Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Feng An

Entitled Embedded System for Sensor Communication and Security

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

Professor Maher Rizkalla
Chair

Professor Michael Knieser

Professor Li Lingxi

Professor Paul Salama

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Professor Maher Rizkalla

Approved by: Professor Yaobin Chen 06/11/2010
Head of the Graduate Program Date

Graduate School Form 20
(Revised 1/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:
Embedded System for Sensor Communication and Security

For the degree of Master of Science in Electrical and Computer Engineering

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Teaching, Research, and Outreach Policy on Research Misconduct (VIII.3.1)*, October 1, 2008.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Feng An
Printed Name and Signature of Candidate

06/11/2010
Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/viii_3_1.html

EMBEDDED SYSTEM FOR SENSOR COMMUNICATION AND SECURITY

A Thesis
Submitted to the Faculty
of
Purdue University
by
Feng An

In Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Electrical and Computer Engineering

August 2010
Purdue University
Indianapolis, Indiana

ACKNOWLEDGEMENTS

I would like to gratefully acknowledge my thesis advisor, Dr. Maher Rizkalla, for his assistance, guidance, and supervision during the entire course of this research and thesis work. Dr. Rizkalla generously shared with me his research experience and showed me his pursuit of perfection in every single detail, for which I am always thankful.

I would like to thank my advisory committee members, Dr. Paul Salama, Dr. Eliza Du, and my oral defense committee members Dr. Yaobin Chen, Dr. Brain King, Dr. Lingxi Li, Dr. Michael Knieser for their time and insight during the completion of this thesis.

I would like to extend my special thanks to Dr. Knieser, CEO of Smart Systems Incorporation, for his valuable contributions and insightful discussions that led to successful completion of this research. The meetings with Dr. Knieser and the valuable insight and scientific guidance provided by him are grateful acknowledged. The experiments conducted at Electrical and Computer Engineering Department of IUPUI and Smart Systems Incorporation for designing a prototype of product presented in this research have been very instrumental for completion of my thesis. I extend my sincere gratitude to them.

I would also like to thank Ms. Valerie Lim Diemer for assisting me in formatting this thesis. Finally, I express my gratitude to my parents for their support and encouragement during all my life.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

ABSTRACT

An, Feng. M.S.E.C.E., Purdue University, August 2010. Embedded System for Sensor Communication and Security. Major Professor: Maher Rizkalla.

In this work, inter-integrated circuit mode (I2C) software was used to communicate between sensors and the embedded control system, utilizing PIC182585 MPLAB hardware. These sensors were built as part of a system on board that includes the sensors, microcontroller, and interface circuitry. The hardware includes the PIC18 processor, FPGA chip, and peripherals. A FPGA chip was used to interface the processor with the peripherals in order to operate at the same clock speed. This hardware design features high level of integration, reliability, high precision, and high speed communications. The software was first designed to operate each sensor separately, then the sensor system was integrated (to combine all sensors, microcontroller, and interfacing circuitries), and the software was updated to provide various actions if triggered by the sensors. Actions taken by the processor may include alarming signals that are based on threshold values received from the sensors, and inquiring temperature and CO2 readings. The system was designed for HVAC (heating, ventilating and air conditioning) applications and industrial settings. The overall system incorporating temperature and CO2 sensors was implemented and successfully tested. The response of the multi-sensor system was agreeable with the design parameters. The system may be expanded to include other sensors such as light senor, pressure sensor, etc. Monitoring the threshold values should add to the security features of the integrated communication system. This design features low power consumption (utilizing the sleeping mode of the processors), high speed communications, security, and flexibility to expansion.

1. INTRODUCTION

1.1 Sensor Applications

A sensor is a device that converts from one type of energy to another. This may include physical, biological, chemical, electrical, or optical energy. Signals received by these sensors are in the analog form, and can be digitally formatted and processed by computers. Smart sensors can communicate efficiently with the external world. Sensors are designed to have on-board functions including energy management, actuation and location tracking, and communication modules. Embedded systems are utilized to connect PCs for data transfer. They are capable of integrating multiple sensors on board. The following applications show the importance of smart sensors in latest applications.

1.1.1 Wireless Applications

Sensors can be used to design integrated data acquisition system [1]. This can provide a high-resolution serial digital output for processing by navigation and guidance system, sequencing system or telemetry system in the aerospace vehicle. The interface system may include analog to digital conversion, and signal conditioning blocks. Sensors may be placed around the plane to transmit the environmental variables (weather, humidity, storm strength, height, wind speed, and temperature) to front computers [2]. Wireless communication schemes and parameters have good anti-interferences property and security performance. Wireless sensor networks (WSN) include hundreds to thousands of nodes with low-power and computational capability features. WSN popularity has increased dramatically due to their potentiality in civil and military applications. Other applications may include surveillance missions in which WSN is used for detecting moving targets or objects [3, 4, 5, 6, 7].

1.1.2 Automotive Applications

Sensor technology is utilized in measuring physical quantities such as position, pressure, torque, and exhaust temperature. Furthermore, in automotive application, angular rate, fuel composition, linear acceleration, and speed/timing can be sensed for monitoring purposes [8, 9]. Sensors enhance the safety, security, reliability for the automotive systems.

1.1.3 Medical Applications

Sensors are fully utilized in medical applications for the benefit of humanity. These include MRI (magnetic resonance imaging), RF (radio frequency), ultrasound, image recognition, electro-magnetic, and thermo-graph. Sensors in these applications are built in small size, low power, and low cost [10].

1.1.4 HVAC Applications

For HVAC (heating, ventilating and air conditioning) sensors system, controllers usually employ a control algorithm to make the HVAC system with high energy-efficiency. HVAC system helps to manage the environmental variables and controls domestic appliances such as windows, fridge, and oven, in addition to controlling temperature, humidity and CO2 density. To maintain temperature at the reference point, the controller adjusts the flow of hot/chilled air for heating/cooling operations [11]. To control CO2 density, fresh air will be regularly sampled for monitoring maximum allowable CO2 density. Embedded sensors are the primary type of sensors in the HVAC applications. Figure 1.1 shows TI solution for HVAC applications with various controlling functions [12].

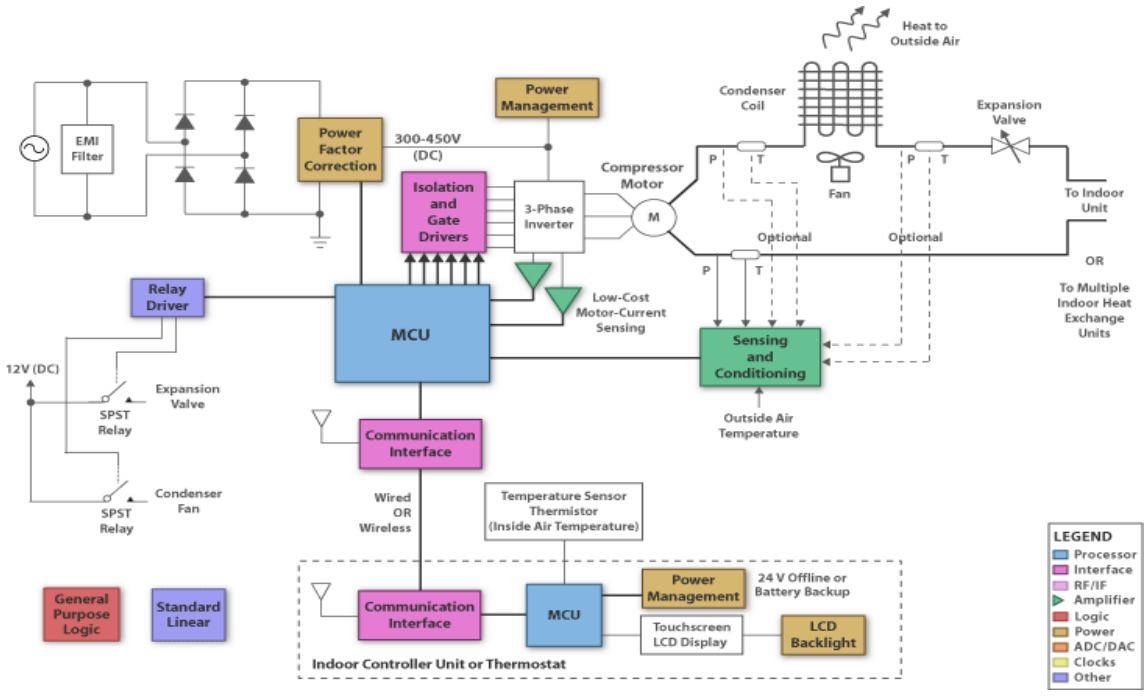


Figure 1.1 TI Solution of HVAC Applications [12]

1.1.5 Civilian Applications

Utilizing radio frequency sensors, in wireless sensing identification (WSID) systems, traffic can be monitored. In this case, hundreds of sensor nodes are used to collect and sense traffic data, and transmit them to the base station [13, 14]. High sophisticated smart sensors are utilized for real-time traffic control [15, 16].

1.2 Sensor Technology

1.2.1 Device Technology

Sensors are made of various materials and devices that are appropriate for sensing the physical parameters: optical, biological, chemical, acoustic, inertial, infrared, magnetic, and radiation.

1.2.1.1 Optical Sensors

Optical sensors [17, 18, 19, 20, 21] cover image sensors with computational imager that is based on CMOS integrated circuit [17]. These sensors are capable of providing simultaneous image data from dynamically reconfigurable regions with different spatial resolution. Optical sensors can be used to monitor mechanical pressure, surface micro-machined deformation, and external applied forces [19]. Furthermore, optical sensors can also be used in surveillance systems. For instance, in optical tomography systems, optical tomography can be customized to control the data acquisition process of multiple optical sensors array, arranged in parallel beam projection. The data collected can be used to measure the velocity profiles [21].

1.2.1.2 Biological Sensors

The MEMS and ASIC technology has contributed significantly in biological sensors. Biological study requires sensors for micro-machining, for lab-on-chip, for detection of chemical and physical changes [22, 23, 24]. ASIC and MEMS systems offer high performance biological sensing capabilities [22]. Those sensors consist of a MEMS capacitive sensor element monolithically integrated with a sensing circuit [11, 13, 14, 15, 25, 26, 27]. Furthermore, some types of ASIC and MEMS technologies such as carbon nano-wires offer features in electronic properties to study novel nano-structures of biological cells. The carbon properties of these sensors are applicable to medicine and biology fields [28].

1.2.1.3 Acoustic Sensors

Acoustic sensors cover capacitor type, resistance type, and modulated FET type, etc, manufactured by micro-mechanical design. In this technology, micro-channels, micro-tubes, micro-pipes, micro-mirrors can be implemented. They are compatible with CMOS microelectronics fabrication techniques.

Acoustic sensors also include pressure sensors, micro-phone, and micro camera. In these applications, acoustic vibration will be detected and utilized. These sensors also are capable of sensing surface acoustic wave (SAW) and optical fiber acoustics. Optical fiber acoustic sensors are based on the theory that sound wave frequency differences will influence amplitude or phase differences in fibers. Each chip can hold hundreds of those kinds of sensors, so that they can work in arrays to make better precision. Surface acoustic wave (SAW) is a device that is fully compatible with CMOS technology and can be used for the bio/chemical applications. ZnO is a common material used for the piezo-electric SAW generation. SAW device is suitable for checking out environmental changes due to their reluctance against electro-magnetic (EM) interferences [29].

1.2.1.4 Inertial Sensors

There are two categories of inertial sensors: gyroscope and micro-accelerator, and both use MEMS technology.

Gyroscope is an angular velocity sensor, and can be used in inertial navigation, automotive chassis control, and rollover detection. Currently, most designs of gyroscopes are micro-machined and vibratory types. They are based on the transfer of energy between two vibration modes by the Coriolis force [30]. Micro-accelerators are used extensively in automotive industry, including vehicle stability and chassis adaptive suspension sensing, vehicle frontal rollover crash sensing, and engine knock detection.

1.2.1.5 Infrared Sensors

Infrared detectors may cover pyro-electric based, carbon nano-tubes (CNT) based, and thermo-electric based infrared sensors.

Pyro-electric infrared sensors are composed of thin films that are grown on the substrates. These sensors are built with signal processing ASIC unit that may include

several amplifier units. Infrared sensors are used to monitor gas concentration using the concept that gas absorbs infrared radiation at specific and unique wavelengths [31]. Carbon nano-tubes (CNT) can also be used to design infrared sensors. They can be fabricated on array forms and these assist in cancelling out electronic noise and interference [32]. Thermo-electric infrared sensors are designed based on the change of thermistor's resistance under the irradiation FIR light [33]. This type of sensors can be used in intrusion alarming [34]. Crystal SiGeC infrared sensor is a good example of this type; it is portable and can be used in rehabilitation system application [35].

1.2.1.6 Magnetic Sensors

Magnetic sensors rely on some compounds which have magnetic properties. Those sensors can introduce magnetic changes that may be reflected in current and voltage changes. Magnetic sensors can utilize the magnetic property of the earth, once any object that contain iron has move, those sensors can check their moving status, such as speed. Recent enhancement in MEMS technology makes this kind of sensor available for use. These sensors can be used in vehicle supervision system [36, 37].

1.2.1.7 Radiation Sensors

Radiation sensors utilize nuclear materials which can emit nuclear waves. Because different substances have different absorption and different reactions to nuclear waves, this kind of sensors can be used to verify the composition of materials.

1.2.2 Device Fabrication Techniques

Semiconductor sensors are major contributors in sensor technology. Most of the design approaches are compatible with MEMS, lab-on-chip, ASIC, and system-on-chip implementation. These are based on novel integrated circuits and have highly advanced microelectronics fabrication processes.

1.2.2.1 Digital Signal Processing Unit

With the recent development of CMOS and MEMS technology, recent sensors can handle signal processing together with data acquisition and transmission. Figure 1.2 is a typical formation of digital signal processing sensor system.

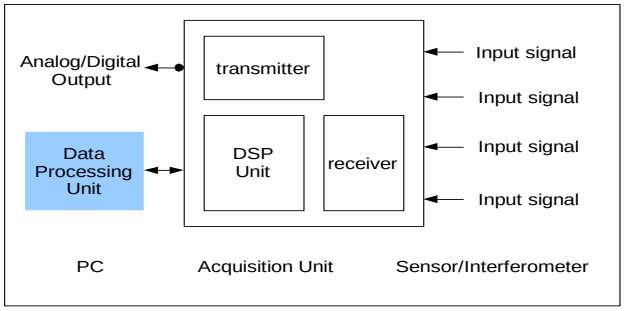


Figure 1.2 A Typical Formation of Digital Signal Processing Sensor System

In Figure 1.2, the three important components used for the system are the PC, the acquisition unit, and the sensor/interferometer. The digital-to-analog converter (DAC) allows the DSP to dictate the modulation type whether sine wave or saw-tooth. The analog-to-digital converter (ADC) is also utilized in converting detected signal to the digital domain before transferring the signals to DSP unit. The digitized signal is then demodulated by the DSP in order to extract the measured signal. The ADCs usually include an amplifier stage that is controlled by the DSP software in real-time. The software will synchronize the incoming modulated signal with the local oscillator signal employed by the detection scheme.

1.2.2.2 CMOS-MEMS Technology

MEMS stand for micro-electro-mechanical systems. They have a feature of integrating the required signal conditioning circuits on the same silicon wafer with the MEMS device. This enables the complete stand-alone system, including the device and

the processing unit. CMOS MEMS based technology help make sensors with high precision, low-power, portable, adaptive, and high computational ability. The use of standard IC processes in developing CMOS MEMS technology has led to mass-production at low cost.

MEMS technology uses surface micromachining that provides a complementary technique in which materials and devices are added above the surface. This makes it the most popular sensor fabrication technology [38].

The CMOS-MEMS technology utilized in sensors has become very popular in recent years because of their high level of integration and reliability. Recently, the level of integration has come to the nano-scale level with features of low parasitic capacitance and low power consumption. This nano-technology based sensors have been utilized in applications that were never been used with micro-devices. This technology can be also used for general purpose sensing, including magnetic, radiation, optical, biological, civilian, military, and automobile applications [39].

CMOS technology deals with monolithic integrated sensors. The features of these sensors are lower manufacturing cost, high performance, compatibility for mass-production requirements, low power, reliability, and proper interfacing with DSP and computer systems.

CMOS-based sensor technology can be made to replace micro-machining devices. Several instances of CMOS-based sensors are CMOS imagers, Hall Effect magnetic sensors, and piezo-resistive stress sensors.

Furthermore, CMOS technology based sensors are used to build for accelerators, gyroscopes, infrared sensors, gravimetric sensors, electro thermal actuators, RF tunable capacitors, and RF mixer-filters. The on-chip CMOS technology based sensors require

preamplifiers to achieve the input voltage sensitivity. This may result in high signal-to-noise ratio for the integrated sensor system. Modern CMOS processes may require stress matching between dielectric and metal layer to enable chem-mechanical thin-film processes [40].

1.2.3 Material Technology

The major types of materials used for sensors are silicon, quartz, glasses, ceramics, plastics, polymers, and metals, while silicon, quartz and glasses are the most widely used in the market. The low resistivity of the silicon substrates helps with the latch-up characteristics of the CMOS circuits [41]. The property of quartz enhances the MEMS mechanical sensors used for resonators, gyroscopes, and accelerometers. Glass can be used for MEMS structures that can be used in thermally unstable environments [42]. This is due to the high thermal expansion coefficient. A newly type of material used for MEMS mechanical sensors includes ceramics such as alumina, and polymers, such as polymides. Furthermore, metallic compounds such as Au, Ni, and ZnO allow high performance mechanical sensors.

1.2.4 Communication Protocol

With the most recent technological improvements, sensor networks enable a high quality detection and measurement. Besides, sensors can be interfaced by protocols such as inter-integrated circuit (I2C) protocol, serial peripheral interface (SPI) protocol, universal asynchronous receiver and transmitter (UART) protocol, controller area network (CAN) protocol, and universal serial bus (USB) protocol, to communicate sensor signals with controllers.

I2C is intended for application in systems which connects microcontrollers and other microcontroller based peripheral devices. SPI is a three wire serial bus for 8-bit data-scheme based communication applications. Two of the three lines transfer data and

the third is a serial clock. Similar to I2C bus, devices on the bus are defined as masters and slaves. A master initiates an information transfer and generates the clock. Each slave device on the bus is controlled by a chip select line, which is a parallel line for each bus node in addition to the three SPI bus signals. SPI is an attractive bus for smart sensor micro-systems. Two separated data lines and the simple shift register data transfer scheme make the hardware of an SPI interface much simpler than that of the I2C bus. Unlike I2C, the SPI bus does not allow new sensors to be easily added to the system without any additional external interfacing. UART is used for asynchronous communication with serial input/output devices. Typically, the UART is used between a central processor and a serial device [43]. RS232 standard is a protocol of asynchronous serial communication and has been widely used in personal computers (PCs) and communication industries. It is based on UART protocol and uses 232 PWL. Other UART devices may have TTL PWL, such as asynchronous communication ports in microprocessors. Many short distance communication peripherals of PC such as printers, disks, and terminals all communicate with a PC via RS232 interface. While RS232 has the advantage of being available in most computer hardware set-ups, and thus easy to implement for the communication between relays and microcomputers, it has the disadvantage of being difficult to form networks (especially in situations when multiple transmit units on one line is required) [44]. CAN is an asynchronous serial CSMA/CD communication protocol for microcontrollers networks, supporting distributed real-time control (bit rate up to 1Mbps) with a very high level of security. CAN communication protocol is based on a distributed scheme, there is no central unit, allowing a direct data transfer between any two or more nodes without a master node mediation [45]. USB works as a Master/Slave bus, where the USB Host is the Master and the devices are the slaves. The only system resources required by a USB system are the memory locations used by USB system software and the memory and/or IO address space and IRQ line used by the USB host controller. USB slave devices can be functional (displays, mice, etc) or hubs, used to connect other devices onto the bus [46].

1.2.5 Wireless Sensor Protocols

Communication schemes facilitate reliable multi-hop data packet transmission and reliable multi-hop control packet transmission. Furthermore, these schemes enable important information to reach its destination with minimum delay and output congestion [47]. Most of the contention based media access control (MAC) protocols follow the operational model of carrier sense multiple access (CSMA), incorporating handshaking signals to reduce the probability of collisions. The widely used standardized IEEE 802.11 distributed coordination function (DCF) is an example of the contention-based protocol. This protocol works in two different modes: the infrastructure mode and ad hoc mode. In the first, communication between nodes go via a central node, while in the second, communicate occurs directly among nodes. The binary exponential back off (BEB) mechanism is used for resolving packet collisions that occur within the uncoordinated nodes in the IEEE 802.11 mechanism [48, 49].

1.3 Sensor Circuitries

Mixed signal processing modules are commonly used in sensor technology. They combine both analog and digital signals processed simultaneously. General types of sensors are specified by the output signal type. Analog sensors output analog signals. Those signals must pass through A/D devices to connect to microcontrollers. They are built with integrated analog sensors and A/D convertors on the same chip. The output signals are formatted to be CMOS compatible with microelectronics technologies [50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60]. Medical image processing is an example for analog signal processing application. Several instances may include nuclear medicine, computed tomography (CT), magnetic resonance imaging (MRI), and positron emission tomography (PET) [51].

In order to maximize signal to noise ratio (SNR) and minimize the noise offset. Magnetic sensor bias circuits and signal amplification circuits suitable for sensor arrays may be used [55].

Digital signal sensors integrate analog signal processing module and A/D convertor so that demodulation and control algorithms can then be implemented digitally.

An application to digital signal processing is in the fiber optic sensors area. This system transmits a modulated signal to the sensing interferometer, demodulates the returning interferometric signal by way of digital signal processing, while simultaneously relaying control and signal data via PC based software [58].

Mixed signal processing circuits utilizing VLSI techniques are commonly used in the implementation of smart sensor design. This may include analog signal processing such as scaling, addition, and integration, discrete time circuits, etc [61].

1.4 Smart Sensors

These sensor systems may include physical sensors, sensor interfaces to microcontrollers, DSP processors, and data communication protocol. High performance smart sensors systems may include self-adaptation and self-identification electronics. In this case, they are called adaptive sensors, and they have the capability to hold past memory, and have computational ability.

Microprocessor is the core of signal processing module. It receives signals from amplitude transformation, linear compensation, and digital filtering. With the advancement of the MEMS and CMOS technologies, actuators and microprocessors can be integrated and packaged together, and have better anti-interference ability.

Compared to traditional sensors, smart sensors are widely used to acquire signals from vision, feeling, sound, smell, memory, thoughts, and logical arbitration. Those types of sensors possess abilities ranging from self-compensation, calculations, and diagnoses. Self-compensation helps to fix temperature migration and non-linear compensation problem, statistics calculation helps to calibrate some sensitive parts in sensors and self-

diagnoses helps to do online diagnosis when system is operating. Many physical and chemical parameters inside the system can be checked out simultaneously. This enhances the capability that makes these sensors accommodate data interfaces, numerical analysis, self-process raw data, and instant learning. This is attributed to the large on-board RAM that helps sensors to have better processing speed and multiple signals processing abilities. Furthermore, these abilities help maintain sensor system to make remote sensing realizable.

1.5 The Thesis

Throughout the sensor technology discussed above, there were issues regarding real time applications, reliability, manufacturing costs, and flexibility of interfacing to external world. One approach utilized here was to use embedded integrated sensing technology, where the sensor and processing unit are all integrated onboard. Sensor communications was accomplished via I2C protocol. A system model was developed and tried on two integrated sensing units: Temperature sensor and CO2 sensor. The processing unit was designed and manufactured at Smart System Incorporation [62].

The software for the individual sensors were written and run successfully. The same protocol was used to communicate these sensors with the processor in order to generalize its use for HVAC. The following section describes the contents of the thesis: Chapter 2 gives the hardware description of the sensors used in the design and their communication protocols. Chapter 3 describes software models of the MPLAB hardware. In addition, FPGA chip information interfaced with PIC182585 microcontroller was detailed. Chapter 4 presents the results and discussions for each sensor separately as well as for the overall integrated system. Chapter 5 concludes the findings of the research work and proposed further ideas for future work.

2. HARDWARE DESIGN

In this chapter, the hardware design of each sensor and for the integrated system is detailed. The interface circuitry including the FPGA, the Parallel Interface, and relays are given. Furthermore, serial peripheral interface with their control logics are presented.

2.1 The System

The integrated system consists of temperature sensor, CO2 sensor and mother board. These components have been implemented at Smart Systems Incorporation [62].

The following is the diagram of the system. The major control unit is the PIC18 microcontroller, and the sensors that are connected to the PIC18 microcontroller via I2C bus interface. A FPGA chip is implemented on board to assist with balancing the speed difference between outside peripherals and the PIC18 microcontroller. Outside peripherals include a PC which is connected to parallel interface. Figure 2.1 shows the system block diagram. Figure 2.2 presents the different component of the smart system board, including the peripheral connected to the PC.

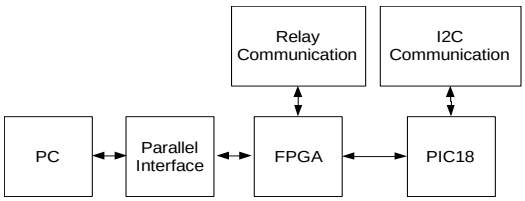


Figure 2.1 System Block Diagram

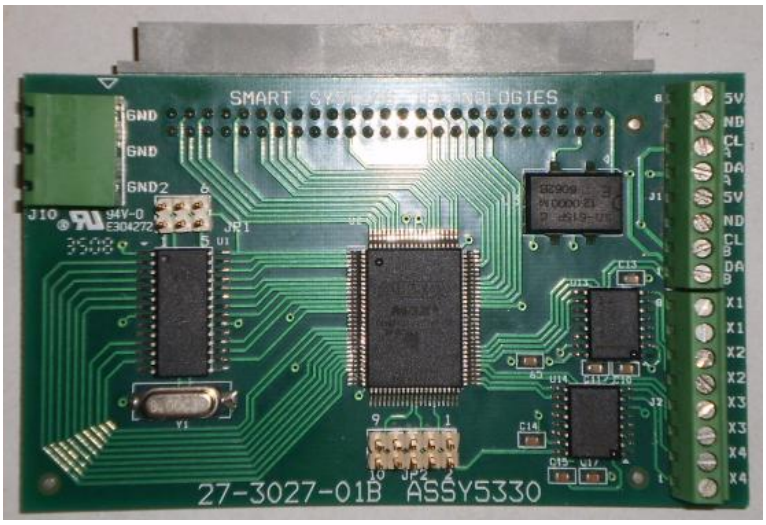


Figure 2.2 Smart System Board [62]

In Figure 2.2, the various components including the PIC18 microcontroller, the FPGA, the crystal oscillators, and the RS232 peripherals (relays) are all connected to the control circuits. Since the PIC18 microcontroller (8-bit processor) runs at 4MHz and the outside circuitries around at 50MHz, (32bit) and the FPGA is designed to allow interfacing the PIC18 microcontroller to the outside control circuits. Figure 2.3 shows the oscillators and the MIXIM chip used for the design.

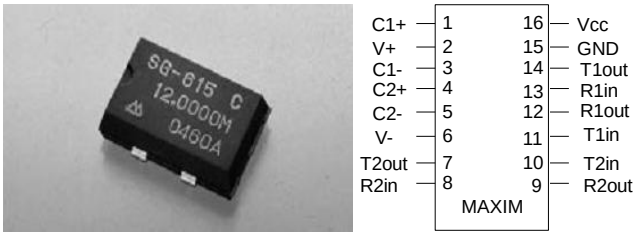


Figure 2.3 Oscillators (left), MAXIM (right) [63]

2.2 Smart System Modules

The schematics of the smart integrated system are given in Figure 2.4. It contains the FPGA chip and the PIC18 microcontroller.

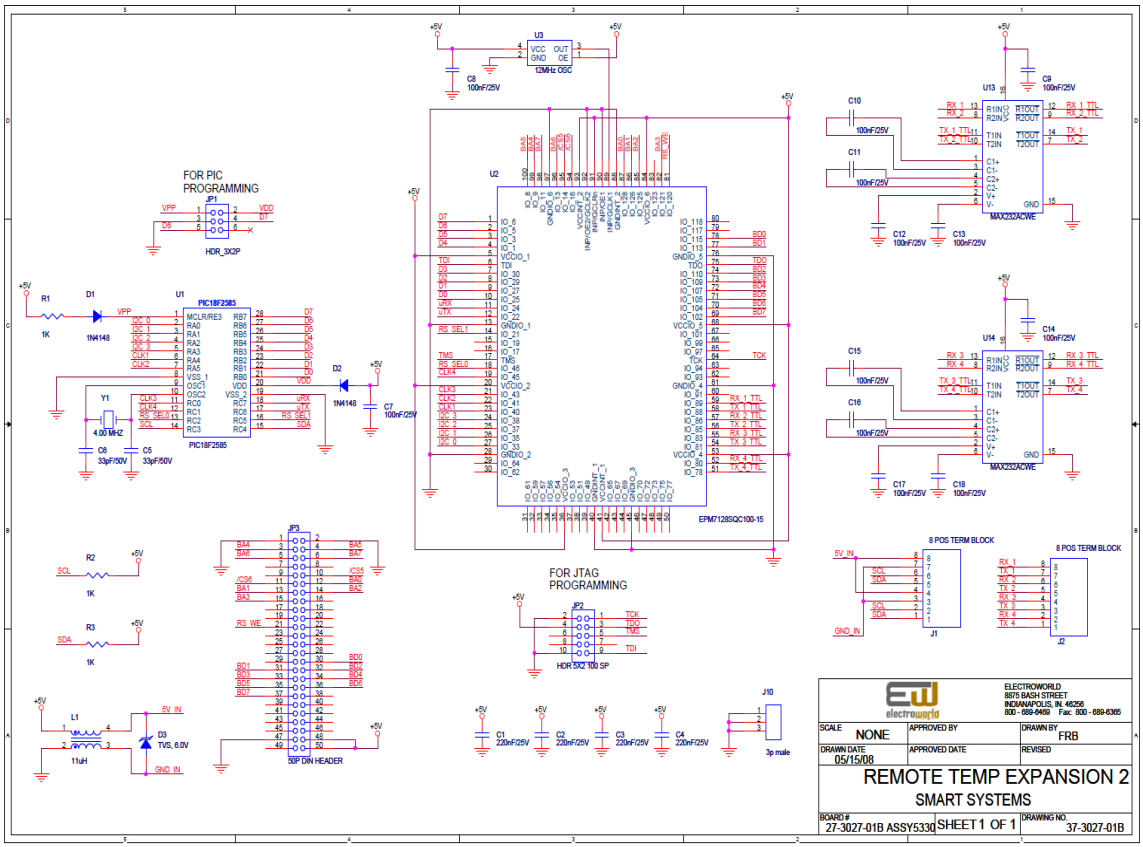


Figure 2.4 The Schematics of the Smart Integrated System [62]

2.2.1 FPGA

A 100-pin FPGA is implemented on board. It is an Altera MAX II FPGA. The software of this chip is copy right reserved for Smart Systems Incorporation. The FPGA on board support several types of communications protocols, such as I2C, SPI, parallel transmission, and RS232 (to be used to interface the microcontroller with relays through FPGA on board). Figure 2.5 gives the FPGA within the smart system board.

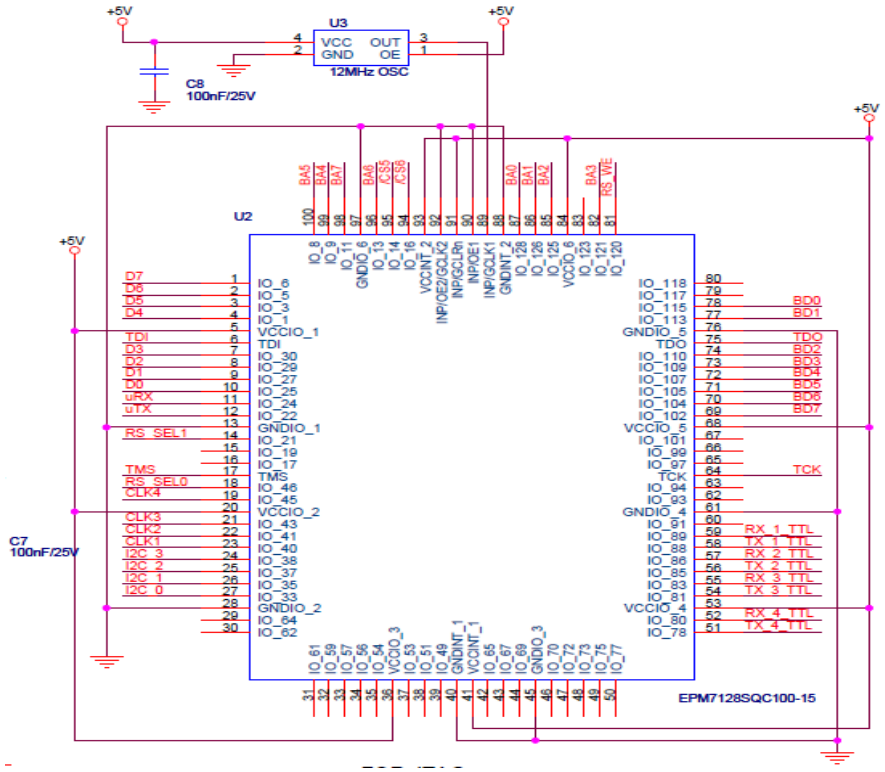


Figure 2.5 Diagram of FPGA on Smart System Board [62]

2.2.1.1 Inter-Integrated Circuit (I2C)

Inter-Integrated Circuit (I2C) is a bidirectional two-wire interface protocol, requiring only two bus lines; a serial data/address line (SDA), and a serial clock line (SCL). Each device connected to the I2C bus is software addressable by a unique address. The I2C bus is a multi-master bus where more than one integrated circuit (IC) capable of initiating a data transfer can be connected to it, which allows masters to function as transmitters or receivers.

This FPGA features a serial 8-bit bidirectional data transfer with a rate up to 100 Kbits per second. The sensor is configured as a slave device with I2C interfaced to serial EEPROMs. The I2C bus protocol of this FPGA has the following features:

- Both SDA and SCL are bidirectional lines.

- Data transfer is activated when the bus is free.
- The SDA line data is stable during the clock period.
- The SDA line transmission when the SCL is high indicates a start/stop condition.

Pin description functions of the PIC18 microcontroller is given by:

1) SDA (serial data/address line), a bidirectional port that can transmit and receive serial data. A wired-AND function can be performed via an open drain output pins from the SDA port.
2) SCL (Serial Clock Line), a bidirectional port is designed to synchronize the serial data transfer. Again a wired-AND function is created via an open drain output port pins of the SCL.
3) A2, A1, A0 (Slave Address Input) are programmable address bits and are set to 1010 by default.

The FPGA chip can perform the following operations.

- Start/stop operation
- Acknowledge operation;
- Device addressing operation;
- Byte Write Operation;
- Page Write Operation;
- Acknowledge Polling operation;
- Write Protection operation;
- Erase Operation;
- Full Erase (Device Slave Address Erase) operation;
- Sector Erase (Byte Address Triggered) operation;
- Sector Erase (A_2 Triggered) operation;
- No Erase operation;
- Read Operation (Current Address Read, Random Address Read, Sequential Read).

2.2.1.2 Serial Peripheral Interface (SPI)

Serial peripheral interface (SPI) is a four-pin serial communication subsystem included within the Motorola 6805 and 68HC11 series microcontrollers. It allows the microcontroller unit to communicate with peripheral devices, and is also capable of inter-processor communications in a multiple-master system.

The SPI bus consists of masters and slaves. The master device initiates and controls the data transfers and provides the clock signal for synchronization. The slave device responds to the data transfer request from the master device. The master device in a SPI bus initiates a service request with the slave devices responding to the service request. SPI Interface Signals are

1) SI Serial Data Input that receives data serially.
2) SO Serial Data Output that transmits data serially.
3) SCK Serial Data Clock that is produced from the master device to synchronize the data transfer.
4) nCS Chip Select that activates low signal that enables the slave device, and receiving or transferring data from the master device.

Data transmitted to the SI port of the slave device is sampled by the slave device at the positive SCK clock edge, and transmitted from the slave device through SO at the negative SCK clock edge. When nCS is asserted, it means the current device is being selected by the master device from the other end of the SPI bus for service. When nCS is not asserted, the SI and SCK ports should be blocked from receiving signals from the master device, and SO should be in High Impedance state to avoid causing contention on the shared SPI bus. All instructions, addresses, and data are transferred with the MSB first and start with high-to-low nCS transition.

2.2.1.3 Parallel Interface

The parallel interface signals are 16-bit data Input; 16-bit data Output; for Address Register, READ Instruction Signal, WRITE Instruction Signal, ERASE Instruction Signal; BUSY Signal; Data Valid. Details of these signals are given in Appendix I. This port on the smart system board can be used to interface the microcontroller and FPGA with the outside control circuits and PCs.

2.2.1.4 RS232 Communications

RS232 communication is used to enable the microcontroller to connect it to relays. The baud rate can be selected based on data format. Software can be developed to implement serial data communications using I/O pin of a microcontroller. It is a form of asynchronous data transmission. That is preceded with a start bit. Universal asynchronous receiver transmitter (UART) software library is used for RS232-based serial communications. In this case, data sent in serial format over the cable bit by bit. Transmission starts with the first bit sent at logic 0, 7 or 8 data bits and an optional parity bit will then follow. The last bit that will be sent is the stop bit at logic high. Figure 2.6 shows an example on how character "A" can be sent using serial communications.

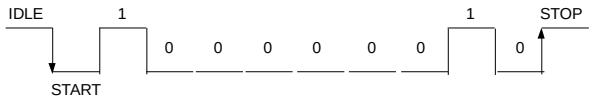


Figure 2.6 Sending Character "A" in Serial Communication

RS232 protocol is powered by +/-12 Volt, however PIC controllers normally work at 5 volt-system. Therefore, a voltage convertor is required to interface with the RS232 interface and the microcontroller. The MAXIM232 is a typical chip that is manufactured by MAXIM Incorporation for use in this purpose.

The importance of the relay communications exists in extending communications in two ways or bidirectional to improve the throughput of the system. This can be performed with both decode and forward (DF), and amplify and forward (AF) schemes [64].

2.2.2 PIC18 Microcontroller

Figures 2.7 and 2.8 give the I/O pins and the schematics of the microcontroller respectively. In this diagram (Figure 2.8), only PORT C and The off-chip 4-MHz oscillator is used. Figure 2.9 gives the microcontroller of the PIC182585, it consists of the following units:

1) Peripherals and timers
2) Oscillator and control bits
3) Arithmetic unit
4) Program unit
5) Ports

Appendix 1.2 gives the detail of the microstructure. Three memory blocks are used in this hardware: enhanced flash program memory, data memory, EEPROM Data Memory. Data and program memory units allow concurrent access of the rest of the blocks.

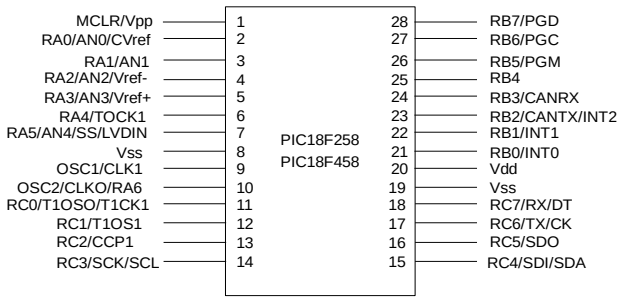


Figure 2.7 PIC18 Microcontroller [65]

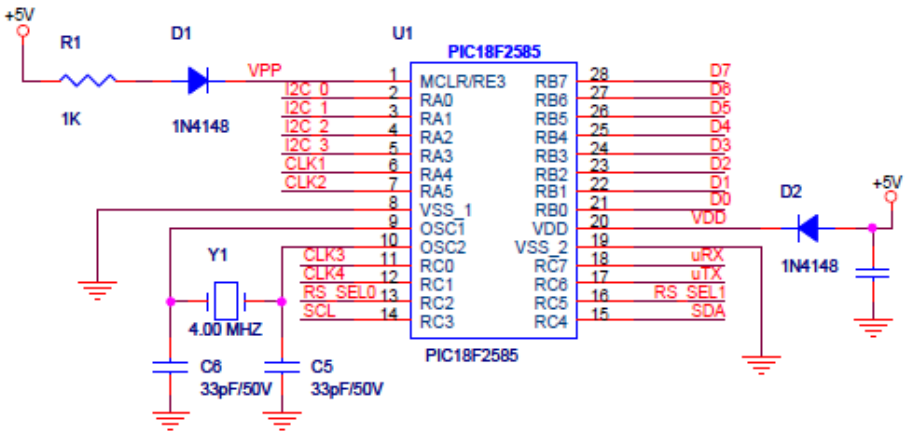


Figure 2.8 PIC18 Microcontroller Schematics [62]

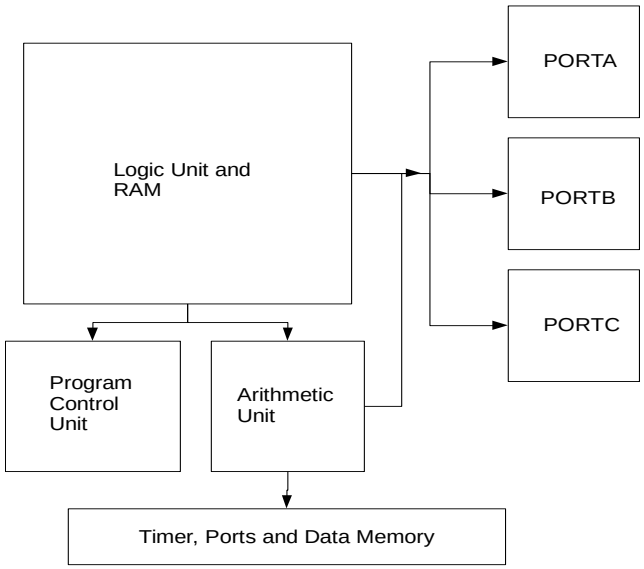


Figure 2.9 The Microstructure of the PIC182585 Microcontroller [65]

2.2.2.1 PORT C

PORT C is an I/O port between the microcontroller and the sensors. The PIN diagram of PORT C is given in Appendix 1.3. Its Pin functions and registers are given in tables 2.1 and 2.2 respectively. PORT C is a multi-functional port, with latches on all its

pins. When output (TRIS C) becomes zero, PORT C will send a signal to the sensor, and the latch (LAT C) register will be open to store the value of the output data. When the input of the latch registers are turned off (by a software command), the TRIS C will go high, and PORT C pins will receive signals from the sensor, and will send acknowledging receipt of the signal to SSPBUF.

Table 2.1 PORT C Pins Functionary [65]

Name	Bit#	Buffer Type	Function
RC0/T1OSO/ T1CK1	bit0	ST	Input/output port pin, Timer1 oscillator output or Timer1/Timer3 clock input.
RC1/T1OSI	bit1	ST	Input/output port pin or Timer1 oscillator input.
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	bit3	ST	Input/output port pin or Synchronous Serial clock for SPI/I2C.
RC4/SDI/SDA	bit4	ST	Input/output port pin or SPI Data in (SPI mode) or Data I/O (I2C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit6	ST	Input/output port pin, Addressable USART Asynchronous Transmit or Addressable USART Synchronous Clock.
RC7/RX/DT	bit7	ST	Input/output port pin, Addressable USART Asynchronous Receive or Addressable USART Synchronous Data.

Table 2.2 PORT C Registers [65]

Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Value on POR, BOR	Value on all other RESETs
PORT C	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
LAT C	LAT C Data Output Register								xxxx xxxx	uuuu uuuu
TRIS C	PORT C Data Direction Register								1111 1111	1111 1111

2.2.2.2 PIC18 Microcontroller Transmission Mechanism

The transmission process is a serial interface between peripheral and devices. Peripherals may include EEPROMs, shift registers, display drivers, and A/D converters. There are two modes of operation: serial peripheral interface (SPI) and inter-integrated circuit (I2C). Both modes are equipped with master and slave mode addresses. The SPI mode allows 8 bits of data transmitted and received simultaneously, while I2C mode uses two pins for data transfer: RC3/SCK/SCL and RC4/SDI/SDA. These pins must be configured as inputs or outputs through the TRIS C<4:3> bits.

2.3 The Sensors

In this study, temperature sensors, humidity sensors, and CO2 sensors were studied for communications to the embedded microcontroller systems. Hardware and software where emphasized for the temperature and CO2 sensors. The integrated system was designed for HVAC applications.

2.3.1 The Temperature Sensor

Various temperature sensors including thermocouples [40, 66, 67], infrared detectors, thermal-resistance [68, 69, 70] devices and thermistors [71, 72, 73] were investigated in the sense of the programmability, interfacing capability, accuracy, precision, and temperature range suitability for HVAC applications. Thermal sensors have issues with thermal distribution that is based on physical dimensions and shapes. Thermistors have issues with efficiency of energy transfer from ballistic phonons to electrons in the sensors. This may lead to nonlinearity in the permittivity and specific heat capacity of the device [72]. Infrared devices are more suitable for noninvasive approaches for thermal structures. Semiconductor temperature sensors built on standard CMOS technology can be interfaced to microcontrollers. They provide high sensitivity with low power dissipation. LM76 is among the semiconductor sensors that provide high accuracy and proper interfacing with the microcontroller. They require 3.3-5 volt power supply, serial bus interface, 12-bit+sign output, and full-scale range of over 127 degree. LM76 digital programmable sensor was chosen for the study. The selection was based on its detecting temperature range from -10 to 45℃. Other types of these sensors can operate at higher temperatures from 70 to 100℃. Therefore, LM76 can operate from -10 to 100℃ with 12-bit plus a sign precision, with an accuracy of 0.0625℃. Figure 2.10 gives an image of the LM76 digital temperature sensor, showing the peripherals associated with it. The schematics of the temperature sensor and its set-up with the CPU and power supply are given in Figure 2.11 and 2.12 respectively.

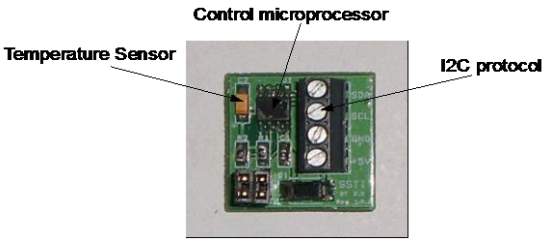


Figure 2.10 Digital Temperature Sensor LM76 [74]

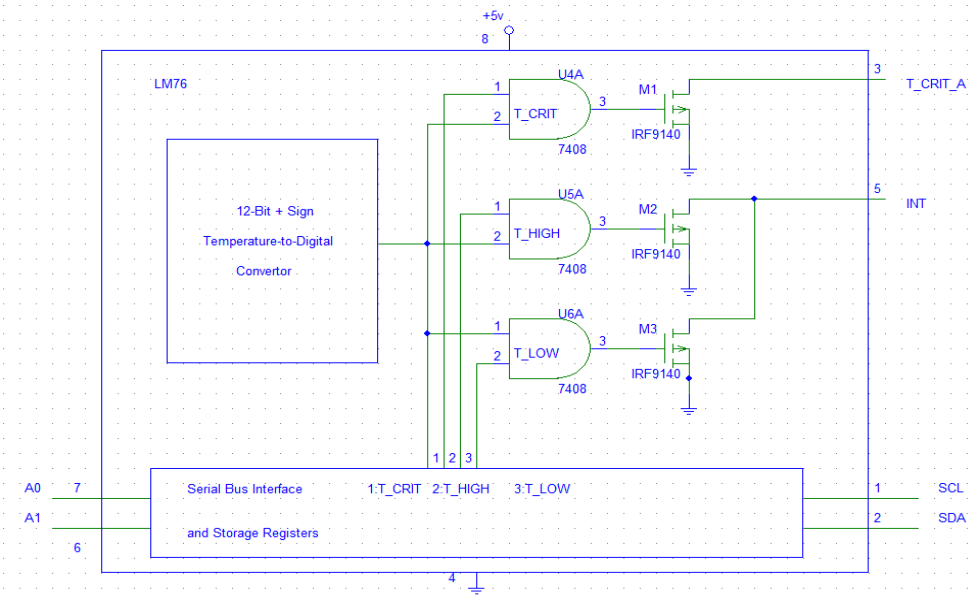


Figure 2.11 LM76 Schematics [74]

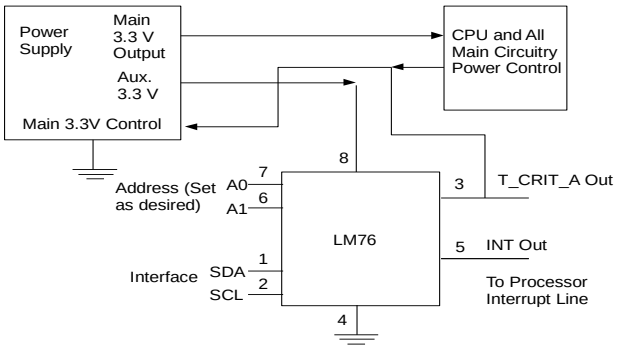


Figure 2.12 LM76 Temperature Sensing System [74]

Figure 2.13 shows the pin diagram of the sensor, while Figure 2.14 gives the register stack associated with the temperature sensor. When programming the sensor, SDA and SCL are used to receive the data, T_CRIT_A is used to set the critical temperature in the sensor system. A1 and A0 were tied to low, and the INT is used for interrupting purposes. Pointer registers are used to store addresses of the referring

registers. Configuration registers are used to set the operation mode of the temperature sensor. There are four registers allocated to store T_{HYST} (minimum temperature precision for making appropriate interrupt decision), T_{CRIT} (critical temperature in temperature system), T_{LOW} (minimum sustainable temperature), and T_{HIGH} (maximum sustainable temperature). A typical circuitry to check out temperature change is shown in the Figure 2.15:

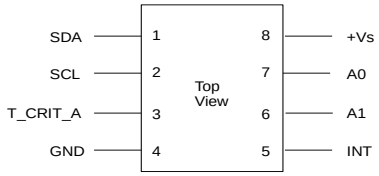


Figure 2.13 The Pin Diagram of the Temperature Sensor [74]

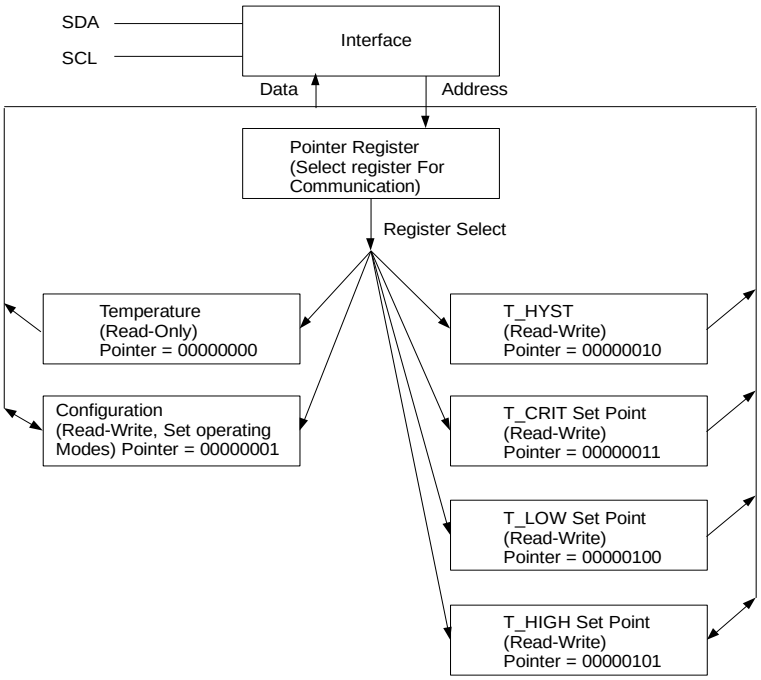


Figure 2.14 Register Stack of the Sensor [74]

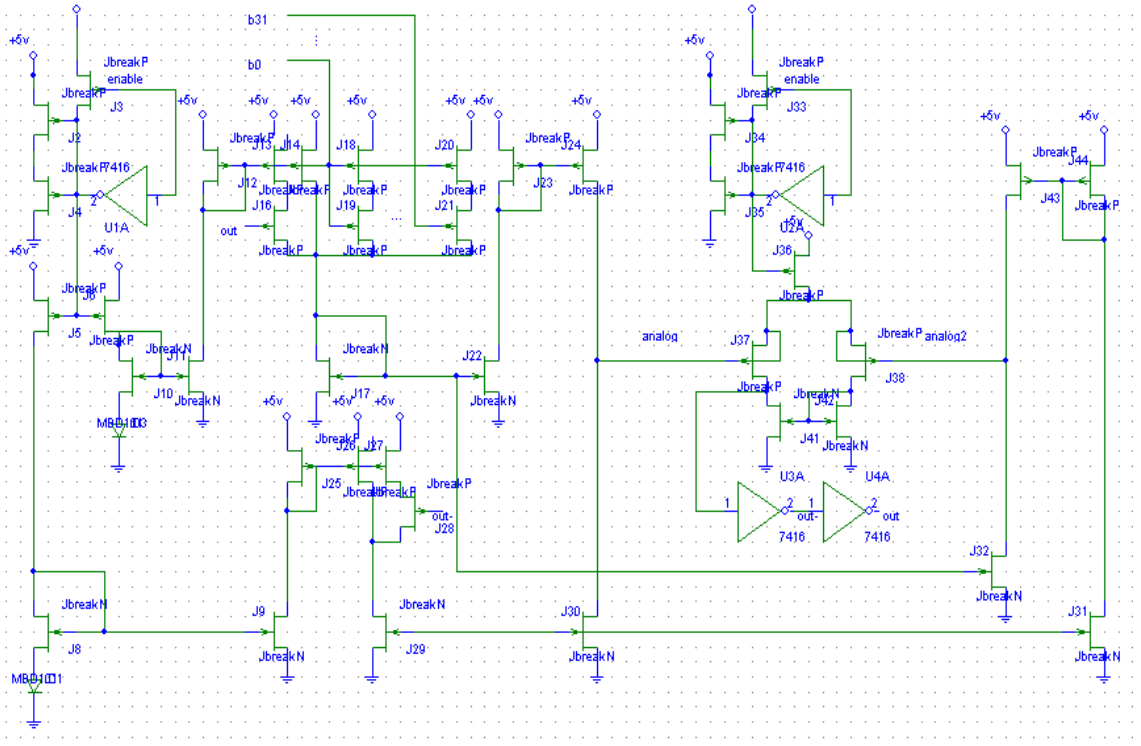


Figure 2.15 Temperature Change Circuitry

2.3.2 The Humidity Sensor

Humidity sensor technology is based on the phenomenon of some materials that change their physical or chemical reaction in response to the humidity of the surroundings. These reactions include frequency change and transmission speed. There are many types of humidity sensors:

- Flexible humidity sensor [75, 76, 77],
- Evaporative humidity sensor [78],
- Dew point humidity [79, 80, 81],
- Resistance humidity sensor [82],
- Capacitance humidity sensor [83, 84],
- Optical-electronic humidity sensor [85],
- Surface Acoustic Wave humidity sensor [86],
- Microwave humidity sensor [87],

- Diode humidity sensor [88],
- Absorption humidity sensor [89], and
- Infrared humidity sensor [90].

Flexible humidity sensors lack precision of data results. Evaporative humidity sensor is highly accurate, however, due to the large size and energy consumption. It cannot be applied in portable applications. Dew point humidity sensors utilize the characteristic that it transforms ice into humidity after adding factors of temperature and air pressures. This makes it inappropriate for HVAC application. Resistance humidity sensors use materials that change their resistances based on vapor abortion. This influences the temperature greatly. Capacitance humidity sensors use materials that change their capacitance based on water absorption. Compared with resistance humidity sensors, they feature high precision, low-power, and small size devices. Furthermore, their fabrication is compatible with CMOS technology, and therefore can be integrated into a microchip. This can be made advanced and built with MEMS to be pursued for different applications. This is the kind of sensors that are recommended to use in the HVAC application. Other types of sensors including optical humidity sensors, and surface acoustic humidity sensors lack electro-magnetic interference and sound speeds respectively.

A Digitally controller humidity sensor system is given by Figure 2.16, while typical solid-state humidity sensors and their circuitries are given in Figures 2.17 and 2.18 respectively. At Smart Systems Incorporation is still working for the ASIC design, incorporating this solid-state humidity sensor into the embedded controller. Therefore, integrating the humidity sensors into the HVAC system is reserved for future consideration.

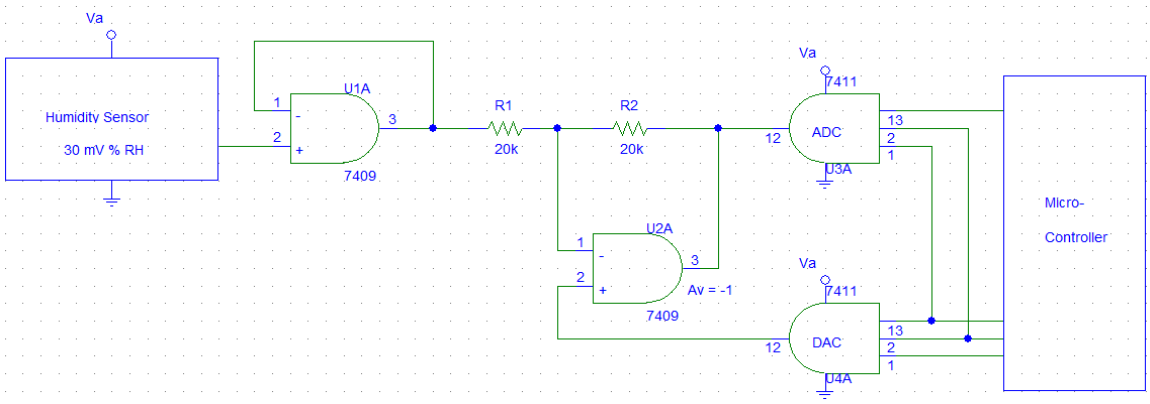


Figure 2.16 Typical Humidity Sensor Diagram

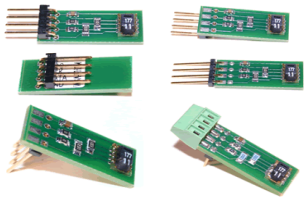


Figure 2.17 Solid-state Humidity Sensor

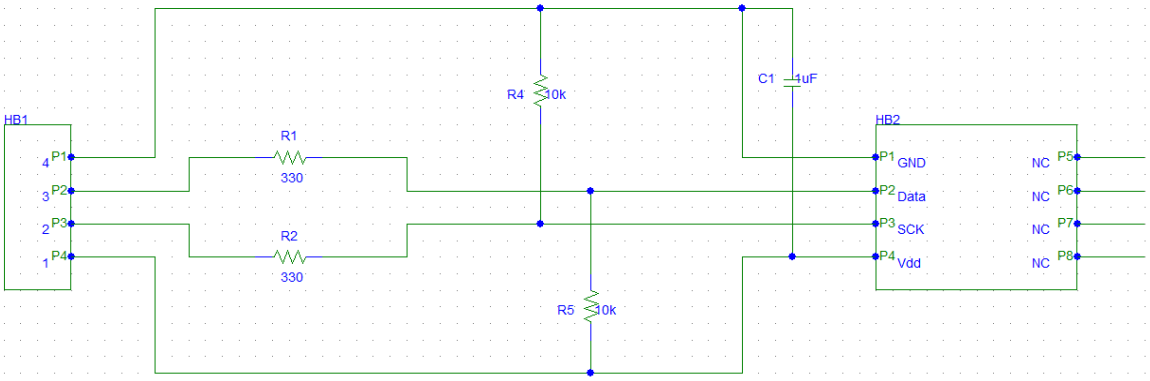


Figure 2.18 Typical Humidity Sensor Circuitry

2.3.3 The CO2 Sensor

There are different methods for measuring CO2. This is based on the utilization of the characteristics of sensitive materials to the density of the CO2 [91, 92, 93]. CO2 sensors may include electrochemical based, infrared based, and metal-oxide based sensors. The latter one features low-cost and long term durability. In addition, electro-acoustic elements can be used for sensing CO2 concentration. Electronic circuits, such as PLL (phase lock loop) oscillators are utilized to provide both voltage and frequency outputs of the sensor [91]. Infrared CO2 sensors utilize the feature that the infrared wavelength will be correlated with the CO2 density after digital filtering and self-compensation. This kind of sensor was used in our research because of its compatibility to CMOS computer system interfacing. K22L0 from Senseair Corporation [94] was used for our design. Figures 2.19 and 2.20 give the image of the CO2 sensor board and its schematics respectively.

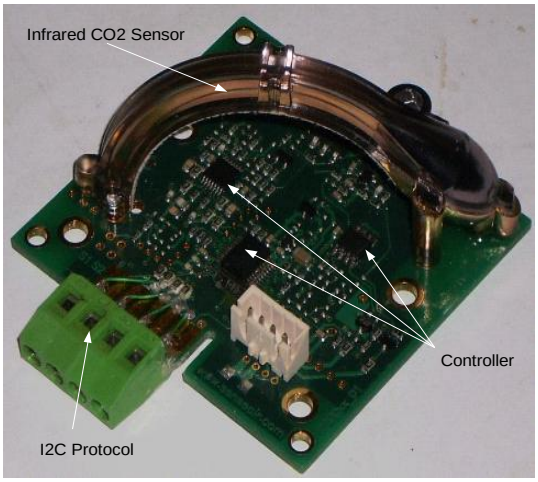


Figure 2.19 CO2 Sensor K22L0 [62, 94]

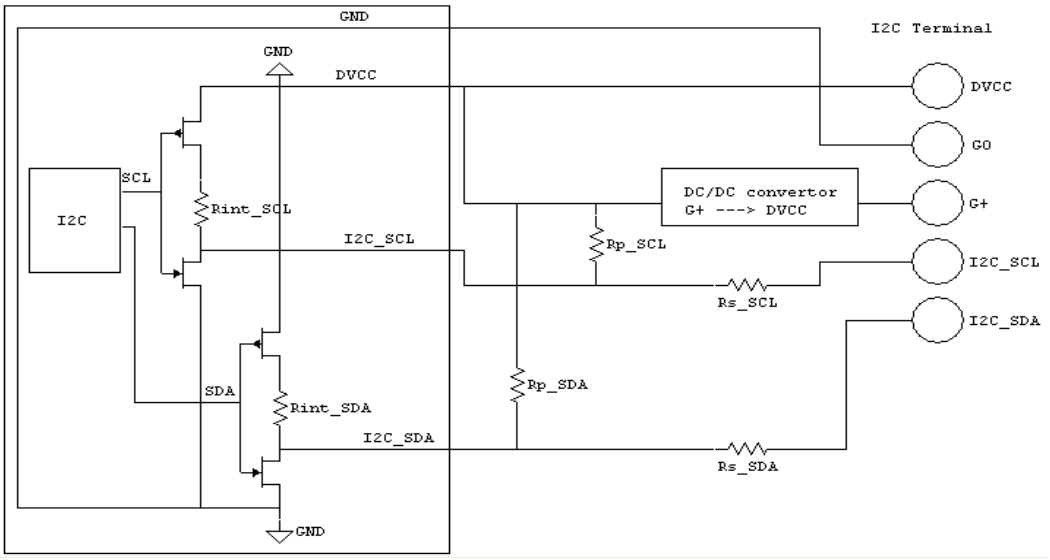


Figure 2.20 Schematics of the CO2 Sensor System on Board [62, 94]

The CO2 sensor uses I2C-bus that has 2 bi-directional lines SCL and SDA. The output stages have an open-drain or open-collector to perform the wired AND function. Figure 2.2, Figure 2.21 and Figure 2.22 give the electrical schematics with their details respectively.

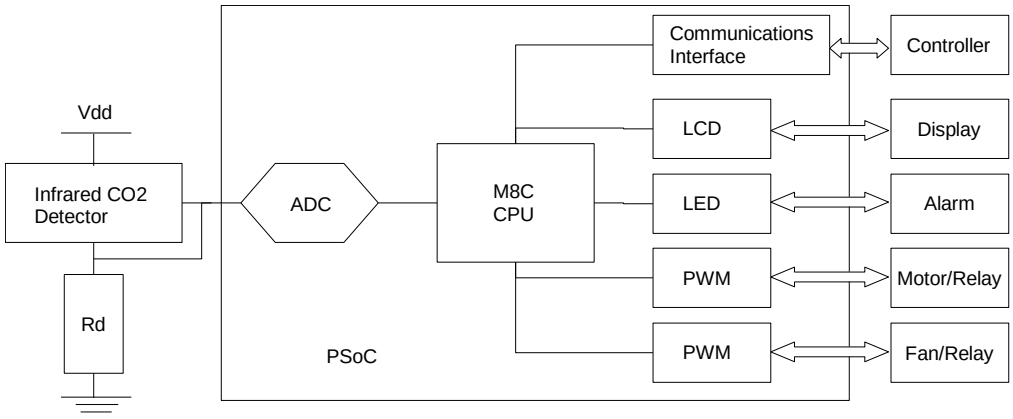


Figure 2.21 Typical Schematics of Digital CO2 Sensor

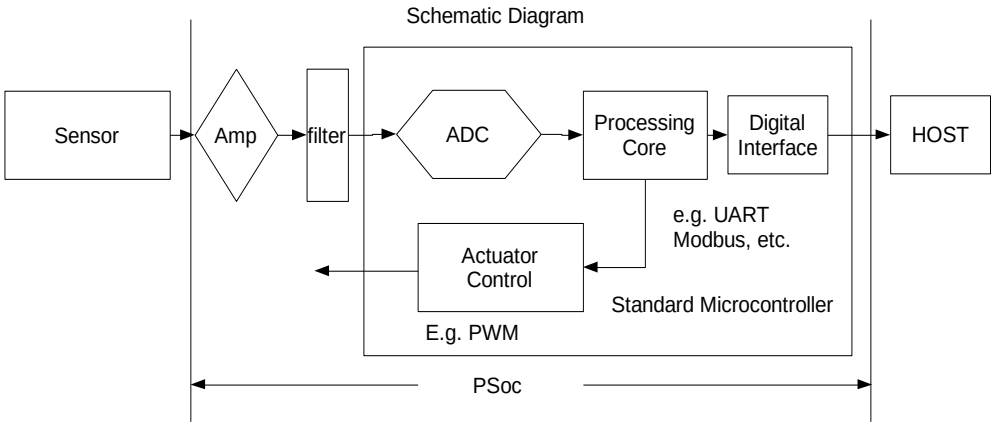


Figure 2.22 Detailed Explanation of Typical CO2 Sensor Schematics

As per our discussions, we have selected LM76 temperature sensor and K22L0 for CO2 sensor integrated with the embedded controller PIC182585 microprocessor and I2C protocol for the prototype of our study. The software that controls the hardware activities is presented in Chapter 3.

3. SOFTWARE MODELING AND SIMULATION

This chapter details the MPLAB PICC software used for the communications between the embedded system controller and the sensors.

3.1 MPLAB software

MPLAB ICD2 from Microchip Company [95] (see Figure 3.1) is used for the software design. MPLAB IDE is the software for MPLAB ICD2. It is a Windows based software program that runs on a PC to develop applications for Microchip micro-controllers and digital signal controllers. It is called an Integrated Development Environment, or IDE, because it provides a single integrated "environment" to develop code for embedded microcontrollers. MPLAB IDE is a "wrapper" that coordinates all the tools, including graphical user interfaces.

Figure 3.1 MPLAB ICD2 Hardware

3.2 PIC18 Microcontroller

PIC182585 micro-controller is used in this software project. It supports several types of communication interfaces, such as SPI mode, I2C mode, CAN mode, etc. In this software program, I2C mode is used.

3.2.1 Registers

In order to implement I2C protocol, six relevant registers in PIC182585 micro-controller are used, those registers are MSSP Control Register1 (SSPCON1); MSSP Control Register2 (SSPCON2); MSSP Status Register (SSPSTAT); Serial Receive/Transmit Buffer (SSPBUF); MSSP Shift Register (SSPSR), which is not directly accessible and MSSP Address Register (SSPADD).

SSPCON, SSPCON2 and SSPSTAT are the control and status registers in I2C mode operation. The SSPCON and SSPCON2 registers are readable and writable. The lower 6 bits of the SSPSTAT are read only. The upper two bits of the SSPSTAT are read/write. SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from. SSPADD register holds the slave device address when the SSP is configured in I2C Slave mode. When the SSP is configured in Master mode, the lower seven bits of SSPADD act as the baud rate generator reload value. In receive operations, SSPSR and SSPBUF together, create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set. During transmission, the SSPBUF is not double-buffered. A write to SSPBUF will write to both SSPBUF and SSPSR. The MPLAB user interface is shown in Figure 3.2.

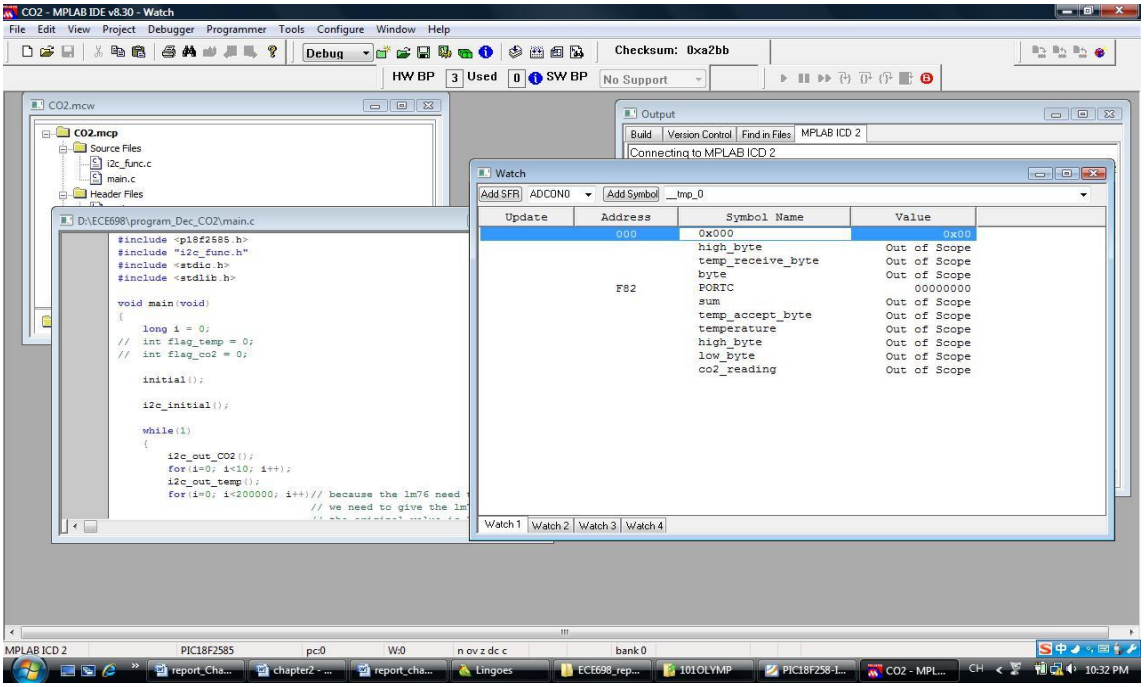


Figure 3.2 MPLAB IDE Desktop

3.2.2 PIC182585 Microcontroller Software Program Format

In this software development, I2C Master Mode is implemented. Master mode is enabled by setting and clearing the appropriate SSPM bits in SSPCON1 and by setting the SSPEN bit. In Master mode, the SCL and SDA lines are manipulated by the MSSP hardware. Master mode of operation is supported by interrupt generation on the detection of the START and STOP conditions. The STOP (P) and START (S) bits are cleared from a RESET, or when the MSSP module is disabled. Control of the I2C bus may be taken when the P bit is set or the bus is IDLE, with both the S and P bits clear.

All I2C bus operations are based on START and STOP bit conditions. Once Master mode is enabled, the user has six options:

1. Assert a START condition on SDA and SCL,
2. Assert a Repeated START condition on SDA and SCL,
3. Write to the SSPBUF register, initiating transmission of data/address,

4. Configure the I2C port to receive data,
5. Generate an Acknowledge condition at the end of a received byte of data, and
6. Generate a STOP condition on SDA and SCL.

The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition, or with a Repeated START condition. Since the Repeated START condition is also the beginning of the next serial transfer, the I2C bus will not be released. In Master Transmitter mode, serial data is output through SDA, while SCL outputs the serial clock. The first byte transmitted contains the slave address of the receiving device (7 bits) and the Read/Write (R/W) bit. In this case, the R/W bit will be logic '0'. Serial data is transmitted 8 bits at a time. After each byte is transmitted, an Acknowledge bit is received. START and STOP conditions are output to indicate the beginning and the end of a serial transfer. The master mode transmission diagram is given in Figure 3.3.

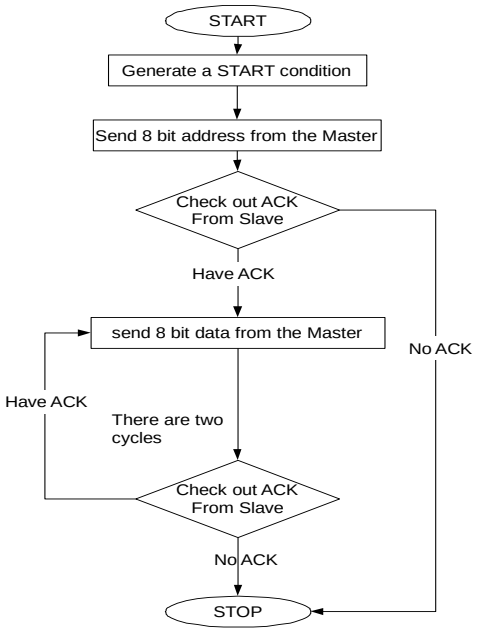


Figure 3.3 I2C Master Mode Transmission Diagram

In Master Receive mode, the first byte transmitted contains the slave address of the transmitting device (7 bits) and the R/W bit. In this case, the R/W bit will be logic '1'. Thus, the first byte transmitted is a 7-bit slave address followed by a '1' to indicate receive bit. Serial data is received via SDA, while SCL outputs the serial clock. Serial data is received 8 bits at a time. After each byte is received, an Acknowledge bit is transmitted. START and STOP conditions indicate the beginning and end of transmission. Figure 3.4 gives the master mode reception diagram.

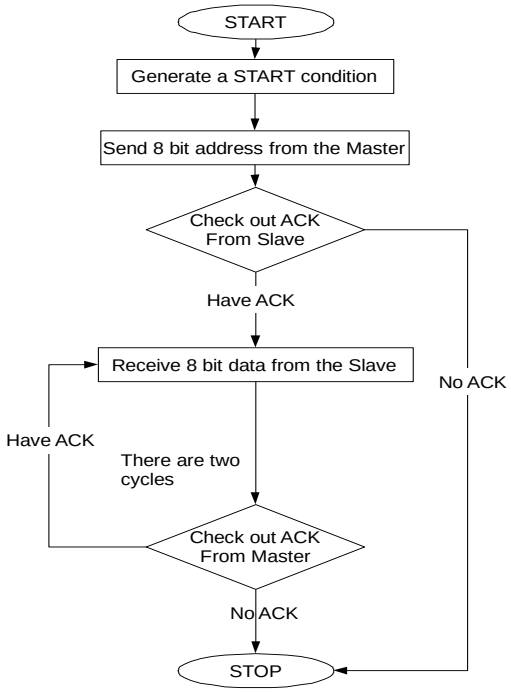


Figure 3.4 I2C Master Mode Reception Diagram

3.3 FPGA

The VHDL code used to match the clock frequencies between the controller (master) and the sensor (slave) was written by Smart Systems Incorporation, and it is right protected for the company [62]. This is necessarily to interface the master and slave units for the integration of the system. The FPGA chip is connected to parallel bus for the

communications with the control unit. The FPGA chip also is connected to 2 relay ports in order to expand the number of sensors to be attached to the system. Figure 2.2 (shown in chapter 2.1) shows the connections of the FPGA with the parallel bus and the relay ports.

3.4 Software Models of the Embedded System

The embedded system has several functionalities: request for data from sensors, receive data from sensors, process raw data and initiate control signals. The embedded sensor system diagram consists of temperature sensor LM76, and CO2 sensor K22L0. Requesting data, receiving data, raw data processing, and initiating control signals will be illustrated later. A complete description of the whole system will be presented. Figure 3.5 shows the embedded sensor system with handshaking communications and feedback control circuits.

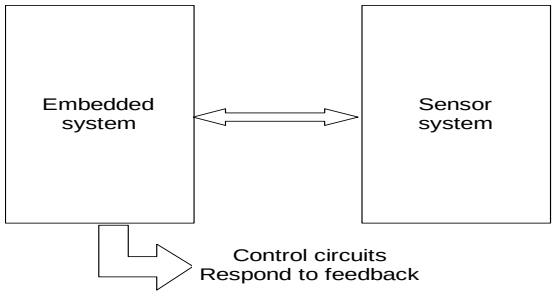


Figure 3.5 Embedded Sensor System

3.5 Temperature Sensor Software

The first step of the temperature sensor software is to set up temperature LM76 in the sensor system. This includes compiling the software, and downloading to the board, and start simulation. A start bit is then generated by the master and the address byte will be outputted, followed by an ACK signal back the board.

Figure 3.6 details the software activity. Based on the ACK condition, if the ACK exists, an 8-bit data will be sent to the sensor. A software loop is used to guarantee the correction of the transmission. Upon the completion of this transmission step, a No-ACK signal will be outputted, leading to the end of the transmission process. For testing purposes, the first cycle that was sent to the slave address was 0x91, with a final bit of '1', which indicates this is a read sequence. At the end this cycle, an ACK from the slave was transmitted to the master indicating if the transmission has been successfully finished.

The second and third cycle is to receive the MSB (most significant byte) and LSB (least significant byte) of the temperature data. After the MSB has been transmitted, the Master will send an ACK back to Slave, indicating that the MSB has been received successfully. When the LSB is transmitted, no ACK will be sent to Slave, and the STOP condition will be generated.

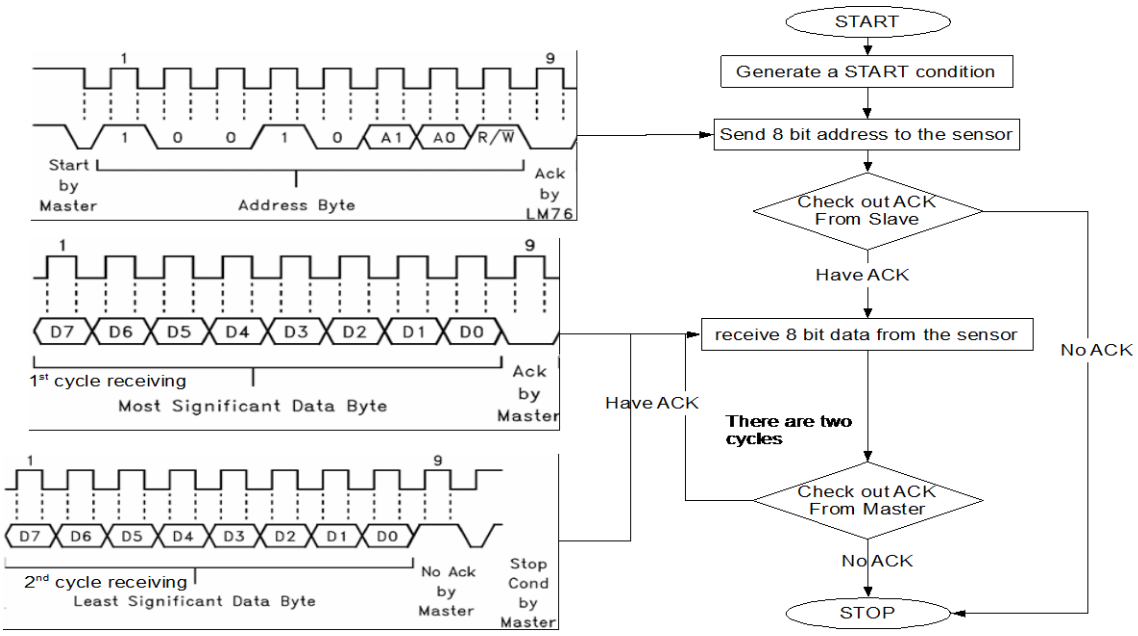


Figure 3.6 Read Data from LM76

Below is a typical section of the program that was used to implement the temperature sensor reading. Timeout protection has not been added to the software program. This is due to the continuous communication between the sensor and the processor, seeking stable communications and checking for the ACK signal. Figure 3.7 gives the temperature sensor reception process.

```
      i2c_initial();// enable I2C functionalities
// define START I2C
      i2c_start();// start the I2C bus transmission protocol
//  send the 1st byte
      TRISC &= 0xE7;  //i2c output
      *temp6 = 0x91;// send the Slave address
      transmission(temp6);
      TRISC |= 0x10;  //i2c input

      PORTCbits.SCL = 1;// try to get ACK from Slave
      for(i = 5; i > 0; i --);
      PORTCbits.SCL = 0;
      for(i = 5; i > 0; i --);

      TRISC |= 0x10;  //i2c input

      for(k = 0; k < 2; k ++)
      {
            temp_accept_byte[k] = receive_byte();
            if(k < 1)
                  ACK_master(); // After MSB reception,
            Else                        // send out ACK from Master
                  No_ACK_master();      // After LSB reception
      }                                 // send out no-ACK from Master
      TRISC &= 0xE7;  //i2c output
      i2c_stop();
```

Figure 3.7 Temperature Sensor Source Code in Receiving Cycle

In real time experiments, LM76's respond may not be as fast as the PIC18 microcontroller, so a long delay is needed. Figure 3.8 is a typical program to acquire temperature data from LM76.

```
while(1)
{
      i2c_out_temp();
      for(i=0; i<200000; i++)// a long delay for LM76 to respond
      {
            ;
      }
}
```

Figure 3.8 Temperature Sensor Sampling Cycle Source Code-Main Program

3.6 CO2 Sensor Software

The second step in the process is to set up the CO2 sensor software in the sensor system. Figure 3.9 is the illustrative diagram for this step.

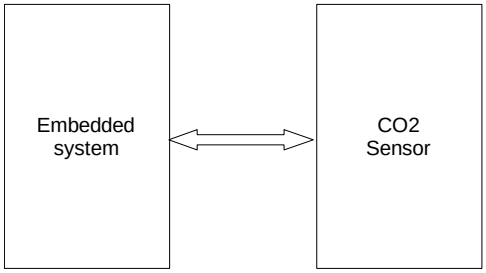


Figure 3.9 CO2 Sensor Communications

A CO2 sensor communication session consists of 2 data transfer slots with one wait slot between them. The first data transfer, named Request, contains command and eventual data to write to the sensor. The wait time slot is required to let the sensor recognize and execute the command and prepare for the response. The second data transfer, named Response, contains the check for completion of the command execution and read data from sensor. The “complete” bit is checked to verify that the command is successfully executed. The master (embedded controller) may need to repeat attempts to read response until a valid frame with “complete” bit set to high is received or timeout is occurred. Busy sensors may be indicated by absence of acknowledge bit as well. Figure 3.10 shows the transmitting/receiving sequence.

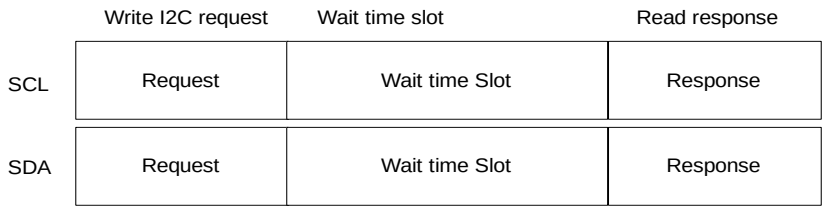


Figure 3.10 Transmitting/Receiving Sequence

The Microstructures and messages sent and received are summarized in table 3.1, and detailed explanation, each bit in the data sequence is detailed in table 3.2. The encoding of the response sequences (when completed or uncompleted) are given in tables 3.3 and 3.4 respectively.

Table 3.1 Microstructure of the Data Frameworks

Request:

I2C START condition	7-bit I2C Address	Read/Write Bit	Command High Nibble	Number of bytes Low nibble	RAM Address	Checksum	I2C STOP condition
	sensor address	0 (write)	0x2	0..0xF			
	1 byte		1 byte		2 bytes	1 byte	

"Read Complete" Response

I2C START condition	7-bit I2C Address	Read/Write Bit	Command High Nibble	Status bit	Read Data	Checksum	I2C STOP condition
	sensor address	1 (Read)	0x21				
	1 byte		1 byte		2 bytes	1 byte	

"Read Incomplete" Response

I2C START condition	7-bit I2C Address	Read/Write Bit	Command High Nibble	Status bit	All Other Bytes	I2C STOP condition
	sensor address	1 (Read)	0x20		0x20	
	1 byte		1 byte			

Table 3.2 Encoding of Fields in Request Sequence

Byte position	Bits in byte	Field	Value	Interpretation
0	7:1	7-bit I2C Slave Address	0x00 to 0x7F	I2C address. Default Sensor Address is 0x68.On Point-to-Point connections Master also can use address 0x7F - “any sensor”.
	0	Direction bit	0 or 1	Read/Write bit encoding Read = 1, Write = 0.
1	7:4	Command	1,2,3,4	0x1 - Write RAM;0x2 - Read RAM;0x3 - Write EE;0x4 - Read EE;0x0, 0x5..0xF – Reserved for future use.
	3:0	Number of Data bytes	0..15	The number of bytes to read/write. Value range from 1 to 15; 0 means 16 bytes. E.g. for write 1 byte to sensor RAM this byte should be 0x11 (Command = WriteRAM, NbrOfBytes = 1)
2:3		RAM Address, MSB first		Address in sensor’s RAM.
4:(3+N)		N bytes of Data (N = 2)		Data for Writing (N = 2 bytes). This field is present only for commands “Write RAM” and “Write EEPROM”. For commands “Read RAM” and “Read EEPROM” N=0.
4+N		Checksum		Arithmetic sum of the bytes sent (not including first byte with address and direction bit).

Table 3.3 Encoding of the Response (Response Completed) Sequence

Byte position	Bits in byte	Field	Value	Interpretation
0	7:1	7-bit I2C-Slave address	0x68	I2C address. Default Sensor address is 0x68. On Point-to-Point connections Master also can use address 0x7F-“any sensor”.
	0	Direction bit	0 or 1	Read/Write bit encoding Read = 1, Write = 0.
1	7:4	Command	1,2,3,4	0x1 - Write RAM;0x2 - Read RAM;0x3 - Write EE;0x4 - Read EE;0x0, 0x5...0xF – Reserved for future use.
	3:1	Reserved	000	Reserved for future use, should be 000 for compatibility.
	0	Complete/Incomplete bit	1 (completed)	Complete/Incomplete bit. 1-Complete, 0-Incomplete.
2:(1+N)		N bytes of data (N = 2)		Data read (N = 2 bytes). This field is present only for commands “Read RAM” and “Read EEPROM”. For command “Write RAM” and “Write EEPROM” N=0.
2+N		Checksum		Arithmetic sum of the bytes sent (not including first byte with address and direction bit).

Table 3.4 Encoding of the Response (Response Uncompleted) Sequence

Byte position	Bits	Field	Value	Interpretation
0	7:1	7-bit I2C-Slave address	0x68	I2C address. Default Sensor address is 0x68. On Point-to-Point connections Master also can use address 0x7F-“any sensor”.
	0	Direction bit	0 or 1	Read/Write bit encoding Read = 1, Write = 0.
1	7:4	Command	1,2,3,4	0x1 - Write RAM;0x2 - Read RAM;0x3 - Write EE;0x4 - Read EE;0x0, 0x5 ... 0xF – Reserved for future use.
	3:2	Reserved	00	Reserved for future use, should be 00 for compatibility.
	1	Invalid Data	0 or 1	External EEPROM has page size of 16 bytes. If data in command “Write EEPROM” crosses bound of the EEPROM page, bit “Write error” is set and write command is not executed.
	0	Complete/Incomplete bit	0 (Incompleted)	Complete/Incomplete bit. 1-Complete, 0-Incomplete.
2:(N)		N bytes of data (N = 2)		N = expected number of Data bytes (N=2) for commands “Read RAM” and “Read EEPROM”.
2+N		Checksum		Arithmetic sum of the bytes sent (not including first byte with address and direction bit).

The I2C communication sequence is given in Figure 3.11, and the error handling in the I2C communication sequence is provided in Figure 3.12.

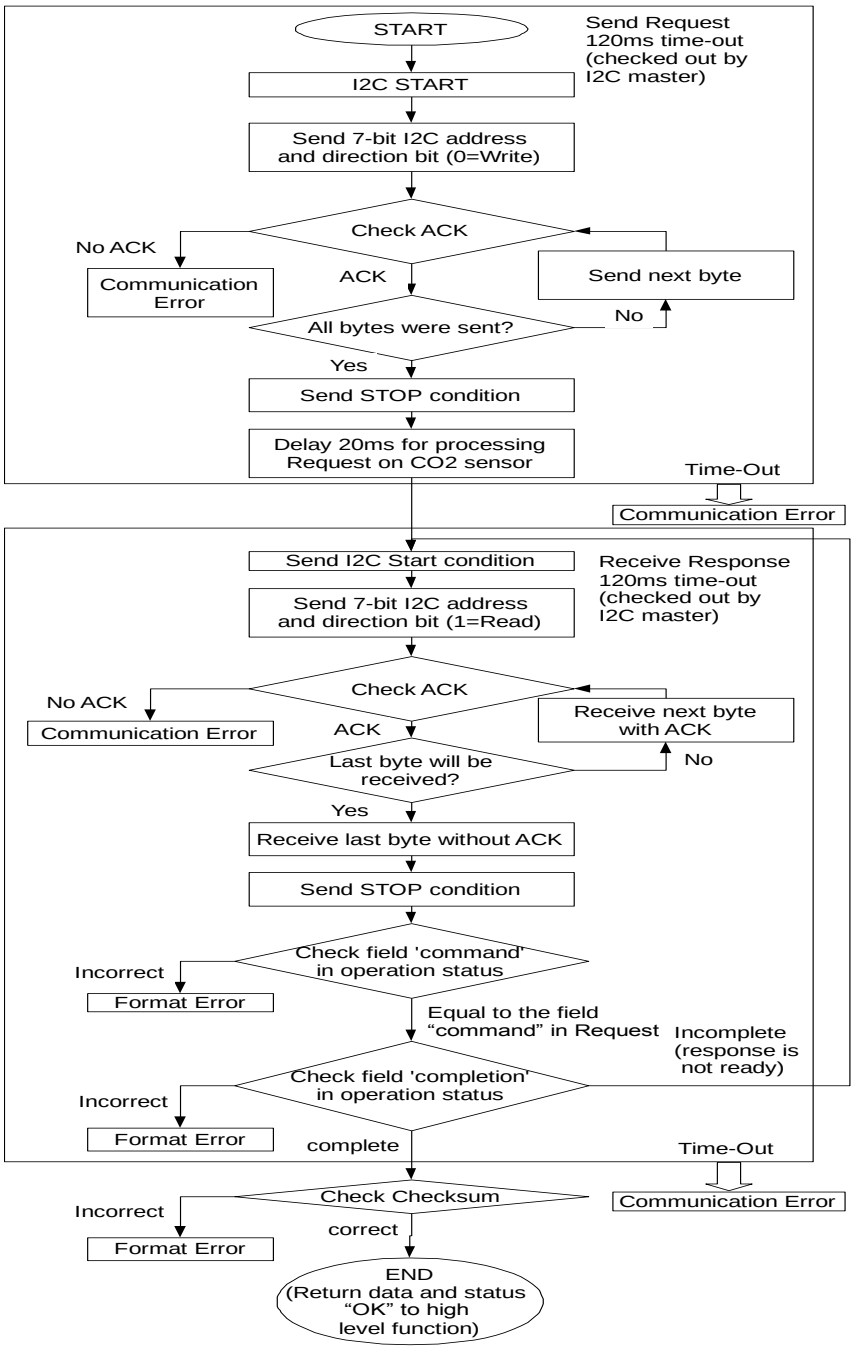


Figure 3.11 The CO2 Sensor Software Diagram

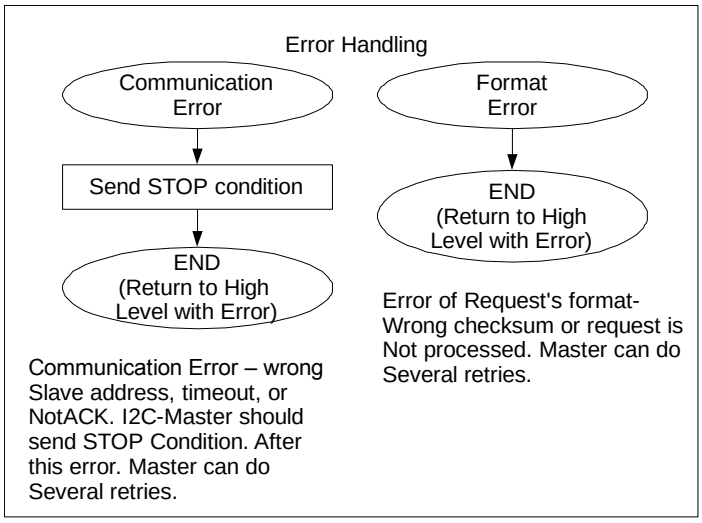


Figure 3.12 Error Handling of the CO2 Sensor Software

In order to demonstrate the CO2 communications, memory locations 0x08(high byte) and 0x09(low byte) are to be read. A sequence of two I2C frames were sent: first an I2C write frame containing the sensor address, command number, number of byes to read, RAM address to read from, and a checksum was sent; then an I2C read frame containing the sensor address (send to the sensor), four bytes data from sensor. Figure 3.13 details the K22L0 CO2 sensor programming process.

The first step is to send data to CO2 sensor to request CO2 readings. The sequence following "Start|0xD0|0x22|0x00|0x08|0x2A|Stop" is to be transmitted:

0xD0 is the sensor address.

(0x68 is the address bit. It is shifted one bit to left.

The R/W bit is 0(Write). This is the final bit.)

0x22 is command number 2 (ReadRAM), and 2 bytes to read.

0x00 is the starting byte of the memory address.

0x08 is the starting byte address of the CO2 sensor reading.

Checksum 0x2A is calculated as sum of byte 2, 3 and 4.

Figure 3.13 is a typical program to demonstrate how the CO2 sensor data can be transmitted. Timeout retry for No-ACK signal that comes back from the sensor has not been added at this stage, because the controller is constantly in communications with the CO2 sensor. The ACK signal can be always checked out for stable communications between the sensor and the board. In this case, there is no need to retry getting ACK signal back from the sensor.

```
      i2c_initial();
// START I2C
      i2c_start();
//  write input
//  send the 1st byte
      TRISC &= 0xE7;  //i2c output
      *temp1 = 0xD0;
      transmission(temp1);
      TRISC |= 0x10;  //i2c input
      delay_for_get_ACK_from_Slave();
      TRISC &= 0xE7;  //i2c output
////  2nd byte transmission
      *temp2 = 0x22;
      transmission(temp2);
      TRISC |= 0x10;  //i2c input
      delay_for_get_ACK_from_Slave();
      TRISC &= 0xE7;  //i2c output
// 3rd byte transmission
      *temp3 = 0x00;
      transmission(temp3);
      TRISC |= 0x10;  //i2c input
      delay_for_get_ACK_from_Slave();
      TRISC &= 0xE7;  //i2c output
// 4th byte transmission
      *temp4 = 0x08;
      transmission(temp4);
      TRISC |= 0x10;  //i2c input
      delay_for_get_ACK_from_Slave();
      TRISC &= 0xE7;  //i2c output
// 5th byte transmission
      *temp5 = 0x2A;
      transmission(temp5);
      TRISC |= 0x10;  //i2c input
      delay_for_get_ACK_from_Slave();
      TRISC &= 0xE7;  //i2c output
// stop bit
      i2c_stop();
```

Figure 3.13 The K22L0 CO2 Sensor Transmission Process

The second step is to set up a delay of at least 20ms to allow the CO2 sensor to respond to the main board, and calculate the CO2 density. The K22L0 CO2 sensor usually takes near 5s to compute the CO2 density around it. Figure 3.14 details the delay process in the CO2 sensor transmission sequence.

```
//-------------------------------------------------------
      delay();
//-------------------------------------------------------

void delay(void)
{
//20ms delay
      int i;
      for(i = 0; i < 9000; i ++)// 8000 nearly 20 ms, 9000 is 20ms delay actually
                                          // every unit space on oscilliscope is 2.5ms
      {
            ;
      }
}
```

Figure 3.14 The Delay Process in the CO2 Sensor Transmission Sequence

The third step is to receive CO2 readings. The sequence following "Start|0xD1|<4 bytes read from sensor>|Stop" will be received. The 1st byte from the sensor will contain operation status, where bit 0 is the read command when successfully executed. The 2nd and 3rd byte will contain high and low CO2 readings byte. The 4th byte contains checksum. Figure 3.15 details the software. In this case, the timeout for command bit wrong retry is 10 ms, and the timeout retry for No-ACK signal received is 50 ms.

```
    do{
        i2c_initial();
        i2c_start();
        TRISC &= 0xE7;  //i2c output
// send the 0xD1 byte to the CO2 sensor
        *temp3 = 0xD1;
        transmission(temp3);

        TRISC &= 0xE7;  //i2c output

        PORTCbits.SCL = 0;
        PORTCbits.SDA = 0;
        for(i = 0; i < 100; i ++);

        TRISC |= 0x10;  //i2c input

        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
    if(PORTCbits.SDA == 0)
    {
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
        TRISC &= 0xE7;  //i2c output

        PORTCbits.SCL = 0;
        PORTCbits.SDA = 0;
        for(i = 0; i < 100; i ++);

        TRISC |= 0x10;  //i2c input
        for(i = 0; i < 100; i ++);
// allow the CO2 sensor to transmit 4 bytes
        for(k = 0; k < 4; k ++)
        {
            temp_receive_byte[k] = receive_byte();   // have i2c output control
            if(k < 3)
                ACK_master();
            else
                No_ACK_master();      // final receiving byte
        }
        TRISC &= 0xE7;  //i2c output
        i2c_stop();
        complete_bit = temp_receive_byte[0] & 0x01;// == 1 if complete
        calc_result_checksum = 0;
        for(i = 0; i < 3; i ++)
        {
            calc_result_checksum += temp_receive_byte[i];
        }// direct sum together for checksum

        high_byte = temp_receive_byte[1];
        low_byte = temp_receive_byte[2];

        temp_receive_byte[0] >> 4;
        if(temp_receive_byte[0] != 0x02)// an error has occurred
                                        // if "if" is correct
        {
            if(time_out_command_bit > 0)
                time_out_command_bit --; // command bit always wrong
                                         // usually wait for 10ms
            else
                break;
        }
    }
    else
    {
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
        TRISC &= 0xE7;  //i2c output
        i2c_stop();
        if(time_out_ACK_reception > 0)// time out time, usually 50 ms, then because
                                     // of no ACK, the sensor will stop
            time_out_ACK_reception --;
        else
            break;
    }
    }while(!complete_bit);
```

Figure 3.15 CO2 Sensor Software in Receiving Cycle

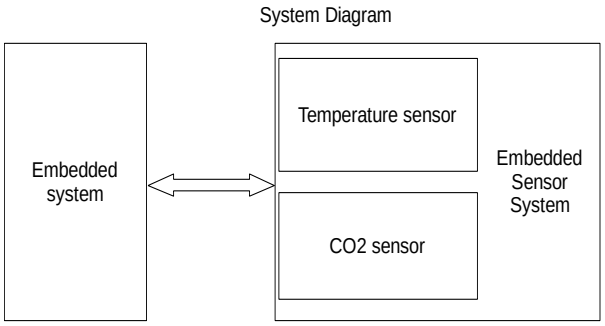
Since CO2 sensor cannot check out CO2 density as fast as the PIC18 microcontroller runs, very long delay is needed to start another CO2 measuring. Figure 3.16 shows source code for one cycle of CO2 measuring.

```
while(1)
{
      i2c_out_CO2();
      for(i=0; i<200000; i++)
      {
            ;
      }
}
```

Figure 3.16 CO2 Measuring Cycle Source Code

3.7 Integrated System Software

This software combines both CO2 and temperature sensors, interfaced to the same embedded controller. Figure 3.17 (a) gives the block diagram of the integrated system, while Figure 3.17 (b) shows how CO2 and temperature readings are sampled in every measuring cycle.



(a)

```
while(1)
{
      i2c_out_CO2();
      for(i=0; i<10; i++);
      i2c_out_temp();
      for(i=0; i<200000; i++)
      {
            ;
      }
}
```

(b)

Figure 3.17 Embedded Sensor System Diagram (a) and Sampling Sequence (b)

3.8 Close-loop System

In actual HVAC sensor system, a feedback control signal is necessary, and monitoring process is applied to various sensor readings. Both the close-loop temperature sensor system and close-loop CO2 sensor system are individually validated, and then combine to communicate together with the embedded system controller. Figure 3.18 gives the block diagram of the temperature sensor system with feedback messages.

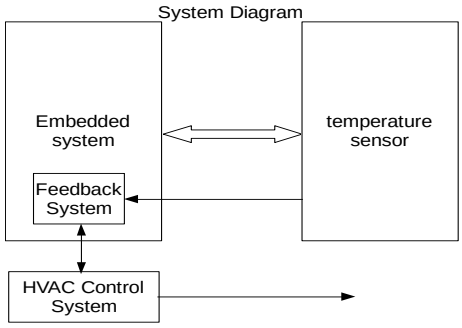


Figure 3.18 Temperature Sensor Close-loop System

In the software program, an operating temperature range was set to between 15℃ and 40℃. If at any time, the sampled temperature is in the predefined range, a feedback response will be generated by the PIC182585 microcontroller indicating that the temperature is within that range. For example, if the temperature is below 15℃, a waveform of 0xB0 will be generated. However, if the temperature is within the range from 15℃ to 40℃, a waveform of 0xC0 data is generated, and a waveform of 0xA0 data will be generated if the temperature exceeds 40℃. Figure 3.19 gives the source code for this close-loop system, encompassing this information. Figure 3.20 gives the CO2 sensor close-loop system diagram.

```
        i2c_initial();
        i2c_start();
        TRISC &= 0xE7;  //i2c output
        if(temperature >= 4000)
        {
                *temp6 = 0xA0;
                transmission(temp6);
        }
// pass the high limit
        else if(temperature <= 1500)
        {
                *temp6 = 0xB0;
                transmission(temp6);
        }
// below the low limit
        else if(1500 < temperature < 4000)
        {
                *temp6 = 0xC0;
                transmission(temp6);
        }
// in between the high limit and low limit
        TRISC &= 0xE7;  //i2c output
        i2c_stop();
```

Figure 3.19 Temperature Sensor Close-loop System Source Code

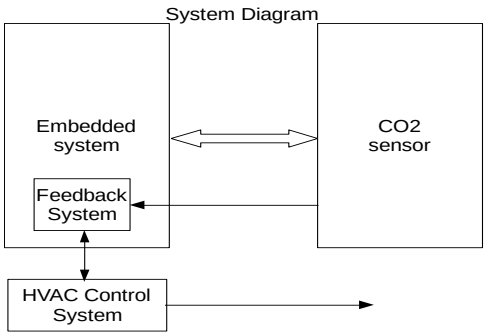


Figure 3.20 The CO2 Sensor Close-loop System Diagram

Because the temperature is semiconductor based sensor, while the CO2 sensor is infrared based sensor, detecting the temperature by the temperature sensor will be much faster than detecting the CO2 reading by the CO2 sensor. Since this speed mismatch between the two readings, data errors may be transmitted and may cause a halt in sensor communication. Thus it is necessary to check if the sensor communications halt valid results in timeout status and valid measurement at stops status. It is also necessary to check if the sensor has completed the measurements. There are two types of timeout: No-ACK signal (waveform 0xFE will be generated) and wrong command bit (waveform

0xED will be generated). It is also necessary to see if the checksum bit is correct, this will help to identify if the measuring cycle is correct (waveform 0xCB will be generated) or not (waveform 0xDC will be generated). Figure 3.21 gives the source code of the CO2 sensor close-loop system communications.

```
if(time_out_ACK_reception != 0)// time_out == 0, the sensor has no
      // ACK back during reception
{
      if(time_out_command_bit != 0)// if it is 0, then the command bit is
                                          // always wrong
      {
            if(calc_result_checksum != temp_receive_byte[3])//if it equals
                                                                  // complete
            {
                  i2c_initial();
                  i2c_start();
                  TRISC &= 0xE7;  //i2c output

                  *temp3 = 0xDC;
                  transmission(temp3);
                  TRISC &= 0xE7;  //i2c output
                  i2c_stop();
            }
            else // this is correct, get the CO2 value at this point
            {
                  i2c_initial();
                  i2c_start();
                  TRISC &= 0xE7;  //i2c output

                  *temp3 = 0xCB;
                  transmission(temp3);
                  TRISC &= 0xE7;  //i2c output
                  i2c_stop();

                  for(i = 0; i < 50; i ++);
                  // at this point, the CO2 sensor readings need to be classified into several types
            }
      }
      else
      {
            i2c_initial();
            i2c_start();
            TRISC &= 0xE7;  //i2c output

            *temp3 = 0xED;
            transmission(temp3);
            TRISC &= 0xE7;  //i2c output
            i2c_stop();
      }
}
else
{
      i2c_initial();// time out
      i2c_start();
      TRISC &= 0xE7;  //i2c output
      *temp3 = 0xFE;
      transmission(temp3);
      TRISC &= 0xE7;  //i2c output
      i2c_stop();
}
```

Figure 3.21 The CO2 Sensor Close-loop Communications Source Code

After this procedure, when the waveform 0xCB is generated, CO2 readings can be specified into several types, to indicate the status of room atmosphere. For example, because the typical room CO2 density is between 400 ppm to 600 ppm, the ranges can be classified into below 400 ppm, between 400 and 600 ppm, and above 600 ppm. For the first case, a waveform of 0xD0 will be generated; for the second case, a waveform of 0xF0 will be generated; and for the third case, a waveform of 0xE0 will be generated. Figure 3.22 gives the CO2 sensor close-loop system software. Figure 3.23 gives the temperature/CO2 sensor close-loop system diagram.

```
                    i2c_initial();
                    i2c_start();
                    TRISC &= 0xE7;  //i2c output , show the room is short of CO2

                    if(co2_reading <= 400)
                    {
                        *temp4 = 0xD0;
                        transmission(temp4);
                    }
// pass the high limit
                    else if(co2_reading >= 600)
                    {
                        *temp4 = 0xE0;
                        transmission(temp4);
                    }
// below the low limit
                    else if(400 < co2_reading < 600)
                    {
                        *temp4 = 0xF0;
                        transmission(temp4);
                    }

                    TRISC &= 0xE7;  //i2c output
                    i2c_stop();
```

Figure 3.22 CO2 Sensor Close-loop System Source Code

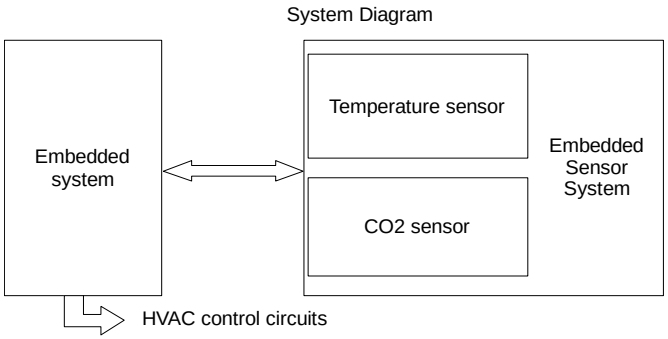


Figure 3.23 The Temperature/CO2 Sensor Close-loop System Diagram

In this software program, both CO2 sensor readings and temperature sensor readings are checked out once in every clock cycle to demonstrate the feedback signal individually. Figure 3.24 gives the source code for the sampling cycle.

```
while(1)
{
	i2c_out_CO2();
	for(i=0; i<10; i++);
	i2c_out_temp();
	for(i=0; i<200000; i++)
	{
		;
	}
}
```

Figure 3.24 Embedded Sensor System Sampling Cycle Source Code

The prescribed integrated System can be expanded to include additional sensors, such as imaging sensor, humidity sensor, EM sensor, pressure sensor, etc. Figure 3.25 shows the block diagram with additional sensors. And together with I2C protocol, relay communications can also implemented to make more sensors run simultaneously with current sensors. Figure 3.26 shows the microstructure of the integrated sensor system for the HVAC application.

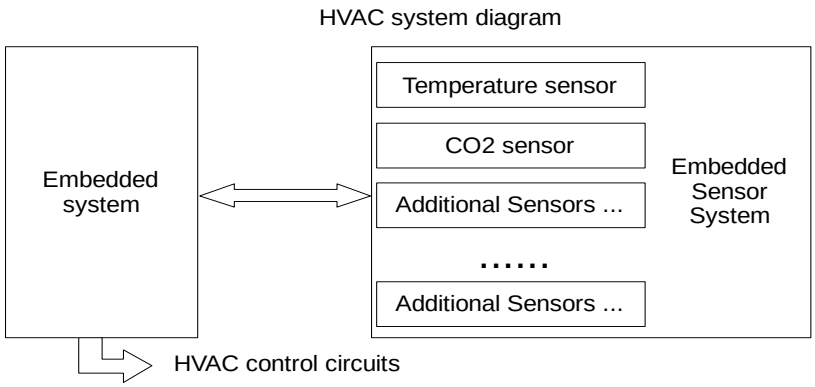


Figure 3.25 The Block Diagram with Additional Sensors

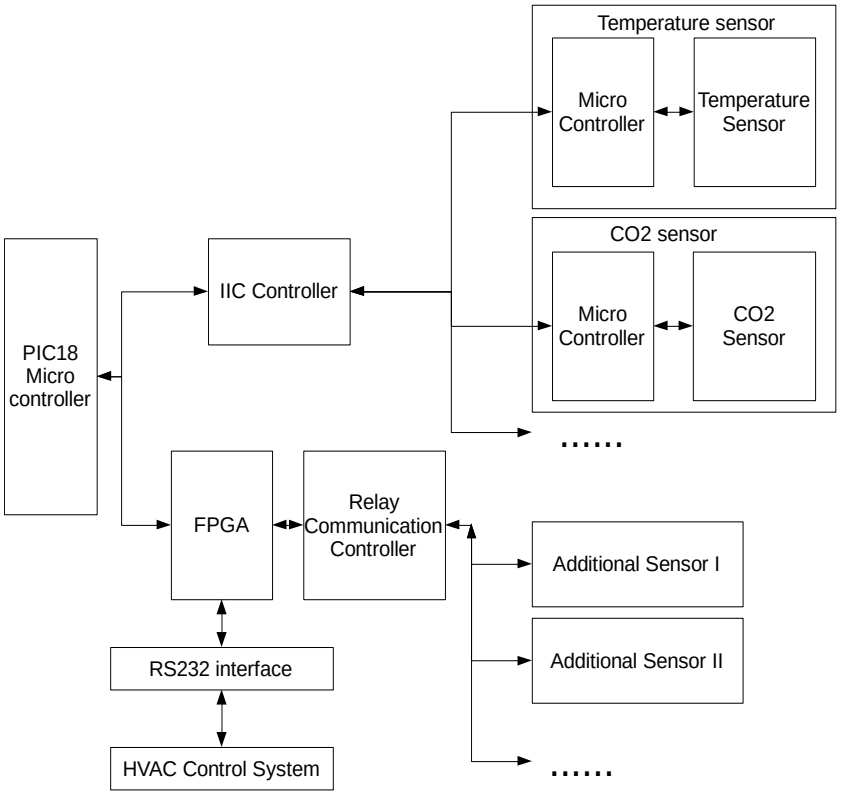


Figure 3.26 Microstructure of HVAC Sensor System

In HVAC applications, control circuitries for air conditioning, windows, etc. must be implemented. Typical HVAC system diagram containing control circuitries is shown in Figure 3.27.

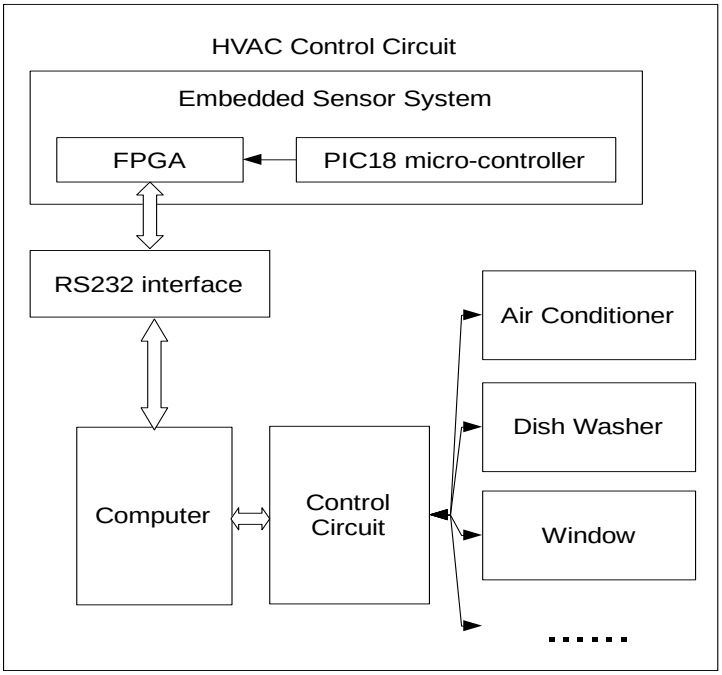


Figure 3.27 HVAC Control System

4. RESULTS AND DISCUSSIONS

In this project, two sensors were chosen for the implementation and communications with the embedded system. The selection of two sensors was based on their availability at Smart Systems Incorporation and familiarity with their processors. The Smart Systems Incorporation provided IUPUI Department of Electrical and Computer Engineering with the hardware components required for the project. The results presented here consist of 3 sets of data. The first presents the temperature sensor reading, the second demonstrates the CO2 sensor reading, while the third set of data shows the integrated system's response, including both the sensors and the microcontroller.

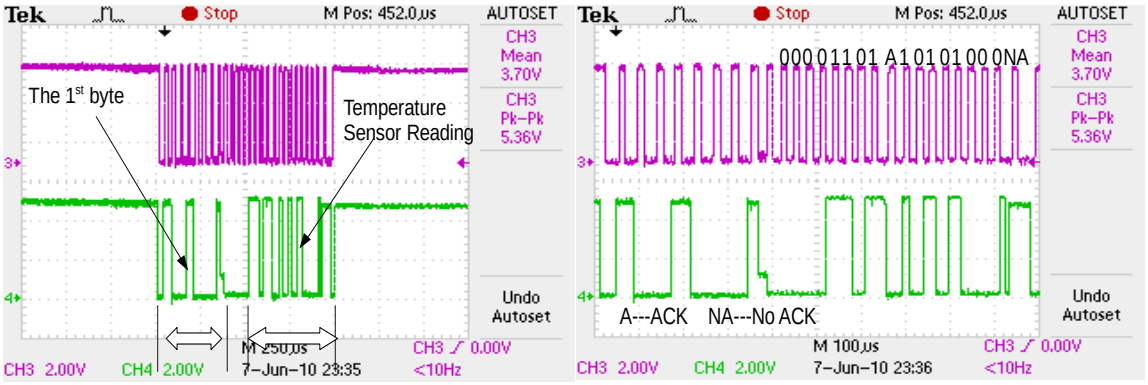
4.1 Temperature Sensor Communication

Figure 4.1 and 4.2 gives the temperature sensor reading. Figure 4.1 gives the software simulation results in which the embedded system does not provide feedback messages to outside control circuits. Figure 4.2 gives the software simulation results in which the embedded system provides feedback messages to outside circuits. These feedback messages help identify the operating region of the sensor. If the sensor reading is beyond the operating range of the sensor, the feedback message will be triggered out.

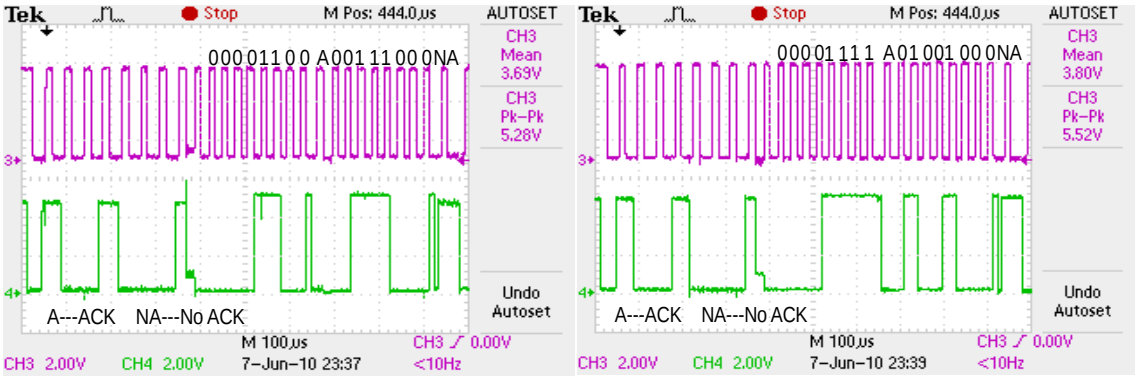
4.1.1 Temperature Sensor Communication without Feedback Messages

Figure 4.1 (a) details one cycle of temperature reading, the first byte reads 10010001 (91 in Hex) which is the address of the temperature sensor. Figure 4.1 (b) is a zoom in for the same reading. The 2^{nd} and 3^{rd} bytes in the cycle show the temperature

reading, 110110101 (1B5 in Hex), which is 27.3125℃. The upper waveform in the Figure is the clock frequency (35KHz). The ACK signal is designated by letter A, and no ACK is designated by NA. Those two signals validate the correct communications between the temperature sensor and the processor. Figure 4.1 (c)-(f) show four data samples of the temperature sensor reading. Those sample data are 110000111 (187 in Hex, 24.4375℃) in Figure 4.1 (c), 111101001 (1E9 in Hex, 30.5625℃) in Figure 4.1 (d), 110010101 (195 in Hex, 25.3125℃) in Figure 4.1 (e), 111000011 (1C3 in Hex, 28.1875℃) in Figure 4.1 (f).

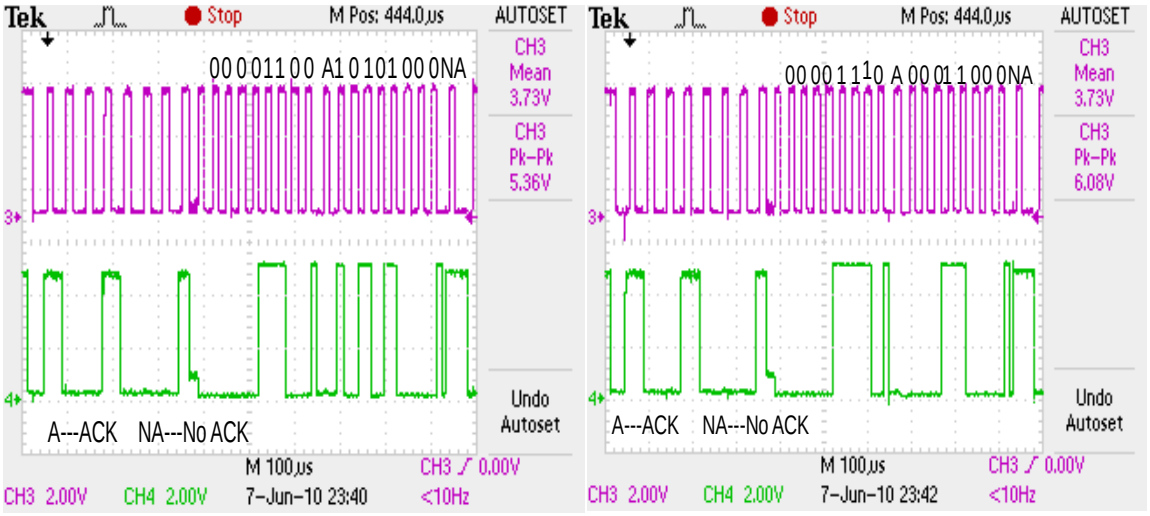


(a) (b)



(c) (d)

Figure 4.1 (a)-(f) Temperature Sensor Reading (without Feedback Messages)



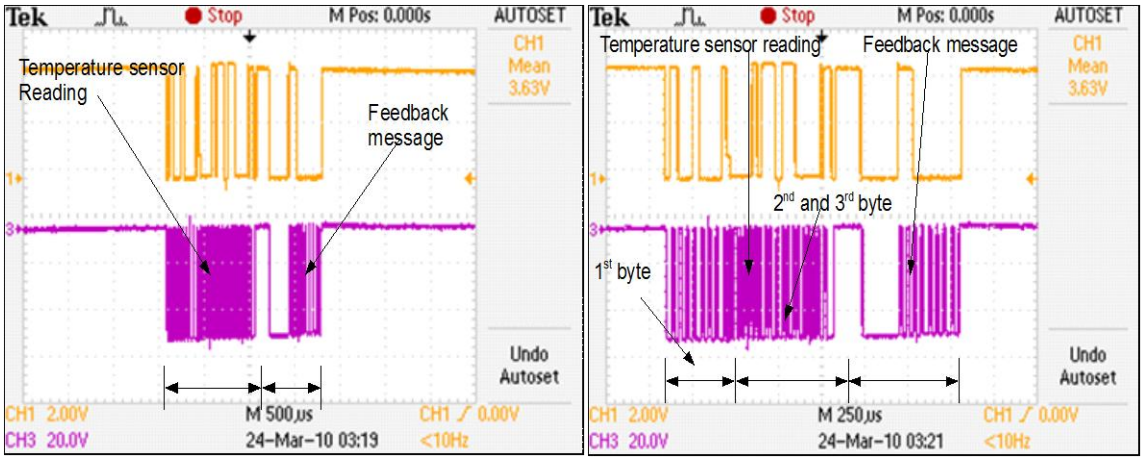
(e) (f)

Figure 4.1 Continued

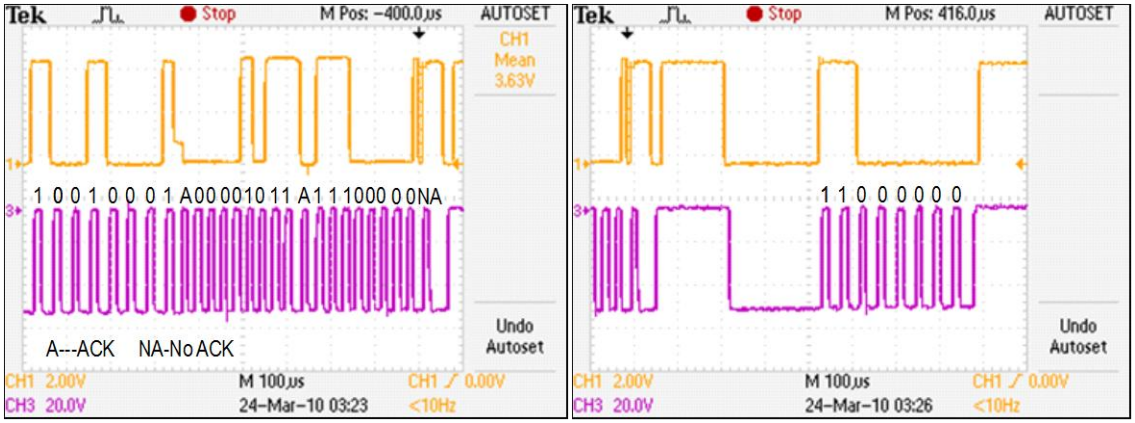
4.1.2 Temperature Sensor Communication with Feedback Messages

Figure 4.2 shows the temperature sensor reading with feedback message sent back to the board. Figure 4.2 (c) shows that the temperature sensor reading (the 2nd and 3rd bytes of the receiving cycle shown in Figure 4.2 (b)) is 0000101111100 (17C in Hex, 23.75℃). The feedback message given in Figure 4.2 (d) is 11000000 (C0 in Hex), which indicates that the temperature is within the normal range (lower threshold was chosen to be 15℃, and upper threshold was chosen to be 40℃). The feedback messages are used to indicate that necessarily actions should be taken to control the HVAC system. The upper waveform in the diagram is the clock frequency (35KHz). The ACK signal is designated by letter A, and no ACK is designated by NA. Those two signals validate the correct communications between the temperature sensor and the processor. Figure 4.2 (e)-(l) show data sets for other 4 samples. The temperature sensor readings are given by 0000111011111 (1DF in Hex, 29.9375℃) and shown in Figure 4.2 (e). The second sample shown in Figure 4.2 (g) is given by 0000111110101 (1F5 in Hex, 31.3125℃). The third sample is given by 0000111000110 (1C6 in Hex, 28.375℃) as shown in Figure 4.2 (i). The fourth sample gives data of 0000111111010 (1FA in Hex, 31.625℃) as given

by Figure 4.2 (k). The feedback messages in those four samples are shown in Figure (f) (h) (j) (l). They correspond to temperature sensor readings in Figure (e) (g) (i) (k) respectively. The value of them are all 11000000 (C0 in Hex), which indicates that the temperature is within the normal range.

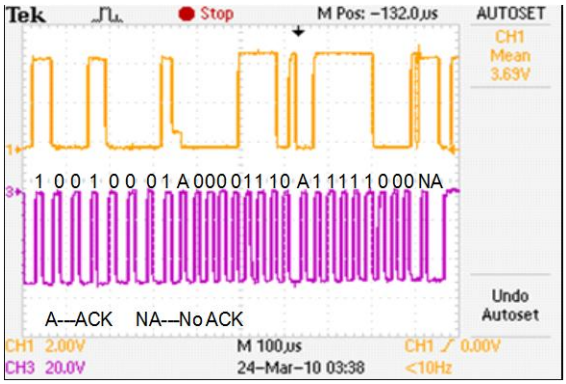


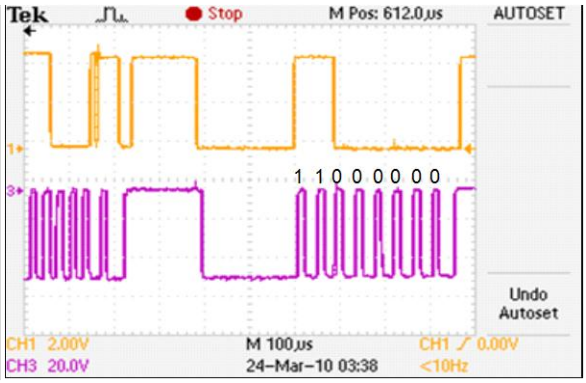
(a) (b)



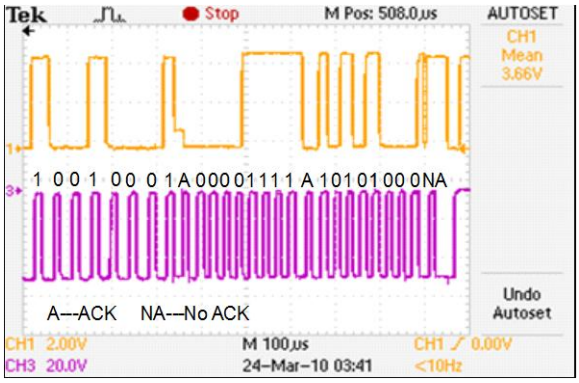
(c) (d)

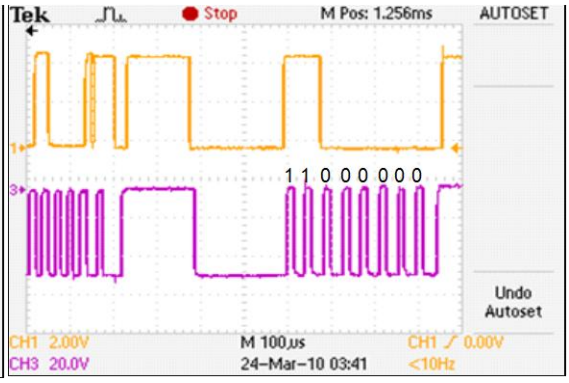
Figure 4.2 (a)-(l) Temperature Sensor Reading (with Feedback Messages)



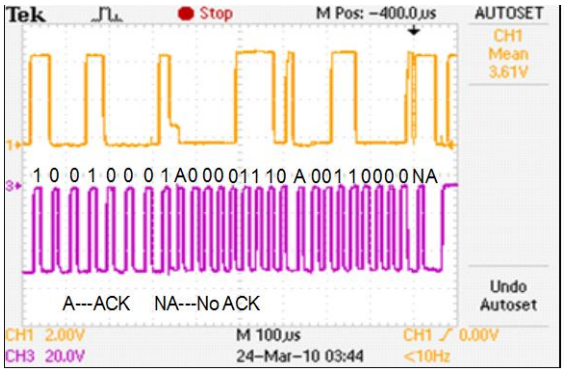


(e) (f)



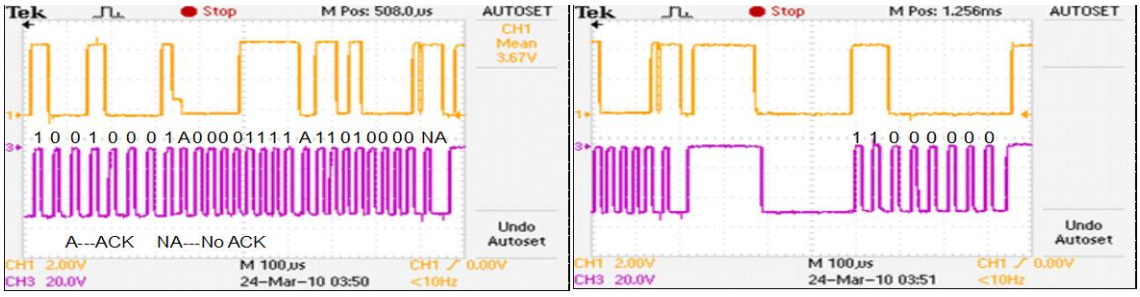


(g) (h)



(i) (j)

Figure 4.2 Continued



(k) (l)

Figure 4.2 Continued

4.2 CO2 Sensor Communications

4.2.1 Communication Format

Table 4.1 gives the CO2 sensor transmission and receiving sequence formats. In the transmission cycle, the slave address and relevant configuration bit are written into the CO2 sensor. This is followed by a command byte, start and stop addresses, and checksum byte. The transmission sequence format (see table 4.1 (a)) is given as follows:

a) Start command
b) D0 in Hex (7-bit slave address 68 in Hex and 1 read/write bit)
c) 22 in Hex (command number 2 in Hex (ReadRAM), and 2 bytes to read)
d) 00 in Hex (start address of the read procedure),
e) 08 in Hex (end address of the read procedure),
f) 2A in Hex (Checksum 2A in Hex, sum of byte 2, 3 and 4), and
g) Stop command.
h) Delay for at least 20ms.

Table 4.1 (b) shows the CO2 sensor receiving sequence. 7-bit slave address and 1 read/write bit are transmitted to the CO2 sensor. The CO2 sensor will then send an operation byte followed by MSB, LSB, and checksum byte back to Master (PIC182585).

The communication sequence is given as follows:

a) Start command
b) D1 in Hex (7-bit slave address 68 in Hex and 1 read/write bit)
c) Operation byte (this indicates the operation status. Bit 0 tells if the read command was successfully executed)
d) CO2 sensor reading (CO2 sensor high byte: CO2 sensor low byte)
e) Checksum byte, and
f) Stop command.

Table 4.1 CO2 Sensor Transmission and Receiving Sequence Format

START condition	7 bit address + Direction bit (write)	Command byte	Address	checksum	STOP condition	Master Wait Delay 20ms
	1 byte	1 byte	2 bytes	1 byte		

(a) CO2 Sensor Transmission Sequence Format

START condition	7 bit address + Direction bit (read)	Operation Status	Read data	checksum	STOP condition
	1 byte	1 byte	2 bytes	1 byte	

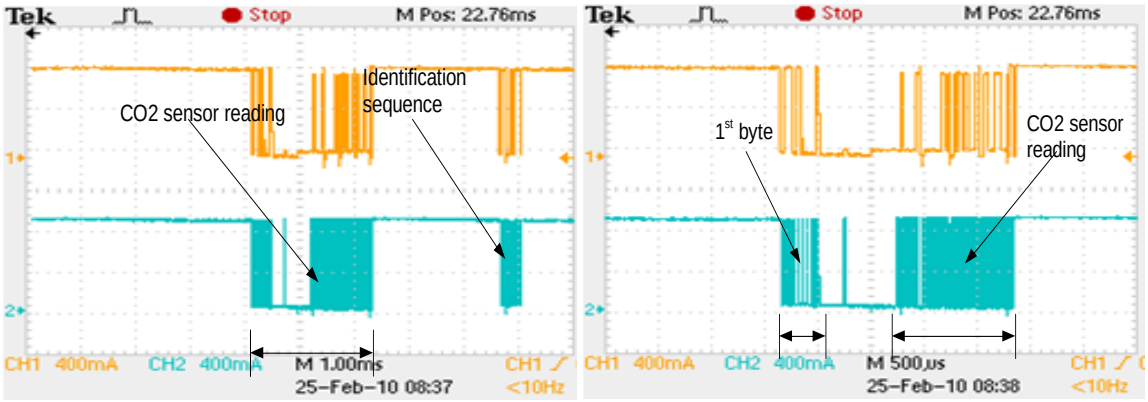
(b) CO2 Sensor Receiving Sequence Format

4.2.2 CO2 Sensor Reading

Figure 4.3 to 4.6 give the CO2 sensor reading waveforms. Figure 4.3 and 4.4 give the software simulation results in which the embedded system does not provide feedback messages to outside control circuits. Figure 4.5 and 4.6 give the software simulation results in which the embedded system provides feedback messages to outside circuits. Those feedback messages help to identify the sensor operating range.

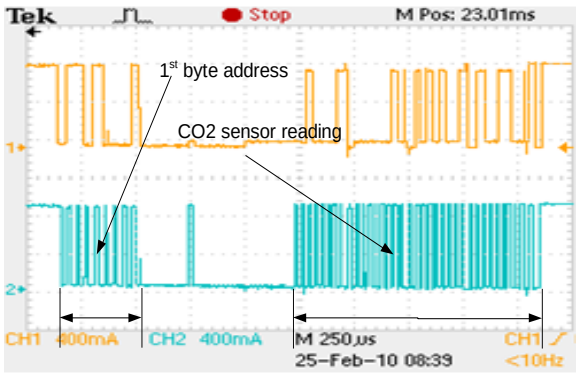
4.2.2.1 CO2 Sensor Communication without Feedback Messages

Figure 4.3 and 4.4 shows the receiving cycle of the CO2 sensor communication. It does not contain the feedback message to be sent to the board. Figure 4.3 gives the overall CO2 sensor waveforms within the receiving cycle. Figure 4.3 (b) shows the first byte in this waveform. It is 11010001 (D1 in Hex) as shown in Figure 4.3 (c), which is the address of the CO2 sensor followed by 1 read/write bit with a value of 1, while Figure 4.3 (d) and (e) give a zoom in diagram for the CO2 sensor reading shown in Figure 4.3 (b) (the 2nd to 5th byte within the receiving cycle). Those two Figures show a CO2 reading of 0000001010101001 (2A9 in Hex, 2nd and 3rd byte of the CO2 sensor reading), which is to 681 ppm in decimal. The sum of the 1st, 2nd, 3rd bytes in the CO2 sensor reading is the same as the value of the 4th byte, which reads 11001100. This indicates that the CO2 sensor works OK. The upper waveform gives a clock frequency of 35KHz. The ACK signal is designated by letter A and no ACK is designated by NA. Those two signals validate the correct communications between the CO2 sensor and the processor. Figure 4.3 (f) and (g) show the identification sequence: 11001011 (CB in Hex). It demonstrates that the CO2 sensor reading is correct.

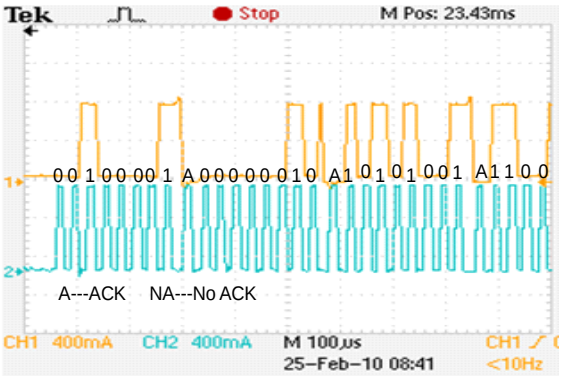


(a) (b)

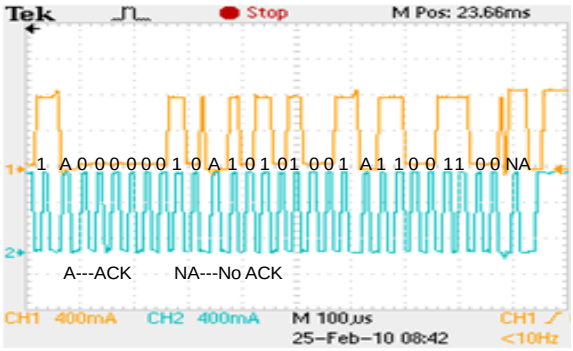
Figure 4.3 (a)-(g) One Cycle of CO2 Sensor Reading (without Feedback Messages)



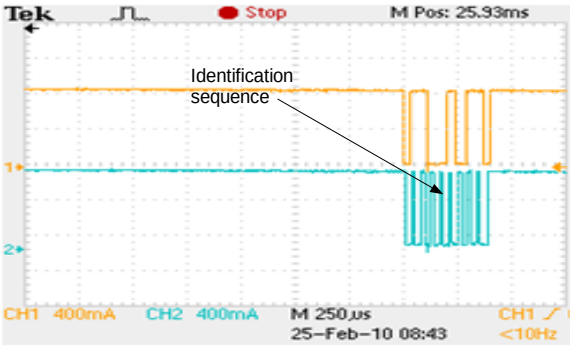
(c)



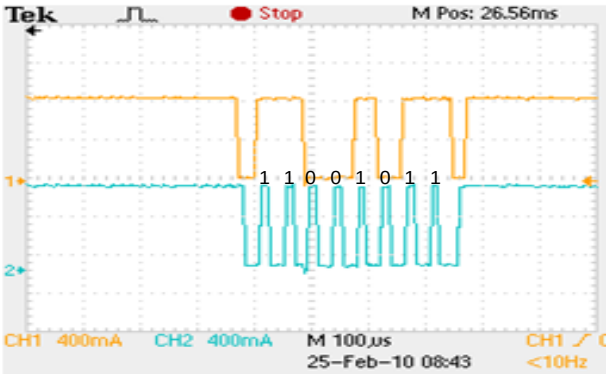
(d)



(e)



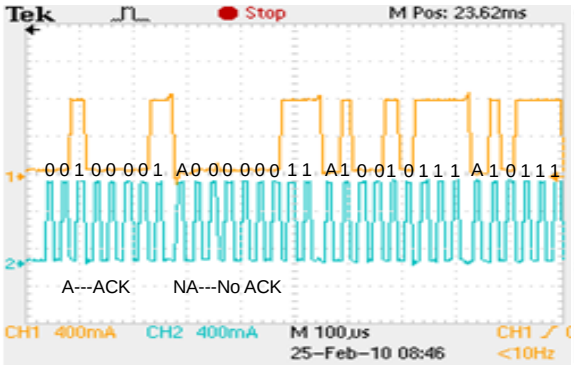
(f)



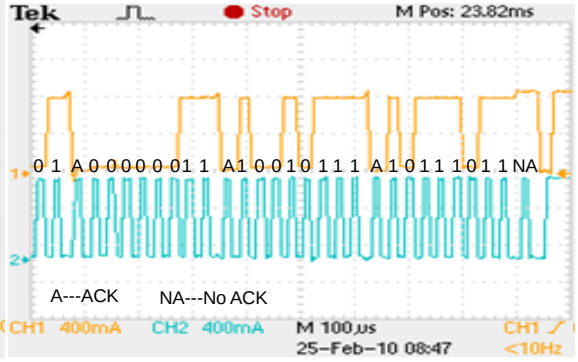
(g)

Figure 4.3 Continued

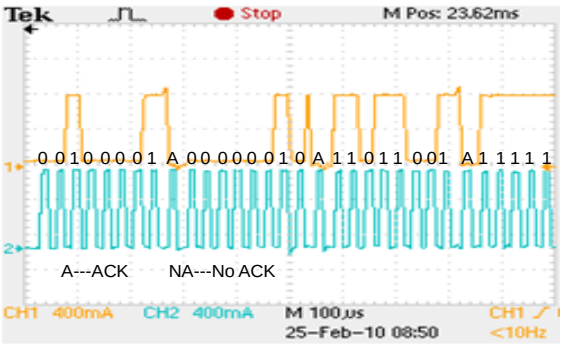
Figure 4.4 (a)-(h) are four sample data sets of the CO2 sensor readings. In summary, the four CO2 sensor readings are given by 0000001110010111 (397 in Hex) in Figure (a)-(b), which is to 919 ppm in decimal; 0000001011011001 (2D9 in Hex) in Figure (c)-(d), which is to 729 ppm in decimal; 0000001110011010 (39A in Hex) in Figure (e)-(f), which is to 922 ppm in decimal; 0000001011001000 (2C8 in Hex) in Figure (g)-(h), which is to 712 ppm in decimal. In all those four sample data sets, the sum of the 1st, 2nd and 3rd byte of the CO2 sensor readings are the same as the 4th byte in the same readings. This indicates that the CO2 sensor works correctly and the identification sequences in those 4 experiments will show 11001011 (CB in Hex).



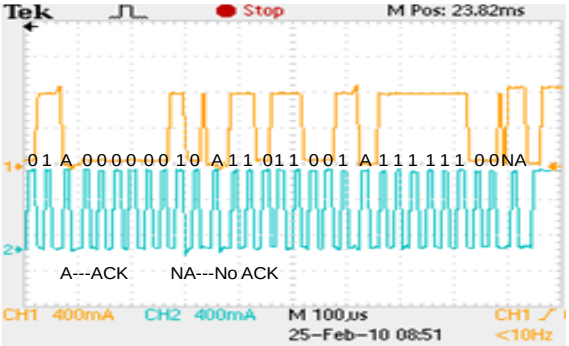
(a)



(b)

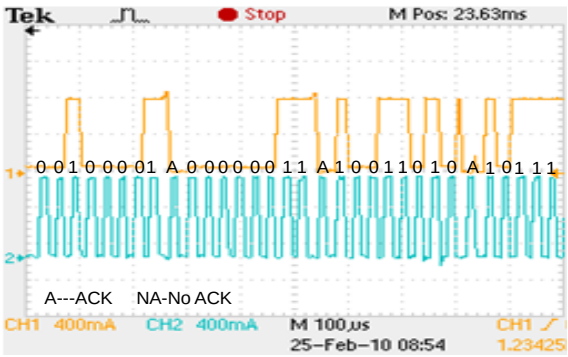


(c)

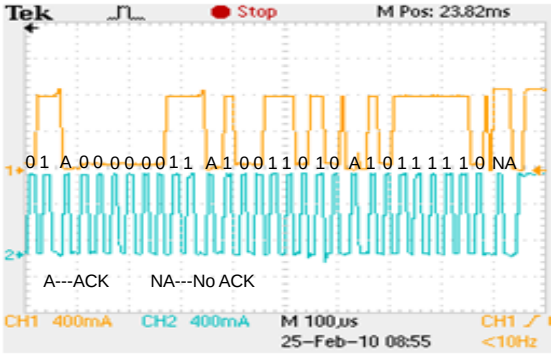


(d)

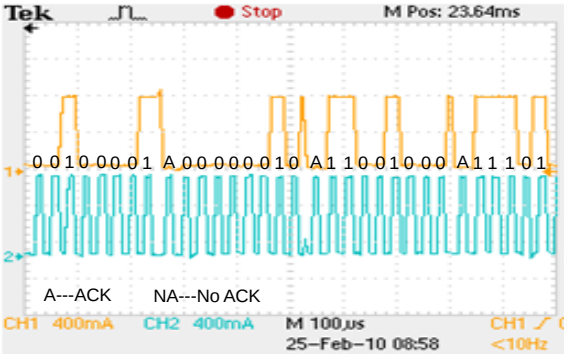
Figure 4.4 (a)-(h) Four Data Sample Sets of CO2 Sensor Readings (without Feedback Messages)



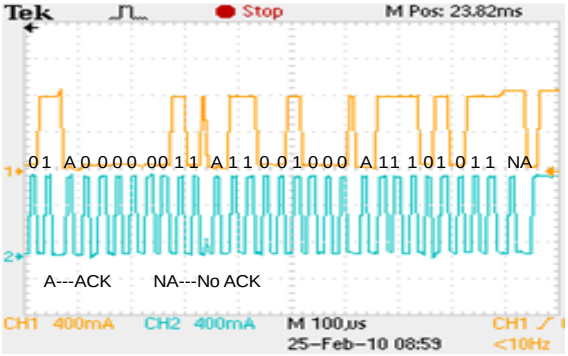
(e)



(f)



(g)



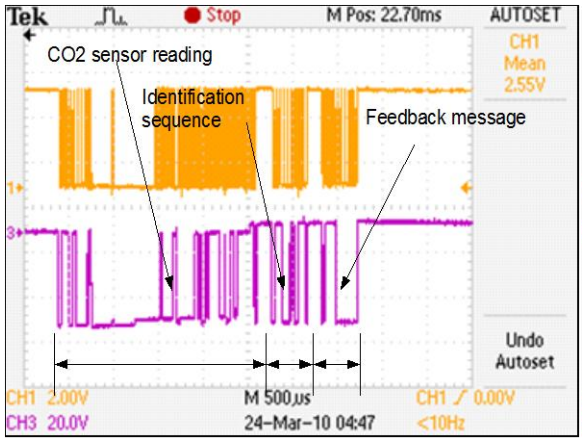
(h)

Figure 4.4 Continued

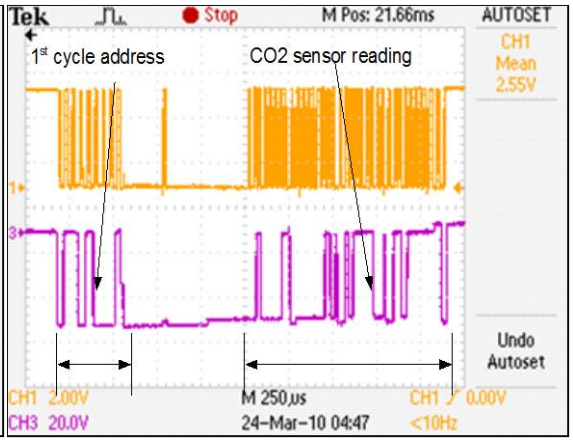
4.2.2.2 CO2 Sensor Communication with Feedback Messages

Figure 4.5 shows the receiving cycle of the CO2 sensor communication that contains the feedback messages to be sent back to the board. Figure 4.5 (a) gives the overall CO2 sensor waveform in the receiving cycle. Figure 4.5 (b) and (c) is the zoom in Figures for Figure 4.5 (a). Figure 4.5 (d) shows that the first byte in this waveform is 11010001 (D1 in Hex), which is the address of the CO2 sensor followed by 1 read/write bit with a value of 1. Figure 4.5 (e)-(f) give a zoom in diagram for the CO2 sensor readings. These two Figures show a CO2 reading of 0000001010111100 (2BC in Hex), which is to 700 ppm in decimal. The sum of the 1st, 2nd, 3rd bytes in the Figures is the value of the 4th byte, which reads 11011111. This indicates that the CO2 sensor works

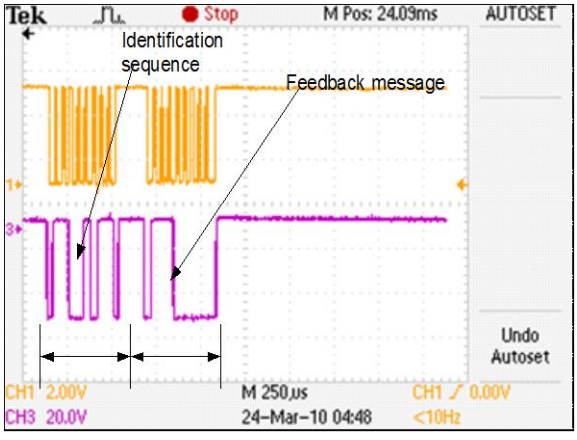
properly. The upper waveform gives a clock frequency of 35KHz. The ACK signal is designated by letter A and no ACK is designated by NA. Those two signals validate the correct communications between the CO2 sensor and the processor. Figure 4.5 (g) shows the identification sequence (1st byte data): 11001011 (CB in Hex), which demonstrates that the CO2 sensor reading is correct. It also gives a feedback message (2nd byte data) of 11100000 (E0 in Hex), which shows that the CO2 reading is above the normal range (the lower threshold is 400 ppm and the upper threshold is 600 ppm). The feedback messages are used to indicate that necessarily actions should be taken to control the HVAC system.



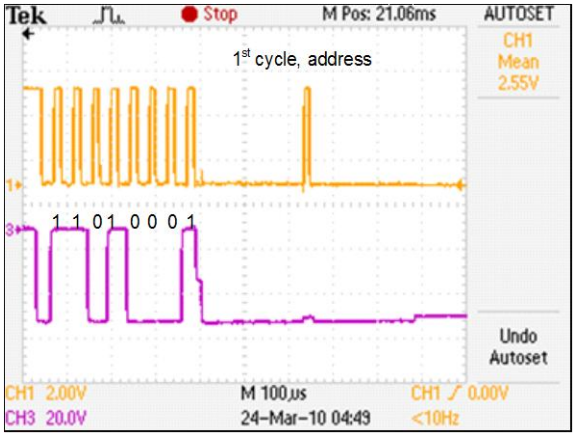
(a)



(b)

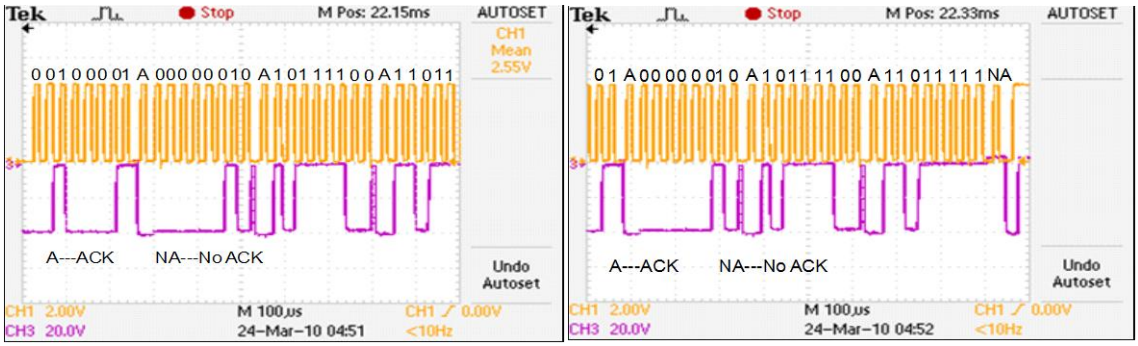


(c)



(d)

Figure 4.5 (a)-(g) One Cycle of CO2 Sensor Readings (with Feedback Messages)

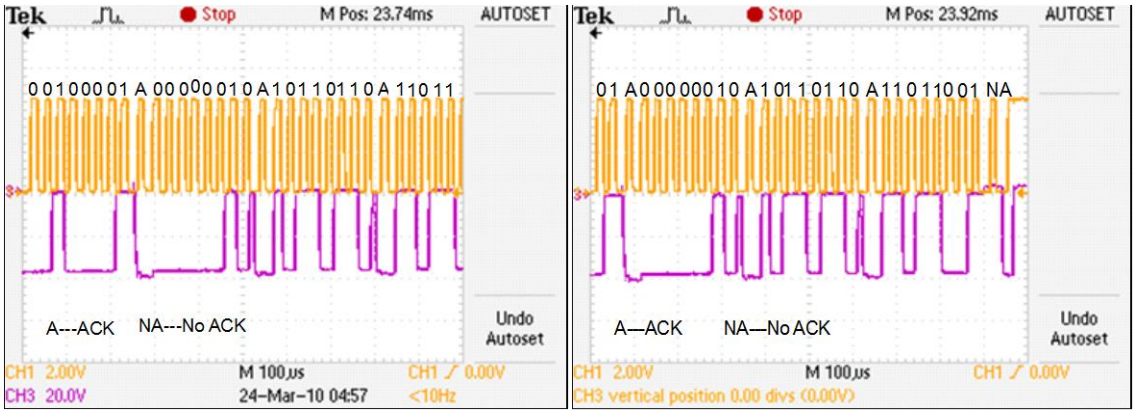


(e) (f)

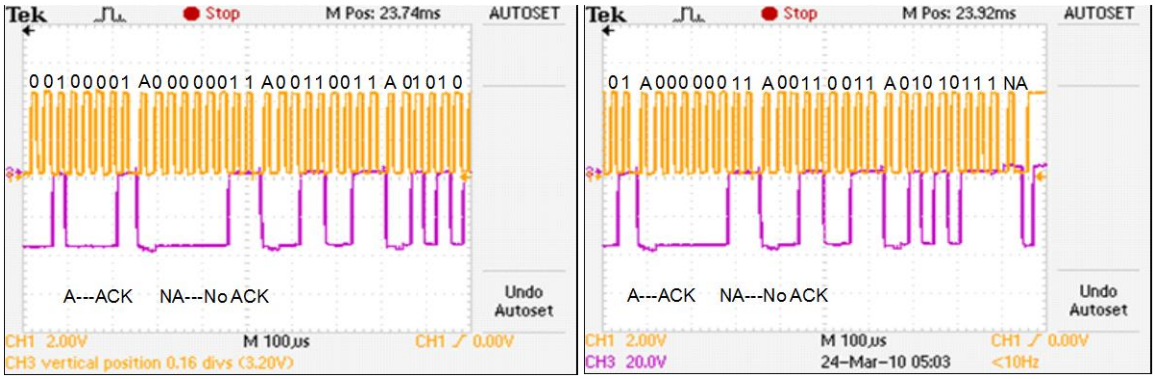
(g)

Figure 4.5 Continued

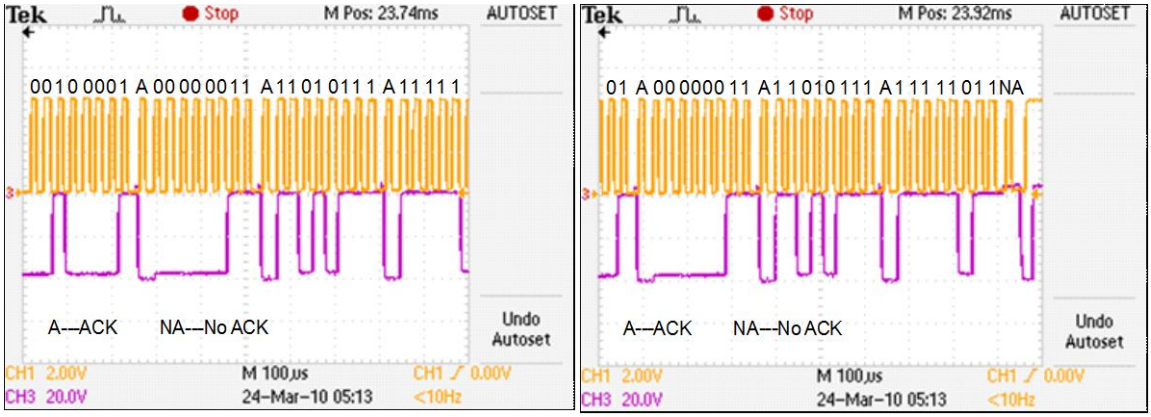
Figure 4.6 (a)-(h) are other four sample data sets. In summary, the four sample data sets of the CO2 sensor readings are 0000001010110110 (2B6 in Hex) in Figure (a) and (b), which is to 694 ppm in decimal; 0000001100110011 (333 in Hex) in Figure (c) and (d), which is to 819 ppm in decimal; 0000001111010111 (3D7 in Hex) in Figure (e) and (f), which is to 983 ppm in decimal; 0000001010111011 (2BB in Hex) in Figure (g) and (h), which is to 699 ppm in decimal respectively. In all those four sample data sets, the sum of the 1st, 2nd and 3rd bytes of the receiving bytes are the same as the 4th byte in the same cycle. This indicates that the CO2 sensor works correctly and the identification sequence will show 11001011 (CB in Hex). Since the CO2 sensor readings exceed the upper threshold 600 ppm, the feedback message will show 11100000 (E0 in Hex).



(a) (b)

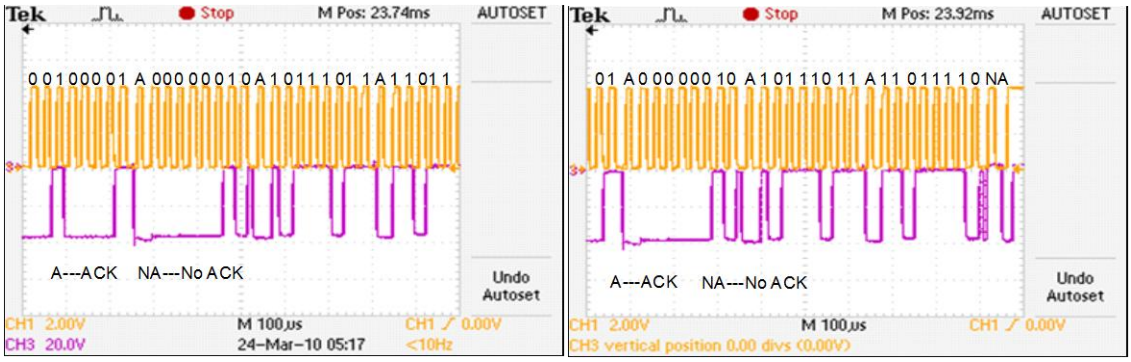


(c) (d)



(e) (f)

Figure 4.6 (a)-(h) Four Sample Data Sets of CO2 Sensor Reading (with Feedback Messages)



(g) (h)

Figure 4.6 Continued

4.3 Temperature/CO2 Sensor Integrated System Communication

Figures 4.7 to 4.13 give the temperature and CO2 sensor integrated system's readings. Figures 4.7 and 4.8 give the software simulation result in which the embedded system does not provide feedback messages to the control circuit. Figure 4.9 to 4.13 give the software simulation results in which the embedded system provides feedback messages to outside circuit. Those feedback messages help to identify if the sensor reading is in the right range.

4.3.1 Temperature/CO2 Sensor Communication without Feedback Messages

Figure 4.7 gives one cycle of the temperature/CO2 sensor integrated system's readings. Figure 4.7 (a) shows the overall Figure for one receiving cycle. It consists of the CO2 sensor communication followed by the temperature sensor communication. It does not contain the feedback messages that those sensors send back to the embedded system board. Figures 4.7 (b)-(f) are the CO2 sensor communication waveforms, and Figure 4.7 (g) gives the temperature sensor communication waveform. Figure 4.7 (b) shows the first byte of the CO2 sensor communication. It is 7-bit address of the CO2 sensor and 1 read/write bit, which is 11010001 (D1 in Hex). Figure 4.7 (c) shows the first byte of the temperature sensor communication. It is 7-bit address of the temperature sensor and 1

read/write bit, which is 10010001 (91 in Hex). Figures 4.7 (d) and (e) show the corresponding CO2 sensor reading. It is 0000001000100010 (222 in Hex), which is 546 ppm in decimal. The upper waveform in the diagram is the clock frequency (35KHz). The ACK signal is designated by letter A and no ACK is designated by NA. Those two signals validate the correct communications between the CO2 sensor and the processor. Figures 4.7 (c) and (f) show the CO2 sensor's identification sequence. Its value is 11001011 (CB in Hex), which shows that the CO2 sensor reading is correct. Likewise, Figure 4.7 (g) shows the temperature sensor reading (shown in Figure 4.7 (c)) as 110001011 (18B in Hex), which is 24.6875℃.

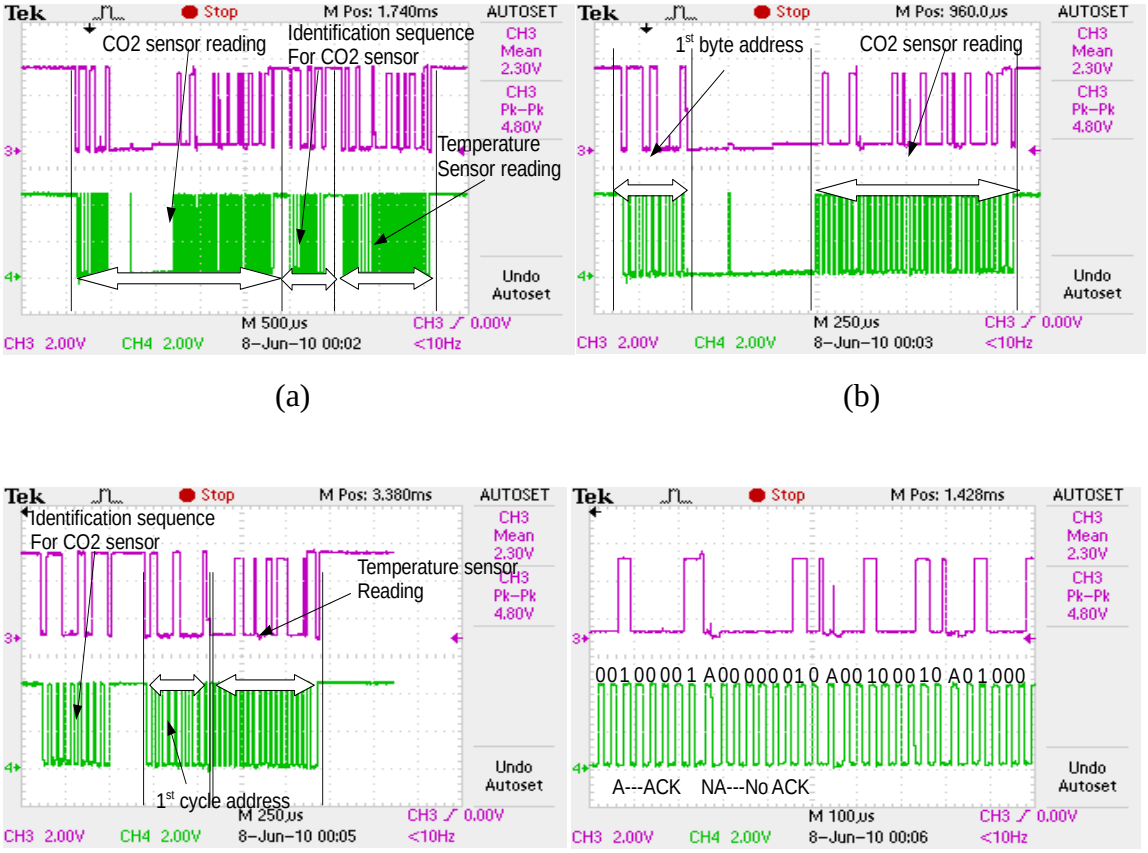
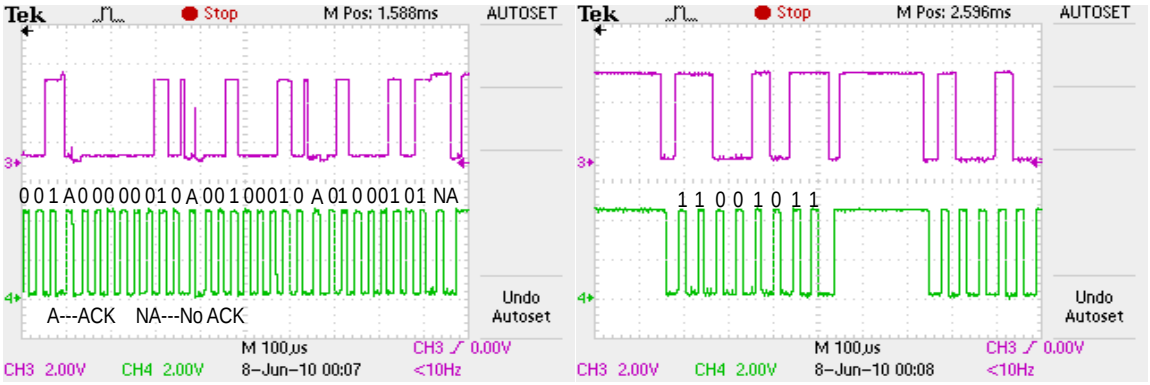
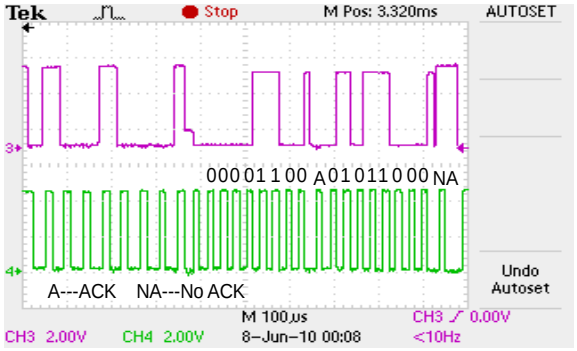


Figure 4.7 (a)-(g) One Cycle of CO2/Temperature Sensors Integrated System's Readings (without Feedback Messages)



(e) (f)



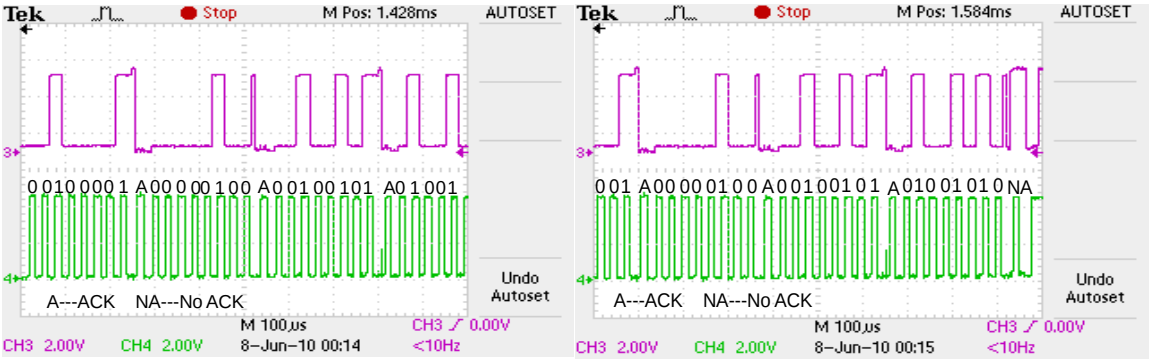
(g)

Figure 4.7 Continued

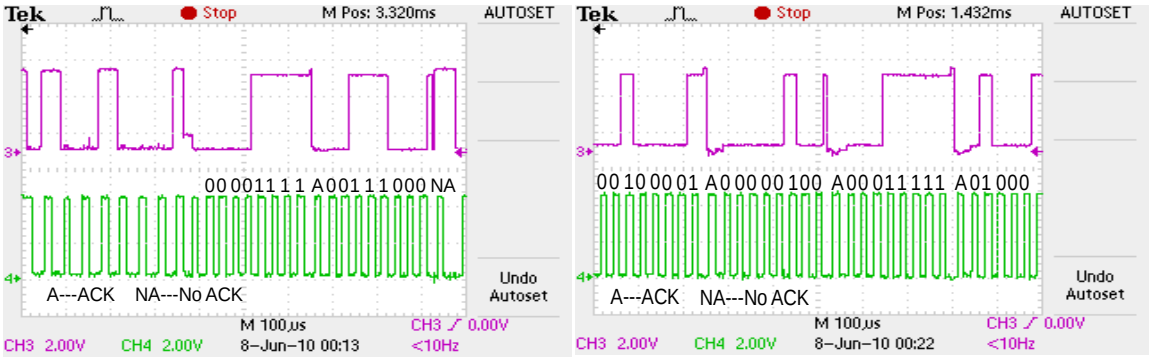
Figures 4.7 (a) to (l) give the other four sample temperature/CO2 sensors integrated system's readings. Figures (a) and (b) show the CO2 sensor reading as 0000010000100101 (425 in Hex), which is 1061 ppm in decimal. Figures (d) and (e) show the CO2 sensor reading as 0000010000011111 (41F in Hex), which is 1055 ppm in decimal. Figures (g) and (h) show the CO2 sensor reading as 0000010011101111 (4EF in Hex), which is 1263 ppm in decimal. Figures (j) and (k) show the CO2 sensor reading as 0000010110110101 (5B5 in Hex), which is 1461 ppm in decimal.

Figures 4.7 (c), (f), (i), (l) show the temperature sensor readings that correspond to these four CO2 sensor readings respectively. In Figure (c), it is 0000111100111000

(F38 in Hex), which is 30.4375℃. Figure (f) gives a reading of 0000111001001000 (E48 in Hex), which is 28.5625℃. Figure (i) gives a reading of 0000111011001000 (EC8 in Hex), which is 29.5625℃. Figure (l) gives 0000111100011000 (F18 in Hex), which is 30.1875℃.

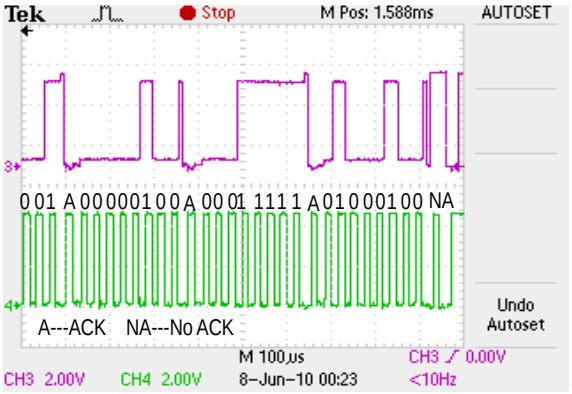


(a) (b)

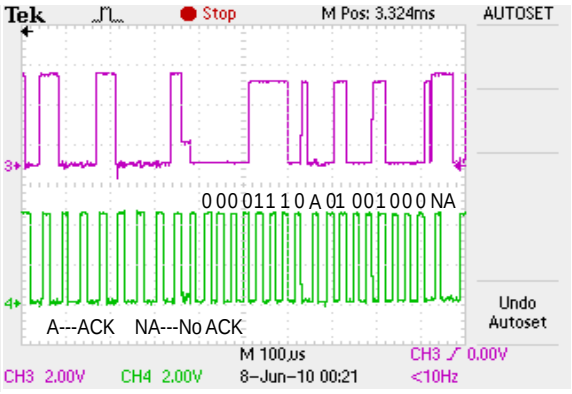


(c) (d)

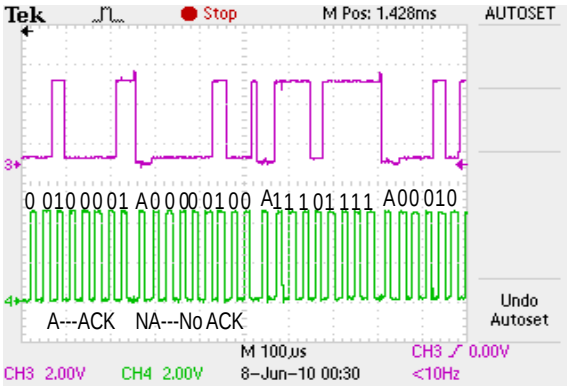
Figure 4.8 (a)-(l) Four Sample Data Sets of Temperature/CO2 Sensors Integrated System's Readings (without Feedback Messages)



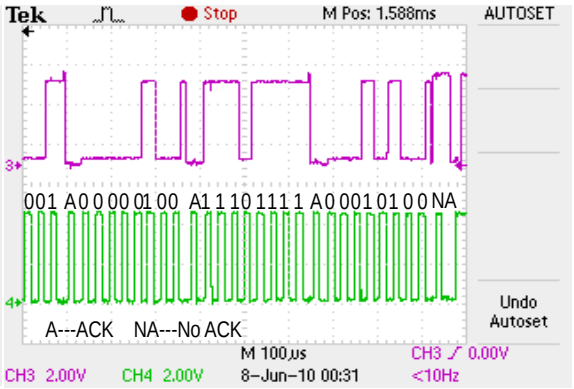
(e)



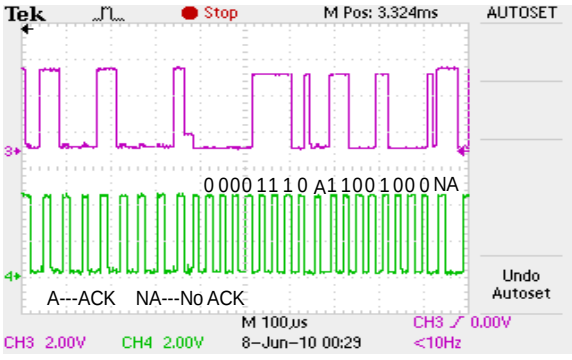
(f)



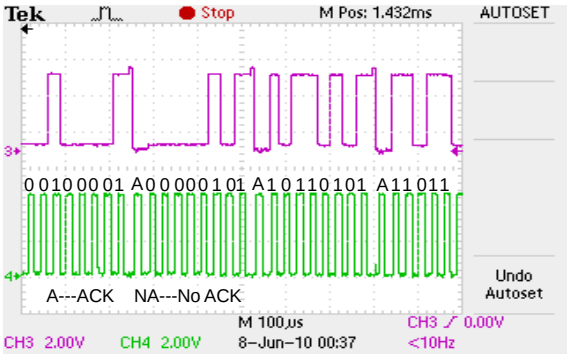
(g)



(h)

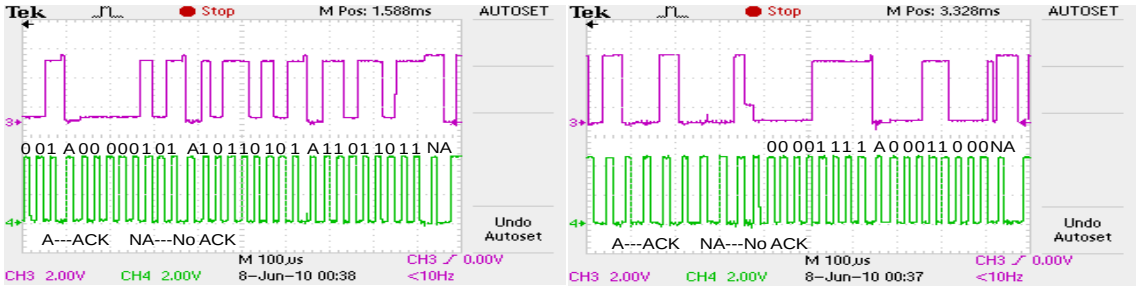


(i)



(j)

Figure 4.8 Continued



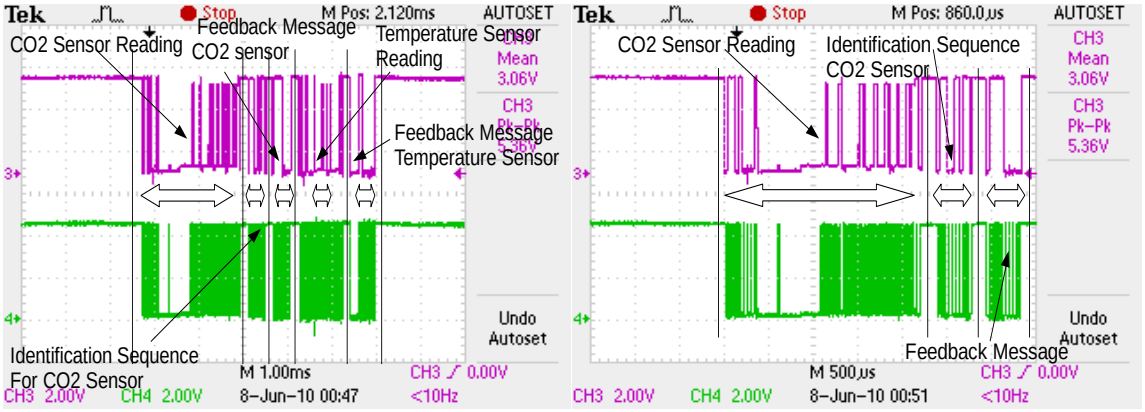
(k) (l)

Figure 4.8 Continued

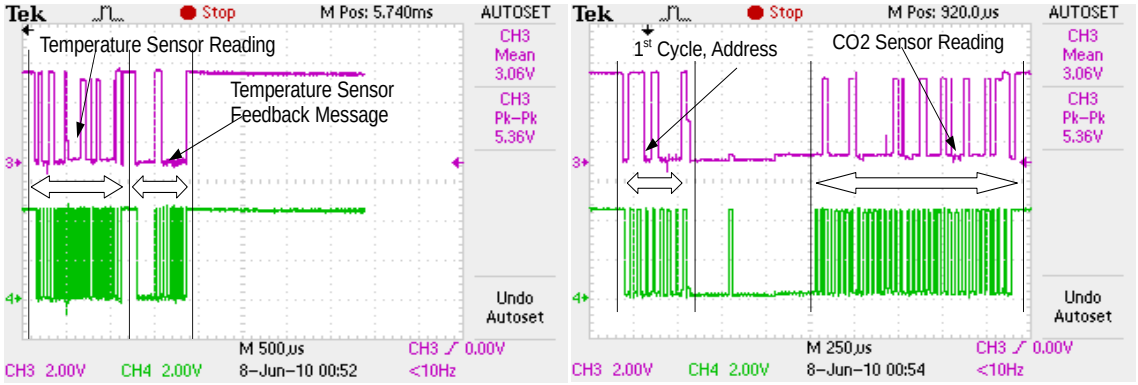
4.3.2 Temperature/CO2 Sensor Communication with Feedback Messages

Figure 4.9 gives the temperature/CO2 sensors integrated system's readings. Figure 4.9 shows one receiving cycle, consisting of the CO2 sensor communication followed by the temperature sensor communication. These Figures show the feedback messages that two sensors send back to the embedded system board. Figures 4.9 (a) to (c) show the overall sensors' waveforms. Figures 4.9 (b), (d), (e), (g), (h), (i), (j) are the CO2 sensor communication waveforms, and Figures 4.9 (c), (f), (k), (l) are the temperature sensor communication waveforms. Figure 4.9 (d) shows the first byte in the CO2 sensor communication. It is 7-bit address of the CO2 sensor and 1 read/write bit, which is 11010001 (D1 in Hex) as shown in Figure 4.9 (g). Figures 4.9 (h) and (i) show the corresponding CO2 sensor reading. It is 0000001000100010 (222 in Hex), which is 546 ppm in decimal. The upper waveform in the diagram is the clock frequency (35KHz). The ACK signal is designated by letter A and no ACK is designated by NA. Those two signals validate the correct communications between the CO2 sensor and the processor. Figure 4.9 (e) shows the CO2 sensor's two control sequences: the 1st sequence is to validate the correct communications, and the 2nd sequence is feedback message. Figure 4.9 (j) shows that the 1st sequence value is 11001011 (CB in Hex), which demonstrates that the CO2 sensor reading is correct. The 2nd sequence value is shown as 11100000 (E0 in Hex), which indicates that the CO2 sensor reading is above the upper threshold value (the lower threshold is 400 ppm and the upper threshold is 600 ppm).

Likewise, Figure 4.9 (f) shows one temperature sensor communication cycle and its feedback message. Figure 4.9 (k) shows the temperature sensor reading as 0000110001000000 (C40 in Hex), which is 24.5℃. Figure 4.9 (l) shows the feedback message as 11000000 (C0 in Hex), indicating that the temperature reading is within the normal range (lower threshold was chosen to be 15℃, and upper threshold was chosen to be 40℃). The feedback messages are used to indicate that necessarily actions should be taken to control the HVAC system.

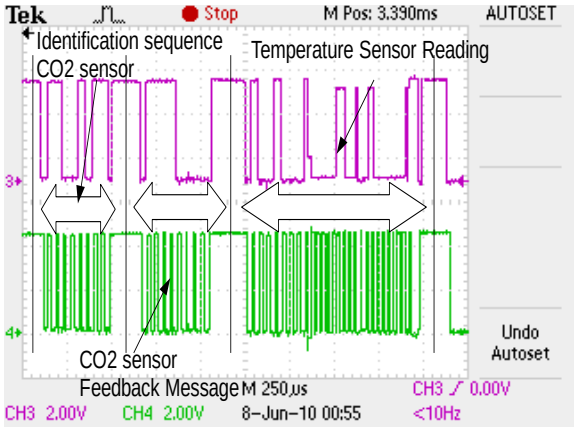


(a) (b)

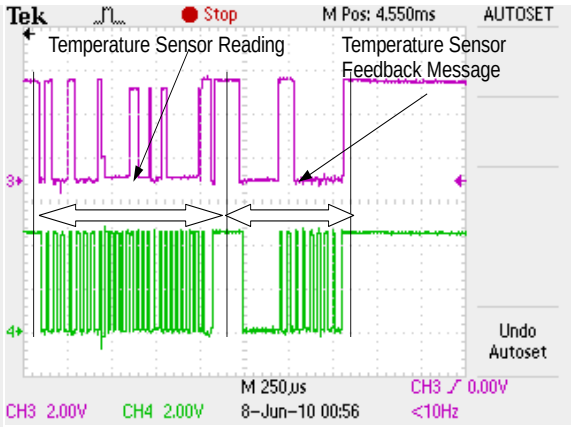


(c) (d)

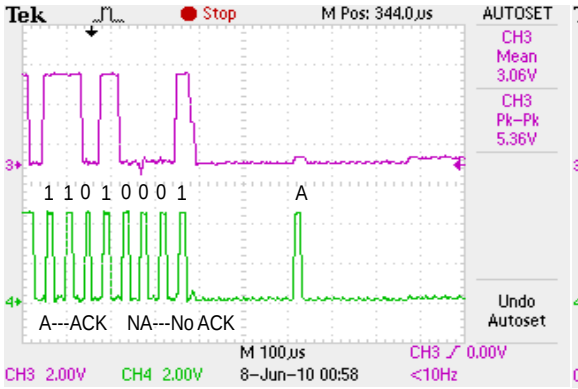
Figure 4.9 (a)-(l) One Cycle of CO2/Temperature Sensors Integrated System's Readings (with Feedback Messages)



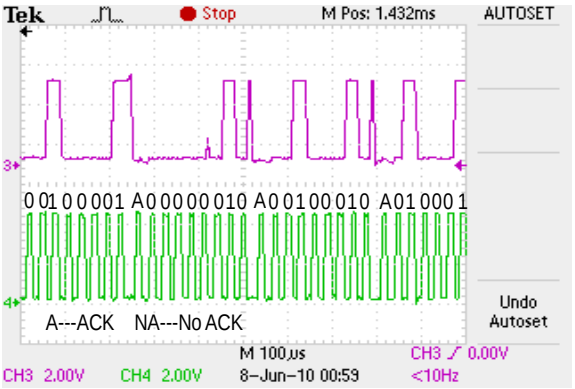
(e)



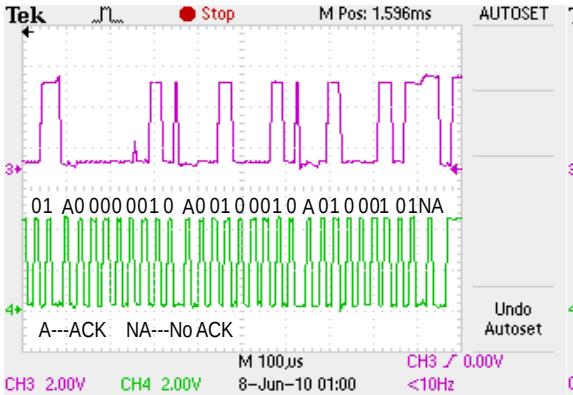
(f)



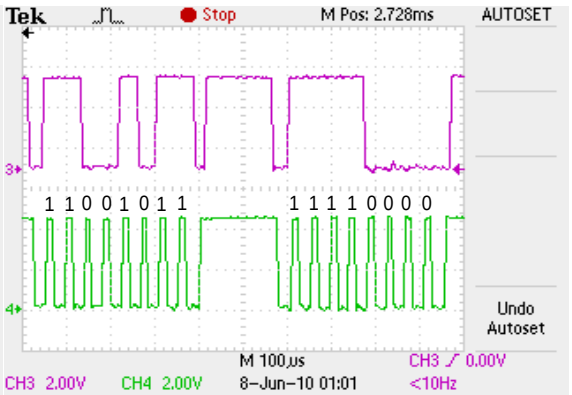
(g)



(h)

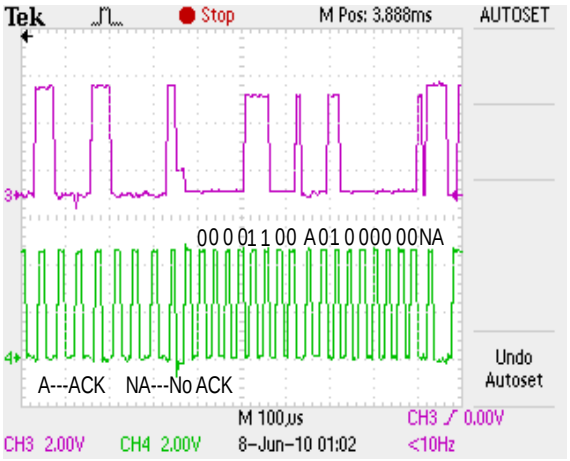


(i)

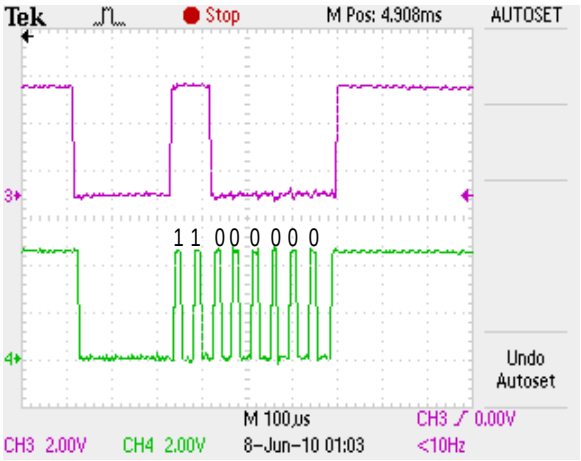


(j)

Figure 4.9 Continued



(k)



(l)

Figure 4.9 Continued

Figures 4.10 to 4.13 give the other four sample temperature/CO2 sensors integrated system's readings. The feedback messages' waveforms are included. In each Figure, Subfigure (a) shows the 7-bit address of the CO2 sensor and 1 read/write bit. Figures 4.10 (b) and (c) show that the CO2 sensor readings are 0000001111010100 (3D4 in Hex), which is 980 ppm in decimal. Figure 4.11 shows a CO2 reading of is 0000010000111000 (438 in Hex), which is 1080 ppm in decimal. Figure 4.12 gives a CO2 reading of 0000010100111110 (53E in Hex), which is 1342 ppm in decimal. Figure 4.13 gives a CO2 reading of 0000001010010010 (292 in Hex), which is 658 ppm in decimal. Subfigures (d) in all Figures 4.10 to 4.13 show the CO2 sensor's two control sequences. The 1st sequence is to validate the correct communications, while the 2nd sequence is a feedback message. As shown in those Subfigures, the 1st sequence value is 11001011 (CB in Hex), which demonstrates that the CO2 sensor reading is correct. The 2nd sequence value is 11100000 (E0 in Hex), which indicates that the CO2 sensor reading is above the upper threshold value. Subfigures (e) in all Figures 4.10 to 4.13 show the temperature sensor readings. They are 0001000000001000 (1008 in Hex) in Figure 4.10, which is 32.0625℃, 0000111011001000 (EC8 in Hex) in Figure 4.11, which is 29.5625℃, 0000111101100000 (F60 in Hex) in Figure 4.12, which is 30.75℃,

0000111111111 (1FF in Hex) in Figure 4.13, which is 31.9375℃. Subfigures (f) in all Figures 4.10 to 4.13 show the temperature sensor's feedback messages. They show a reading of 11000000 (C0 in Hex), indicating that the temperature sensor readings are within the normal range of the HVAC system.

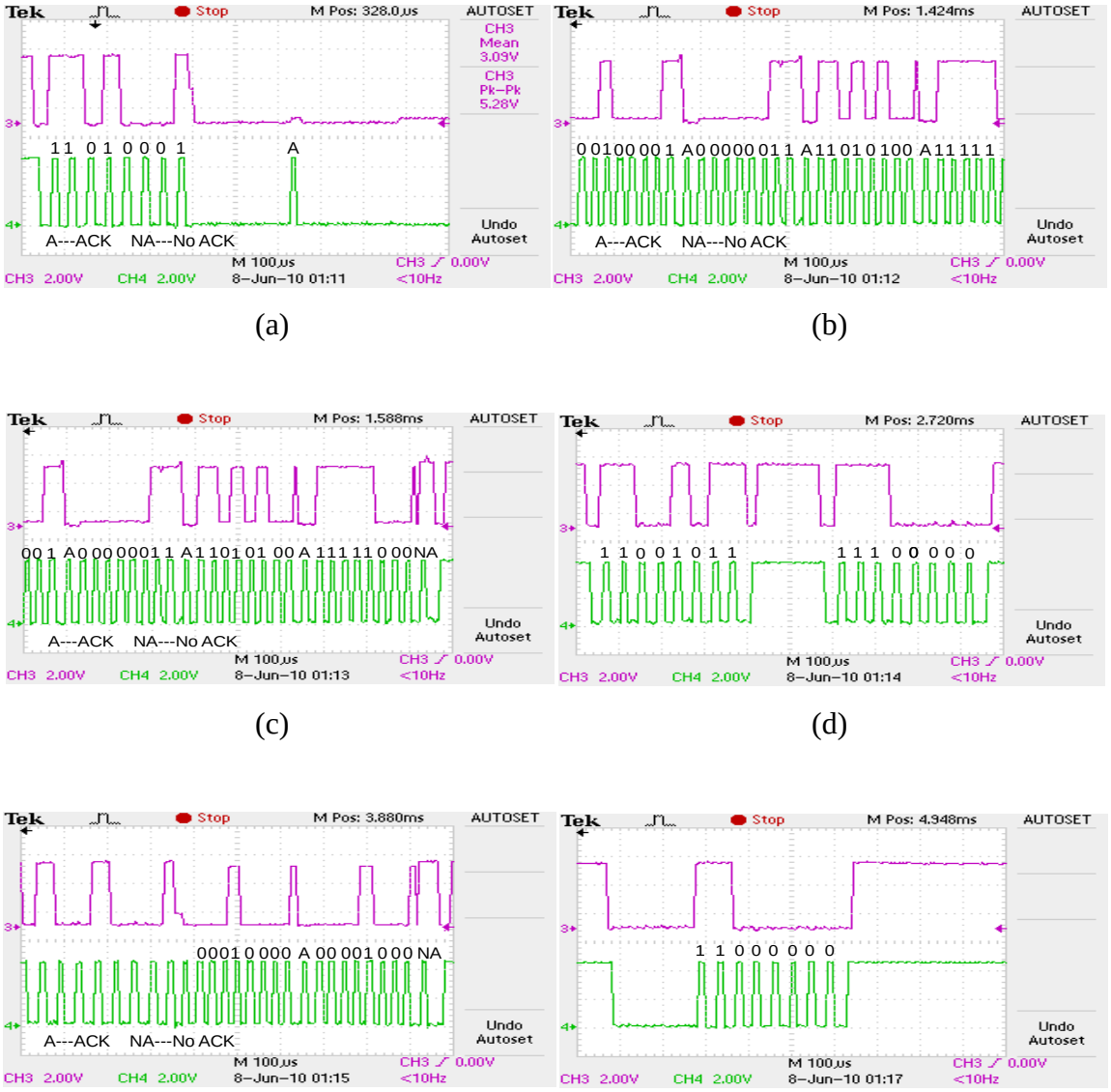
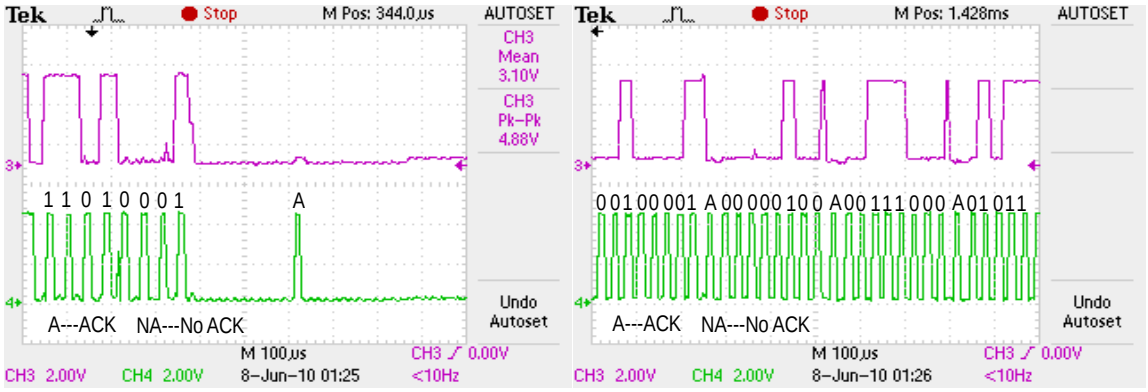
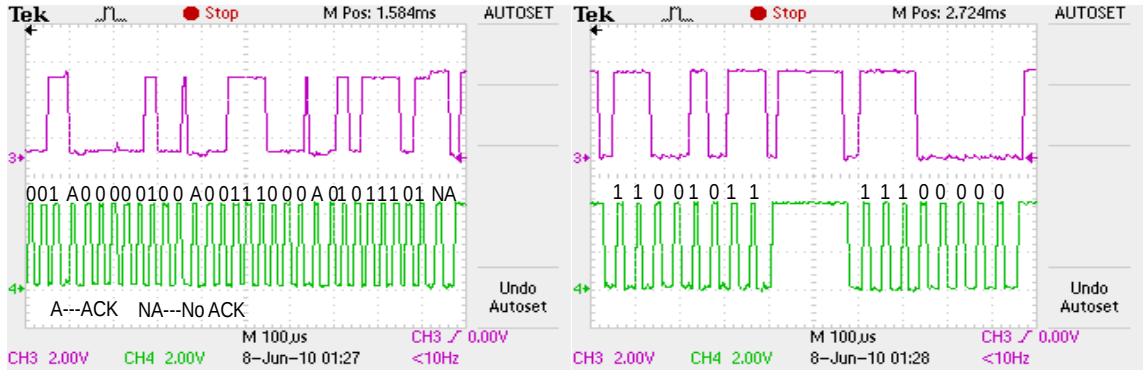


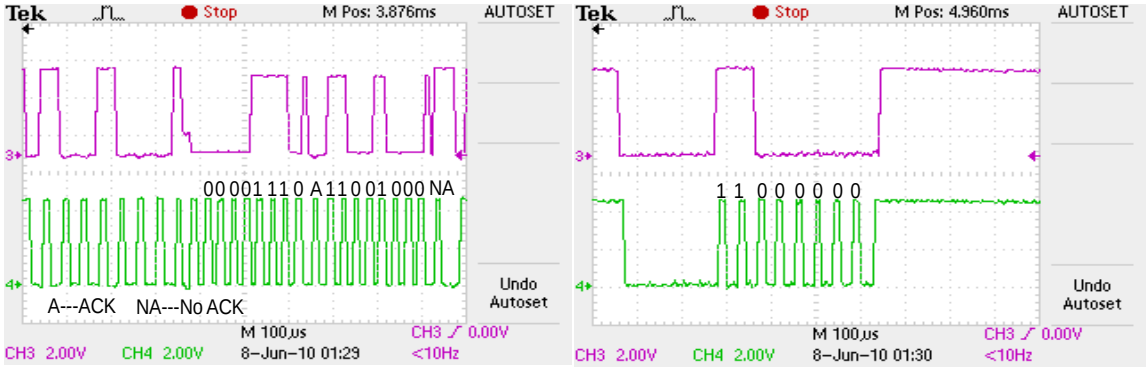
Figure 4.10 (a)-(f) 1st Sample Data Set of Temperature Sensor/CO2 Sensors Integrated System's Readings (with Feedback Messages)



(a) (b)

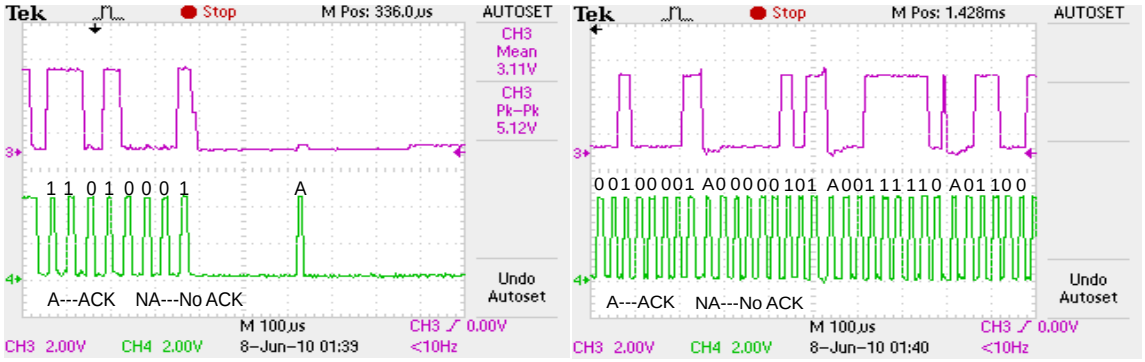


(c) (d)

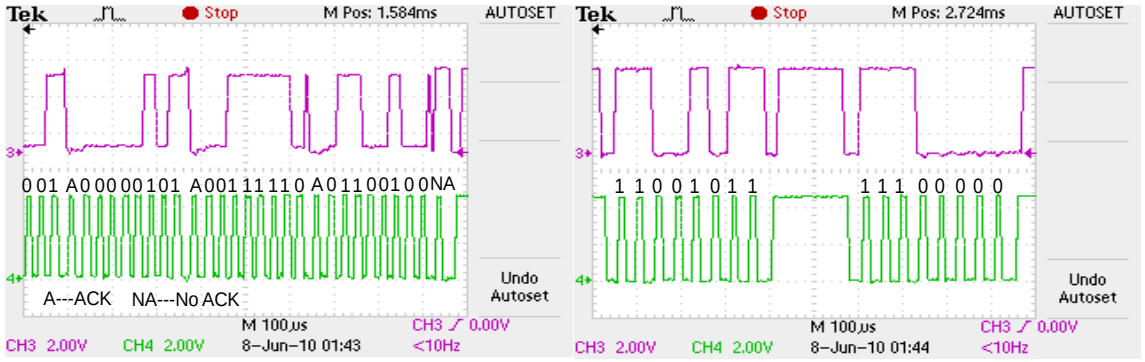


(e) (f)

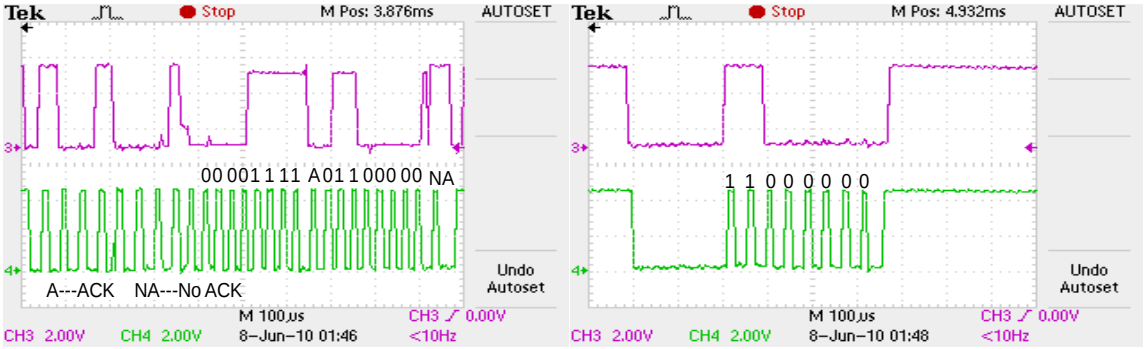
Figure 4.11 (a)-(f) 2nd Sample Data Set of Temperature Sensor/CO2 Sensors Integrated System's Readings (with Feedback Messages)



(a) (b)

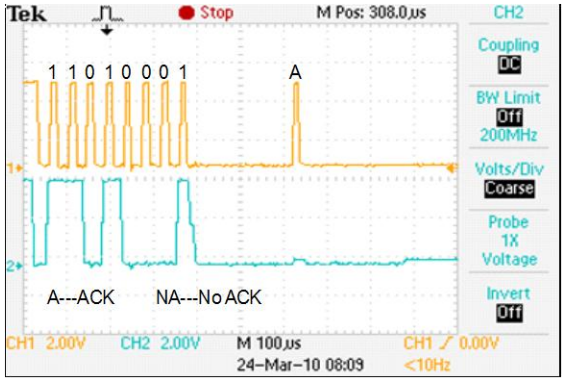


(c) (d)

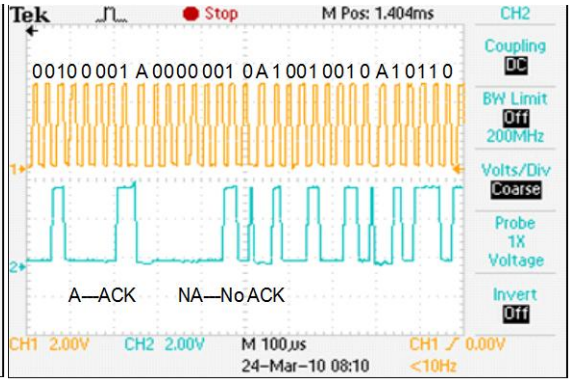


(e) (f)

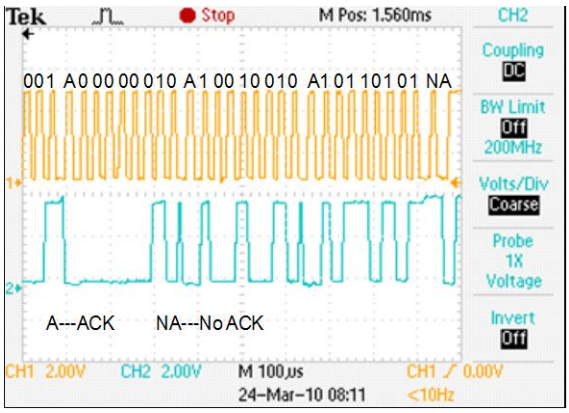
Figure 4.12 (a)-(f) 3rd Sample Data Set of Temperature Sensor/CO2 Sensors Integrated System's Readings (with Feedback Messages)



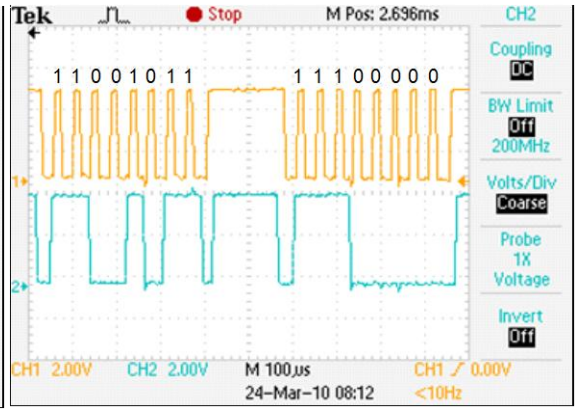
(a)



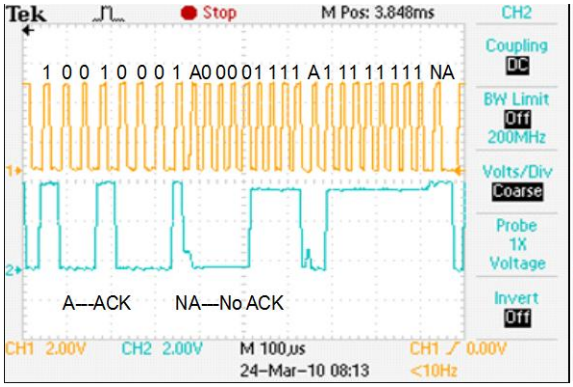
(b)



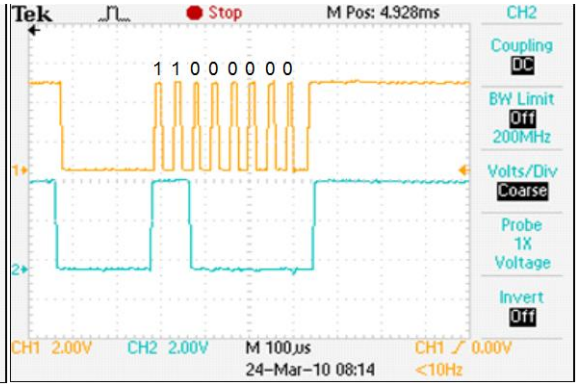
(b)



(d)



(e)



(f)

Figure 4.13 (a)-(f) 4th Sample Data Set of Temperature Sensor/CO2 Sensors Integrated System's Readings (with Feedback Messages)

5. CONCLUSION AND FUTURE WORK

This chapter concludes our findings and summarizes the future work for improving the system by equipping it with high-tech wireless.

5.1 Conclusion

In this work, a smart system board encompassing HVAC embedded system has been implemented. Temperature sensor LM76 and CO2 sensor K22L0 were utilized. The HVAC system has high accuracy and fast response to environmental parameters change. The sensor readings can help manage the control circuits in real time mode. The results of the overall system indicate that the design criteria were met. In comparison with the current existing devices in HVAC application, this design features high speed response in real time mode, design flexibility in accommodating more sensors, cost effective, and compactness.

The ASIC integration part combining sensors and processor is a unique feature of this design. The FPGA chip is designed to balance the speed difference between the controller and the sensor. The popular embedded processor PIC18 with I2C protocol used in the project made it of a cost effective design.

5.2 Future Work

In this work, we have demonstrated the design with two sensors, temperature and CO2 sensors, however, more sensors can be added to have full control of a HVAC system. This may include optical/electromagnetic sensors, light sensors, pressure sensors,

miscellaneous detectors, smoke detector, motion detector, exterior siren, interior siren, glass break and home exits, etc. The expansion of adding more sensors is reserved for future consideration. This is primarily based on the board capability. Figure 5.1 demonstrates a HVAC system with more sensors added.

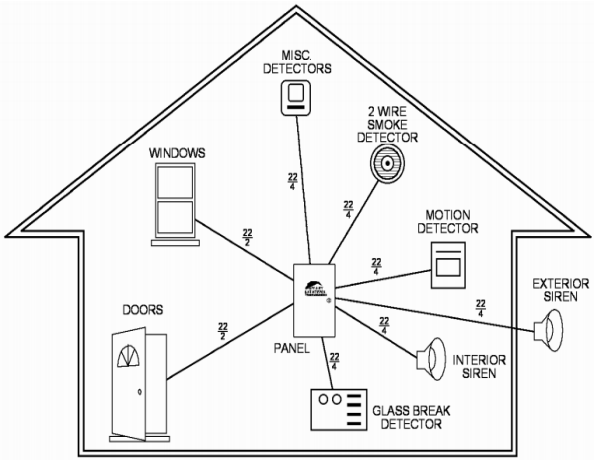


Figure 5.1 HVAC Sensors System

Another expansion for the proposed HVAC system is to include connections related to internet/Ethernet, telephone lines, and wireless to the embedded system. This will allow the HVAC system manage larger areas in buildings and houses. In this expansion, the sensor communication system can be interfaced to a computer by a motherboard in which either Modem or Ethernet card can be used to get/receive messages on the internet or on the cell phone. The alarming signals can be detected by the embedded system and be sent out of the motherboard by wireless communications via internet or phone line. Figure 5.2 shows the mother board from Smart Systems Incorporation. It can be used for this purpose. The alarming signals from the embedded system designed in this project will be decoded and fed into the motherboard, and later be transmitted to outside control circuits by wireless communications via internet or phone line.

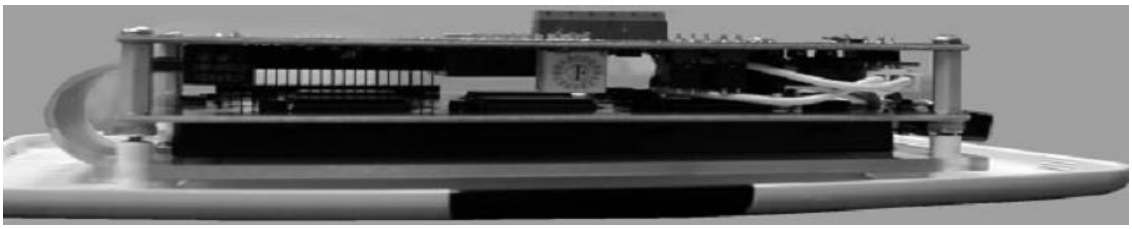


Figure 5.2 The Mother Board from Smart Systems Incorporation [62]

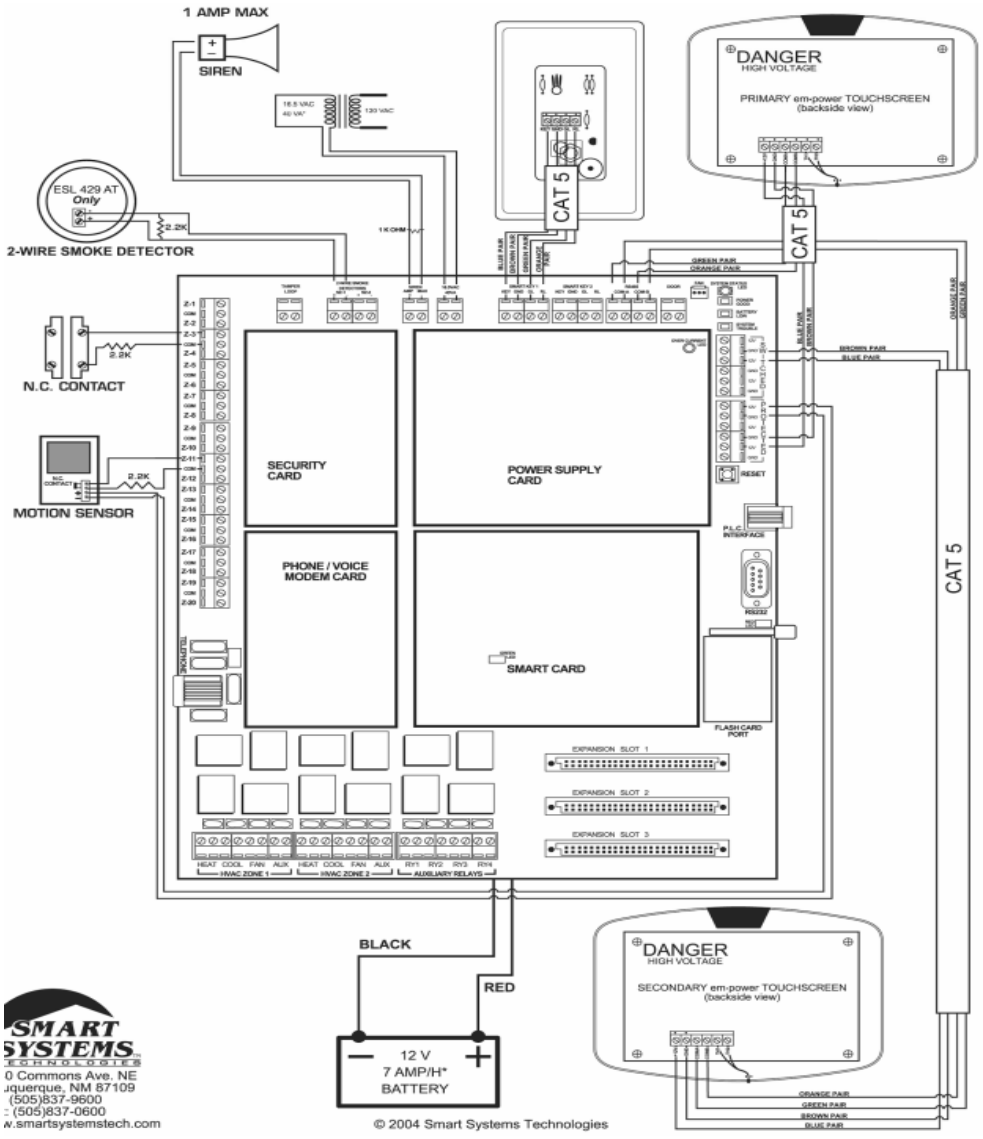
Figure 5.3 shows the set up for the motherboard with several types of sensors (available at Smart Systems Incorporation). Those sensors include optical and humidity sensor. This can enhance the security level for the HVAC applications. The addition of the optical sensor allows detection of actions of the alarm system, such as windows and door security. Some other types of sensors, such as smoke detector, motion sensor, touch screen, sire, etc. can also be implemented in the embedded system.

One drawback of this design is related to the absence of the interrupt service/priority service between sensors. Therefore, in this design, one sensor is allowed to communicate with the processor at a time. In the future, the system can be redesigned to include interrupt service/priority service. With this feature, various parameters can be prioritized according to their importance.

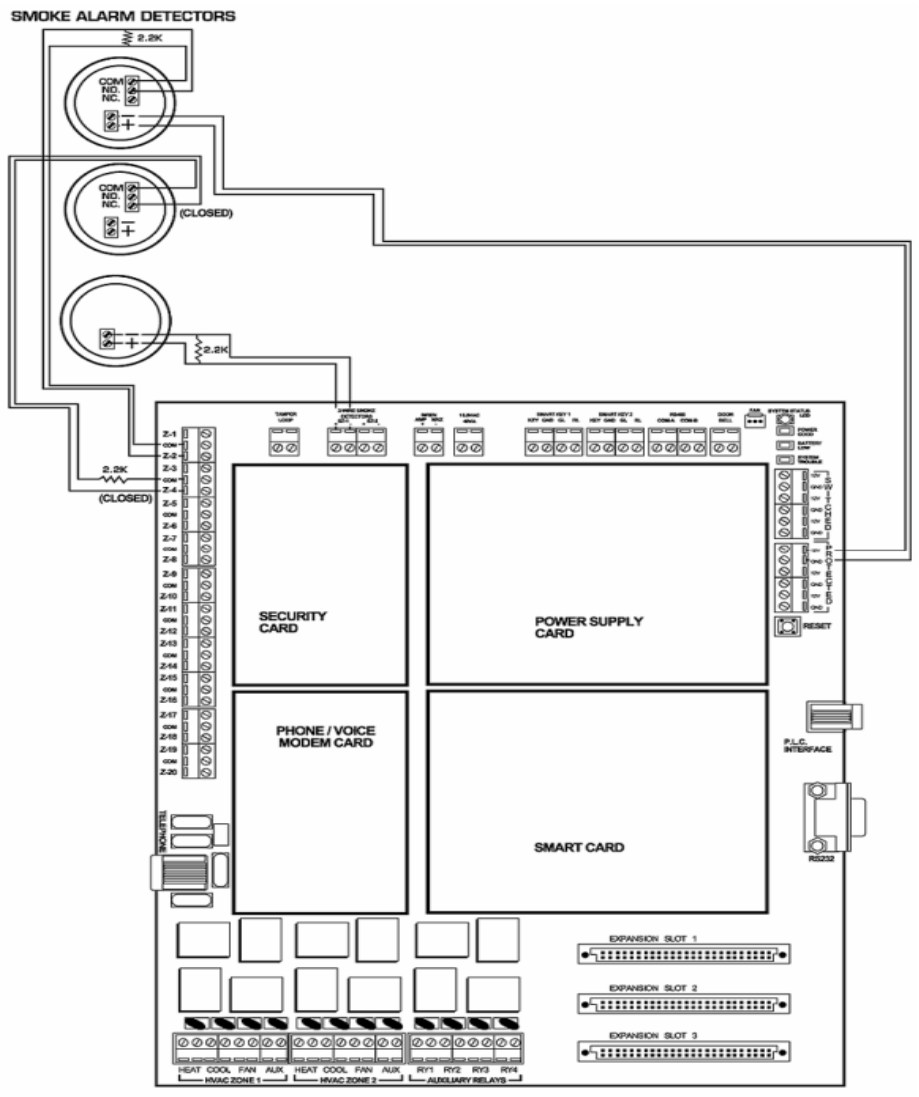
FPGA chip can be modified to be in control of the communication activities. In this case, the microcontroller can be omitted. With this future consideration, the integrated system will be more compact with better cost effectiveness and high efficiency.

The integrated system including FPGA as a controller can be portable. In this case, a battery system may be utilized, and minimum power consumption should be advantageous. The power estimation of the integrated system is reserved for future consideration.

The minimum number of sensors that can be supplied depends on the number of address bits and power delivery effectiveness. A modification to the address bus line may result in maximizing sensor system capability.



(A)



(B)

Figure 5.3 The Mother Board with Several Types of Sensors [62]

Using these mother boards, Figure 5.1 can have the solution shown in Figure 5.4.

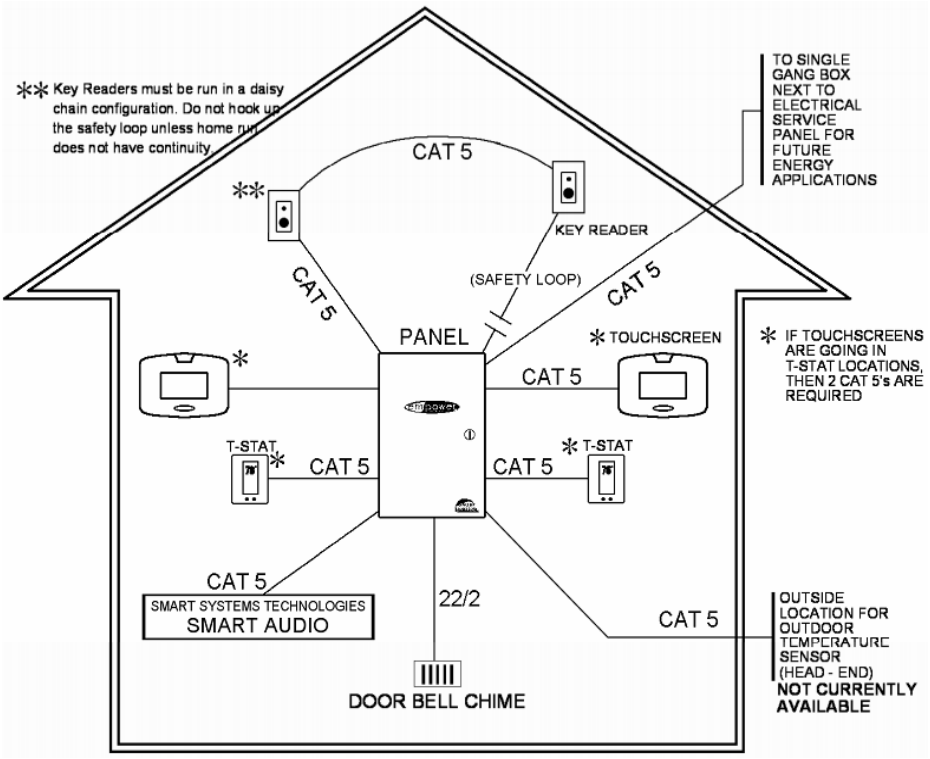


Figure 5.4 The Solution to Implement More Sensors in the Embedded System

The software of this work will be available from the Department of Electrical and Computer Engineering, Indiana University Purdue University Indianapolis and can be obtained from the Appendix of the paper.

LIST OF REFERENCES

LIST OF REFERENCES

[1] S. P. Sreelal, S. P. Sumadevi, P. R. Vinod, T. Varghese, S. Pillai, M. Pillai, and K. Damodaran, "A high resolution integrated data acquisition system for aerospace applications," in *2004 AIAA/IEEE Digital Avionics Systems Conference*, October 2004, pp. 13.E.3-1 - 13.E.3-10.

[2] J. Liu, I. Demirkiran, T. Yang, and A. Helfrick, "Communication schemes for aerospace wireless sensors," in *2008 AIAA/IEEE Digital Avionics Systems Conference*, October 2008, pp. 5D41-5D49.

[3] K. Ku, R. Bradbeer, P. Hodgson, K. Lam, and L. Yeung, "A low-cost, three-dimensional and real-time marine environment monitoring system, Databuoy with connection to the internet," in *2008 OCEANS'08, MTS/IEEE Kobe-Techno-Ocean'08 - Voyage toward the Future, OTO'08*, April 2008, Article number 4530964.

[4] J. Sorribas, R. Del, J. Trullols, and E. A. Manuel, "A smart sensor architecture for marine sensor networks," in *2006 International Conference on Networking and Services*, July 2006, Article number 1690563.

[5] I. Lucifredi, P. J. Stein, Alix. L. Kevin, J. C. Herman, A. S. Frankel, W. T. Ellison, D. E. Egnor, C. W. Clark, and D. DeProspo, "Integrated marine mammal monitoring and protection system (IMAPS), in *2005 MTS/IEEE OCEANS*, September 2005, Article number 1640170.

[6] N. E. Cater and P. Eng, "Smart ocean sensors web enabled ocean sensors for aquaculture," in *2008 OCEANS*, September 2008, Article number 5151870.

[7] D. Liao and K. Sarabandi, "Network of RF ground sensors for applications in precision agriculture," in *2006 IEEE International Geoscience and Remote Sensing Symposium, IGARSS*, August 2006, pp. 3943-3944.

[8] W. J. Fleming, "New automotive sensors a review," *IEEE Sensors Journal*, volume 8, issue 11, pp. 1900-1921, November 2008.

[9] R. Abachi, "Overview of automotive sensors," *Sensors*, volume 13, issue 4, pp. 82-85, April 1996.

[10] Z. Escudero, M. Mai, and A. SantaFe, "Temperature sensor for medical applications based on erbium doped optical fiber," in *2003 Annual International Conference of the IEEE Engineering in Medicine and Biology*, volume 4, September 2003, pp. 3444-3445.

[11] M. Arima, E. H. Hara, and J. D. Katzberg, "Fuzzy logic and rough sets controller for HVAC systems," in *1995 IEEE WESCANEX Communications, Power, and Computing Conference.* Part 1 (of 2), May 1995, pp. 133-138.

[12] "TI Solution of HVAC System," http:sssss//focus.ti.com/docs/solution/folders/print/399.html; Last accessed June 2010.

[13] J. Gan, L. Yuan, Z. Sheng, and T. Xu, "Construction and implementation of an integrated WSID traffic monitoring network system," in *2009 Chinese Control and Decision Conference*, June 2009, pp. 4726-4731.

[14] Y. Zhang, X. Huang, and L. Cui, "Lightweight signal processing in sensor node for real-time traffic monitoring," in *2007 International Symposium on Communications and Information Technologies*, October 2007, pp. 1407-1412.

[15] X. Li, W. Shu, M. Li, H. Huang, P. Luo, and M. Wu, "Performance evaluation of vehicle-based mobile sensor networks for traffic monitoring," *IEEE Transactions on Vehicular Technology*, volume 58, issue 4, pp. 1647-1653, 2009.

[16] X. Laisheng, P. Xiaohong, W. Zhengxia, X. Bing, and H. Pengzhi, "Research on traffic monitoring network and its traffic flow forecast and congestion control model based on wireless sensor networks," in *2009 International Conference on Measuring Technology and Mechatronics Automation*, April 2009, pp. 142-147.

[17] H. Cai, M. Wu, and Z. Cui, "An integrated optical sensor for online monitoring of lactate concentration," in *2005 Fourth IEEE Conference on Sensors*, November 2005, pp. 967-970.

[18] B. Pain, C. Sun, C. Wrigley, and G. Yang, "Dynamically reconfigurable vision with high performance CMOS active pixel sensors(APS)," in *2002 First IEEE International Conference on Sensors*, volume 1, issue 1, June 2002, pp. 21-26.

[19] A. Crespi, Y. Gu, N. Bellini, K. C. Vishnubhatla, R. Ramponi, R. Osellame, and G. Cerullo, "Femtosecond laser fabrication of optical sensors integrated in a lab-on-a-chip," in *2009 International Symposium on Optomechatronic Technologies*, September 2009, pp. 392-397.

[20] A. Simoni, L. Gonzo, and M. Gottardi, "Integrated optical sensors for 3-D vision," in *2002 First IEEE International Conference on Sensors*, June 2002, pp. 1-4.

[21] R. A. Rahim, K. T. Chiam, M. H. F. Rahiman, and P. Jayasuman, "Velocity profile measurement using digital signal processor-based optical tomography system," *IEEE Sensors Journal*, volume 9, issue 9, pp. 1076-1083, September 2009.

[22] S. H. Yoon and D. J. Kim, "Fabrication and characterization of ZnO films for biological sensor application of FPW device," in *2006 15th IEEE International Symposium on Applications of Ferroelectrics*, August 2006, pp. 1-4.

[23] A. Delâge, D. X. Xu, A. Densmore, S. Janz, P. Waldron, J. Lapointe, B. Lamontagne, T. Mischki, G. Lopinski, J. Sehmid, and P. Cheben, "Label-free biological sensors based on ring resonators," in *2008 10th Anniversary International Conference on Transparent Optical Networks*, June 2008, pp. 2-5.

[24] M. Lanzoni, C. Stagni, and B. Riccò, "Smart sensors for fast biological analysis," in *2007 2nd IEEE International Workshop on Advances in Sensors and Interfaces*, June 2007, pp. 1-5.

[25] N. I. Maluf, G. T. A. Kovacs, and D. Gee, "Recent advances in medical applications of MEMS," in *1996 Wescon Conference*, October 1996, pp. 60-63.

[26] W. Zhao, J. Xing, and C. Ju, "A real-time data monitoring system based on sensor network protocol," in *2009 Second ISECS International Colloquium on Computing, Communication, Control, and Management*, August 2009, pp. 177-181.

[27] T. De Laet, W. Deeré, J. Rutgeerts, H. Bruyninckx, and J. De Schutter, "An application of constraint-based task specification and estimation for sensor-based robot systems," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, November 2007, pp. 1658-1664.

[28] A. M. Zaitsev, A. M. Levine, and S. H. Zaidi, "Temperature and chemical sensors based on FIB-written carbon nanowires," *IEEE Sensors Journal*, volume 8, issue 6, pp. 849-856, June 2008.

[29] O. Tigli and M. E. Zaghloul, "Design and fabrication of a novel SAW Bio chemical sensor in CMOS," in *2005 Fourth IEEE Conference on Sensors*, November 2005, pp. 137-140.

[30] J. Marek, "MEMS for automotive and consumer electronics," in *2010 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, February 2010, pp. 9-17.

[31] Q. Tan, W. Zhang, C. Xue, J. Xiong, J. Li, L. Ting, and Y. Shi, "The characterization and fabrication of pyroelectric infrared sensors and application of gas monitoring," in *2008 2nd IEEE International Nanoelectronics Conference*, March 2008, pp. 776-781.

[32] J. Zhang, X. Ning, H. Chen, and C. L. King Wai, "Fabrication and experimental testing of individual multi-walled carbon nanotube(CNT) based infared sensors," *IEEE Sensors*, pp. 511-514, October 2007.

[33] M. G. Zanchi, R. Venook, J. M. Pauly, and G. C. Scott, "An Optically Coupled System for Quantitative Monitoring of MRI-Induced RF Currents Into Long Conductors," *IEEE Transactions on Medical Imaging*, volume 29, issue 1, pp. 169-178, January 2010.

[34] R. Lenggenhager, D. Jaeggi, P. Malcovati, H. Duran, H. Baltes, and E. Doering, "CMOS membrane infrared sensors and improved TMAHW etchant," in *1994 IEEE International Electron Devices Meeting*, December 1994, pp. 531-534.

[35] M. C. Hsieh, Y. K. Fang, P. M. Wu, C. C. Yang, Y. C. Lin, W. D. Wang, S. F. Ting, and J. J. Ho, "Design and fabrication of a novel crystal SiGeC far infared sensor with wavelength 8-14 micrometer," *IEEE Sensors Journal*, volume 2, issue 4, pp. 360-365, Aug 2002.

[36] J. Wang, G. Li, Y. Liu, Y. Lu, X. Gao, Y. Zhang, and K. Tao, "Vehicle Supervision System Based on MEMS Geomagnetic Sensor," in *2009 IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, January 2009, pp. 331-334.

[37] J. Trontelj, "Optimization of Integrated Magnetic Sensor by Mixed Signal Processing," in *1999 16th IEEE Instrumentation and Measurement Technology Conference - Measurements for the new Millenium*, volume 1, June 1999, pp. 299-302.

[38] S. Beeby, G. Ensell, M. Kraft, and N. White, *MEMS Mechanical Sensors*. Artech House Incorporation, 2004.

[39] G. K. Fedder, "CMOS-Based Sensors," in *2005 Fourth IEEE Conference on Sensors*, November 2005, pp. 125-128.

[40] E. Aldrete-Vidrio, D. Mateo, and J. Altet, "Differential temperature sensors fully compatible with a 0.35-um CMOS process," *IEEE Transactions on Components and Packaging Technologies*, volume 30, issue 4, pp. 618-626, December 2007.

[41] R. Gharpurey and E. Charbon, "Substrate coupling modeling simulation and design perspectives," in *2004 5th International Symposium on Quality Electronic Design*, March 2004, pp. 283-290.

[42] O. C. Allegranza and T. Wu, "Soft error rate results of thin film media on glass substrates," in *1995 33rd Annual IEEE International Magnetics Conference, Part 1 (of 3)*, volume 31, issue 6 pt 1, April 1995, pp. 2797-2799.

[43] M. S. Rahaman, M. H. Chowdhury, I. Nasir, and L. T. Hwang, "VSIB: A Sensor Bus Architecture for Smart-Sensor Network," in *2009 WRI World Congress on Computer Science and Information Engineering*, volume 3, April 2009, pp. 436-439.

[44] Y. Sun and D. Zhu, "Design and research of RS232 communication circuit used for 0.6um CMOS sensor IC," in *2006 8th International Conference on Solid-State and Integrated Circuit Technology*, October 2006, pp. 557-559.

[45] J. Diaz, E. Rodriguez, L. Hurtado, H. Cacique, N. Vazquez, and A. Ramirez, "CAN Bus Embedded System for Lighting Network Applications," in *2008 IEEE International 51st Midwest Symposium on Circuits and Systems*, August 2008, pp. 531-534.

[46] L. D. De Almeida Pereira Zuquim, C. J. N. Coelho, A. O. Jr., Fernandes, M. P. de Oliveira, and A. I. Tavares, "An Embedded Converter from RS232 to Universal Serial Bus," in *2001 14th Symposium on Integrated Circuits and Systems Design*, September 2001, pp. 91-96.

[47] R. Kambhatla, S. Kumar, and J. D. Matyjas, "Reliable multihop remote sensing and control using wireless sensor and actor networks," in *2008 IEEE Military Communications Conference - Assuring Mission Success*, November 2008, article number 4753641.

[48] T. Nadeem and A. Agrawala, "Performance of IEEE 802.11 based wireless sensor networks in noisy environments," in *2005 24th IEEE International Performance, Computing, and Communications Conference*, April 2005, pp. 471-476.

[49] J. Liu, I. Demirkiran, T. Yang, and A. Helfrick, "Feasibility study of IEEE 802.13.4 for aerospace wireless sensor networks," in *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, October 2009, pp. 1.B.3-1 - 1.B.3-10.

[50] G. Cancelo and M. Mayosky, "A parallel analog signal processing unit based on radial basis function networks," in *1997 Nuclear Scienc Symposium & Medical Imaging, Part 1 (of 2)*, volume 45, issue 3 part 1, November 1997, pp. 792-797.

[51] R. Dlugosz and K. Iniewski, "Novel CMOS Analog Signal Processing Technique for Solid-State X-Ray Sensors," in *2007 IEEE North-East Workshop on Circuits and Systems*, August 2007, pp. 770-771.

[52] M. Li, and M. Liu, "Micro-system for shock and vibration measurement," in *2009 9th International Conference on Electronic Measurement and Instruments*, August 2009, pp. 3920-3923.

[53] C. F. Chiasserini, and R. Rao, "On the concept of distributed digital signal processing in wireless sensor networks," in *2002 MILCOM Proceedings; Global Information Grid - Enabling Transformation Through 21st Century Communications*, volume 1,October 2002, pp. 260-264.

[54] M. Saukoski, L. Aaltonen, T. Salo, and K. Halonen, "Readout electronics with bandpass delta-sigma A-D converter for a bulk micromachined capacitive gyroscope," in *2005 IEEE Instrumentation and Measurement Technology Conference*, volume 2, May 2005, pp. 769-774.

[55] B. Tongprasit, K. Ito, and T. Shibata, "A computational digital-pixel-sensor VLSI featuring block-readout architecture for pixel-parallel rank-order filtering," in *2005 IEEE International Symposium on Circuits and Systems*, May 2005, pp. 2389-2392.

[56] V. C. Fernicola, and L. Crovini, "Digital signal processing for fiber-optic thermometers," in *1994 Conference on Precision Electromagnetic Measurements*, volume 44, issue 2, July 1994, pp. 447-450.

[57] B. Griffin, and M. J. Connelly, "Digital signal processing of interferometric fiber optic sensors," in *2004 1st IEEE Lightwave Technologies in Instrumentation and Measurement Conference*, October 2004, pp. 153-156.

[58] S. Mikko, A. Lasse, H. Kari, and S. Teemu, "Integrated readout and control electronics for a micro-electro-mechanical angular velocity sensor," in *2006 32nd European Solid-State Circuits Conference*, September 2006, pp. 243-246.

[59] J. Trontelj, "Optimization of integrated magnetic sensor by mixed signal processing," in *1999 16th IEEE Instrumentation and Measurement Technology Conference - Measurements for the new Millenium*, volume 1, June 1999, pp. 299-302.

[60] D. Dammers, C. Domingues, D. Schollän, and L. M. Vokämper, "Mixed signal system design verification accelerated with detector-based diagnostic method," in *2009 IEEE International Behavioral Modeling and Simulation Workshop*, September 2009, pp. 66-72.

[61] H. Song, “Novel mixed domain VLSI signal processing circuits for high performance low power and area penalty SOC signal processing,” in *2008 IEEE International SOC Conference*, September 2008, pp. 309-312.

[62] “Smart Systems Incorporation product datasheet,” http://www.smartsystemstech.com/flash_index.htm; Last accessed June 2010.

[63] “MAXIM datasheet,” http//datasheet.octopart.com/MAX232ACWE%2B-Maxim-datasheet-3759.pdf; Last accessed June 2010.

[64] A. Bravo and P. M. Djuric, "Cooperative relay communications in mesh network," in *2009 IEEE 10th Workshop on Signal Processing Advances in Wireless Communications*, June 2009, pp. 499-503.

[65] “PIC18 data sheet,” http://ww1.microchip.com/downloads/en/DeviceDoc/41159e.pdf; Last accessed June 2010.

[66] S. Grassini, D. Mombello, A. Neri, M. Parvis, and A. Vallan, “Plasma deposited thermocouple for non-invasive temperature measurement,” in *2009 IEEE Intrumentation and Measurement Technology Conference*, May 2009, pp. 732-736.

[67] R. F. Dos and A. Carlos, “An integrated 4-20-mA Two-wire transmitter with intrinsic temperature sensing capability,” *IEEE Journal of Solid-State Circuits*, volume 24, issue 4, pp. 1136-1142, August 1989.

[68] Z. Ma, X. Zheng, W. Liu, X. Lin, and W. Deng, “Fast thermal resistance measurement of high brightness LED,” in *2005 6th International Conference on Electronics Packaging Technology*, September 2005, article number 1564685.

[69] W. S. Lee, and K. Y. Byun, “The availability of the thermal resistance model in flip-chip packages,” in *2007 International Conference on Electronic Materials and Packaging*, November, 2007, article number 4510327.

[70] M. Afridi, C. B. Montgomery, E. Cooper-Balis, S. Semancik, K. G. Kreider, and J. Geist, “Analog BIST functionality for micro hotplate temperature sensors,” *IEEE Electron Device Letters*, volume 30, issue 9, pp. 928-930, September 2009.

[71] J. Bonhaus, D. Borchert, A. Denisenko, and W. R. Fahrner, “Diamond heat spreaders for high power devices with integrated temperature sensors,” in *1998 IEEE 14th Annual Semiconductor Thermal Measurement and Management Symposium*, March 1998, pp. 139-146.

[72] N. Wang, B. Sadoulet, T. Shutt, J. Beeman, E. E. Haller, A. Lange, I. Park, R. Ross, C. Stanton, and H. Steiner, "A 20mK temperature sensor," *IEEE Transactions on Nuclear Science*, volume 35, issue 1, pp. 55-58, February 1988.

[73] H. Van Haren, M. Laan, D. J. Buijsman, L. Gostiaux, M. G. Smit, and E. Keijzer, "NIOZ3 independent temperature sensors sampling yearlong data at a rate of 1Hz," *IEEE Journal of Oceanic Engineering*, volume 34, issue 3, pp. 315-322, July 2009.

[74] "Temperature sensor LM76 data sheet," http://www.national.com/ds/LM/LM76.pdf; Last accessed June 2010.

[75] Z. Chen and M. C. Jin, "An Alpha-alumina moisture sensor for relative and absolute humidity measurement," in *1992 Industry Applications Society Annual Meeting*, volume 2, October 1992, pp. 1668-1675.

[76] Y. Wang, J. N. Chen, D. M. Ke, and J. Hu, "Modeling of a CMOS capacitive relative humidity sensor," in *2009 1st International Workshop on Education Technology and Computer Science*, volume 3, March 2009, pp. 209-212.

[77] M. E. MacDonald, E. C. Wack, M. W. Kelly, D. P. Ryan-Howard, M. M. Coakley, D. M. Weitz, H. R. Finkle, D. E. Weidler, G. W. Carlisle, and L. M. Candell, "Architectural trades for an advanced geostationary atmospheric sounding instrument," in *2001 IEEE Aerospace Conference*, volume 4, March 2001, pp. 41693-41711.

[78] M. Iman, "Integrated multi-sensors for industrial humidity measurement," in *2008 34th Annual Conference of the IEEE Industrial Electronics Society*, November 2008, pp. 1730-1735.

[79] Y. Miyoshi, K. Mitsubayashi, T. Sawada, M. Ogawa, K. Otsuka, and T. Takeuchi, "Wearable humidity sensor with porous membrane by soft-MEMS techniques," in *2005 13th International Conference on Solid-State Sensors and Actuators and Microsystems*, volume 2, June 2005, pp. 1288-1291.

[80] M. Kunze, H. Glosch, B. Ehrbrecht, S. Billat, S. Messner, M. Ashauer, and R. Zengerle, "Thermal dew point sensing a new approach for dew point detection and humidity sensing," in *2007 20th IEEE International Conference on Micro Electro Mechanical Systems*, January 2007, pp. 119-122.

[81] Y. Miyoshi, T. Tkeuchi, T. Saito, H. Saito, H. Kudo, K. Otsuka, and K. Mitsubayashi, "A wearable humidity sensor with hydrophilic membrane by soft-MEMS techniques," in *2007 2nd IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, January 2007, pp. 211-214.

[82] C. Y. Lee and G. B. Lee, "MEMS-based humidity sensors with integrated temperature sensors for signal drift compensation," in *2003 Second International Conference on Sensors: IEEE Sensors*, volume 2, issue 1, October 2003, pp. 384-388.

[83] P. Thoma, J. Colla, and R. Stewart, "A capacitance humidity-sensing transducer," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, volume 2, issue 3, pp. 321-323, September 1979.

[84] A. Tetelin and C. Pellet, "Modeling and optimization of a micro scale capacitive humidity sensor for HVAC applications," *IEEE Sensors Journal*, volume 6, issue 3, pp. 714-720, June 2006.

[85] M. V. Fuke, A. Vijayan, P. Kanitkar, and R. C. Aiyer, "Optical humidity sensing characteristics of Ag-polyaniline nanocomposite," *IEEE Sensors Journal*, volume 9, issue 6, pp. 648-653, June 2009.

[86] J. D. N. Cheeke, N. Tashtoush, and N. Eddy, "Surface acoustic wave humidity sensor based on the changes in the viscoelastic properties of a polymer film," in *1996 IEEE Ultrasonics Symposium. Part 2 (of 2)*, volume 1, November 1996, pp. 449-452.

[87] Z. C. Alex and K. Govardhan, "Design and simulation of MEMS based humidity sensor," in *2005 Asia-Pacific Microwave Conference*, volume 2, December 2005, article number 1606536.

[88] A. Salehi, A. Nikfarjam, and D. J. Kalantari, "Highly sensitive humidity sensor using Pd/porous GaAs Schottky contact," *IEEE Sensors Journal*, volume 6, issue 6, pp. 1415-1421, December 2006.

[89] V. Matko and D. Donlagic, "Highly sensitive humidity sensor for dielectric measurement (water absorption) in glass-fiber resins," in *1997 23rd International Conference on Industrial Electronics, Control and Instrumentation*, volume 3, November 1997, pp. 1148-1151.

[90] S. K. Jae, K. Ki-Young, K. Kwang-Ho, M. Nam-Ki, and K. Moon-Sik, "A locally cured polyimide-based humidity sensor with high sensitivity and high speed," in *2008 IEEE Sensors*, October 2009, pp. 434-437.

[91] F. Granstedt, B. Hok, U. Bjurman, M. Ekstrom, and Y. Backlund, "New CO2 Sensor with high resolution and fast response," in *2001 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 3, 2001, pp. 3100-3103.

[92] A. Haeusler and J. U. Meyer, "A novel CO2 Sensor Based on changes in conductivity of metal oxides," in *1995 8th International Conference on Solid-State Sensors and Actuators*, volume 2, June 1995, pp. 866-869.

[93] C. Hung, L. C. Hsu, T. Ativanichayaphong, J. Sin, H. E. Stephanou, and J. C. Chiao, "An Infant Monitoring System Using CO2 Sensors," in *2007 IEEE International Conference on RFID*, March 2007, pp. 134-140.

[94] "Senseair Incorporation," http://www.senseair.com; Last accessed June 2010.

[95] "Microchip Company and Products," http://www.microchip.com; Last accessed June 2010.

[96] "MAX II Handbook," http://www.altera.com/literature/hb/max2/max2_mii5v1.pdf; Last accessed June 2010.

APPENDICES

Appendix A Relevant Hardware Information

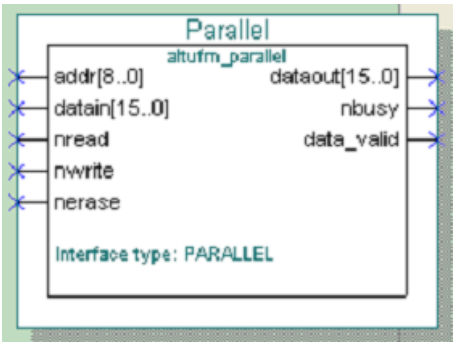
A.1 Parallel Interface Hardware

This interface allows for parallel communication between the UFM block and outside logic. Once the READ request, WRITE request, or ERASE request is asserted (active low assertion), the outside logic or device (such as a microcontroller) are free to continue their operation while the data in the UFM is retrieved, written, or erased. During this time, the nBUSY signal is driven "low" to indicate that it is not available to respond to any further request. After the operation is complete, the nBUSY signal is brought back to "high" to indicate that it is now available to service a new request. If it was the Read request, the DATA_VALID is driven "high" to indicate that the data at the DO port is the valid data from the last read address. Asserting READ, WRITE, and ERASE at the same time is not allowed. Multiple requests are ignored and nothing is read from, written to, or erased in the UFM block. There is no support for sequential read and page write in the parallel interface. For both the read only and the read/write modes of the parallel interface, OSC_ENA is always asserted, enabling the internal oscillator.

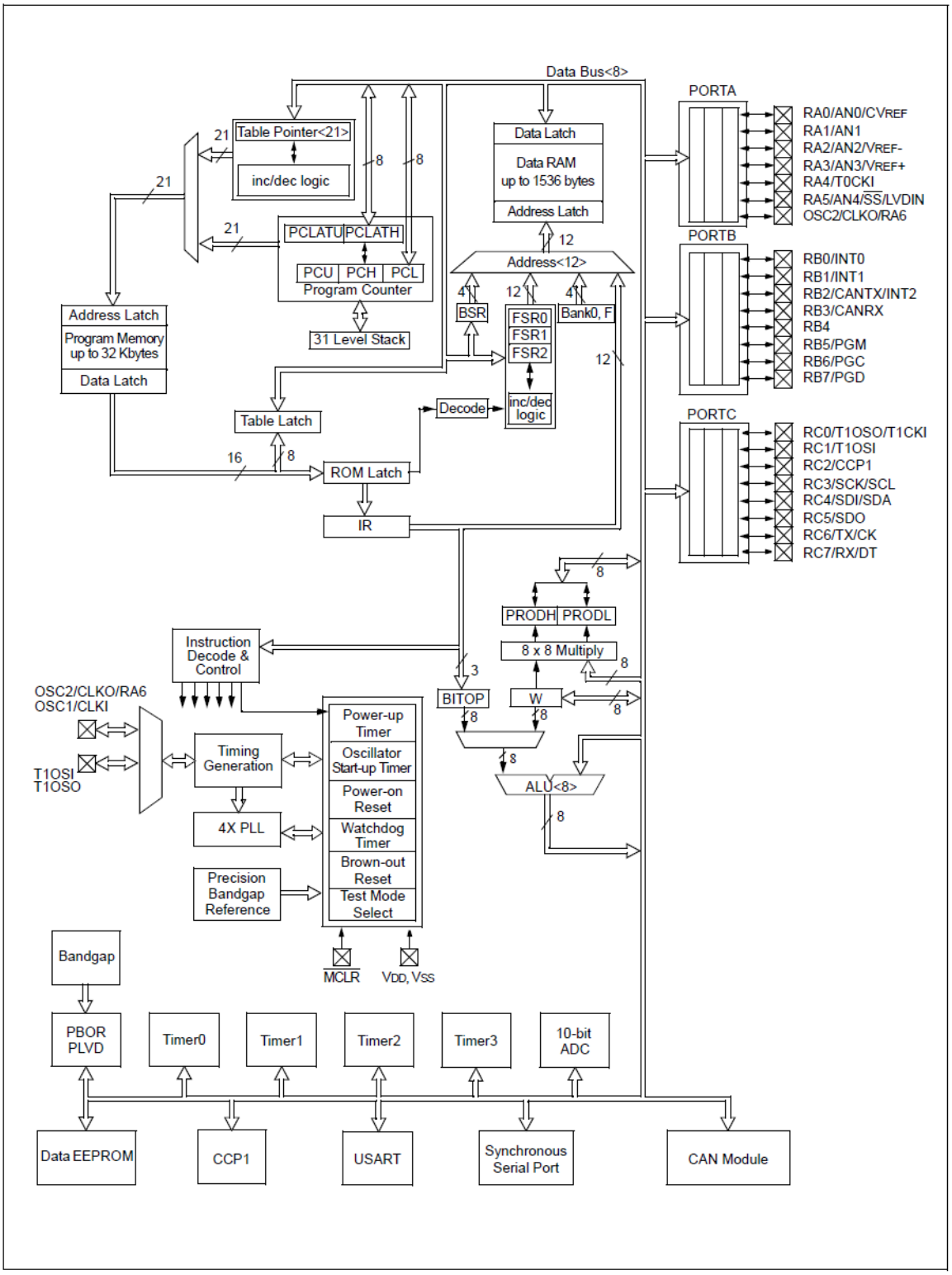
Parallel Interface Signals [96]:

1) DI[15:0], 16-bit data Input. Receive 16-bit data in parallel. You can select an optional width of 3 to 16 bits using the altufm megafunction.
2) DO[15:0], 16-bit data Output. Transmit 16-bit data in parallel. You can select an optional width of 3 to 16 bits using the altufm megafunction.
3) ADDR[8:0], Address Register. Operation sequence refers to the data that is pointed to by the address register. You can determine the address bus width using the altufm megafunction.
4) nREAD, READ Instruction Signal. Initiates a read sequence.
5) nWRITE, WRITE Instruction Signal. Initiates a write sequence.
6) nERASE, ERASE Instruction Signal. Initiates a SECTOR-ERASE sequence indicated by the MSB of the ADDR port.

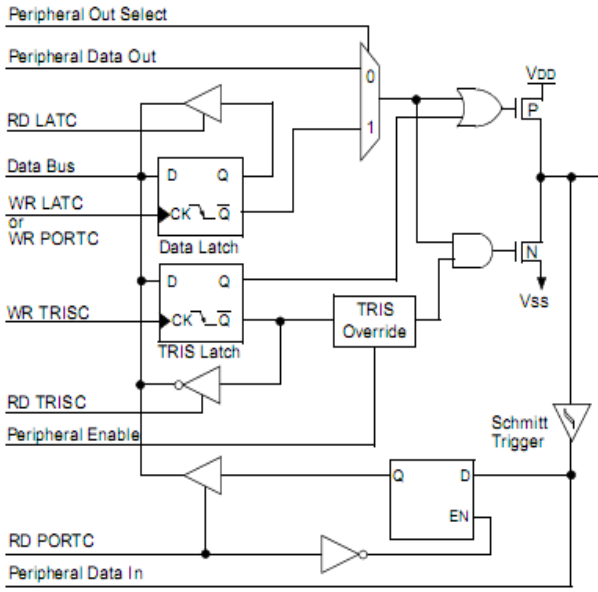
7) nBUSY, BUSY Signal. Driven low to notify that it is not available to respond to any further request.
8) DATA_VALID, Data Valid. Driven high to indicate that the data at the DO port is the valid data from the last read address for read request.



A.2 The Microstructure of the PIC182585 Microcontroller



A.3 PORT C Pins Microstructure



TRIS OVERRIDE

Pin	Override	Peripheral
RC0	Yes	Timer1 OSC for Timer1/Timer3
RC1	Yes	Timer1 OSC for Timer1/Timer3
RC2	No	—
RC3	Yes	SPI/I^2C Master Clock
RC4	Yes	I^2C Data Out
RC5	Yes	SPI Data Out
RC6	Yes	USART Async Xmit, Sync Clock
RC7	Yes	USART Sync Data Out

Appendix B Source Code

There are three source files in the program. Main.c is the main function. i2c_func.c contains all of the sub-functions in the program. i2c.func.h contains all of the sub-functions' head-files. The source files are for the CO2/temperature sensor integrated system. The program contains feedback messages to the outside control circuits. To get the program that do not contain feedback messages, simply comment out the feedback messages generation parts in sub-function "i2c_out_temp" and "i2c_out_CO2" .

To get software programs for sensor system that contains temperature or CO2 sensor individually, simply comment out the line "i2c_out_CO2()" or "i2c_out_temp()" in main.c.

B.1 Main.c

```
#include <p18f2585.h>
#include "i2c_func.h"
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
	long i = 0;
	initial();
	i2c_initial();
	while(1)
	{
		i2c_out_CO2();
		for(i=0; i<10; i++);
		i2c_out_temp();
		for(i=0; i<200000; i++)
		{
			;
		}
	}
}
```

B.2 i2c_func.c

```
#include <i2c.h>
#include "i2c_func.h"
void initial(void)
{
	INTCON = 0;
	PIE1 = 0;
	PIE2 = 0;
	PIE3 = 0;
}
void i2c_initial(void)
{
	SSPCON1 = 0x08;
	TRISC &= 0xE7;
	SSPADD = 0xE0;
	SSPSTAT = 0x80;
	SSPCON2 = 0x00;
	SSPCON1bits.SSPEN = 0;
	LATC |= 0x18;
}
void i2c_start(void)
{
	int i;
	for(i = 5; i > 0; i --);
	PORTCbits.SDA = 0;
	for(i = 5; i > 0; i --);
	PORTCbits.SCL = 0;
	for(i = 5; i > 0; i --);
}
void transmission(unsigned char* temp)
{
	unsigned char mask = 0x80;
	int i, j;
	for(j=0; j<8; j++)
	{
		if(mask & *temp)
		{
			PORTCbits.SDA = 1;
		}
		else
		{
			PORTCbits.SDA = 0;
		}
		for(i = 2; i > 0; i --);
		PORTCbits.SCL = 1;
```

```
                for(i = 5; i > 0; i --);

                PORTCbits.SCL = 0;
                for(i = 5; i > 0; i --);
                mask = mask >> 1;
        }
}
void delay_for_get_ACK_from_Slave(void)
{
        int i = 0;
        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
}
void i2c_stop(void)
{
        int i;
        PORTCbits.SCL = 0;
        PORTCbits.SDA = 0;
        for(i = 5; i > 0; i --);
        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
        PORTCbits.SDA = 1;
        for(i = 5; i > 0; i --);
}
unsigned char receive_byte(void)
{
        int i, sum;
        unsigned char byte;
        TRISC |= 0x10;
        byte = 0x00;
        for(sum = 0; sum < 7; sum ++)
        {
                PORTCbits.SCL = 1;
                for(i = 5; i > 0; i --);
                if(PORTCbits.SDA)
                {
                        byte |= 0x01;
                        byte = byte << 1;
                }
                else
                {
                        byte &= 0xFE;
                        byte = byte << 1;
```

```
                }
                PORTCbits.SCL = 0;
                for(i = 5; i > 0; i --);
        }
        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
        if(PORTCbits.SDA)
        {
                byte |= 0x01;
        }
        else
        {
                byte &= 0xFE;
        }
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
        TRISC &= 0xE7;
        return byte;
}
void ACK_master(void)
{
        int i;
        PORTCbits.SDA = 0;
        for(i = 2; i > 0; i --);
        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
}
void No_ACK_master(void)
{
        int i;
        PORTCbits.SDA = 1;
        for(i = 2; i > 0; i --);
        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
}
void delay(void)
{
        int i;
        for(i = 0; i < 9000; i ++)
        {
                ;
```

```
        }
}
void i2c_out_CO2(void)
{
        int i, j, k;
        unsigned char* temp1;
        unsigned char* temp3;
        unsigned char* temp4;
        unsigned char* temp5;
        unsigned char* temp_receive;
        unsigned char temp_receive_byte[4];
        unsigned char calc_result_checksum;
        unsigned char complete_bit;
        unsigned char* wrong_time_out_ACK;
        unsigned char* wrong_time_out_command;
        unsigned char* wrong_checksum;
        unsigned char high_byte;
        unsigned char low_byte;
        unsigned int co2_reading;
        unsigned int time_out_ACK_reception = 22500;
        unsigned int time_out_command_bit = 4500;
        i2c_initial();
        i2c_start();
        TRISC &= 0xE7;
        *temp1 = 0xD0;
        transmission(temp1);
        TRISC |= 0x10;
        delay_for_get_ACK_from_Slave();
        TRISC &= 0xE7;
        *temp3 = 0x22;
        transmission(temp3);
        TRISC |= 0x10;
        delay_for_get_ACK_from_Slave();
        TRISC &= 0xE7;
        *temp3 = 0x00;
        transmission(temp3);
        TRISC |= 0x10;
        delay_for_get_ACK_from_Slave();
        TRISC &= 0xE7;
        *temp4 = 0x08;
        transmission(temp4);
        TRISC |= 0x10;
        delay_for_get_ACK_from_Slave();
        TRISC &= 0xE7;
        *temp5 = 0x2A;
```

```
transmission(temp5);
TRISC |= 0x10;
delay_for_get_ACK_from_Slave();
TRISC &= 0xE7;
i2c_stop();
delay();
do{
        i2c_initial();
        i2c_start();
        TRISC &= 0xE7;
        *temp3 = 0xD1;
        transmission(temp3);
        TRISC &= 0xE7;
        PORTCbits.SCL = 0;
        PORTCbits.SDA = 0;
        for(i = 0; i < 100; i ++);
        TRISC |= 0x10;
        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
if(PORTCbits.SDA == 0)
{
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
        TRISC &= 0xE7;
        PORTCbits.SCL = 0;
        PORTCbits.SDA = 0;
        for(i = 0; i < 100; i ++);
        TRISC |= 0x10;
        for(i = 0; i < 100; i ++);
        for(k = 0; k < 4; k ++)
        {
                temp_receive_byte[k] = receive_byte();
                if(k < 3)
                        ACK_master();
                else
                        No_ACK_master();
        }
        TRISC &= 0xE7;
        i2c_stop();
        complete_bit = temp_receive_byte[0] & 0x01;
        calc_result_checksum = 0;
        for(i = 0; i < 3; i ++)
        {
                calc_result_checksum += temp_receive_byte[i];
        }
```

```
		high_byte = temp_receive_byte[1];
		low_byte = temp_receive_byte[2];
		temp_receive_byte[0] >> 4;
		if(temp_receive_byte[0] != 0x02)
		{
			if(time_out_command_bit > 0)
				time_out_command_bit --;
			else
				break;
		}
}
else
{
		PORTCbits.SCL = 0;
		for(i = 5; i > 0; i --);
		TRISC &= 0xE7;
		i2c_stop();
		if(time_out_ACK_reception > 0)
			time_out_ACK_reception --;
		else
			break;
}
}while(!complete_bit);
for(i = 0; i < 50; i ++);
if(time_out_ACK_reception != 0)
{
		if(time_out_command_bit != 0)
		{
			if(calc_result_checksum != temp_receive_byte[3])
			{
				i2c_initial();
				i2c_start();
				TRISC &= 0xE7;
				*temp3 = 0xDC;
				transmission(temp3);
				TRISC &= 0xE7;
				i2c_stop();
			}
			else
			{
				i2c_initial();
				i2c_start();
				TRISC &= 0xE7;
				*temp3 = 0xCB;
				transmission(temp3);
```

```
                        TRISC &= 0xE7;
                        i2c_stop();
                        for(i = 0; i < 50; i ++);
                        co2_reading = high_byte;
                        co2_reading = co2_reading << 8;
                        co2_reading = co2_reading | low_byte;
                        i2c_initial();
                        i2c_start();
                        TRISC &= 0xE7;
                        if(co2_reading <= 400)
                        {
                                *temp4 = 0xD0;
                                transmission(temp4);
                        }
                        else if(co2_reading >= 600)
                        {
                                *temp4 = 0xE0;
                                transmission(temp4);
                        }
                        else if(400 < co2_reading < 600)
                        {
                                *temp4 = 0xF0;
                                transmission(temp4);
                        }
                        TRISC &= 0xE7;
                        i2c_stop();
                }
        }
        else
        {
                i2c_initial();
                i2c_start();
                TRISC &= 0xE7;
                *temp3 = 0xED;
                transmission(temp3);
                TRISC &= 0xE7;
                i2c_stop();
        }
}
else
{
        i2c_initial();
        i2c_start();
        TRISC &= 0xE7;
        *temp3 = 0xFE;
```

```
                transmission(temp3);
                TRISC &= 0xE7;
                i2c_stop();
        }
        for(i = 0; i < 50; i ++);
}
void i2c_out_temp(void)
{
        int i, j, k;
        unsigned char* temp6;
        unsigned char temp_accept_byte[2];
        unsigned char high_byte;
        unsigned int temperature = 0;
        i2c_initial();
        i2c_start();
        TRISC &= 0xE7;
        *temp6 = 0x91;
        transmission(temp6);
        TRISC |= 0x10;
        PORTCbits.SCL = 1;
        for(i = 5; i > 0; i --);
        if(PORTCbits.SDA == 0)
{
        PORTCbits.SCL = 0;
        for(i = 5; i > 0; i --);
        TRISC |= 0x10;
        for(k = 0; k < 2; k ++)
        {
                temp_accept_byte[k] = receive_byte();
                if(k < 1)
                        ACK_master();
                else
                        No_ACK_master();
        }
        high_byte = temp_accept_byte[0];
        high_byte = high_byte << 1;
        high_byte = high_byte >> 1;
        temperature = high_byte;
        temperature = temperature << 8;
        temperature = temperature | temp_accept_byte[1];
        temperature = temperature >> 3;
}
        i2c_stop();
        for(i = 0; i < 50; i ++);
        i2c_initial();
```

```
        i2c_start();
        TRISC &= 0xE7;
        if(temperature*0.0625 >= 40)
        {
                *temp6 = 0xA0;
                transmission(temp6);
        }
        else if(temperature*0.0625 <= 15)
        {
                *temp6 = 0xB0;
                transmission(temp6);
        }
        else if(15 < temperature*0.0625 < 40)
        {
                *temp6 = 0xC0;
                transmission(temp6);
        }
        TRISC &= 0xE7;
        i2c_stop();
        for(i = 0; i < 50; i ++);
}
```

B.3 i2c_func.h

```
#include <i2c.h>
void initial(void);
void i2c_initial(void);
void i2c_start(void);
void transmission(unsigned char*);
void delay_for_get_ACK_from_Slave(void);
void i2c_stop(void);
unsigned char receive_byte(void);
void ACK_master(void);
void No_ACK_master(void);
void delay(void);
void i2c_out_temp(void);
void i2c_out_CO2(void);
```

B.4 Feedback Message Generation Parts in i2c_func.c

B.4.1 CO2 Sensor Feedback Message Generation Part

```
if(time_out_ACK_reception != 0)
{
	if(time_out_command_bit != 0)
	{
		if(calc_result_checksum != temp_receive_byte[3])
		{
			i2c_initial();
			i2c_start();
			TRISC &= 0xE7;
			*temp3 = 0xDC;
			transmission(temp3);
			TRISC &= 0xE7;
			i2c_stop();
		}
		else
		{
			i2c_initial();
			i2c_start();
			TRISC &= 0xE7;
			*temp3 = 0xCB;
			transmission(temp3);
			TRISC &= 0xE7;
			i2c_stop();
			for(i = 0; i < 50; i ++);
			co2_reading = high_byte;
			co2_reading = co2_reading << 8;
			co2_reading = co2_reading | low_byte;
			i2c_initial();
			i2c_start();
			TRISC &= 0xE7;
			if(co2_reading <= 400)
			{
				*temp4 = 0xD0;
				transmission(temp4);
			}
			else if(co2_reading >= 600)
			{
				*temp4 = 0xE0;
				transmission(temp4);
```

```
                        }
                        else if(400 < co2_reading < 600)
                        {
                                *temp4 = 0xF0;
                                transmission(temp4);
                        }
                        TRISC &= 0xE7;
                        i2c_stop();
                }
        }
        else
        {
                i2c_initial();
                i2c_start();
                TRISC &= 0xE7;
                *temp3 = 0xED;
                transmission(temp3);
                TRISC &= 0xE7;
                i2c_stop();
        }
}
else
{
        i2c_initial();
        i2c_start();
        TRISC &= 0xE7;
        *temp3 = 0xFE;
        transmission(temp3);
        TRISC &= 0xE7;
        i2c_stop();
}
for(i = 0; i < 50; i ++);
```

B.4.2 Temperature Sensor Feedback Message Generation Part

```
i2c_initial();
i2c_start();
TRISC &= 0xE7;
if(temperature*0.0625 >= 40)
{
        *temp6 = 0xA0;
        transmission(temp6);
}
else if(temperature*0.0625 <= 15)
```

```
{
        *temp6 = 0xB0;
        transmission(temp6);
}
else if(15 < temperature*0.0625 < 40)
{
        *temp6 = 0xC0;
        transmission(temp6);
}
TRISC &= 0xE7;
i2c_stop();
for(i = 0; i < 50; i ++);
```