) {
    int tx, ty;
    int deg;

    if (data.x < calib.mid.x)
        tx = (calib.mid.x - data.x);
    else
        tx = (data.x - calib.mid.x);

    if (data.y < calib.mid.y)
        ty = (calib.mid.y - data.y);
    else
        ty = (data.y - calib.mid.y);

    if (data.x < calib.mid.x) {
        if (data.y > calib.mid.y)
            deg = 90 - atan2f(ty, tx) * 180 / 3.14159;
        else
            deg = 90 + atan2f(ty, tx) * 180 / 3.14159;
    } else {
        if (data.y < calib.mid.y)

            deg = 270 - atan2f(ty, tx) * 180 / 3.14159;
        else
            deg = 270 + atan2f(ty, tx) * 180 / 3.14159;
    }

    deg += declination;

    while (deg >= 360)
        deg -= 360;
    while (deg < 0)
        deg += 360;

    return deg;
}

void DemoClearCalibration(CMPS2_CalibrationData *calib) {
    calib->max.x = 0x8000; // Center point of 0x0000 -> 0xFFFF
    calib->max.y = 0x8000;
    calib->max.z = 0x8000;
    calib->min.x = 0x8000;
    calib->min.y = 0x8000;
}

```

```

    calib->min.z = 0x8000;
    calib->mid.x = 0x8000;
    calib->mid.y = 0x8000;
    calib->mid.z = 0x8000;
}

void DemoCalibrate(PmodCMPS2 *InstancePtr, CMPS2_CalibrationData *calib,
    CMPS2_DataPacket data) {
    if (data.x > calib->max.x) calib->max.x = data.x; // Track maximum / minimum
    if (data.y > calib->max.y) calib->max.y = data.y; // value seen per axis
    if (data.z > calib->max.z) calib->max.z = data.z;
    if (data.x < calib->min.x) calib->min.x = data.x;
    if (data.y < calib->min.y) calib->min.y = data.y;
    if (data.z < calib->min.z) calib->min.z = data.z;
    calib->mid.x = (calib->max.x >> 1) + (calib->min.x >> 1); // Find average
    calib->mid.y = (calib->max.y >> 1) + (calib->min.y >> 1);
    calib->mid.z = (calib->max.z >> 1) + (calib->min.z >> 1);
}

char *DemoGetCardinalDirectionString(int deg, char *cardinal_table[]) {
    float fdeg = deg;
    if (fdeg > 337.5)
        fdeg -= 337.5;
    else
        fdeg += 22.5;
    fdeg /= 45.0;
    return cardinal_table[(int) fdeg];
}

void DemoCleanup() {
    DisableCaches();
}

void EnableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheEnable();
#endif
#ifdef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheDisable();
#endif
#ifdef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheDisable();
#endif
#endif
}

```

APPENDIX C: Python Language Implementation

C1. Code for GUI (transmission)

```
import csv
import serial

ser = serial.Serial("COM7", 115200)
print("connected to: " + ser.portstr)
line = []
filename = "sensordata.csv"
while True:
    cc=str(ser.readline())
    # writing to csv file
    with open(filename, 'w') as csvfile:
        # creating a csv writer object
        csvwriter = csv.writer(csvfile)
        # writing the data rows
        csvwriter.writerow([[str(cc[2:][:-3])]])
    #self.lcdNumber_4.display(cc[2:][:-3])
    print(cc)
    #QtWidgets.QApplication.processEvents()
    ser.close()
```