

**DEPARTMENT OF ELECTRICAL AND  
ELECTRONIC ENGINEERING**

**Serial Interfaces: SPI and I2C**

AUTHOR	Mr. Chi Liu
SUPERVISOR	Dr. John Crowe
MODERATOR	Dr. Chung See
DATE	10 <sup>th</sup> May 2016

Third (Four if applicable) year project report is submitted in part fulfilment of the requirements of the degree of Bachelor (Master if applicable) of Engineering.

**Abstract:**

In this thesis, the software implementation of using Serial communication interfaces to connect selected sensors with selected microprocessors and the hardware development of building a simple microprocessors on stripboard are presented. Serial communication is the process of transmitting data one bit a time, sequentially between a microprocessor and peripherals over a communication channel. There are different types of Serial communication interfaces. In this thesis, Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C) and Universal Asynchronous Receiver/Transmitter (UART), Serial communication interfaces are discussed in detail. A laboratory session is provided for Year 1 undergraduate students to introduce and explain them how to use these common serial interfaces.

The following information includes the details of how the sensors connect with the microprocessors through the SPI and I2C interfaces, how the microprocessors connect with the computer through the UART interface, and how to build a microprocessor (Arduino) on the stripboard.

## Contents:

Abstract: .....	1
Section 1: Introduction: .....	4
1.1 Aim: .....	4
1.2 Objectives: .....	4
Section 2: Background .....	5
2.1 Serial Peripheral Interface (SPI): .....	5
2.2 Inter-integrated Circuit (I2C): .....	6
2.3 Universal Asynchronous Receiver Transmitter (UART): .....	8
2.4 Analog to Digital Converter: .....	8
2.5 Digital to Analog Converter: .....	9
2.6 Accelerometer: .....	10
2.7 Environmental sensor: .....	11
2.8 Microcontrollers: .....	11
Section 3: Software design and implementation .....	13
3.1 UART software design: .....	13
3.1.1 PIC18F46k20 UART software design: .....	13
3.1.2 Arduino Leonardo UART software design: .....	16
3.2 SPI software design: .....	17
3.2.1 PIC18F46k20 SPI Interface initialization: .....	17
3.2.2 Arduino Leonardo SPI Interface initialization: .....	18
3.3 PmodACL2 SPI interface software design: .....	18
3.4 BME280 SPI interface software design: .....	19
3.5 ADC SPI software design: .....	21
3.6 DAC SPI software design: .....	22
3.7 I2C software design: .....	24
3.7.1 PIC18F46k20 I2C interface initialization: .....	24
3.7.2 Arduino Leonardo I2C interface initialization: .....	25
3.8 BME280 I2C software design: .....	26
3.9 MMA8452 accelerometer I2C software design: .....	27
Section 4: Results and Software Evaluation .....	28

4.1 UART: .....	28
4.1.1 PIC18F46k20 UART Test and Results: .....	28
4.1.2 Arduino Leonardo UART Test and Results: .....	29
4.2 SPI: .....	30
4.2.1 PIC SPI interface: .....	30
4.2.2 Arduino SPI interface: .....	30
4.2.3 DAC: .....	31
4.2.4 ADC: .....	31
4.2.5 PmodACL2: .....	33
4.2.6 BME280: .....	35
4.3 I2C: .....	39
4.3.1 PIC I2C interface: .....	39
4.3.2 Arduino I2C interface: .....	39
4.3.3 MMA8452: .....	40
4.3.5 BME280 I2C interfaces: .....	42
Section 5: Hardware design and evaluation.....	43
5.1 Hardware design: .....	43
5.2 Burn Bootloader: .....	44
5.3 Evaluations: .....	44
Section 6: Discussion .....	45
6.1 Compare SPI and I2C interfaces: .....	45
6.2 Compare PIC vs Arduino: .....	46
Section 7: Conclusion.....	47
Section 8: References .....	48
Appendix A: BME 280 SPI and I2C lab sessions .....	51
Appendix B: Time plan .....	59

## Section 1: Introduction:

In the present days, microprocessors and sensors are widely used in our daily life. This project covers the software implementation of using microprocessors to communicate with sensors such as accelerometer by SPI and I2C protocols and the hardware development of build a simple Arduino on the stripboard. The first section of this thesis presents the aim and objectives of the whole project. The second section of this thesis is about the background information of the principles of the SPI, I2C, UART interfaces and the microprocessors and sensors used in this project. Section 3 to Section 5 comprises the software design and implementation of the Serial communications interfaces, hardware design and implementation of building the microprocessor on the stripboard, results, software and hardware design evaluations. In Section 6, the difference and comparison of SPI and I2C are stressed and discussed. In addition, the comparison of different microprocessors is also highly emphasized and underlined. In section 7, the conclusion is discussed to illustrate the reflection on original plan and what has been learned during the project. The appendix A provides the detail of the lab session for Year 1 students.

### 1.1 Aim:

The aim of this project is to select several sensors with SPI or I2C interface and produce suitable material for a Year 1 undergraduate laboratory session that introduces and explains how to use these common serial interfaces on the microprocessor. These sensors are accelerometer and environmental (temperature, pressure and humidity) sensor. In addition to these sensors, Analog to Digital Converter (ADC), Digital to Analog Converter, real time clock and memory chips are also connected with the microprocessors through the SPI or I2C interfaces. In this project, PIC18 and Arduino are chosen to be used to operate the Serial communication interfaces.

### 1.2 Objectives:

From the aims, the objectives of the project are the following:

- Connect the accelerometer with Arduino and PIC18 through both SPI and I2C interfaces
- Connect the environmental sensor with Arduino and PIC18 through both SPI and I2C interfaces
- Connect the DAC and ADC sensor with Arduino and PIC18 through SPI interface
- Connect the Arduino and PIC18 to the computer via UART interface and display the recording data from the sensors on the computer
- Build an Arduino on the stripboard
- Design and document a lab session of the SPI interface with a specific sensor for first year undergraduate students
- Design and document a lab session of the I2C interface with a specific sensor for first year undergraduate students

## Section 2: Background

### 2.1 Serial Peripheral Interface (SPI):

Serial Peripheral Interface (SPI) is one of the most popular low end communication protocols. The SPI was developed by Motorola [1] and introduced with the first microcontroller that derived from the same architecture as the popular Motorola 68000 microprocessor, which was the best on the market at that time, announced in 1979 [2]. The Motorola M68HC11 microcontroller [3], still on the market, is also developed from this architecture.

The SPI is a full-duplex, synchronous, serial communication between the Microprogrammed Control Unit (MCU) and peripheral devices [4]. SPI is a master-slave architecture. SPI has only one master, but it could have more than one slaves. The master provides the clock signal and sends commands to or reads data from the slaves. The slaves receive the signals and send data back to the master. This procedure is implemented through four wires and these four wires are corresponding to four port pins. They are Slave select (SS) or Chip select (CS), Serial clock (SCK), Master out/slave in (MOSI) and Master in/slave out (MISO). The basic connection of SPI interface shows in Figure 1.

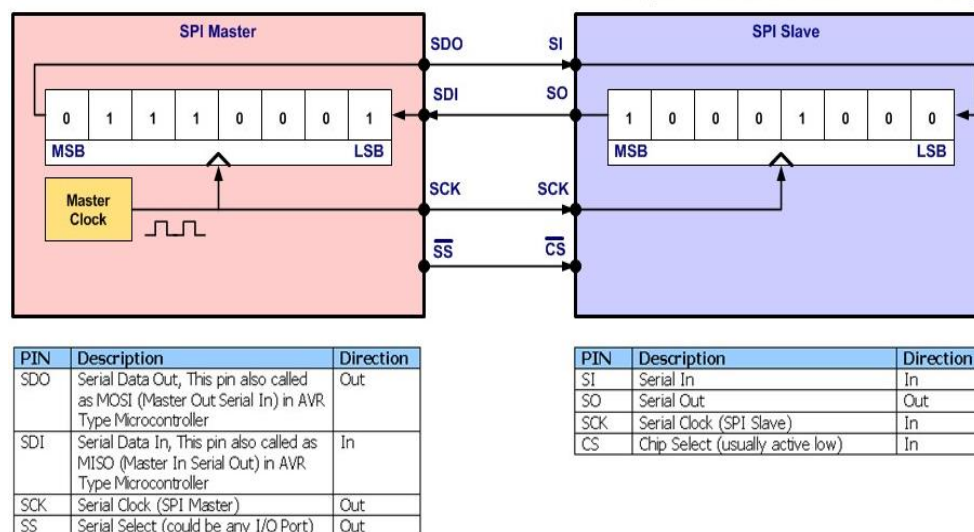


Figure 1: The SPI master and SPI slave basic connection [5].

To start the communication, the master configures the clock with a suitable frequency and then the master select the slave by setting the SS pin to low. During each clock cycle, the master sends one bit to the slave through SDO (Serial Data Out) and reads one bit from the slave through SDI (Serial Data In), at the same time, the slave sends one bit to the master through SO (Serial Out) and reads one bit from the master through SI (Serial In). Two shift registers are used during the transmission. One is in the master, the other one is in the slave. Usually, the register size is 8 bit. When the bits in the registers have been totally exchanged, the transmission is completed.

The master also sets the clock polarity and clock phase during the transmission by writing related configuration code to the master. CPOL is the clock polarity control bit and CPHA is the clock phase control bit. Figure 2 shows the timing diagram of the SPI clock and illustrates how these 2 bits configure the clock.

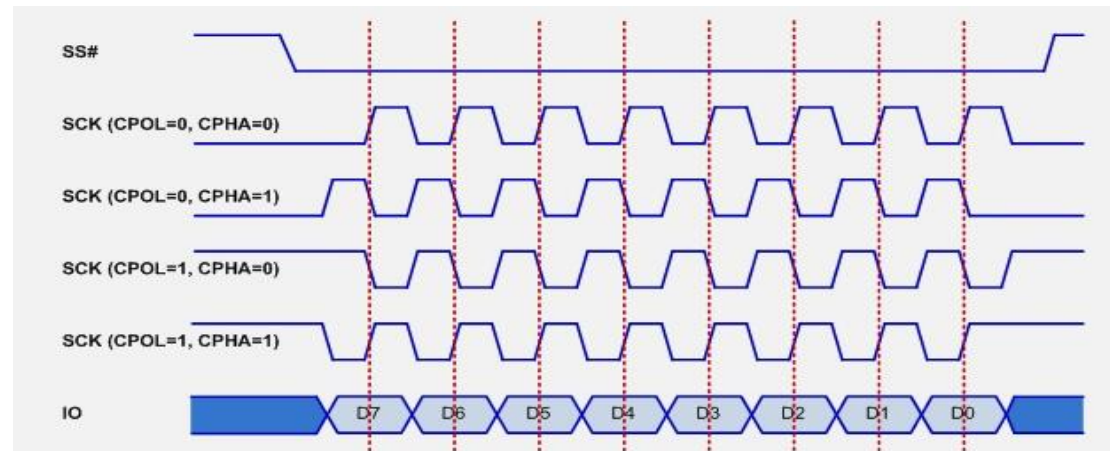


Figure 2: SPI clock diagram [6]

When CPOL=0 and CPHA=0, data is sampled at the leading rising edge of the clock.

When CPOL=0 and CPHA=1, data is sampled at the lagging falling edge of the clock.

When CPOL=1 and CPHA=0, data is sampled at the leading falling edge of the clock.

When CPOL=1 and CPHA=1, data is sampled at the lagging rising edge of the clock.

For SPI bus speed, the SPI bus speed is flexible. Users can set by their demand through configuring the microcontroller (master) internal clock frequency.

## 2.2 Inter-integrated Circuit (I2C):

Inter-integrated Circuit shorts for I2C pronounced I-squared-C, which is also a type of low-end communication protocol. The I2C bus was first developed by Philips Semiconductor in 1982 in order to provide an easy way to connect a CPU to peripheral chips in a television set. Peripheral devices require lots of wiring on the printed circuit board (PCB) and extra “glue logic” to decode the devices’ address in the embedded systems. In this case, to solve these problems, the I2C interface was invented by Philips labs in Eindhoven. It only requires 2 wires to connect all the peripheral devices with the host microprocessors. Now, more than 50 companies in the world implements this interface in over 1000 different manufactured integrated circuits [7].

I2C is a multi-master and multi-slave serial bus. It has 2 lines. One is the Serial data line (SDA), the other one is the Serial clock line (SCL). They are bidirectional open drain lines with pull-up resistors. Usually, I2C bus speed are 100kbit/s in the standard mode and there are also low-speed mode (10 Kbit/s), fast mode (400 Kbit/s) and high speed mode (3.4Mbit/s) [7]. Each peripheral devices with I2C interface has a unique slave address. The most common address is 7-bit or 10-bit [8]. 7 bit addressing is widely used in the standard mode. Figure 3 shows the detail of it.



Figure 3: I2C 7-bit Address [8]

The first 7 bits consists of the slave address, the LSB is the bit indicating the read or write command. 0 means write and 1 means read. 10 bit addressing is used to extend the standard mode.

A typical connection of I2C interface is shown in the figure 4.

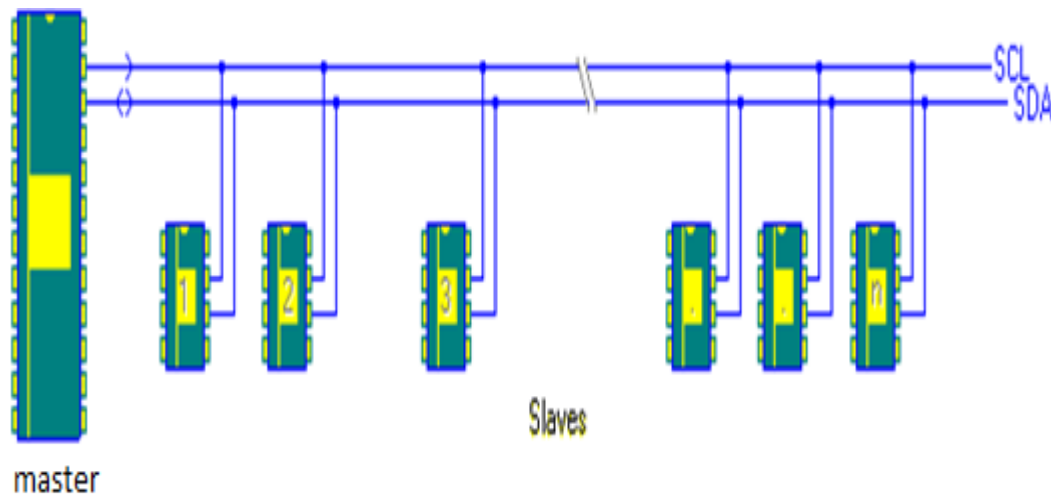


Figure 4: Connection of I2C interface [9].

To start the communication, the master first issues a start condition. This start condition enables all the slaves on the bus to listen for incoming command. Then the master sends a byte comprising of target slave address and a write/read bit. All the slaves compare the sent address with their own address. If the address matches, the slave will response with an acknowledgment signal. Data transmissions procedure operates between the master and slaves. When all the transmission completes, a stop condition is issued by the master.



### 2.3 Universal Asynchronous Receiver Transmitter (UART):

Universal Asynchronous Receiver Transmitter shorts for UART, which is a hardware device to change data between the parallel to serial forms. UART are now common used with popular communications standards such as RS-232, RS-456, RS-485 [10]. In some microprocessors, such as microchip's PIC18, a mode called Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is developed to facilitate communication through a computer's serial port using the RS-232C protocol. The difference between the USART and UART is that USART supports synchronous mode [11]. In this project, the UART are used to enable the communications between microcontrollers and the computers.

UART is full-duplex data communication and only needs two transmission lines TXD (transmit data) and RXD (receive data) [12]. Figure 5 shows the data frame of the UART.

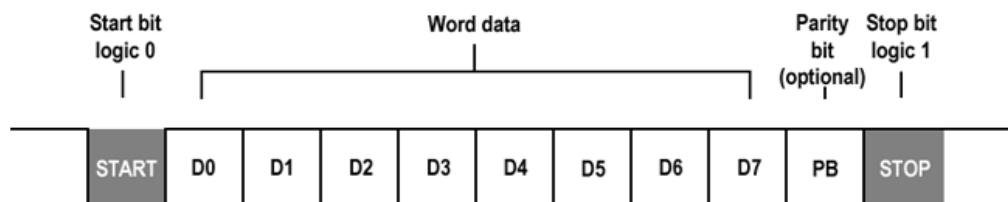


Figure 5: Data frame of the UART [13].

To start the communication, a start bit (logic 0) is sent. This make the receiver listen to the transmitter and synchronize its own clock to that bit. Then a byte carried with message is sent to the receiver by the transmitter. The LSB is sent first. After that, a parity bit could be sent to operate the simple error check. When the transmission is finished, a stop bit (logic 1) is sent by the transmitter.

### 2.4 Analog to Digital Converter:

An Analog to Digital Converter with SPI interface is used in this project. The ADC is built on the printed circuit board (PCB) and was designed by a PhD student in the department. This ADC contains an ADS8326 chip [14]. It is a 16-bit ADC for the supply voltage range from 2.7V-5.5V. The reference voltage is 4.096V. Figure 6 shows the PCB layout of the ADC and figure 7 shows the design schematic of ADC



Figure 6: PCB layout of the ADC.

## Analogue to Digital Converter

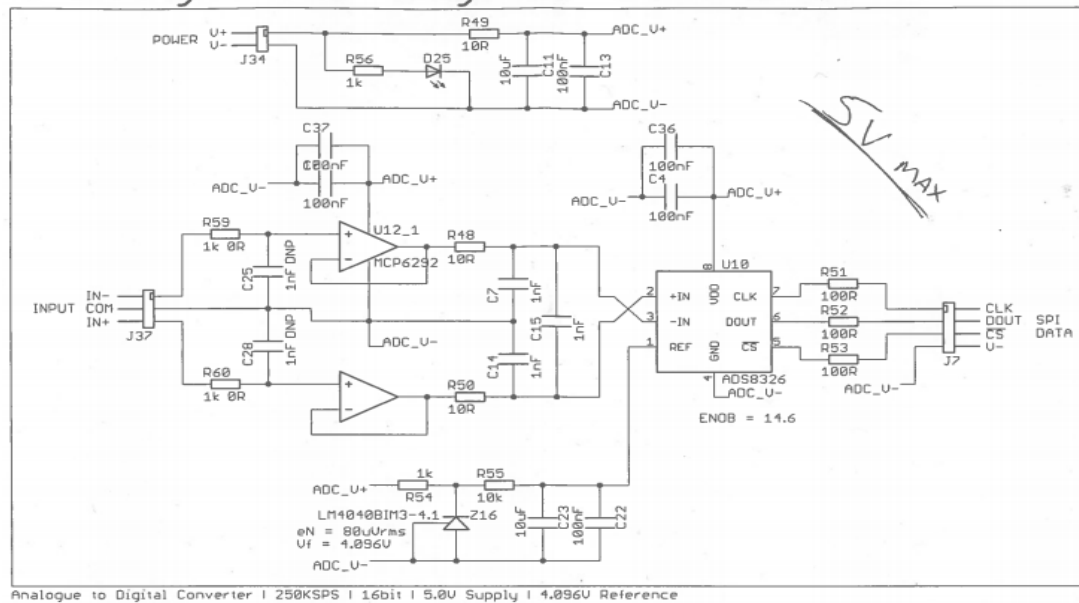


Figure 7: The design schematic of the ADC.

### 2.5 Digital to Analog Converter:

A Digital to Analog Converter with SPI interface is also used in this project. The DAC is also developed by the same person. This DAC contains a MCP4922 chip [15]. It is a 12-bit DAC for the supply voltage range from 2.7V-5.5V. The reference voltage is also 4.096V. Figure 8 shows the PCB layout of the DAC and Figure 9 shows the design schematic of the DAC.

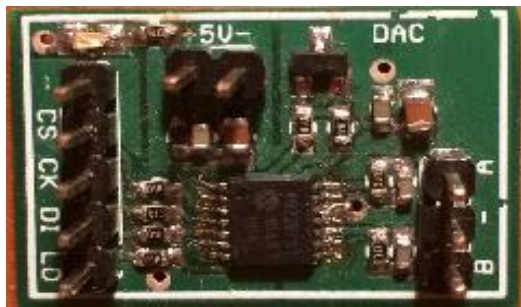


Figure 8: The PCB layout of the DAC.



## 2.7 Environmental sensor:

In this project, a breakout board designed by SPARKFUN with BME280 sensor [20] manufactured by BOSCH is used. The BME280 sensor can measure barometric pressure, humidity, and temperature. It can communicate with a microcontroller by both I2C and SPI interface. The temperature range is  $-40\sim 85^{\circ}\text{C}$ . The humidity range is 0-100% and the pressure range is 30,000Pa to 110,000Pa. Figure 12 shows the layout of the board.

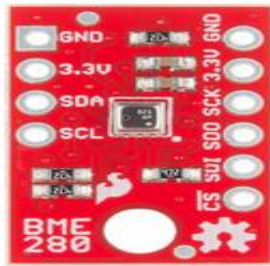


Figure 12: The layout of the Environmental sensor [21].

## 2.8 Microcontrollers:

In this project, two development boards are used as the master to develop the software of the serial communication interfaces. One is the PIC18F4Xk20 Starter Kit manufactured by Microchip and the other one is Arduino LEONARDO produced by Arduino.

PIC18F4xK20 Starter Kit is based on PIC18F46k20 [22], a 44 pin 8-bit microcontroller. PIC18F46k20 has 64 KB flash program memory, 16 MIPS CPU, 3936 RAM bytes, 1024 data EEPROM bytes. For digital communication peripherals, it includes UART, I2C and SPI modules. In addition, it also has Capture/Compare/PWM Peripherals, 2 comparators, 4 timers and 10-bit ADC with 13 channels. PICKit2, in circuit debugger, is used to program the microcontroller. Figure 13 shows the PICKit2 and figure 14 shows the details of PIC18F4Xk20 Starter Kit Board.



**Legend:**

- |                |                        |                          |
|----------------|------------------------|--------------------------|
| 1. Status LEDs | 3. Lanyard Connection  | 5. Pin 1 Marker          |
| 2. Push Button | 4. USB Port Connection | 6. Programming Connector |

Figure 13: PICKit2 [23]



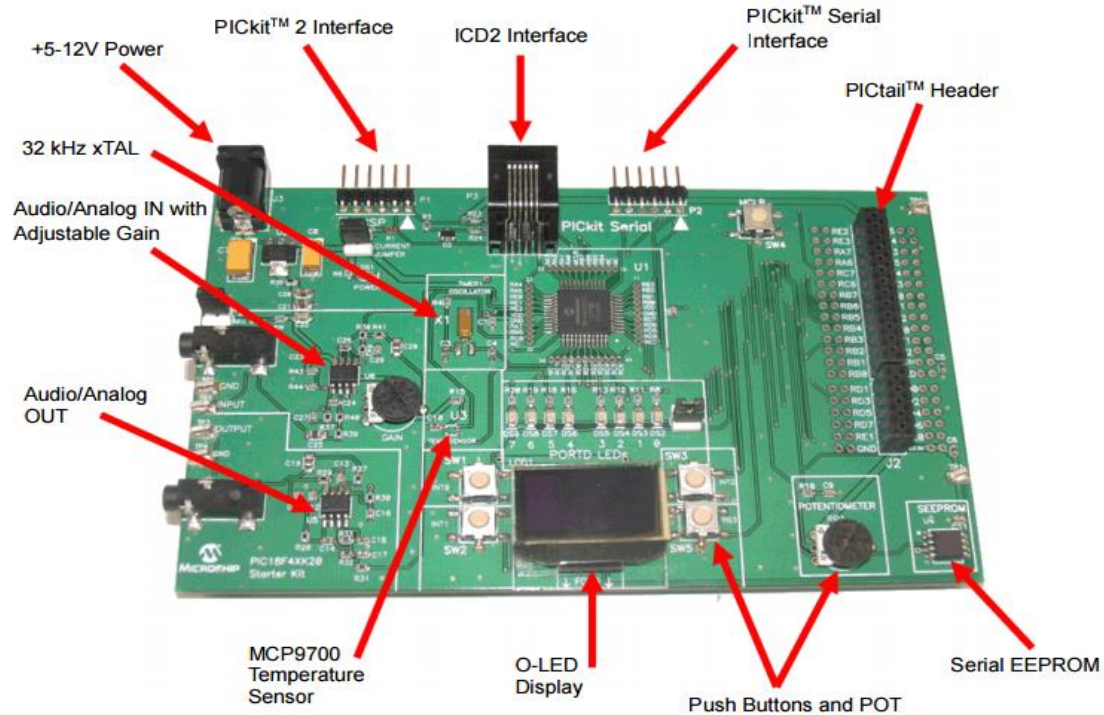


Figure 14: PIC18F4Xk20 Starter Kit Board [24].

The Arduino Leonardo is a board with the ATmega32u4 microcontroller manufactured by Atmel [25]. ATmega32u4 has 32KB flash memory, 2.5 KB RAM, 1 KB EEPROM. Arduino Leonardo has 20 digital I/O pins, 7 of which can be used as PWM and 12 of which can be used as analog inputs. It also has 16 MHz crystal oscillator, In-Circuit Serial Programming (ICSP) header, a power jack, and a micro-USB connection. Users can directly program the microcontroller through the micro-USB. Figure 15 shows the Arduino Leonardo board.

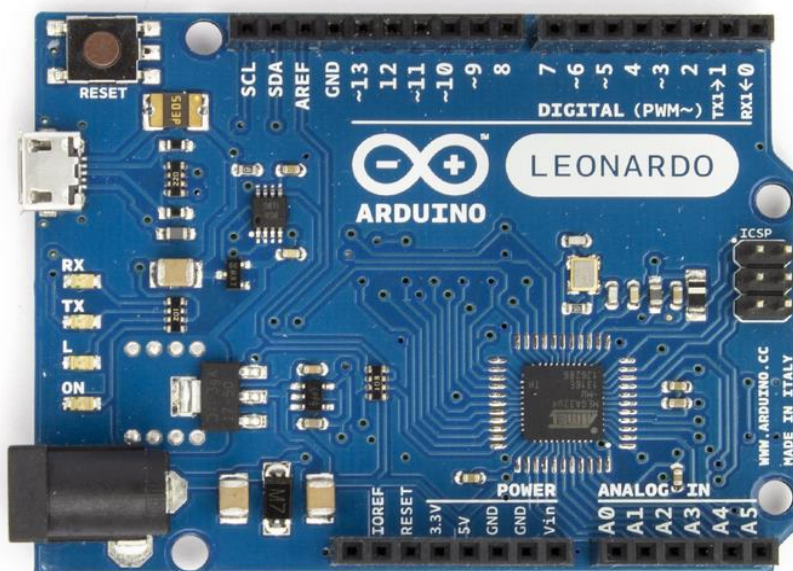


Figure 15: Arduino Leonardo board. [26]

## Section 3: Software design and implementation

### 3.1 UART software design:

In this project, the UART is used to establish the communications between the microcontrollers and the computer. The microcontroller is configured as transmitter and the computer is configured as receiver.

#### 3.1.1 PIC18F46k20 UART software design:

##### Header file

---

```
#include <usart.h>          //usart library.
```

---

<usart.h> is the UART library provided by XC8 compiler for PIC18f46k20.

##### UART initialization:

---

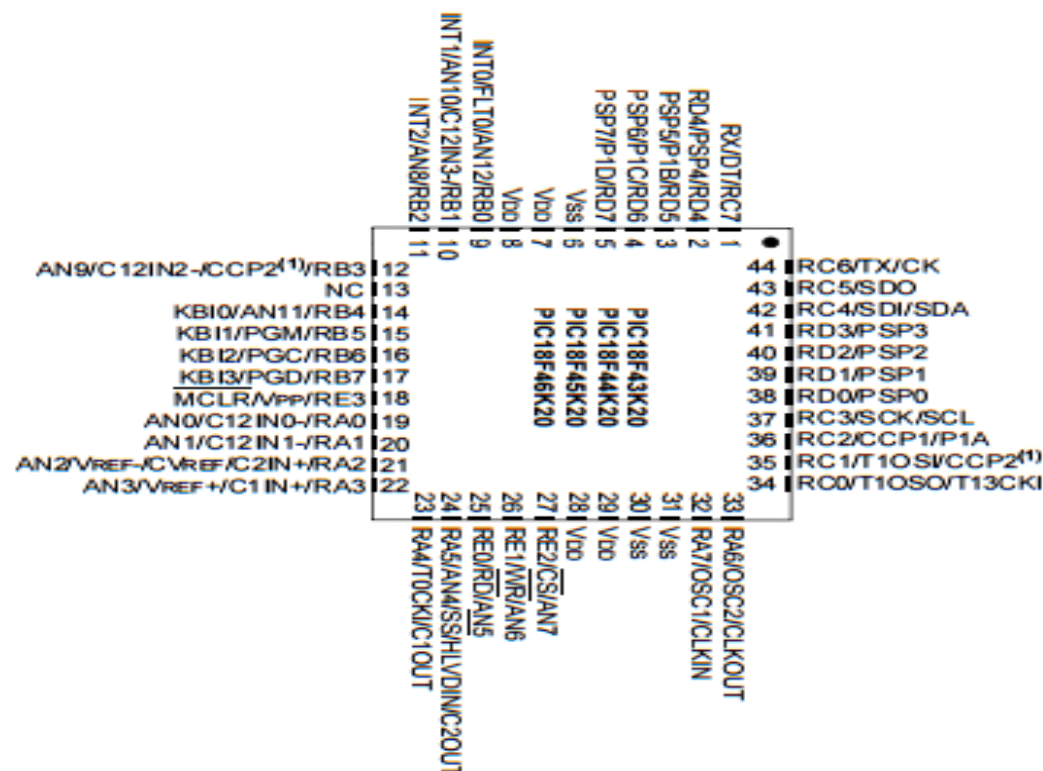


Figure 16: The pin map of the PIC18F46k20 [22]

From the figure 16, it can be found that TX and RX pin are RC6 and RC7 (PORTC). To start the communication, function `OpenUSART()` is called from the <usart.h> library to configure and open the UART mode.

---

```

OpenUSART( USART_TX_INT_OFF & // TX interrupt off
USART_RX_INT_OFF &          // RX interrupt off
USART_ASYNCH_MODE &         // Asynchronous mode
USART_EIGHT_BIT &           // 8 bit format
USART_CONT_RX &              // continue reception
USART_BRGH_HIGH,             //high baud rate mode
12                           ); //sprbg is 12. Baud rate generator

```

---

This above code means the UART mode is configured as TX and RX interrupt off, Asynchronous, 8-bit transmission format, continues reception and High baud rate mode [11]. In this project, the internal clock is configured as 8MHz by setting OSCCON register. In this case the baud rate is  $\frac{F_{osc}}{16 * (sprbg + 1)} = \frac{8MHz}{16*(12+1)} = 38461.53bps$ . In addition, several functions has been designed by myself to allow users to write a byte, a string and an integer from the microcontroller to the computer. The following table shows the details of them.

Table 1: The UART functions

Function	Description
UARTWriteByte	Write a byte to the computer
UARTWriteString	Write a string to the computer
UARTWriteInteger	Write an Integer (0~8 digit) to the computer

### Connect to the Computer:

Because the PIC output signal is 3.3V and the voltage captured by computer is 5V, a smart cable called FTDI TTL-232R-3V3 CABLE is used to simplify the connection between the computer and the microcontroller, and to change the 3.3V output voltage level to 5V. It is a USB to UART cable with 3.3V TTL level UART signals. Figure 17 shows Connector Pin Out and Mechanical details of the cable.

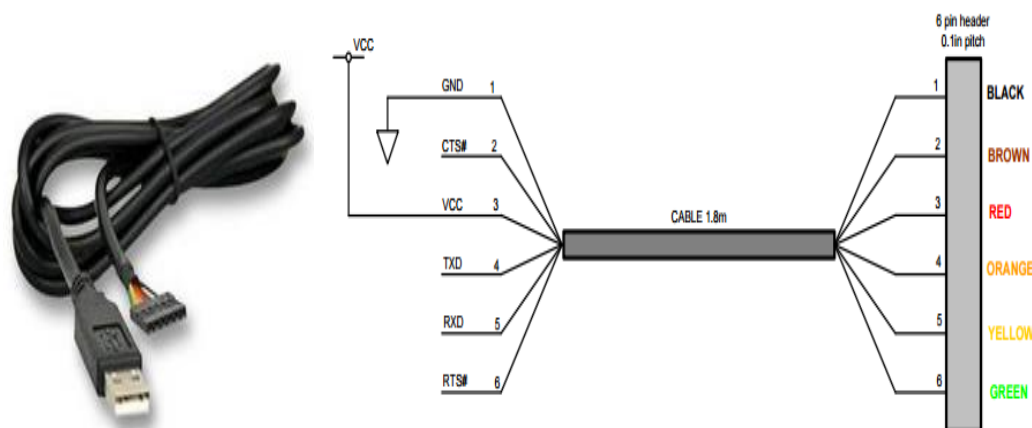


Figure 17: The Connector Pin Out and Mechanical details of the cable [28].

To establish the connection, connect the Pin 5(RXD) of the cable to the RC6 (TX) of the PIC18F46k20 microcontroller and plug the USB on the computer. After a driver is installed, a virtual COM port will be recognized by the computer.

In order to monitor the communication, a software called “Realterm” to use as terminal to receive and display the sent information from PIC18F46k20 is installed on the computer. Figure 18 shows the interface of the Realterm.

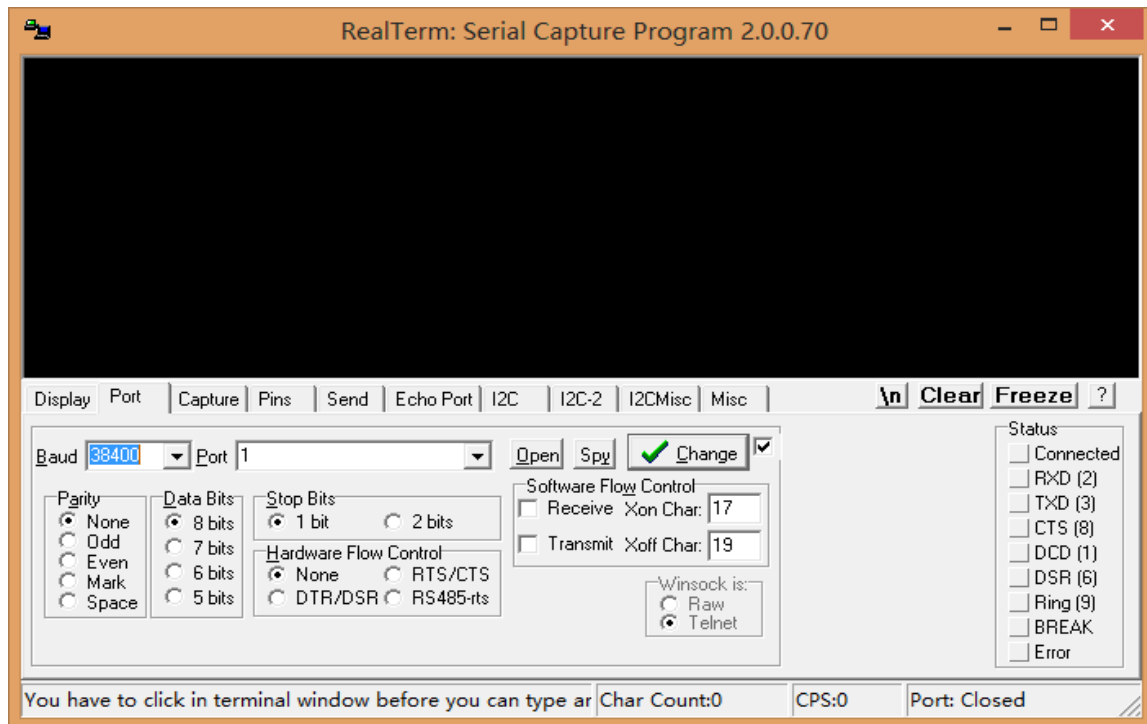


Figure 18: The interface of the Realterm

ASCII [29] is used to translate the signals and display the results from the microcontroller. In this case, the function UARTWriteInteger mentioned above needs to be well organized to ensure the computer can read the signals. The following flowchart shows the design of the UARTWriteInteger function.

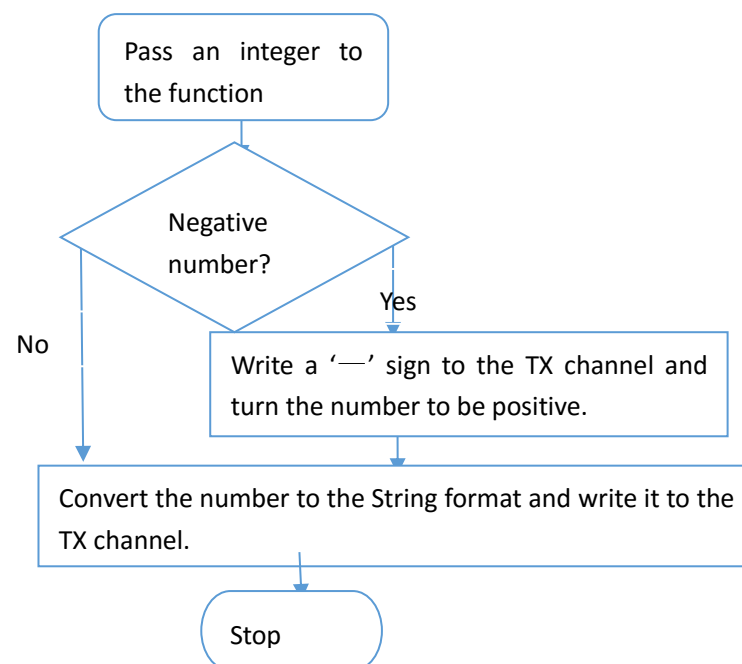


Figure 19: The flowchart of the UARTWriteInteger function.



### 3.1.2 Arduino Leonardo UART software design:

#### Header file:

---

```
#include <SoftwareSerial.h>
```

---

<SoftwareSerial.h> is provided by Arduino to allow the Atmega chip to communicate with other hardware. The following table illustrates the functions in this library used in this project.

Table 2: Functions of the <SoftwareSerial.h> library

Function:	Description:
Serial.begin	Sets the speed (baud rate) for the serial communication.
Serial.write	Writes binary data to the serial port
Serial.print	Prints data to the serial port as human-readable ASCII text
Serial.println	Prints data to the serial port as human-readable ASCII text and followed by a new line character.

The supported baud rates of the Arduino are 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600, and 115200 bps. In order to establish the communication, the Arduino Leonardo board is connected with the computer through the mirco-USB. In addition, a Serial monitor is provided by Arduino IDE to monitor the communication. Figure 20 shows the interface of it.

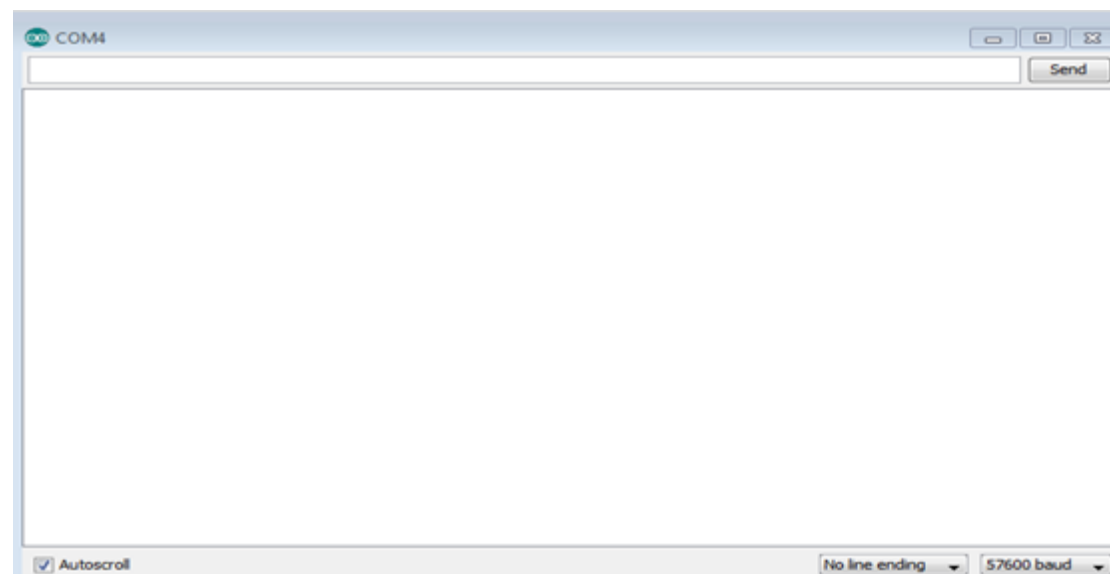


Figure 20: Serial monitor of the Arduino

ASCII is also used to translate the messages and display the results on the computer. Serial.print function could support different types of format. It could turn the float and integer to ASCII digit automatically.

## 3.2 SPI software design:

### 3.2.1 PIC18F46k20 SPI Interface initialization:

#### Header file:

##### #include:

---

```
#include <spi.h>           //spi library.
```

---

<spi.h> is a SPI peripheral library provided by XC8 complier. The following table introduces the functions of the SPI library for PIC18F46k20 used in this project.

Table 3: PIC18F46k20 SPI library functions

Function:	Description:
CloseSPI	Disable SPI communications.
DataRdySPI	Determine if a new value is available from the SPI buffer.
getcSPI	Read a byte from the SPI bus.
getsSPI	Read a string from the SPI bus.
OpenSPI	Initialize the SSP module used for SPI communications.
putcSPI	Write a byte to the SPI bus.
putsSPI	Write a string to the SPI bus.
ReadSPI	Read a byte from the SPI bus.
WriteSPI	Write a byte to the SPI bus.

#### SPI configuration:

Using PIC18F46K20 as master, according to the figure 16, SPI pins are connected on PORTC. RC3 is SCK, RC4 is MISO. RC5 is MOSI. (PORTB) RB0 is SS (This pin is not fixed. Other Pins could be set by users.). RB0, RC5 and RC3 should be set as digital output. RC4 should be set as digital input.

To set the I/O pins, TRIS Register should be configured. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., disable the output driver). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., enable the output driver and put the contents of the output latch on the selected pin). TRISB is similar. In addition, in order to set SPI clock frequency, internal clock should also be set.

To open the SPI interface, function *OpenSPI(SPI\_FOSC\_4, MODE\_00, SMPMID)* is used. This means the SPI is configured to Master mode with clock frequency  $\frac{F_{osc}}{4} = \frac{8\text{MHz}}{4} = 2\text{MHz}$  and the SPI clock is configured to Mode 0 (CPOL=0, CPHA=0). In addition, the input data is sampled at the mid of data out.

### 3.2.2 Arduino Leonardo SPI Interface initialization:

#### Header file:

#### #include:

---

```
#include <SPI.h>           //spi library.
```

---

<SPI.h> is a provided SPI peripheral library for Arduino. The following table introduces the functions of the SPI library for PIC18F46k20 used in this project.

Table 4: SPI library for Arduino

Function	Description
SPI.begin	Initializes the SPI bus.
SPI.beginTransaction	Initializes the SPI bus using the defined SPISettings.
SPI.Settings	configure the SPI port for your SPI device
SPI.end	Disables the SPI bus (leaving pin modes unchanged).
SPI.transfer	send a byte to the SPI bus and receive a byte from the SPI bus

#### SPI configuration:

Using Arduino Leonardo as master, according to datasheet, SPI pins (MOSI, MISO, SCK) are connected on the ICSP Header. Digital Pin10 has been used as SS. Figure 21 shows the image of ICSP Header. To configure the Pin10, “pinMode(10,OUTPUT)” is called to set Digital Pin10 as output. Call “SPI.beginTransaction” and “SPI.Settings” to set the SPI clock to 2MHz.



Figure 21: Arduino Leonardo ICSP [27]

### 3.3 PmodACL2 SPI interface software design:

Based on the command structure figures shown in the accelerometer datasheet, two functions are designed by myself to send the SPI commands from the microcontrollers to the PmodACL2. One of them operates the Write command, which could allow users to send data to the corresponding register of the accelerometer. The other one operates the Read command, which could allow users to read data from the corresponding register of the accelerometer. The following pseudo code shows the detail of them.

---

#### Write data function:

*Put CS down;*

*Send the write command byte (0x0A) to the accelerometer;*

*Send the address byte to the accelerometer;*

*Send the data byte to the accelerometer;*

*Put CS up;*

---

**Read data function:**

*Put CS down;*

*Send the read command byte (0x0B) to the accelerometer;*

*Send the address byte to the accelerometer;*

*Read the data byte from the accelerometer and store it in an unsigned char x;*

*Put CS up;*

*Return x*

---

The following flowchart shows the whole process of getting the 3-axis acceleration from the PmodACL2 through the SPI interface and displaying the result on the computer through the UART interface.

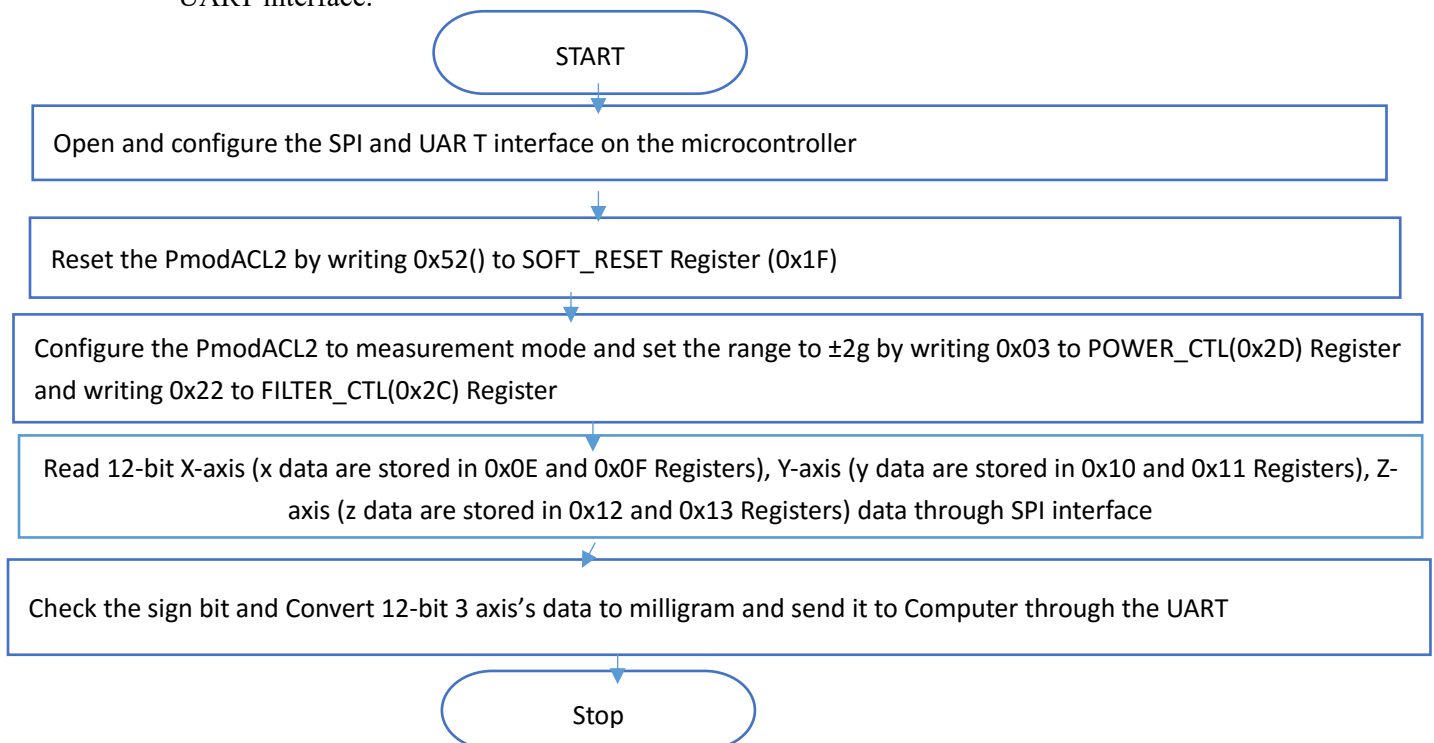


Figure 22: The flowchart of getting acceleration data from the PmodACL2.

### 3.4 BME280 SPI interface software design:

Based on the command structure diagram in the BME280 datasheet, five functions are designed by myself to help users to obtain target data from the BME280. Two of them are SPI Write command and SPI Read command, which could allow users to send/read data to/from the corresponding register of the BME280 sensor. The other three functions are to implement the compensation formulas of temperature, pressure and humidity to obtain accurate result because of each sensing element behaves differently. The following pseudo code illustrates the details of these five functions.

---

**BME280 SPI write command:**

*Change the address byte bit 7 to 0 (because the bit 7 is R/W bit and default value is 1. 0 means write, 1 means read)*

*Put CS down*

*Write address byte (bit 7 is zero) to the BME280*

*Write data byte to the BME280*

*Put CS up*

---

**BME280 SPI Read command:**

*Put CS down*

*Write address byte (bit 7 is one) to the BME280*

*Read data byte from the BME280 and store it in an unsigned char x;*

*Put CS up*

*Return x*

---

**BME280 Temperature compensation function:**

*Read coefficient parameters dig\_T1, dig\_T2, dig\_T3 from the sensor through SPI interface*

*Use the coefficient parameters to operate the compensation formula and store the result in T*

*Return T;*

---

**BME280 Pressure compensation function:**

*Read coefficient parameters dig\_P1~dig\_P9 from the sensor through SPI interface*

*Use the coefficient parameters to operate the compensation formula and store the result in P*

*Return P;*

---

**BME280 Humidity compensation function:**

*Read coefficient parameters dig\_H1~dig\_H6 from the sensor through SPI interface*

*Use the coefficient parameters to operate the compensation formula and store the result*

*Return the result;*

---

The following flowchart shows the whole process of how to obtaining the temperature, pressure and humidity data from the BME280 through the SPI interface.

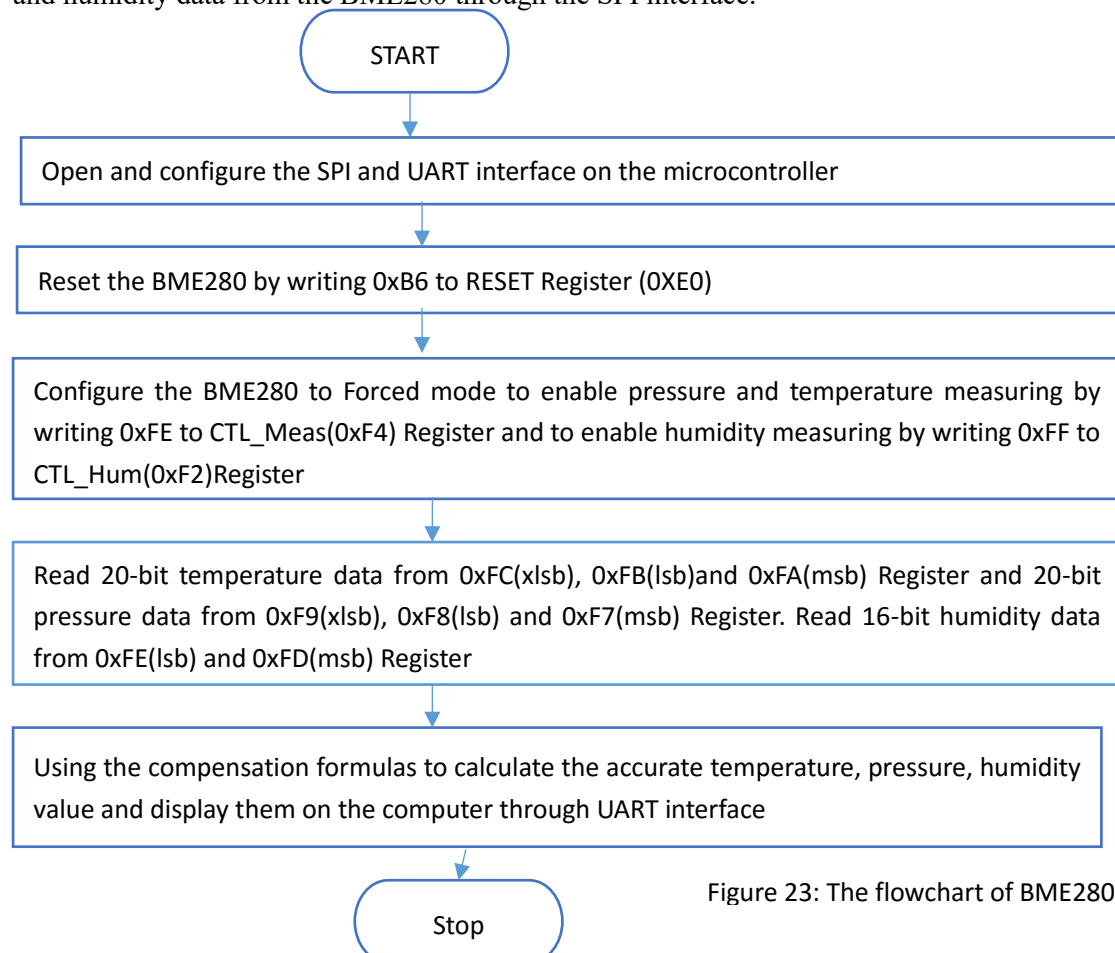


Figure 23: The flowchart of BME280

### 3.5 ADC SPI software design:

According to the timing graph from the datasheet shown in the figure 24, only one function is designed to implement the SPI Read command to obtain data from the ADS8326. The following shows the pseudo code of the function.

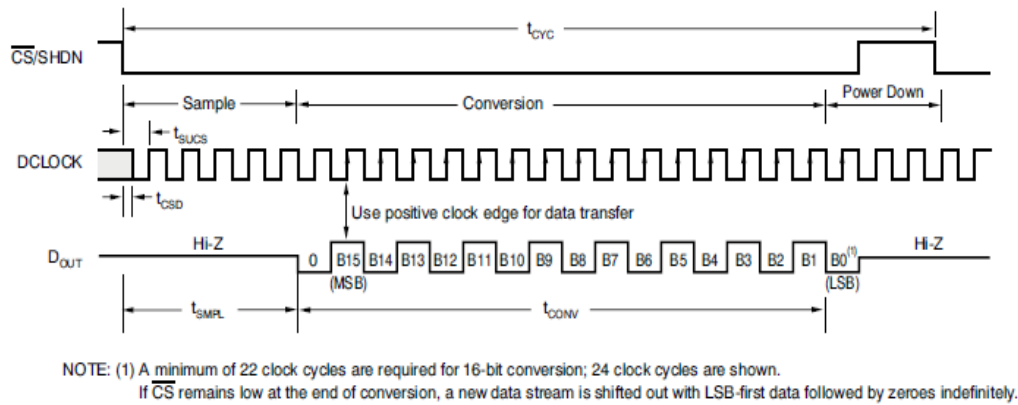


Figure 24: The timing diagram of the ADS8326 [14].

Read ADC function:

*Put CS down;*

*Read the first byte; //8 clock cycles*

*Read the second byte; //8 clock cycles*

*Read the third byte; //8 clock cycles*

*Using bit operation to reduce 24-bit to obtain 16-bit ADC data and store it in a unsigned integer x;*

*Return x;*

*Put CS high*

The reference voltage of this 16-bit ADC is 4.096V. In this case, the resolution of this ADC is

$$\frac{4096mV}{65536} = 0.0625mV$$

The following flowchart illustrates the procedure of how to obtain the ADC result through the SPI interface and to display it on the Computer.

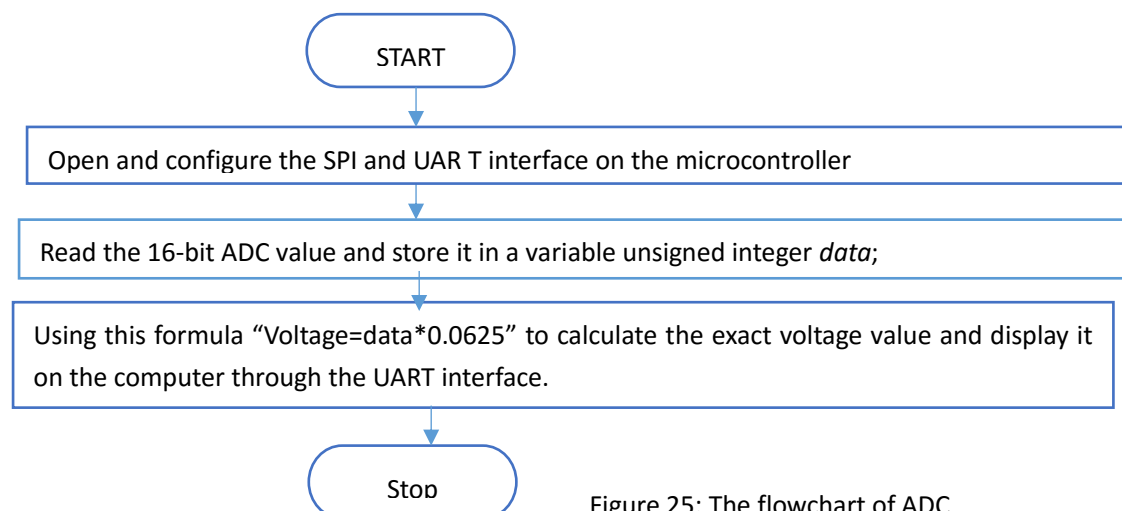


Figure 25: The flowchart of ADC

### 3.6 DAC SPI software design

According to the timing graph from the datasheet shown in the figure 26, only one function is written to implement the SPI Write command to send data to the DAC. The following shows the pseudo code of the function.

---

DAC Write function:

*Put CS down;*

*Send the MSB byte;*

*Send the LSB byte;*

*Put CS up*

---

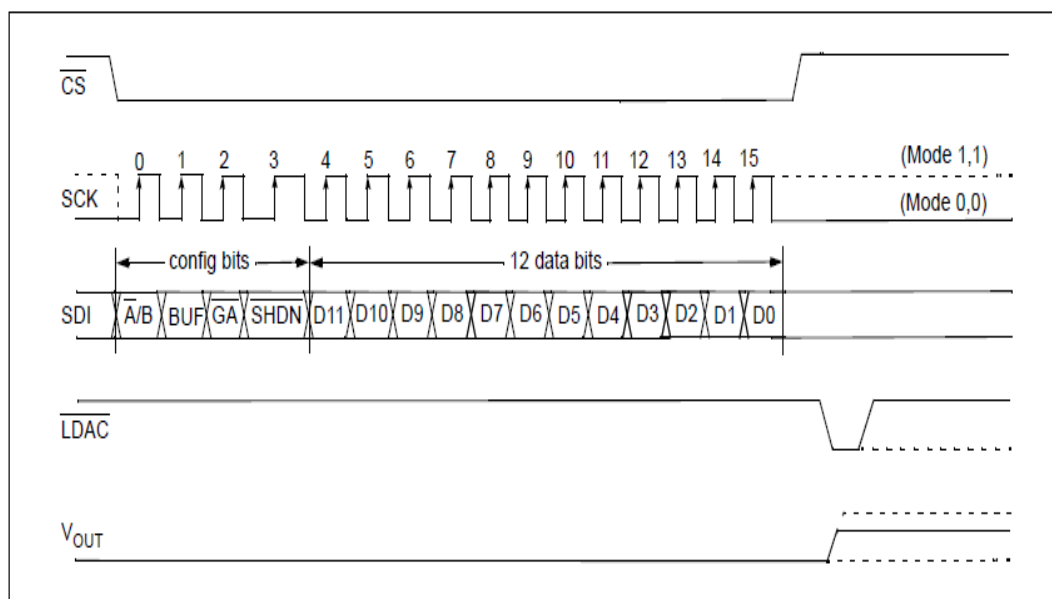


Figure 26: The SPI write command structure of DAC

From the figure 26, it can be concluded that the first 4-bit in the MSB byte are config bits. The first 4-bit are fixed to “0111”. This means DAC channel A is working and the input reference voltage is buffered. In addition, the output voltage value is set to

$$V_{out} = \frac{V_{ref}}{4096} * Data$$

$$V_{out} = Data \text{ mV}$$

*Data* is the 12-bit value.

In order to check the DAC is working properly, a program is write to use this DAC to generate a simple sawtooth waveform. The following flowchart describes the details of it.

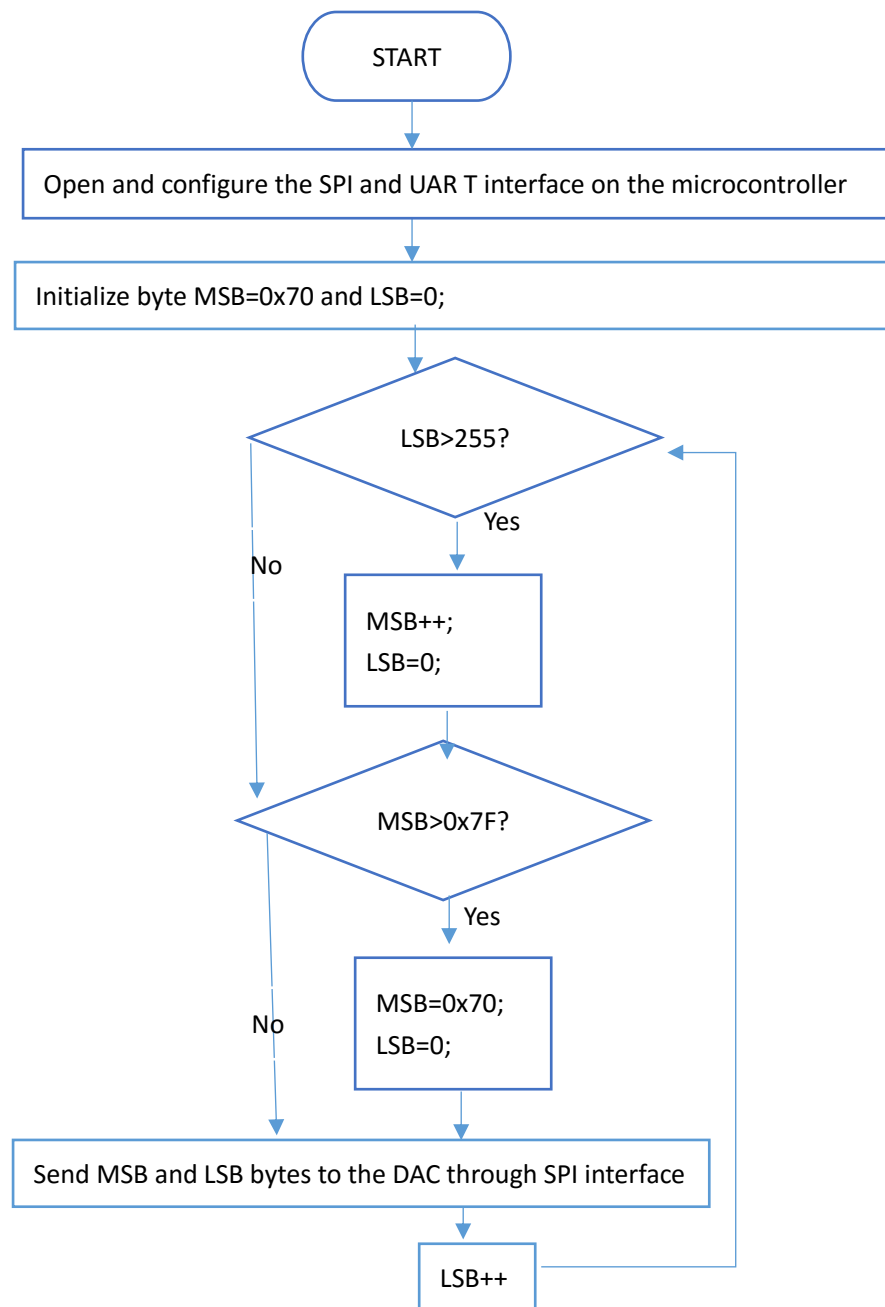


Figure 27: The flowchart of generating sawtooth waveform



### 3.7 I2C software design:

#### 3.7.1 PIC18F46k20 I2C interface initialization:

##### Header file:

##### #include:

---

```
#include <i2c.h>           //i2c library.
```

---

<spi.h> is an I2C peripheral library provided by XC8 compiler. The following table introduces the functions of the I2C library for PIC18F46k20 used in this project.

Table 5: PIC18F46k20 I2C library functions

Function:	Description:
CloseI2C	Disable I2C communications.
IdleI2C	Loop until I2C bus is idle.
getcI2C	Read a byte from the I2C bus.
getsI2C	Read a string from the I2C bus.
OpenI2C	Initialize the SSP module used for I2C communications.
putcI2C	Write a byte to the I2C bus.
putsI2C	Write a string to the I2C bus.
ReadI2C	Read a byte from the I2C bus.
WriteI2C	Write a byte to the I2C bus.
StartI2C	Generate an I <sup>2</sup> C bus Start condition.
RestartI2C	Generate an I <sup>2</sup> C bus Restart condition.
StopI2C	Generate an I <sup>2</sup> C bus Stop condition.

##### I2C Configuration:

Using PIC18F46k20 as master, according to the figure 16, the I2C pins are also connected on the PORTC. RC3 is SCL, RC4 is SDA. There is no need to configure the I2C pins manually. If the *OpenI2C* function is called, the RC3 and RC4 will be configured automatically.

To open the I2C communication, call the function *OpenI2C(MASTER, SLEW\_OFF)* and set *SSPADD=19*; This means the I2C is set as master mode and slew rate is disabled for 100kHz clock frequency mode. Because the internal clock frequency of the PIC18F46k20 is set to 8MHz, the I2C bus clock frequency is  $\frac{F_{osc}}{4 * (SSPADD + 1)} = \frac{8MHz}{4 * 20} = 100kHz$ . Therefore the bit rate is 100kbits/s (standard mode).

### 3.7.2 Arduino Leonardo I2C interface initialization:

#### Header file:

#### #include:

---

```
#include <wire.h>           //wire library.
```

---

<wire.h> is an I2C peripheral library provided by Arduino. The following table introduces the functions of the I2C library used in this project.

Table 6: Wire library for Arduino

Function	Description
Wire.begin	Initializes and start the I2C interface as master or slave.
Wire.beginTransmission	Begin a transmission to the I2C slave device with the given address. Subsequently, queue bytes for transmission with the <i>Wire.write()</i> function and transmit them by calling <i>Wire.endTransmission()</i> .
Wire.endTransmission	Ends a transmission to a slave device that was begun by <i>Wire.beginTransmission()</i> and transmits the bytes that were queued by <i>Wire.write()</i> .
Wire.requestFrom	Used by the master to request bytes from a slave device.
Wire.write	send a byte to the I2C bus
Wire.read	Read a byte from the I2C bus

#### I2C configuration:

Using Arduino Leonardo as master, SDL and SDA are connected on the digital pin 2 and pin 3. When the function *Wire.begin* is called, the pin 2 and pin 3 will be configured automatically.

To start the I2C communication, call the *Wire.begin()* function. This means the Arduino Leonardo is configured as master mode and the I2C clock frequency is set to 100 kHz mode as default. Therefore the bit rate is 100kbits/s (standard mode).

In *Wire.beginTransmission(address)* and *Wire.requestFrom(address, count)* functions, the arguments of the address is 7-bit address. If an 8-bit address is passed, the MSB will be ignored. These two functions will generate a START condition to the I2C bus. If *Wire.endTransmission()* is called, a STOP condition will be generated.

### 3.8 BME280 I2C software design:

According to the datasheet of the BME280, there are also five functions are designed to help users to obtain target data from the BME280. Two of them are I2C Write command and I2C Read command, which could allow users to send/read data to/from the corresponding register of the BME280 sensor. The other three functions are also to implement the compensation formulas of temperature, pressure and humidity to obtain accurate results through the I2C interface because of each sensing element behaves differently. The following pseudo code illustrate the details of these five functions.

---

BME280 I2C Write data function:

*Generate a START condition*

*Select slave by writing the 7-bit address(0x77) to the I2C bus to operate the Write command*

*Write an address register byte to the I2C bus*

*Write a data byte to the I2C bus*

*Generate a STOP condition*

---

BME280 I2C Read data function:

*Generate a START condition*

*Select slave by writing the 7-bit address(0x77) to the I2C bus to operate the Write command*

*Write an address register byte to the I2C bus*

*Generate a START condition again*

*Select slave by writing the device address to the I2C bus to operate the Read command*

*Read a data byte from the I2C bus and store in a variable x;*

*Generate a STOP condition*

*Return x*

---

BME280 Temperature compensation function:

*Read coefficient parameters dig\_T1, dig\_T2, dig\_T3 from the sensor through I2C interface*

*Use the coefficient parameters to operate the compensation formula and store the result in T*

*Return T;*

---

BME280 Pressure compensation function:

*Read coefficient parameters dig\_P1~dig\_P9 from the sensor through I2C interface*

*Use the coefficient parameters to operate the compensation formula and store the result in P*

*Return P;*

---

BME280 Humidity compensation function:

*Read coefficient parameters dig\_H1~dig\_H6 from the sensor through I2C interface*

*Use the coefficient parameters to operate the compensation formula and store the result*

*Return the result;*

---

The process of obtaining the temperature, pressure and humidity data through the I2C interface is very similar to the process of obtaining these data through the SPI interface.

### 3.9 MMA8452 accelerometer I2C software design:

According to the datasheet of the MMA8452 accelerometer, there are two functions are designed to send the I2C commands from the microcontrollers to the PmodACL2. One of them operates the Write command, which could allow users to send data to the corresponding register of the accelerometer. The other one operates the Read command, which could allow users to read data from the corresponding register of the accelerometer. The following pseudo code shows the detail of them.

---

MMA8452 I2C write function:

*Generate a START condition*

*Select slave by writing the 7-bit address (0x1D) to the I2C bus to operate the Write command*

*Write an address register byte to the I2C bus*

*Write a data byte to the I2C bus*

*Generate a STOP condition*

---

MMA8452 I2C Read function:

*Generate a START condition*

*Select slave by writing the 7-bit address (0x1D) to the I2C bus to operate the Write command*

*Write an address register byte to the I2C bus*

*Generate a RESTART condition*

*Select slave by writing the device address to the I2C bus to operate the Read command*

*Read a data byte from the I2C bus and store in a variable x;*

*Generate a STOP condition*

*Return x*

---

The following flowchart shows the whole process of getting the 3-axis acceleration from the MMA8452 through the I2C interface and displaying the result on the computer through the UART interface.

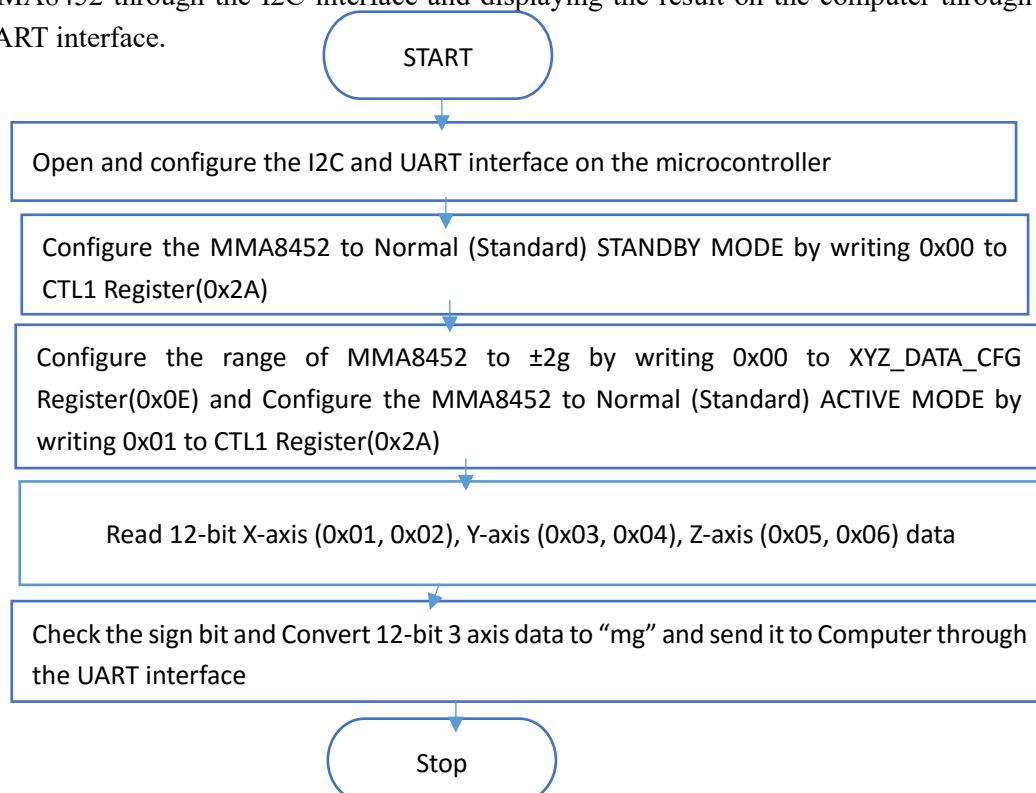


Figure 28: The flowchart of obtaining acceleration data from MMA8452

## Section 4: Results and Software Evaluation

### 4.1 UART:

#### 4.1.1 PIC18F46k20 UART Test and Results:

To test the designed function of PIC18F46k20 UART interface, a simple test code is written.

---

```

UARTWriteInt(222222,6); // 222222 is the integer. 6 represents to display 6 digits.
UARTWriteByte('\r'); //Carriage Return (Enter key)
UARTWriteByte('L'); // character L
UARTWriteByte('\r'); //Carriage Return (Enter key)
UARTWriteString("PIC UART TEST! "); //string

```

---

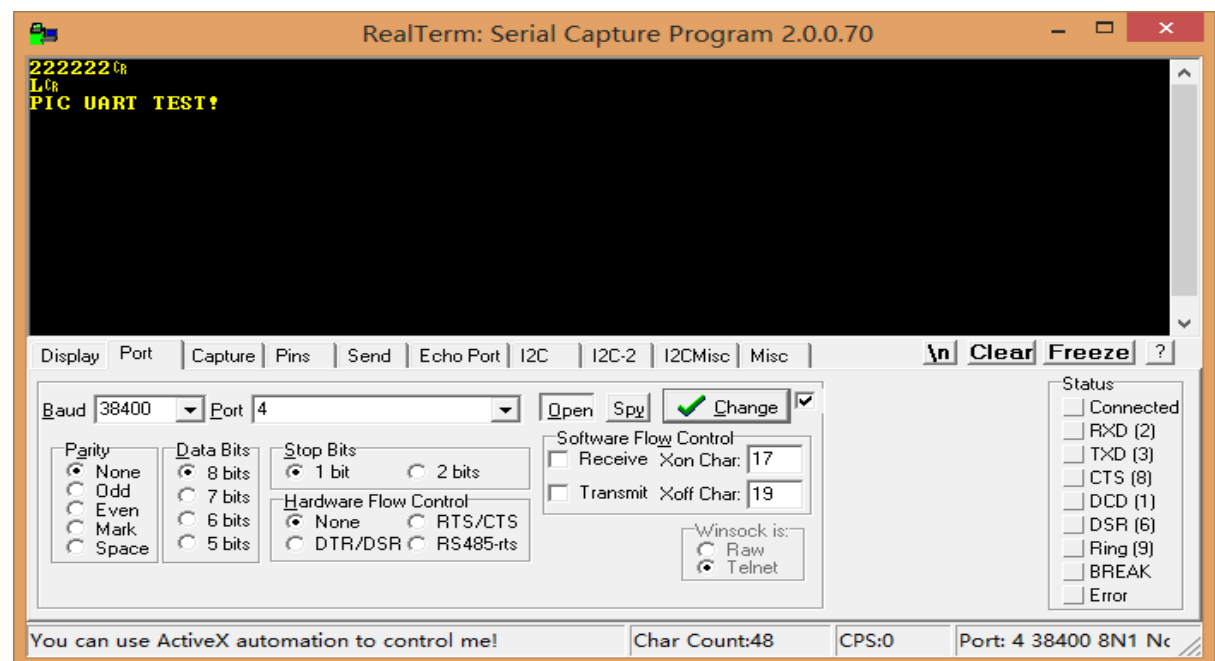


Figure 29: Test of PIC18F46k20 UART interface.

Figure 29 shows the results of test code. From the figure 29, it can be concluded that the three designed function works properly. Users can successfully send a byte, an 8-digit integer and string messages to the computer. In addition, the designed baud rate is 38461.53bps. The terminal baud rate is set to 38400 bps. In this case, the error is

$$\frac{38461.53 - 38400}{38400} * 100\% = 0.16\%$$

The error is relatively small and because the transmission will be resynchronized with the aids of start bit and stop bit (shown in figure 30). This error will not affect the transmission of the signals. In this case, the computer will receive the information correctly.

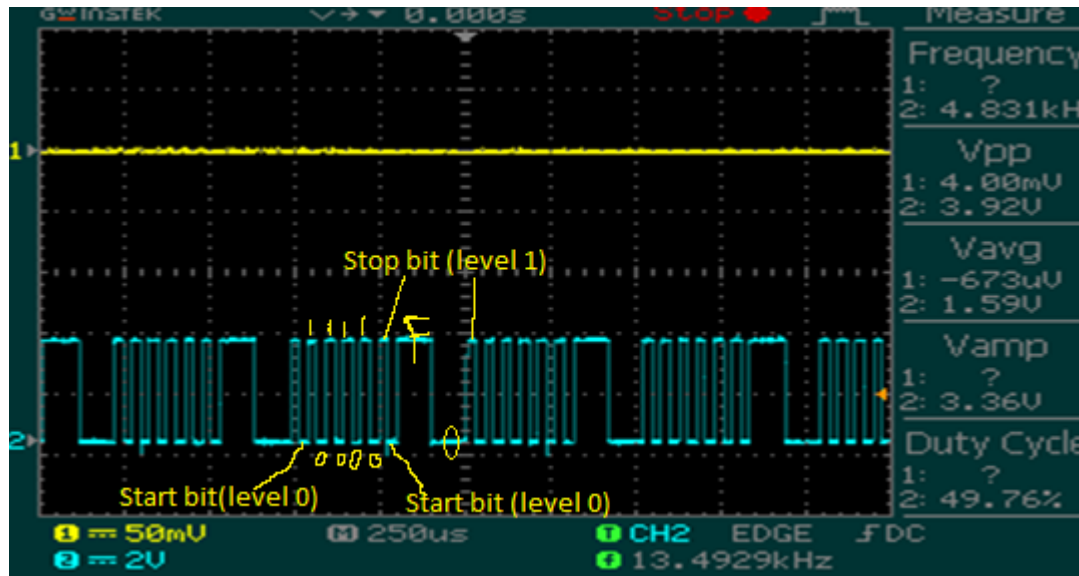


Figure 30: Waveform of transmitting 2 bytes 0x55 and 0xF0 to the Computer through PIC UART interface.

In the figure 30, counting from the stop bit to the start bit (LSB sent first), the first byte is 0b01010101 (0x55) and the second byte is 0b11110000 (0xF0). This agrees on the configurations of our design and the theory mentioned in the background.

#### 4.1.2 Arduino Leonardo UART Test and Results:

To test the Arduino Leonardo UART interface, the some simple test code is written to the microcontroller. Figure 31 shows the result of test code.

---

```
Serial.println(222222); //print 222222
Serial.println("L"); //print character L
Serial.println(0.05); //print float 0.05
Serial.println("ARDUINO LEONARDO UART TEST! "); //print string
```

---

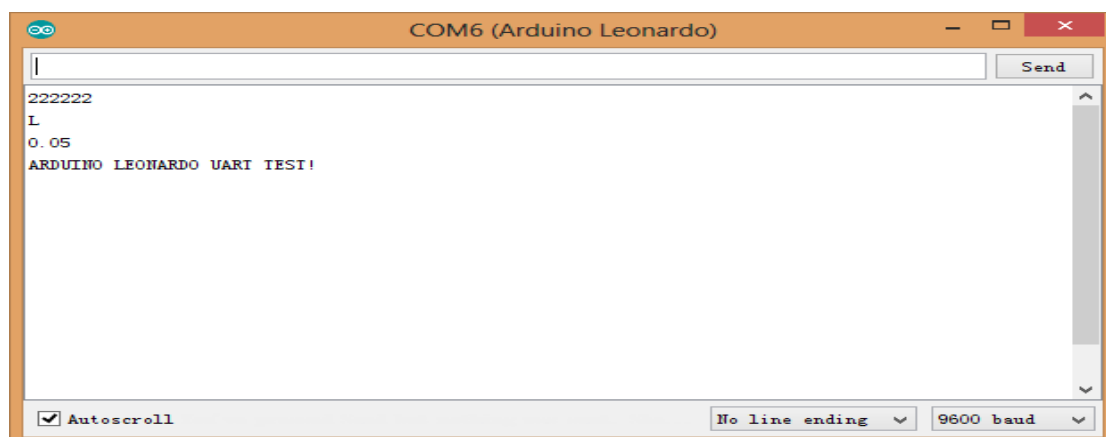


Figure 31: Test of Arduino Leonardo UART interface.

In the Figure 31, it can be concluded that the UART interface of Arduino works properly and it allows users to send a float, an integer, a byte and string messages to the computer.

## 4.2 SPI:

### 4.2.1 PIC SPI interface:

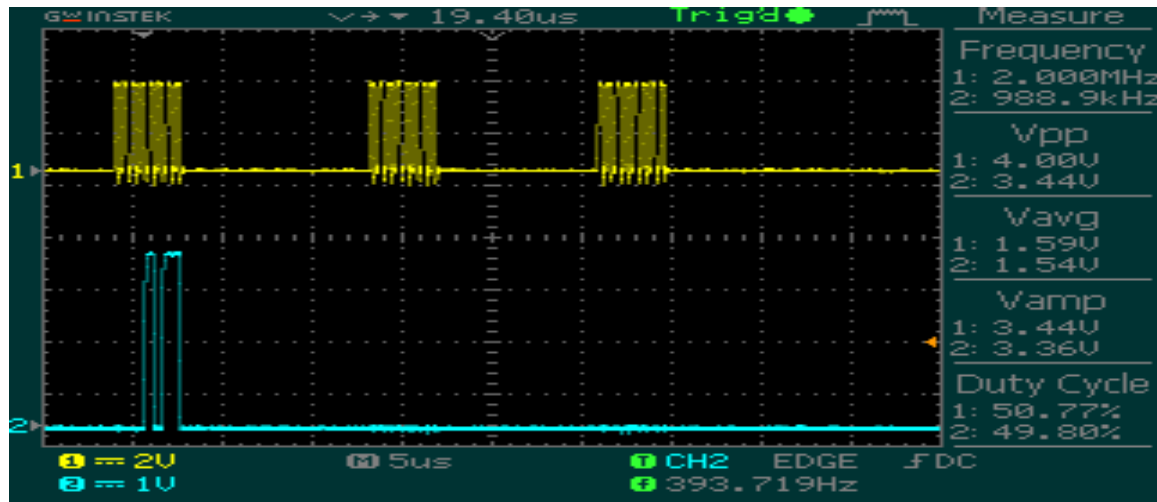


Figure 32: PIC SPI 2MHz clock frequency.

In the figure 32, the Channel 1(yellow line) is the clock frequency. It can be concluded that the clock frequency is 2.000MHz, which agrees on our original design and configuration (2MHz). The Channel 2 is MOSI. These digital signals represents 0x0B, 0x00 and 0x00 separately. It means 0x0B, 0x00 and 0x00 are written to the SPI bus.

### 4.2.2 Arduino SPI interface:

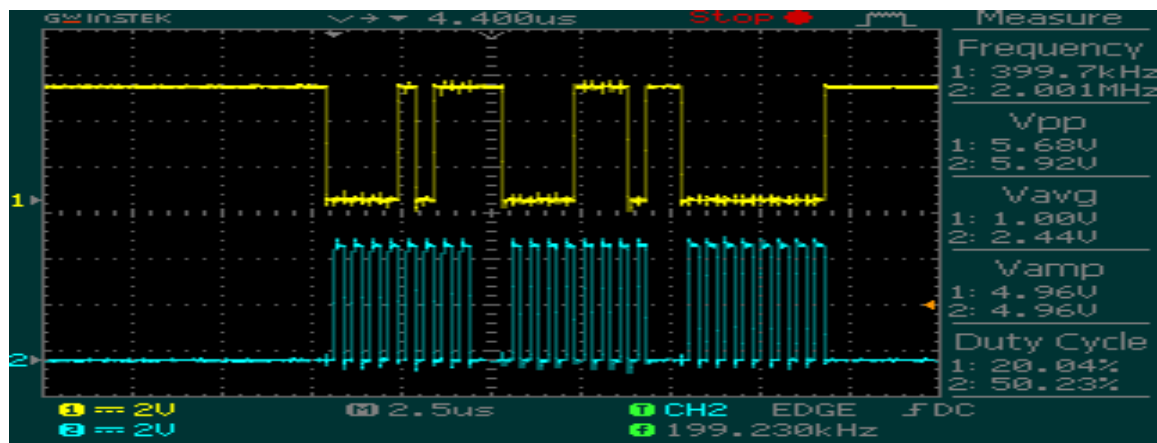


Figure 33: Arduino SPI interface.

In the figure 33, the Channel 1(yellow line) is MOSI. These digital signals represents 0x0B, 0x0E and 0x00. It means these three bytes are written to the Arduino SPI bus. The Channel 2 is SCK which represents the clock frequency of the SPI bus. From this figure, the clock frequency is 2.001MHz, which is 0.05% larger than our original design and configuration (2MHz). This error is very small which could be accepted. Because the transmission will be resynchronized every 8-bit.

### 4.2.3 DAC:

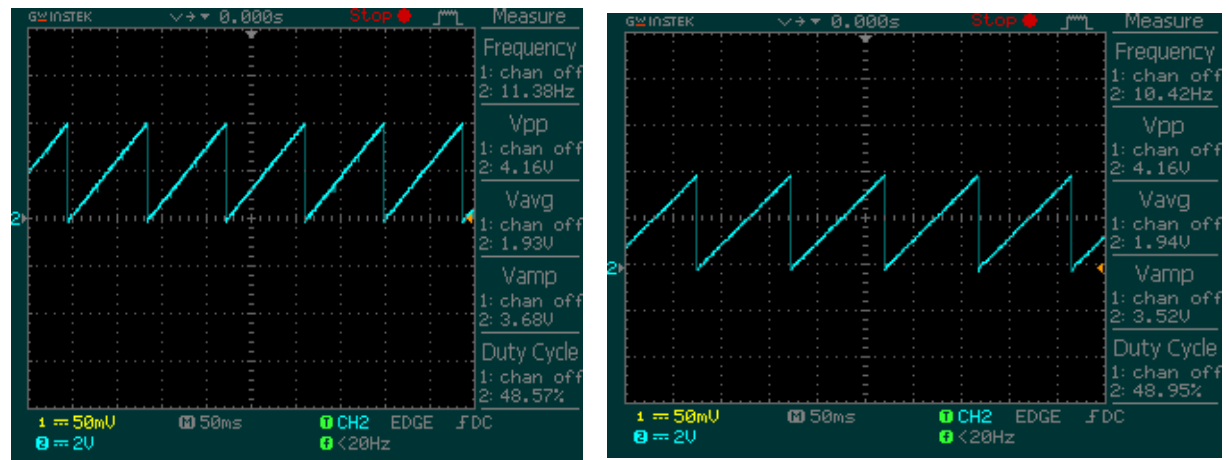


Figure 34: Sawtooth waveform output by PIC and Arduino SPI interface.

In the figure 34, the first screenshot on the left shows the DAC output sawtooth waveform produced by PIC SPI interface. The frequency of the waveform is 11.38Hz and the peak to peak voltage is 4.16V. However, the original design of the peak to peak voltage is 4.096V. The error is about 1.56%. This may be caused by the noises of the oscilloscope and the inductance of the connection wires. In addition, the system error also has a certain effect on the measurement. The second screenshot on the right shows the DAC output sawtooth waveform produced by Arduino SPI interface. The frequency of the waveform is 10.42Hz and the peak to peak voltage is also 4.16V. The difference between the frequencies of the waveforms may be caused by the compilers. This means the same design code segments may be transferred to different machine code and may consume different instructions cycles. Above all, the two screenshots demonstrate that DAC works properly on both microcontrollers through the SPI interface. They also demonstrates that the DAC could provide 0-4.096V output.

### 4.2.4 ADC:

There are three figures shows the test results of ADC through SPI interface. The sampling rate of ADC is 250kHz.

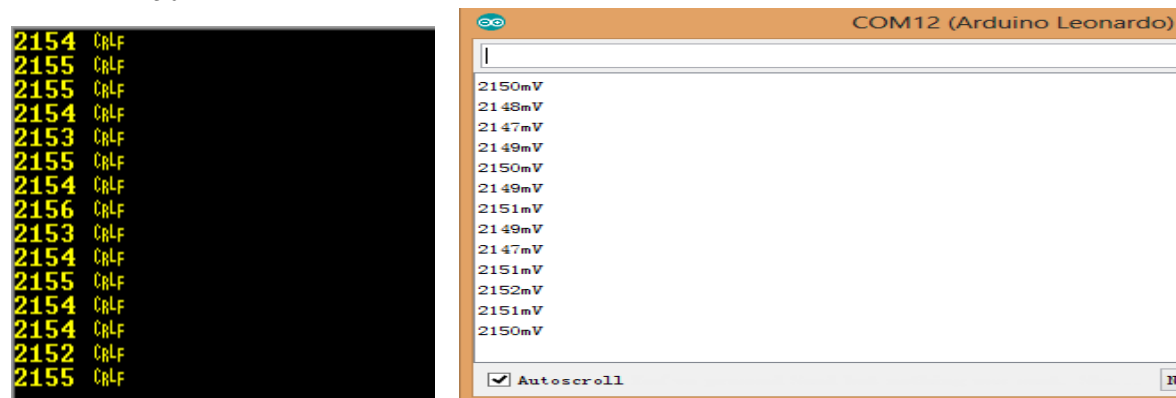


Figure 35: 2.15V ADC test results of PIC and Arduino through SPI interface.



2.15V DC signal supplied by digital power supply is passed to ADC. In the figure 35, the screenshot on the right is the result from PIC SPI interface and the screenshot on the left is the result from Arduino SPI interface. The average voltage of the first screenshot is about 2.154V. The average voltage of the second screenshot is about 2.150V. The error may be caused by the noises of digital power supply.

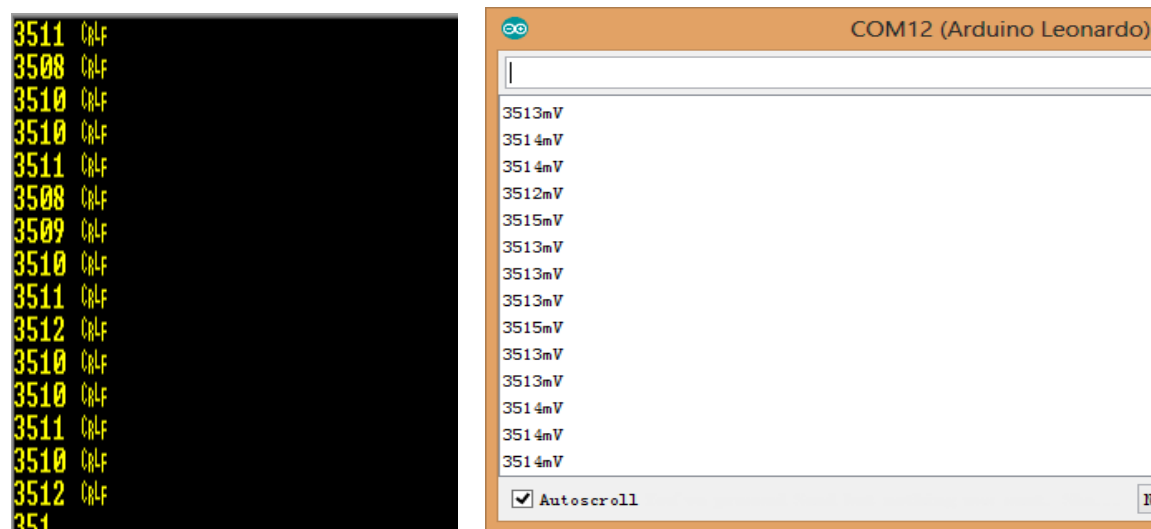


Figure 36: 3.51V ADC test results of PIC and Arduino through SPI interface.

3.51V DC signal supplied by digital power supply is passed to ADC. In the figure 36, the screenshot on the right is the result from PIC SPI interface and the screenshot on the left is the result from Arduino SPI interface. The average voltage of the first screenshot is about 3.511V. The average voltage of the second screenshot is about 3.513V. The error may also be caused by the noises of digital power supply.

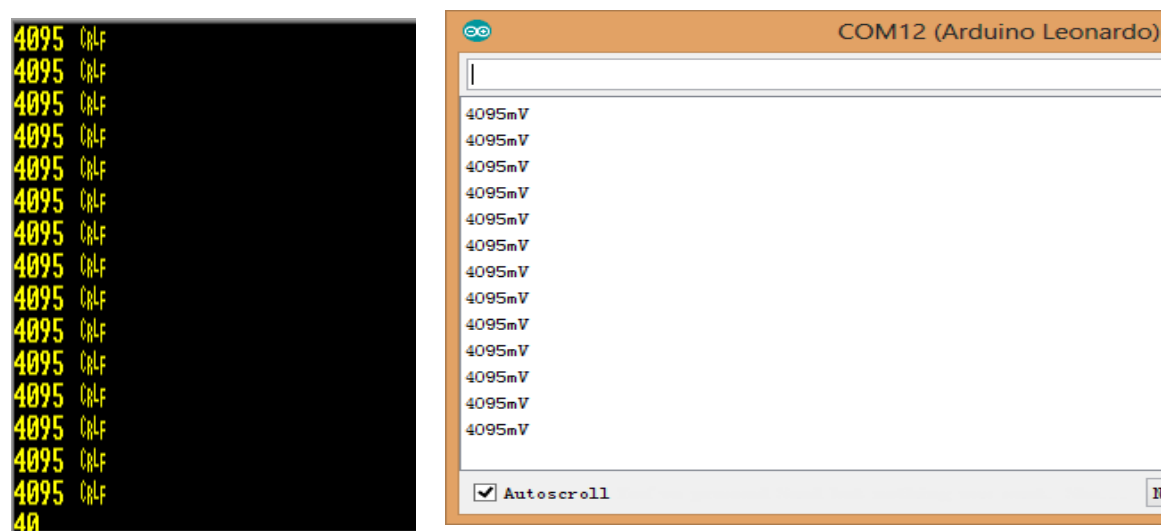


Figure 37: 4.50V ADC test results of PIC and Arduino through SPI interface.

3.51V DC signal supplied by digital power supply is passed to ADC. In the figure 37, the screenshot on the right is the result from PIC SPI interface and the screenshot on the left is the result from Arduino SPI interface. Both average voltage of the first and second screenshot are

4.095V. This is because the reference voltage of this ADC is 4.096V. This means, if the voltage is larger than 4.096V, the output is always 4.096V. In this case, the input voltage is 4.5V. Therefore, 4.096V is obtained.

Above all, these three figures demonstrates that SPI interfaces of PIC and Arduino for the ADC work properly and it could read 0~4.096V signals.

#### 4.2.5 PmodACL2:

The following figures shows the test results of PmodACL2 3-axis accelerometer through SPI interface.

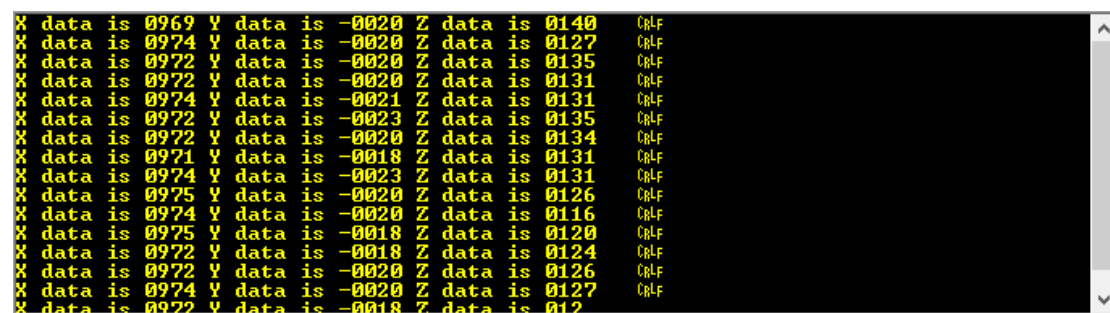


Figure 38: Test result of PIC SPI interface when x-axis along the direction of gravity.

In the figure 38, it can be obtained that when x-axis is parallel to the direction of gravity, the x-axis acceleration data is about 973mg, y-axis acceleration data is about -20mg and z-axis acceleration data is about 125mg.

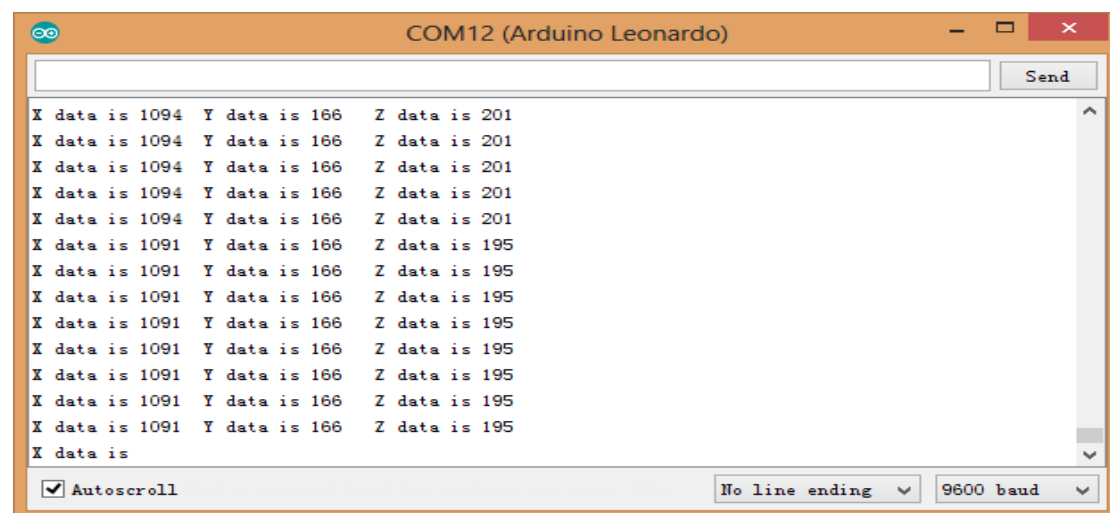


Figure 39: Test result of Arduino SPI interface when x-axis along the direction of gravity.

In the figure 39, it can be obtained that when x-axis is parallel to the direction of gravity, the x-axis acceleration data is about 1093mg, y-axis acceleration data is about 166 mg and z-axis acceleration data is about 197mg.

According to the theory, when one axis is along with the direction of gravity, this axis' acceleration data will be 1g and the other axes' acceleration data will be 0. From figure 38 and figure 39, it can be concluded that the test results follow this theory. This demonstrates that our SPI interfaces for PmodACL2 works properly. However, there is a significantly obvious errors. These errors may be caused by the noises during the measurement. In addition, it is really difficult to make the accelerometer axes parallel to the direction of gravity accurately. External force such as the tension applied by the wires for the connections may also affect the measurement.

In order to measure the dynamic acceleration data, the PmodACL2 sensor has been fixed on a swivel chair. A force has been applied to the chair for a while by hands to make the swivel chair rotating. In this case, the acceleration data has been captured and recorded for 8 seconds by the software Realterm. Matlab has been used to collect these data and the figure 40 has been plotted.

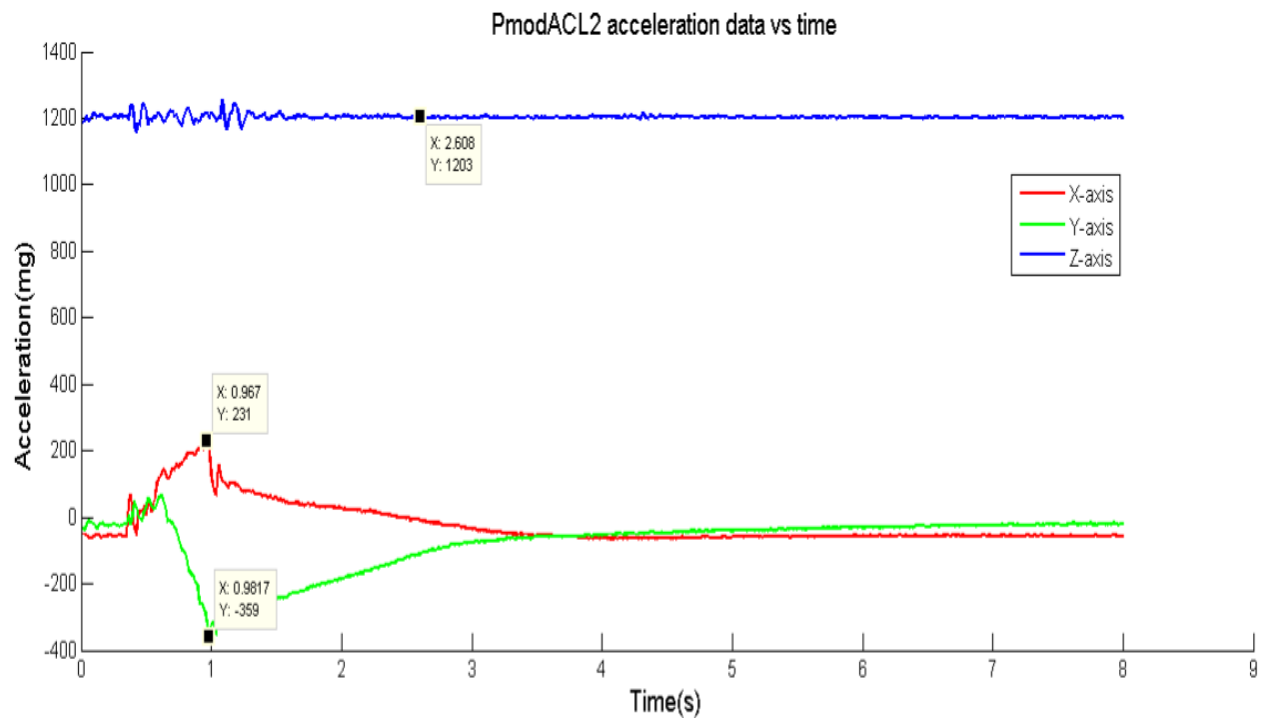
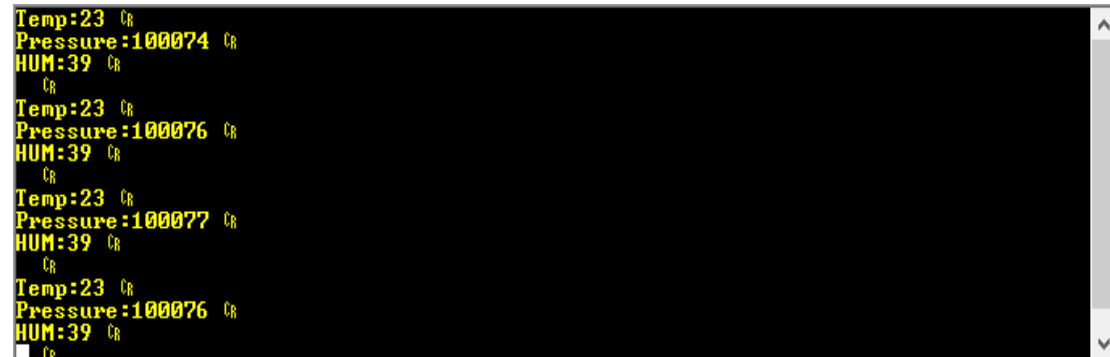


Figure 40: PmodACL2 acceleration data on the rotating chair.

In the figure 40, the blue line is the Z-axis acceleration data and its direction is always along with the gravity. Therefore, the z-axis acceleration remains to about 1203mg. The x-axis acceleration data is the red line and it has been increased to 231mg at 0.967 seconds and then it is reduced to about 0mg at 3.8 seconds steadily by the frictional force. Similar, the y-axis acceleration (green line) has been increased to -359 mg at 0.9817 seconds and then it is reduced to about 0mg at 3.8 seconds by the frictional force steadily. It indicates that the force applied to the chair lasts about 0.5 seconds and it takes about 2.8 seconds for the chair to be stopped by the friction. This demonstrates that the PmodACL2 works properly for the dynamic measurement.

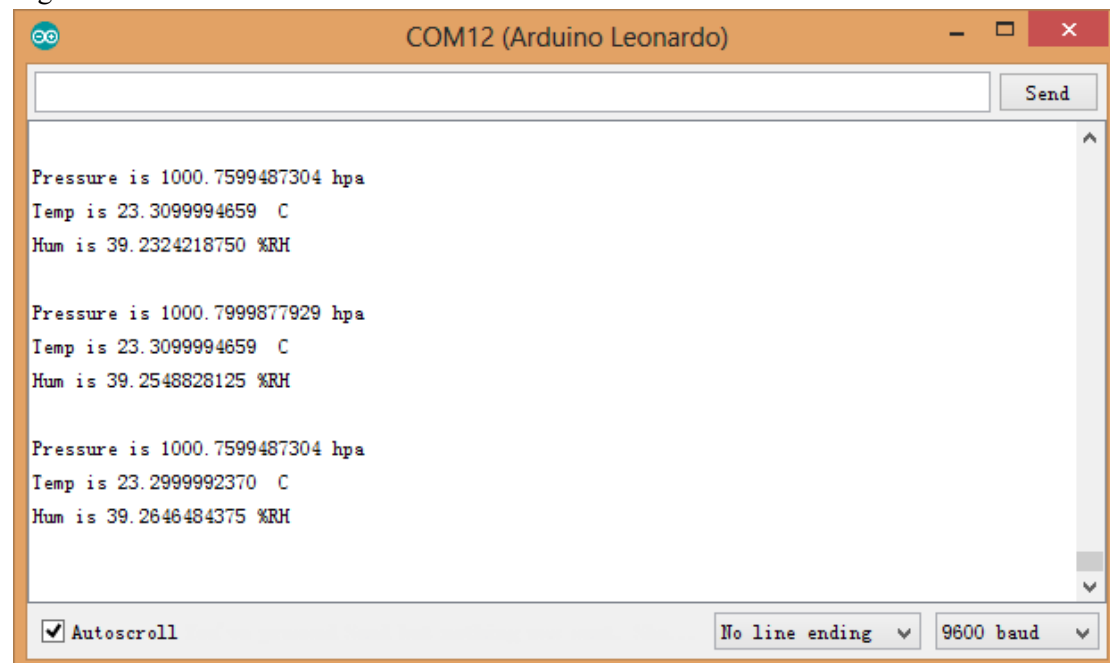
#### 4.2.6 BME280:

The following figures show the test results of BME280 environmental sensor through SPI interface.



```
Temp:23 CR
Pressure:100074 CR
HUM:39 CR
Temp:23 CR
Pressure:100076 CR
HUM:39 CR
Temp:23 CR
Pressure:100077 CR
HUM:39 CR
Temp:23 CR
Pressure:100076 CR
HUM:39 CR
```

Figure 41: Test result of BME280 PIC SPI interface



```
Pressure is 1000.7599487304 hpa
Temp is 23.3099994659 C
Hum is 39.2324218750 %RH

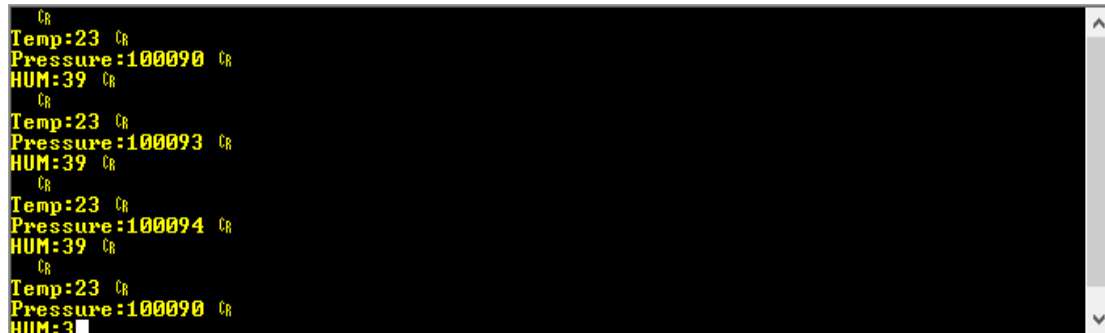
Pressure is 1000.7999877929 hpa
Temp is 23.3099994659 C
Hum is 39.2548828125 %RH

Pressure is 1000.7599487304 hpa
Temp is 23.2999992370 C
Hum is 39.2646484375 %RH
```

Figure 42: Test result of BME280 Arduino SPI interface.

From the figure 41 and figure 42, it can be concluded that the results from SPI interfaces of Arduino and PIC are almost the same. Therefore, on the 12<sup>th</sup> April 2016, in Nottingham, the temperature of my room is about 23°C, the barometric pressure is about 1000.76hpa and the humidity is about 39% RH.

The above test has been done on my desk. In order to check the barometric pressure sensor works properly, the sensor has been placed in a lower place, for example the sensor is placed on the floor. In this case, figure 43 and figure 44 displays the test result.

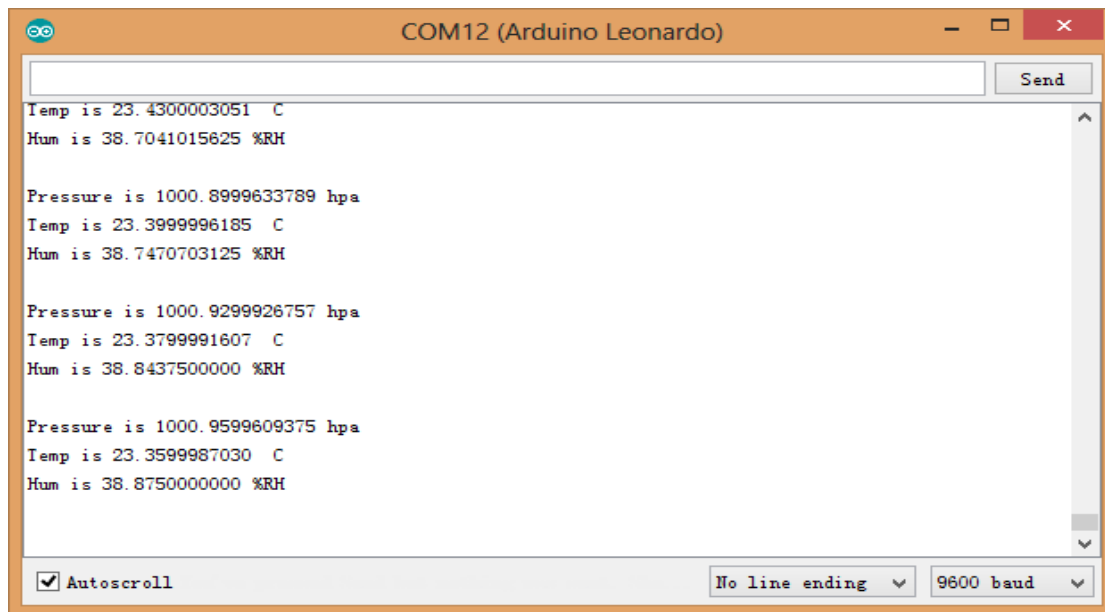


```

CR
Temp:23 CR
Pressure:100090 CR
HUM:39 CR
CR
Temp:23 CR
Pressure:100093 CR
HUM:39 CR
CR
Temp:23 CR
Pressure:100094 CR
HUM:39 CR
CR
Temp:23 CR
Pressure:100090 CR
HUM:39

```

Figure 43: Test result of BME280 PIC SPI interface when the sensor is placed on the floor.



```

Temp is 23.4300003051 C
Hum is 38.7041015625 %RH

Pressure is 1000.8999633789 hpa
Temp is 23.3999996185 C
Hum is 38.7470703125 %RH

Pressure is 1000.9299926757 hpa
Temp is 23.3799991607 C
Hum is 38.8437500000 %RH


Pressure is 1000.9599609375 hpa
Temp is 23.3599987030 C
Hum is 38.8750000000 %RH

```

Figure 44: Test result of BME280 Arduino SPI interface when the sensor is placed on the floor.

From the figure 43 and figure 44, and compared with the figure 41 and figure 42, it can be found that the measurement values of temperature and humidity remain the same when the sensor is placed on the floor. However, the barometric pressure has been increased to 1000.93 hpa. The increase is because in lower place, there is a greater column of air. The larger density of air, the larger barometric pressure. This demonstrates that the barometric pressure sensor in the BME280 works properly through the SPI interface.

In order to check the temperature sensor works properly, the sensor has been warmed by hands. In this case, figure 45 and figure 46 displays the test results.

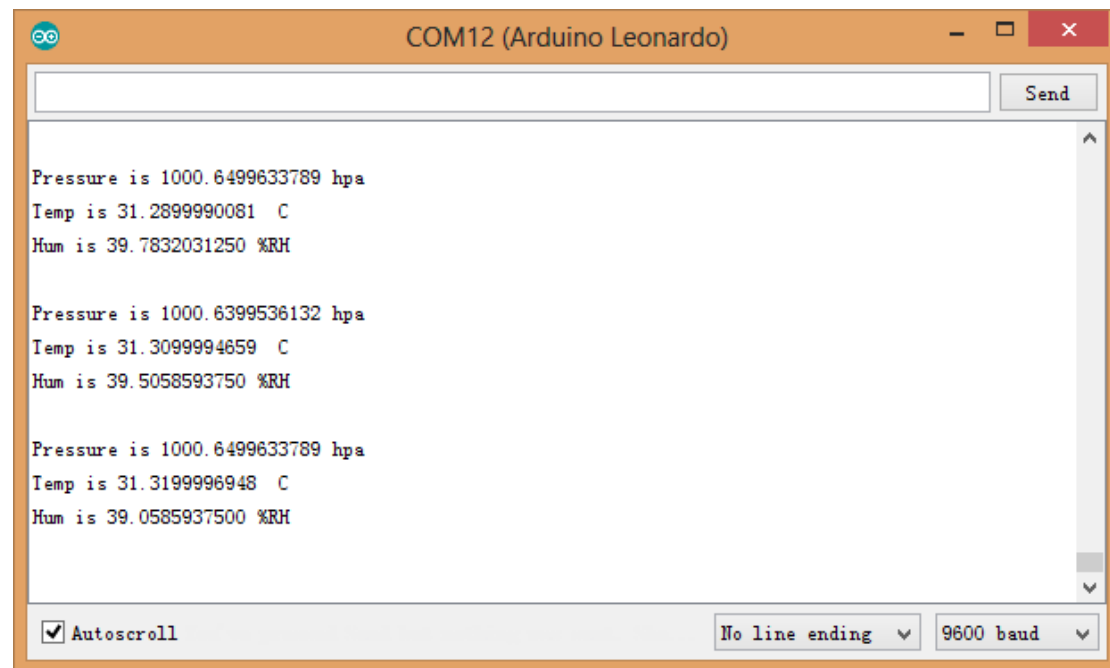


```

CR
Temp:32 CR
Pressure:100062 CR
HUM:39 CR
CR
Temp:32 CR
Pressure:100074 CR
HUM:39 CR
CR
Temp:32 CR
Pressure:100078 CR
HUM:39 CR
CR
Temp:32 CR
Pressure:100082 CR
HUM:39

```

Figure 45: Test result of BME280 PIC SPI interface when the sensor is warmed by hands



```

Pressure is 1000.6499633789 hpa
Temp is 31.2899990081 C
Hum is 39.7832031250 %RH

Pressure is 1000.6399536132 hpa
Temp is 31.3099994659 C
Hum is 39.5058593750 %RH

Pressure is 1000.6499633789 hpa
Temp is 31.3199996948 C
Hum is 39.0585937500 %RH

```

Figure 46: Test result of BME280 Arduino SPI interface when the sensor is warmed by hands

From the figure 45 and figure 46, compared with the figure 41 and 42, it can be concluded that the humidity almost remain the same, the barometric pressure has a slightly change and the temperature has been increased steadily. This demonstrates that the temperature sensor in the BME280 works properly through the SPI interface. The slight change of the pressure may be caused by the noises. In addition, the temperature also has an effect on the barometric pressure.

In order to check the humidity sensor works properly, the sensor has been breathed upon an exhaled air. In this case, figure 47 and figure 48 displays the test results.

```

Temp:23 CR
Pressure:100090 CR
HUM:63 CR
CR
Temp:23 CR
Pressure:100095 CR
HUM:64 CR
CR
Temp:23 CR
Pressure:100092 CR
HUM:65 CR
CR
Temp:23 CR
Pressure:100093 CR
HUM:66 CR
CR

```

Figure 47: Test result of BME280 PIC SPI interface when the sensor is blown by mouth

```

COM12 (Arduino Leonardo)

Pressure is 1000.8999633789 hpa
Temp is 23.9499988555 C
Hum is 41.3925781250 %RH

Pressure is 1000.8899536132 hpa
Temp is 24.4099998474 C
Hum is 60.1123046875 %RH

Pressure is 1000.8899536132 hpa
Temp is 24.0799999237 C
Hum is 67.1093750000 %RH

☒ Autoscroll
No line ending 9600 baud

```

Figure 48: Test result of BME280 Arduino SPI interface when the sensor is blown by mouth

From the figure 47 and figure 48, compared with the figure 41 and 42, it can be concluded that the humidity has been raised rapidly, the barometric pressure and the temperature has a slight increase. This slight increase of the temperature and pressure is because the exhaled air is at core body temperature which is higher than the environmental temperature and the exhaled air will influence the air flow around the sensor. The reason for the humidity value grows rapidly is because the exhaled air carry a large amount of steam inside the human body. This demonstrates that the humidity sensor in the BME280 works properly through the SPI interfaces.

Above all, from the figure 41 to figure 48, it can be concluded that the SPI interfaces for BME280 works properly.

### 4.3 I2C:

#### 4.3.1 PIC I2C interface:

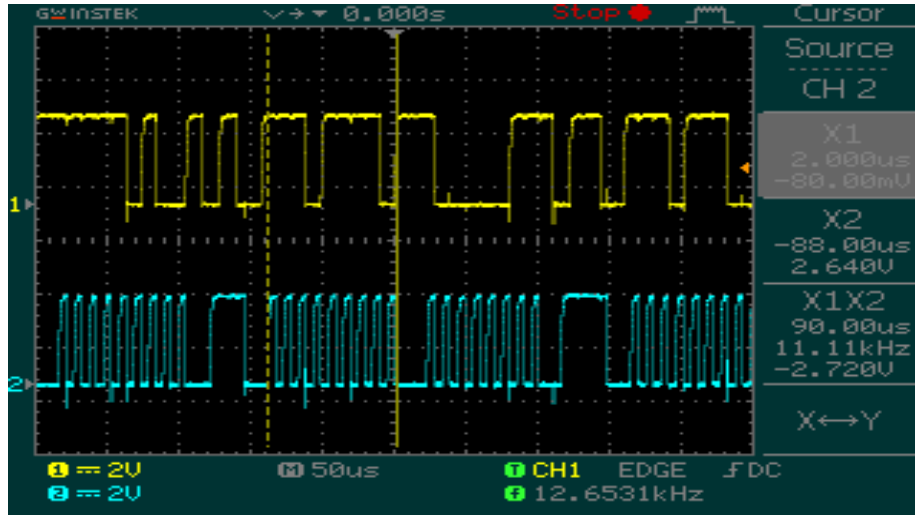


Figure 49: PIC I2C interface clock.

Figure 49 shows the transmission of the PIC I2C interface. The Channel 1 (Yellow line) is SDA and the Channel 2 (blue line) is SCL. There are 9 clock cycles counted between the two cursors. The time period between these two cursors is 90us. In this case, the clock frequency can be calculated as following:

$$\text{one clock cycle time period} = \frac{90\mu\text{s}}{9} = 10\mu\text{s}$$

$$\text{clock frequency} = \frac{1}{10\mu\text{s}} = 100\text{kHz}$$

In this case, the clock frequency is matched with our original design.

#### 4.3.2 Arduino I2C interface:

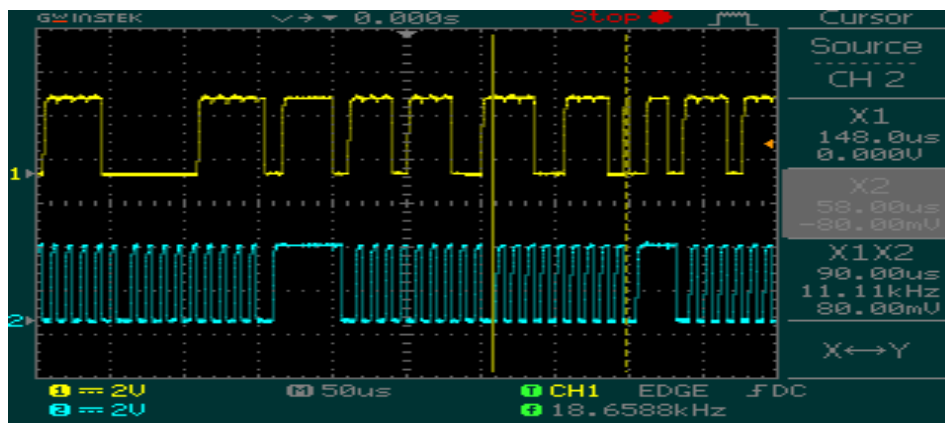


Figure 50: Arduino I2C interface clock.



Figure 50 shows the transmission of the Arduino I2C interface. The Channel 1 (Yellow line) is SDA and the Channel 2 (blue line) is SCL. There are also 9 clock cycles counted between the two cursors. The time period between these two cursors is also 90us. In this case, the clock frequency can be calculated as the same as above. Therefore, the clock frequency is also 100 kHz, which agrees with the original design.

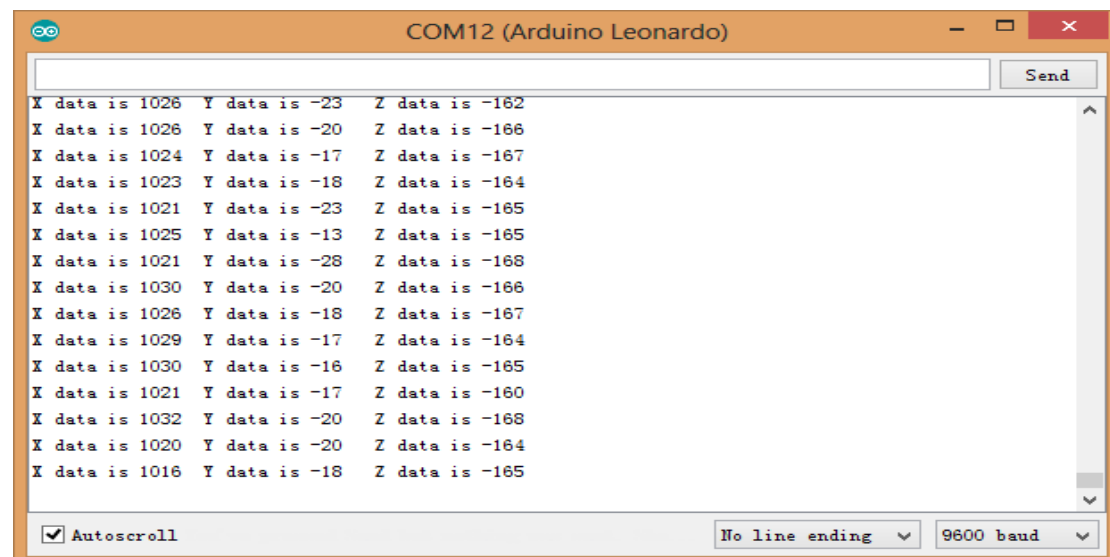
### 4.3.3 MMA8452:

The following figures shows the test results of MMA8452 through I2C interface.

```
X data: 1038 Y data: 0000 Z data: 0085 CrLf
X data: 1038 Y data: 0004 Z data: 0085 CrLf
X data: 1036 Y data: 0002 Z data: 0088 CrLf
X data: 1035 Y data: 0001 Z data: 0088 CrLf
X data: 1037 Y data: 0007 Z data: 0074 CrLf
X data: 1034 Y data: -0001 Z data: 0074 CrLf
X data: 1033 Y data: -0001 Z data: 0075 CrLf
X data: 1034 Y data: -0001 Z data: 0086 CrLf
X data: 1029 Y data: 0004 Z data: 0093 CrLf
X data: 1035 Y data: 0006 Z data: 0085 CrLf
X data: 1037 Y data: 0007 Z data: 0082 CrLf
X data: 1034 Y data: -0001 Z data: 0073 CrLf
X data: 1034 Y data: 0002 Z data: 0075 CrLf
X data: 1032 Y data: 0007 Z data: 0076 CrLf
X data: 1041 Y data: 0004 Z data: 0078 CrLf
```

Figure 51: Test result of PIC I2C interface when x-axis along the direction of gravity.

In the figure 51, it can be obtained that when x-axis is parallel to the direction of gravity, the x-axis acceleration data is about 1035mg, y-axis acceleration data is about 3mg and z-axis acceleration data is about 82mg.



```
X data is 1026 Y data is -23 Z data is -162
X data is 1026 Y data is -20 Z data is -166
X data is 1024 Y data is -17 Z data is -167
X data is 1023 Y data is -18 Z data is -164
X data is 1021 Y data is -23 Z data is -165
X data is 1025 Y data is -13 Z data is -165
X data is 1021 Y data is -28 Z data is -168
X data is 1030 Y data is -20 Z data is -166
X data is 1026 Y data is -18 Z data is -167
X data is 1029 Y data is -17 Z data is -164
X data is 1030 Y data is -16 Z data is -165
X data is 1021 Y data is -17 Z data is -160
X data is 1032 Y data is -20 Z data is -168
X data is 1020 Y data is -20 Z data is -164
X data is 1016 Y data is -18 Z data is -165
```

Figure 52: Test result of Arduino I2C interface when x-axis along the direction of gravity.

In the figure 52, it can be obtained that when x-axis is parallel to the direction of gravity, the x-axis acceleration data is about 1030mg, y-axis acceleration data is about -20mg and z-axis acceleration data is about -165mg.

Similar as the PmodACL2 accelerometer, the test results of MMA8452 accelerometer matches the principles of 3-axis MEMS accelerometer. This demonstrates that our I2C interfaces for MMA8452 works properly. However, compared with the PmodACL2 accelerometer, the errors of the MMA8452 is much smaller.

In order to measure the dynamic acceleration data, the MMA8452 sensor has also been fixed on a swivel chair. Similarly, a force has been applied to the chair for half a second by hands to make the swivel chair rotating. The acceleration data has been captured and recorded for 8 seconds and figure 53 has been plotted.

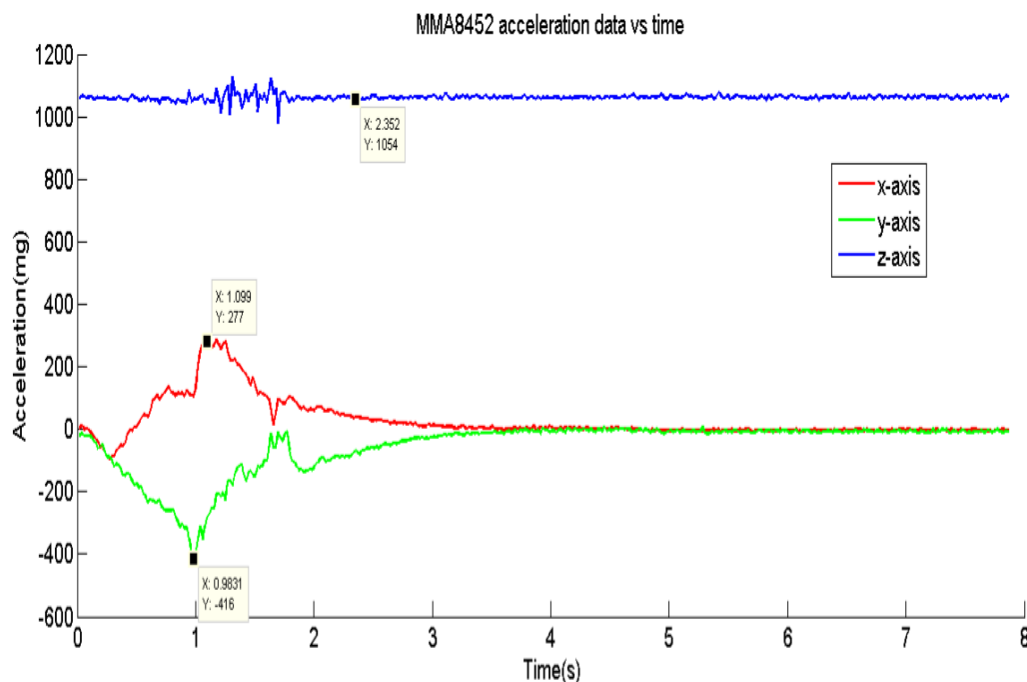


Figure 53: MM8452 acceleration data on the rotating chair.

In the figure 53, the blue line is the Z-axis acceleration data and its direction is always along with the gravity. Therefore, the z-axis acceleration remains to about 1064mg. The x-axis acceleration data is the red line and it has been increased to 277mg at 1.099 seconds and then it is reduced to about 0mg at 3.7 seconds steadily by the frictional force. Similar, the y-axis acceleration (green line) has been increased to -416mg at 0.9831seconds and then it is reduced to about 0mg at 3.7 seconds by the frictional force steadily. It indicates that the force applied to the chair lasts about 0.8 seconds and it takes about 2.7 seconds for the chair to be stopped by the friction. This demonstrates that the MMA8452 works properly for the dynamic measurement.

### 4.3.5 BME280 I2C interfaces:

The test results of the BME280 I2C interfaces are similar to the test results of the BME280 SPI interfaces. There is no significant difference between the SPI interfaces and I2C interfaces for this environmental sensor. In addition, a long term record of my room's environment is made by using I2C interfaces. The following table shows this long term record.

Table 7: A long term record of my room's environment

Date	Temperature	Barometric pressure	Humidity
12 <sup>th</sup> April 2016	23.3 °C	1000.79 hpa	39.3% RH
13 <sup>th</sup> April 2016	22.2 °C	1003.95 hpa	41.6% RH
14 <sup>th</sup> April 2016	21.7 °C	1003.46 hpa	46.2% RH
15 <sup>th</sup> April 2016	20.3 °C	993.97 hpa	53.4% RH
16 <sup>th</sup> April 2016	20.2 °C	1001.43 hpa	39.3% RH
17 <sup>th</sup> April 2016	21.7 °C	1012.97 hpa	35.5% RH
18 <sup>th</sup> April 2016	21.6 °C	1016.04 hpa	37.6% RH
19 <sup>th</sup> April 2016	21.8 °C	1026.61 hpa	35.9% RH
20 <sup>th</sup> April 2016	21.4 °C	1025.03 hpa	34.6% RH
21 <sup>th</sup> April 2016	21.3 °C	1019.90 hpa	36.9% RH
22 <sup>th</sup> April 2016	17.7 °C	1016.88 hpa	37.6% RH
23 <sup>th</sup> April 2016	20.4 °C	1020.83 hpa	35.1% RH
24 <sup>th</sup> April 2016	16.7 °C	1014.08 hpa	40.3% RH
25 <sup>th</sup> April 2016	20.8 °C	1002.77 hpa	44.1% RH
26 <sup>th</sup> April 2016	20.3 °C	1003.61 hpa	35.2% RH
27 <sup>th</sup> April 2016	20.5 °C	1006.82 hpa	35.6% RH
28 <sup>th</sup> April 2016	20.1 °C	1004.12 hpa	36.1% RH

## Section 5: Hardware design and evaluation

### 5.1 Hardware design:

With the help of the tutorial [30] on the Arduino website, a self-made Arduino microcontroller is built on the strip board to test our SPI and I2C interfaces. The following shows the details of how to build the Arduino with Atmega328 AVR microcontroller and FTDI FT232 USB to Serial converter.

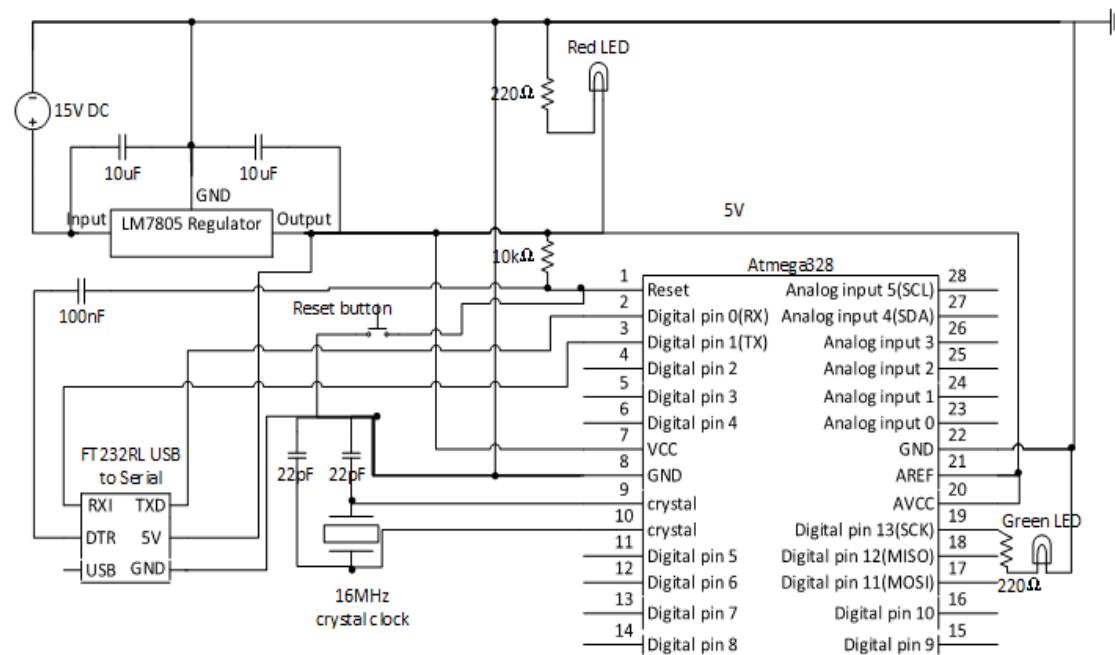


Figure 54: Schematics of Arduino design.

Figure 54 shows the details of the design of the self-made Arduino. The designed Arduino includes these features:

1. LM7805 voltage regulator parallel with two 10uF decoupling capacitors is used to convert 15V DC voltage to 5V DC voltage to provide the power supply for the Arduino.
2. A 16MHz crystal oscillator with 2 22pF capacitors is connected between pin 9 and pin 10 to provide an external clock for the microcontroller.
3. A red LED in serial with a 220Ω resistor is connected with the power supply for indicating the power is on.
4. A green LED in serial with a 220Ω resistor is connected with pin 13 to implement the LED Blink program for the check and evaluations.
5. FTDI FT232 USB to Serial converter is used to burn the program into the Atmega328 microcontroller from the computer. It can also provide 5V DC voltage as power supply for the Arduino.
6. A button with 10kΩ is used as reset button for users to reset the microcontroller during the operations.

Figure 55 shows the stripboard design of the self-made Arduino.

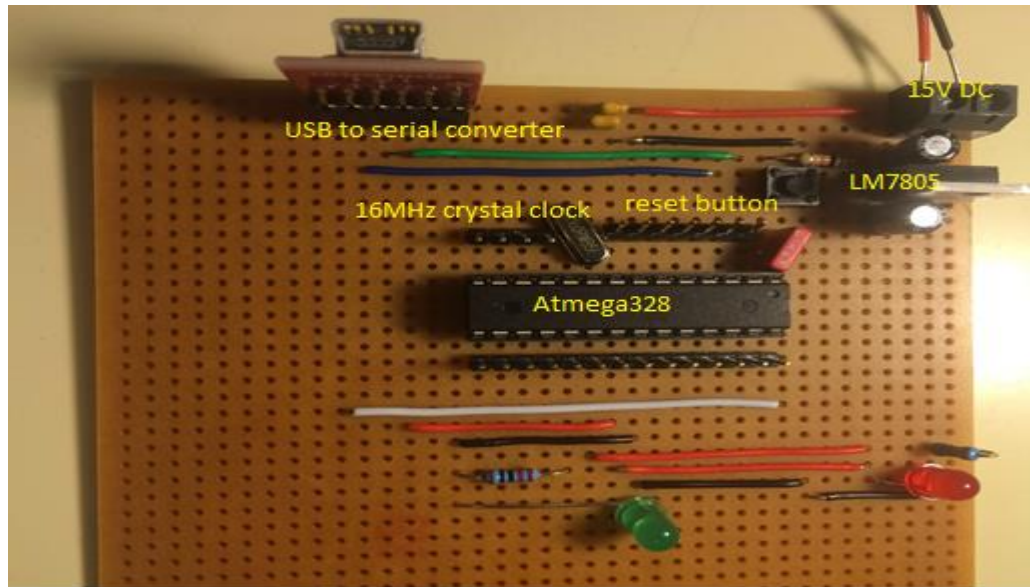


Figure 55: The stripboard design of the Arduino.

## 5.2 Burn Bootloader:

In order to burn the program into the self-made Arduino, bootloader should be burnt to the Atmega328 microcontroller first. With the help of the tutorial [31] on the Arduino website, the following discuss how to use Arduino Leonardo to burn the bootloader for the Atmega328. First, upload the ArduinoISP sketch to the Arduino Leonardo board. Second, wire up the Atmega328 and the Arduino Leonardo as illustrated in the Table 8.

Table 8: Connections of using Arduino Leonardo to burn the bootloader for the Atmega328

Atmega328	Arduino Leonardo
MISO pin18	ICSP pin MISO
MOSI pin 17	ICSP pin MOSI
SCK pin 19	ICSP pin SCK
Reset pin 1	Digital pin 10
VCC pin 7	5V pin
GND pin 8	GND pin

Then, Select "Arduino Duemilanove or Nano w/ ATmega328" -Tools. Finally, select "Burn bootloader" –Tools and wait for the procedure of "Burn bootloader" to be finished.

## 5.3 Evaluations:

After the bootloader is burnt, the self-made Arduino could be programed by FTDI FT232 USB to Serial converter. A "Blink" sketch which makes the green LED on or off for 2 second, is uploaded to the self-made Arduino. It can be observed that the green LED will blink every two seconds. In addition, the sensors mentioned above are used to test with the self-made Arduino. The test results are same to the results shown in the Section 4. Therefore, both SPI and I2C interfaces of self-made Arduino work properly.

## Section 6: Discussion

### 6.1 Compare SPI and I2C interfaces:

#### Bus topology:

Normal SPI interface requires four wires to establish the communications. However, I2C interface only needs two wires to transfer information between the devices. Furthermore, if more slaves are added in a system, SPI will need more pins to provide SS or CS signals for the master to select the slave and more wires for the connections, while I2C will not need extra pins and less wires for the connections. In this case, to build a complex network system, I2C interface is better than SPI interface.

#### Clock frequency and Speed:

For the clock frequency, SPI is not limited. This means, the SPI clock frequency could be configured to any value. However, the clock frequencies of I2C interface are fixed. I2C interfaces can only operate in low-speed mode (10 kHz), standard mode (100 kHz), fast mode (400 kHz) and high-speed mode (3.4 MHz). Usually, SPI interface clock is implemented above 10MHz. In this case, for high speed data communications, SPI interface is better than I2C interface.

#### Stability:

Both SPI and I2C interfaces are robust. SPI is full-duplex. In this case, SPI is more flexible and easier to monitor by Oscilloscope. I2C is half-duplex. In this case, I2C interface is not as easy as the SPI interface in monitoring by Oscilloscope. However, I2C is less vulnerable to noise than SPI and I2C is more suitable for long distance transmission.

Above all, whether SPI interface or I2C interface is better is a trade-off problem. Both SPI and I2C interfaces are stable and robust and both SPI and I2C interfaces have their own advantages and disadvantages. The following table shows the detail of them.

Table 9: I2C vs SPI

	Advantages	Disadvantages
I2C	2-wire connections, less pins needed	Only 4 types of clock speed and the Maximum speed is 3.4Mbit/s
	Enable multiple masters on the same bus	Chip addressing may cause address conflicts
SPI	Full-duplex communications, simpler protocols and easier to monitor.	Need more wires to establish the connections.
	More flexible and faster clock speed and longer ranges	Only support single master on the same bus.

## 6.2 Compare PIC vs Arduino:

### Programming language:

In PIC, C is the primary language for PIC used in this project. C is a mid-level language. This means C is relatively hard for human to read, but C is relatively simple for machine to recognize. While in Arduino, the programming language is a set of C/C++ functions but a slightly different. The language used in Arduino supports object-oriented programming. This means this programming language is a high-level language. Therefore, it is easy for users to read but hard for the microcontroller to operate.

### Implementations of I2C and SPI interfaces:

For the implementations of I2C and SPI interfaces, both PIC and Arduino provide libraries. The Arduino's libraries are much more convenient and friendlier. For example, to configure the pins as analog or digital I/O, in Arduino, only one function: *pinMode(pin, mode(INPUT/OUTPUT))* is needed to be called, while in the PIC, users need to go through the PIC datasheet to find several function registers, such as TRISE and ANSEL registers and users need to set these registers with related values. For Arduino, users do not require many complex configurations and settings, while the PIC's libraries are basic to some extent. This means, users need a certain hardware knowledge to support them to establish the SPI and I2C interfaces in PIC. In addition, the settings and configurations in PIC are more complex, but this complexity makes the PIC more powerful than Arduino. For example, to start the I2C interfaces, for Arduino, just call *Wire.begin()* function and the Arduino I2C interface will be set to standard mode(100kbit/s), while for PIC, in addition to calling the *OpenI2C()* function, the Register SSPADD also needs to be set a suitable value to configure the clock speed to use the PIC I2C interface as standard mode. This means, users can using different I2C modes on the PIC by setting different SSPADD values.

### Programmer and Monitoring:

To burn the program into the microcontrollers, PIC needs PICKit2 in circuit debugger, while Arduino needs a serial converter. PICKit2 is larger than a simple serial converter and takes more space. In this case, the Arduino is better than PIC. In addition, Arduino is convenient for monitoring the data and test results because of serial monitor provided Arduino IDE. However, PIC requires extra devices and software to realize monitoring. It requires smart cable to connect it with the computer and software as terminal to receive the information and data. In this case Arduino is better than PIC.

Arduino is convenient, friendly and easy to approach. Above all, for education purpose, Arduino is better than PIC to implement Serial interfaces.

## Section 7: Conclusion

Originally, this project is designed to use the BBC micro:bit and PIC as microcontrollers to implement the serial interfaces. However, because the BBC micro:bit is not available due to the issues of its power supply [32], Arduino is used to substitute the BBC micro:bit to implement the serial interfaces and the project has been changed to select several sensors with SPI or I2C interface and produce suitable material for a Year 1 undergraduate laboratory session that introduces and explains how to use these common serial interfaces on the Arduino. After reviewing the objectives of the project, it can be found that all of the required tasks were mostly completed successfully. The project moved on step by step fast and stably following the time plan (shown in the Appendix B), although there were some difficulties in interfacing different sensors and tricks in building a self-made Arduino.

Basically, this project consists of software development and hardware development. In the software development part, the main issue is that different sensors will have different SPI or I2C commands. This means different sensors will have different structures and formats of reading data and writing data functions. The sensors' datasheet has been read carefully to design the software program code to investigate the principles of how to implement these SPI or I2C commands. Another issue is that the whole process is embedded. When a problem occurs, it spent a long time to debug the code and to find out where went wrong with oscilloscope.

The hardware development part started after the software development part was finished. The design and constructions of the self-made Arduino on the strip board achieved easily. However, the trick was that the bought Atmega328 chip did not have pre-burnt bootloader so that the sketch cannot be uploaded to the self-made Arduino. There are several methods to burn bootloader to the self-made Arduino, but they all require to buy extra devices such as AVR-ISP and USBtinyISP. After doing some research [33], the method, using Arduino Leonardo to burn the bootloader was implemented to solve the problems. After the bootloader was burnt, the self-made Arduino worked properly and its supported functions are as powerful as Arduino Leonardo board.

Finally, lab sessions of using Arduino to communicate with BME280 sensor by both SPI and I2C interfaces is provided for first year undergraduate students in the Appendix A in order to introduce them how to use these two popular low-end communication protocols.

Overall, although this project has a slight change, it still follows the time plan. The aims and objectives of this project are highly achieved. The communication between the microcontrollers and all types of sensors works properly through SPI or I2C interface and UART interface successfully realize the monitoring of the test results. In addition, self-made Arduino works properly.



## Section 8: References

- [1] Total Phase. (2016). *SPI Background*. Available: <http://www.totalphase.com/support/articles/200349236-SPI-Background>. Last accessed 20th Apr 2016.
- [2] Leens, F. An Introduction to I2C and SPI Protocols, *IEEE Instrumentation & Measurement Magazine*, pp. 8-13, February 2009.
- [3] Freescale Semiconductor. (2007). *M68HC11 Reference Manual*. Available: [http://www.nxp.com/files/microcontrollers/doc/ref\\_manual/M68HC11RM.pdf](http://www.nxp.com/files/microcontrollers/doc/ref_manual/M68HC11RM.pdf). Last accessed 20th Apr 2016.
- [4] Freescale Semiconductor. (2004). *SPI Block Guide V04.01*. Available: <http://read.pudn.com/downloads87/ebook/336282/S12SPIV4.pdf>. Last accessed 20th Apr 2016.
- [5] Ermicroblog. (2010). *Using Serial Peripheral Interface (SPI) with Microchip PIC18 Families Microcontroller*. Available: <http://www.ermicro.com/blog/?p=1846>. Last accessed 20th Apr 2016.
- [6] Corelis. (2016). *SPI Interface*. Available: [http://www.corelis.com/education/SPI\\_Tutorial.htm](http://www.corelis.com/education/SPI_Tutorial.htm). Last accessed 20th Apr 2016.
- [7] NXP Semiconductors N.V. (2014). *UM10204 I2C-bus specification and user manual*. Available: [http://www.nxp.com/documents/user\\_manual/UM10204.pdf](http://www.nxp.com/documents/user_manual/UM10204.pdf). Last accessed 20th Apr 2016.
- [8] Total Phase. (2016). *7-bit, 8-bit, and 10-bit I2C Slave Addressing*. Available: <http://www.totalphase.com/support/articles/200349176-7-bit-8-bit-and-10-bit-I2C-Slave-Addressing>. Last accessed 20th Apr 2016.
- [9] Embedded Systems Academy. (2016). *I2C Bus Protocol*. Available: <http://www.esacademy.com/en/library/technical-articles-and-documents/miscellaneous/i2c-bus/general-introduction/i2c-bus-protocol.html>. Last accessed 20th Apr 2016.
- [10] National Instruments. (2015). *RS-232, RS-422, RS-485 Serial Communication General Concepts*. Available: <http://www.ni.com/white-paper/11390/en/>. Last accessed 20th Apr 2016.
- [11] Microchip. (2001). *Using the USART in Asynchronous Mode*. Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/usart.pdf>. Last accessed 20th Apr 2016.

- [12] Wang, Y. and Song, K. (2011). *A New Approach to Realize UART*. Available: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6023602>. Last accessed 20th Apr 2016.
- [13] Electric Imp. (2016). *UART Explained Introduction to the standard serial bus*. Available: <https://electricimp.com/docs/resources/uart/>. Last accessed 20th Apr 2016.
- [14] TEXAS INSTRUMENTS. (2009). ADS8326, 16-Bit, High-Speed, 2.7V to 5.5V microPower Sampling ANALOG-TO-DIGITAL CONVERTER. *Datasheet*, Available: <http://www.ti.com/lit/ds/sbas343c/sbas343c.pdf>. Last accessed 20th Apr 2016.
- [15] Microchip. (2009). MCP4902/4912/4922, 8/10/12-Bit Dual Voltage Output Digital-to-Analog Converter with SPI Interface. *Datasheet*, Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/22250A.pdf>. Last accessed 20th Apr 2016.
- [16] Digilent. (2015). *PmodACL2™ Reference Manual*. Available: [https://reference.digilentinc.com/\\_media/pmod:pmod:pmodACL2\\_rm.pdf](https://reference.digilentinc.com/_media/pmod:pmod:pmodACL2_rm.pdf). Last accessed 20th Apr 2016.
- [17] Freescale Semiconductor. (2013). Xtrinsic MMA8452Q 3-Axis, 12-bit/8-bit Digital Accelerometer. *Datasheet*, Available: <https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/MMA8452Q-rev8.1.pdf>. Last accessed 20th Apr 2016.
- [18] Andrejašić, M. (2008). *MEMS ACCELEROMETERS*. Available: [http://mafija.fmf.uni-lj.si/seminar/files/2007\\_2008/MEMS\\_accelerometers-koncna.pdf](http://mafija.fmf.uni-lj.si/seminar/files/2007_2008/MEMS_accelerometers-koncna.pdf). Last accessed 20th Apr 2016.
- [19] Sparkfun. (2016). *SparkFun Triple Axis Accelerometer Breakout - MMA8452Q*. Available: <https://www.sparkfun.com/products/12756>. Last accessed 20th Apr 2016.
- [20] Bosch Sensortec. (2013). BME280 Combined humidity and pressure sensor. *Datasheet*, Available: [https://cdn.sparkfun.com/assets/learn\\_tutorials/4/1/9/BST-BME280\\_DS001-10.pdf](https://cdn.sparkfun.com/assets/learn_tutorials/4/1/9/BST-BME280_DS001-10.pdf). Last accessed 20th Apr 2016.
- [21] Sparkfun. (2016). *SparkFun Atmospheric Sensor Breakout - BME280* Available: <https://www.sparkfun.com/products/13676>. Last accessed 20th Apr 2016.
- [22] Microchip. (2016). PIC18F2XK20/4XK20 28/40/44-Pin Flash Microcontrollers with XLP Technology. *Datasheet*, Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/40001303H.pdf>. Last accessed 20th Apr 2016.

- [23] Pantech. (2016). *INTRODUCTION OF PICKIT2*. Available: <https://www.pantechsolutions.net/microcontroller-boards/getting-started-with-pickit2>. Last accessed 20th Apr 2016.
- [24] Microchip. (2009). *PIC18F4XK20 Starter Kit User's Guide*. Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/41344C.pdf>. Last accessed 20th Apr 2016.
- [25] Atmel. (2016). ATmega16U4/ATmega32U4 8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller. *Datasheet*, Available: [http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4\\_Datasheet.pdf](http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf). Last accessed 20th Apr 2016.
- [26] Arduino. (2016). *Arduino Leonardo*. Available: <https://www.arduino.cc/en/Main/ArduinoBoardLeonardo>. Last accessed 20th Apr 2016.
- [27] Arduino. (2016). *SPI library*. Available: <https://www.arduino.cc/en/Reference/SPI>. Last accessed 20th Apr 2016.
- [28] FTDI. (2016). TTL-232R TTL to USB Serial Converter Range of Cables. *Datasheet*, Available: <http://www.farnell.com/datasheets/814049.pdf>. Last accessed 20th Apr 2016.
- [29] Ascitable. (2010). *ASCII Table and Description*. Available: <http://www.asciitable.com/>. Last accessed 20th Apr 2016.
- [30] Mellis, D. (2008). *Building an Arduino on a Breadboard*. Available: <https://www.arduino.cc/en/Main/Standalone>. Last accessed 20th Apr 2016.
- [31] Arduino. (2016). *From Arduino to a Microcontroller on a Breadboard*. Available: <https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard> Last accessed 20th Apr 2016.
- [32] Shepherd, A and LEE, A. (2016). *BBC launches microcomputer for schoolchildren after multiple delays*. Available: <http://www.itpro.co.uk/public-sector/24938/bbc-micro-bit-specs-features-and-release-date-bbc-begins-rolling-out-1m-micro#latestnews>. Last accessed 20th Apr 2016.
- [33] Arduino. (2016). *Using an Arduino as an AVR ISP (In-System Programmer)*. Available: <https://www.arduino.cc/en/Tutorial/ArduinoISP>. Last accessed 20th Apr 2016.

## Appendix A: BME 280 SPI and I2C lab sessions

### Background materials:

**BME280 datasheet:** [https://cdn-shop.adafruit.com/datasheets/BST-BME280\\_DS001-10.pdf](https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf)

#### Arduino board:

It is suggested to use Arduino Leonardo or Arduino Uno as microcontrollers to be the masters.

Leonardo: <https://www.arduino.cc/en/Main/arduinoBoardLeonardo>

UNO: <https://www.arduino.cc/en/Main/ArduinoBoardUno?setlang=Eng>

### 9.1 Introduction:

#### 9.1.1 Aim:

The aim of this lab is to introduce the SPI and I2C interfaces and use these two interfaces to establish the communication between the Arduino and the BME280 breakout board produced by the Sparkfun separately.

#### 9.1.2 Objectives:

1. Obtain the Humidity, temperature and barometric pressure data through the SPI interface and display them on the serial monitor of the Arduino.
2. Obtain the Humidity, temperature and barometric pressure data through the I2C interface and display them on the serial monitor of the Arduino.

#### 9.1.3 Arduino serial monitor:

Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART).

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board.

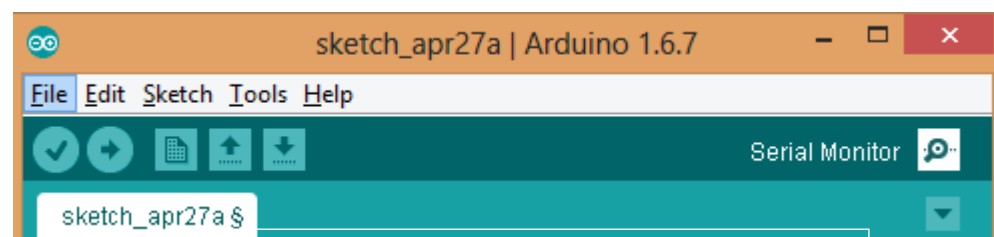


Figure A: Serial Monitor

### 9.1.4 How to use the serial monitor?

Serial Monitor is used to debug Arduino Software Sketches or to read data sent by a working Sketch program. Arduino must be connected by USB with your computer to be able to activate the Serial Monitor.

To test serial monitor, please try the following codes and find their functions.

```
void setup() {  
  // put your setup code here, to run once:  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println("Hello world");  
  Serial.print(55, DEC);  
  Serial.println(55, HEX);  
  Serial.println(55.55);  
}
```



Figure B: Test code segments

Upload this sketch and open the serial monitor to see what will happen. Record the screen shot of the serial monitor.

Q1: What is the function of *Serial.begin(9600)*? What does 9600 mean?

Q2: What are the function of *Serial.print()* and *Serial.println()*?

**Hint:** <https://www.arduino.cc/en/Reference/Serial>

## 9.2 BME280 SPI interface design:

### 9.2.1 Arduino pins configuration:

Find out MOSI, MISO, SCK pins on the Arduino. Record them in the report.

Choose the SS/CS pin of the SPI interface and configure it as digital output:

Connect the Arduino with BME280 through SPI interface.

SPI interface connection:

1. Connect BME280 Vin to the power supply, 3V or 5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
2. Connect BME280 GND to Arduino common power/data ground
3. Connect the BME280 SCK pin to Arduino SCK pin
4. Connect the BME280 SDO pin to Arduino MISO pin
5. Connect the BME280 SDI pin to Arduino MOSI pin
6. Connect the BME280 CS pin to Arduino CS pin but any pin can be used later

**Hint:** *pinMode()* may be used to configure the pins.

### 9.2.2 Arduino SPI library:

In Arduino, <spi.h> library is provided for users to implement spi interface. The following table shows the main functions of the library. Do not forget to include the <spi.h> in your code.

Table A: SPI functions

Function	Description
SPI.begin	Initializes the SPI bus.
SPI.beginTransaction	Initializes the SPI bus using the defined SPISettings.
SPI.Settings	configure the SPI port for your SPI device
SPI.end	Disables the SPI bus (leaving pin modes unchanged).
SPI.transfer	send a byte to the SPI bus and receive a byte from the SPI bus

*Hint: Go to <https://www.arduino.cc/en/reference/SPI> to find more details of these functions.*

### 9.2.3 Set the SPI clock frequency:

Q3: Try to find out the code to open the SPI interface with setting the Arduino as master. Set the SPI clock frequency to be 2MHz and configure the SPI interface to be MODE0 (CPOL = CPHA = '0'). Show the code segments in the report.

*Hint: SPI.begin(), SPI.beginTransaction() and SPI.Settings() may be used.*

### 9.2.4 Design two functions to implement BME280 SPI interface read command and write command:

#### 9.2.4.1 Write command:

Please read the BME280 datasheet Section 6.3.1 carefully and design a function to implement Write command for BME280.

*Hint: The pseudo code of Write command is following:*

*Change the address byte bit 7 to 0*

*Put CS down*

*Write address byte (bit 7 is zero) to the BME280*

*Write data byte to the BME280*

*Put CS up*

#### 9.2.4.2 Read command:

Please read the BME280 datasheet Section 6.3.2 carefully and design a function to implement Read command for BME280.

*Hint: The pseudo code of Read command is following:*

*Put CS down*

*Write address byte (bit 7 is one) to the BME280*

*Read data byte from the BME280 and store it in an unsigned char x;*

*Put CS up*

*Return x*

**9.2.5 Test the SPI interface:**

Reset the BME280 by sending 0xB6 to Register 0xE0;

Read the BME280 device ID through reading Register 0xE0 and display the device ID number through the serial monitor. Record the screenshot of the serial monitor.

Use oscilloscope to monitor the SCK pin and check the clock frequency (it should be 2MHz) Record the figures of SCK (Channel 1) with MISO (Channel 2) and SCK(Channel 1) with MOSI(Channel 2).

Q5: Evaluate your codes and illustrate the figures and discuss your SPI interface.

**9.3 BME280 I2C interface design:****9.3.1 Arduino pins configuration:**

Find out SDA and SCL pins on the Arduino. Record them in the report.

Connect the Arduino with BME280 through I2C interface.

I2C interface connection:

- Connect **Vin** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect BME280 **GND** to Arduino common power/data ground
- Connect the BME280 **SCL** pin to the I2C clock **SCL** pin on your Arduino.
- Connect the BME280 **SDA** pin to the I2C data **SDA** pin on your Arduino.

**9.3.2 Arduino I2C library:**

<wire.h> is an I2C peripheral library provided by Arduino. The following table introduces the functions of the I2C library used in this lab.

Table B:I2C functions

Function	Description
Wire.begin	Initializes and start the I2C interface as master or slave.
Wire.beginTransmission	Begin a transmission to the I2C slave device with the given address. Subsequently, queue bytes for transmission with the <i>Wire.write()</i> function and transmit them by calling <i>Wire.endTransmission()</i> .
Wire.endTransmission	Ends a transmission to a slave device that was begun by <i>Wire.beginTransmission()</i> and transmits the bytes that were queued by <i>Wire.write()</i> .
Wire.requestFrom	Used by the master to request bytes from a slave device.
Wire.write	send a byte to the I2C bus
Wire.read	Read a byte from the I2C bus

**Hint:** Go to <https://www.arduino.cc/en/reference/wire> to find more details of these functions.

*In Wire.beginTransmission(address) the arguments of the address is 7-bit address. If an 8-bit address is passed, the first MSB will be ignored. This will generate a START condition to the I2C bus. If Wire.endTransmission() is called, a STOP condition will be generated.*

**9.3.3 Start the I2C transmission**

Q6: Try to find out the code to open the I2C interface with setting the Arduino as master. Show the code segments in the report. State your clock frequency of I2C interface.

**Hint:** *Wire.begin();*

Q7: What is the slave 7-bit address of the BME280 breakout board?

**Hint:** *datasheet section 6*

### **9.3.4 Design two functions to implement BME280 I2C interface read command and write command:**

#### **9.3.4.1 Write command:**

Please read the BME280 datasheet Section 6.2.1 carefully and design a function to implement Write command for BME280.

**Hint:** *The pseudo code of Write command is following:*

*Generate a START condition*

*Select slave by writing the 7-bit address to the I2C bus to operate the Write command*

*Write an address register byte to the I2C bus*

*Write a data byte to the I2C bus*

*Generate a STOP condition*

---

#### **9.3.4.2 Read command:**

Please read the BME280 datasheet Section 6.2.2 carefully and design a function to implement Read command for BME280.

**Hint:** *The pseudo code of Read command is following:*

*Generate a START condition*

*Select slave by writing the 7-bit address(0x77) to the I2C bus to operate the Write command*

*Write an address register byte to the I2C bus*

*Generate a START/RESTART condition again*

*Request bytes from the sensor by writing the device address to the I2C bus to operate the Read command*

*Generate a STOP condition*

*Read a data byte from the I2C bus and store in a variable x;*

*Return x*

---

**Important notes ! ! ! : *Wire.requestFrom (address, number of bytes); should be used to do the I2C interface read command. First the Wire.requestFrom(address, number of bytes) will generate a START/RESTART condition to the I2C bus. And it will request bytes from a slave device. The bytes will be retrieved with the Wire.available()***



*and Wire.read() functions. Finally, after the request, a STOP condition will be generated and sent to the I2C bus. In this Wire.requestFrom (address, number of bytes), the Parameters address is the 7-bit address of the device to request bytes from (you do not need to add RW bit manually. This is slightly different to the Figure 10 in the BME280 datasheet section 6.2.2)*

### 9.3.5 Test the I2C interface:

Reset the BME280 by sending 0xB6 to Register 0xE0;

Read the BME280 device ID through reading Register 0xE0 and display the device ID number through the serial monitor. Record the screenshot of the serial monitor.

Use oscilloscope to monitor the SCL pin and check the clock frequency (it should be 100 KHz)

Record the figures of SCL (Channel 1) with SDA (Channel 2).

Q8: Evaluate your codes and illustrate the figures and discuss your I2C interface.

## 9.4 Obtain Temperature, pressure and humidity data from the BME280 sensor through.

In this coursework, the BME280 is configured to be Forced mode. Please read BME280 datasheet section 3.3.3 carefully and read section 5.4.5 carefully to find out how to configure the BME280 to be Forced mode.

Q9: How to configure BME280 to be forced mode?

*Hint: ctrl\_meas Register*

### 9.4.1 Temperature:

#### 9.4.1.1 Temperature measurement:

The temperature measurement can be enabled or skipped. Please read BME280 datasheet section 3.4.3 carefully. Set the temperature measurement to be enabled.

Q10: How to configure temperature measurement to be enabled?

*Hint: ctrl\_meas Register*

#### 9.4.1.2 Design temperature compensations function:

Please read BME280 datasheet section 4 carefully and find out why do we need output compensations?

Q11: Why do we need output compensation?

Find the temperature compensation formula in 32 bit fixed format and find out the compensation related value (register address) we need to read from the BME280 sensor.

Q12: Which Register address values are used for temperature compensation formula?

**Hint: *dig\_T and adc\_T***

Read this related compensation values through the SPI or I2C interfaces and design the compensation function to obtain the temperature data and display them on the serial monitor.

Q13: Show the designed function in your report and record the screenshot of the serial monitor

Q14: Use your hands to warm the sensors. Observe the changes. Record and discuss the results.

## **9.4.2 Pressure:**

### **9.4.2.1 Pressure measurement:**

The pressure measurement can be enabled or skipped. Please read BME280 datasheet section 3.4.2 carefully. Set the pressure measurement to be enabled.

Q15: How to configure pressure measurement to be enabled?

**Hint: *ctrl\_meas Register***

### **9.4.2.2 Design pressure compensations function:**

Find the pressure compensation formula in 32 bit fixed format and find out the compensation related value (register address) which we need to read from the BME280 sensor.

Q16: Which Register address values are used for humidity compensation formula?

**Hint: *dig\_P and adc\_P***

Read this related compensation values through the SPI or I2C interfaces and design the compensation function to obtain the pressure data and display them on the serial monitor.

Q17: Show the designed function in your report and record the screenshot of the serial monitor

Q18: Move the sensor up and down. Observe the changes. Record and discuss the results.

### 9.4.3 Humidity:

#### 9.4.3.1 Humidity measurement:

The humidity measurement can be enabled or skipped. Please read BME280 datasheet section 3.4.1 carefully. Set the humidity measurement to be enabled.

Q19: How to configure pressure measurement to be enabled?

**Hint:** *ctrl\_Hum Register*

#### 9.4.3.2 Design pressure compensations function:

Find the pressure compensation formula in 32 bit fixed format and find out the compensation related value (register address) which we need to read from the BME280 sensor.

Q20: Which Register address values are used for humidity compensation formula?

**Hint:** *dig\_H and adc\_H*

Read this related compensation values through the SPI or I2C interfaces and design the compensation function to obtain the pressure data and display them on the serial monitor.

Q21: Show the designed function in your report and record the screenshot of the serial monitor

Q22: Blow to the BME280 sensor. Observe the changes. Record and discuss the results.

### 9.4.4 Final Results:

**Read temperature, pressure, and humidity data from the BME280 sensor through SPI and I2C interfaces. Display them together on the serial monitor. Record and discuss the results. Draw the flowcharts of using SPI interface to obtain these data and using I2C interface to obtain these data. Compare the SPI and I2C interfaces in your report**

## Appendix B: Time plan

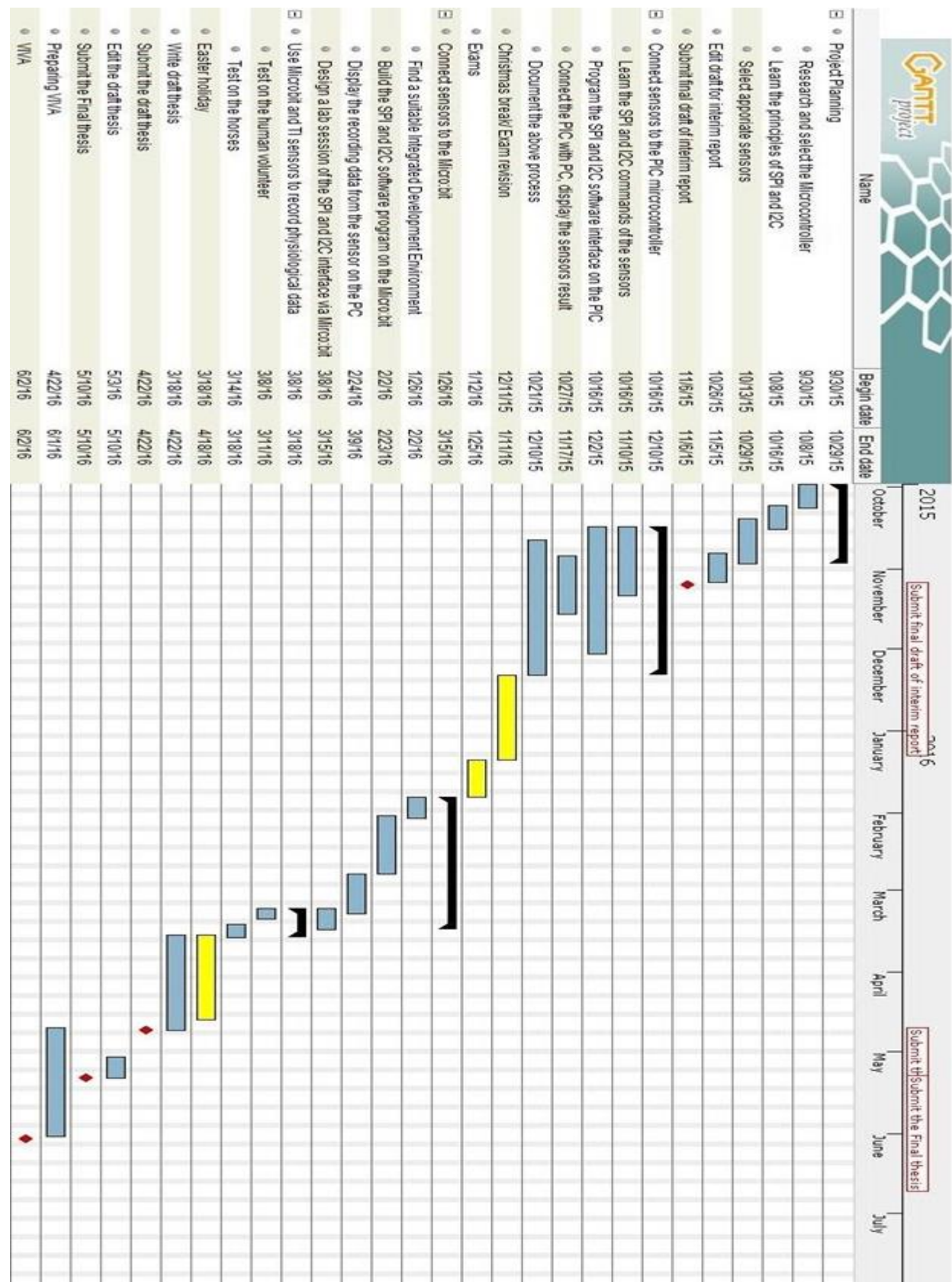


Figure C: Time plan