Smart Thermostat using FPGA

A Major Qualifying Project Report
Submitted to the Faculty of Worcester Polytechnic Institute

In partial fulfillment of the requirements for the
Degree of Bachelor of Science in Electrical & Computer Engineering

Khoa Nguyen
December 31st, 2015

Advisor: Professor R. James Duckworth

Acknowledgements

I would like to express my sincere gratitude to Professor Duckworth for his indispensable and invaluable guidance throughout the project. His inputs insightful comments had encouraged me to broaden my views and explore different aspects of the project.

I also would like to thank Professor Bitar for advising me to complete analog circuits.

I would also like to thank Drew Vojslavek from Omron Electronic Components LLC, and Steve Courcy from CTC Associates. Inc. fir sending me samples to help me finish the prototype of this project.

Table of Acronyms

AXI	Advanced eXtensible Interface
LUT	Look-up table
FPGA	Field Programmable Gate Array
HVAC	Heating, Ventilating and Air Conditioning
OLED	Organic Light-Emitting Diode
PL	Programmable Logic
PS	Processing System
SoC	System-on-Chip
GPIO	General peripheral in/out
FOV	Field of view
ADC	Analog digital converter
SDA	Serial data
SCL	Serial clock
WPI	Worcester Polytechnic Institute

Abstract

The focus of this project is to develop a smart thermostat that used a human detection to automatically adjust desired indoor temperatures. The system was implemented on the Zybo, a development board with a Xilinx Zynq All Programmable System-on-Chip (Soc), which integrates a dual-core ARM Cortex-A9 processor with a Xilinx 7-series Field Programmable Gate Array (FPGA) logic. The completed design was able to receive pre-set desired temperature values from users and automatically control the heating, ventilating and air conditioning (HVAC) system to maintain the comfort zone and maximize energy saving. The smart thermostat was supplied by real-time embedded software running on the ARM microprocessor. All communications between peripherals and the smart thermostat were designed in Verilog and implemented on programmable logic.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Thermoregulation is an essential part of human survival needs; thus heating and cooling have always been two imperative factors in our everyday life. There are numerous technologies that have been developed to support these needs by using different sources of energy. For instance, in residential units, the majority of energy expenses is for heating and cooling. According to the U.S. Department of Energy, “Space heating is likely the largest energy expense in a house, accounting for about 45% of the average American family’s energy bill [1], while home cooling is 6% of the average household’s energy use [2].”

Moreover, according to the U.S. Energy Information Administration (EIA), Massachusetts households spent 22 percent more on energy than the U.S. average, paying about $2500/year [3]. The EIA reported, “Since the weather in Massachusetts and New England is cooler than other areas of the United States, space heating makes up a greater portion of energy use in homes (59%) compared to the U.S. average, and air conditioning makes up only 1% of energy use” [4].

In the current market, every available residential heating, ventilating and cooling (HVAC) system needs a thermostat to measure temperatures and control the starting and stopping of the HVAC. Therefore, the thermostat plays a crucial role in regulating indoor temperatures, which in turns determines the level of energy consumed by the HVAC. A traditional thermostat comes with a built-in thermometer and an interface to receive input temperature from users. It is usually mounted on a wall indoor, thus its measured temperatures are mostly temperatures of the surrounding area. If a measured temperature is not the same as the temperature set by the user, the thermostat will start the HVAC system and let it run until the measured temperature reaches the desired level.

A residential unit such as a family house, or an apartment, usually has only one HVAC system and one thermostat to control it. It is very common for such unit to be divided into multiple rooms, such as living room, bedroom, and kitchen. Depending on the construction of the unit, there is always a difference in temperature level across the rooms. The larger the unit is, the greater of such difference will be. However, it is impossible for the thermostat to accurately measure temperature of every room, leading to insufficient energy usage and creating discomfort for residents. People usually work around the issue by either offsetting the temperature set in the thermostat or they use a portable heating or cooling device. Both approaches do not solve the inherent issues surrounding the thermostat, and at the same time cause more energy consumption.

To resolve the issues, modern HVAC systems have built-in technologies to regulate temperature. A zoning system [citation] treats a room in a residential unit as a zone, which has its own thermostat and vent with a controller damper. Such systems allow regulated air to flow only to zones that are in need. However, such zoning system is hard to install into an existing HVAC system and is usually expensive. Moreover, since there might be multiple thermostats in multiple zones, the resident needs to frequently adjust temperatures across the thermostats to prevent the HVAC from operating inefficiently. Modern thermostats come with timers which allow users to maintain different temperature levels at different times. Even so, the user needs to put a lot of effort into the initial configuration setup and the efficiency of the system depends on the timer.

Recently, there are multiple thermostats developed to help users to have better control of indoor temperatures, as well as saving energy bills. They are easier to install than the zoning systems, and they come with interfaces using mobile applications users are able to control the thermostat without having to go to its location even when he or she is away. However, these thermostats do not fully solve the problem of temperature differences across rooms or allow users to set their desired temperatures. Without using a thermostat's timer, if a user wants to save on his energy bill, he or she still needs to change temperature levels when leaving and arriving home. Even with the help of a timer, the user does not always stay at his desired temperature level if he changes the routine set in the timer.

In this project, I propose a system design to help solve the aforementioned problem by using Zybo as a thermostat, along with a set of peripherals. The system includes sets of sensors, each is placed in a room of a home to detect the presence of a human and to measure the temperature of the room. Zybo would receive data from the sensors, analyze it and control the HVAC system accordingly. This report describes the system design in details. Chapter 2 introduces background knowledge of relevant technologies and terminologies, chapter 3 describes the detailed design of the entire thermostat's component. Chapter 4 explains how the design is implemented, and chapter 5 reports the testing procedures as well as the results.

2. Background

In this project, a Zync-7010 (Zybo) development board was used as a central controlling unit along with add-on components: thermal sensors for human detection, temperature sensors for temperature measurement, booster fans for enhanced air circulation, relays for HVAC control and an OLED screen for information display. This chapter introduces the concept and background of heating, ventilation and air conditioning (HVAC) system, a smart thermostat, comfort zone and add-on components listed above.

2.1 Heating, Ventilation and Air Conditioning (HVAC)

"The purpose of a heating, ventilating and air conditioning (HVAC) system is to provide and maintain a comfortable environment within a building for the occupants or a suitable environment for the process being conducted" [5]. The principal functions of an HVAC system were to provide desired cooling and heating outputs regardless of affecting factors. Besides, the system maintained comfortable conditions using as little energy as possible while providing a healthy environment for occupants and safe conditions for equipment [5].

An HVAC system often includes a cooling system to cool indoor air and a hot furnace to warm up the air pictured in Figure 2-1. Modern HVAC systems use a heat pump to supply both cool air and hot air because they have a special valve in the refrigeration piping that allows the refrigeration cycle operate in reverse [6]. According to an explanation of the U.S. Department of Energy, "In heating mode, liquid refrigerant in the outside coils extracts heat from the air and evaporates into a gas. The indoor coils release heat from the refrigerant as it condenses back into a liquid. A reversing valve, near the compressor, can change the direction of the refrigerant flow for cooling as well as for defrosting the outdoor coils in winter" [7].

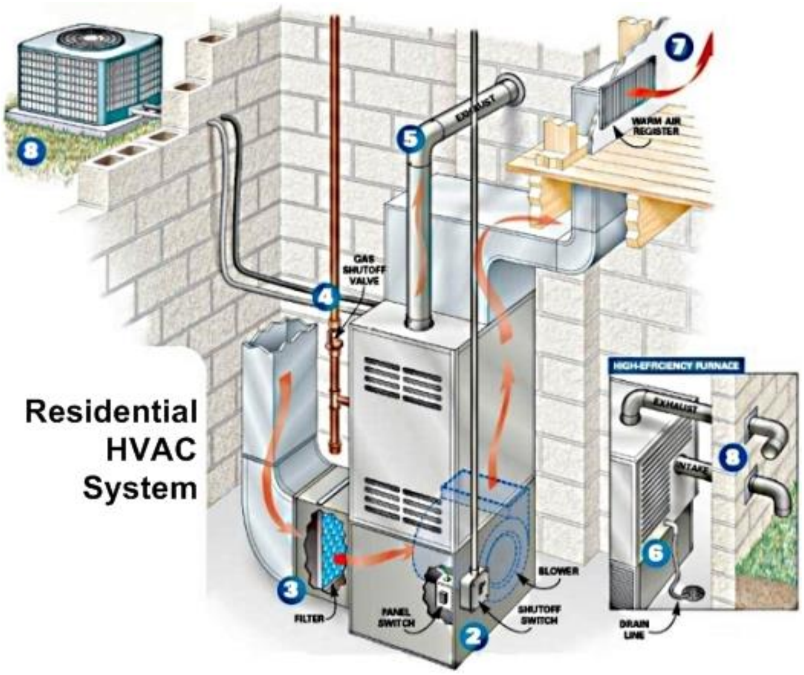


Figure 2-1 Residential HVAC System [8]

There are two types of HVAC systems: single-stage and multi-stage. A single stage HVAC system have five basic wires with preset standard colors that are red (R) for power, black (C) for common, yellow (Y) for cooling system, white (W) for heating system and green (G) for blower/fan [9]. In case the HVAC system has a second stage of cooling and heating, there is another set of wires to connect to the thermostat. An HVAC system has a 24VAC transformer to transform high-voltage to low-voltage source.

Along with the HVAC system, a thermostat is used as a controller. A basic thermostat allows user to select a desired temperature and switches between cooling or heating mode. In a forced-air HVAC system, the thermostat is connected to an air compressor, furnace and a blower via the five basic wires. When the thermostat calls for cooling, both the air compressor and the blower are activated simultaneously. When the thermostat calls for heating, only the furnace is activated. However, there is a model of thermostat that activates both the hot furnace and blower when calling. In such case, the thermostat has to call for either the heating or cooling system; it is not allowed to run both systems at the same time. Figure 2-2 shows a basic thermostat circuit and its connection to an existing HVAC system.

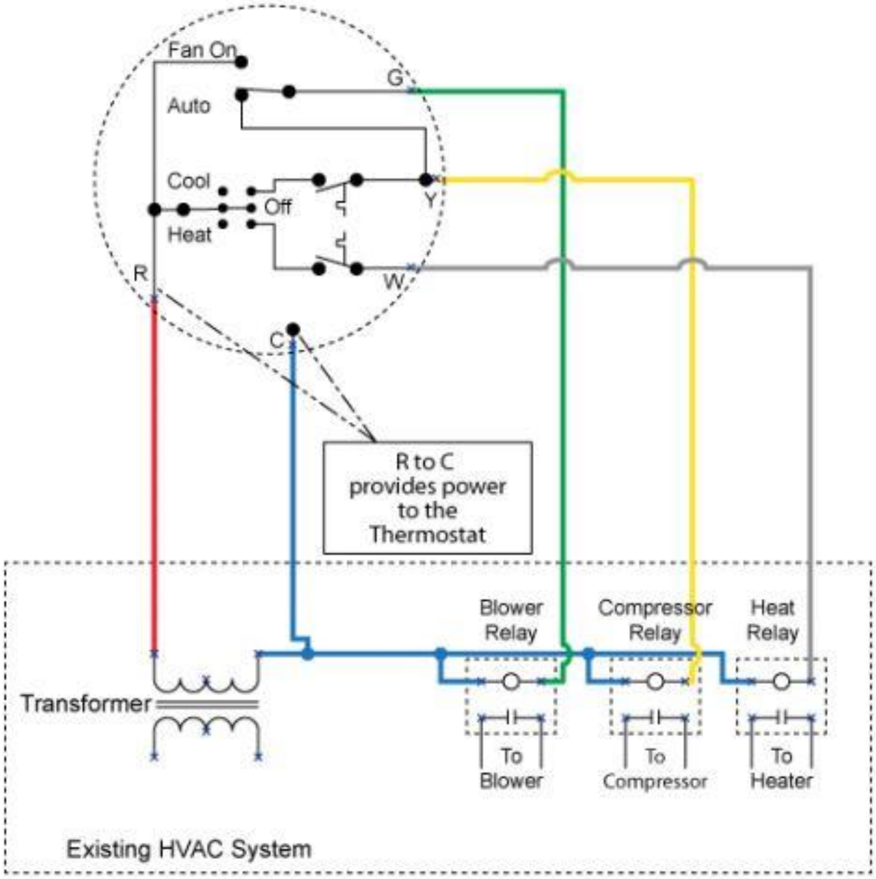


Figure 2-2 Basic Thermostat Circuit [9]

2.2 Smart thermostat

A smart thermostat is defined as a device that shows intelligence or good judgment in automatically adjusting a room's temperature to a desired level [10]. While a user has to adjust the desired temperature manually on a regular thermostat, a smart thermostat has programmable functions that can maintain the desired level automatically. Using a smart thermostat, the user does not have to worry about forgetting to turn off his or her HVAC system when leaving home nor feeling uncomfortable when coming back waiting for it to turn on. A smart thermostat saves its user from repeated manual tasks such as turning on/turning off, temperature setting or timing.

2.2.1 Characteristics of a smart thermostat

A smart thermostat can save energy by controlling an HVAC system efficiently with programmable functions and automatic operations. Such features save a user from repeating the task of turning the HVAC on and off while intelligently controlling the system to maintain a desired temperature level. In addition, there is also an energy-saving feature which a user can turn on when he or she is not at home for an extended period of time, such as going to work or on vacation. If an HVAC system has multiple cooling or heating stages, the smart thermostat is able to use these stages to improve humidity control and maintain the comfort zone by circulating just enough indoor air when necessary.

2.2.2 Smart thermostats on the market

There are many smart thermostats on the market developed with a variety of technologies, but almost all of them have common core features. Depending on its design, each smart thermostat has different temperature swings. With the swing, the HVAC runs to maintain indoor temperature at a desired level when the room drifts a given number of degrees away from the given range. The smart thermostat can also control a whole-house fan and auto-change heating/cooling mode to keep indoor air fresh and increase efficiency of the air conditioning cycles. It may also have a keypad lock to avoid accidental modifications from unexpected users.

Besides the common functionalities, designs of the "smart" features vary from one brand to another. Current state-of-the-art thermostats usually require users to manually control it for seven-day before setting a running schedule automatically. This feature creates a personal schedule based on the temperature changes and input date/time from users. By doing this, a thermostat know when users are away as well as when they are home, and adjust indoor temperatures accordingly. How the personal schedule is set up, and how the thermostat use the schedule to control the HVAC system is proprietary to the manufacturer. Moreover, some thermostats even have motion sensors to automatically turn on away/vacation mode. Nowadays, smart thermostats have Wi-Fi connectivity that let users control an HVAC system and adjust desired temperatures remotely. With that function, the users are able to pre-heat/pre-cool the house before they come home. Table 2-1 lists five popular smart thermostats currently on the market.

	Nest Learning Thermostat	Ecobee3	Honeywell Lyric	Radio Thermostat CT-80	Honeywell Wi-Fi Smart
Temperature Control					
Heating Stages	3	4	3	3	3
Cooling Stages	2	2	2	2	2
Temperature Swing	+/- 1^0F	+/- 1^0F	+/- 1^0F	+/- 0.5^0F	+/- 1^0F
Programmable Fan	Yes	Yes	Yes	Yes	Yes
Keypad Lock	Yes	Yes	Yes	Yes	Yes
Auto Changeover	Yes	Yes	Yes	Yes	Yes
Energy Management					
Seven-day scheduling	Yes	Yes	Yes	Yes	Yes
Away/Vacation Features	Yes	Yes	Yes	Yes	Yes
Auto-Schedule	Yes	Yes	Yes	Yes	Yes
Sensors					
Weather Conditions	Yes	Yes	Yes	No	Yes
Humidity Sensor	Yes	Yes	Yes	Yes	Yes
Motion Sensor	Yes	Yes	Yes	No	No
Design & Setup					
Wi-Fi	Yes	Yes	Yes	Yes	Yes

Table 2-1 Smart thermostats and their features on the current market [11]

2.2.3 Smart Thermostat Advantage

A 2007 Gas Networks study showed savings of 6.2% of total household annual natural gas consumption associated with the installation of an ENERGY STAR rated programmable thermostat [12]. A smart thermostat automates energy saving behaviors, such as lowering the temperature during work hours, when users are most likely not at home, which in turn helps deliver on that 6.2% actual saving.

In conclusion, a thermostat is marketed as "smart" if it provides at least one of the following two features: (1) Wi-Fi connection for remote access to the thermostats, even when users are not at home; (2) Automatic HVAC system control to maintain a preset temperature level without the need of repeating human attention. Almost all smart thermostats on the market have a Wi-Fi connection, while only the Nest, ecobee3, and Honeywell Lyric smart thermostats have built-in motion sensors to sense when no one is home and adjust indoor temperature accordingly.

2.3 Comfort zone

In this project, a comfort zone is defined as a temperature range in which humans feel comfortable. In a person's comfort zone, he or she should not feel uncomfortable by being cold nor hot. The American Society of Heating, Refrigerating and Air-Conditioning Engineers' (ASHRAE's) publication on "thermal environmental conditions for human occupancy" defined comfort zones for summer and winter season [13]. Assuming normal indoor clothing, it pointed out that a person's age, activity level, and physiology affected the ideal thermal comfort for that individual. "Air speed and thermal radiation are predominantly outdoor effects that are difficult to measure and control. As a result, literature on thermal comfort concentrates on temperature and humidity. Although temperature ranges are specified per season, the relative humidity is set between 70%RH and 30%RH in summer and winter time, respectively" [14]. Figure 2-3 below shows comfort zones of winter and summer season, where temperatures are between 73.0^{0}F and 76.0^{0}F, while humidity varies from 23.0% to 79.5%. The average comfort temperature is 74.5^{0}F and the average humidity is 51%. If the temperature is above 76^{0}F, the perceived air quality is worse regardless of the actual air quality. Similarly, high relative humidity might make users feel hotter and promote the growth of mold and mildew, while low relative humidity causes discomfort due to drying of the nose, throat, mucous membranes and skin.

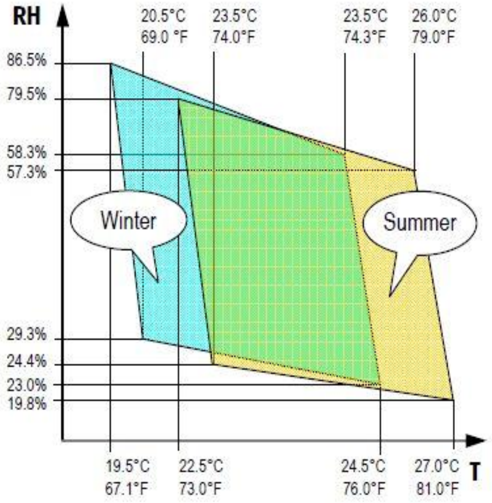


Figure 2-3 Relative humidity (RH)/temperature (T) diagram [14]

2.4 Zybo (Xilinx Zynq 7010) Development Board

Zybo (Xilinx Zynq 7010) is a development board produced by Xilinx. Zybo is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with a Xilinx 7-series Field Programmable Gate Array (FPGA) logic [15] pictured in Figure 2-4. This device features four binary slider switches, LEDs, push-buttons and a USB UART[1] connectivity. Additionally, Zybo has totally six Pmod connectors, available for communication with add-on sensors, OLED screen and relay.

[1] UART stands for Universal Asynchronous Receiver/Transmitter

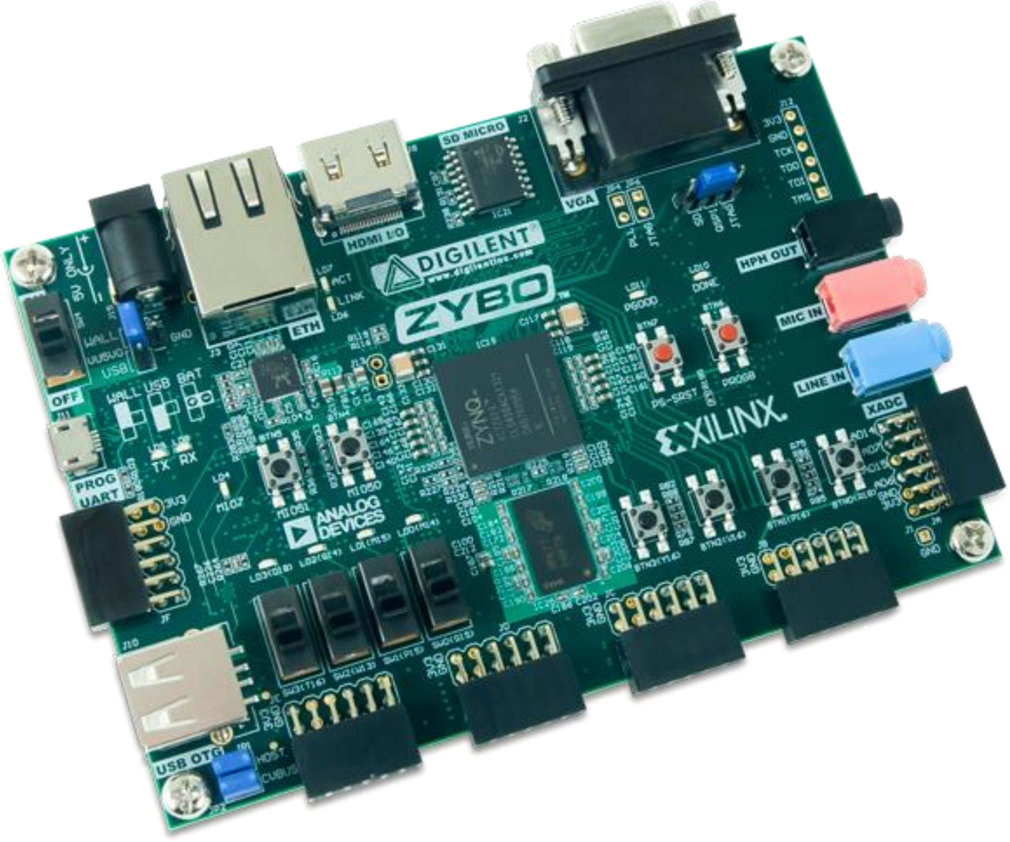


Figure 2-4 Zynq 7010 (Zybo) Development Board [16]

The 650MHz dual-core Cortex-A9 processor can be used as an embedded microprocessor, while the Artix-7 FPGA with 17,600 LUTs (Look-Up Tables[2]) and 35,200 flip-flops can be used to create complex digital logic [17]. On the Zynq board, the Cortex-A9 communicates with the FPGA using the Advanced eXtensible Interface (AXI) Interconnect. The clock speed for both Programmable Logic (PL) and Processing System (PS) is 100MHz, which is also used to run add-on peripherals. Figure 2-5 showes the overview of Zynq block design and its architecture, containing the processing system and programmable logic.

[2] Look-Up Table (LUT) implements truth table to define outputs for any given combinational logics of inputs

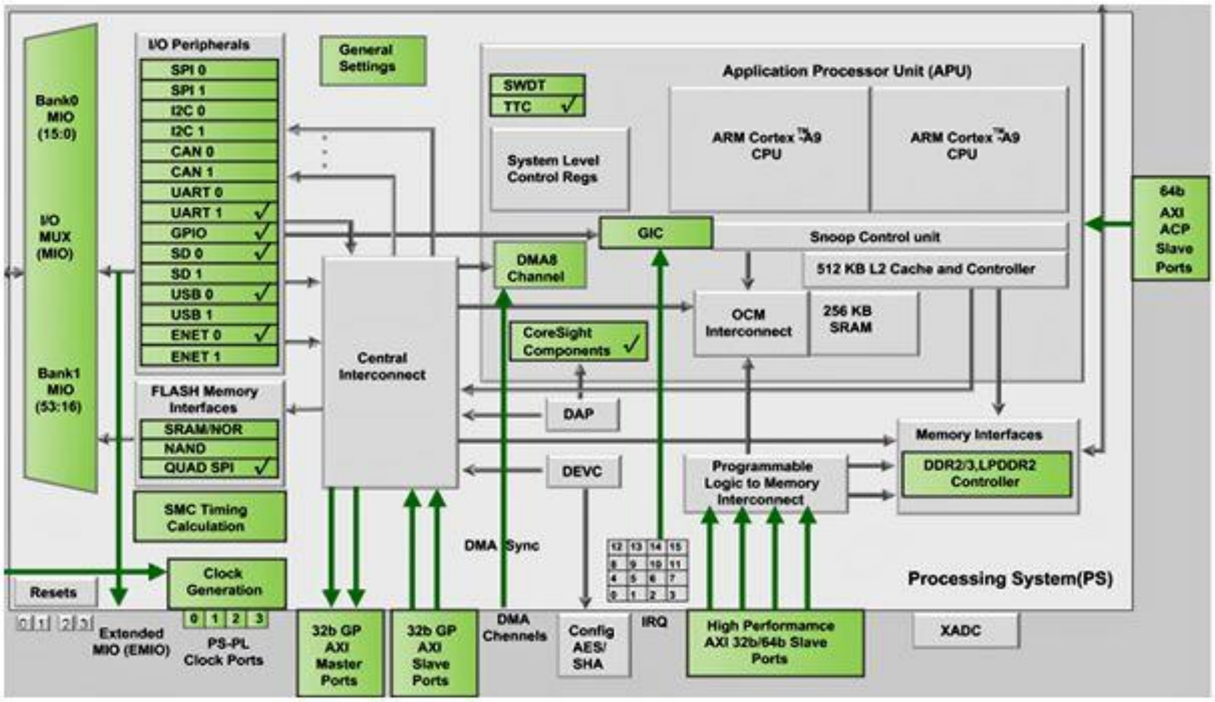


Figure 2-5 Overview of Zynq block design and its architecture [18]

I chose the Zybo as a central controlling unit because it could host a whole system design that connected hardware components and software programs together. The AXI Interconnect was convenient and flexible to transfer data and signals between Cortex-A9 and programmable logic. Additionally, the Zybo had enough Pmod connectors to communicate with sensors and to control the HVAC unit. All prebuilt sliders switches and push buttons acted like an interaction module of a thermostat.

2.5 Omron thermal sensor D6T series

This project used Omron D6T-44L thermal sensors pictured in Figure 2-6 with a 4x4 pixel resolution for human detection. “The product measures the surface temperature of the material by detecting intensity of the infrared radiation” [19]. Different from pyroelectric sensors which only detected human presence by change of signal, the Omron thermal sensor could catch the signal of a stationary person by continuously detecting the far-infrared ray of an object [19]. Figure 2-7 shows a difference between a pyroelectric sensor and an Omron thermal sensor.

Figure 2-6 Omron thermal sensor D6T-44L [19]

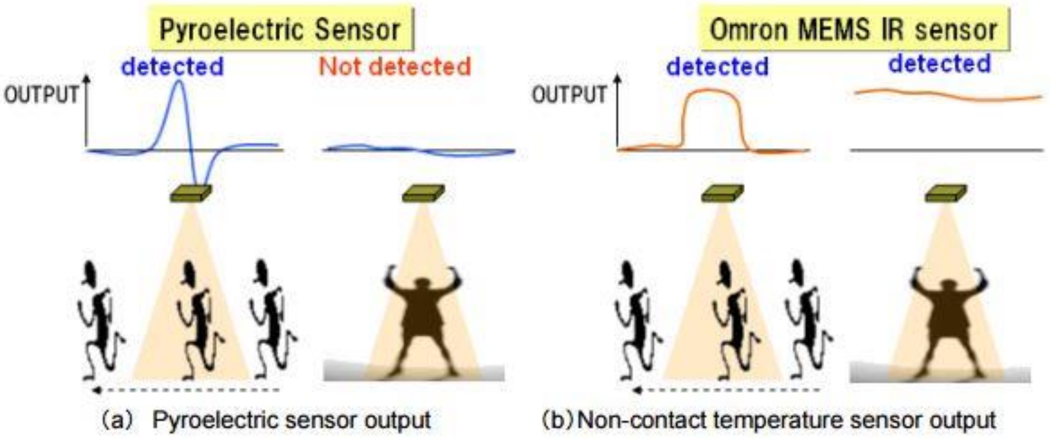


Figure 2-7 Difference between pyroelectric and non-contact temperature sensor [19]

This component performed its sensitivity characteristics over an object view angle by using a silicon lens. The Field Of View (FOV) was generally specified as an area of 50% for maximum sensitivity [19]. Figure 2-8 shows the Omron thermal sensor's FOV image and sensitivity.

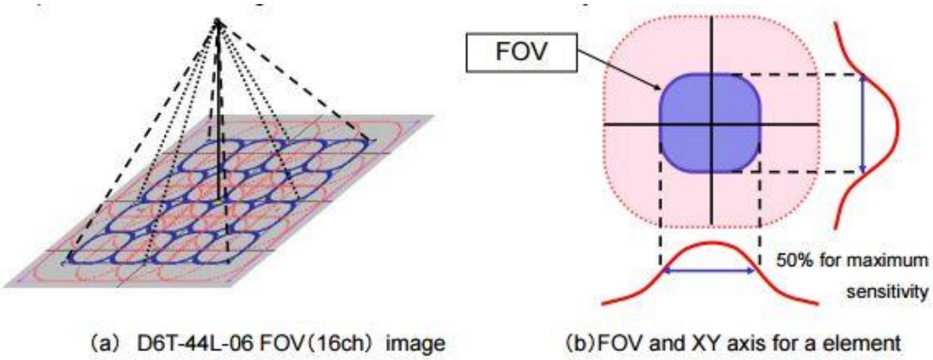

FOV
50% for maximum sensitivity
(a) D6T-44L-06 FOV(16ch) image
(b)FOV and XY axis for a element

Figure 2-8 Sensitivity characteristics: FOV image [19]

An Omron thermal sensor communicated with Zybo (master device) via an I^{2}C interface[3]. This module was driven by a 5V power source, its I^{2}C data and clock lines used the same voltage power. Since Zybo did not support 5V tolerant, an I^{2}C level translating IC was used to connect Zybo and the sensor. Usage of I^{2}C level translating IC would be discussed later in section 3.6.

2.6 Digilent temperature sensor

The Pmod TMP2 is a temperature sensor and thermostat control board built around the Analog Devices ADT7420 pictured in Figure 2-9.

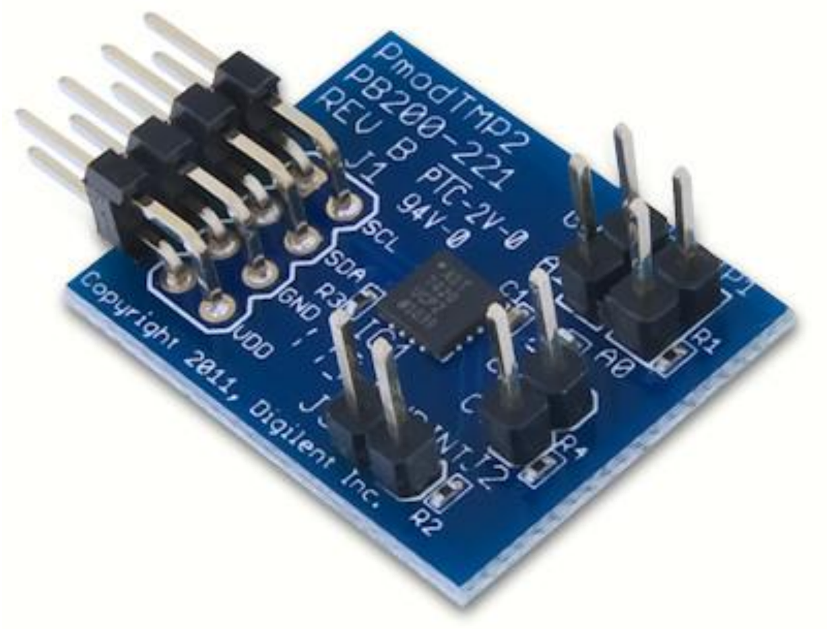


Figure 2-9 Digilent temperature sensor Pmod TMP2 [20]

This sensor uses an 8-pin connector that allows communication via I^{2}C. It also provides two 2-pin headers for the I^{2}C chip address selection, and two 2-pin headers for controlling external devices based on temperature thresholds [21]. The ADT7420 is a high accuracy digital temperature sensor. It contains a 16-bit ADC to monitor and digitize the temperature to 0.0078^{0}C resolution. By default, the ADC resolution is set to 13-bits (0.0625^{0}C). In this project, the temperature sensor was used with default 13-bit resolution only because it was sufficient to get indoor temperature values. This item has a typical accuracy of around than 0.25^{0}C and 240ms continuous conversion time.

[3] The I2C (Inter-IC) bus is a bi-directional two-wire serial bus that provides a communication link between integrated circuits (ICs). Philips introduced the I2C bus 20 years ago for mass-produced items such as televisions, VCRs, and audio equipment. Today, I2C is the de-factor solution for embedded applications. [61]

The Pmod TMP2 temperature sensor acted as a slave device using an I^2C interface. Zybo had to specify a slave address (0x4B)[4] to communicate with this sensor and a flag indicating the communication was read only. Using I^2C interface standard, this communication used two signal lines for I^2C data and I^2C clock. Based on the data sheet of the ADT7420 chip, these signals mapped to the serial data (SDA) and serial clock (SCL) respectively on the ADT 7420 [22].

2.7 Register booster fan

The purpose of a register booster fan was to increase air circulation from the HVAC system to a room when the distance between them was long. In the test environment section described in 5.3.1, each room in the apartment has one ceiling register to which three booster fans were attached. Each fan operated at 12VDC with a maximum power consumption of 2 Watts. The fan had the speed of approximately 3200 rounds per minute (RPM) and airflow of 41 Cubic feet per minute (CFM). A ceiling register with three booster fans might supply up to 123 CFM. In the thermostat design, the register booster fan of a room was turned on only if this room had positive human detection and its desired temperature needed to be maintained. Figure 2-10 below shows the back of a combined register booster fan. This was a register booster fan prototype with three attached 80mm fans powered by an external 12VDC power supply. The blue tape was used to cover an empty hole of the register. This combined register booster fan was attached to the ceiling duct register shown in Figure 2-11.

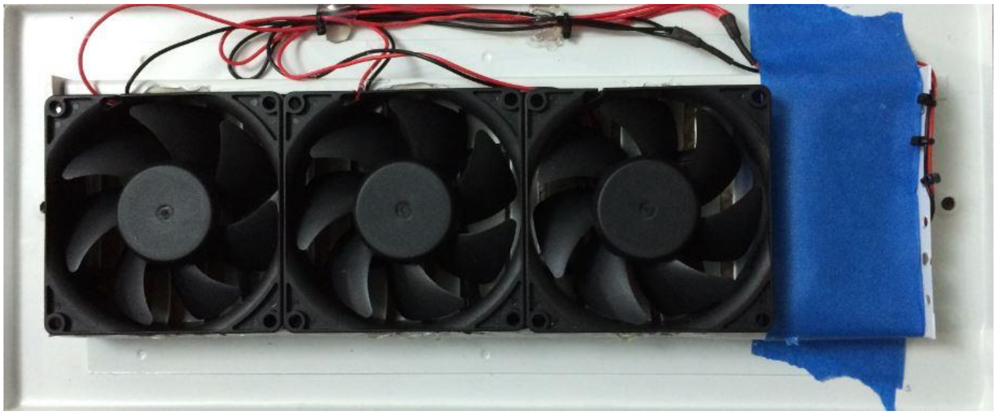


Figure 2-10 The Back side of a customized register booster fan.

[4] The Pmod TMP2 temperature sensor has four slave addresses from 0x48 – 0x4B. This project uses 0x4B for the communication between the master device and the Pmod TMP2 sensor.

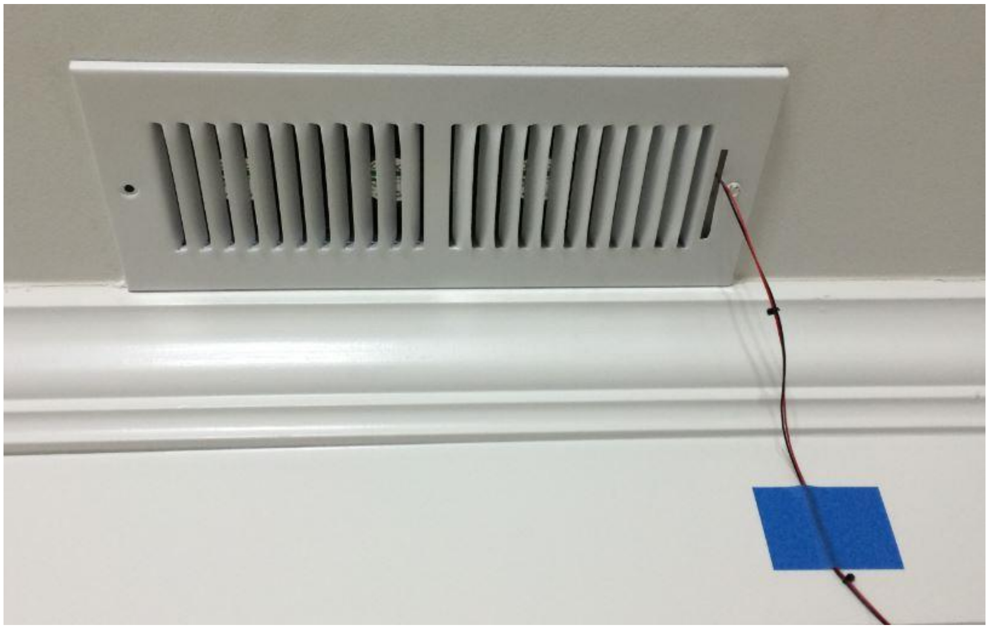


Figure 2-11 A register booster fan was attached to the duct register.

2.8 Digilent OLED screen

PmodOLED is a monochrome Organic LED graphic display produced by Digilent Inc pictured in Figure 2-10. This module uses a 128x32 OLED graphic display panel, measured 0.9" and was write-only (pictured in Figure 2-11).

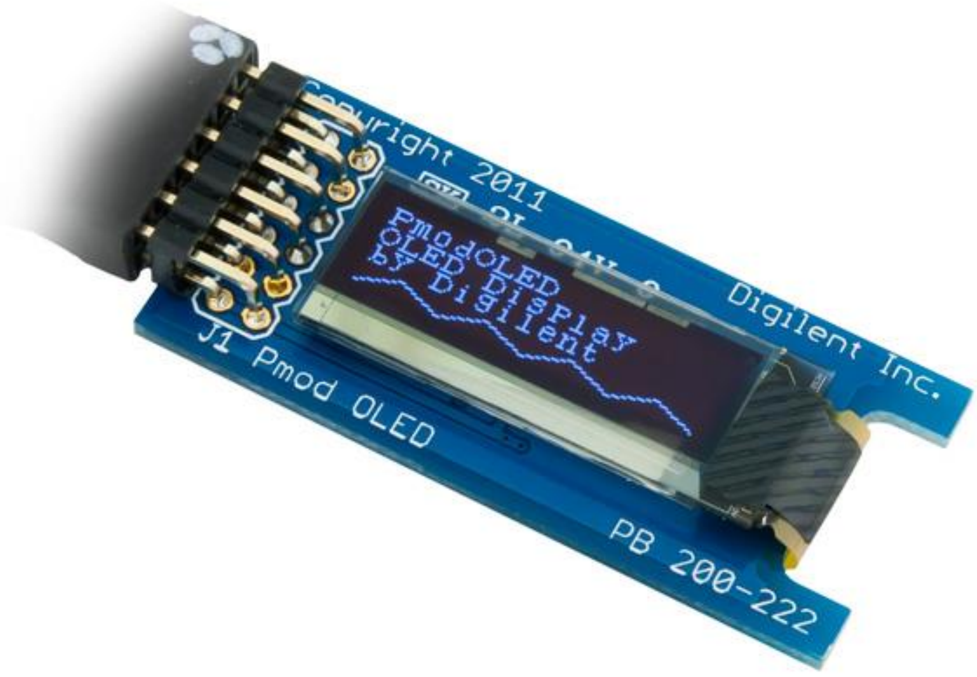


Figure 2-12 Digilent Pmod Organic LED Graphic Display [23]

In this project, the PmodOLED communicated with the Zybo via a standard SPI interface[5]. Zybo used this interface to configure the display, and then sent the bitmap data to the PmodOLED. The OLED graphic display panel could keep displaying the last image on the screen until it was turned off or cleared out [24]. This module was write-only, hence it only used Master Out Slave In (MOSI)[6] data method to draw bitmap data on the screen.

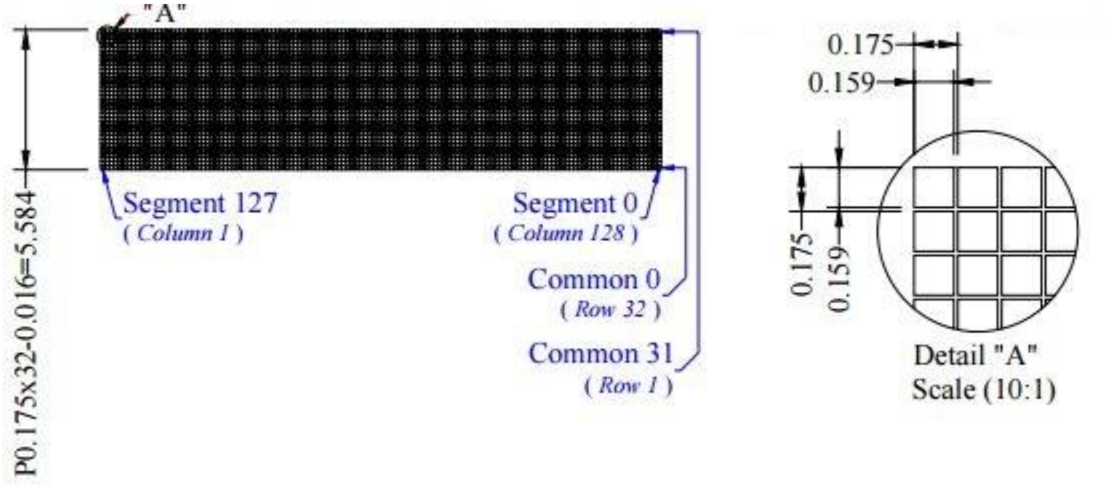


Figure 2-13 The OLED graphic display panel (UG2832) [25]

As mentioned, this OLED panel has 128x32 pixels that could be divided logically into 4 pages/ 4 rows. Each page can display up to 16 characters, where each character is represented by 8x8 bitmap showed in figure 2-12 [26]. The method used to display information to the screen will be explained in more details in section 3.7.

[5] Serial Peripheral Interface (SPI) interface is an interface bus commonly used to send data between microcontrollers and small peripherals. The SPI master controller uses separate four basic wires, clock, data and select lines to communicate with slave peripherals [62]

[6] Master Out Slave In (MOSI) signal is generated by a master device and is sent to a slave device as a recipient

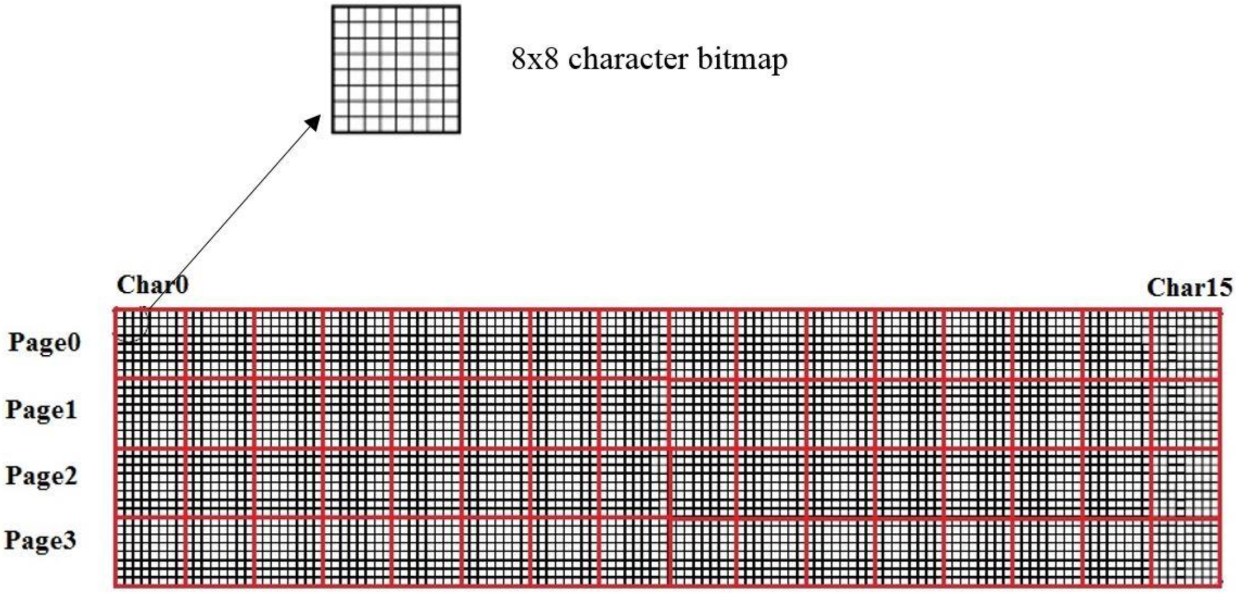


Figure 2-14 Logical division of the OLED display module [26]

2.9 Human Detection Algorithm

2.9.1 Visual cameras and thermal cameras

Visual cameras are a standard in general imaging purposes, serving a huge range of applications from personal and commercial to military. Modern visual cameras capture scenes and place them into colored images, thus the images' qualities depend on the scenes' illumination. If there is not enough lighting at a scene captured by a camera, the resulted image might appear darker than reality. In contrast, using the same camera configuration, if there is too much lighting, the resulted image might appear saturated. In both cases, actual color differences among objects in the scene are reduced, resulting in a lower image quality. Although visual cameras are becoming more affordable than before, this inherent limitation makes them less ideal when chosen for applications that need night vision.

Thermal cameras, on the other hand, do not require illumination. They are sensors that produce images based on object surface temperatures in the captured scene. Particularly, a thermal camera "captures the infrared radiation emitted by all objects with a temperature above absolute zero" [27]. Since there is no dependency on the scene's lighting condition, this type of sensors do not have the mentioned problem of visual cameras. However, they have their own limitations, which Fang et al. [28] summarize into three different types: First, it is impossible to distinguish between human and other heat sources based only on their brightness. Second, image intensities are not uniform across a human body, due to clothes, accessories or orientation, making detection tasks more difficult. Third, intensity ranges in most thermal images are smaller than those of comparable visual cameras, which lead to lower image quality.

Another important characteristic of a thermal camera is, it is generally much more expensive than a visual camera. It is because a special detector is required to capture thermal-infrared radiation [27]. The detector was first only used for the military before being commercialized [27]. Some of the thermal cameras available in the U.S. market are Omron D6T-44L, Panasonic AMG8832, Melexis MLX90260ESF, which were priced \$32, \$39 and estimated \$55 respectively. Among available options, Omron D6T-44L is the cheapest version, which makes it an ideal choice for a residential smart thermostat in term of price.

2.9.2 Human detection algorithms

Since thermal cameras are only available commercially recently [27], human detection research based on them is not as popular and plentiful as the one based on visual cameras. Most thermal image based research results are based on human shape templates [29] [30] [31] [32]. These methods are found common between thermal and visual images. They usually leverage different techniques to extract region of interests, such as the use of contour maps [33] [34], human features [33] [35]. The regions are then put through a classifier to identify whether the object inside is human or not. Classifiers used include but not limited to support vector machines [35], AdaBoost [36] [34] or Naive Bayers [37] [38].

Meanwhile, the only shape-independent method found were proposed by Fang et al. [28]. In this research, regions of interests were extracted using a strategy based on image intensity differences among pixels. The regions were then passed through a classifier to be compared against a generic template.

Existing research efforts inspired me to apply similar methods in detecting a human. However, the exact implementation was different, since the target environment of this project was indoor with a fixed setup, and the Omron sensor had unique characteristics that negatively affected detection results.

2.9.3 Human temperature

Human body is known to have a temperature of 98.6^{0}F/37^{0}C [39]. Research efforts also found out that human body temperature remains fairly constant regardless of changes in surrounding environments, including seasonal changes [40]. The average human skin temperature is also found to be 33^{0}C [41]. In multiple tested environments, the forehead and the back are parts that maintain the most stable and highest temperatures [41].

In spite of constant body temperature, skin temperatures actually change when the surrounding environment gets colder or hotter. Y.Liu et al. [42] found that when a room temperature changed from 25^{0}C to 32^{0}C, facial skin temperature changed from 34.5^{0}C to 35.5^{0}C. The research also showed that the average outer surface temperatures of clothes worn on human also increased from 30.5^{0}C to 33.5^{0}C when the same room temperature change happened.

The findings imply that, regardless of a room temperature, when a thermal image is captured, pixels containing human are usually brighter than other objects' pixels. Testing results of this project also confirmed the implication, made it a key factor in the human detection algorithm used in this project.

This chapter reviewed the concepts of an HVAC system, the characteristics of a smart thermostat and its benefits in controlling an HVAC unit. The concepts of comfort zone, a key factor in designing a smart thermostat, were also explained. This chapter also introduced all main hardware components and how they were connected together to be a smart thermostat.

The smart thermostat designed in this project took a different approach from others available on the market, by using a human detection algorithm to automatically identify human presence and control the HVAC system accordingly without repeated user manual input.

The next chapter describes the smart thermostat design, including temperature and thermal sensors, an OLED display and a relay module, system block diagrams, system implementation on Zybo and embedded software development. The HVAC controller and human detection algorithm were parts of the software written in C.

3. Project Design

This chapter describes the theory and methods of the overall smart controller design and how the peripherals were implemented in the programmable logic and in the processing system of the smart controller. The hardware, software designs and controlling algorithm are discussed to clarify the roles of each module and its application in this project.

3.1 System Block Diagram

There were three main components in the system: a peripheral unit, a programmable logic unit and a processing system. The system was designed for two rooms in a residential property. The system layout of this project is defined graphically by the diagram shown in Figure 3.1.

The peripheral unit consisted of two sets of sensors, each included a thermal sensor and a temperature sensor to detect human presence and measure room temperature in a room. The unit also had push buttons on Zybo to receive configuration inputs such as mode selection, temperature choice, etc.

The programmable logic unit was responsible for receiving, processing and passing data from peripherals to the processing system. In the unit, a thermal data processing module read thermal data from two rooms, stored them in an input buffer before sending to the processing system. Similarly, a temperature data processing module read temperature data from two rooms via I^{2}C interface, saved to it another input buffer, and then sent them to the processing system. Additionally, a user could set configuration data through push buttons. Signals from the buttons were also sent through the corresponding module toward the processing system. All input data was sent and received in real time.

In the processing system, the human detection algorithm used thermal data to detect human presence and sent result to the HVAC controlling algorithm. The controlling algorithm used it in combination with user configuration data to control the HVAC automatically. In order to turn the HVAC on and off, the microprocessor sent a low voltage signal to a relay module in the programmable logic. The controlling algorithm also sent output information to an OLED display through the programmable logic.

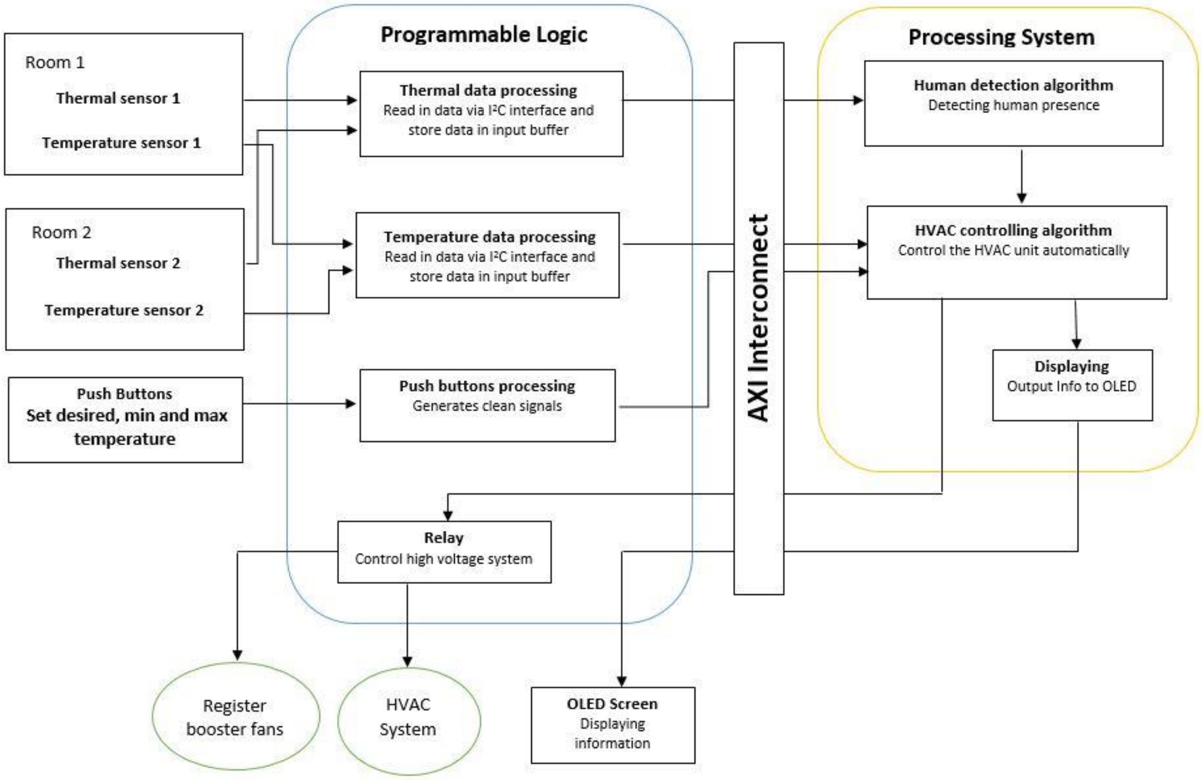


Figure 3-1 Overall system block diagram

3.2 Zynq SoC and Architecture

As mentioned, the general architecture of Zynq comprised both the Processing System (PS) and the Programmable Logic (PL). The Zynq SoC offered substantial flexibility to implement such a design mentioned in 3.1, as it allowed access to a microcontroller and programmable logic on one chip. Parallel high-speed logic signals received from sensors were processed by the FPGA of the PL, while human detection, HVAC control, and OLED display tasks were completed by the microprocessor of the PS. Figure 3.2 shows the primary communication interface of the Programmable Logic and Processing System via an AXI Peripheral Interconnect.

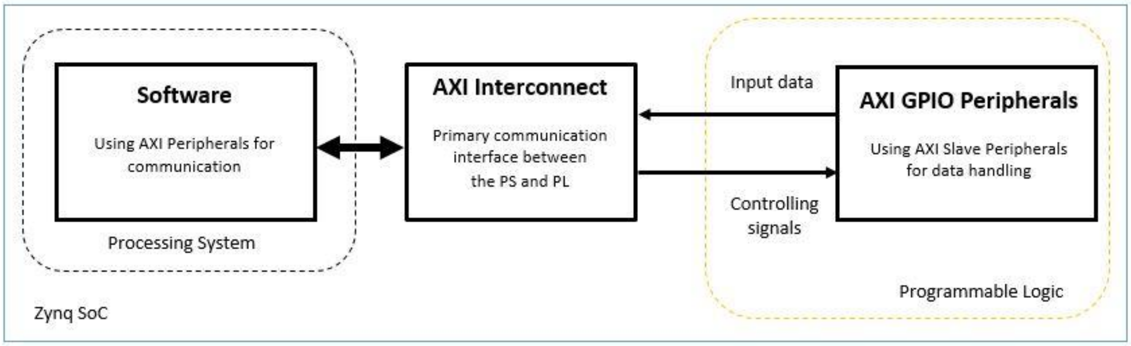

Software
Using AXI Peripherals for communication
Processing System
AXI Interconnect
Primary communication interface between the PS and PL
Input data
Controlling signals
AXI GPIO Peripherals
Using AXI Slave Peripherals for data handling
Programmable Logic
Zynq SoC

Figure 3-2 Zynq SoC interface

3.3 AXI Peripheral Interconnect

The Advanced eXtensible Interface (AXI) interconnect was introduced as a communication interface between the programmable logic component and the processing system powered by an ARM Cortex-A9 hard processor. Particularly, in this project, an AXI Interconnect module was implemented to receive input data from temperature sensors, thermal sensors and other peripherals and send output data to the processing system. Additionally, the AXI module was used to get data back from the software and to send it to the output of the programmable logic to the OLED display, HVAC controller relay and booster fan controller relay. The AXI Interconnect utilized transmission data lines in parallel at 100MHz bus. This communication was done through the use of AXI GPIO Peripheral modules, where each module used 32-bit channels for communication.

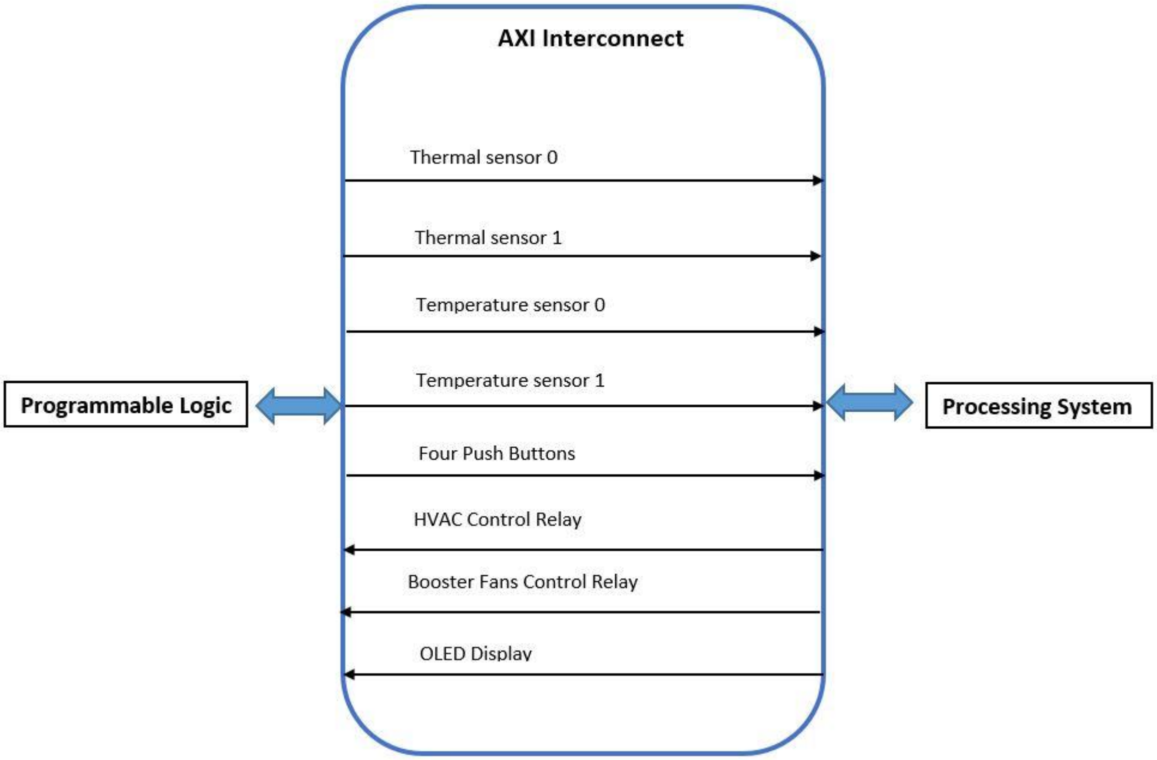


Figure 3-3 AXI Interconnect

3.4 The processing system on ARM Cortex-A9 processor.

The Processing System (PS) played several important roles in controlling the smart thermostat system. It was a C program run on the ARM Cortex-A9. First, the program had an interface connecting itself to the PL. Second, it stored data received every second from the PL in memory, for later use by the algorithms. Third, it had two fundamental algorithms to run the thermostat: The human detection algorithm which used thermal data to detect human presence; the HVAC controlling algorithm which used human detection result, temperature data, and configuration data to issue control commands for the HVAC and display data for the OLED. Last, the program sent the commands back to the PS, before such it was converted into signals and sent to the HVAC unit as well as the display.

3.5 IP Module Generation for FPGA Processing

Each component in this project was designed as an independent module using the Verilog HDL. Xilinx Vivado Design Suite was also used to create a top-level hardware design of the system and seven components as standalone projects called Xilinx IP blocks. The top-level design allowed integrating IP blocks easily using a GUI of a system block diagram. Since all modules were independent, they could be interchanged, revised and tested without affecting other modules.

3.6 Temperature Sensor and Data Transmission to FPGA

The Digilent PmodTMP2 temperature sensor used a standard I^{2}C interface that provided two I^{2}C signals, serial data (SDA) and serial clock (SCL). The I^{2}C interface needed pull-up resistors to ensure that the SDA and SCL wires were pulled to a high logical level in the absence of a driving signal.

A Pmod connector was able to drive bus signals on cables up to 18" in length [43], while almost all the sensors in this project were located far away from Zybo (longer than 18"). Since sensors used in the project were mounted far away from Zybo, I used a P82B715 I^{2}C bus extender to drive I^{2}C bus signals on long cables. The P82B715 was a bipolar IC intended for application in I^{2}C bus and derivative bus systems [44]. This component kept all operating modes and features of the I^{2}C bus, while offered extension of I^{2}C bus signals across a long distance between components by buffering the data and clock lines [44]. Besides, the I^{2}C bus extender supported up to approximately 50 meters distance or 3000 pF [44] and it operated at the frequency from 100 kHz – 400 kHz with supply voltage from 3V to 12V. These specifications was suitable to drive the I^{2}C bus signal for the temperature sensor.

Based on the instruction of the I^{2}C bus extender [45], when the I^{2}C bus operated at the frequency 100 kHz, the pull-up resistor for each I^{2}C bus was 3 kΩ as shown in Figure 3-4.

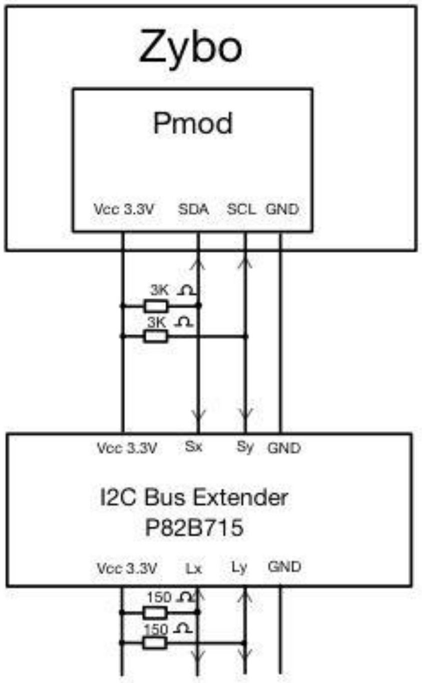


Figure 3-4 Pull-up resistors for each I^{2}C bus

Figure 3-5 shows the interface between Zybo and the I^{2}C bus extender P82B715 and the pull-up resistors.

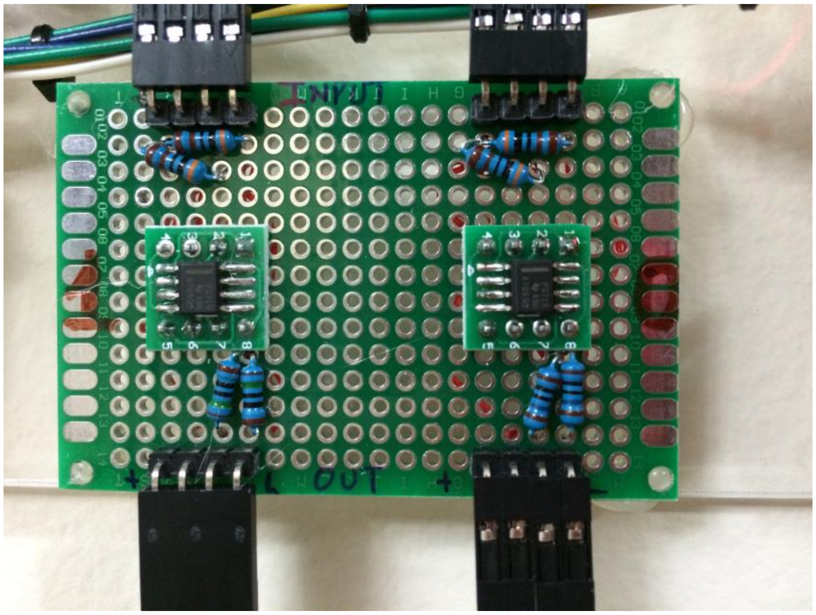


Figure 3-5 I^{2}C bus extender P82B715 and pull-up resistors

Additionally, the net pull-up resistors on the cable bus could be smaller than 235Ω. In this project, the net pull-up resistors were 150Ω for the I^{2}C bus and were placed on each side of the I^{2}C extender module. Figure 3-6 shows totally four pull-up resistors set on each side of the bus extender.

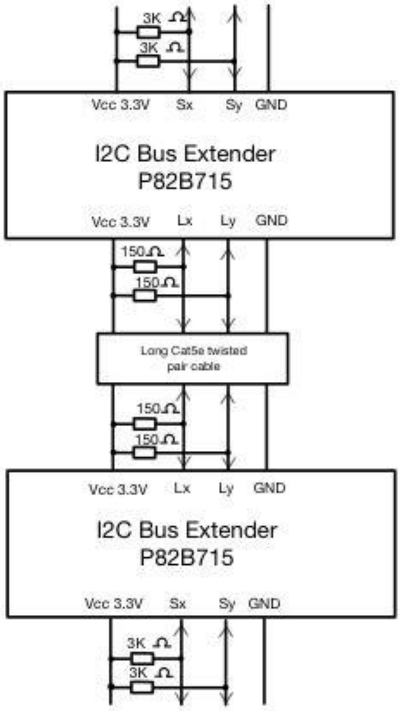


Figure 3-6 Net pull-up resistors on the cable bus

Finally, the I^{2}C bus between the bus extender and temperature sensor was 3 k**Ω** as shown in Figure 3-7. Also, Figure 3-8 shows the real Digilent PMOD TMP2 temperature sensor connected to an I^{2}C bus extender with pull-up resistors.

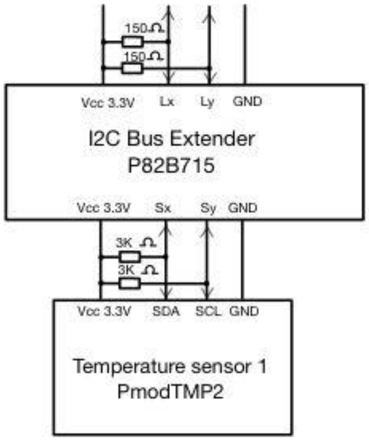


Figure 3-7 Pull-up resistors between a temperature sensor and the bus extender

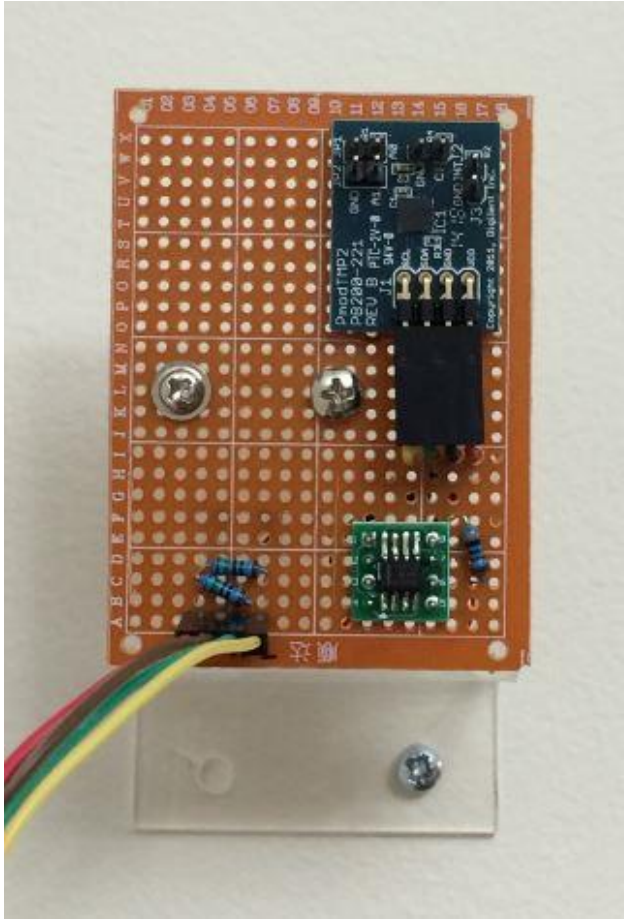


Figure 3-8 The interface between a temperature sensor and an I^{2}C bus extender

This project used two Digilent Pmod temperature sensors, each located in a room of the residential property. Both temperature sensor interface circuits were built the same way and used similar pull-up resistors. Since the net pull-up resistors worked for a cable up to 20 meters long, the temperature sensor interface circuits worked fine with a Cat5e twisted pair cable with the average length of about 4 meters. Figure 3-9 shows the overall temperature sensor interface circuits connected to Zybo.

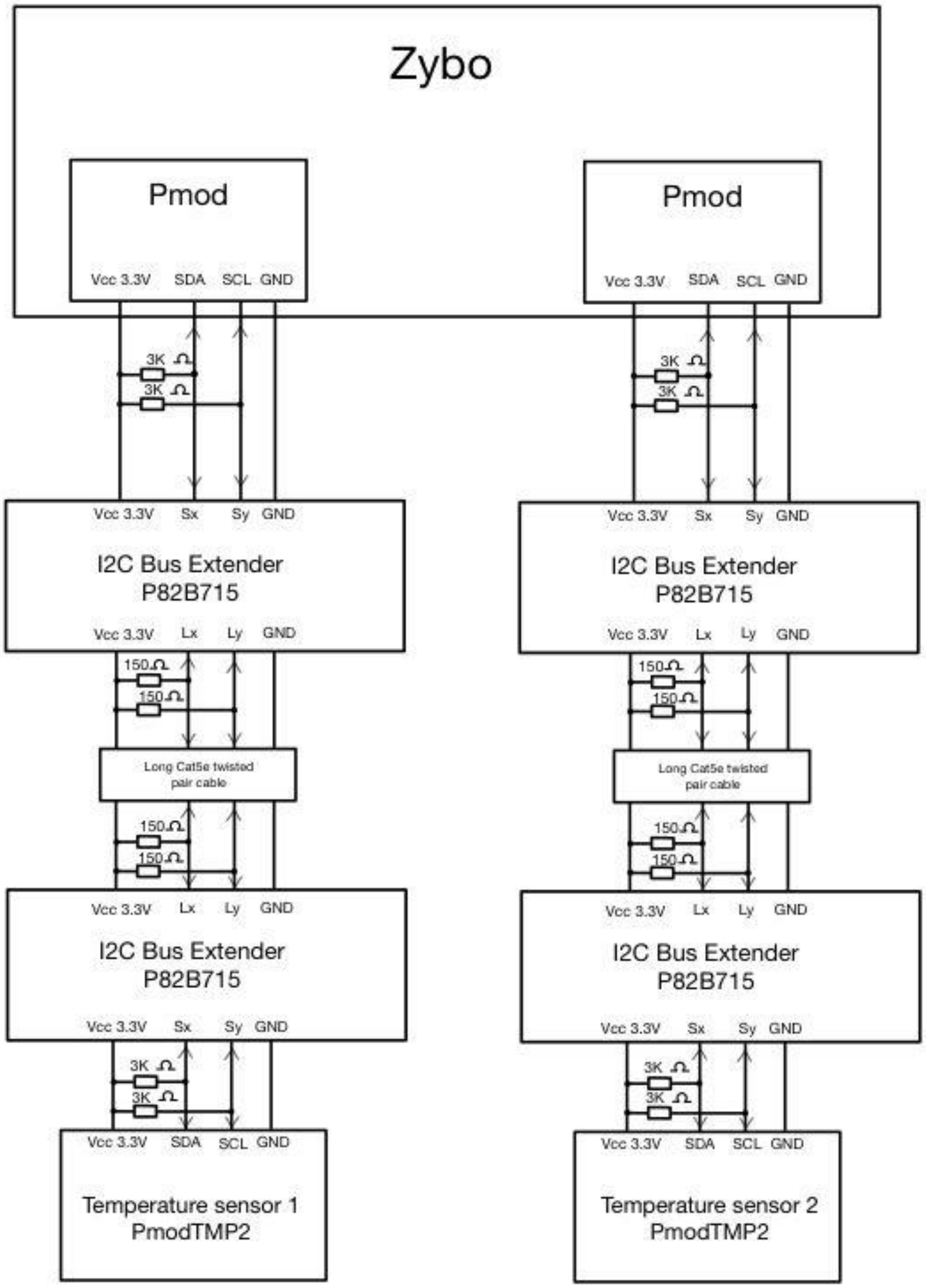


Figure 3-9 Overall temperature sensor interface circuits connected to Zybo

A temperature sensor module was generated on the programmable logic as a slave AXI-peripheral. It was used to acquire temperature data from the sensor and to allow the processor system access to the data through memory-mapped registers. This module had one 32-bit access register used to hold 13-bit data from temperature sensor. This register was controlled by the software in the processing system.

The temperature sensor module was a hierarchical design that delegated different functionalities to different blocks. Figure 3-10 showed the top level of the temperature sensor. The top-level wrapper module connected to the Zynq processing system and used the 100 MHz clock supplied by the processing system. The temp_sensor_v1_0_S00_AXI interconnect connected the programmable logic and the processing system. This module stored the measured temperature value in a 32-bit access register, and output the data to the processing system. Additionally, the AXI module could receive data from the processing system and output bus signals to the peripherals. The temp_ctrl module at the innermost level was set to receive measured temperature data (in this project) from the peripherals through the Pmod connection. The temp_ctrl module generated a slower serial clock 100 kHz for I^2C bus from the clock 100 MHz of the processing system. Since the top level temperature sensor module was generated as an IP module, it could be reused for the second temperature sensor in the second room.

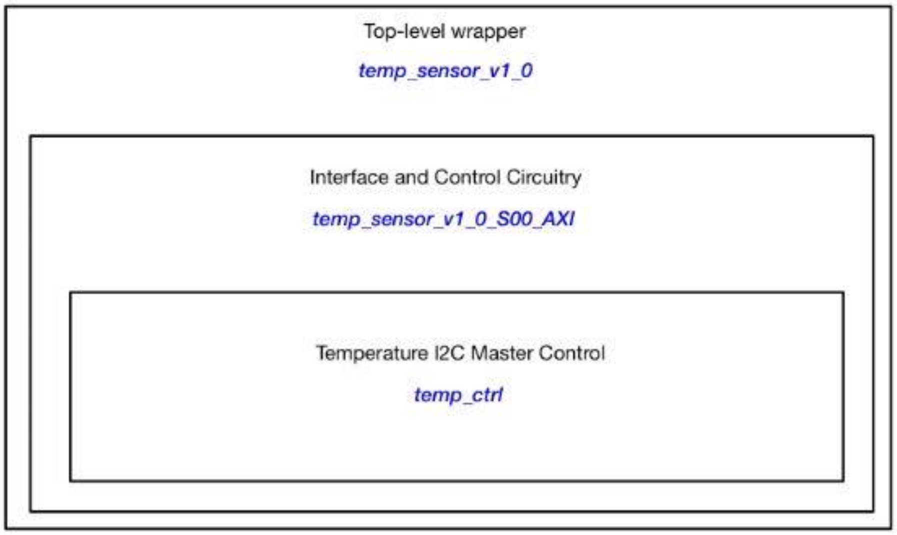


Figure 3-10 Top level of a temperature sensor on programmable logic

In the programmable logic unit, Zybo, as a master device, had to specify 7-bit address 0x4B to access the sensor. Once addressed, Zybo could issue commands, such as writing to or reading from the slave's registers on the temperature sensor. The logical procedure used for writing address to the sensor and reading measured data was implemented as finite state machine (FSM) to fit measured time intervals. Figure 3-11 shows a data read-back procedure from the temperature value's most significant byte and least significant byte register.

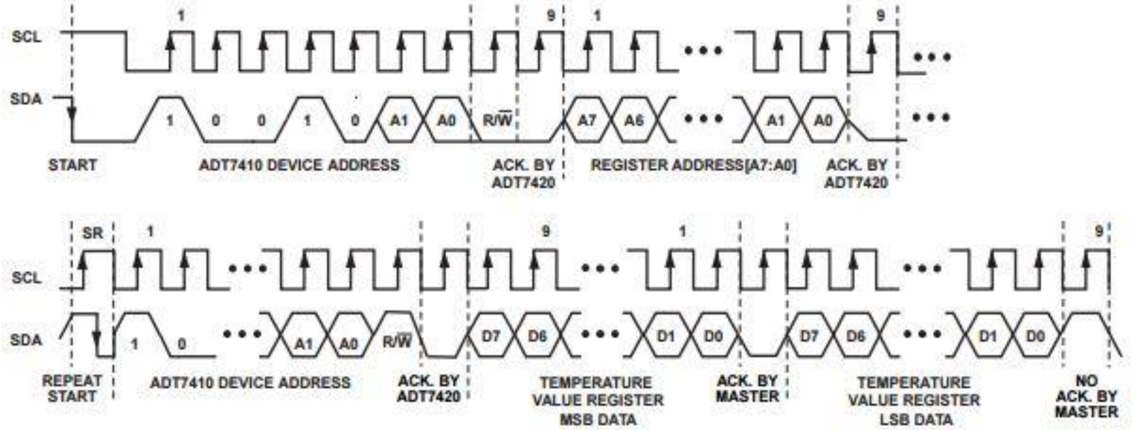


Figure 3-11 Reading back data from the temperature value MSB and LSB register [46]

The temperature sensor module was built on the programmable logic unit based on an open source code "I2C simple master for typical 7-bit EEPROM" [47]. This module was customized to fit I^{2}C timing specifications and conditions for writing and reading data. The communication between Zybo and the temperature sensor was implemented as a finite state machine in the following order:

- In pre_start_up state, the master device waited for debounced SDA input signal to go high while clocking SCL as necessary. The pre_start_up and start_up state only initialized the temperature sensor module at the beginning shown in step 1 and 2 of Figure 3-12. After that, the master device transitioned to the main states to get measured data.
- Zybo checked the status of the temperature sensor in idle_state, if the sensor was not busy, Zybo would transitioned to start_state showed in step 3 of Figure 3-12. At this state, when SDA input signal was driven low, Zybo assigned a control frame that addressed the ADT7420 device address at 0x4B to access the sensor. A write bit (R/W bit was set to zero) was added accordingly that indicated Zybo would write to the slave device next and the address of the most significant byte (MSB) register within the ADT7420 that was going to be read from. And then, the master device transitioned to spin_state (step 4) to wait for an I^{2}C timing specification before jumping to the clock_low state (step 5). The current state sent timers based on I^{2}C timing specification from the data sheet of ADT7420 to spin_state for counting time period and then returning back to the next state.
- In clock_low, the SCL was asseted to low and wait for t_hold (step 6) before changing the SDA signal. When the counter completed, the master device moved to the shift_data state (step 7) to shift each bit of the control frame out to the temeprature sensor. When the transaction finished, this state had to wait for the timing specification (step 8) before transitioning to clock_high state.
- The clock_high state released low drive on SCL and when the SCL input signal went high, the master device would sampled SDA signals and moved on to next state that showned in step 9, 10 and 11 of Figure 3-12. Also, in this state, a bit_count flag and write_cycle flag

were used to keep track when the temeprature sensor was in write cycle and when it was in read cycle. If it was in the write cycle, the master device would come back stop_state and return to start_state to start sending a control frame that addressed the ADT7420 device and a read bit (R/W bit was a one) before reading data. If the temperature sensor was in read cycle, the master device would received the 13-bits measured data from temprature sensor and moved on to the stop_state. After reading the temperature value from the MSB register, the address pointer of the ADT7420 automatically increased to the least significant byte (LSB) register to read the rest of temperature value

- In stop_state, if the write_cycle was one, the master device had to return to start_state to start a new cycle with write_cycle was zero as described above. If the write_cycle already was zero, Zybo would output 13-bits data to the temp_sensor_v1_0_S00_AXI module and store in a 32-bits accessed register.

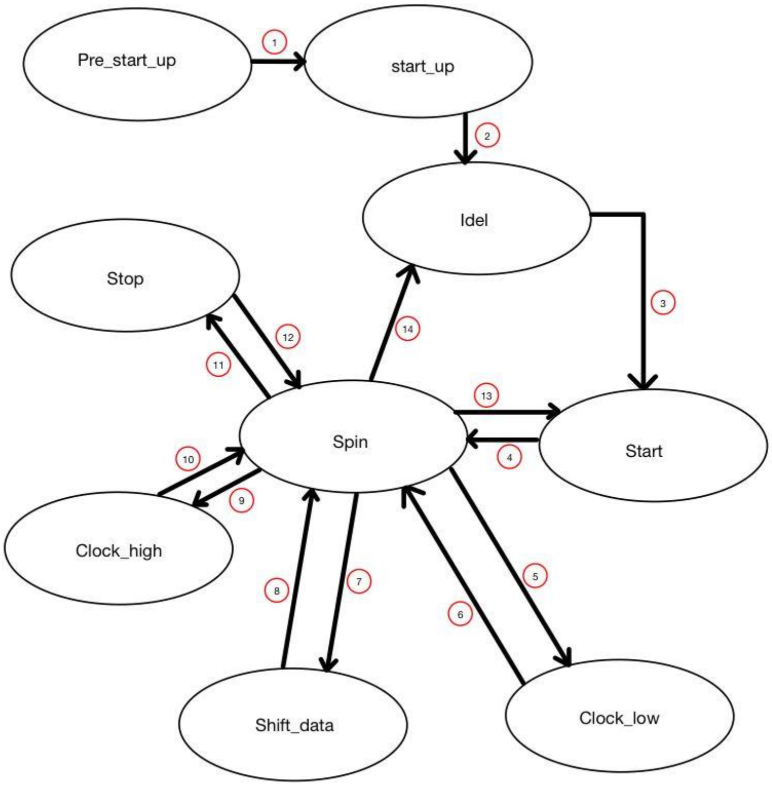


Figure 3-12 Finite state machine of temperature sensor module

The temp_sensor_v1_0_S00_AXI module used a 32-bit access register to store measured data into 13 LSBs of the register. Other bits within 32-bits were set to a zero. This assigned value was convenient when the software used a pointer to get value from this 32-bit register. These binary values were converted to Celsius degree by multiplying with 0.0625, a 13-bit temperature resolution. The measured temperature was updated every second in the software.

3.7 Thermal Sensor and Thermal Data Transmission to FPGA

The Omron D6T-44L-06 thermal sensor also output measured value through an I^{2}C bus. The thermal sensor used power source at 5V for both SDA and SCL buses, while the maximum output voltage of the Zybo was 3.3V. Therefore, a PCA9517 level-translating I^{2}C was added to provide bidirectional voltage level translation between low voltages (0.9V – 5.5V) and high voltages (2.7V – 5.5V) in mixed-mode application [48].

Similar to the case of the temperature sensor, since the Pmod could not drive bus signal on long cables, the thermal sensor interface circuit need a pair of I^{2}C bus extender to drive boost I^{2}C bus on the cable. Per the standard I^{2}C system, pullup resistors were required to provide the logic-high levels on the buffered bus. Based on the instruction of quick design multi-point circuit from NXP semiconductor [44], this design was applied to build the thermal sensor module interface.

A 4.7 kΩ pull-up resistor was chosen for the connection between the master device, the level translating I^{2}C-bus repeater PCA9517 and the I^{2}C bus extender when the thermal sensor operated at 100 kHz. Figure 3-13 shows how the pull-up resistors were attached to the interface circuit. Note that the level translating I^{2}C bus repeater had an active high enable (EN) output to allow Zybo to select when the repeater was active. One side of the bus repeater was connected to Zybo with voltage power at 3.3V, while the other side used voltage powered at 5V coming from a 5V external power source. The rest of the components could operate at 5V, so the external power source could drive 5V power signals over a long cable to the thermal sensor. Figure 3-13 shows how the pull-up resistors were connected to I^{2}C buses between Zybo, the bus repeater and the bus extender.

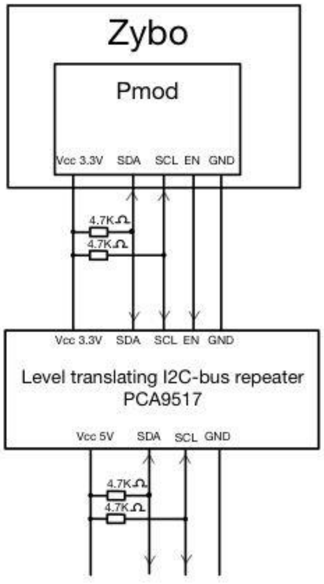


Figure 3-13 Pull-up resistors for the connection between Zybo, I^{2}C bus repeater and bus extender

Two pairs of 470Ω pull-up resistors were chosen for the connection between two I^{2}C bus extender. The 470Ω pull-up resistors set a cable bus limit at 5000pF [49]. Figure 3-14 showed net pull-up resistors on a long cable with the power voltage at 5V. Besides, Figure 3-15 indicated the 5V power connector from an external power source to the circuit. The I^{2}C bus extenders and two pairs of 470Ω resistors were placed in the front of the prototype board, while the I^{2}C bus repeater with 4.7KΩ pull-up resistors were placed in the back.

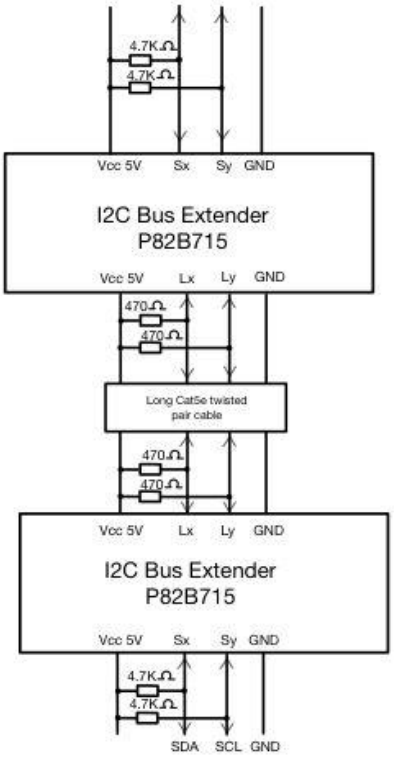


Figure 3-14 Pull-up resistors on a cable bus.

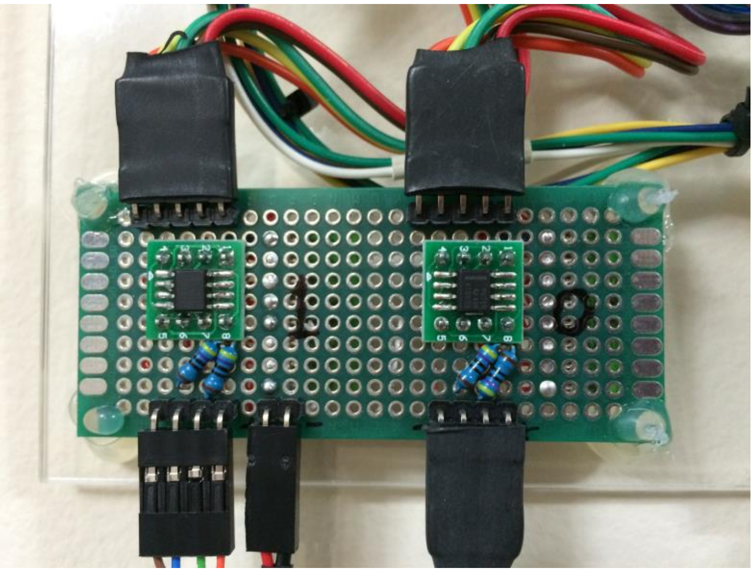


Figure 3-15 Thermal sensor interface circuit with the I^{2}C bus repeater, bus extender and pull-up resistors

A pair of 4.7 kΩ pull-up resistors were connected between the bus extender and the Omron thermal sensor. As a result, the longest cable used to connect Zybo over the I^{2}C bus repeater and the bus extender was 15 meters. The interface circuit worked well without losing the connection. Figure 3-16 shows the connection between the bus extender and the thermal sensor with two pull-up resistors for I^{2}C bus. Figure 3-16 also demonstrates the real thermal sensor connection with an I^{2}C bus extender.

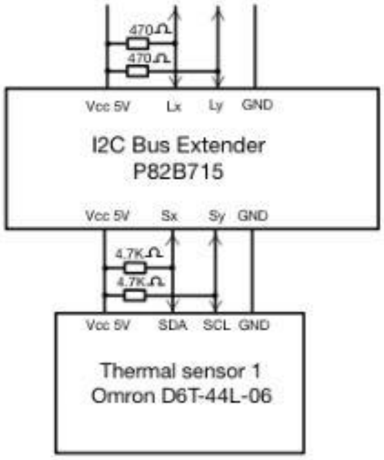


Figure 3-16 Pull-up resistors between the I^{2}C bus extender and the thermal sensor.

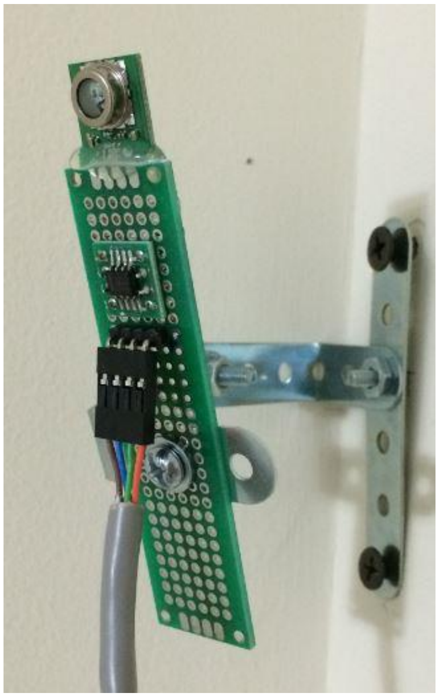


Figure 3-17 Thermal sensor with the I^2C bus extender

Another similar interface circuit was generated for another thermal sensor located in the second room. Figure 3-18 indicates the overall connection between Zybo and two thermal sensors.

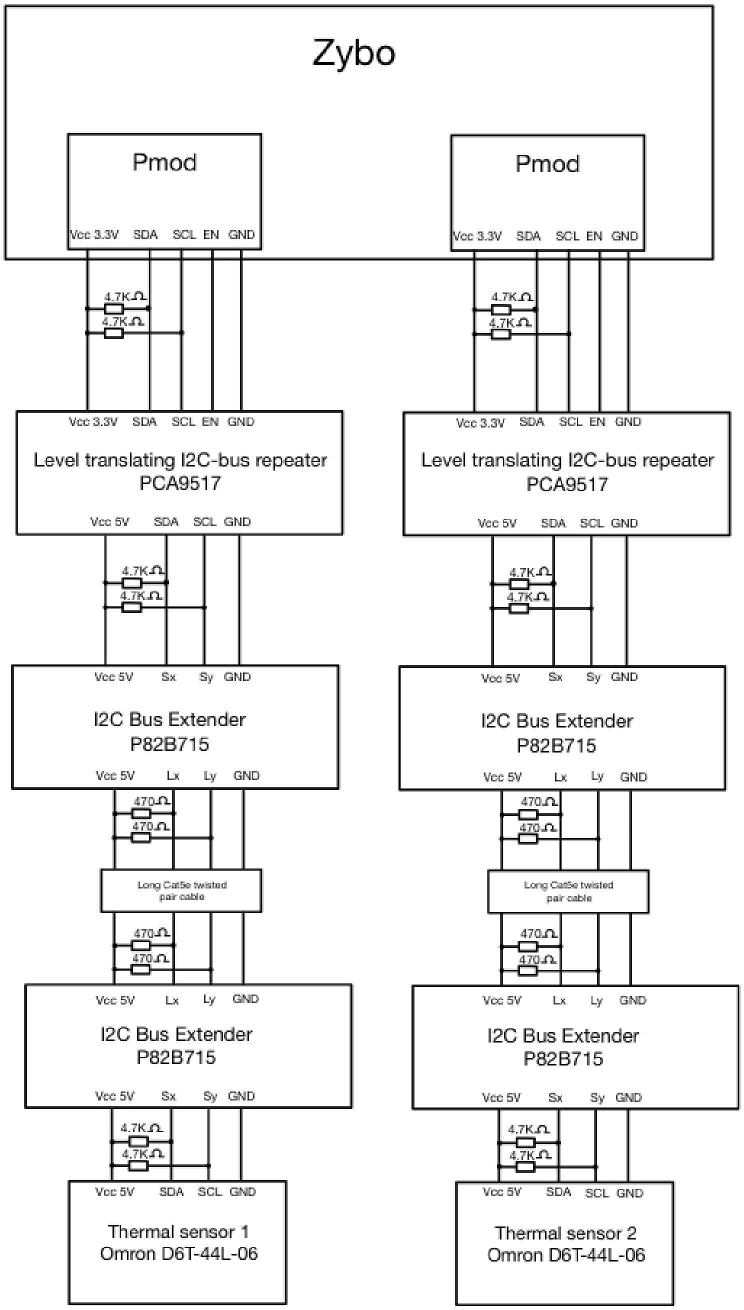


Figure 3-18 Overall thermal sensor interface circuit

In the programmable logic unit, the thermal sensor module was generated based on the Verilog code of the temperature sensor. The hierarchy of the thermal sensor module was similar to the temperature sensor module. It had a top-level wrapper contained a thermal_sensor_v1_0_S00_AXI interconnect and a thermal controller module. The thermal controller module received measured data from the thermal sensor. The AXI module would store data from the controller into nine 32-bit access registers and send them to the software on processing system component. Figure 3-19 shows the top-level of the thermal sensor module.

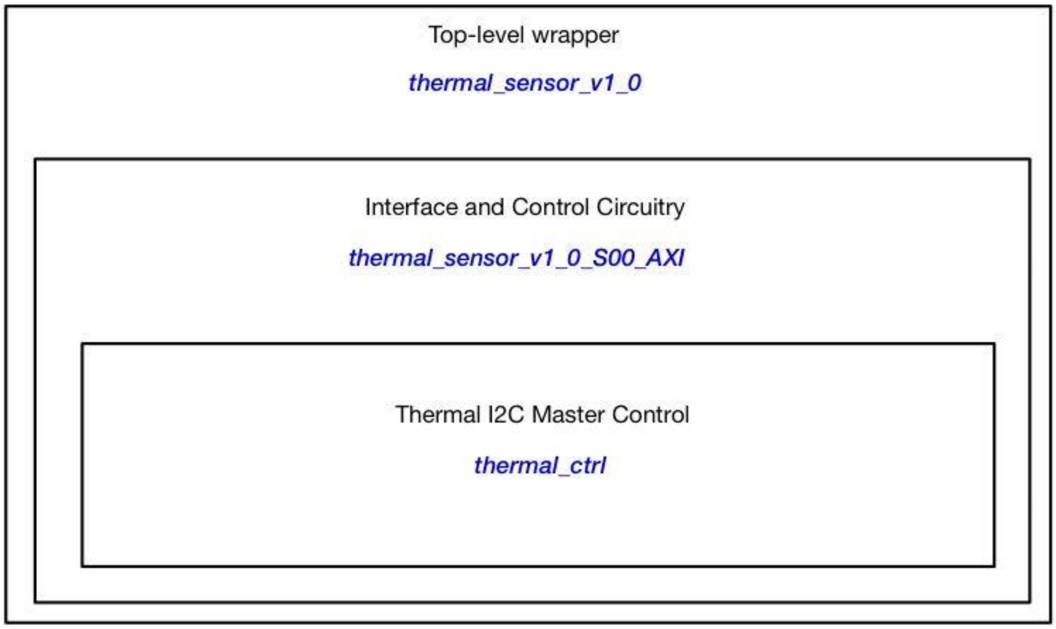

Top-level wrapper
thermal_sensor_v1_0
Interface and Control Circuitry
thermal_sensor_v1_0_S00_AXI
Thermal I2C Master Control
thermal_ctrl

Figure 3-19 Top-level of thermal sensor module

The communication method of the thermal sensor and Zybo was similar to the one of the temperature sensor. The process in sending a control frame that addressed the device address, read/write bit, and command bits to write data to or read data from thermal sensor was similar to what described in chapter 3.6. However, there were some differences in the amount of receiving bus signals and an enable input to let Zybo select when the level-translating I^2C bus repeater was active. In the thermal sensor controller, this module needed a register to hold 35-bytes bus signals from the thermal sensor. Additionally, the enable input was activated in Idle state, just before starting the start state. Figure 3-20 shows the signal of the thermal sensor. After initial steps of writing device address, the transaction would return back to the start state when the master device recognized no-acknowledge reply. A bit counter operated during a data read-back from the thermal sensor to keep track of when the master should read output data from the thermal sensor. The output data format of the Omron thermal sensor had 16-bit width for each temperature value. There were totally 17 such values, with the first one being a PTAT, a reference temperature value only used internally by the sensor. The last 8-bit data of the output was a packet error check code (PEC) that only worked for SM bus interface.

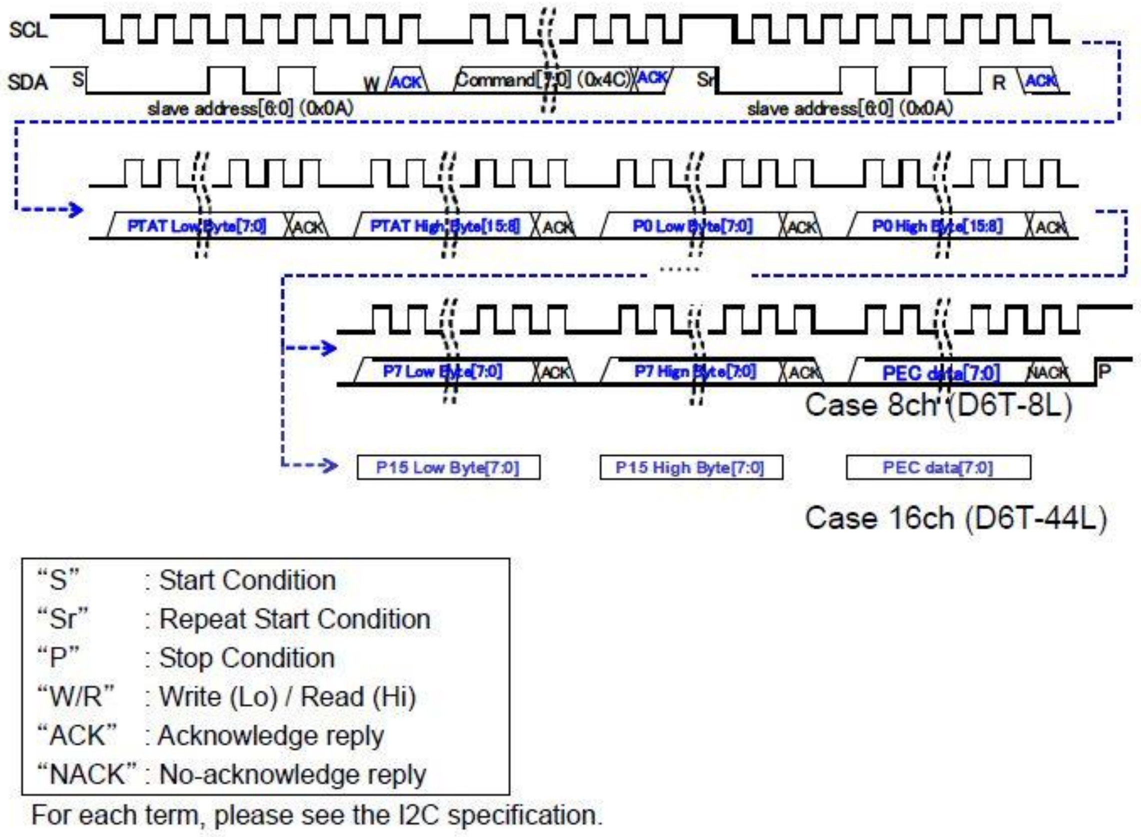


Figure 3-20 – Signal chart of the Omron D6T-44L thermal sensor

In the thermal sensor AXI module, a 280-bit register named thermal_value was created to hold the input data from the thermal sensor. Besides, this module used nine 32-bit access registers to send the input data to processing system. Table 3-1 shows the output data format of the software in the processing system.

Accessed registers	thermal_value bits position	Relevant pixels
slv_reg0	thermal_value[279:248]	PTAT low & high - P0 low & high
slv_reg1	thermal_value[247:216]	P1 low & high - P2 low & high
slv_reg2	thermal_value[215:184]	P3 low & high - P4 low & high
slv_reg3	thermal_value[183:152]	P5 low & high - P6 low & high
slv_reg4	thermal_value[151:120]	P7 low & high - P8 low & high
slv_reg5	thermal_value[119:88]	P9 low & high - P10 low & high
slv_reg6	thermal_value[87:56]	P11 low & high - P12 low & high
slv_reg7	thermal_value[55:24]	P13 low & high - P14 low & high
slv_reg8	thermal_value[23:8], thermal_value[7:0], 8'b0	P15 low & high – PEC and 8 bit 0

Table 3-1 Output data format to the processing system

In the processing system, the software program used a loop to get all data from nine 32-bits registers and parsed it accordingly. The data was then converted to Celsius degree values. The final results were saved into an array for further use in human detection algorithm. The Omron thermal sensor could capture data four times per second. However, the software only updated data from thermal sensor one time per second.

3.8 OLED display nethod

As mentioned in chapter 2.6, this module used a 128x32 OLED graphic display panel with write mode only. An OLED controller was built as a slave AXI peripheral to allow the processor system to access to the OLED display buffer through memory-mapped registers. This controller had seventeen 32-bit access registers, where sixteen of them were data registers and the seventeenth register was used for commands to display data or clear screen. Values of these registers were controlled by the software in the processing system.

Zybo communicated with the Digilent PmodOLED screen through a standard SPI interface. Similar to temperature and thermal sensor modules, the OLED controller had a hierarchical design where it delegated its functionalities to different blocks. Figure 3-21 shows the OLED controller design. The top-level wrapper module connected to the Zybo processing system. Additionally, the top level got the 100 MHz clock supplied by the processing system, and then generated slower serial clock in sub-modules. The following paragraphs will explain the sub-modules in more details.

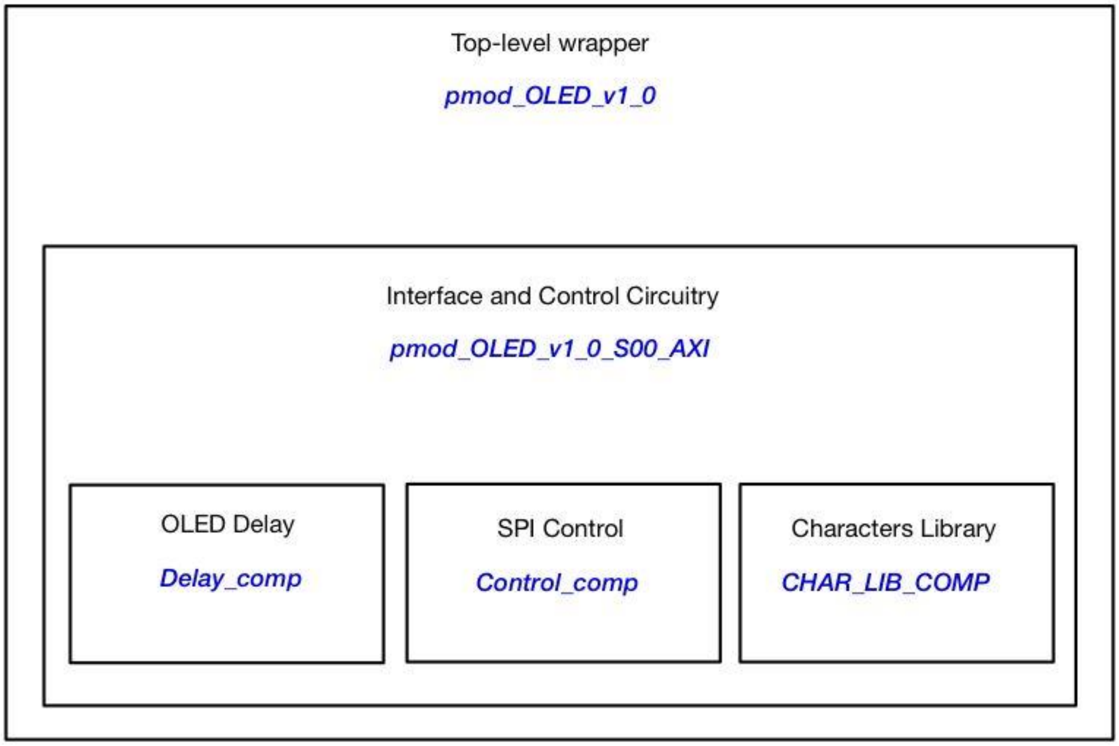


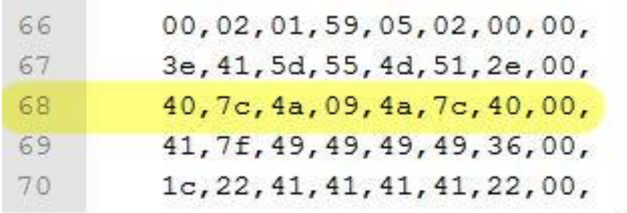
Figure 3-21 Top-level wrapper of pmod_OLED_v1_0

In the programmable logic component, the OLED controller was implemented using Verilog from the PmodOLED open source code of Digilent [50]. The original example code was developed on a Spartan-6 based Nexys3 board so it had to be retargeted to the programmable logic unit of Zybo in this project. The OLED controller was responsible for initializing the OLED display panel according to the manufacturer’s specifications. The OLED delay block, SPI Control block and Characters Library from the manufacturer were applied to initialize the OLED display panel. All modules in the programmable logic were implemented as a finite state machine (FSM) to fit measured time intervals. This SPI interface had an enable D/C pin for data/command control. This pin was set high for display buffer access and low for command access [51]. Therefore, when D/C pin was set low, the initialization was done by the OLED controller sending bursts of bytes as commands, and then when D/C pin was set high, the OLED controller would send bursts of bytes as display data to the OLED panel. These steps were separated by measured time intervals [52].

To communicate and transfer data from the OLED controller to the OLED screen panel, this controller used a SPI Control block developed by Digilent to perform an SPI transaction. This module used the 100 MHz clock from the processing system to generate a 3.125 MHz serial clock (SCLK) as a data clock [50]. The SPI Control waited until an SPI_EN was on, then it switched to “Send” state. The module then started shifting out data byte hold in SPI_DATA to serial data out (SDO) on the rising edge of SCLK. Once it finished, the module transitioned to “Done” state and the SPI_FIN was pulled high. The module waited at “Done” state until the SPI_EN was off, and then it transitioned back to its “Idle” state.

The OLED controller also used a delay module which used a100 MHz clock supplied from the processing system to generate a 1 kHz counter to count in milliseconds. This module supplied precise timing capabilities for other modules within the OLED controller. When the DELAY_EN of this module was asserted, the counter started counting until it reached the delay value. The DELAY_FIN was pulled high when the delay module transitioned to “Done” state [50].

In order to render received data correctly as ASCII characters, Character library (CharLib), a block memory contained pre-built bitmaps was used. Digilent built this character library and made it open-source for users [50]. Since each character in this library was an 8x8 bitmap, it was stored as 8-byte parts in hexadecimal notation. These pre-built contents were found in CharLib.coe, a coefficient file for the block memory, and it was implemented in the programmable logic component. Figure 3-22 shows the bitmaps of letter “A” stored in 8 bytes in hexadecimal notation in CharLib.coe. The ASCII value of letter “A” was 65 [53]. The fixed offset of the coefficient file was 3 because of the heading [54]. Therefore, the address of letter “A” was 68.



```
66      00,02,01,59,05,02,00,00,
67      3e,41,5d,55,4d,51,2e,00,
68      40,7c,4a,09,4a,7c,40,00,
69      41,7f,49,49,49,49,36,00,
70      1c,22,41,41,41,41,22,00,
```

Figure 3-22 Bitmaps of letter “A” in CharLib.coe [50].

Pmod_OLED_v1_0_S00_AXI module was the connection between the programmable logic and processing system components. It contained the major parts of the controller including the AXI interface, sixteen data registers, and a control register. This module used a FSM to implement all required initialization states of the PmodOLED after the screen was turned on and before displaying information on the screen. Once all initialization states were finished, the OLED controller waited for a trigger on the control register. When the software set the display trigger on, the OLED control could display characters on the screen. If the software set clear trigger on, the OLED control would clear the screen.

In the processing system component, after the initialization was completed, the controller provided the processor system access to the OLED display buffer through a memory-mapped register. As mentioned, the OLED Controller used sixteen 32-bit data registers to store character addresses in the coefficient file and a control register as a trigger to display information on the screen or clear the screen. Each data register could hold four characters, 8 bits for each. Figure 3-7 shows these data registers and their display location on the OLED screen. A driver was developed similar to the driver of ZedboardOLED used to communicate with the OLED Controller [26]. The driver implemented the functions to print a message, print a character and clear the screen. The software program sent the data from sixteen data registers and display/trigger of control register over the SPI interface to the OLED screen. To display new information on the screen, the OLED screen had to be cleared before sending new characters.

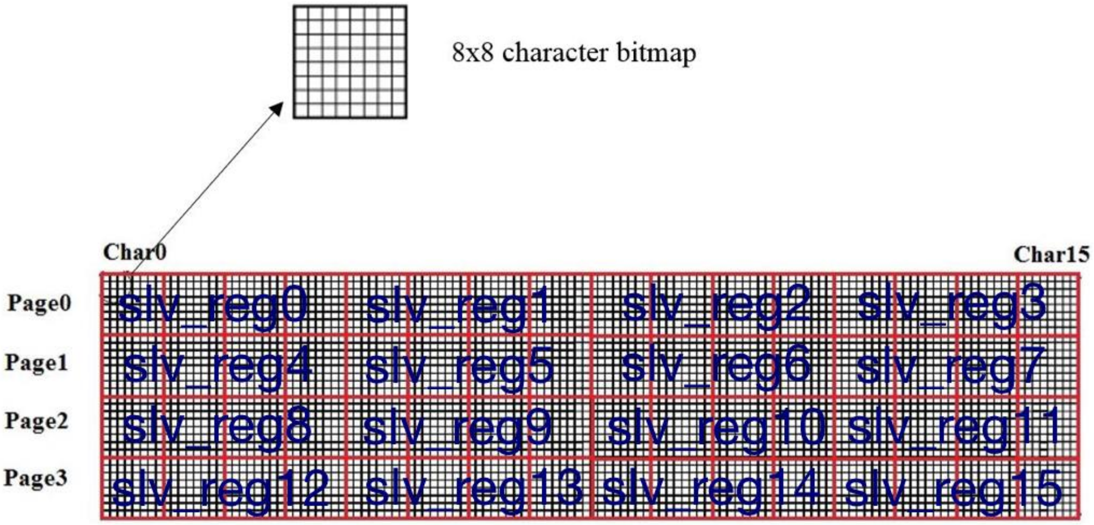


Figure 3-23 Data register and its relation to the physical OLED screen [26].

3.10 Four push-buttons

The push buttons were implemented in the programmable logic to allow users to input configuration options to the thermostat. The push buttons controller got signals from four buttons and then sent these signal values over a debounce button module to make sure the buttons were definitely pressed and to avoid unpredictable results. The push button controller also dedicated a 32-bit access register for each button. The push button AXI module would send an output signal to the processing system via AXI interconnect interface. In the processing system, the software used a 32-bit value of the pressed button to receive user configuration input. Figure 3-24 shows the push-buttons on Zybo.

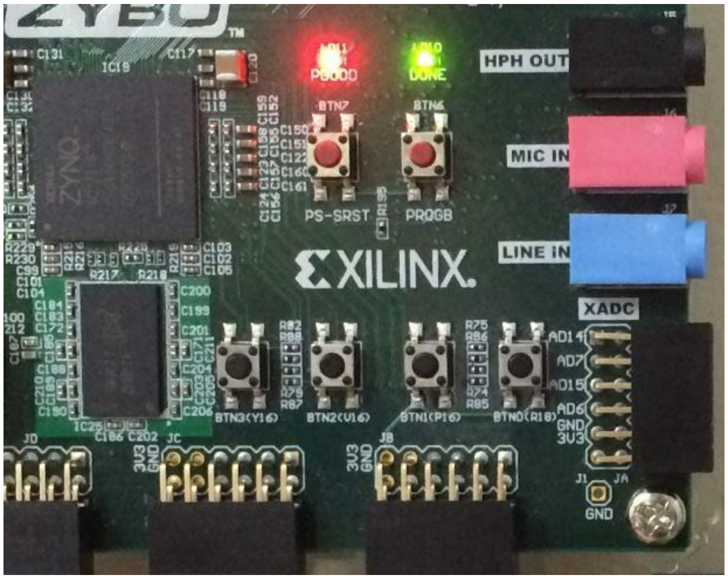


Figure 3-24 Four push-buttons attached on Zybo

3.11 HVAC unit transmission to Zybo

Zybo has a maximum output voltage at 3.3V while the HVAC system uses the transformed low-voltage source at 24V to connect to the thermostat. As mentioned in section 2.1, a single stage HVAC system has five main wires. The red wire drives the power at 24V to yellow wire (AC), the white wire (Heater) or green wire (Fan) if necessary. The black wire is terminated to yellow, white and green wires to complete the circuit. For Zybo to control the HVAC system, a four channel relay interface board was used. Input ports of the relay received controlling signals from the Zybo (active-low signals) while the output ports of the relay connected to the air compressor (Yellow), the heater (White) and the fan (Green). Figure 3-25 shows a 4-channel relay interface board used in this project.

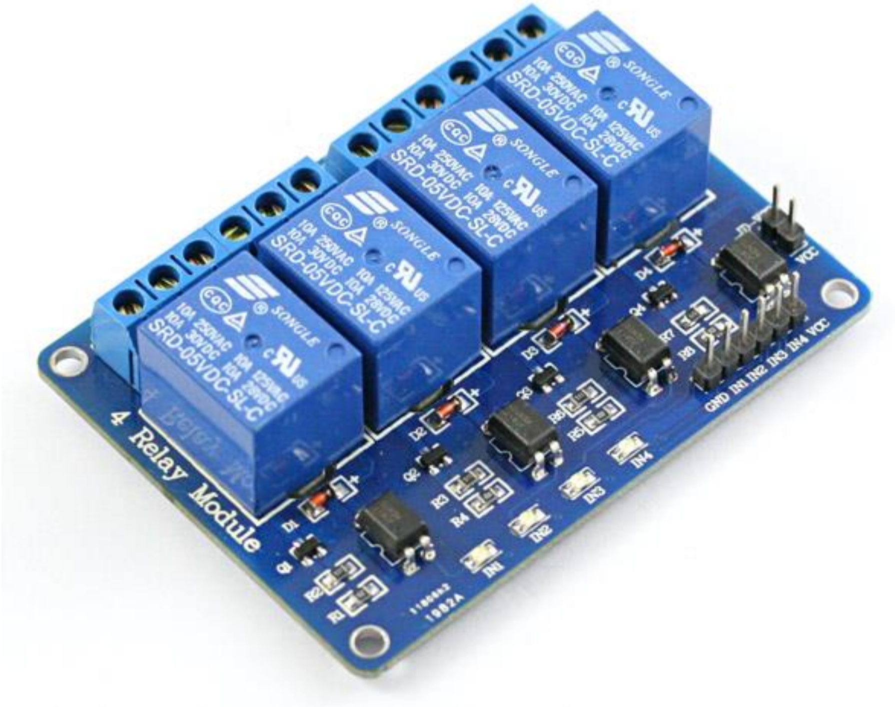


Figure 3-25 4-channel relay interface board used to control the HVAC system [55].

An HVAC relay module was implemented as a slave AXI peripheral on the programmable logic component, with one 32-bit access register for control. Figure 3-26 shows a block diagram of the relay model used to implement the HVAC system.

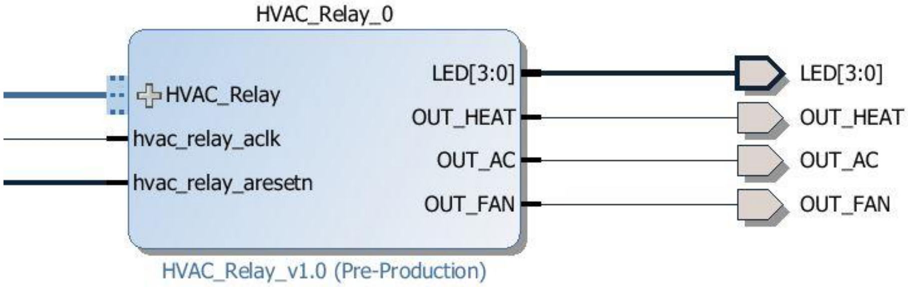


Figure 3-26 HVAC relay block diagram

When a certain temperature level needed to be maintained, the software assigned a 32-bit value to the control register based on the appropriate HVAC mode. The HVAC relay module on programmable logic would read this data from the processing system component, and map Boolean expression in low active to relevant output pins on the programmable logic. Table 3-2 shows a truth table of the HVAC relay module. This table explains how Zybo converted input signal data from the processing system and to low active output signals and sent to the relay.

HVAC relay truth table in active-low signal				
Input (32 bits)	OUT_HEAT	OUT_AC	OUT_FAN	Appropriate HVAC Mode
0x00000000	1	1	1	Off
0x0000000F	1	1	0	Fan
0x000000F0	1	0	0	Cool
0x000000FF	0	1	0	Heat

Table 3-2 HVAC relay truth table

In the HVAC relay module, the 32-bit data from the processing system used to operate the HVAC system was also used to control indicated built-in LEDs on Zybo. Table 3-3 shows the LEDs for each relevant mode when the HVAC relay module received an appropriate input value from the processing system. Figure 3-27 below demonstrates how Zybo operated the HVAC system in the heating mode indicated by the 3rd and 4th green LED.

HVAC indicated LEDs truth table in active-high signal			
Input (32 bits)	LED[3:0]	HVAC Mode	Indicated LEDs
0x00000000	4’b0001	Off	1st LED
0x0000000F	4’b0010	Fan	2nd LED
0x000000F0	4’b0110	Cool	2nd & 3rd LED
0x000000FF	4’b1100	Heat	3rd & 4th LED

Table 3-3 HVAC indicated LEDs truth table

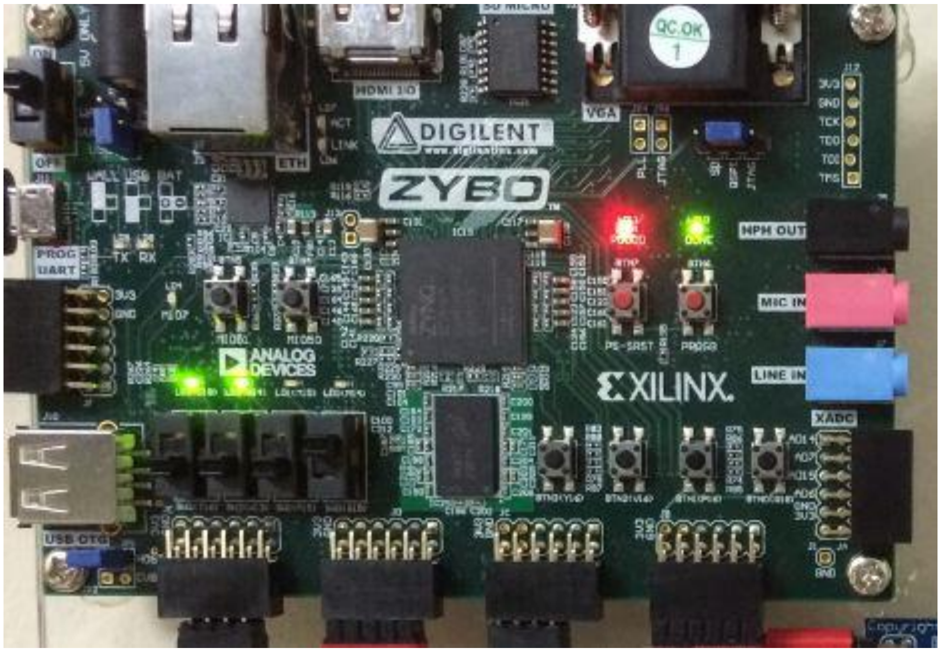
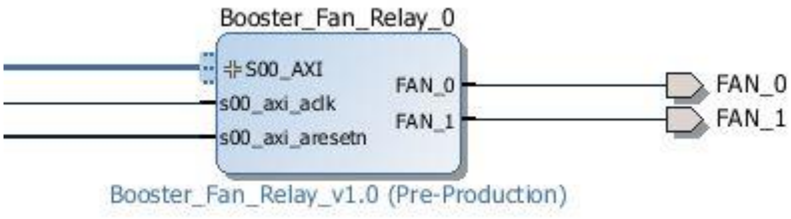


Figure 3-27 3rd and 4th indicated LED showed Zybo turned on heating mode

3.11 Register Booster Fan

Since the register booster fan operated at high voltage mentioned in Chapter 2-5, Zybo used a 4-channel relay interface board to control two register booster fans. The booster_fan_relay top module was generated similarly to the HVAC_Relay module. The top module connected to the processing system to receive bus signals from the software. The booster_fan_relay_v1_0_S00_AXI stored the data in an accessed register, and then it sent to the booster_fan_relay_ctrl to check conditions of the truth table. Based on the result shown in Table 3-4, Zybo would send a active-low output signal to the booster fan relay to turn on the true fan.



In data from the software	Fan 1 (active low signal)	Fan 2 (active low signal)	Operation
32’h00000000	1’b1	1’b1	No fans
32’h0000000F	1’b0	1‘b1	Fan 1
32’h000000F0	1’b1	1’b0	Fan 2
32’h000000FF	1’b0	1’b0	Fan 1 & Fan 2

Table 3-4 Truth table of booster fan relay module

3.12 Human Detection Algorithm

3.12.1 Received data

Data received from a thermal sensor was a 4x4 matrix, containing temperatures of objects in a room. The thermal sensor captured temperatures of different objects in a region, and presented them as one average temperature value in the 4x4 matrix. Figure 3-28 below shows an Omron thermal sensor that generates a thermal image contained 16 pixels.

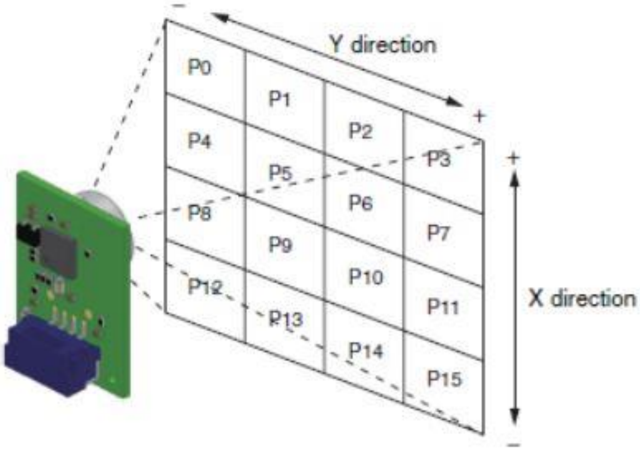


Figure 3-28 Omron thermal sensor generated thermal image with 16 pixels.

For convenience, the following terms are going to be used in the report:

- The 4x4 matrix is called ‘thermal image’ and a value in the matrix is called ‘pixel’.
- When a pixel is called bright, its temperature value is higher than room temperature. Respectively, a pixel is dim when its temperature value is lower than room temperature.
- A pixel containing human is called a human pixel.
- A pixel not containing human is called a non-human pixel. It can contain any other objects, including heat sources.

Naturally, since objects in a room have different properties, their surface temperatures are also different. Objects in a living room typically include but not limited to tables, chairs, sofa, TVs, computers, lights, made from some materials: wood, leather, fabric, metal, plastic, etc. Objects in a bedroom usually include a bed with a mattress covered with bedding sheets, a dresser, a wardrobe closet and a light. Other objects such as fans, lights, portable A/C unit or heater, might be found in different rooms of the house.

Although human skin temperature tends to be stable at 33 ^{0}C, data received from the thermal sensors does not show the same degree. In some cases, the data shows that pixels containing human are brighter than pixels without a human. However in other cases, especially when a human is far from a thermal sensor, human pixels may have temperatures lower than room temperatures. This factor leads to a conclusion that, one instance of data taken does not show much information, and I should look at how the data change throughout a period of time.

In order to obtain a training sample dataset, I set up a test environment as described in Chapter 5. After setup, a series of samples were taken. Every sample was 5- minute long, during which data was taken from thermal and temperature sensors of 2 rooms every second. A sample featured 1 of these 3 cases: (1) an empty room, (2) at least a sedentary human or (3) a moving human. In total, fourteen pairs of samples were obtained in preparation for analysis.

3.12.2 Temporal changes of data

3.12.2.1 Sample data

In a sample, a data point was taken every second in a room, which made up to 300 data points for the whole 5 minutes. Each data point contained 19 comma-separated numbers, in the following order shown in Figure 3-29. Table 3-4 shows the format of a data point.

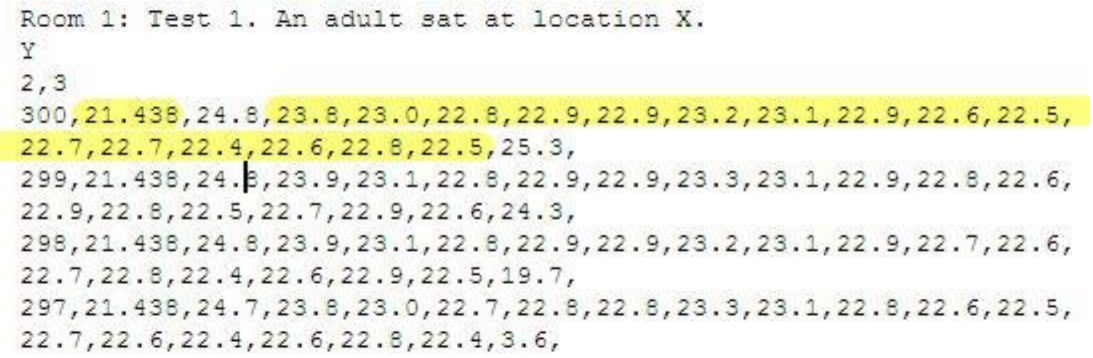
Position	Description
1	Time count in second, starting from 300 and ending at 1.
2	Temperature value read by temperature sensor
3	PTAT (from thermal sensor, not used)
4-19	Thermal values read by thermal sensors
20	PEC (from thermal sensor, not used)

Table 3-5 Format of data point

The two important set of values were numbered at position 2 and 4-19:

- Temperature value: Indoor temperature of the room in a test at a moment. The indoor temperature was taken in Celsius degree.
- Thermal values: pixel values in the room in the test. The values were also taken in Celsius degree. Pixel indices were conventionally marked from 0-15.

For each sample where at least a human was present, its human pixel locations were manually marked for further analysis. Below is an example of a sample. The temperature and thermal values at position 2 and 4-19 were highlighted in yellow:



```
Room 1: Test 1. An adult sat at location X.
Y
2,3
300,21.438,24.8,23.8,23.0,22.8,22.9,22.9,23.2,23.1,22.9,22.6,22.5,
22.7,22.7,22.4,22.6,22.8,22.5,25.3,
299,21.438,24.8,23.9,23.1,22.8,22.9,22.9,23.3,23.1,22.9,22.8,22.6,
22.9,22.8,22.5,22.7,22.9,22.6,24.3,
298,21.438,24.8,23.9,23.1,22.8,22.9,22.9,23.2,23.1,22.9,22.7,22.6,
22.7,22.8,22.4,22.6,22.9,22.5,19.7,
297,21.438,24.7,23.8,23.0,22.7,22.8,22.8,23.3,23.1,22.8,22.6,22.5,
22.7,22.6,22.4,22.6,22.8,22.4,3.6,
```

Figure 3-29 Data points of a test result

A sample had the following format:

- Line 1: room number, test number, and a short description.
- Line 2: A single character: Y if the room had a human presence, N if the room was empty. This character was input manually after the sample was taken.
- Line 3: A comma separated list of human pixel indices in the sample. If the room was empty, the list contained a single number -1.
- Rest of the lines: Each line was a data point in the described format.

3.12.2.2 Data normalization

Another observation that I had was, every sample had a different room temperature, which made analysis tasks much more difficult when comparing pixel values across different samples. Therefore, all data points were normalized using the following formula:

$$P_{i\_t} = P_{i\_t} - \overline{room\_temp} \quad [1]$$

Where:

- P_{i_t}: value of pixel i in time t.
- $\overline{room\_temp}$: average room temperature during the 5-minute period.

3.12.2.3 Descriptive statistics of sample data

To observe how data was formed throughout the test, and what statistical distribution of the data might be, I calculated the mean and standard deviation values every pixel in the thermal image, based on the following formulas:

$$\overline{\imath_{\imath}} = \frac{\mathrm{P}[i_1] + \mathrm{P}[i_2] + \ldots\text{.}+ \mathrm{P}[i_{299}] + \mathrm{P}[i_{300}]}{300} \quad [2]$$

$$\sigma_i = \frac{\sqrt{(\mathrm{P}[i_1] - \bar{\imath}_1)^2 + \cdots + (\mathrm{P}[i_{300}] - \bar{\imath}_{300})^2}}{300} \quad [3]$$

Where:

- i : a pixel in the thermal image, i = 1 to 16.
- $\overline{\imath_{\imath}}$: mean value of pixel i in 5 minutes (300 values).
- σ_i: standard deviation of the 300 pixel i values.

The following two graphs are of the same sample. The x-axis is the pixel number from 1 to 16. The y-axis is the mean temperature of the pixel in 5 minutes. Figure 3-30 shows the data before normalization, and Figure 3-31 shows the data after normalization. The normalization process not only preserved descriptive statistics of samples, but also helped further analysis in comparing thermal values across samples.

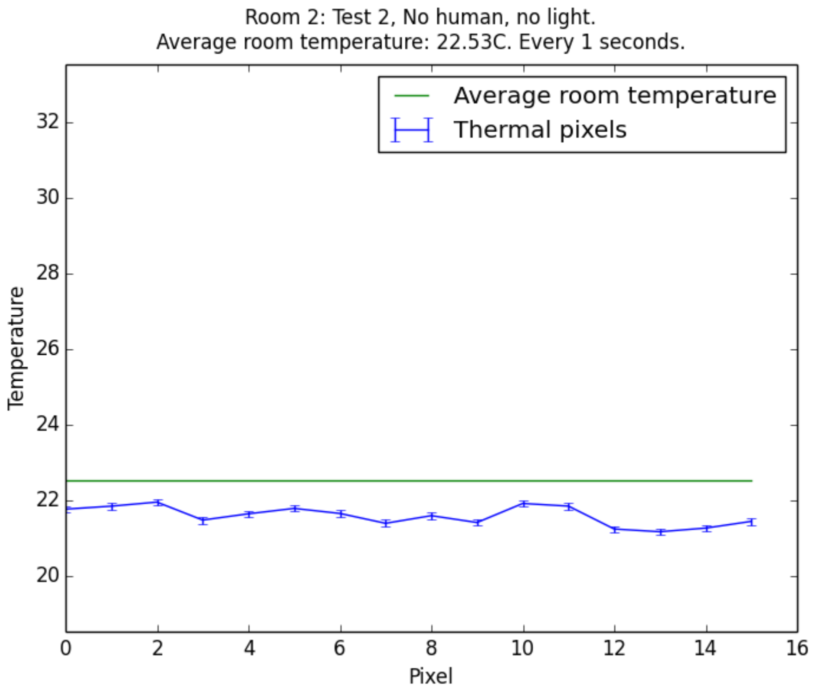


Figure 3-30 Calculated values from the thermal sensor before normalizing.

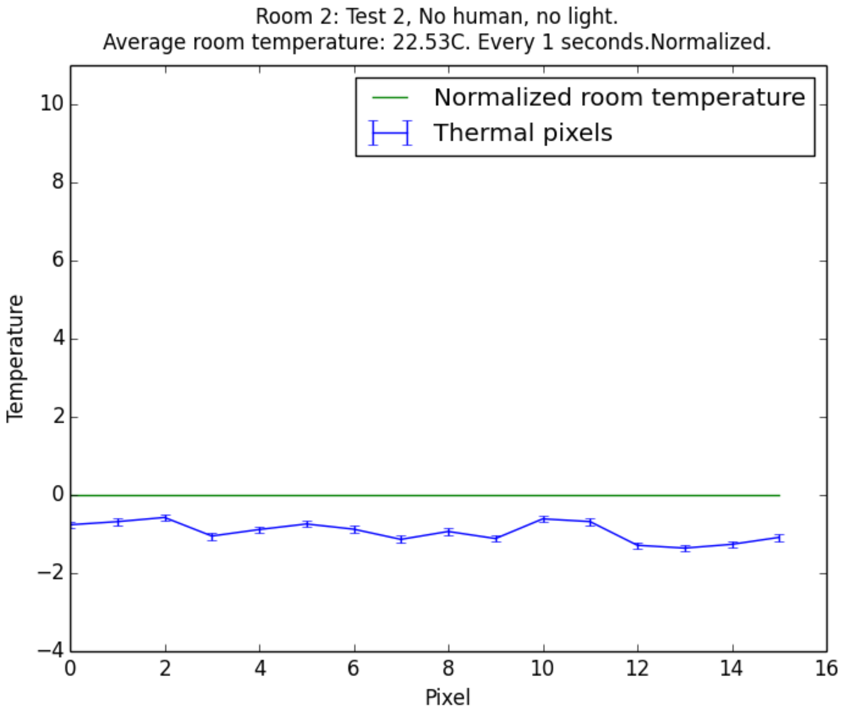


Figure 3-31 Calculated values from the thermal sensor after normalizing.

3.12.3 Analysis

3.12.3.1 Temporal changes versus spatial changes of data

After taking descriptive statistics of all pixels in all samples, each pixel was represented by two values:

- Mean brightness: the mean value of the pixel during the 5-minute period. It was $\overline{\iota_i}$ in formula [2].
- Brightness volatility: the standard deviation value of the pixel during the 5-minute period. It was σ_i in formula [3].

Spatial change was captured in a heat map of a thermal image, which showed the mean brightness of all pixels at a moment. By looking at a heat map, one should be able to identify a bright spot in an image. The heat maps below shows two thermal images, all of which contained a human. The heat map values were the **mean brightness**. Figure 3-32 shows the heat map with a bright spot, indicating the human pixel. In Figure 3-33 the heat map does not show any visible bright spot.

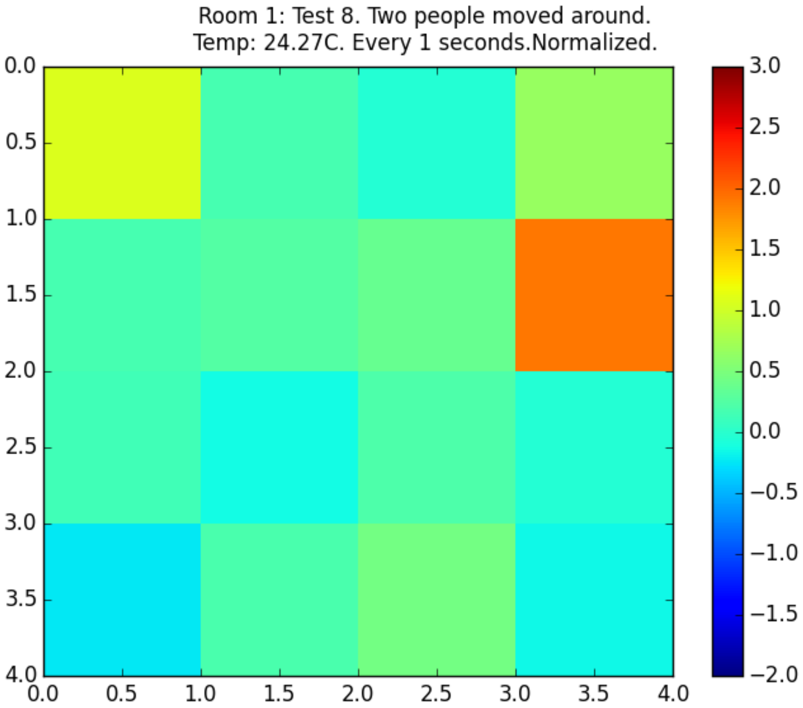


Figure 3-32 Heat map showed a bright spot indicating the human pixel

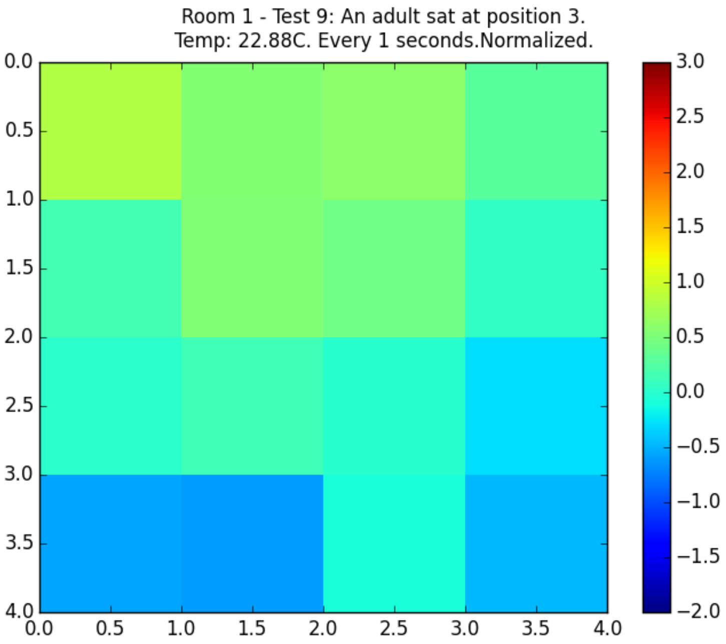


Figure 3-33 Heat map showed invisible bright spot with human presence

Temporal change was change of one-pixel brightness through time. In this project, temporal change corresponded to **brightness volatility**. The figures below shows the difference in temporal changes between two images, one with human shown in Figure 3-34 and one without human in Figure 3-35.

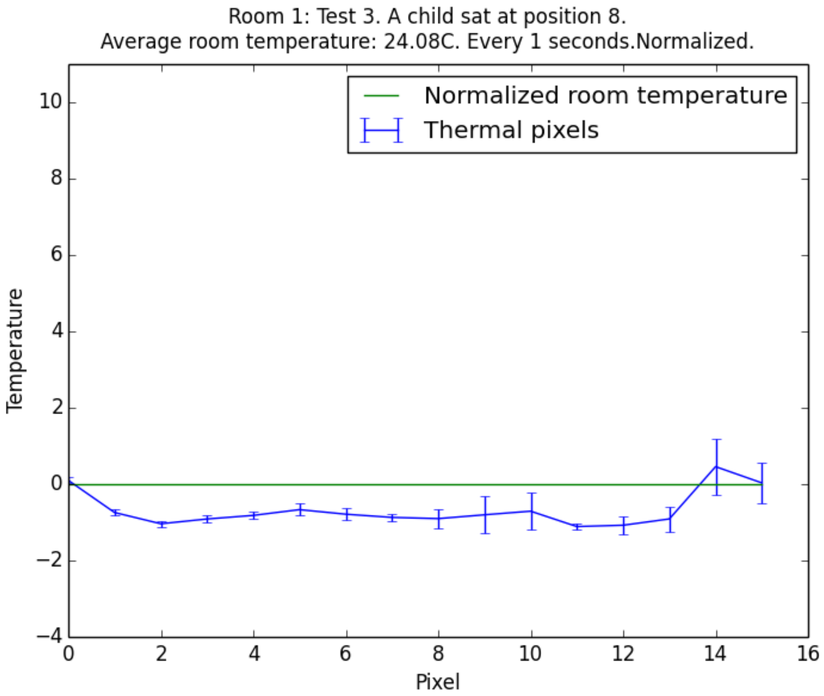


Figure 3-34 Calculated values from thermal images of test 3 in room 1 with human presence

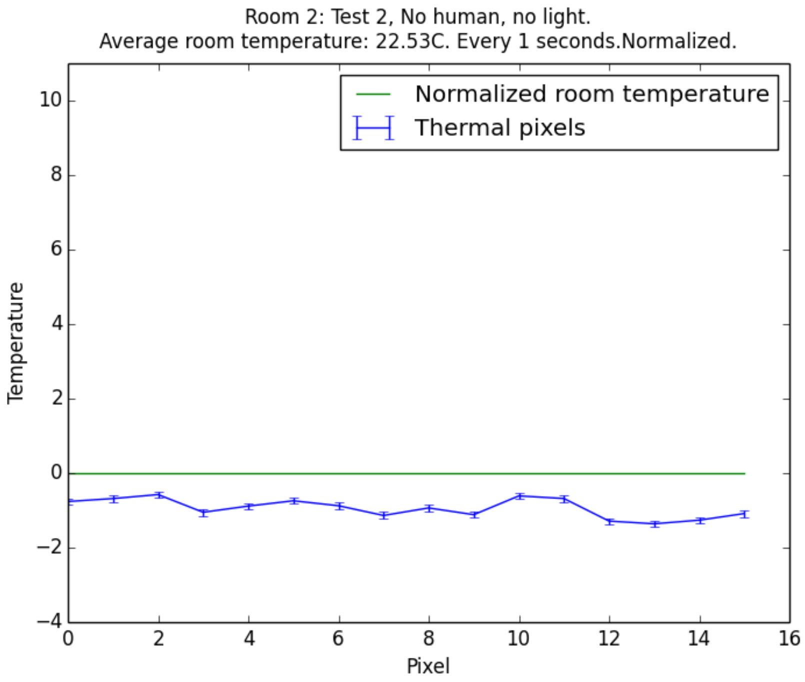


Figure 3-35 Calculated values from thermal images of test 2 in room two without human presence

Figure 3-34 and 3-35 show that human pixels have much higher brightness volatility than non-human pixels. This finding leads to a conclusion that in the sample collection process, the Omron thermal sensor was able to capture subtle temperature changes caused by natural human movements. Temporal changes of pixel values greatly contributed in detecting human presence in a room. Natural human movements included ones made by a sedentary person, such as typing, a hand or leg movement, a head turn, etc.

3.12.3.2 Human pixels versus non-human pixels

In order to identify the differences between human and non-human pixels, pixels of all samples were classified into two corresponding classes and their mean brightness and brightness volatility were calculated. The purpose of these statistical values was to know how the mean brightness and brightness volatility distributed across all samples. The following subsections describe results of this calculation.

3.12.3.2.1 Mean brightness

Table 3-6 displays descriptive statistics of the **mean brightness** of all classified pixels in 28 collected samples. Figure 3-36 to 3-38 show its histograms, where each bin is 0.1 ^{0}C.

	Min	Median	Mean	Max	Volatility
Human	-1.764	-0.545	-0.395	2.923	0.806
Non-human	-2.928	-0.76	-0.845	5.511	1.140

Table 3-6 Statistics of human and non-human pixels mean brightness values in 60 seconds

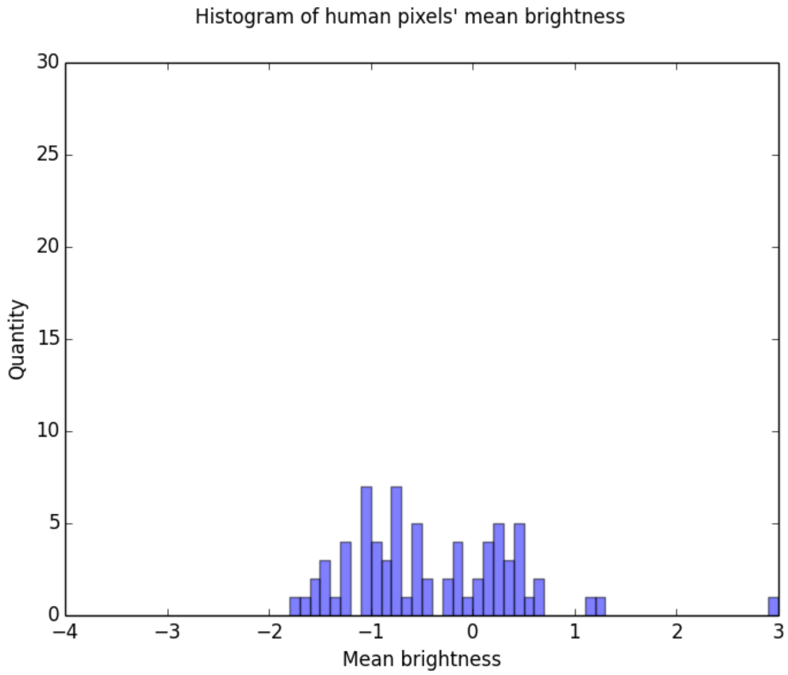


Figure 3-36 Histogram of human pixels' mean brightness

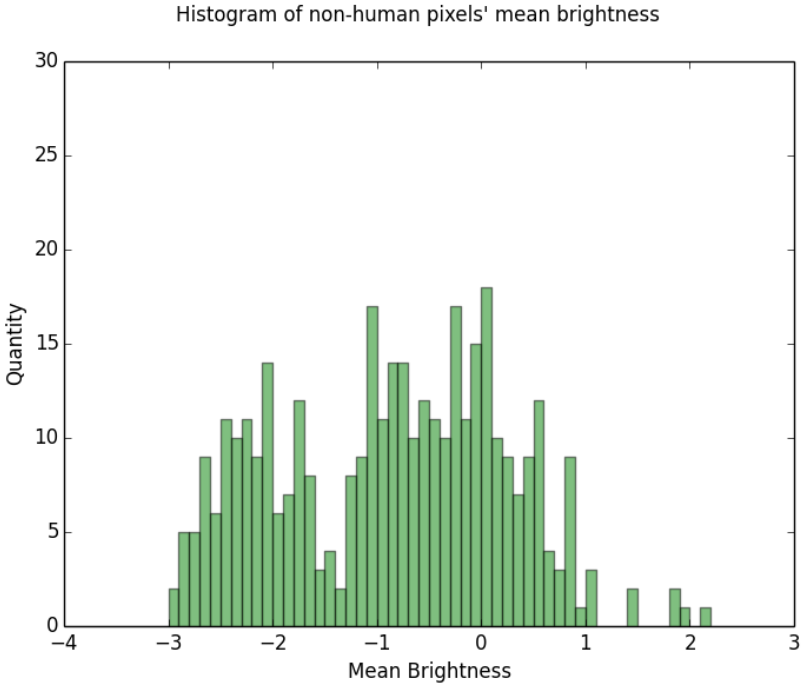


Figure 3-37 Histogram of non-human pixels' mean brightness

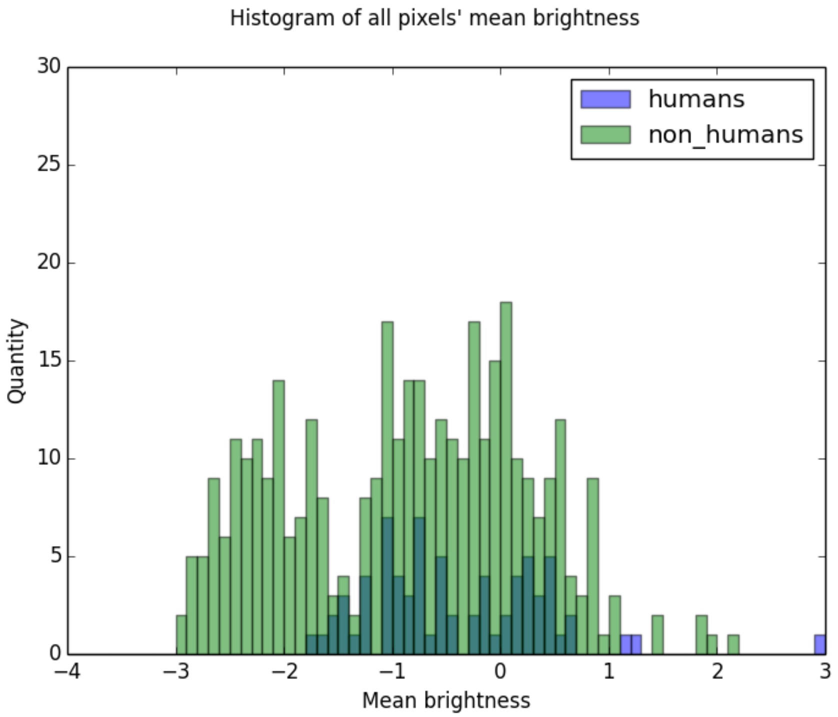


Figure 3-38 Histogram of all pixel's mean brightness

The histograms above show that human pixels were only slightly brighter than non-human pixels, hence, if the human detection algorithm were based only on pixel brightness, its accuracy could only be around 50%.

3.12.3.2.2 Brightness volatility

As mentioned before, since the thermal sensors were very sensitive in capturing temperature changes resulted from human movements, brightness volatility helped determine whether a pixel was human or non-human. Table 3-7 displays descriptive statistics of **brightness volatility** of all classified pixel in 28 collected samples. Figure 3-39 to 3-40 show their histograms, where each bin is 0.01 ^{0}C.

	Min	Median	Mean	Max	Volatility
Human	0.067	0.394	0.442	1.126	0.281
Non-human	0.05	0.085	0.104	3.840	0.203

Table 3-7 Statistics of human and non-human pixels brightness volatility in 60 seconds

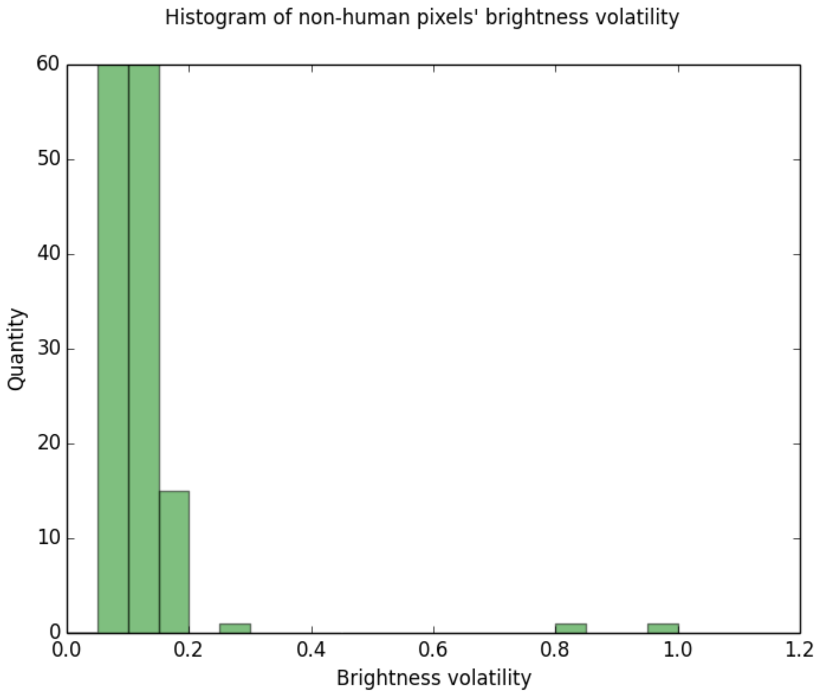


Figure 3-39 Histogram of non-human pixels' brightness volatility

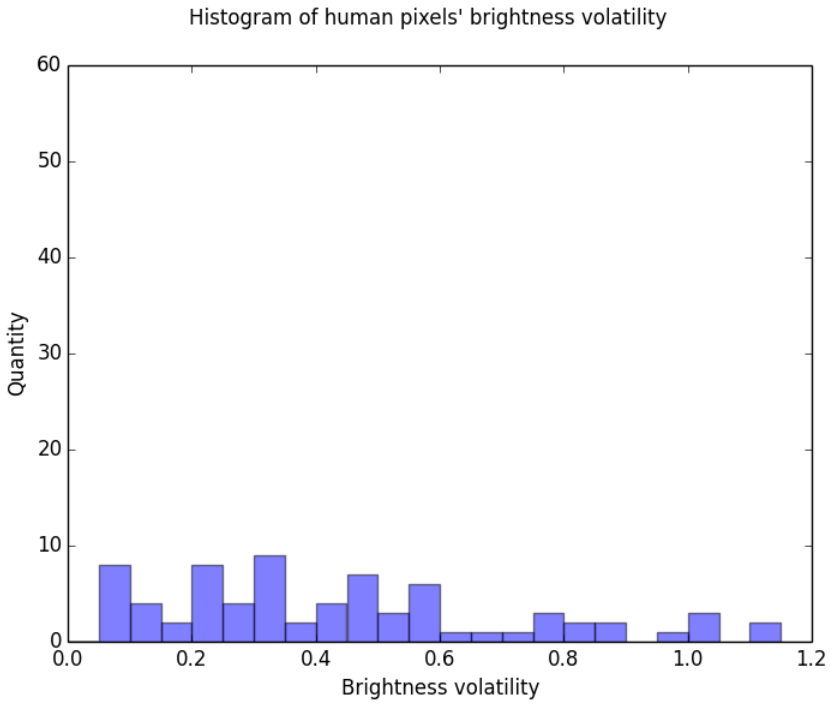


Figure 3-40 Histogram of human pixels' brightness volatility

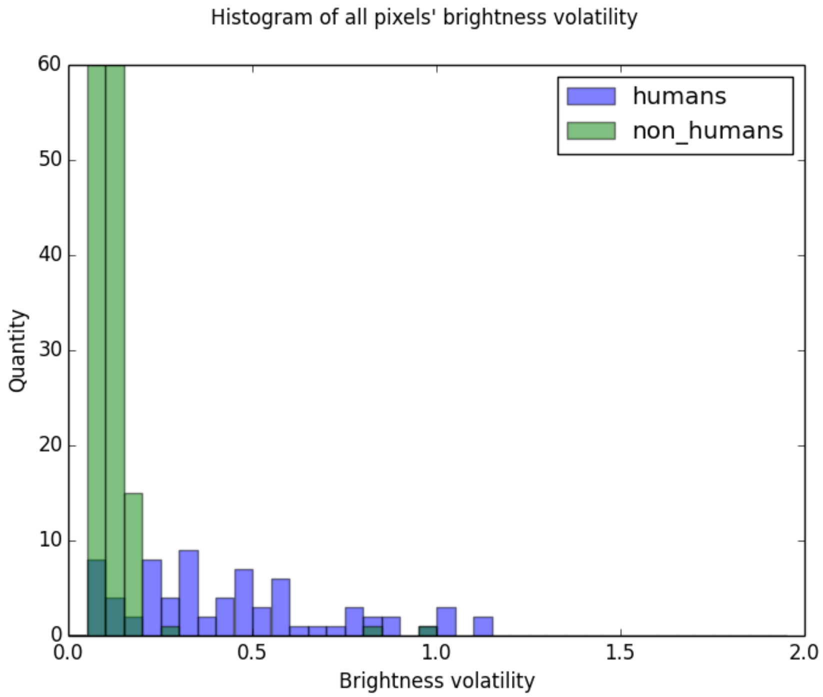


Figure 3-41 Histogram of all pixels' brightness volatility

The histograms show that while non-human pixel brightness volatility was around 0.1 ^{0}C, human pixel volatility was significantly higher, around 0.4 ^{0}C, depending on human locations and movements.

3.12.3.2.3 Building a human detection algorithm

Through findings in 3.12.3.2.1 and 3.12.3.2.2, human pixels had mean brightness ranging from -1.764 to 2.923, taken from Table 3-6 and brightness volatility ranging from 0.067 to 1.126, taken from Table 3-7. A human detection algorithm was built as follows:

```
1.  For second s = 1 to 60:
2.  T[s] = room temperature at second s
3.  P[][s] = pixel brightness at second s. (16 values)
4.  mean_temp = mean(T) // mean of 60 temperature values
5.  human = False
6.  For each pixel p in the thermal image:
7.  mean_brightness[p] = mean(P[p])
8.  brightness_volatility[p] = standard_deviation(P[p])
9.  if (brightness_volatility[p] > minimum_volatility and
10. brightness_volatility[p] < maximum_volatility and
11. mean_brightness[p] > minimum_brightness and
12. mean_brightness[p] < maximum_brightness):
13. human = True
```

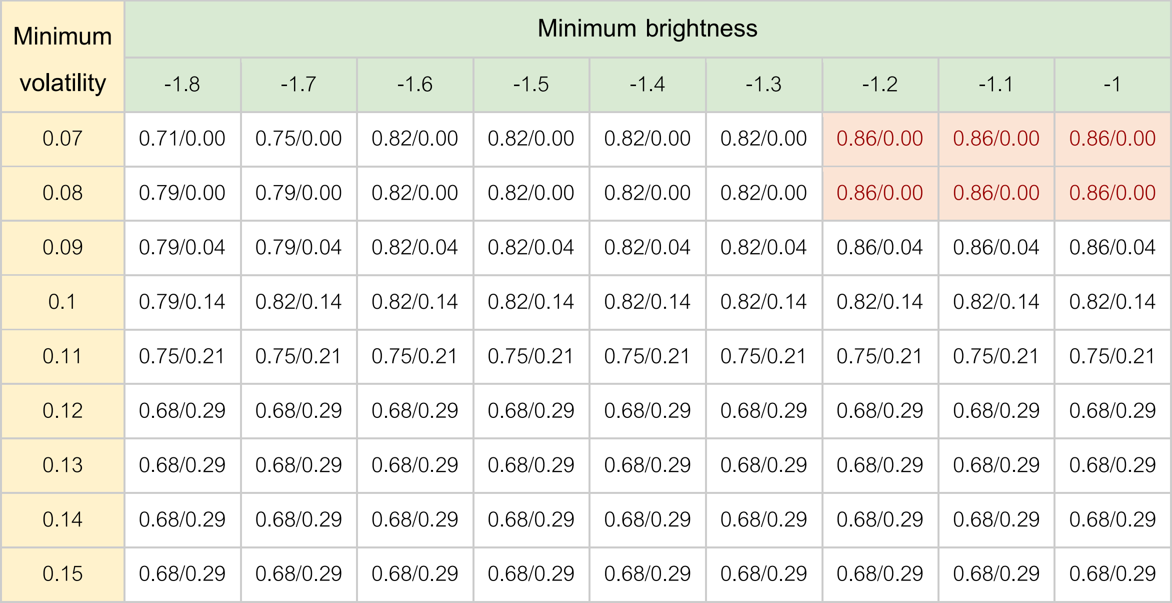
At line 1-3, data from sensors was collected in 60 seconds, as opposed to 300 seconds (5 minutes) when collecting sample data. This decision was a design choice after analyzing the sample data. Such data was taken in 5 minutes as an exploration step only to discover the human detection algorithm. In each sample, there was no big difference in data values among each 60 seconds in the 5-minute period, since the setup scenario was the same from the beginning to the end of the period, i.e. an empty room remained empty, a room with human presence remained the same. Moreover, 60 seconds were long enough to capture natural human movements, such as head turns, leg or hand or torso movements. There was no need to collect data for 300 seconds before calculating statistics.

Line 10 and 12 were added to eliminate outliers caused by heat sources other than human, such as portable heaters, light bulbs, etc. maximum_brightness was 2.923. maximum_volatility was 1.126.

To identify the best minimum_brightness and minimum_volatility parameter pairs, the algorithm was run on the samples multiple times, each time with a unique combination of the two values. Chosen values were based on the statistics in Table 3-6 and 3-7, with minimum_brightness started from -1.764, and minimum_volatility started from 0.067. In each of the run, result from the algorithm was compared with the result manually marked, to identify whether the algorithm was correct or not. Results were classified into the following four categories:

- True positive: there was at least a human in a room and the algorithm recognized him or her.
- True negative: there was no human in a room and the algorithm identified no human.
- False positive: there was no human presence in a room, but the algorithm falsely identified a person.
- False negative: there was a person in a room, but the algorithm did not recognize him or her.

Among the four categories, false negative results had the most negative impact on the overall functionality of the smart thermostat, since if it happened, the HVAC would not run even if a human was present in a room. Therefore, the best strategy in choosing the best parameter pair was to maximize true detections and minimize false negative detections. Table 3-8 below displays the most significant results from all the runs. Other results are omitted due to low accuracy. Each of the cells contains **accuracy rate/false negative rate** results from the run of the corresponding parameter pair. In the results table, the best pairs of mean brightness/volatility are marked in red.



Minimum volatility	Minimum brightness								
	-1.8	-1.7	-1.6	-1.5	-1.4	-1.3	-1.2	-1.1	-1
0.07	0.71/0.00	0.75/0.00	0.82/0.00	0.82/0.00	0.82/0.00	0.82/0.00	0.86/0.00	0.86/0.00	0.86/0.00
0.08	0.79/0.00	0.79/0.00	0.82/0.00	0.82/0.00	0.82/0.00	0.82/0.00	0.86/0.00	0.86/0.00	0.86/0.00
0.09	0.79/0.04	0.79/0.04	0.82/0.04	0.82/0.04	0.82/0.04	0.82/0.04	0.86/0.04	0.86/0.04	0.86/0.04
0.1	0.79/0.14	0.82/0.14	0.82/0.14	0.82/0.14	0.82/0.14	0.82/0.14	0.82/0.14	0.82/0.14	0.82/0.14
0.11	0.75/0.21	0.75/0.21	0.75/0.21	0.75/0.21	0.75/0.21	0.75/0.21	0.75/0.21	0.75/0.21	0.75/0.21
0.12	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29
0.13	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29
0.14	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29
0.15	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29	0.68/0.29

Table 3-8 Accuracy of the human detection algorithm based on minimum brightness and volatility

3.13 HVAC controlling algorithm

The thermostat allowed users to input 3 pre-set temperature values: desired temperature, low temperature and high temperature. The desired temperature was a level at which a user wanted the thermostat to maintain. This value was the optimal temperature, where a user would feel most comfortable. A user also set low and high-temperature values to specify a temperature range in which the user would feel comfortable. These 3 values could be change any time as necessary.

3.13.1 Hysteresis

The HVAC system had an on/off control action to turn the unit on/off based on the desired temperatures. The HVAC control signal outputs from Zybo were frequently changed when it updated new results each minute. This problem could shorten the life of the four-channel relay and of the HVAC system. To prevent such issue, the thermostat needed a temperature band called hysteresis between on and off operations. A user could adjust the hysteresis by changing the temperature swing value. The hysteresis calculation would be explained in next couple paragraphs.

In this project, hysteresis was the sum of the swing value and temperature sensor accuracy. The accuracy of the temperature sensor was +/- 0.25^0C mentioned in Chapter 2.5 while the swing values were either 1^0C or 2^0C depending on user input. Table 3-1 shows the calculated hysteresis from different input temperature swing value.

Temperature swings	Accuracy	Hysteresis
1^0C	+/- 0.25^0C	1.5^0C
2^0C	+/- 0.25^0C	2.5^0C

Table 3-3-9 Hysteresis of controlling algorithm

The hysteresis of HVAC modes was:

$$Hysteresis = \ swing + 2 * accuracy \text{ [4]}$$

Figure 3.5 shows the hysteresis of cool mode with a desired temperature at 23^{0}C. The AC was on when the measured temperature was higher than 24.25^{0}C. The AC was off when the measured temperature was lower than 22.75^{0}C. The hysteresis of cool mode was 1.5^{0}C.

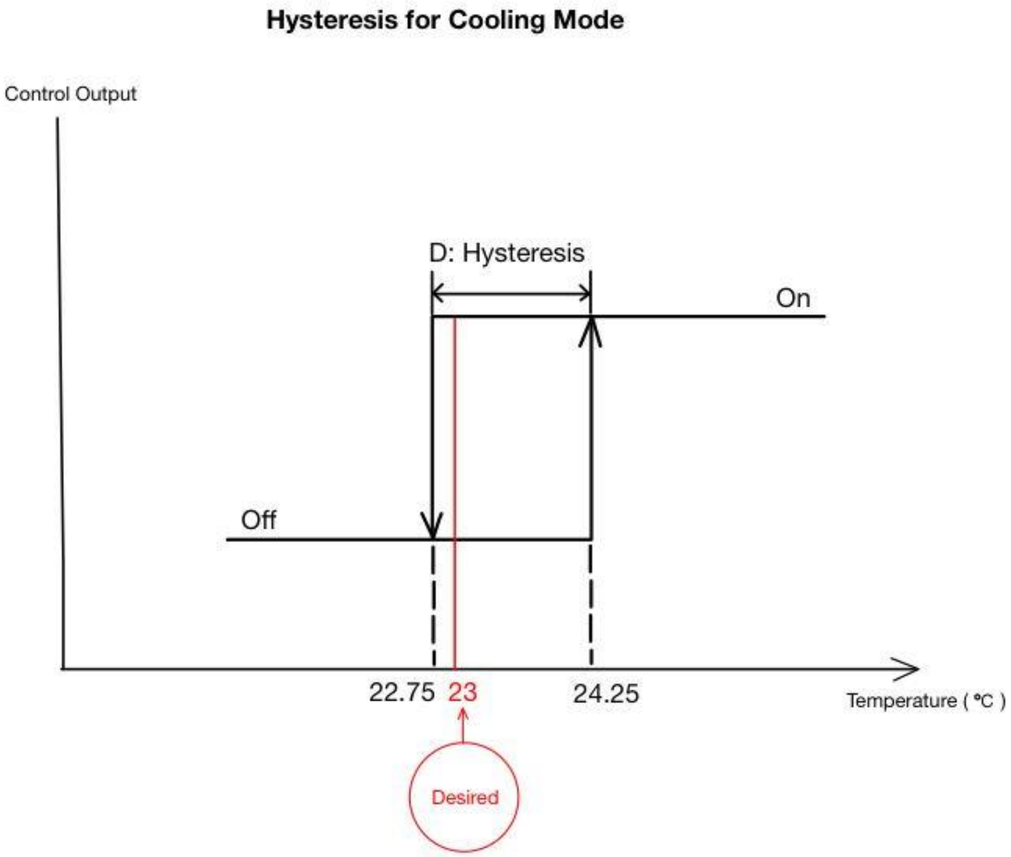


Figure 3-3-42 Hysteresis of cool mode with the desired temp at 23^{0}C

On/off condition of cool mode:

- On condition = desired temperature + (swing + accuracy)
- Off condition = desired temperature – accuracy

Similarly, Figure 3.6 shows the hysteresis of heat mode. The heater was on when measured temperature was lower than 21.75^{0}C, and then it would be off when measured temperature was higher than 23.25^{0}C. The hysteresis for heat mode was also 1.5^{0}C.

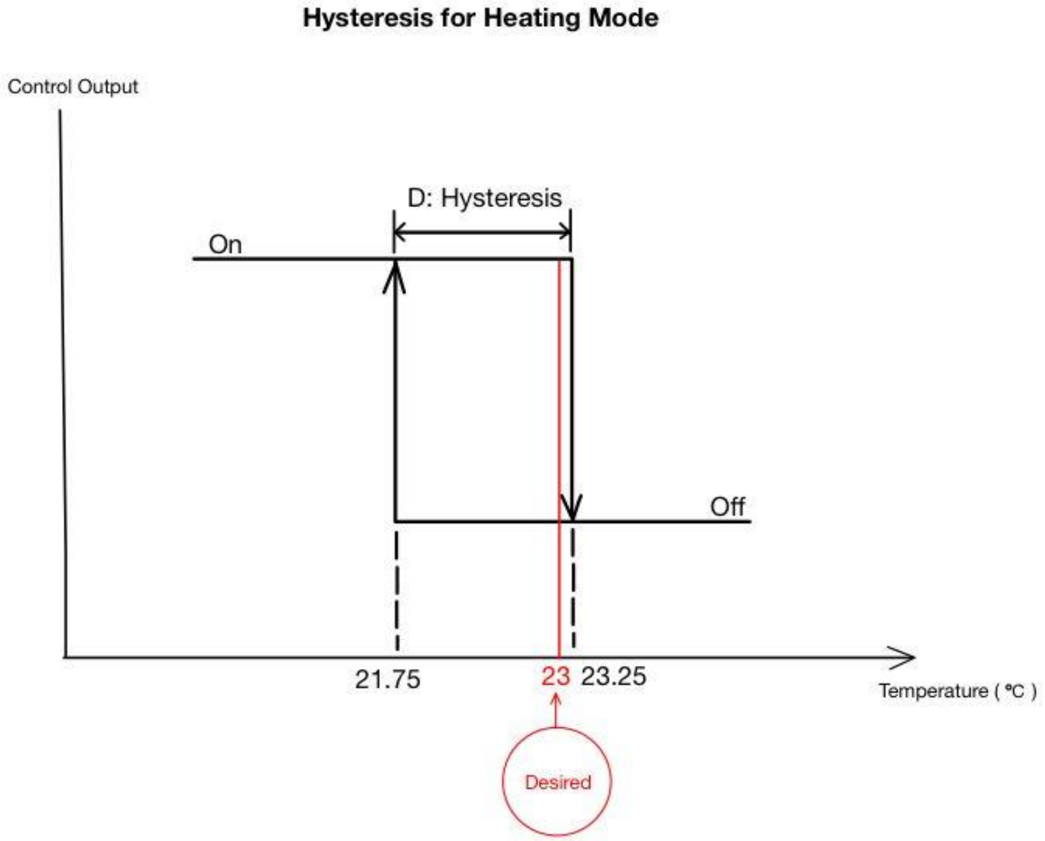


Figure 3-43 Hysteresis for heat mode with the desired temperature at 23^{0}C

On/off condition of heat mode:

- On condition = desired temperature – (swing + accuracy)
- Off condition = desired temperature + accuracy

The smart thermostat would turn on the HVAC system when either room 1 or room 2 did not satisfy the comfort zone. While the rooms might have different temperatures, the HVAC system would run until both rooms reached the desired temperature level. It meant that at least one room might already passed the desired level by a considerable amount of degrees. There was no need to wait for both rooms to pass such level by a swing value anymore, since such HVAC run would take a longer time, and at least one room temperature might become too hot (in heat mode), or too cold (in cool mode).

3.13.2 HVAC mode

The smart thermostat offered three basic modes to maintain desired temperatures in different weather conditions: heat, cool and auto heat&cool mode. When the thermostat detected human presence, it maintained indoor temperature at the desired temperature if the heat mode or the cool mode was on. Meanwhile in the auto heat and cool mode, it maintained a comfort zone between the preset low and high temperatures.

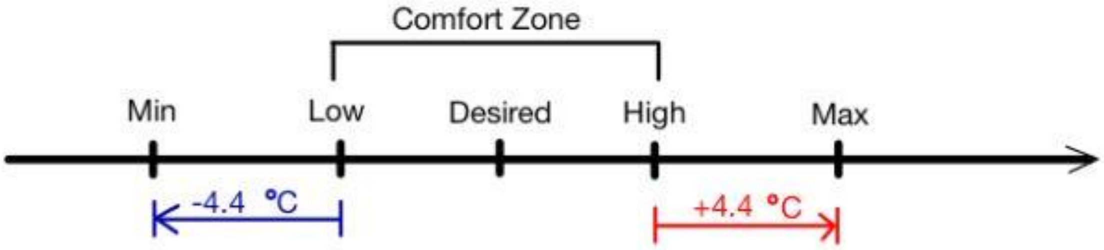


Figure 3-44 Preset comfort zone

Additionally, the thermostat had a night operation that helped users saving energy in the evening when using cool or heat mode. A set could set night mode to be on, off or auto. When the auto night operation was on, the thermostat automatically maintained the comfort zone within the low and high temperature range from 12 AM to 8 AM only. However, if the night operation was not auto, the thermostat would keep the indoor temperature within such temperature range permanently. This night operation option did not work for the auto heat&cool mode, since operation already keep the indoor temperature within such range.

To avoid the HVAC from being turned on unnecessarily, the thermostat had an energy saving state used when there was no positive human detection. Particularly, if the thermostat detected no human in the house for fifteen consecutive minutes, it automatically activated this energy saving state to maintain room temperature at larger range, between minimum and maximum temperature. Based on a guide to energy-efficient heating and cooling of ENERGY START program, if users set back, at least, 4.4^0C during winter or set up 4.4^0C during summer, they could save about $180 every year in energy costs [56]. Therefore, the minimum and maximum temperature was calculated in Celsius degree unit as follow:

$$Minimum\ temperature = \ Low\ temperature - \ 4.4℃ \ \ [5]$$

$$Maximum\ temperature = \ High\ temperature + \ 4.4℃\ [6]$$

No matter what HVAC mode was operated, the thermostat automatically maintained room temperatures at a minimum and maximum temperature when no human was home. The thermostat would automatically switch this state off within 5 minutes if it detected human.

The thermostat's hardware and software designs were explained in chapter 3. The human detection algorithm and HVAC controlling algorithm and their application in this project were also discussed. Chapter 3 showed that Zybo could be implemented as a smart thermostat, where it could detect human presence and automatically adjusted the HVAC system to maintain a user's comfort zone. Chapter 4 will explain the actual detailed implementation of the thermostat modes introduced in 3.13.2, the system user interface and it software program. Chapter 4 also discuss the total cost of the thermostat prototype, set up in 2 rooms of an apartment.

4. Implementation

Previous chapters described how the peripherals in the project interacted with Zybo as well as how the developed algorithms were used to control the HVAC system. This chapter explains how Zybo used human detection results and captured data and from other peripherals to maintain desired comfort zone automatically. Additionally, this chapter shows how Zybo received user inputs with four push-buttons and display information onto the OLED screen. The implementation to be mentioned in this chapter was set up for two rooms, which were used in testing described in chapter 5.

4.1 HVAC Modes

Zybo was developed as a smart thermostat used to control the HVAC system automatically based on pre-set desired temperatures from users and measured data from sensors. It only operated the HVAC system to maintain the comfort zone in rooms with human presence. In case no human was home after fifteen minutes, the thermostat automatically changed into energy-saving state regardless of the chosen HVAC mode. The following sections explain how the smart thermostat controlled the HVAC in each mode: cool, heat and auto cool/heat.

As mentioned in Chapter 3.13, users were allowed to enter three pre-set values for desired temperature, low and high temperature. The minimum and maximum temperature formulas were computed from low and high temperatures explained in 3.13:

$$Minimum\ temperature = \ Low\ temperature - \ 4.4℃\ [5]$$

$$Maximum\ temperature = \ High\ temperature + \ 4.4℃\ [6]$$

4.1.1 Cool mode

When cool mode was selected, the thermostat only controlled the HVAC’s air compressor (AC) to cool down indoor temperature. The thermostat turned on AC when measured temperature of a room was higher than a cutoff temperature. The value of this cutoff temperature depended on results from the human detection algorithm and the night mode switch. As explained in 3.13, The HVAC start and stop conditions in cool mode were:

$$Start\ condition = cutoff\_temperature + (swing + accuracy)\ [7]$$

$$Stop\ condition = cuttoff\_temperature - accuracy\ [8]$$

The thermostat used results from human detection algorithm in room 1 and room 2 as the highest priority to maintaining desired temperatures. There were four particular situations of human presence.

(1) When no one was at home. This situation was considered as the energy-saving state of the cool mode shown in case 1 of Table 4-1. The HVAC was not started until room temperatures went higher than the cutoff value, the maximum temperature.

(2) & (3) When only one room had human presence. The thermostat would run the HVAC when it needed to bring the room temperatures down to the specified cutoff

values. Note that the cutoff temperatures of the rooms are different, as there was no need to maintain the desired temperature level in a room without human. In other words, the thermostat would keep such room's temperature below the high level, since users may move between rooms. This design helped users stay within their reasonable comfort zone as long as possible. Additionally, only the booster fan of the room with human presence was turned on with a belief that air would be circulated to the room faster, as discussed in 3.11. This situation corresponds to case 2 and 3 of Table 4-1.

(4) When both rooms had human presence, the smart thermostat would maintain the rooms at the desired temperature level, and started HVAC with all booster fans turned on, as shown in case 4 of Table 4-1. The smart thermostat would not turn on the AC when in both rooms' temperatures were within the comfort zone.

Cool mode									
Case	**Human room 1**	**Human room 2**	**Cutoff temperature Room 1**	**Current Temperature room 1 (T1)**	**Cutoff temperature Room 2**	**Current Temperature room 2 (T2)**	**Fan room 1**	**Fan room 2**	**HVAC**
1	No	No	Maximum temperature	≥ cutoff	Maximum temperature	≥ cutoff	On	On	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off
2	No	Yes	High temperature	≥ cutoff	Desired temperature	≥ cutoff	Off	On	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off
3	Yes	No	Desired temperature	≥ cutoff	High temperature	≥ cutoff	On	Off	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off
4	Yes	Yes	Desired temperature	≥ cutoff	Desired temperature	≥ cutoff	On	On	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off

Table 4-1 Cool Mode

While cool mode was selected, if the night operation was set, the thermostat would set cutoff temperatures as in Table 4-2. Even though temperature of the room without human fell out of the comfort zone, the operation was designed with an assumption that there would be limited human activities during night operation, hence human movements between rooms were at the minimum. In case the night operation was set to be auto, the thermostat would switch to night operation at midnight and switch back to normal operation at 8:00 AM. This timeframe was set to support testing in chapter 5.

Cool operation – Night operation				
Case	**Human room 1**	**Human room 2**	**Cutoff temperature Room 1**	**Cutoff temperature Room 2**
1	No	No	Maximum temperature	Maximum temperature
2	No	Yes	Maximum temperature	High temperature
3	Yes	No	High temperature	Maximum temperature
4	Yes	Yes	High temperature	High temperature

Table 4-2 Night mode of the cool mode

4.1.2 Heat mode

In heat mode, the thermostat only controlled the heater to warm up room temperatures until it reached the desired level. The algorithm used in heat mode was similar to the one in cool mode, with values and conditions changed to support heating instead of cooling. The HVAC start and stop conditions in heat mode explained in 3.13 were:

$$Start\ condition = room\_temp < cutoff\_temperature - (swing + accuracy)\ [9]$$

$$Stop\ condition = room\_temp < cutoff\_temperature + accuracy\ [10]$$

The cutoff temperature values were also different from the ones in cool mode, depending on results from the human detection algorithm and the night operation switch. Table 4-3 below shows all HVAC start and stop conditions in details. They are similar to the ones in Table 4-1, with changes of cutoff temperature values to support heating. As in cool mode, a booster fan of a room was turned on only if the room had human presence.

Heat mode									
Case	Human room 1	Human room 2	Cutoff temperature Room 1	Current Temperature room 1 (T1)	Cutoff temperature Room 2	Current Temperature room 2 (T2)	Fan room 1	Fan room 2	Operation
1	No	No	Minimum temperature	≤ cutoff	Minimum temperature	≤ cutoff	On	On	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off
2	No	Yes	Low temperature	≤ cutoff	Desired temperature	≤ cutoff	Off	On	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off
3	Yes	No	Desired temperature	≤ cutoff	Low temperature	≤ cutoff	On	Off	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off
4	Yes	Yes	Desired temperature	≤ cutoff	Desired temperature	≤ cutoff	On	On	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off

Table 4-3 Heat mode operation

Table 4-4 shows HVAC start and stop conditions in night operation. Auto night operation also turned on night operation at midnight and turned it off at 8:00AM automatically. Regardless of whether night operation was on, the thermostat switched to an energy-saving state when no one was at home, by setting to minimum temperature to be the cutoff value.

Heat mode – Night operation				
Case	**Human room 1**	**Human room 2**	**Cutoff temperature Room 1**	**Cutoff temperature Room 2**
1	No	No	Minimum temperature	Minimum temperature
2	No	Yes	Minimum temperature	Low temperature
3	Yes	No	Low temperature	Minimum temperature
4	Yes	Yes	Low temperature	Low temperature

Table 4-4 Night mode of the heat mode

4.1.3 Auto heat&cool mode

Different from cool and heat modes which maintained room temperatures at the desired temperature only, the auto heat&cool mode automatically kept the room in a comfort zone between a cutoff temperatures range. This mode was useful at places with extreme weather, where temperatures could be very high during daytime but were very low at night. The HVAC start and stop conditions of this mode were as follow:

$$Heat\ start\ condition = low\_cutoff\_temperature - (swing + accuracy)\ [11]$$

$$Heat\ stop\ condition = low\_cuttoff\_temperature + accuracy\ [12]$$

$$Cool\ start\ conditi\square n = high\_cuttoff\_temperature + (swing + accuracy)\ [13]$$

$$Cool\ stop\ condition = high\_cutoff\_temperature - \ accuracy\ [14]$$

In this mode, when both rooms were empty, the thermostat switched to its energy-saving state by maintaining room temperatures at a larger range between the minimum and maximum temperatures, avoiding unnecessary HVAC starts and stops. Table 4-5 shows the detailed conditions of each HVAC operation.

Auto heat&cool mode when both rooms are empty									
Case	**Human room 1**	**Human room 2**	**Cutoff temperature Room 1**	**Current Temperature room 1 (T1)**	**Cutoff temperature Room 2**	**Current Temperature room 2 (T2)**	**Fan room 1**	**Fan room 2**	**Operation**
Heat	No	No	Minimum temperature	≤ cutoff	Minimum temperature	≤ cutoff	On	On	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off
Cool	No	No	Maximum temperature	≥ cutoff	Maximum temperature	≥ cutoff	On	On	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off

Table 4-5 Auto heat&cool mode when no human presence in room 1 and 2

On the other hand, the thermostat would maintain the comfort zone between low and high-temperature range if it detected human presence in at least one room. This mode was different from the two previous modes, since it did not use different cutoff values for a room with human presence and a room without human presence. This designed was made to help users stayed in their comfort zone longer while avoiding starting and stopping the HVAC multiple times. Nevertheless, the thermostat would only run the booster fan of the room with human presence. Table 4-6 shows the auto heat&cool mode with human presence in a room.

Auto heat&cool mode when either room 1 and room 2 has human presence									
Case	**Human room 1**	**Human room 2**	**Cutoff temperature Room 1**	**Current Temperature room 1 (T1)**	**Cutoff temperature Room 2**	**Current Temperature room 2 (T2)**	**Fan room 1**	**Fan room 2**	**Operation**
Heat	Yes	No	Desired temperature	≤ cutoff	Low temperature	≤ cutoff	On	Off	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off
Cool	Yes	No	Desired temperature	≥ cutoff	High temperature	≥ cutoff	On	Off	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off
Heat	No	Yes	Low temperature	≤ cutoff	Desired temperature	≤ cutoff	Off	On	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off
Cool	No	Yes	High temperature	≥ cutoff	Desired temperature	≥ cutoff	Off	On	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off

Table 4-6 Auto heat&cool mode when human was present in a room

When the smart thermostat detected human presence in both rooms, it would maintain the comfort zone similarly to the previous situation in Table 4-6, except that it would run the booster fans of both rooms. Table 4-8 shows how the smart thermostat handled the auto heat&cool mode in this situation:

Auto heat&cool mode when both rooms have human presence									
Case	**Human room 1**	**Human room 2**	**Cutoff temperature Room 1**	**Current Temperature room 1 (T1)**	**Cutoff temperature Room 2**	**Current Temperature room 2 (T2)**	**Fan room 1**	**Fan room 2**	**Operation**
Heat	Yes	Yes	Desired temperature	≤ cutoff	Desired temperature	≤ cutoff	On	On	On
				≤ cutoff		> cutoff			
				> cutoff		≤ cutoff			
				> cutoff		> cutoff	Off	Off	Off
Cool	Yes	Yes	Desired temperature	≥ cutoff	Desired temperature	≥ cutoff	On	On	On
				≥ cutoff		< cutoff			
				< cutoff		≥ cutoff			
				< cutoff		< cutoff	Off	Off	Off

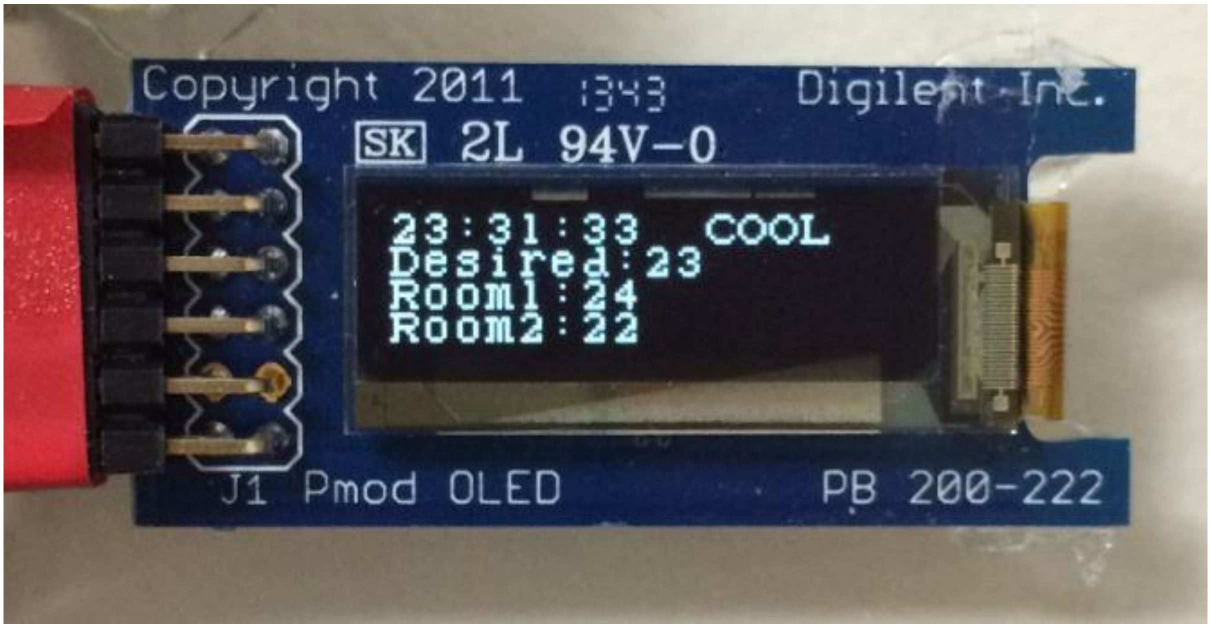
Table 4-7 Auto heat&cool mode when human appears in both room 1 and 2

4.2 User Interface

The user interface of this project’s thermostat was developed to demonstrate the capabilities of the design described in chapter 3 and section 4.1. Although there were still limitations to the user interface, the use of four push buttons and an OLED screen were sufficient to build a proof-of-concept prototype, allowing the thermostat to display information and receive user input for monitoring and debugging purposes.

4.2.1 Default screen: information display

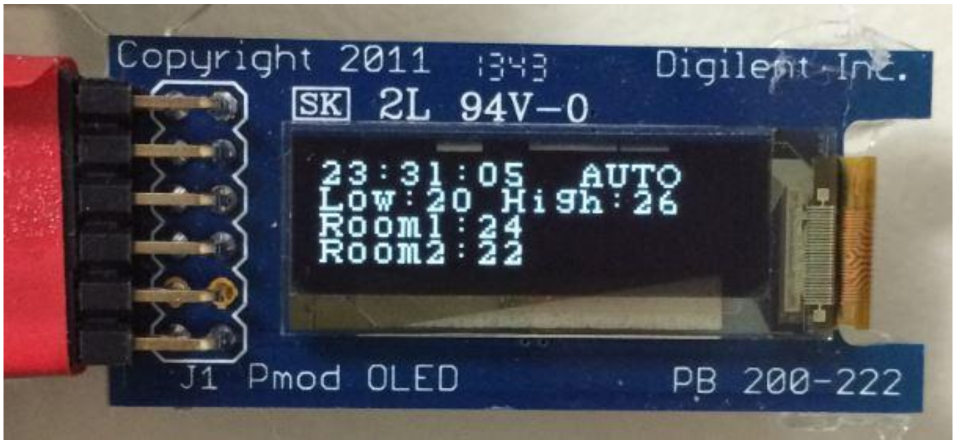
By default, the thermostat only output information relevant to the current status of HVAC operation. The first row displayed current time and the name of the mode. The second row displayed the desired temperature. The third and fourth row displayed current room 1’s and room 2’s temperatures. Figure 4-1 below shows a default display when the thermostat was in cool mode.



```
23:31:33  COOL
Desired:23
Room1:24
Room2:22
```

Figure 4-1 Standard information display on the OLED screen for a single mode.

When a user set auto heat&cool mode for the thermostat, the word “Auto”, a short title of the mode, was shown on the first row. Different from the other 2 modes, this mode showed the pre-set low and high temperatures since the rooms needed to be maintained within this temperatures range. Other information was displayed similarly to the other modes. Figure 4-5 below shows the default screen of the thermostat in auto heat&cool mode.



```
23:31:05  AUTO
Low:20 High:26
Room1:24
Room2:22
```

Figure 4-2 Standard information display on the OLED screen for auto heat&cool mode.

Since the OLED screen could only display up to 64 characters, with each row up to only 16 characters, the default screen only showed the most relevant information as described above. Other information, such as night operation or swing value, was not displayed but could be viewed when a user used four push-buttons, which details are to be discussed in 4.2.2. The temperature values were displayed in integer number instead of floating number to avoid confusion, and to follow a convention found in other thermostats available on the market.

4.2.2 User input

This thermostat needed only three input temperature values from users to operate in all three described modes, along with a few other configuration parameters. Users could set these values to the thermostat using four push-buttons. Figure 4-1 below displays how four push-buttons were laid out on Zybo.

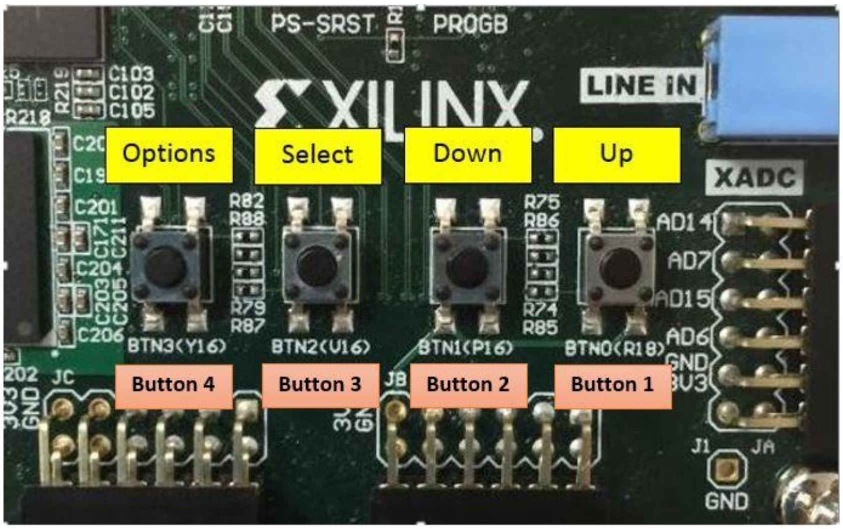


Figure 4-3 Four push-buttons and its functions.

The four push-buttons worked as follow:

- A user pressed button 4 (BTN3) to see the setting options.
- The user used button 1 (BTN0) to move up or button 2 (BTN1) to move down the option list shown in the orange blocks of Figure 4-2.
- The user pressed button 3 (BTN2) to select one of the setting options in the list.
- The user used button 1 or button 2 again to increase/decrease available values of the selected option, as shown in the purple blocks in Figure 4-2.
- The users pressed button 3 to save his choice.
- Alternatively, the user could pressed button 4 to cancel the current choice. After cancellation, the thermostat switched back to the default screen.

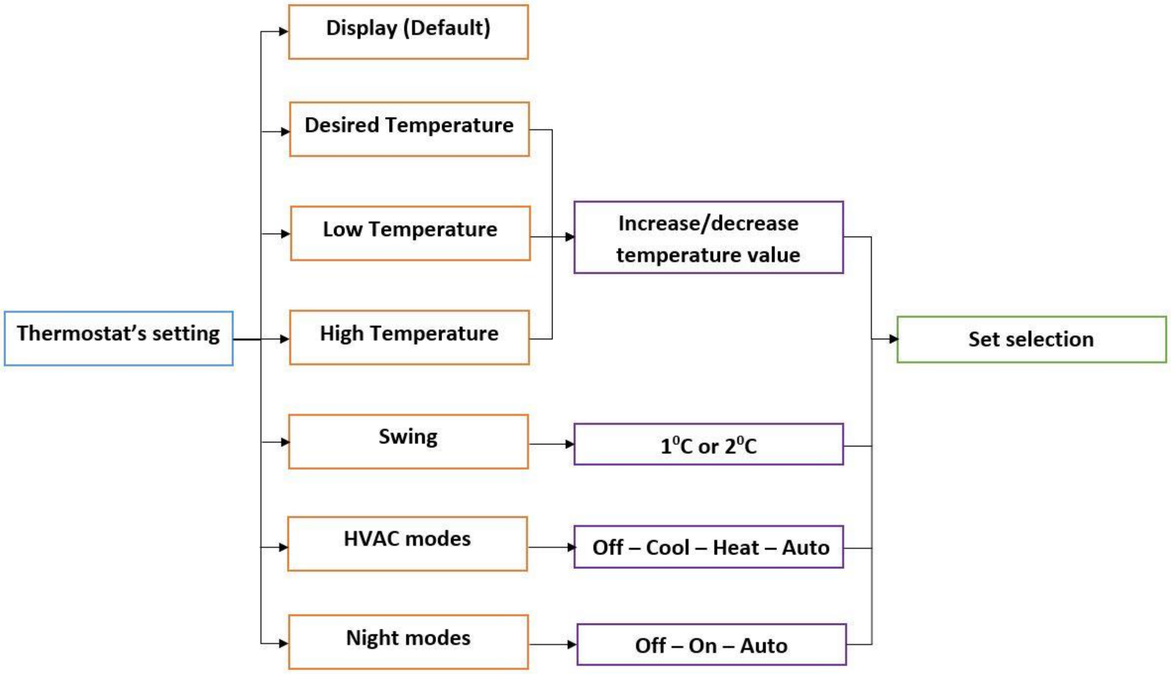
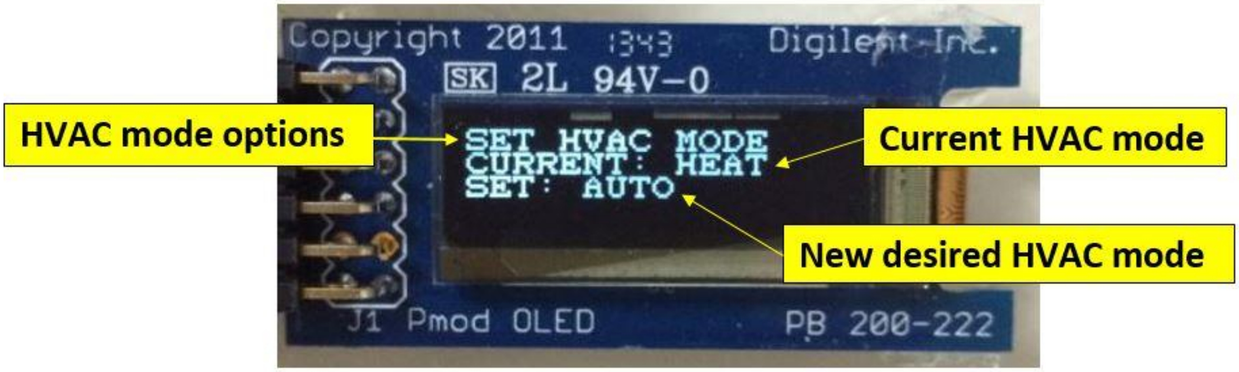


Figure 4-4 Thermostat's setting options

The OLED screen displayed information of setting options in real-time. Every command from the buttons would be shown on the screen immediately to show users the option they were setting.

Figure 4-3 below shows an example of the OLED screen when a user was selecting an HVAC mode. The first row displayed the current option, indicating that the user was choosing an HVAC mode. The second row displayed the current mode of the thermostat, in this case was "Heat". The third row displayed current user's selection, "Auto". The user could press button 1 or 2 to change his selection, button 3 to save, or button 4 to cancel the selection and go back to the default screen.



SET HVAC MODE
CURRENT: HEAT
SET: AUTO

Figure 4-5 OLED screen when a user was selecting HVAC mode.

4.3 Software program

To implement the design described in Chapter 3, a software program was written in C language using Xilinx SDK. The program started as the thermostat turned on, and ran continuously until the thermostat turned off. Below is a summary of the program:

```
1. While (true) {
2. D1, D2 = Data from room 1, data from room 2.
3. Human1, Temp1 = Human_detect(D1), Room_temperature(D1)
4. Human2, Temp2 = Human_detect(D2), Room_temperature(D2)
5. if (buttons are pressed)
6.   UInput = receive user input
7. Display_to_OLED(Human1, Temp1, Human2, Temp2, UInput)
8. Control_HVAC(Human1, Temp1, Human2, Temp2, UInput)
9. }
```

The program was indeed an infinite loop that constantly received, processed data and made HVAC control decisions. At line 2, the program accessed data captured by the sensor modules by reading from their specified memory addresses. It processed the data, stored human detection results in Human1, Human2, and current room temperatures in Temp1 and Temp2 (line 3 and 4). If any of the push buttons were pressed, it would receive user input and store in UInput (line 5 and 6). It then displayed relevant data to the OLED screen (line 7). After that the data was used to control the HVAC system, according to the specifications in 3.13 and 4.1 (line 8).

4.4 Cost

In this project, a thermostat prototype was built to test two rooms in an apartments, with all components and peripherals described in pervious chapters. Overall the cost for the complete 2-room prototype was USD$404. Table 4-8 shows the breakdown cost of all components of the prototype.

Products	Unit price	Quantity	Cost
Zybo	$129	1	$129
Digilent temperature sensor TMP2	$24.99	2	$49.98
Omron thermal sensor D6T-44L-06	$49.99	2	$99.98
4-channel relay	$8	2	$16
Register booster fan	$25	2	$50
Connection wires	$38.92	1000ft/case	$38.92
I^{2}C bus extender	$1.89	8	$15.12
Translating I^{2}C bus repeater	$2.7	2	$5.4
Total			$404.4

Table 4-8 Smart thermostat prototype cost.

Hardware for each additional room will cost around $106 shown in Table 4-9.

Products	Unit price	Quantity	Cost
Digilent temperature sensor TMP2	$24.99	1	$24.99
Omron thermal sensor D6T-44L-06	$49.99	1	$49.99
Register booster fan	$25	1	$25
I^{2}C bus extender	$1.89	2	$3.78
Translating I^{2}C bus repeater	$2.7	1	$2.7
Total			$106

Table 4-9 Hardware cost for each additional room

Overall, this chapter explained how hardware and software components were developed on Zybo and its peripherals to become a smart thermostat. A software program was developed to use aforementioned algorithms to control the HVAC while receiving user input and displaying information in real time. Additionally, the prototype cost was quoted and analyzed, hinting potential rooms for commercialization. Chapter 5 will describe a testing process conducted to verify the thermostat’s functionalities as well as effectiveness. It also show collected test results and analysis.

5. Testing and Results

This chapter describes in details how the smart thermostat was tested using several methods at different stages of the design process. An oscilloscope was used primarily to verify data bus of the hardware components, including I^{2}C interfaces of the sensors and an SPI interface of the OLED screen. Software debugging and testing were done through Xilinx SDK console as well as log data written to a microSD memory card.

5.1 Hardware testing

During the development process of the smart thermostat, the sensors, push-buttons, the relay and the OLED screen modules were tested individually before integration with the top-level system design. Each module was tested as a standalone unit through the use of an oscilloscope. The oscilloscope not only helped in testing the functionality of the digital circuit but also helped in evaluating the accuracy of input signals from sensors and output signals to the OLED screen.

5.1.1 Oscilloscope

The oscilloscope was first used to verify I^{2}C signals between Zybo and the sensors. After the hardware setup and the prototype interface circuit of each sensor had been completed, an Oscilloscope with a programmable logic analyzer was used to verify device's address signed by a master device and I^{2}C buses on SCL and SDA line. Figure 5-1 shows an oscilloscope with programmable logic analyzer used to verify serial bus signals.

Figure 5-1 An oscilloscope with programmable logic analyzer [57]

The logic analyzer of the oscilloscope was used to determine logic ones and zeroes. After connecting the analyzer cable to the connection between Zybo and a sensor shown in Figure 5-1, the oscilloscope was turned on to capture data being transferred across the bus. Logic ones and zeros of SCL and SDA shown on the oscilloscope screen reflected the current I^2C bus signals transmitted between Zybo and the sensor.

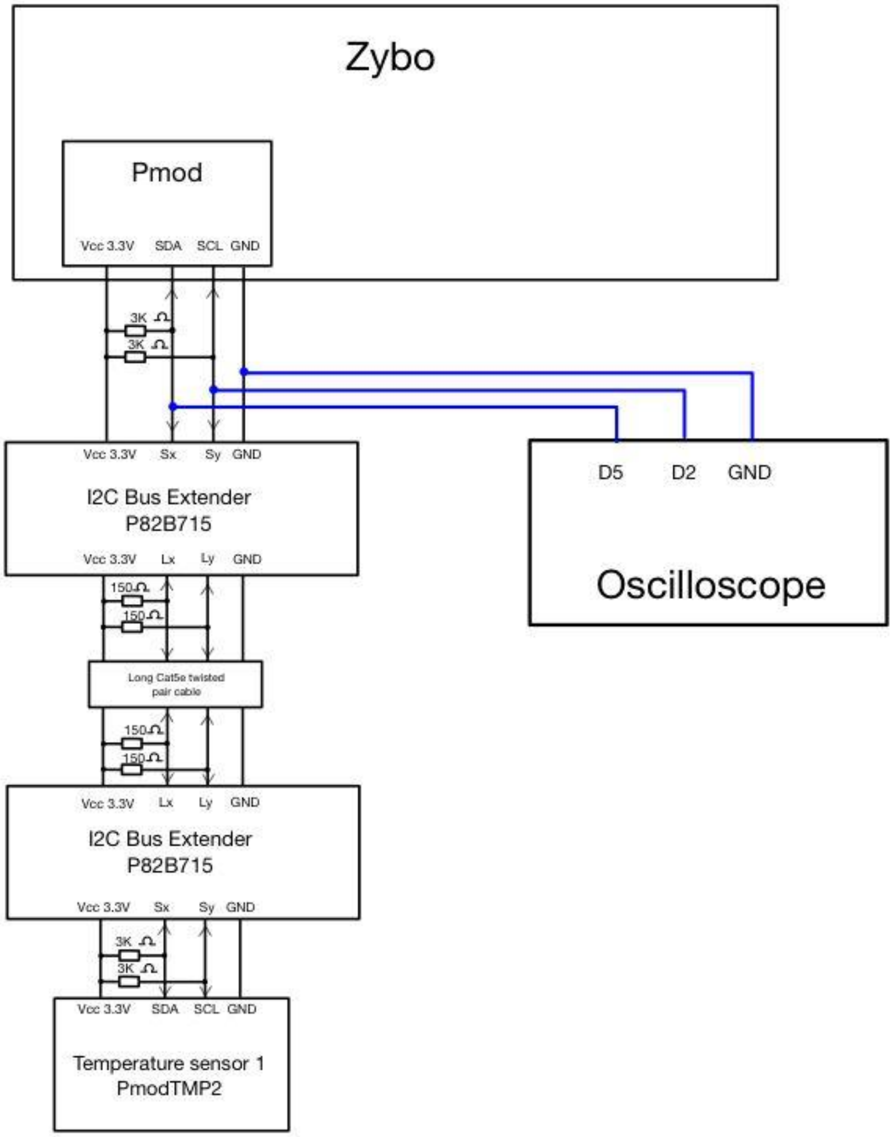
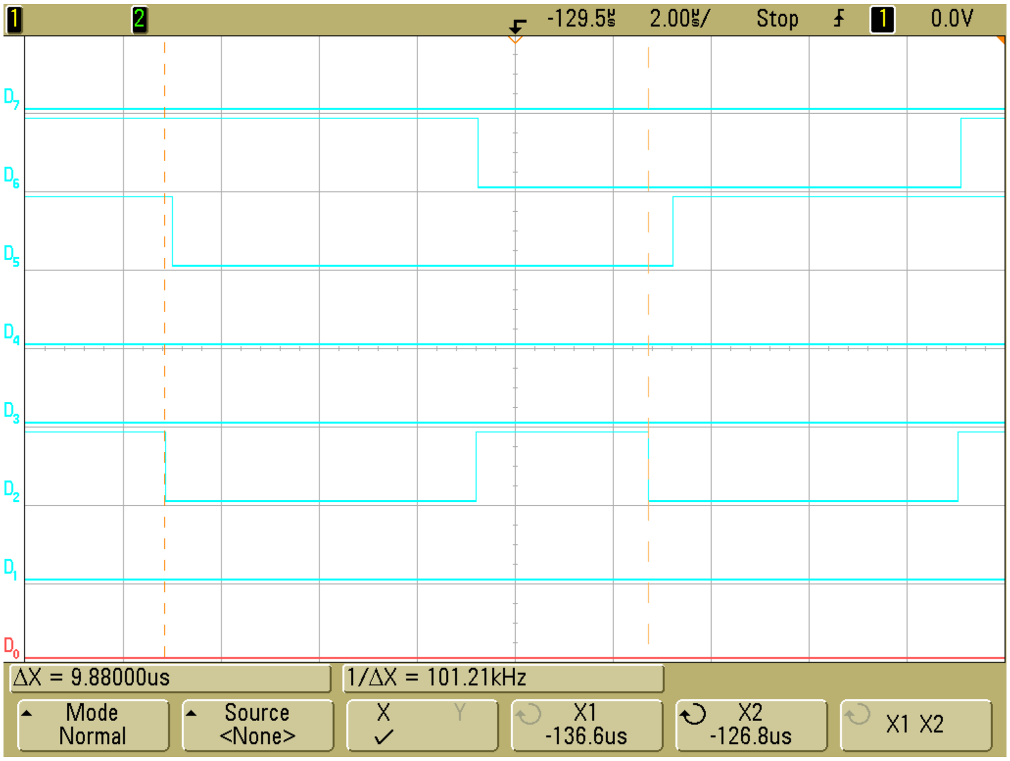


Figure 5-2 Hardware testing using an oscilloscope



-129.5us 2.00us/ Stop 1 0.0V

ΔX = 9.88000us

1/ΔX = 101.21kHz

Mode Normal

Source <None>

X ✓

X1 -136.6us

X2 -126.8us

Figure 5-3 Using an oscilloscope to measure the frequency on SCL of a temperature sensor

Figure 5-3 shows the SCL (channel D2) and SDA (channel D5) bus signals of the communication between Zybo and a temperature sensor in one SCL clock cycle. In this figure, a pair of triggers was used to measure the frequency of SCL, which turned out to be 101.21 kHz. The clock speed of the SCL satisfied the requirement from the manufacturer. Figure 5-4 below shows the logic bytes of SCL (channel D2) and SDA (channel D5). The SDA line shown on the oscilloscope screen were **0000 1110 0110 1000**, equivalent to 28.81^{0}C. Although the result displayed on the oscilloscope was 16 bits (2 bytes), this project only used 13-bit resolution (bit 15-3) of the temperature sensor for conversion.

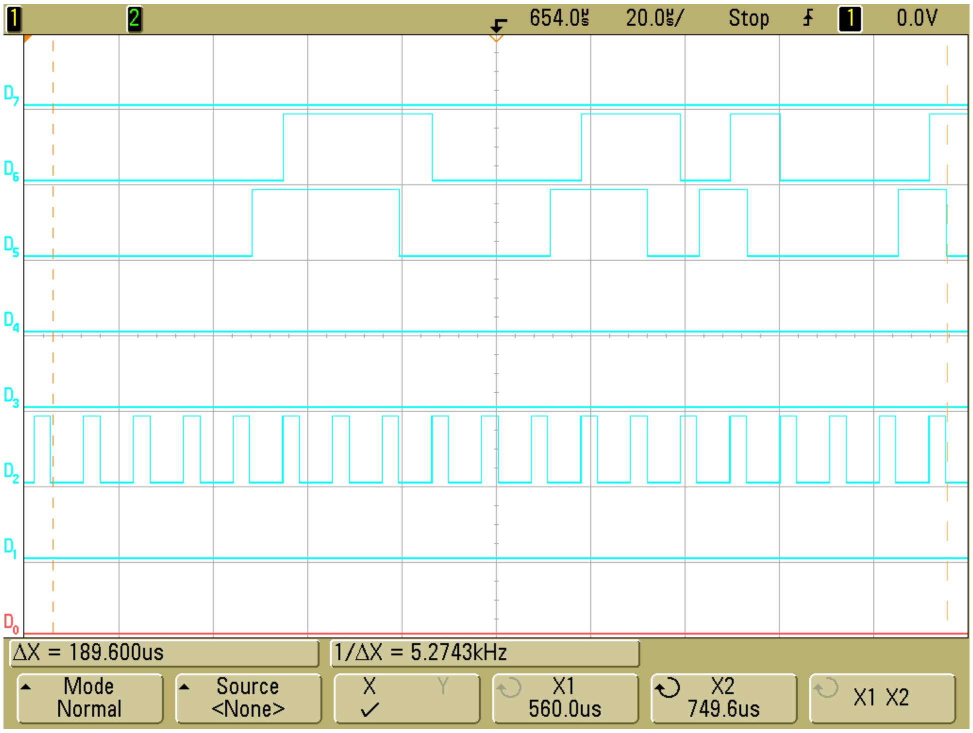


Figure 5-4 The oscilloscope showed I^2C bus signals of the temperature sensor

5.1.2 Resource utilization

Zybo development board contains a Xilinx Zynq chipset that features an Artix-7 FPGA with 17,600 LUTs and 35,200 flip-flops for generating complex digital logics. Table 5-1 below shows the system resource utilization of the post-implementation from the Vivado software. The system occupied less than 21% of all Look-up Tables and 14% of Flip Flop state memory devices. This information represented the logical efficiency implemented in the design of a complex system. The rest of unused logic cells could be used in future project development.

Resource	**Utilization**	**Available**	**Utilization %**
FF	5089	35200	14.46
LUT	3577	17600	20.32
Memory LUT	66	6000	1.10
I/O	32	100	32.00
BRAM	0.5	60	0.83
BUFG	2	32	6.25

Table 5-1 System resource utilization

5.2 System setup

All sensors, register booster fans and the HVAC system were connected to Zybo before testing. Figure 5-5 shows how all peripherals were connected together. Each room had a pair of temperature and thermal sensors. In each room, a combined booster fan was also attached to the room’s register vent on the ceiling, as explained in Chapter 2.7. One 4-channel relay was used to control the HVAC system and another 4-channel relay was used to control the register booster fans. The black arrows in Figure 5-5 represent the wires connecting between Zybo and its peripherals.

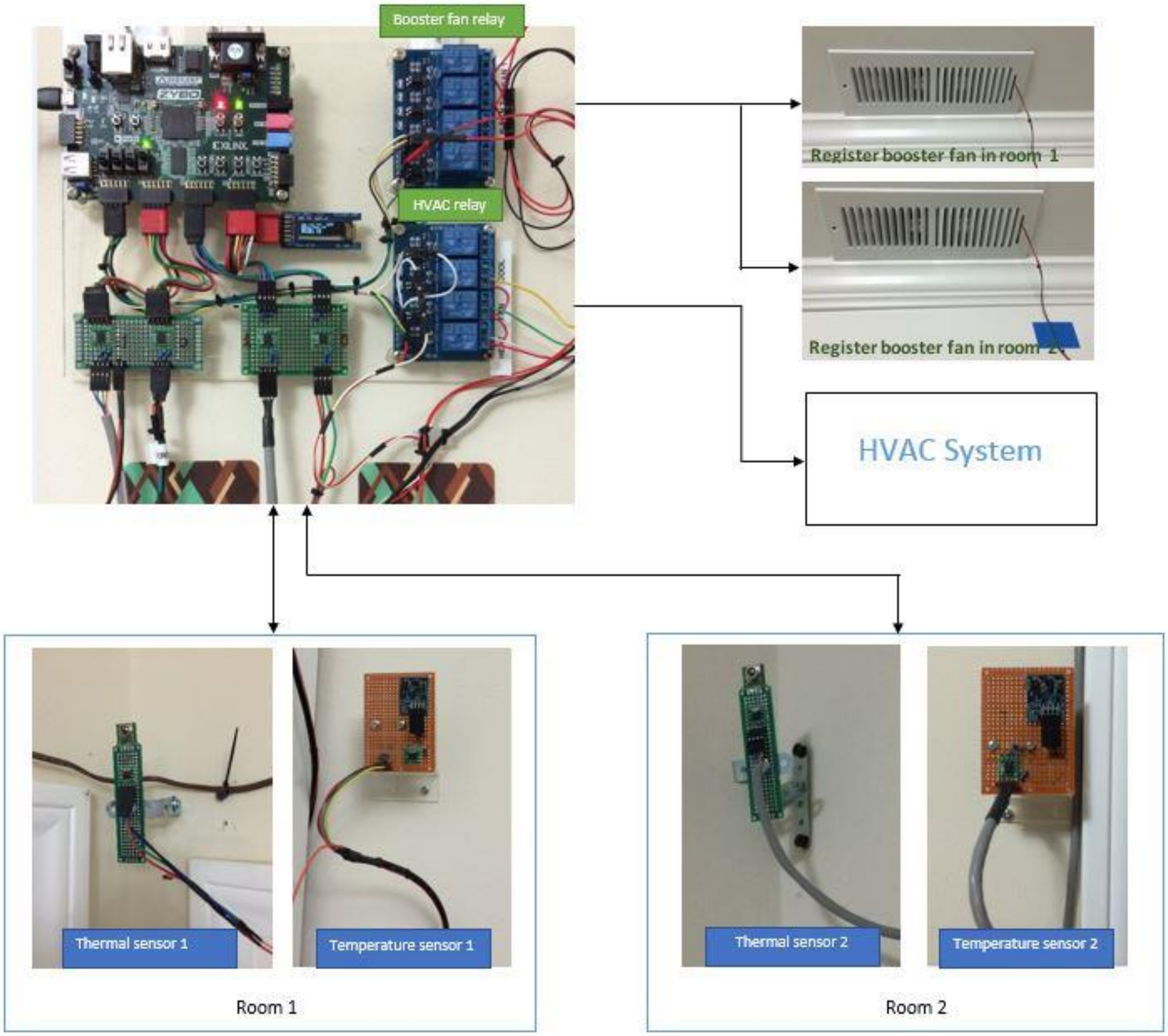


Figure 5-5 Real system prototype used to perform the HVAC system.

5.2.1 Test environment

A test environment was set up in two rooms of a residential apartment on the first floor of a 3-floor building. The first room (room 1) was an open living and dining area connected to a kitchen. This room was where most human activities took place during testing. The second room (room 2) was an adjacent bedroom, located wall-to-wall with room 1. The room usually had human presence only during nighttime. The two rooms were connected by a door. Figure 5-5 below shows the apartment floorplan and the location where the original basic thermostat was placed. All furniture was placed at the same position during the experiment.

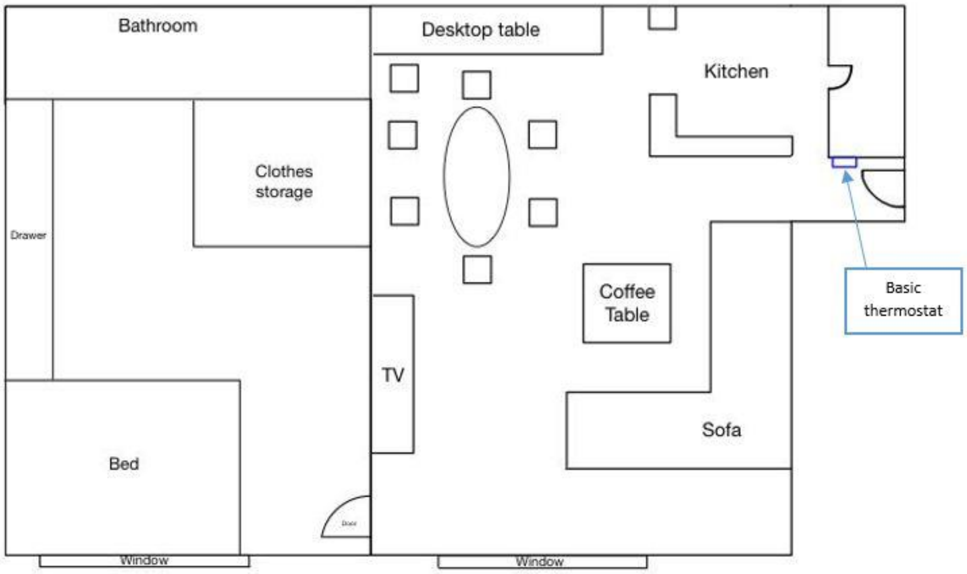


Figure 5-6 Floorplan and the thermostat's location

5.2.2 Equipment setup

The Omron thermal sensor had its own field of view (FOV) specifications in direction X (vertical) and direction Y (horizontal), which maxima were 44.2^0 and 45.7^0 respectively. Figure 5-7 shows the direction of the thermal sensor and specifications of the FOV.

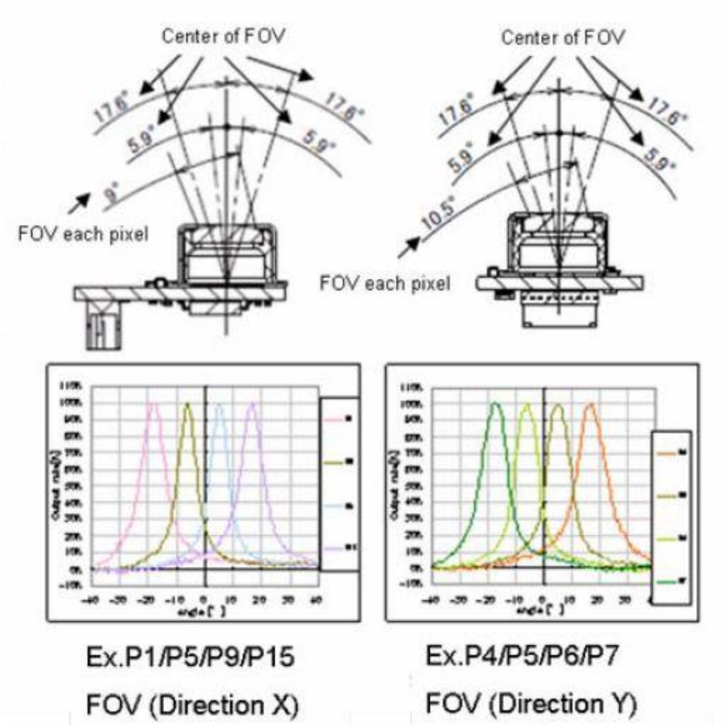


Figure 5-6 Field of View (FOV) in direct X and direction Y of the Omron thermal sensor

Figure 5-7 shows the equipment setup in room 1. In this figure, the blind spots and the detecting area was sketched based on the horizontal FOV of the thermal sensor. The left figure shows where the smart thermostat, temperature sensor 1 and thermal sensor 1 were placed. It also shows the blind and visible areas room 1's thermal sensor. The right figure shows the numbered rectangles indicating different spots where human residents usually occupied. Note that position 10 and 13 were still in the visible area of the thermal sensor. Positions 1 to 6 were the seats of a dining table. Positions 7 to 12 were sitting areas on the sofa and near the coffee table. Position 13 and 14 were used to verify the visible area of the thermal sensor.

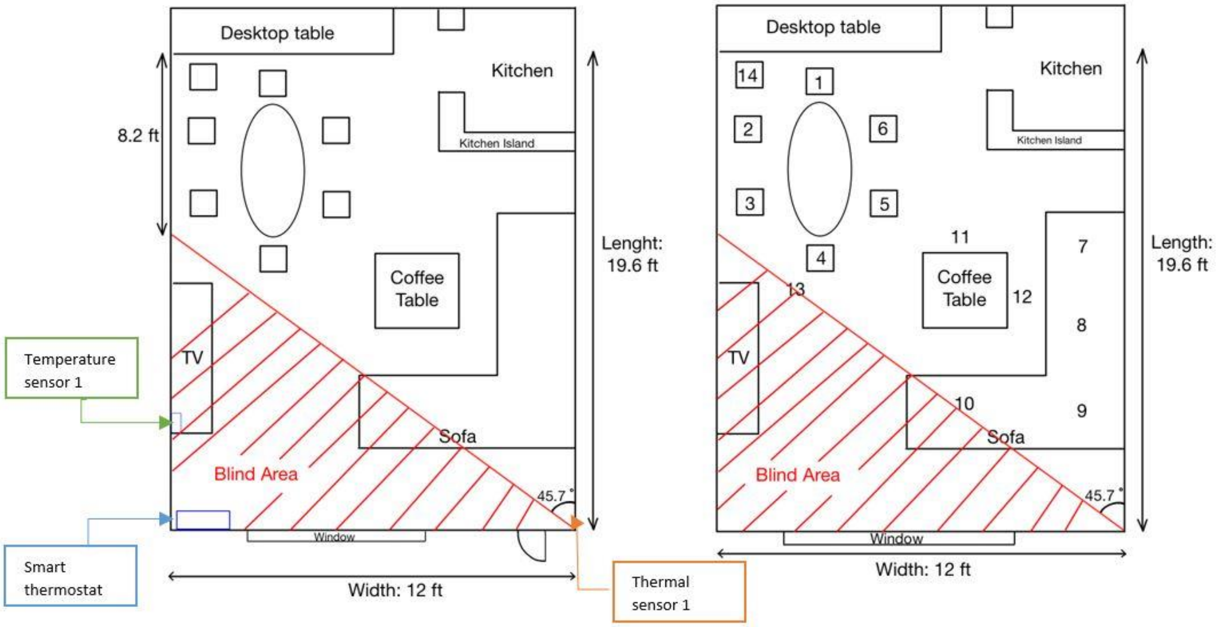


Figure 5-7 Testing environment in direction Y of the thermal sensor in room 1

The thermal sensor was placed at the corner of room 1 shown in Figure 5-7 to get achieve the largest visible area. At this position, the thermal sensor was able to view all the numbered positions, thus potentially detected human presence. Note that usually there was no human occupation in the blind area marked in red, as that area was just in front of a TV, and near a door. Figure 5-8 below shows a picture of room 1 captured by a regular camera placed precisely at the thermal sensor location.

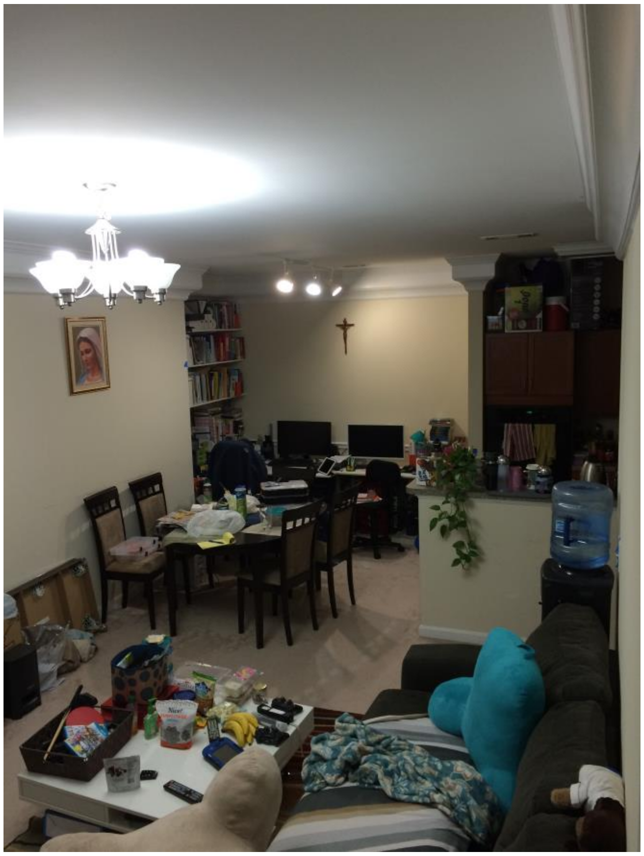


Figure 5-8 Real testing environment of room 1

The thermal sensor was placed at 7.1ft high, and tilted to a 33^0 angle from the wall to achieve the largest visible area, according to its maximum vertical FOV. If the thermal sensor was placed at a higher position, position 9 and 10 in Figure 5-7 would fall into the blind area. At this position, the thermal sensor could detect objects of maximum 3.3ft high at the furthest position. Figure 5-9 below shows the described calculations in more details.

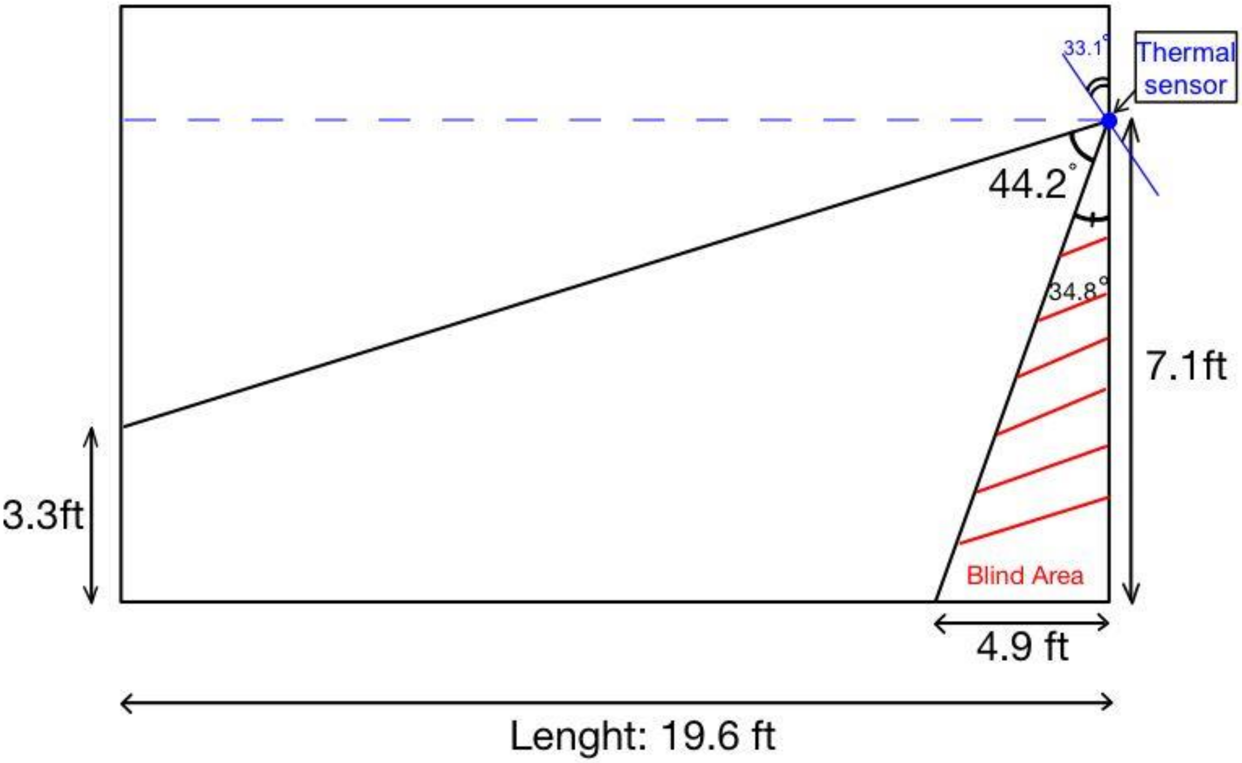


Figure 5-9 Testing environment in direction X of the thermal sensor in room 1

Similarly, Figure 5-10 shows the equipment setup in room 2 with the specific position of thermal sensor to achieve the largest visible area in the room. The temperature sensor was placed near the door as shown in Figure 5-10, far away from the window to avoid outdoor temperature influence. The right figure showed the numbered rectangles indicating different spots where the residents usually occupied. The thermal sensor was placed at the corner of room 2 shown the left figure, to ensure that all human occupied spots were covered. Room 2 had a king-size bed marked at position 1 and 2. Position 3 and 4 were where at least a human usually sit. A portable heater was located at position 6. Figure 5-11 shows a picture of room 2 captured by a regular camera placed at the thermal sensor's position.

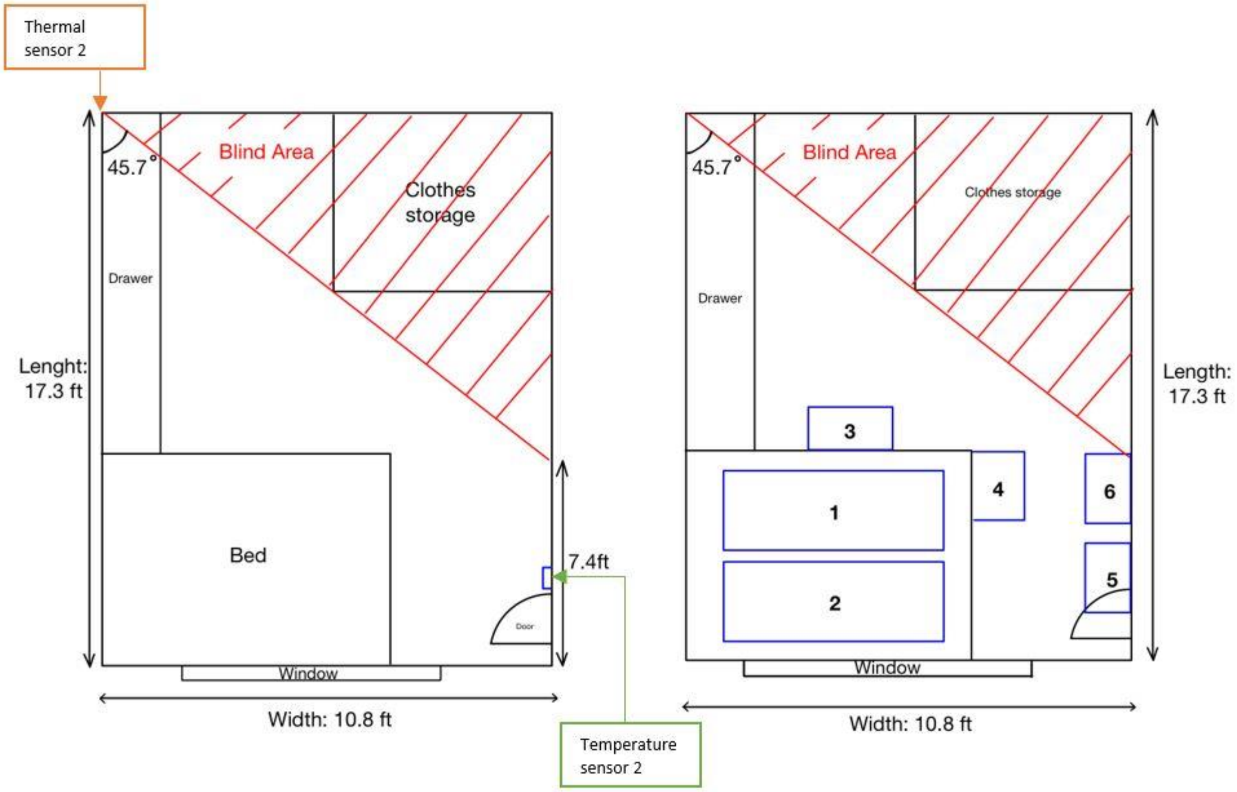


Figure 5-10 Testing environment in room 2

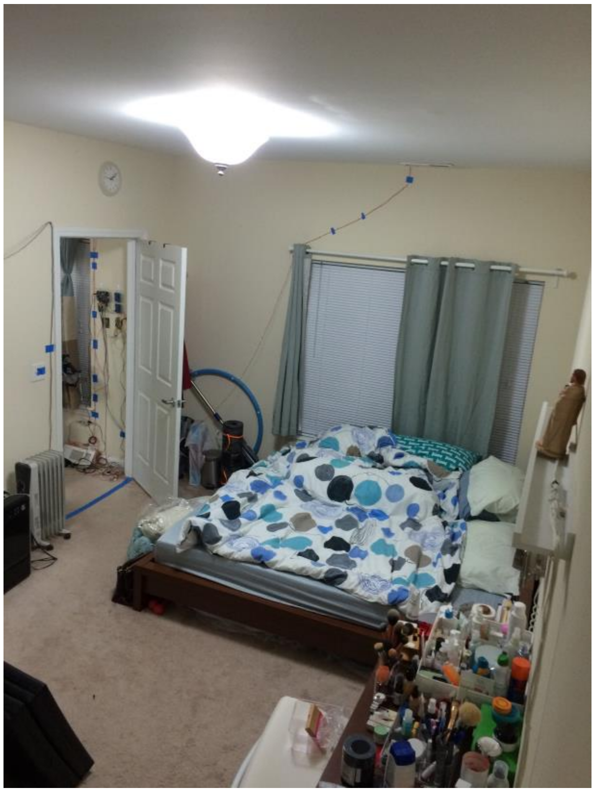


Figure 5-11 Real testing environment in room 2

In room 2, the thermal sensor was placed at a position higher than different from the one in room 1 in order to detect object of 3.3ft high at the furthest position. Based on the calculations shown in Figure 5-12, the blind area was only within 3.7ft below the sensor. This area was in right in front of a bathroom door, hence there was usually no human occupation.

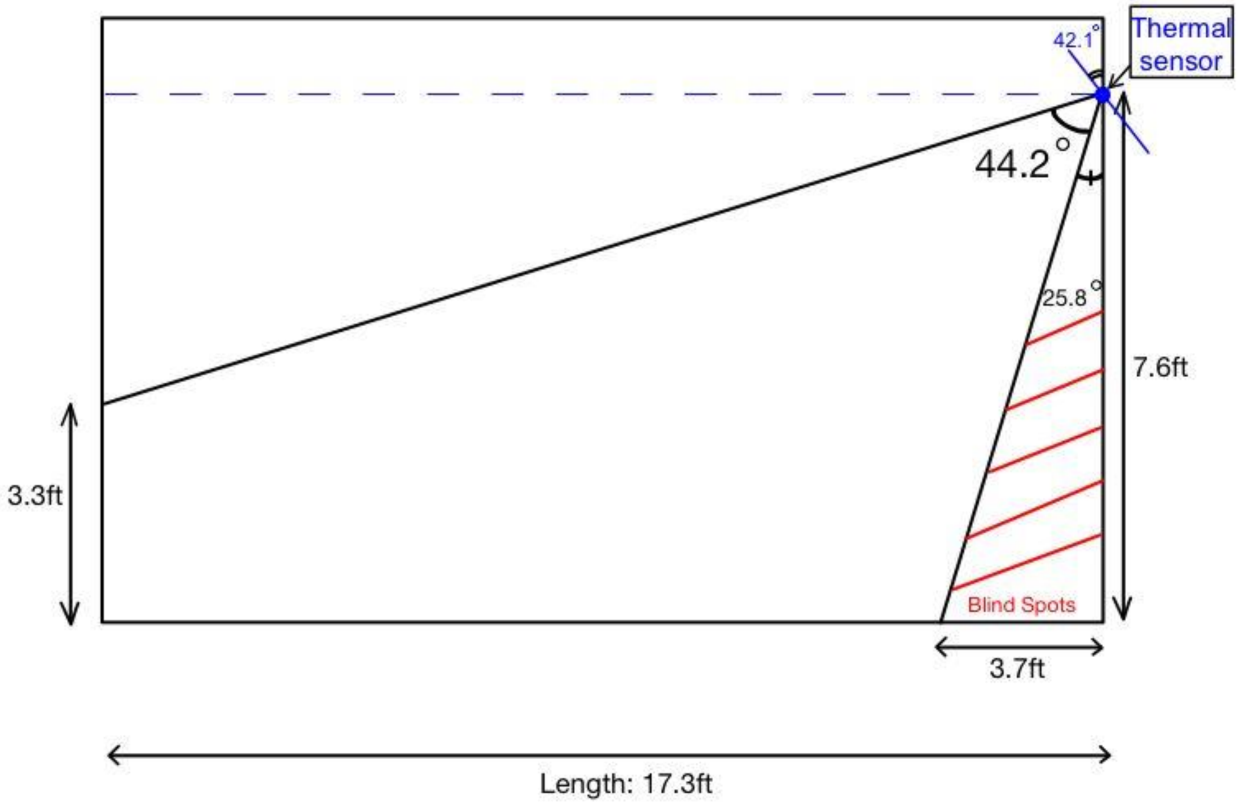


Figure 5-7 Position of thermal sensor and its detecting dimension in room 2

The purpose of this equipment setup in two rooms was to capture and analyze data from the thermal sensors to build a human detection algorithm. After the algorithm had been built, the same setup was also used to verify the algorithm's accuracy rate and to conduct functional testing. Results of the testing are going to be mentioned in section 5.5.

5.2.2 Data collection

Log data was collected in real time as the thermostat was running during sample collection and testing. In particular, the software program printed out log messages at runtime. The messages were stored in files saved to a microSD memory card. They were used for analysis in section 5.5, and consisted of the following information:

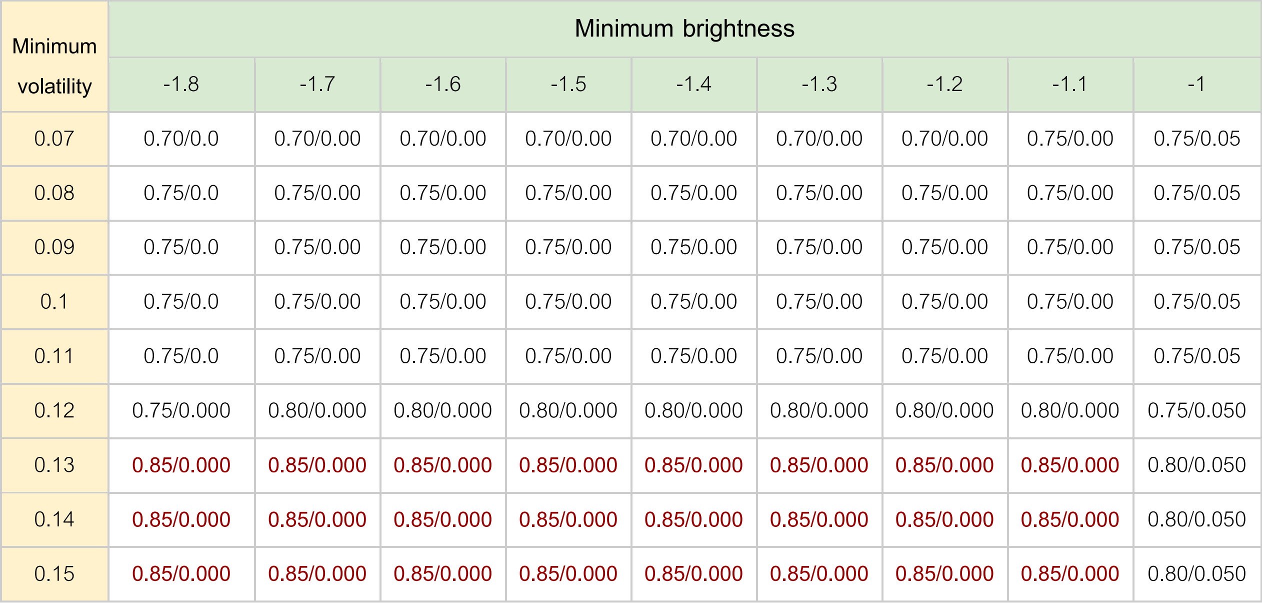
- Values from thermal sensors: logged from both rooms every second, only during sample collection.
- Values from temperature sensors: logged from both rooms every second during sample collection and every 60 seconds during testing.
- Mean brightness and brightness volatility: logged from both rooms every 60 seconds, only during testing.

- Human detection algorithm results: logged as soon as there was a new result.
- HVAC start/stop: logged as soon as a start/stop command was issued by the thermostat.
- Current user input: logged at the moment of HVAC start/stop.

5.3 Tests and Results

5.3.1 Human detection algorithm

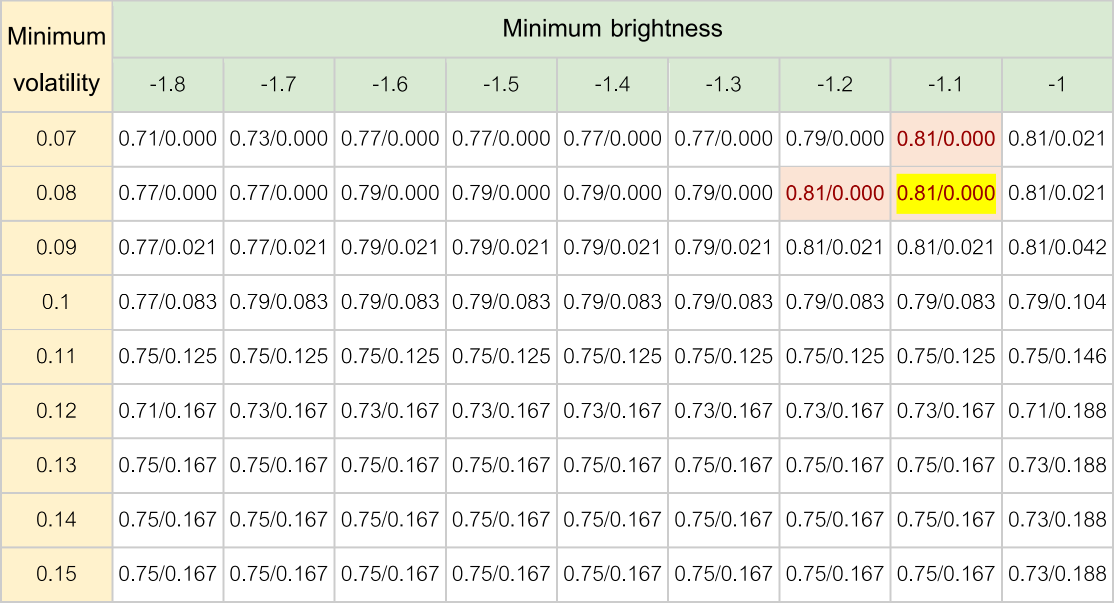
To test the human detection algorithm, a set of twenty 5-minute tests were taken, including both human and non-human presence test cases. In tests where at least a human were present, the person could either stayed at a location as marked in Figure 5-8 for room 1 and Figure 5-11 for room 2 or moved around the room. As in section 3.12.3.2.3, data from the tests were run through the detection algorithm using different combinations of mean brightness and brightness volatility. Table 5-2 below shows results of these runs, with the best results marked in red.



Minimum volatility	Minimum brightness								
	-1.8	-1.7	-1.6	-1.5	-1.4	-1.3	-1.2	-1.1	-1
0.07	0.70/0.0	0.70/0.00	0.70/0.00	0.70/0.00	0.70/0.00	0.70/0.00	0.70/0.00	0.75/0.00	0.75/0.05
0.08	0.75/0.0	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.05
0.09	0.75/0.0	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.05
0.1	0.75/0.0	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.05
0.11	0.75/0.0	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.00	0.75/0.05
0.12	0.75/0.000	0.80/0.000	0.80/0.000	0.80/0.000	0.80/0.000	0.80/0.000	0.80/0.000	0.80/0.000	0.75/0.050
0.13	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.80/0.050
0.14	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.80/0.050
0.15	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.85/0.000	0.80/0.050

Table 5-2 Different combinations of mean brightness and brightness volatility

As seen, the lowest accuracy rate was 70% with no false negative results. Moreover, volatility values larger than 0.13 and mean brightness values smaller than -1.1 yielded an accuracy rate of 85% with no false negative either. Table 5-3 below shows the combined results when the algorithm was run on 28 training samples and 20 tests. Highest accuracy rates were marked in red.



Minimum volatility	Minimum brightness								
	-1.8	-1.7	-1.6	-1.5	-1.4	-1.3	-1.2	-1.1	-1
0.07	0.71/0.000	0.73/0.000	0.77/0.000	0.77/0.000	0.77/0.000	0.77/0.000	0.79/0.000	0.81/0.000	0.81/0.021
0.08	0.77/0.000	0.77/0.000	0.79/0.000	0.79/0.000	0.79/0.000	0.79/0.000	0.81/0.000	0.81/0.000	0.81/0.021
0.09	0.77/0.021	0.77/0.021	0.79/0.021	0.79/0.021	0.79/0.021	0.79/0.021	0.81/0.021	0.81/0.021	0.81/0.042
0.1	0.77/0.083	0.79/0.083	0.79/0.083	0.79/0.083	0.79/0.083	0.79/0.083	0.79/0.083	0.79/0.083	0.79/0.104
0.11	0.75/0.125	0.75/0.125	0.75/0.125	0.75/0.125	0.75/0.125	0.75/0.125	0.75/0.125	0.75/0.125	0.75/0.146
0.12	0.71/0.167	0.73/0.167	0.73/0.167	0.73/0.167	0.73/0.167	0.73/0.167	0.73/0.167	0.73/0.167	0.71/0.188
0.13	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.73/0.188
0.14	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.73/0.188
0.15	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.75/0.167	0.73/0.188

Table 5-3 Combinational result of human detection algorithm after running 28 samples and 20 tests.

As discussed in 3.12.3.2.3, the mean brightness and volatility value pair served as cutoff values to distinguish a human pixel from a non-human pixel. The values of these parameters had to be large enough to yield the highest accuracy rate, but at the mean time they could not be too large that could yield false negatives. In the above table, among the parameter pairs with the best rates, the pair of mean brightness = -1.1 and volatility = 0.08 was chosen for use in functional testing, as they were the maximum values that gave the highest accuracy rate of 81% and no false negative results.

5.3.2 Functional testing

With the minimum brightness and minimum volatility values found in the previous section, I conducted four tests to verify functionalities of the thermostat design, investigate the differences in applying smart algorithms to control the HVAC, and examine the role of booster fans in bringing indoor temperature to the desired level:

a) *Smart operation with booster fan*: The thermostat used the human detection algorithm to control the HVAC automatically. It turned on booster fan as necessary. Night mode automatically switched on at 12:00 AM and switched off at 8:00 AM.

b) *Smart operation without booster fan*: This test was similar to (a), except that no booster fan was used.

c) *Non-smart operation*: The thermostat controlled the HVAC based only on data taken from room 1’s temperature sensor. This test mimicked functionalities of the original thermostat.

d) *Original thermostat operation*: The original thermostat of the apartment was used instead of this project’s thermostat. This test served as a baseline to compare with other tests.

The tests had a few common configurations: the thermostats were run in heat mode only, with starting indoor temperatures of around 19^0C. Each test lasted around 23 hours, from

12:00 AM to around 11 PM. In (a) and (b), data from thermal sensors and temperature sensors were recorded. In (c), only data from temperature sensors were recorded. Times of HVAC starts and stops were also logged along with room temperatures. The desired temperature was set at 24^{0}C, the low temperature was set at 19^{0}C and the high temperature was set at 26^{0}C. Below are figures and tables of results collected from the tests.

5.3.2.1 Smart operation with booster fan

Figure 5-7 below shows log data collected from the first test mentioned above. The top left graph showed data of room 1, the top right graph showed data from room 2, and the bottom left graph showed both rooms' temperatures for easy comparison. The HVAC started two times, one in the morning and one in the evening. Room 1's temperature was always higher than room 2's temperature since residents usually stay in the living area during the day, and the window in room 2 was not tight even when completely closed, letting cold air to leak in and bring the room temperature down. Note that in room 2, there were times when the room temperature fell out of comfort zone, but the HVAC was not started since the room was empty. Table 5-4 represents the runs' statistics taken from the log data.

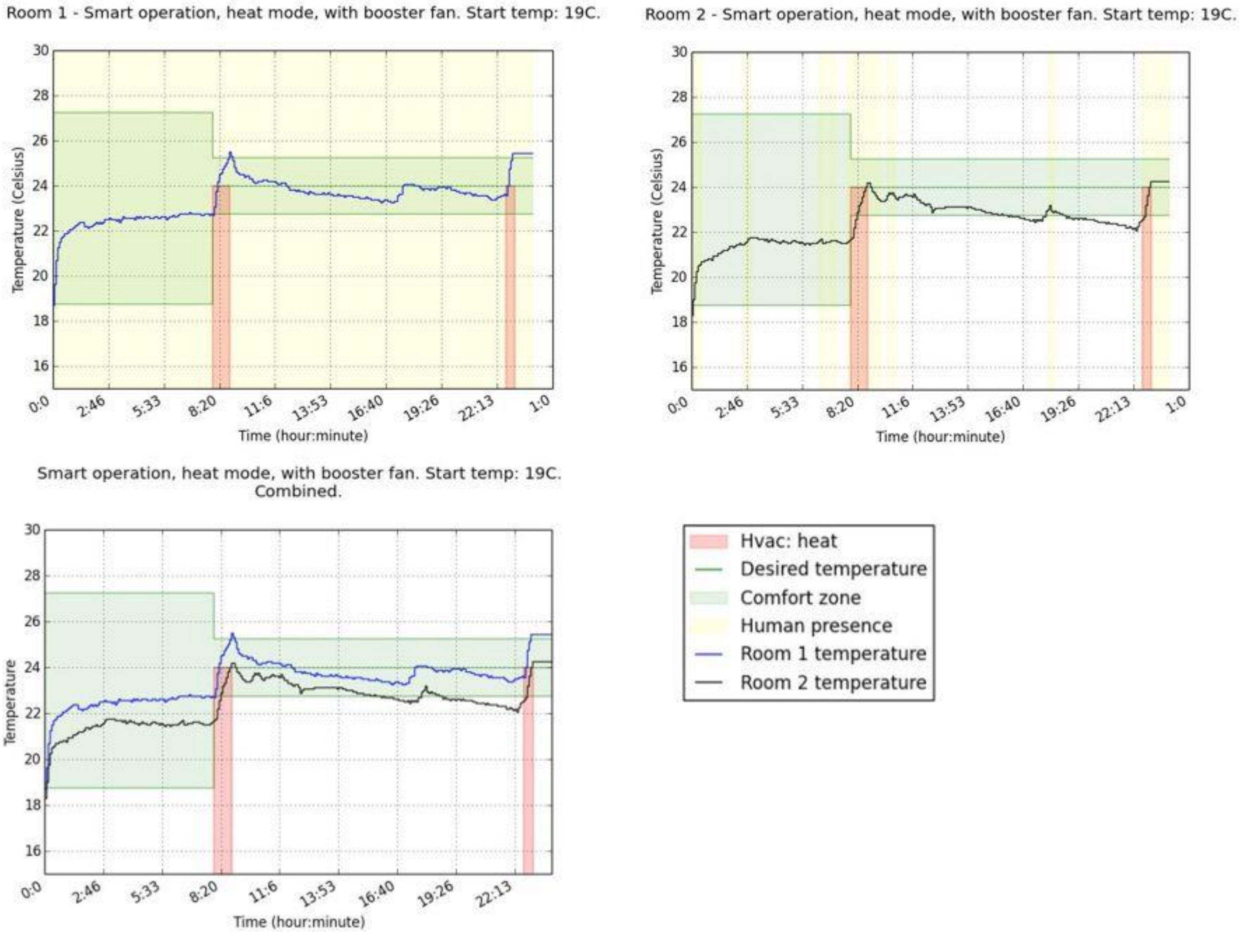


Figure 5-8 Smart operation with booster fans

Smart operation with booster fans Starting temperature at 19^0C		
Run number	1	2
Starting temp R1	22.688	23.562
Ending temp R1	25.5	25.438
Starting temp R2	21.688	22.562
Ending temp R2	24.312	24.312
Start time	8:00:00	22:40:00
End time	8:50:59	23:06:23
Total runtime	0:50:59	0:26:23
R1 temp change	2.812	1.876
R2 temp change	2.624	1.75
R1 rate(min/degree)	0:18:08	0:14:04
R2 rate(min/degree)	0:19:26	0:15:05

Table 5-4 Statistics of HVAC runs in smart operation without booster fans

5.3.2.2 Smart operation without booster fan

This test was similar to the previous test, except that there was no booster fan turned on. The test result was also similar to the test of smart operation with booster fan, where the HVAC was turned on only if a room temperature fell below the comfort zone and the room had human presence. Figure 5-8 and Table 5-5 below summarize results and log data from the test. The last two rows in the table show the rooms' temperature change rates during the runs. These rates give estimation of how many minutes it took a room to increase temperature by one Celsius degree.

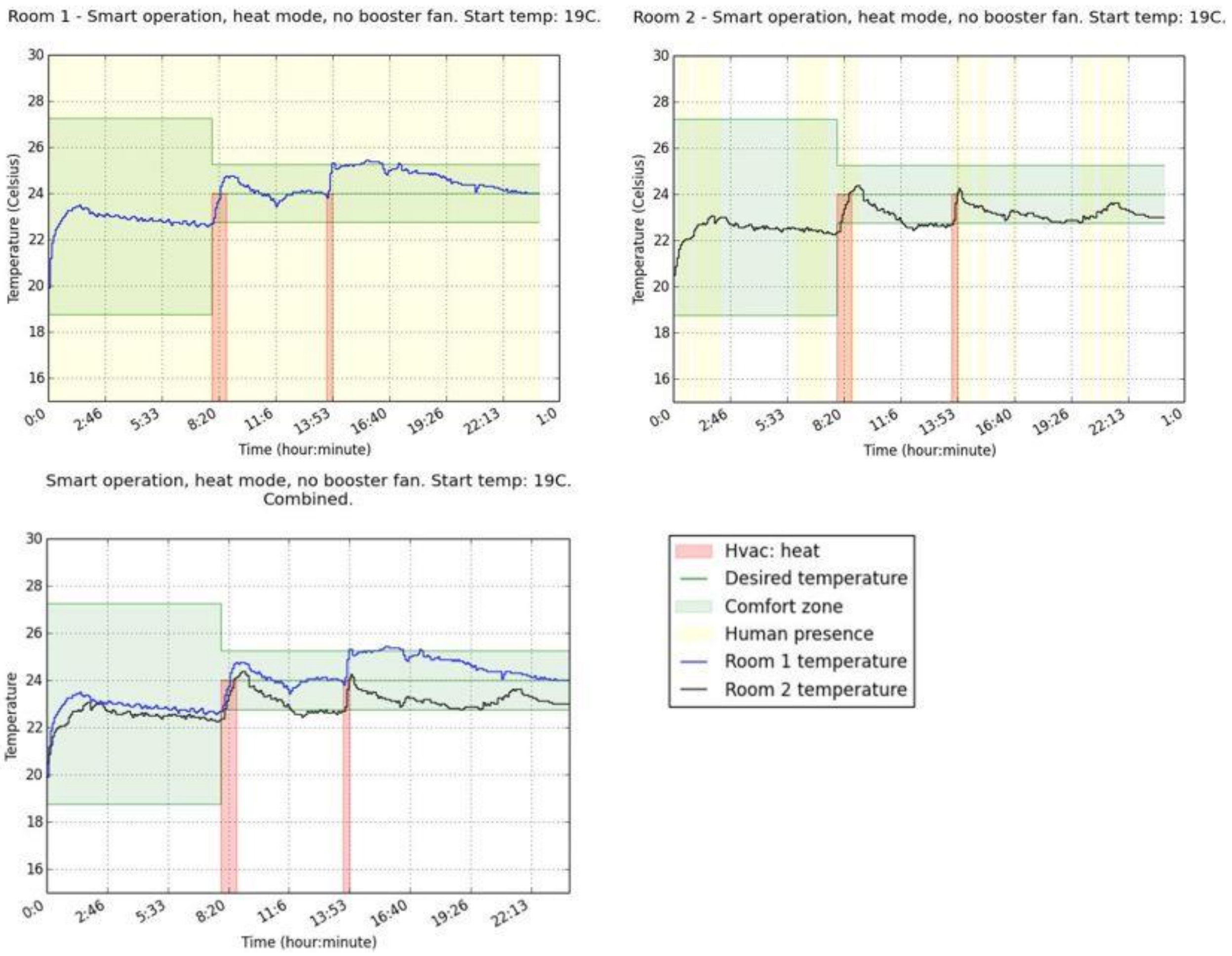
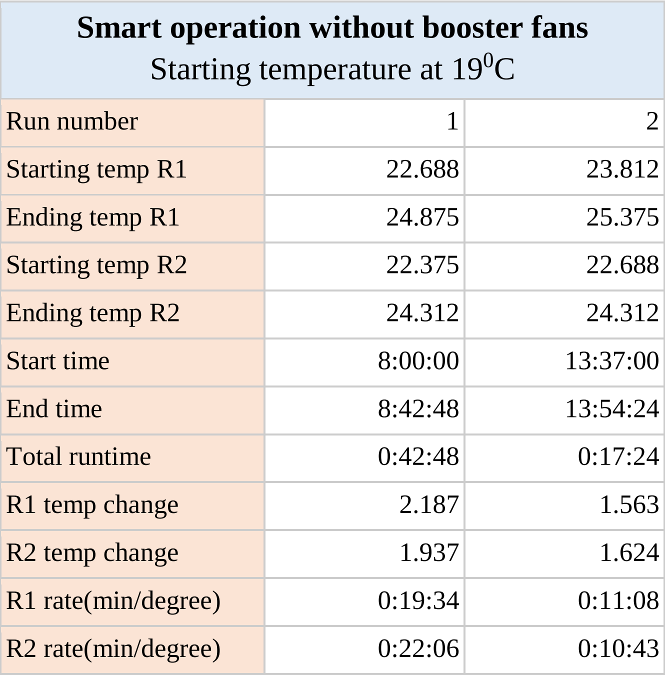


Figure 5-9 Smart operation without booster fans



Smart operation without booster fans Starting temperature at 19^0C		
Run number	1	2
Starting temp R1	22.688	23.812
Ending temp R1	24.875	25.375
Starting temp R2	22.375	22.688
Ending temp R2	24.312	24.312
Start time	8:00:00	13:37:00
End time	8:42:48	13:54:24
Total runtime	0:42:48	0:17:24
R1 temp change	2.187	1.563
R2 temp change	1.937	1.624
R1 rate(min/degree)	0:19:34	0:11:08
R2 rate(min/degree)	0:22:06	0:10:43

Table 5-4 Statistics of HVAC runs in smart operation without booster fans

5.3.2.3 Non-smart operation

In this test, the smart functionalities of the thermostat such as human detection algorithm and automatic HVAC control was turned off, and the thermostat operated as a regular device. The purpose of the test was to investigate the differences between smart and non-smart operations. As a result, there were more HVAC runs in this test than in the previous tests with the system's smart features. The total numbers of HVAC runs were six, compared with two in the smart operations. As seen in Figure 5-9, room 2's temperature almost never reached the desired level. The issue was because the thermostat only retrieved measured temperature values from room 1's temperature and operated the HVAC based on such data, which was always higher than room 2's. Table 5-6 describes the runs in more details.

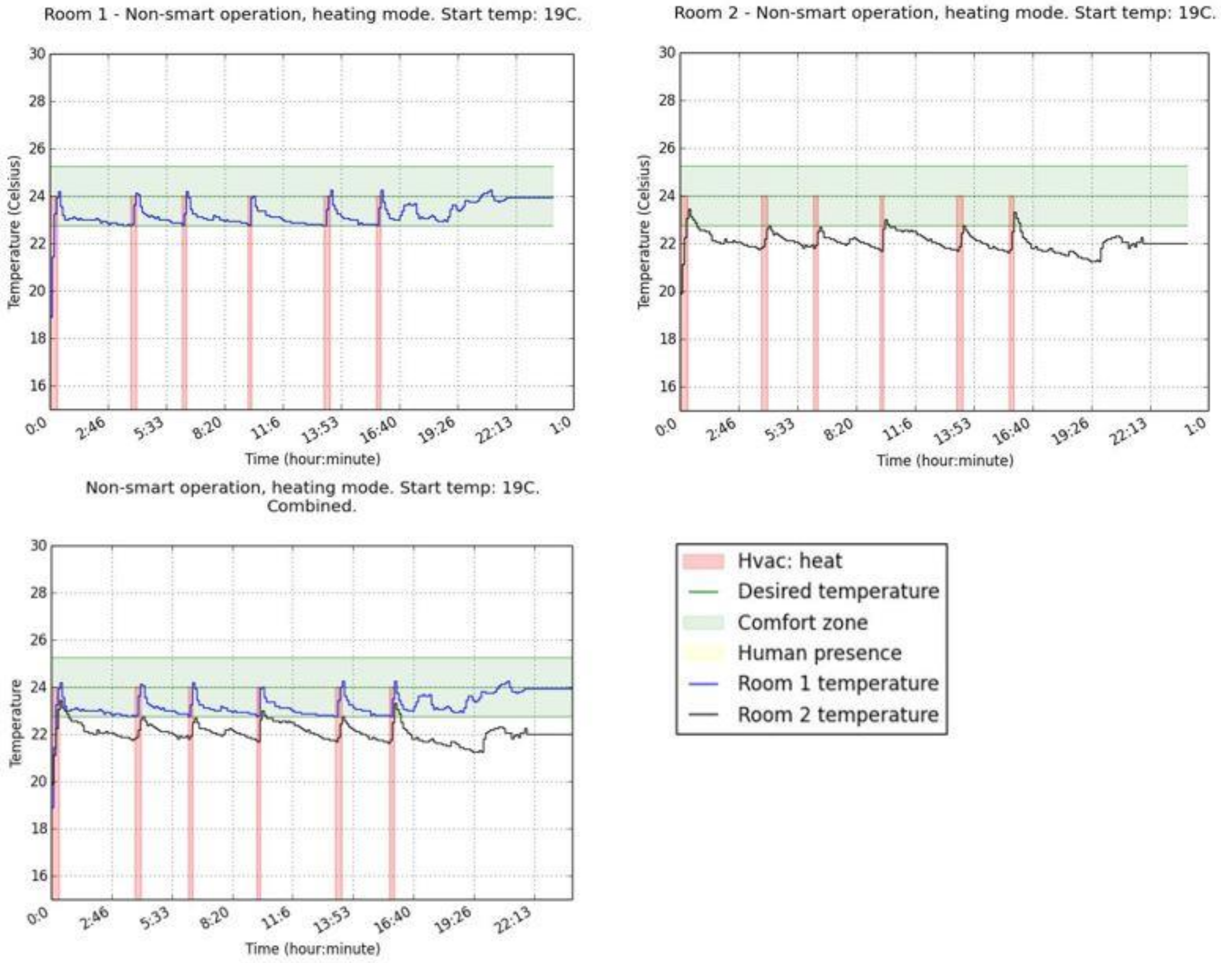


Figure 5-10 Non smart operation without booster fans.

Table 5-5 Statistics of HVAC runs in non-smart operation.

Non smart thermostat using one temperature sensor at room 1 and no booster fans Starting temperature at 19^{0}C						
Run number	1	2	3	4	5	6
Starting temp R1	18.875	22.688	22.688	22.688	22.688	22.688
Ending temp R1	24.312	24.312	24.312	24.312	24.312	24.312
Starting temp R2	19.9	21.8	21.8	21.7	21.7	21.8
Ending temp R2	23.3	22.7	22.5	22.9	22.7	23.3
Start time	0:00:24	3:51:25	6:19:07	9:28:05	13:05:15	15:34:55
End time	0:21:30	4:07:48	6:30:49	9:37:31	13:22:22	15:47:28
Total runtime	0:21:06	0:16:23	0:11:42	0:09:26	0:17:07	0:12:33
R1 temp change	5.437	1.624	1.624	1.624	1.624	1.624
R2 temp change	3.4	0.9	0.7	1.2	1	1.5
R1 rate(min/degree)	0:03:53	0:10:05	0:07:12	0:05:49	0:10:32	0:07:44
R2 rate(min/degree)	0:06:12	0:18:12	0:16:43	0:07:52	0:17:07	0:08:22

5.3.2.4 Original thermostat operation

In this test, the original thermostat of the apartment was used to operate the HVAC, and Zybo was used as a data collection device. The desired temperature was also set to 24^{0}C, and the swing value was 1^{0}C. As seen in Figure 5-10, the original thermostat used the swing value differently, where it would let the HVAC system run until room 1’s temperature reached 25^{0}C before stopping the system. As a result, room 1’s temperature was higher than the desired level, but room 2’s temperature was able to reach the comfort zone.

Comparable with the non-smart operation in 5.3.2.3, this operation started the HVAC totally seven times during testing. It also had knowledge of room 1’s temperature only, which caused the HVAC to fail to run at times when room 2’s temperature fell out of comfort zone. This issue has been a common drawback of thermostats on the market, in which it was unable to measure temperature of all areas in a house.

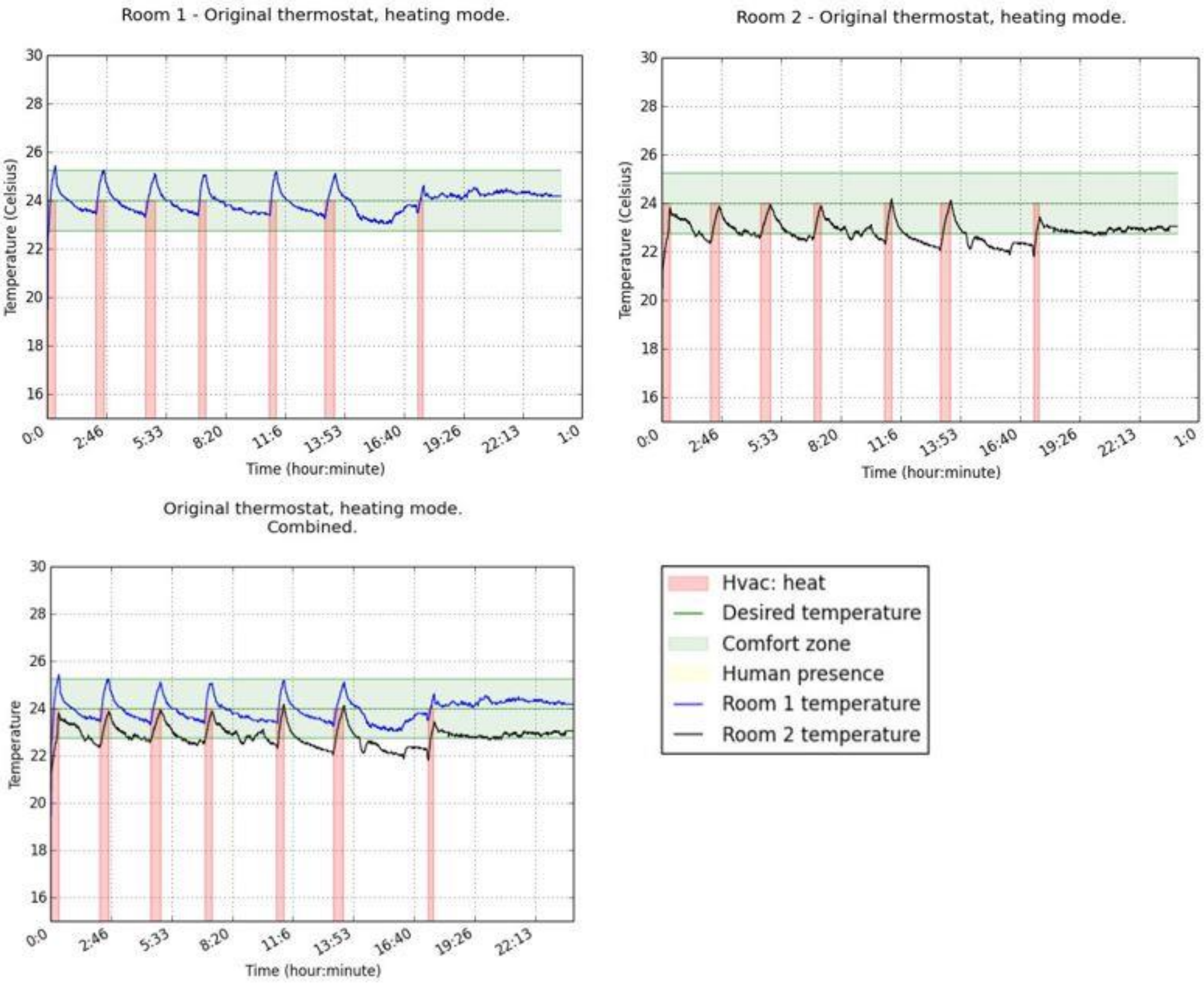


Figure 5-11 Original thermostat with basic function.

Original thermostat with basic function and no booster fans Starting temperature at 19^0C							
Run number	1	2	3	4	5	6	7
Starting temp R1	19.50	23.44	23.44	23.37	23.37	23.25	23.50
Ending temp R1	25.44	25.25	25.06	25.06	25.19	25.12	24.56
Starting temp R2	20.50	22.44	22.62	22.56	22.31	22.06	21.81
Ending temp R2	23.81	23.81	23.87	23.88	24.19	24.12	23.25
Start time	00:00:31	02:15:36	04:35:49	07:05:20	10:22:20	12:58:46	17:19:46
End time	00:22:40	02:39:55	05:03:10	07:24:40	10:41:42	13:25:43	17:33:43
Total runtime	00:22:09	00:24:19	00:28:01	00:19:20	00:19:22	00:27:37	00:13:57
R1 temp change	5.94	1.81	1.62	1.69	1.82	1.87	1.06
R2 temp change	3.31	1.37	1.25	1.32	1.88	2.06	1.44
R1 rate(min/degree)	0:03:44	0:13:26	0:17:18	0:11:26	0:10:38	0:14:46	0:13:10
R2 rate(min/degree)	0:06:42	0:17:45	0:22:25	0:14:39	0:10:38	0:13:24	0:09:41

Table 5-7 Statistics of HVAC runs with the original thermostat

5.3.4 Result analysis

Table 5-8 below shows a summary of all the HVAC runs in the tests:

	No booster fan	**With booster fan**	**Non-smart**	**Original**
Total test hours	~23	~23	~23	~23
Total HVAC runs	2	2	6	7
Total HVAC run hours	2:30:25	3:28:27	3:05:38	2:34:45
Average temperature change rate	0:16:04	0:15:27	0:10:59	0:12:09

Table 5-8 Summary of all the HVAC operations in the tests

As summarized, the thermostat in non-smart operation as well as the original thermostat ran the HVAC many times more than in smart operations, even though the total run hours were about the same across all operations. These operations without the help of the human detection algorithm would potentially waste more energy, as an HVAC run required initial extra overheads to start the system and to produce hot air.

Test results from the non-smart operation of this project’s thermostat and the original thermostat were fairly similar. There were about 6-7 HVAC runs in 23 hours of testing, twice as many as the number of runs in smart operations. This is a strong indication that the smart thermostat helped avoiding multiple HVAC starts and stops, which potentially helped save energy consumption.

Another remarkable result was, the average temperature change rate in non-smart operations was 11 to 13 minutes/degree, lower than the rate of around 16 minutes/degree in smart operations. This result was due to a fact that the thermostat's night operation turned on and off automatically in smart operations, which required the HVAC to run longer. During the test period, indoor temperatures never went below 20 ^{0}C, a preset low temperature, hence the HVAC was never turned on at night. However at 8AM, when night operation switched off, the thermostat started the HVAC when it detected human, to bring the rooms to 24 ^{0}C, the desired level. These runs at 8AM were longer than other runs, since temperatures in the morning were well below 24^{0}C.

Moreover, the thermostat in smart operations only ended an HVAC run when all rooms with human presence had reached the desired temperature. Meanwhile, the thermostat in non-smart operation and the original thermostat had no knowledge of room 2's temperature, hence it stopped the HVAC when only room 1 reached the desired temperature. This operation left room 2 out of comfort zone most of the time during the test. This issue has been well known with regular thermostats, since most of them only have one temperature sensor, not enough to capture the whole air condition in different rooms of a house.

Between the two smart operations, the one with booster fans helped increasing the rooms' temperatures slightly faster than the one without booster fans. This result was not very reliable and required further testing, since outdoor temperatures had considerable influence on the temperature change rate.

5.3.5 Comparison with Nest thermostat

In this section, the FPGA thermostat is going to be compared with Nest, one of the best-selling smart thermostats available on the markets. Table 5-9 below shows the comparison summary.

This project's thermostat	Nest thermostat
In prototype state, no friendly interface yet	Friendly user interface with control wheel
No wifi connection	Wifi connection
Uses wires to connect with peripherals	Compact hardware design
Thermal sensor and temperature sensor are placed in each room	Has only one temperature and one motion sensor
Detects human in real time	Learns user habits in 7 days
Does not need repeated input	Needs repeated input in 7 days.
All rooms are in pre-set comfort zone	Does not guarantee comfort zone in all rooms.

Table 5-9 Comparison with Nest thermostat. Strengths are in blue cells, weaknesses are in yellow cells.

The first drawback of this project's thermostat is, as described in 4.2, it used four push-buttons to receive user inputs and displayed information on an OLED screen. Since the thermostat is now still in prototype phase, it does not have a friendly and intuitive user interface as the one provided by Nest. Nest includes a LED display with well-presented information, including current temperature, date-time, user selection, etc. Nest also receives user input from a control wheel, in which one left turn or one right turn corresponds to one value of choice. This design makes the Nest thermostat itself a nice decorative addition to a residential home.

The second drawback is that this project's thermostat does not have a wireless connection to receive input and display information to users. The Nest thermostat, on the other hand, comes with a Wi-Fi connection that allows it to connect to a home's network. A user could view Nest's status and provide inputs through its mobile application. This feature is very handy especially in case when a user forgets to turn off the HVAC before leaving home, or wants to start the system in preparation for his arrival at home.

The third drawback comes from wiring across the rooms. Since this project's thermostat is still in prototyping phase, it used wires to connect between the main board and all peripherals. Since each room monitored by the thermostat needed to have a set of thermal sensors and temperature sensors, there was a considerable amount of wiring between rooms of the test environment. Meanwhile, a Nest thermostat is self-contained and easy to install, it has a round shape and can be mounted to a wall.

Despite the mentioned drawbacks, the thermostat in this project does have advantages compared to a Nest device. As described in chapter 5, since there was a pair of thermal and temperature sensors placed in each room in test, the thermostat had a good knowledge of current temperatures and human presence in every room it supported. This real-time knowledge allowed the thermostat to better control the HVAC by starting and stopping it only when there was human presence in a room, and when temperatures went out of comfort zone. On the other hand, since Nest only has one pair of built-in motion and temperature sensors, it can only capture temperature and motion happening around it. As a result, a Nest thermostat's efficiency depends on where it is mounted in a house.

Moreover, as described in chapter 5, the sensor pairs mounted in each monitored room also allowed this project's thermostat to know specifically what temperature each room had. Thus, it started and stopped the HVAC accordingly to bring the rooms with human presence into a preset comfort zone, even though the temperature at the thermostat area was already at desired level. This feature helped users stayed in their comfort zone longer without the burden of repeatedly changing temperatures set in the thermostat. On the contrary, the Nest thermostat only captures temperature of its surrounding space, hence it cannot not start or stop the HVAC automatically if the temperature of another room in the house goes out of comfort zone. In this case, if a user wants to bring the other room to comfort zone, he or she has to adjust the thermostat manually or with the help of a timer. Even though a timer keeps a user from repeated input, it becomes ineffective if the user's routine timing changes.

In addition, the Nest thermostat needs initial user inputs for seven days right after it is connected to an HVAC system. Since it uses a learning algorithm to gain knowledge of a user's habits, the algorithm does not work properly until enough data is received. Alternatively, with this project's thermostat, the user only needs to set three temperatures one time: desired, low and high, which define their comfort zone. The thermostat will detect human presence and make sure that the comfort zone is maintained in rooms where there is human presence. The real-time knowledge provided by the rooms’ sensors allows the thermostat to automatically start and stop the HVAC without the need of repeated input from a user.

This chapter explained how an experiment environment was setup, preparing the thermostat to get data from thermal sensors, temperature sensors and operate the HVAC system. To debug the thermostat, an oscilloscope was used to look for hardware issues. To debug software issues, log data was collected and analyzed. After the setup, multiple testing efforts were done to verify the thermostat’s functionalities, and results were also reported and analyzed. Finally, the smart thermostat was compared with Nest thermostat, a famous smart thermostat on the market, in which both advantages and disadvantages of both systems were discussed.

6. Conclusion

In this project, I applied different electrical and computer engineering technologies to build a thermostat that is capable of real time detecting the presence of a person and adjusting the temperature of the room accordingly. The system was designed on Zybo, a board featuring a dual-core ARM Cortex-A9 processor with a Xilinx 7-series Field Programmable Gate Array (FPGA) logic. The peripherals used in this project include two thermal sensors, two temperature sensors, two booster fans, two relays and an OLED screen. The thermostat was able to detect the presence of a person and capture the temperature in each of the room that is equipped with the peripherals. The thermostat used the data from the peripherals to automatically control the HVAC system allowing users to stay in their comfort zone longer while avoiding multiple HVAC start/stop routines.

Since the design of the thermostat has limitations as mentioned in 5.3.5, there are several potential improvements. A keyboard might be added to receive user input, and a screen with higher resolution can be used to display more information. Also, there are multiple low cost thermal sensors available on the markets that have resolution higher than 4x4 pixels. They should potentially yield better human detection results.

The thermostat can be improved by implementing a wireless connection between the peripherals and Zybo to make the installation simpler. This improvement greatly increases the flexibility of the thermostat by allowing users to place the sensors wherever they want. It also allows additional sets of peripherals to link to the Zybo, since wireless connections do not depend on the number of ports available on the system. If the software program is able to handle more than two rooms, the thermostat can better assist users by allowing them to stay in their comfort zone even longer now that there are more sensors to detect human presence. Additionally, a new management application can be developed to use the wireless connection to help users manage the thermostat remotely, one that is similar to the mobile app that comes with the Nest thermostat.

References

[1] S. Gerrity, "Energy saver 101 inforgraphic home heating," 16 12 2013. [Online]. Available: http://energy.gov/articles

[2] S. Gerrity, "Energy saver 101 infographic home cooling," 13 06 2014. [Online]. Available: http://energy.gov/articles/

[3] Department of Energy Resources, "Background," [Online]. Available: http://www.mass.gov/eea/energy-utilities-clea

[4] U.S. Energy Information Administration (EIA), "Household Energy Use in Massachusetts," 2009. [Online]. Available: http://www.eia.gov/consumption/residential/reports/2009/state_briefs/pdf/ma.pdf. [Accessed 04 11 2015].

[5] G. W. J. Gupton, HVAC Controls - Operation and Maintenance (3rd Edition), Fairmont Press, Inc., 2002.

[6] Florida Solar Energy center, "HVAC Systems," [Online]. Available: http://www.fsec.ucf.edu/en/consumer/buildings/

[7] U.S. Department of Energy, "Air source heat pumps," [Online]. Available: http://energy.gov/energysaver/air-source-

[8] Advance NRG, "Heating Ventilating and Air Conditioning with Advance NRG," [Online]. Available: http://www.advan

[9] P. Bob Scaringe Ph.D., "Thermostatic Wiring Principles," [Online]. Available: http://www.epatest.com/store/resourc

[10] Merriam-Webster, "Dictionary - thermostat," [Online]. Available: http://www.merriam-webster.com/dictionary/the

[11] J. Carlsen, "Programmable Thermostats Review," 23 06 2015. [Online]. Available: http://programmable-thermostats

[12] RLW Analytics, "Validating the Impact of Programmable Thermostats," Middletown, 2007.

[13] Amerivcan Society of Heating, Refrigerating and Air-Conditioning Engineers, "ANSI/ASHRAE 55a-1995," Atlanta, 199

[14] AZoSensors.com Staff Writers, "Determining Thermal Comfort Using a Humidity and Temperature Sensor," [Online]. 11 2015].

[15] Xilinx , "Zybo Reference Manual," 14 02 2014. [Online]. Available: http://www.xilinx.com/support/documentation/u [Accessed 06 11 2015].

[16] D. Inc., "ZYBO Zynq™-7000 Development Board," [Online]. Available: http://digilentinc.com/Products/Detail.cfm?Na

[17] D. Inc., "ZYBO Reference Manual," 14 02 2014. [Online]. Available: http://www.xilinx.com/support/documentation/ [Accessed 15 11 2015].

[18] julieclarke, "MicroZed Timers, Clocks and Watchdogs: Adam Taylor's MicroZed Chronicles Part 14," 06 01 2014. [Onl directory/partner-xilinx/blog/2014/01/06/microzed-timers-clocks-and-watchdogs-adam-taylor-s-microzed-chronicle

[19] Omron Corporation, "Applicatation Note 01 - Usage of D6T-44L Thermal Sensors," 21 07 2015. [Online]. Available: h 01EN_r2.pdf. [Accessed 04 11 2015].

[20] D. Inc., "PmodTMP2 - Thermometer/thermostat," [Online]. Available: http://www.digilentinc.com/Products/Detail.c

[21] Digilent Inc., "PmodTMP2 - Thermometer/thermostat," [Online]. Available: http://digilentinc.com/Products/Detail.c

[22] Analog Devices Inc., "I2C Temperature Sensor - ADT7420," 2012. [Online]. Available: http://www.analog.com/media
[Accessed 06 11 2015].

[23] D. Inc., "PmodOLED - Organic LED Graphic Display," [Online]. Available: http://www.digilentinc.com/Products/Detai

[24] Digilent Inc., "PmodOLED reference manual," 19 10 2011. [Online]. Available: https://www.digilentinc.com/Data/Pro

[25] Univision Technology Inc., "OEL Display Module - UG2832HSWEG04," 24 07 2009. [Online]. Available: https://www.a

[26] Embedded Centric, "Zedboard OLED," [Online]. Available: http://embeddedcentric.com/zedboardoled-v1-0-ip/. [Acc

[27] R. Gade and T. B. Moeslund, "Thermal cameras and applications: a survey," *Machine VIsion and Applications,* vol. 25

[28] Y. e. a. Fang, "A Shape-Independent Method for Pedestrian Detection with Far-Infrared Images," *IEEE Transactions o*

[29] J. &. S. V. Davis, "Robust Detection of People in Thermal Imagery," in *17th International Conference on Pattern Reco*

[30] W. Z. J. &. S. C. Wang, "Improved Human Detection and Classification in Thermal Images," in *2010 IEEE International*

[31] A. e. a. Fernandez-Caballero, "Real-time human segmentation in infrared videos," *Expert Systems With Applications,*

[32] W. e. a. Wong, "Homw alone faint detection surveillance system using thermal camera," in *2nd International Confer*

[33] D. &. G. J. Gavrila, "Shape-based pedestrian detection and tracking," p. 8, 2003.

[34] J. &. K. M. Davis, "A Two-Stage Template Approach to Person Detection in Thermal Imagery," vol. 1, p. 364, 2005.

[35] M. e. a. Oren, "Pedestrian detection using wavelet templates," in *Computer Vision and Pattern Recognition*, 1997.

[36] P. Viola, M. J. Jones and D. Snow, "Detecting Pedestrians Using Patterns of Motion and Appearance," vol. 63, no. 2, p

[37] T. &. B. N. Elguebaly, "A nonparametric Bayesian approach for enhanced pedestrian detection and foreground segm

[38] L. &. P. K. Fan, "A comparative study of PCA, ICA and class-conditional ICA for naive bayes classifier," p. 16, 2007.

[39] C. A. (. A. Wunderlich, "Medical Thermometry and Human Temperature," 1871.

[40] G. Kelly, "Body temperature variability (part1): A review of the history of body temperature and its variability due to
vol. 11, no. 4, pp. 272-293, 2006.

[41] Z. H. Y. H. J. &. J. L. Wang, "Human skin temperature and thermal responses in asymmetrical cold radiation environn

[42] Y. W. L. L. J. &. D. Y. Liu, "A study of human skin and surface temperatures in stable and unstable thermal environme

[43] D. Invcc, "Digilent Pmod Interface Specification," [Online]. Available: https://www.digilentinc.com/Pmods/Digilent-P

[44] N. Semiconductors, "P82B715 I2C-bus extender," [Online]. Available: http://www.nxp.com/documents/data_sheet/

[45] NXP, "P82B715 I2C Bus extender," [Online]. Available: http://www.nxp.com/documents/data_sheet/P82B715.pdf. [

[46] A. Devices, "Data sheet of ADT7420," [Online]. Available: http://www.analog.com/media/en/technical-documentati

[47] Xilinx, "Xilinx general technical discussion," [Online]. Available: https://forums.xilinx.com/t5/General-Technical-Disc

[48] T. Instruments, "PCA9517 Level-Translating I2C bus repeater," [Online]. Available: http://www.ti.com.cn/cn/lit/ds/s

[49] NXP, "I2C bus extender P82B715," [Online]. Available: http://www.nxp.com/documents/data_sheet/P82B715.pdf. [

[50] D. Inc, "PmodOLED Example Code," [Online]. Available: https://reference.digilentinc.com/pmod:pmod:oled:example

[51] D. Inc, "Pmod OLED," [Online]. Available: https://www.digilentinc.com/Data/Products/PMOD-OLED/PmodOLED_rm

[52] M. Grusin, "Serial Peripheral Interface-SPI," [Online]. Available: https://learn.sparkfun.com/tutorials/serial-peripher

[53] ASCIItable.com, "ASCII Table and Description," [Online]. Available: http://www.asciitable.com/. [Accessed 07 12 201

[54] T. A. U. a. Qatar, "ZedboardOLED Display Controller IP v1.0," [Online]. Available: https://embeddedcentric.files.word

[55] 4tronix, "5V 4-channel relay interface board," [Online]. Available: http://4tronix.co.uk/store/index.php?rt=product/

[56] U. E. P. A. (EPA), "A guide to energy-efficient heating and cooling," [Online]. [Accessed 03 12 2015].

[57] K. Cheung, "Tektronix MSO70000 Mixed Signal Oscilloscope," 13 10 2009. [Online]. Available: http://edablog.com/2

[58] J. Carlsen, "Programmable Thermostats Review," 23 06 2015. [Online]. Available: http://programmable-thermostats

[59] Nest Support, "nest.com," 17 06 2015. [Online]. Available: https://nest.com/support/article/What-is-a-multistage-s

[60] Omron Corporation, "Infrared MEMS Thermal Sensor," [Online]. Available: http://datasheet.octopart.com/D6T8L06

[61] M. rouse, "I2C bus (Inter-IC bus)," 09 2005. [Online]. Available: http://whatis.techtarget.com/definition/I2C-bus-Inte

[62] ELectronics Egnineering Herald, "Module 12 - SPI Bus interface," [Online]. Available: http://www.eeherald.com/sect

[63] K. Tweed, "Smart Thermostats Begin to Dominate the Market in 2015," 22 07 2015. [Online]. Available: http://www
market-in-2015. [Accessed 07 11 2015].

[64] D. Inc., "Digilent Pmod Interface Specification," [Online]. Available: http://www.digilentinc.com/Pmods/Digilent-Pm

[65] Sunon, "MagLev Motor Fan," 24 09 2009. [Online]. Available:
http://portal.sunon.com.tw/pls/portal/sunonap.sunon_html_d_pkg.open_file?input_file_name=7264646F632F323
[Accessed 30 11 2015].

[66] C. Mechanical, "Frequently asked questions," [Online]. Available: http://climatemechanical.com/faq.htm. [Accessed

[67] Amazon, "JBtek 8-channel relay," [Online]. Available: http://www.amazon.com/JBtek-Channel-Relay-Arduino-Raspb

[68] Digikey, "Omron D6T44L06," [Online]. Available: http://www.digikey.com/product-detail/en/D6T44L06/Z3637-ND/3

Appendix A: Development Code

Verilog code of thermal sensor module

Verilog code of temperature sensor module

Appendix B: Embedded Software Excerpts