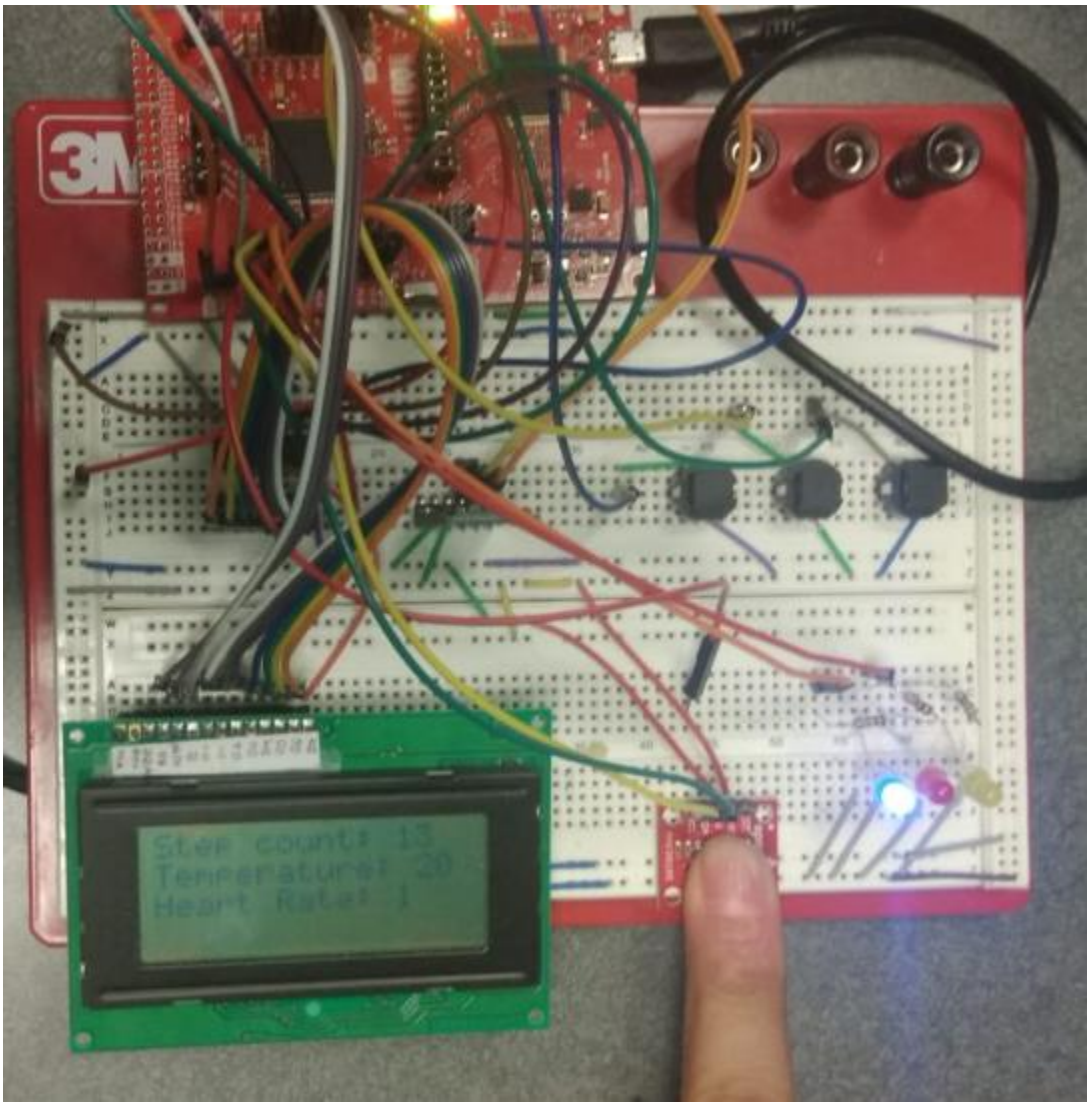
FITNESS TRACKER



Vikrant Waje
Final Project Report
ECEN 5613 Embedded System Design
December 9, 2018

Table of Contents

1 INTRODUCTION **4**

1.1 SYSTEM OVERVIEW 4

2 TECHNICAL DESCRIPTION **5**

2.1 **BOARD DESIGN** 5

2.2 **NEW HARDWARE COMPONENT #1: HEARTBEAT SENSOR** 7

2.2.1 GENERAL DESCRIPTION 7

2.2.2 JUSTIFICATION OF CHOICE OF SENSOR 7

2.2.3 ARCHITECTURE OF SENSOR 7

2.2.4 PRINCIPLE OF OPERATION 7

2.2.5 ALGORITHM USED TO DETECT HEART RATE IN BPM 8

2.2.6 CONNECTION WITH MICROCONTROLLER SYSTEM 9

2.3 **NEW HARDWARE COMPONENT #2: PEDOMETER** **11**

2.3.1 GENERAL DESCRIPTION 11

2.3.2 JUSTIFICATION OF CHOICE OF SENSOR 11

2.3.3 ARCHITECTURE OF SENSOR 11

2.3.4 PRINCIPLE OF OPERATION 12

2.3.5 ALGORITHM USED TO COUNT NUMBER OF STEPS 12

2.3.6 CONNECTION WITH MICROCONTROLLER SYSTEM 13

2.4 **NEW HARDWARE COMPONENT #3: SHIFT REGISTER** **14**

2.4.1 GENERAL DESCRIPTION 14

2.4.2 JUSTIFICATION OF CHOICE OF SENSOR 14

2.4.3 ARCHITECTURE OF SENSOR 14

2.4.4 PRINCIPLE OF OPERATION 15

2.4.5 ALGORITHM USED TO CONVERT SERIAL TO PARALLEL 15

2.4.6 CONNECTION WITH MICROCONTROLLER SYSTEM 16

2.5 **NEW HARDWARE COMPONENT #4: BLUETOOTH MODULE** **17**

2.5.1 GENERAL DESCRIPTION 17

2.5.2 JUSTIFICATION OF CHOICE OF SENSOR 17

2.5.3 ARCHITECTURE OF SENSOR 17

2.5.4 PRINCIPLE OF OPERATION 18

2.5.5 CONNECTION WITH MICROCONTROLLER SYSTEM 18

2.6 **FIRMWARE DESIGN** **19**

2.6.1 SPI DRIVER 20

2.6.2 I2C DRIVER 20

2.6.3 UART DRIVER 21
2.6.4 REAL TIME CLOCK DRIVER 21
2.6.5 SHIFT REGISTER DRIVER 22
2.6.6 CIRCULAR BUFFER DRIVER 23
2.6.7 CLOCK DRIVER 24
2.6.8 SYSTICK TIMER DRIVER 24
2.6.9 HEARTBEAT/PULSERATE DRIVER 24
2.6.10 PEDOMETER DRIVER 25
2.6.11 LCD DRIVER 25
2.7 SOFTWARE DESIGN 27
2.8 TESTING PROCESS 29
3 RESULT AND ERROR ANALYSIS.....34
3.1 COMPARISON OF COMMUNICATION PROTOCOL USED
3.1.1 COMPARING TIME OF SINGLE BYTE TRANSFER 34
3.1.2 COMPARING POWER OF SINGLE BYTE TRANSFER 34
3.1.3 COMPARING ENERGY OF SINGLE BYTE TRANSFER 35
3.2 PERFORMANCE METRICS 35
3.2.1 RESPONSE TIME 35
3.2.2 MEMORY LOADING 35
3.3 LESSONS LEARNT 38
4 CONCLUSION **38**
5 FUTUTRE DEVELOPMENT IDEAS **39**
6 ACKNOWLEDGEMENTS **40**
7 REFERENCES..... **41**
8 APPENDICES..... **42**
8.1 Appendix - Bill of Materials 42
8.2 Appendix - Schematics 42
8.3 Appendix - Firmware Source Code 42
8.4 Appendix - Software Source Code 42
8.5 Appendix - Data Sheets and Application Notes.....42

CHAPTER1 INTRODUCTION

Being a fitness freak and an athlete, I have spent countless hours on training in the field. For an athlete, it is necessary to have some kind of measure how much work or energy he has spent, and gain some understanding of how their body responds to various types of exercise. This idea led me to build a project based on fitness tracker that could measure two or three parameters that can help judge an individual about how well his body reacts to various type of exercise.

Although, a more sophisticated and well-built design (such as Fitbit) is already available in the market, this project was designed with a rationale to get a thorough understanding of how actually various types of sensors are interfaced to the main microcontroller, and how complex the algorithm needs to be in order to measure those physical phenomenon (such as temperature, heartbeat, steps, etc.)

With this, I have analyzed how the system that I have designed performs vs the performance of system that is already available in market. This helped to get knowledge of what improvements I need to make if I were to design the same system that would be sold in market.

1.1 System Overview

Since my goal was to design a fitness tracker, I thought it would be a good idea to integrate some more functionality which would be used by an athlete when he is not exercising, and which would be helpful to him in day to day life. This inspired me to have a system with a real time clock running on the fitness tracker system under normal condition. Only when the athlete/ user wants to measure the body associated parameters, the actual sensor values would be read and displayed on the output device. This design incorporates several new components spanning hardware, firmware, and software. It is inspired by the Fitbit but does not try to recreate it.

Overall, the system hosts a microcontroller based on ARM cortex M4 architecture (TI MSP432) which is connected to Heart beat sensor and temperature sensor using I2C interface, Pedometer using SPI interface, and Bluetooth module interfaced using UART interface. There are two modes of operation namely timing mode and sensor mode. In Timing mode, Date , day, and time is displayed on LCD, while in sensor mode sensor values are displayed on LCD as well on Bluetooth based Android application.

.

CHAPTER 2 TECHNICAL DESCRIPTION

The following sections detail the design and implementation of the Fitness Tracker. The sections are outlined as follows: Board Design includes all new hardware components such as shift register, Bluetooth module, Heartbeat sensor, and Pedometer
Firmware Design covers SPI driver, I2C driver, Bluetooth driver(UART), Shift register driver, Real time Clock peripheral driver, GPIO interrupt setup, and Software Design includes the Android application which had Bluetooth connection interface.

2.1 Board Design

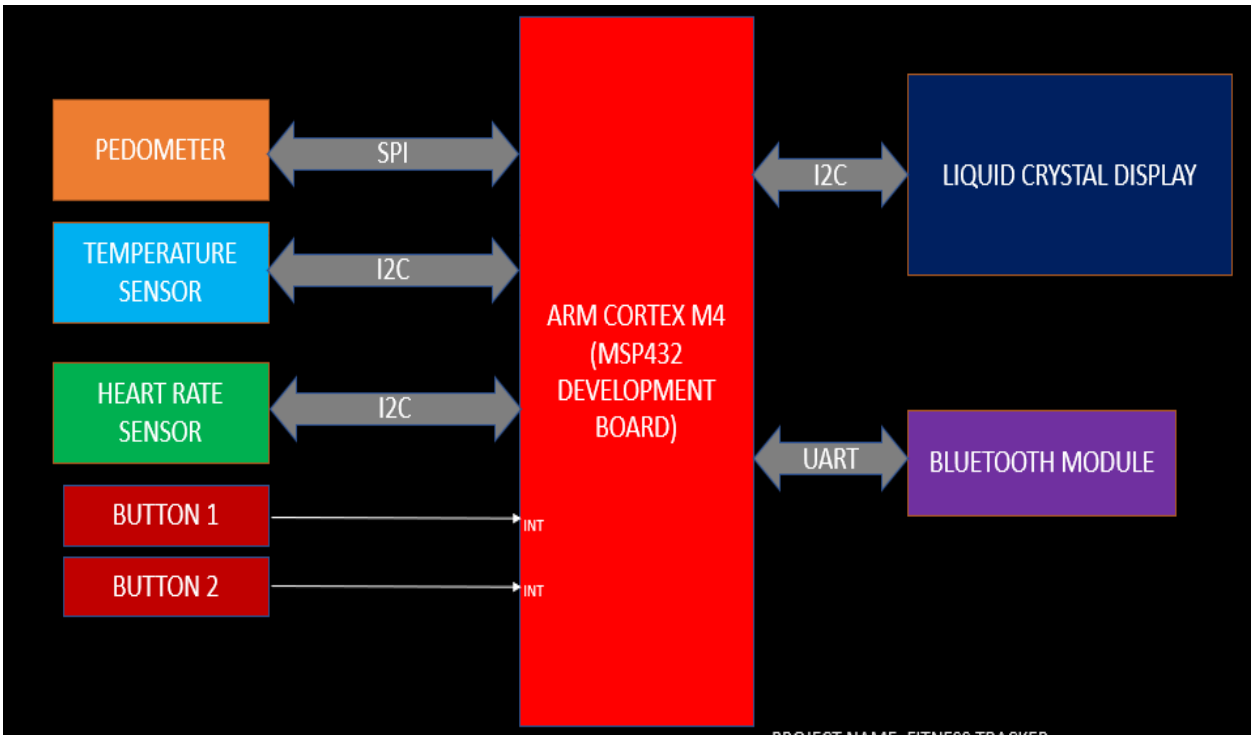


Figure 1 :Block diagram

The system consists following:

Microcontroller unit:

The main microcontroller used is ARM Cortex M4 which is hosted on the TI MSP432 kit. The MSP432 is capable to run at maximum speed of 48MHz with default DCO clock frequency of 3MHz.

Sensors and interface:

1) Pulse Oximeter Sensor: A Pulse oximeter sensor that measures heartbeat is interfaced to the MSP432 using I2C sensor. For this, I used the MAX30105 sensor manufactured by Maxim Integrated corporation.
   There is temperature sensor which is present on the Pulse-oximeter sensor that is used to measure the body temperature
2) Pedometer sensor: A Pedometer that counts number of steps taken by athlete which is interfaced using SPI interface. The pedometer sensing unit was inbuilt in Accelerometer sensor LSM6DS3 manufactured by STMicroelectronics.

Note: All the sensors were purchased as Break-out boards from Sparkfun Electronics.

Output and display device:

1) LCD: A Dot-Matrix LCD controller driver was used to display the time and sensor values.
2) Serial in Parallel Out Shift Register: This integrated circuit was used to convert the serial data sent by the TI MSP432 and converted into parallel form which was then given as 8-bit data to the LCD. The reason behind using this was to save the number of IO lines.

User buttons , LED's and miscellaneous parts:

1) Buttons: Two buttons were used in the entire system. The function of Button 1 was to switch from clock mode to sensor mode while the purpose of button 2 was to switch from sensor mode to clock mode.
2) LED's: This was to just indicate whether the button was pressed or not. Note that there were two LED's used, one for each button.
3) 10k potentiometer: The potentiometer was used to vary the contrast signal of LCD on Vee pin.

Radio module:

1) Bluetooth HC-05: The Bluetooth module is used to transmit data to an Android based application running on mobile phone. The data to be transmitted is the Heartbeat, Steps and temperature value read from the sensor.

2.2 NEW HARDWARE COMPONENT #1 DESCRIPTION: HEART BEAT SENSOR

2.2.1 General Description

Pulse rate sensor MAX30105 is an integrated particle-sensing module. It consists internal LED's, photodetectors, optical elements, and low-noise electronics with ambient light rejection. Due to its relatively small size, it is used in wearable technology.

2.2.2 Justification of choice of sensor

Reason 1: Highly sensitive optical reflection which seems suitable to detect heartbeat accurately.
Reason 2: It consumes very low power due to its programmable LED current and sample rate.
Reason 3: It can be interfaced with widely used communication protocol named I2C .
Reason 4: Capable of operating at 3.3 V which allows me to have only single power rail.

2.2.3 Architecture of sensor

The MAX30105 has three on board LED's. The datasheet specifies that the LED's should be powered from 5V, but it also works fine at 3.3V. The green LED may need a voltage of 3.5V for its operation, but since we only need IR LED's in our use case, we don't need to worry about powering up the green LED.

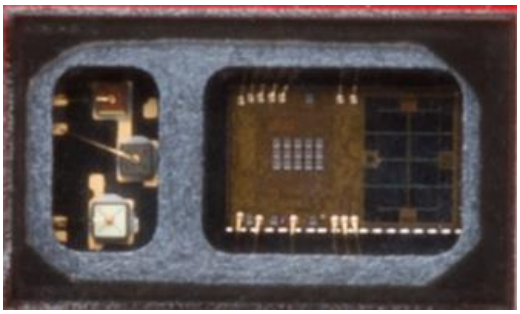


Figure 2: Architecture of Heart rate sensor

The IC itself runs at voltage of 1.8V and can be interfaced using I2C protocol. To interface this sensor to microcontroller, we only need SCL, SDA, power supply lines from the sensor thus making it easy for us while debugging.

2.2.4 Principle of operation

The heart beat sensor which is essentially a pulse oximeter has a LED on one side and photo detector on other side. Light is emitted from one side of the finger which travels through tissue, venous blood and arterial blood and is received/collected by the detector. Most proportion of light is scattered or absorbed before it reaches the photo detector on other side of finger. The flow of blood which is essentially a function of heartbeat is induced, which is pulsatile in nature so the transmitted light changes with time.

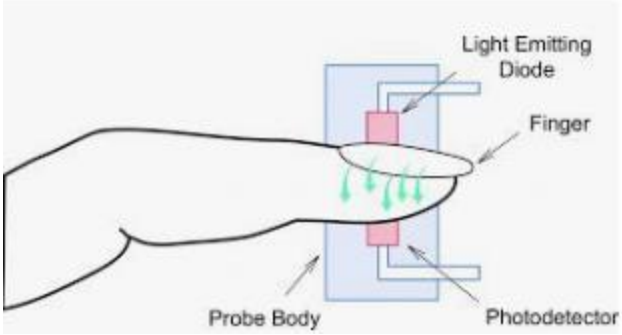


Figure 3: Operation of Heart Rate sensor

The changing/pulsating signal which is basically an AC signal is superimposed on DC signal representing absorption in other substance like pigmentation in tissue, venous, capillary bone.

2.2.5 Algorithm used to detect Heart Rate in bpm

The algorithm consists following steps:

1) Signal differentiation: It brings signal's average close to zero and get rid of signal's baseline rising up or falling down that occur. After this step, we can easily compare height of the peaks.
2) Collecting set of k highest peaks of signal. For 10 seconds measurement, we consider n equals 20.
3) Calculating variances of distances between adjacent peaks for each set of peaks collected on previous step.
4) Choosing appropriate set of peaks: In this step we chose set of peaks, whose variance value calculated in previous step, is minimum among other variances.
5) Heart Rate Calculation: We calculate mean value of distances between adjacent peaks for the set chosen on previous step. Frame rate of measurement is multiplied by 60 and divided by this mean value which gives heart rate in bpm for current measurement.

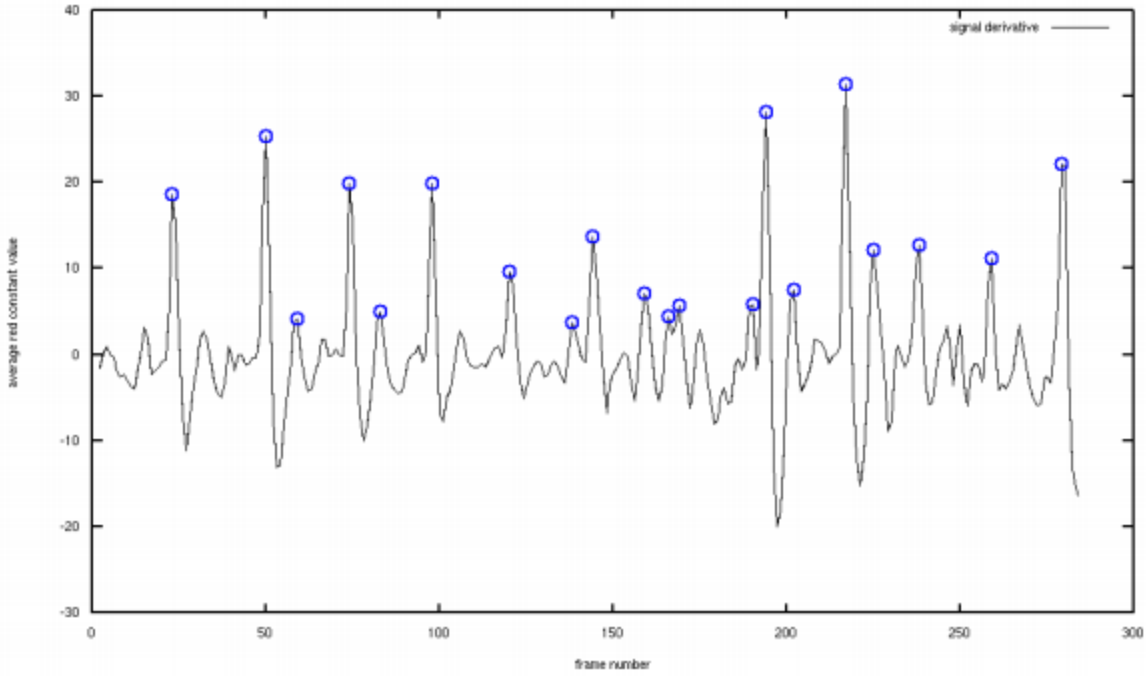


Figure 4: Peaks of Heart beat used for calculation

2.2.6 Connection with microcontroller system

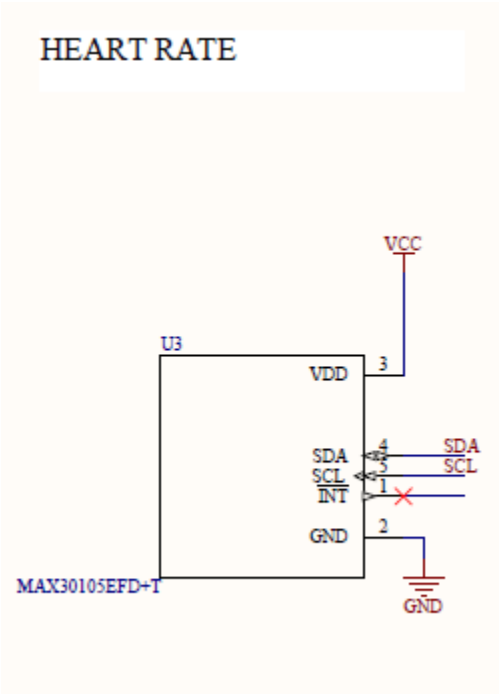


Figure 5: Connection of MAX30105 with MSP432

SDA: Serial Data is connected to P3.6 of MSP432.
SCL: Serial Clock is connected to P3.7 of MSP432.
INT: Interrupt line is not connected and left floating
VCC and GND: This are connected to 3.3V and GND respectively.

Note: Pull up resistors are not required since they were already present on the Sparkfun breakout board.

2.3 NEW HARDWARE COMPONENT #2 DESCRIPTION: PEDOMETER

2.3.1 General Description

The LSM6DS3 is an accelerometer and gyroscope sensor with a giant 8KB buffer specifically targeted at the cellphone market. It is capable of detecting shocks, tilt, motion, taps as well count number of steps(Pedometer).

2.3.2 Justification of choice of sensor

Reason 1: This sensor has an inbuilt mechanism to count steps(Pedometer function)
Reason 2: It consumes very low power as it operates at 1.25mA for upto 1.7 ksps.
Reason 3: It can be interfaced with widely used communication protocol named I2C/SPI .
Reason 4: Capable of operating at 3.3 V which allows me to have only single power rail.

2.3.3 Architecture of sensor

The LSM6D3 is an accelerometer as well as gyroscope with an addition of inbuilt mechanism to count number of steps taken by the user. The IC runs at voltage of 3.3V and has the option of being interfaced using SPI/I2C interface. Since, we are interfacing the sensor using SPI protocol, we need only four lines namely MOSI, MISO, SCL and CS.

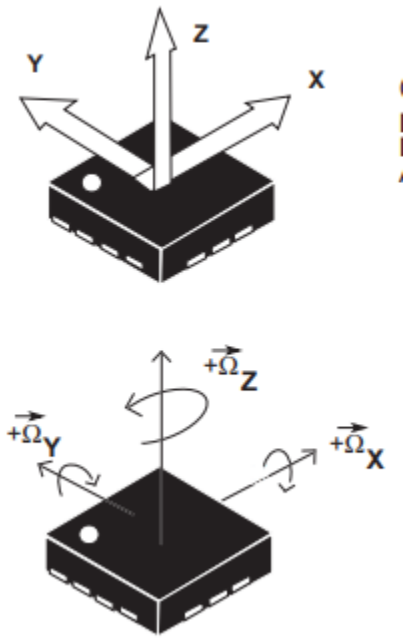


Figure 6: Architecture of Pedometer

2.3.4 Principle of operation

The pedometer counts the number of steps taken by user. Usually, the idea behind pedometer is a circuit which has a fixed metal contact and a swinging end. Under normal condition, the swinging end and metal contact are away from each other as shown in the figure forming an open circuit until there is no motion/tilt. Since the circuit is open, no current flows through it.

When a user takes a step, the sensor gets tilted which causes the floating(swinging) end to touch the metal contact and complete the circuit, allowing the current to flow. The flow of the current energizes the circuit and adds one to step count. As a step is completed, the floating end swings back again, away from the metal circuit which causes open circuit and no current flows, effectively resetting the pedometer for next step.

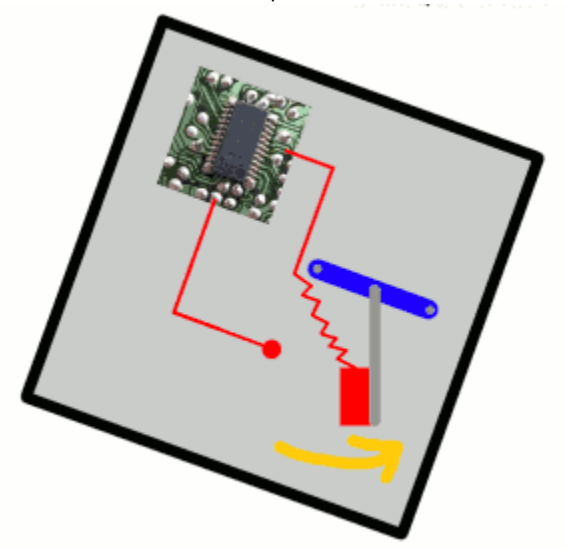


Figure 7: Operation of Pedometer

Note that since the counting of steps in pedometer is based on movement, it is just an approximate reading and does not gives an accurate one. For example, if there jolt in road as you drive the car, it will be counted as step which is incorrect.

2.3.5 Algorithm used to count number of steps

There is no algorithm required to count the number of steps since the LSM6DS3 sensor has predefined register for pedometer that stores the step count when user takes any step. So all you have to do is configure the pedometer using the LSM6DS3_ACC_GYRO_CTRL1_XL register and then enabling the pedometer using the LSM6DS3_ACC_GYRO_TAP_CFG1 register

2.3.6 Connection with microcontroller system

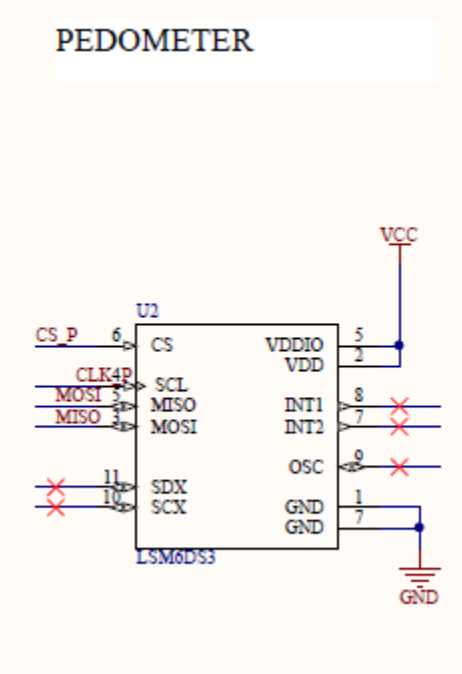


Figure 8: Connection of LSM6DS3 with MSP432

CS: Chip Select is connected to P2.3 of MSP432.
SCL: Serial Clock is connected to P1.5 of MSP432.
MOSI: MOSI is connected to P1.6 of MSP432
MISO: MISO is connected to P1.7 of MSP432
INT1,INT2, SDX, SCX, OCS: This line are not connected and left floating.
VCC and GND: This are connected to 3.3V and GND respectively.

2.4 NEW HARDWARE COMPONENT #3 DESCRIPTION: SHIFTREGISTER(SERIAL TO PARALLEL CONVERTER)

2.4.1 General Description

The 74HC595 is a shift register integrated circuit responsible for converting serial input into parallel output. Shift registers are often used for the purpose of saving pins on a microcontroller. Every microcontroller has limited number of pins for GPIO. If we have to interface a device that needs 8 bit parallel data, we need 8 GPIO lines. In order to save the number of lines, we use a shift register to convert the serial data into parallel form.

2.4.2 Justification of choice of hardware

Reason 1: Since I am using LCD which requires 8 bit parallel data, a shift register that can output 8 bit data was needed. 74HC595 perfectly met that use case
Reason 2: 74HC595 is easier to interface/program and less costly.

2.4.3 Architecture of 74HC595

The 74HC595 consists of 8 D flipflop connected in cascaded fashion as shown in diagram below. The

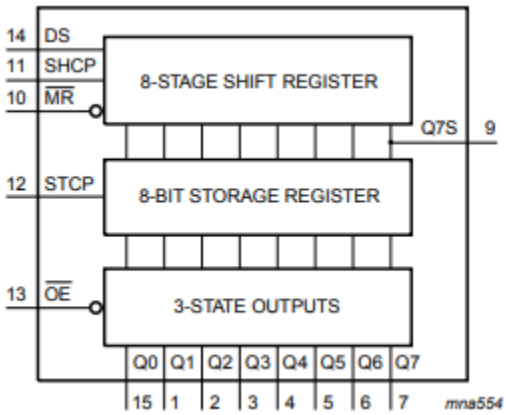


Figure 9: Architecture of ashift register 74HC595

The architecture shows 4 control lines namely OE(output enable), RCLK/STCP(strobe signal),SRCLR/MR(serial clear), SRCLK/SHCP(serial clock),Data input SER/DS(serial input),Data output Qa-Qh/Q0-Q7(8 bit data output).

2.4.4 Principle of operation

With every clock cycle of SHCP, data gets shifted in each of the flipflop(each of the 8 stages) within the shift register. When eight clock cycles of SHCP are done, the 8 bit data gets loaded into the 8 bit shift register. Note that Output enable signal(OE) should be active during that period.

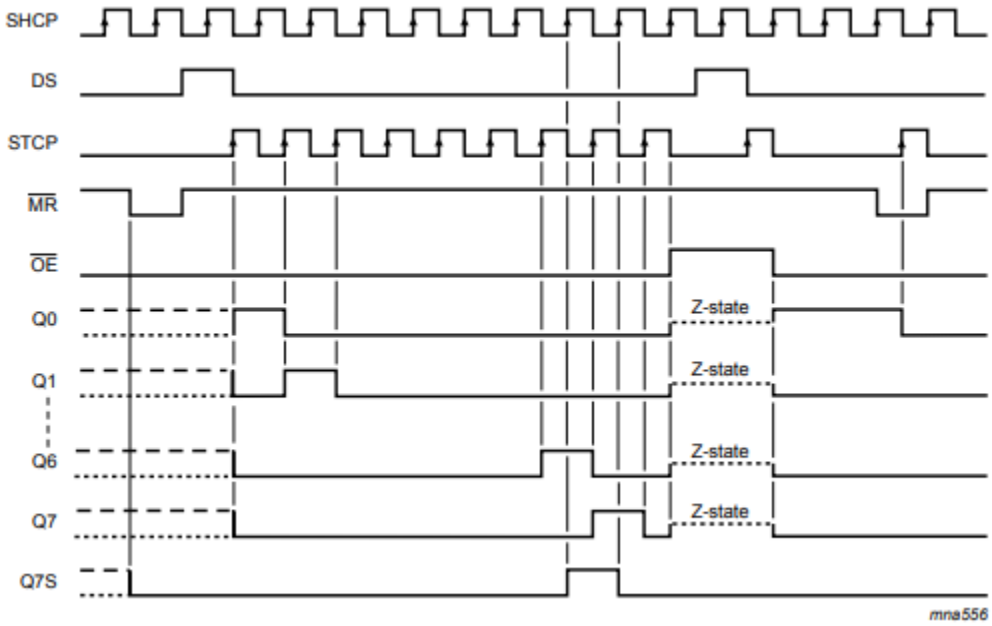


Figure 10: Operation of shift register 74HC595

When the data needs to be shifted out of the IC in parallel form, a pulse strobe(RCLK/STCP) needs to be applied. To clear the shift register, Master clear signal needs to be asserted.

2.4.5 Algorithm used to convert serial data into parallel form

The following sequence needs to be followed in order to convert serial data into parallel form:

1) For every clock pulse applied, put the data on serial data line
2) The 8 bits which are input serially gets stored into the 8 bit storage register.
3) To get the parallel data, we need to apply strobe signal
4) The sequence is followed repeatedly till the time we want parallel data.
5) To clear the shift register, we need to apply master clear signal.

2.4.6 Connection with microcontroller system

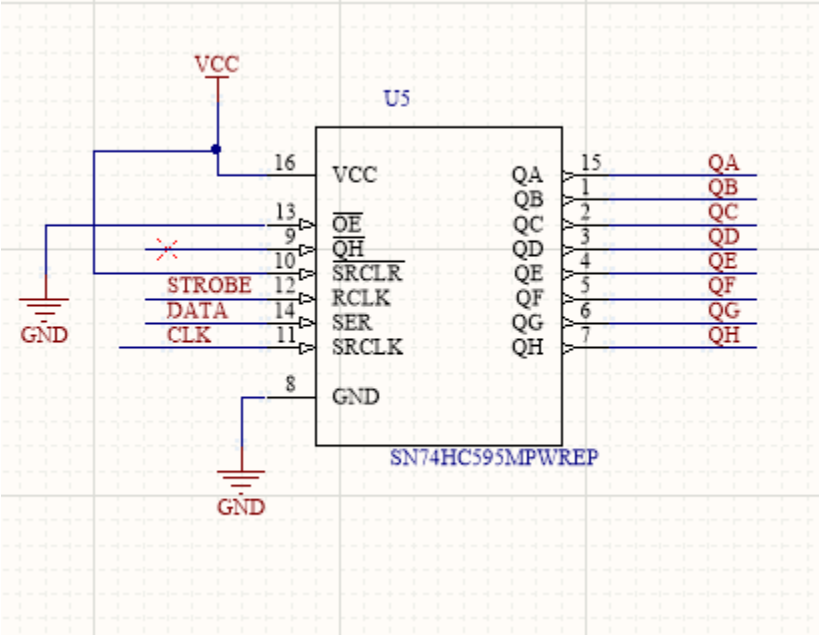


Figure 11: Connection of shift register with MSP432

QA-QH: 8 bit Data output lines connected to LCD
VCC and GND: Power supply connection of the IC
SRCLR: Serial clear control signal to clear the shift register
SER: Serial input data line
SRCLK: Store register clock(Needs 8 clock pulses to shift the data)
RCLK: Strobe signal to clock out the data
OE: Output enable signal connected to ground

2.5 NEW HARDWARE COMPONENT #4 DESCRIPTION: BLUETOOTH CLASSIC MODULE HC-05

2.5.1 General Description

The HC-05 is a Bluetooth module which supports serial protocol designed for transparent wireless connection setup. It provides two modes namely master and slave mode. It is TTL compatible and can connect to laptop/mobile phone without much issues

2.5.2 Justification of choice of hardware

Reason 1: Since it communicates with microcontroller using UART communication, and I had already developed UART driver, it was kind of easy to interface to this sensor.
Reason 2: You don't need any level converter for TXD and RXD lines since they support voltage of 3.3V which makes it very easy to interface
Reason 3: It supports range around 60 – 70 m and consumes very little current(about 30 mA).

2.5.3 Architecture of 74HC595

As seen in the diagram, the HC-05 module has two power supply pins namely VCC and GND. It has TXD and RXD pin which is interfaced to microcontroller to support UART communication. STATE pin is an input signal, if it low or connect to air, the module is at paired or communication mode. If it is high, the module will enter AT mode.

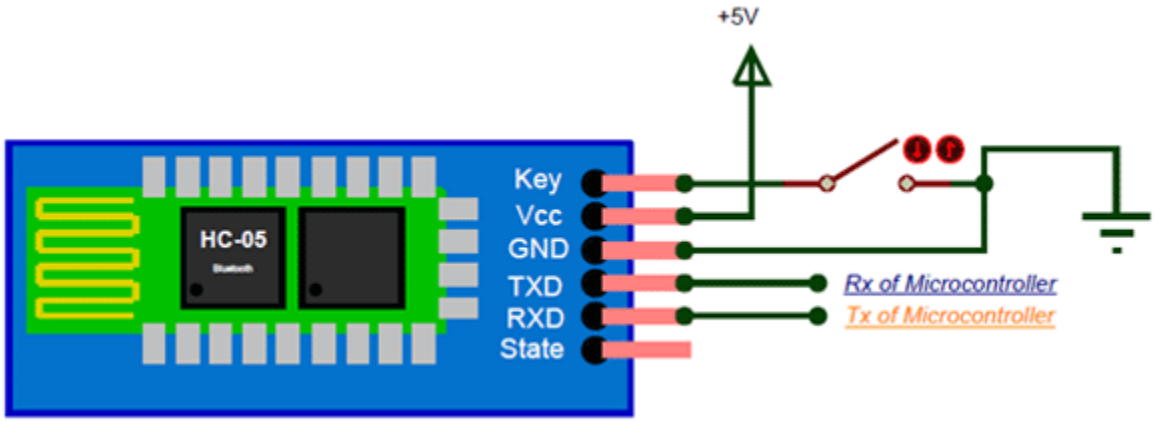


Figure 12: Architecture of Bluetooth module HC-05

2.5.4 Principle of operation

The HC-05 has two operating modes namely Data mode in which it can transmit or receive data from other devices, and other is AT command mode where the default configuration or settings are changed. The mode can be switched by using the KEY pin.
It is very easy to communicate with microcontroller using this module because it supports serial communication. During power up, the key pin can be grounded to enter into command mode, if it is left free or floating it will enter into data mode by default.

2.5.5 Connection with microcontroller system

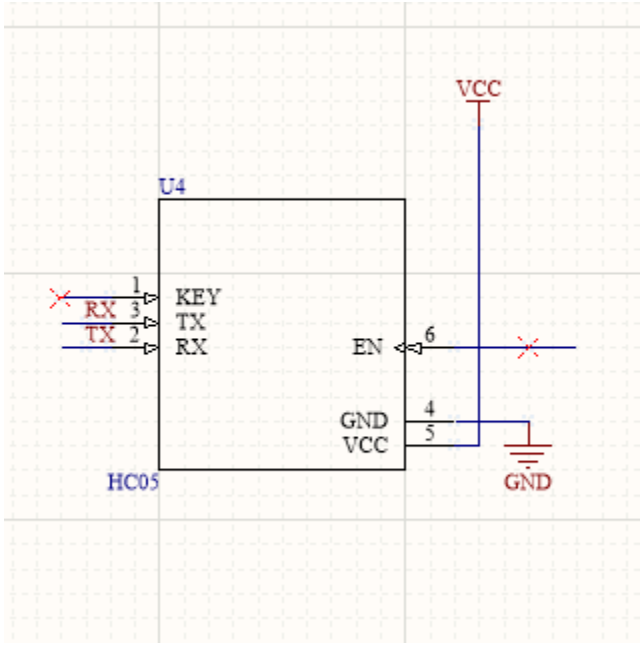
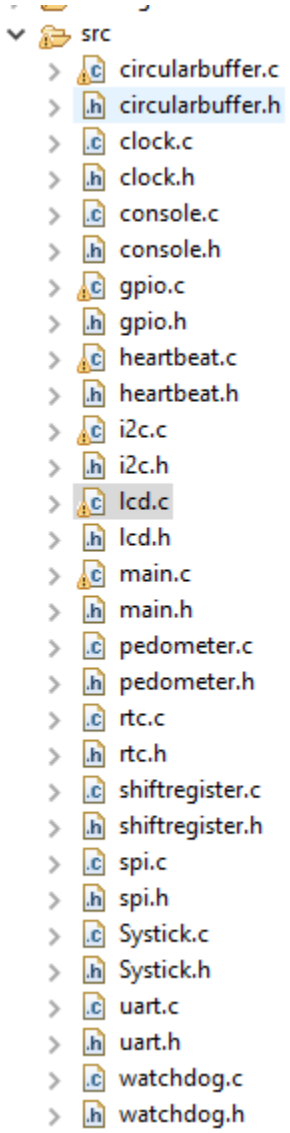


Figure 13: Connection of Bluetooth module HC-05 with MSP432

KEY: Left floating so that enters data mode by default.
TX: Connected to RXD pin of MSP432 which is
RX: Connected to TXD pin of MSP432 which is
EN: Not connected
VCC and GND: Power supply pins.

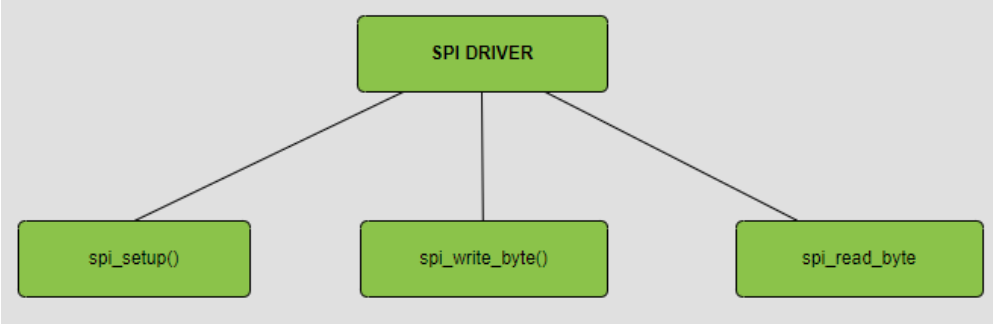
2.6 FIRMWARE DESIGN

The file structure of my project looked something as follows:
There were about 14 source files which were written at register level.



2.6.1 SPI driver

A SPI driver was written at register level to interface with Pedometer sensor.

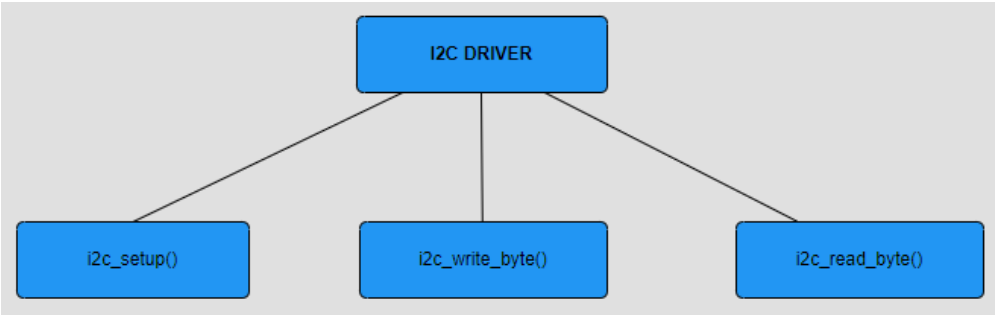


Functions that were implemented:

1) SPI initialize function: The purpose of this function was to set the bit rate, GPIO for MOSI,MISO and enable the SPI module.
   Prototype: void spi_setup();
2) SPI read byte: The purpose of this function is to read a byte from sensor using SPI interface
   Prototype: uint8_t spi_read_byte(uint8_t reg_addr);
3) SPI write byte:The purpose of this function is to write a byte to sensor using SPI interface
   Prototype: void spi_write_byte(uint8_t reg_addr, uint8_t data_byte);

2.6.2 I2C driver

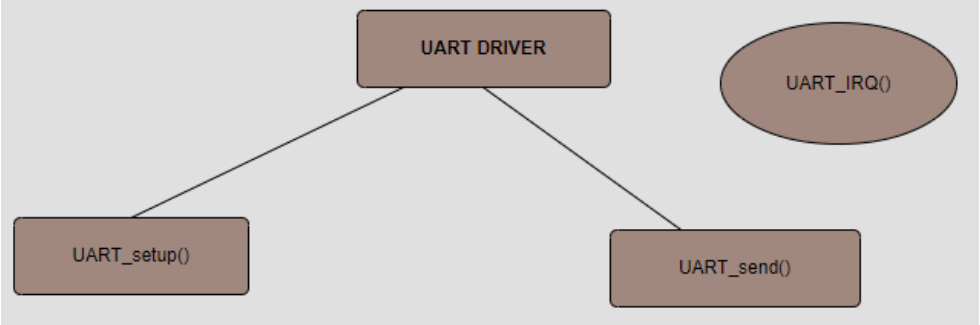
An I2C driver was written at register level to interface with Heart beat sensor and temperature sensor.



1) I2C initialize function: The purpose of this function is to set bit rate, GPIO for SDA, SCL and enable the I2C module
   Prototype: void i2c_setup();
2) I2C read byte: The purpose of this function is to read a byte from sensor using I2C interface
   Prototype: void i2c_read_byte();
3) I2C write byte: The purpose of this function is to write a byte to sensor using I2C interface
   Prototype: void i2c_write_byte(uint8_t slave_addr,uint8_t reg_addr,uint8_t data_byte);

2.6.3 UART driver

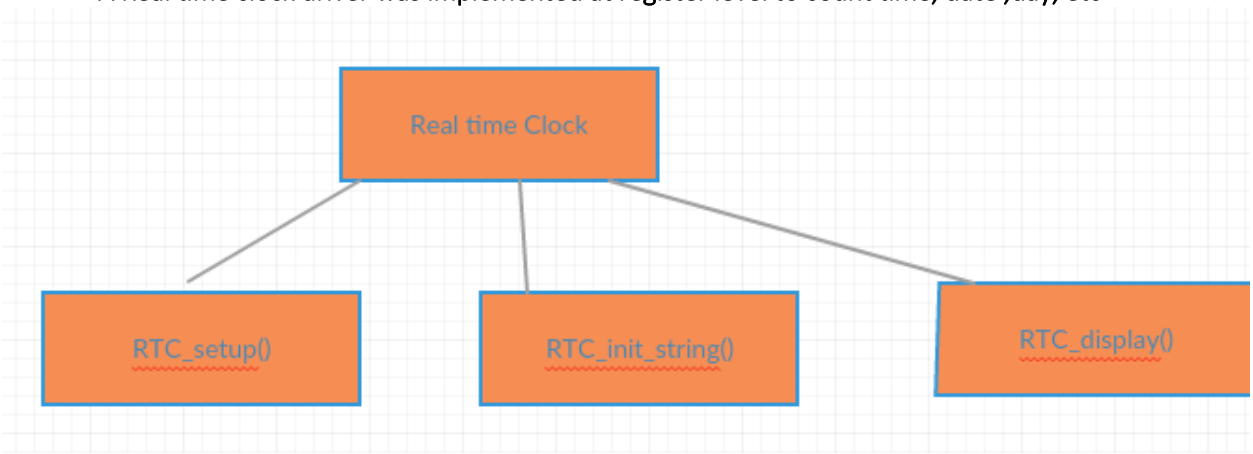
An UART driver was written at register level to interface with Bluetooth module HC-05.



1) UART initialize function: The purpose of this function is to set the GPIO(RX, TX), baud rate and enable the UART module
   Prototype: void UART_setup();
2) UART send function: The purpose of this function is to write a byte to serial terminal
   Prototype: UART_send(uint8_t *str);
3) UART IRQ Handler: The purpose of this IRQ handler is to handle Tx and Rx interrupts.
   Prototype: void EUSCIA2_IRQHandler();

2.6.4 Real time Clock Peripheral driver

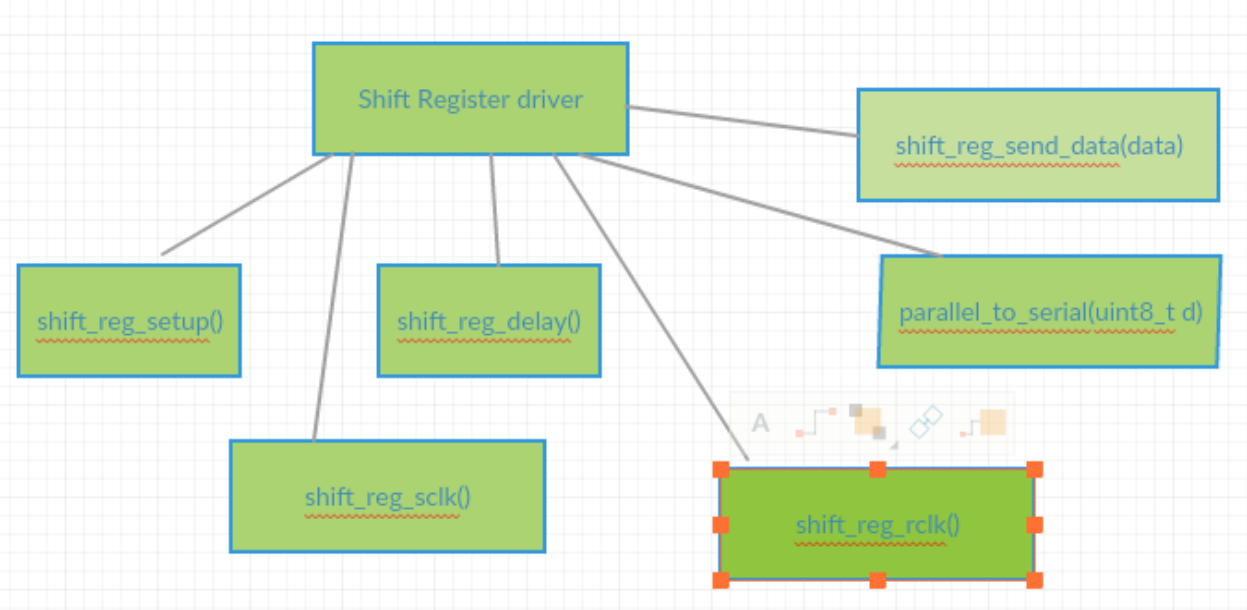
A Real time clock driver was implemented at register level to count time, date ,day, etc



1) RTC initialize function: The purpose of this function is to initialize the RTC i.e Date, Day and time.
   Prototype: void RTC_setup();
2) RTC intialise string: This is responsible for printing the date, day and time on LCD each time LCD goes to ISR and returns back from it.
   Prototype: void RTC_init_string();
3) RTC display string: Responsible for printing date, day and time on LCD
   Prototype: void RTC_display();

2.6.5 Shift Register driver

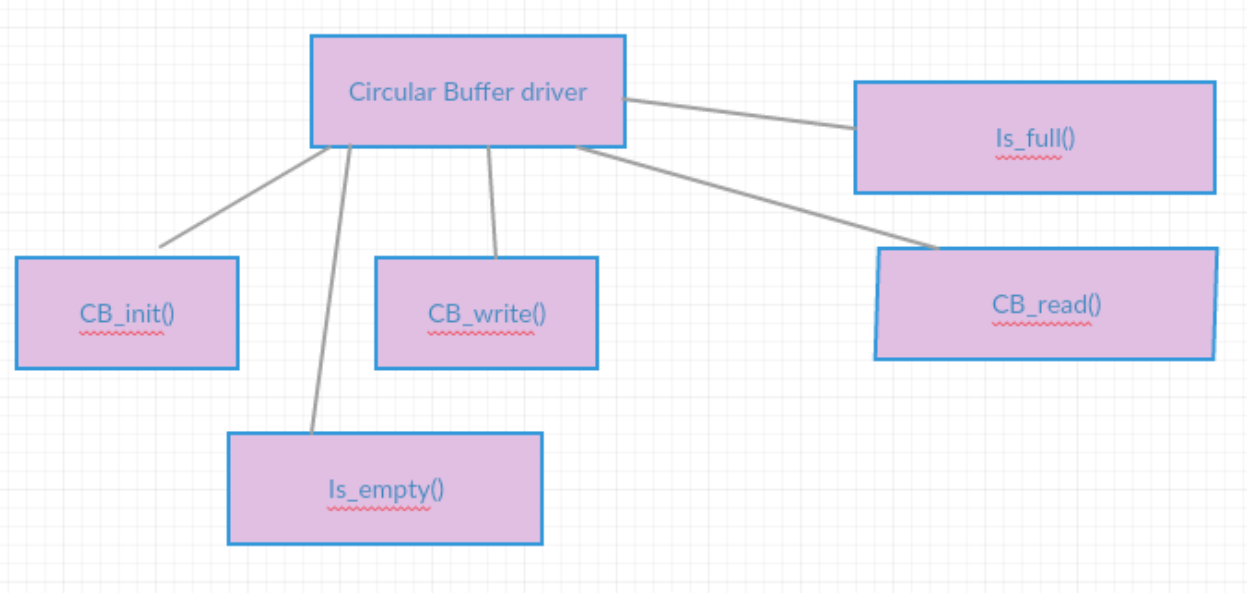
A shift register driver was implemented to convert serial data into parallel one.



1) Shift register initiliaze: This function is responsible for setting up the GPIO pins which are associated with shift register functionality.
   Prototype: void shift_reg_setup();
2) Shift register delay: This function is responsible for adding delay between positive and negative level of clock pulse
   Prototype: void shift_reg_delay();
3) Parallel to serial conversion: The purpose of this function is to convert the 8 bit data value in register into single bit stream
   Prototype: void parallel_to_serial(uint8_t data_byte);
4) Shift register source clock: The function is responsible to provide sclk to the storage register to shift the data each time new data is available on serial line
   Prototype: void shift_reg_sclk();
5) Shift register strobe clock: This function is responsible for strobing out the serially inputted data into parallel form.
   Prototype: void shift_reg_rclk();
6) Send data to shift register: This function is responsible for sending data to serial register
   Prototype: void shift_reg_send_data(uint8_t data_byte);

2.6.6 Circular Buffer driver

A circular buffer was implemented to solve the producer consumer problem of UART

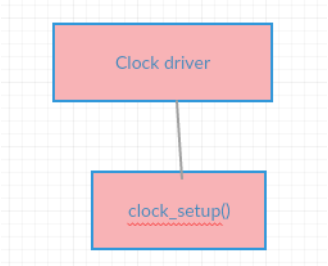


1) Circular buffer initialize: This function is responsible for setting the size of buffer on heap, head and tail pointer as well as total length
   Prototype: void CB_init(CB_t *src, uint8_t length);
2) Circular buffer write: This function is responsible for writing data into circular buffer.
   Prototype: CB_status CB_write(CB_t *src, uint8_t data_byte);
3) Circular buffer read: The purpose of this function is to read data from circular buffer
   Prototype: CB_status CB_read(CB_t *src,uint8_t *data);
4) Check whether circular buffer is full: The purpose of this function is to check whether circular buffer is full to prevent overwriting the data
   Prototype: CB_status Is_full(CB_t *src);
5) Check whether circular buffer is empty: The purpose of this function is to check whether circular buffer is empty to prevent underflow
   Prototype: Cb_status Is_empty(CB_t *src);

   Note: CB_t is a structure which contains the pointer to buffer, head pointer, tail pointer, length of buffer and current count
   CB_status is an enum which contains status value of circular buffer

2.6.7 Clock driver

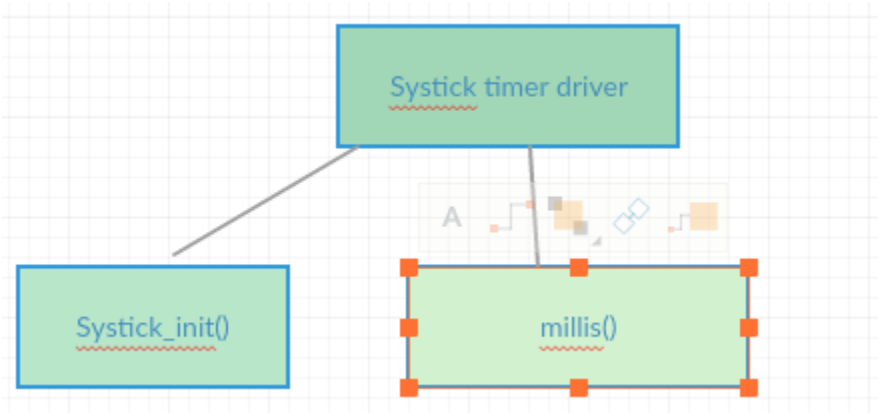
The function of this driver is to set clock frequency to 12 MHz.



1) Clock initialize function: This function is responsible for setting the clock frequency to 12 MHz using the DCO as clock source.
   Prototype: void clock_setup();

2.6.8 Systick timer driver

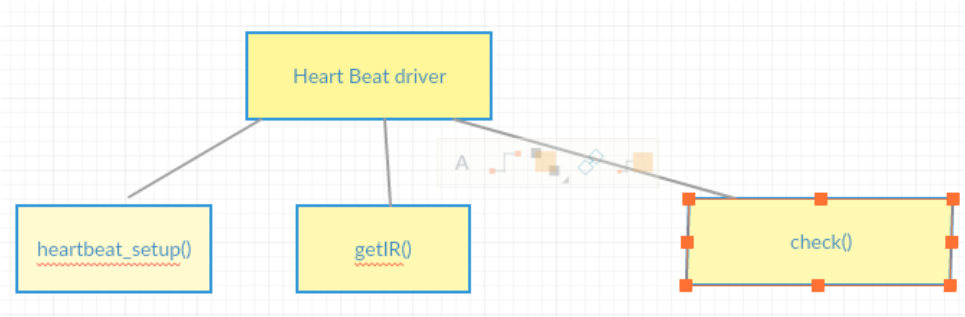
The function of this driver to create a very accurate and precise delay.



1) Systick timer initialize: This function is responsible for setting the reload value of Systick timer, and interrupt handler of Systick timer
   Prototype: void Systick_init();
2) Counts delay since program start in millisec: This function is responsible for calculating the elapsed time in milliseconds
   Prototype: uint32_t millis();

2.6.9 Heartbeat / Pulse-rate sensor

The function of this driver to make the HeartBeat sensor operational



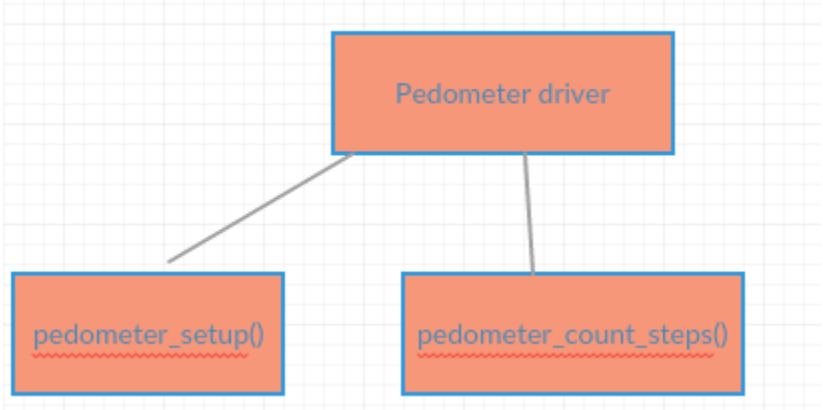
1) HeartBeat sensor initialize: This function is responsible for setting up the various configuration register of HeartBeat sensor

Prototype: void heartbeat_setup();

2) **Get IR reading:** Get IR readings based upon which heart rate is calculated
Prototype: uint32_t getIR();
3) **Check for new data:** This function is responsible for checking whether any new data is available in FIFO buffer
Prototype: int32_t check(void);

2.6.10 Pedometer sensor driver

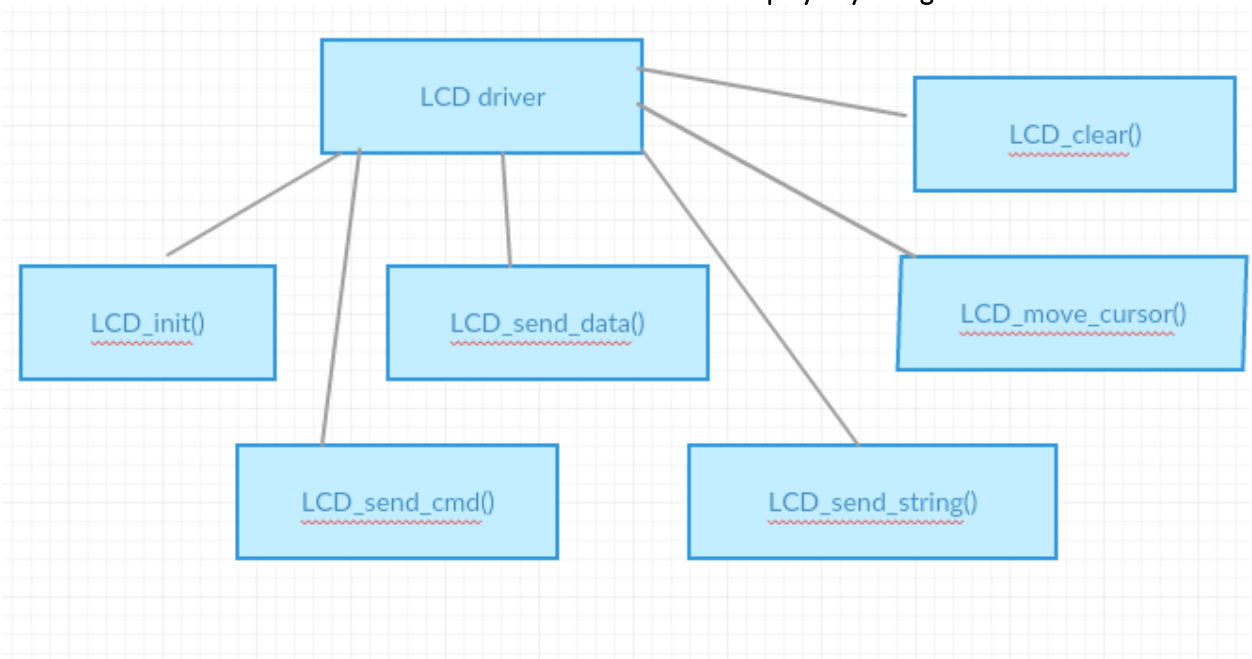
The function of this driver to make the Pedometer sensor operational



1) **Pedometer sensor initialize:** Responsible for setting various configuration register and make the accelerometer work as pedometer.
Prototype: void pedometer_setup();
2) **Count the number of steps:** Responsible for counting the number of steps in pedometer
Prototype: uint8_t pedometer_count_steps();

2.6.11 LCD driver

The function of this driver is to make LCD functional to display any strings.



1) **LCD initialize function:** The purpose of this function is to initiliase LCD to display string
Prototype: void LCD_init();

2) LCD command register access: The purpose of this function is to send command to LCD.
   Prototype: void LCD_send_cmd(uint8_t cmd);
3) LCD data register access: The purpose of this function is to send data to LCD.
   Prototype: void LCD_send_data(uint8_t data);
4) LCD send string: This function is responsible for sending string to LCD
   Prototype: void LCD_send_string(uint8_t *src);
5) LCD move cursor: This function is responsible for moving cursor to specified position on LCD
   Prototype: void LCD_move_cursor(uint8_t address);
6) LCD clear function: This function is responsible for clearing all the contents on LCD
   Prototype: void LCD_clear()

2.7 SOFTWARE DESIGN

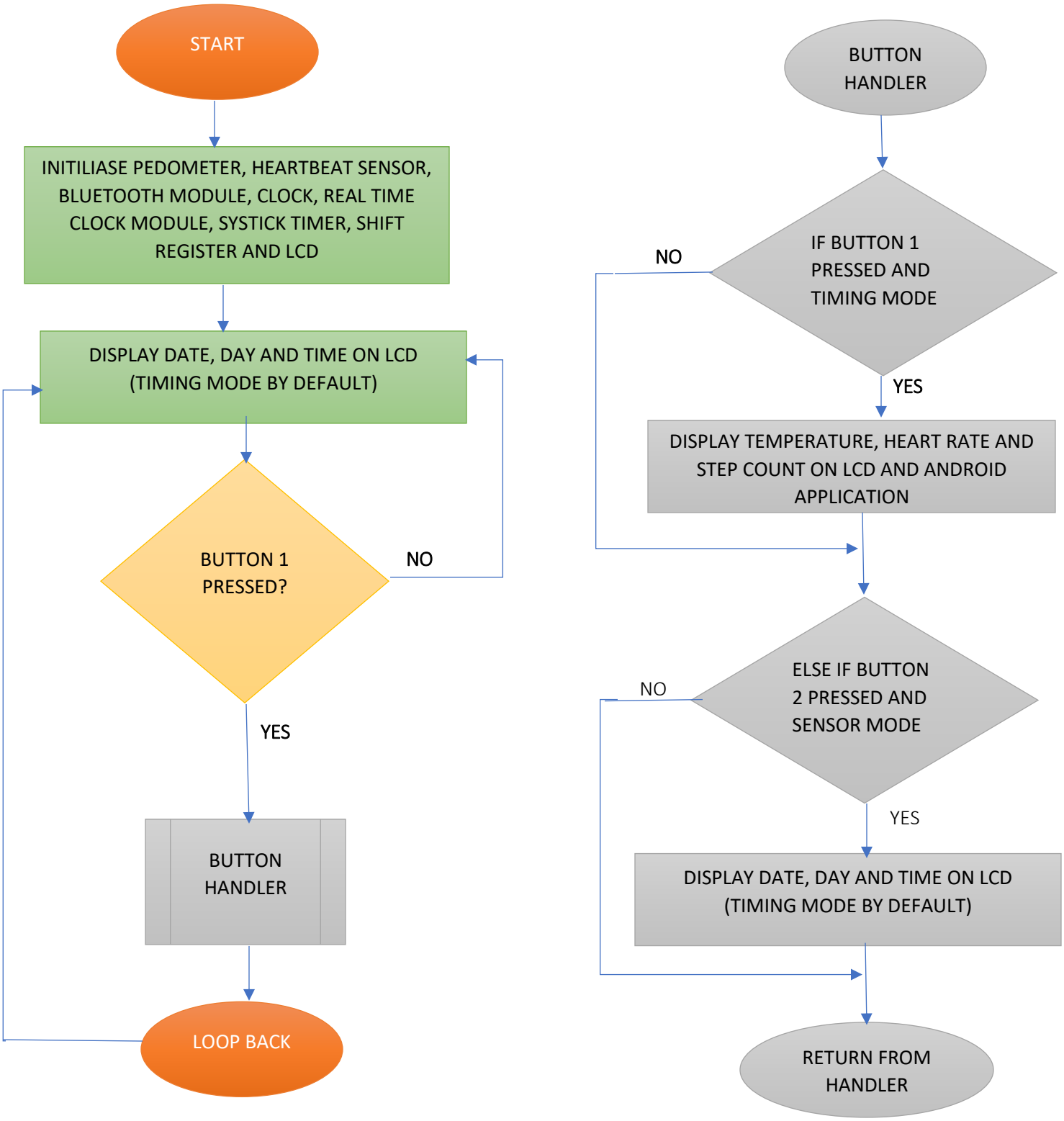


Figure 14: Software design of System

Description of Software flow:

1) Initially when the system is powered ON, all the system initialization takes place such as initialization of LCD, Bluetooth, Heart-rate, pedometer, and shift register.
2) After system initialization is done, the system displays the Date, Day and Time on LCD
3) Unless any interrupt occurs on GPIO i.e Button 1 or Button is pressed, the system continuously displays the clock. This is called as the timing mode.
4) When button 1 is pressed the system switches from timing mode to sensor mode in which values of sensor is displayed on LCD and also on Android application.
5) If Button 2 is pressed when the system is in timing mode, no switch happens. Also when button 1 is pressed when system is in sensor mode, no switch happens since system is already in sensor mode.

2.8 Testing Process

Testing SPI interface:

SPI write byte: A byte having value 0x08 is written into one of the register of pedometer at location 0x01

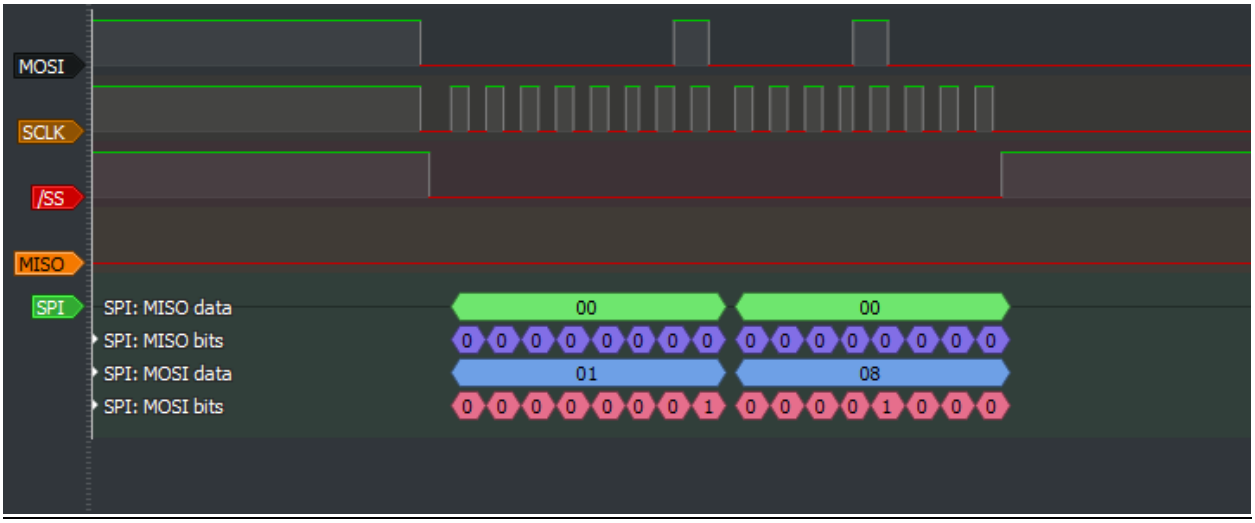


Figure 15: Logic Analyzer view – SPI single byte write

SPI read byte: A byte having value 0x08 is read from pedometer register having address 0x01

Note: The 0x81 address is because the MSb specifies read/write bit

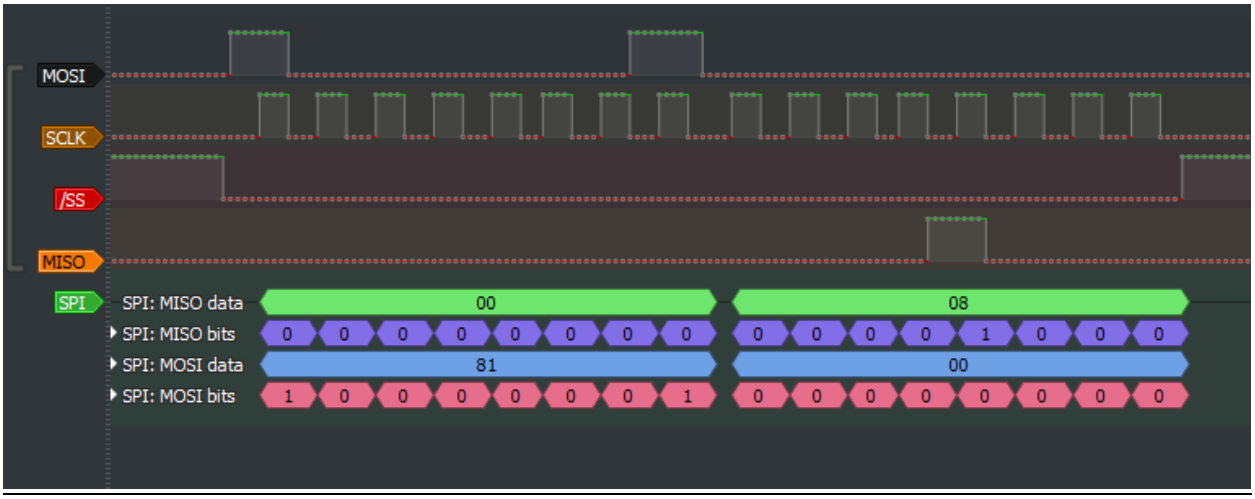


Figure 16: Logic Analyzer view – SPI single byte read

Testing I2C interface:

Reading three bytes from I2C:

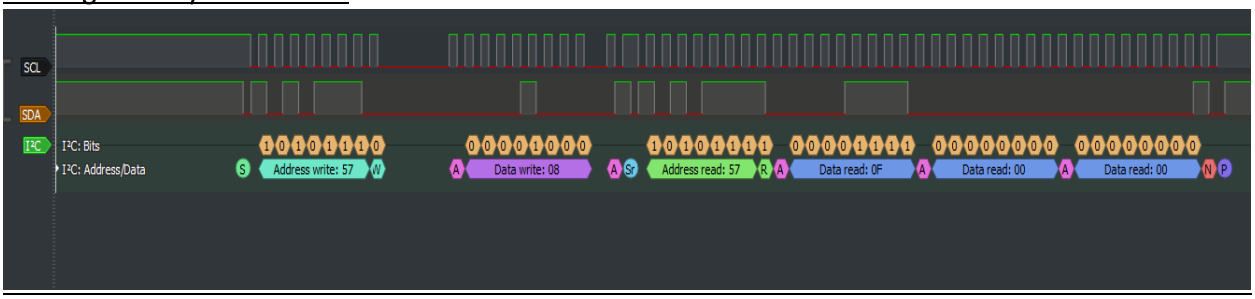


Figure 17: Logic Analyzer view – I2C three bytes read

Shift register testing: Sending data 0x0A followed by 0xA1. LSb is transmitted first

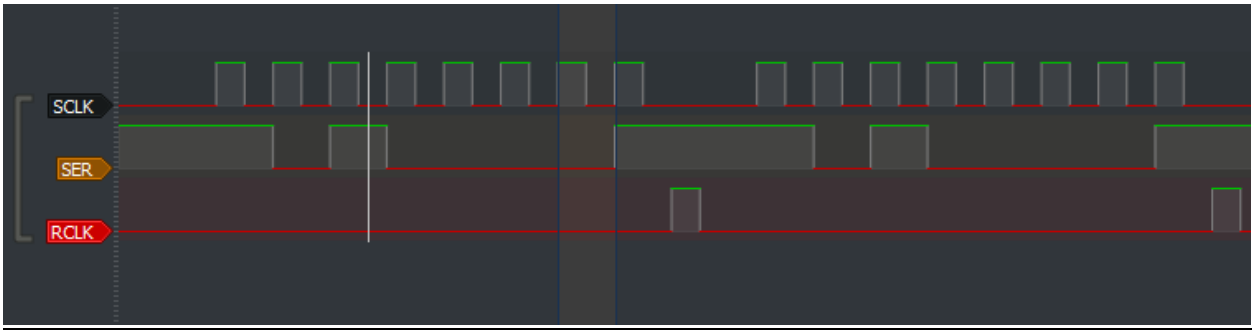


Figure 18: Logic Analyzer view – shift register 74HC595 transaction

UART testing:

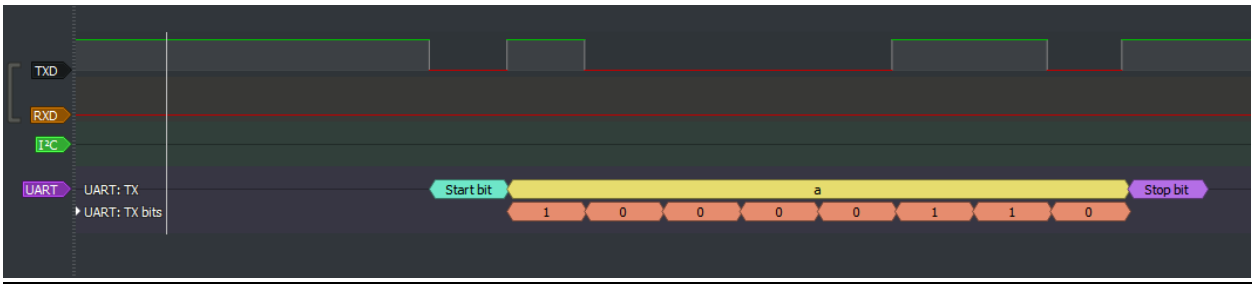


Figure 19: Logic Analyzer view – UART TRANSMIT

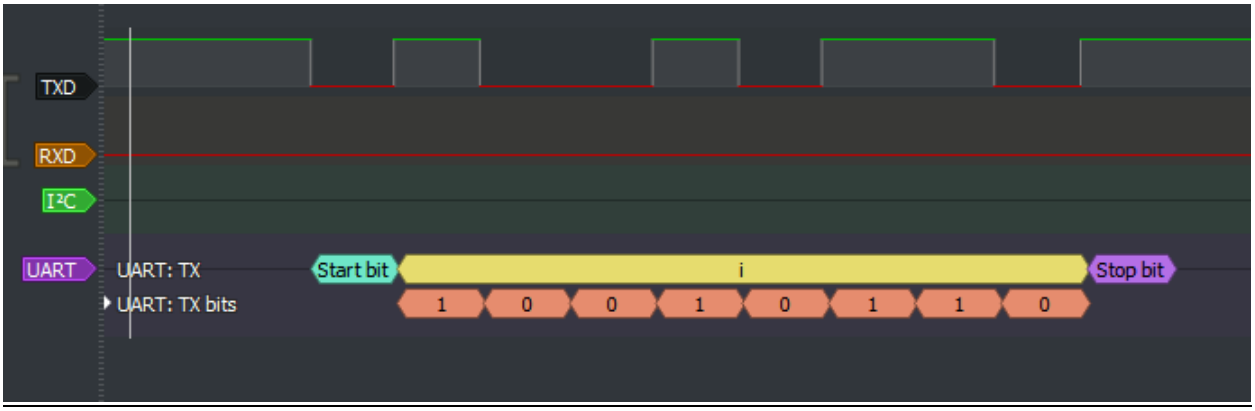


Figure 20: Logic Analyzer view – UART WRITE

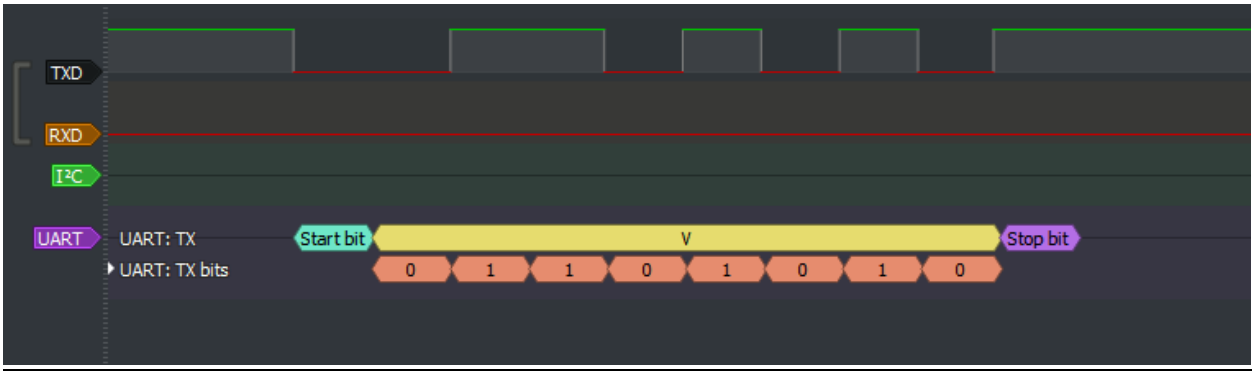


Figure 21: Logic Analyzer view – UART WRITE

Bluetooth based android application that displays sensor values when button 1 is pressed:

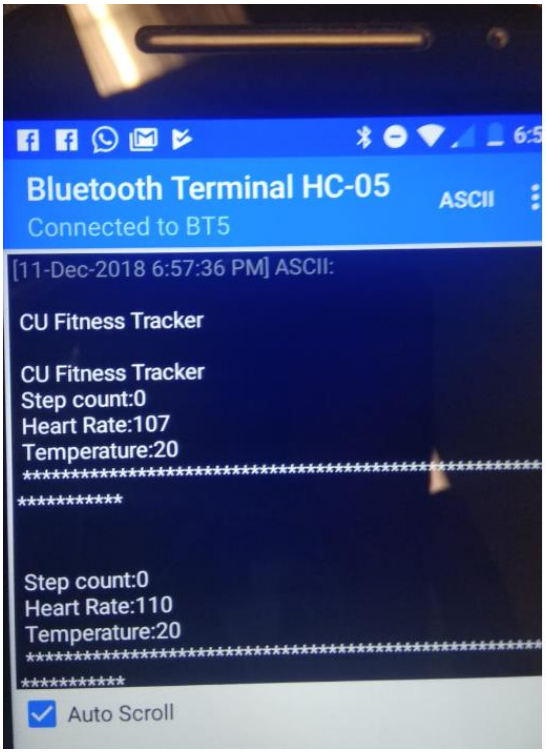


FIGURE 22: ANDROID APPLICATION

Screenshot taken when finger is not placed on Heart Rate sensor:

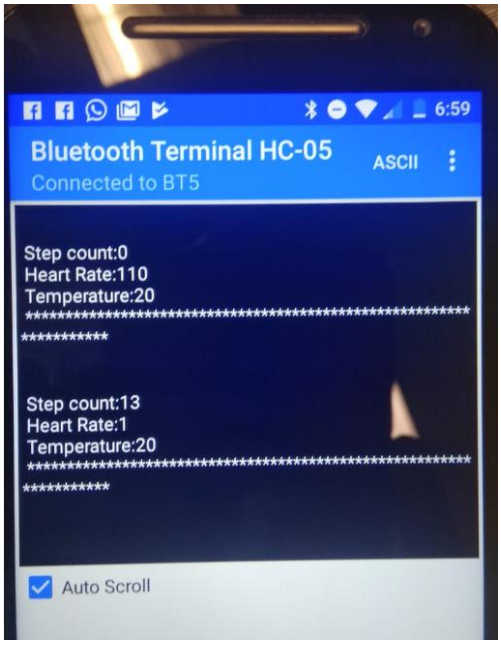


FIGURE 23: ANDROID APPLICATION

System in sensor mode:

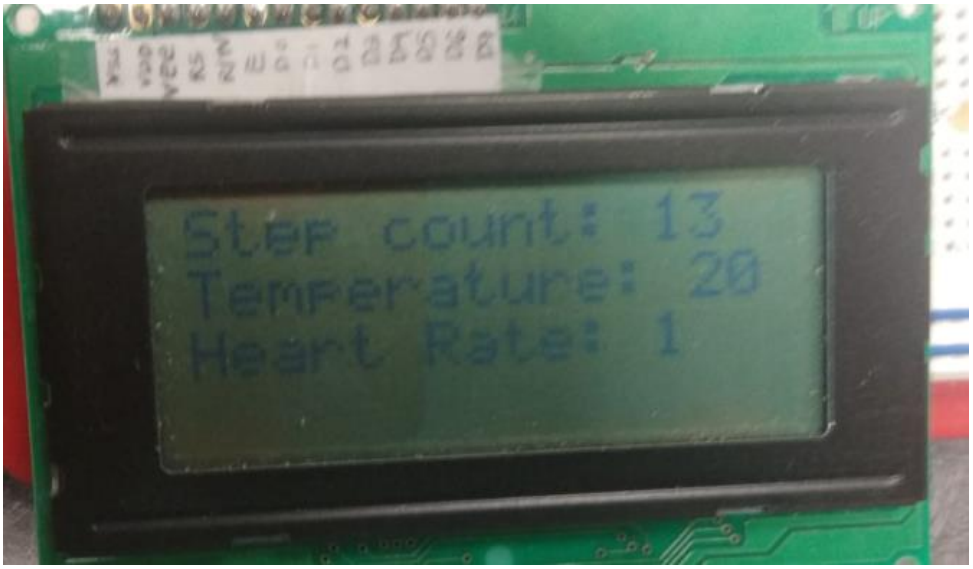


FIGURE 24: SYSTEM IN SENSOR MODE

System in timing mode:

FIGURE 25: SYSTEM IN TIMING MODE

CHAPTER 3 RESULT AND ERROR ANALYSIS

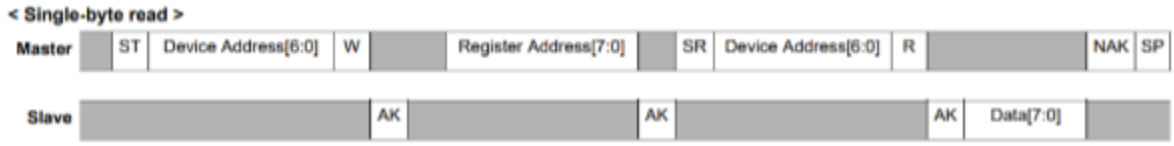
3.1 COMPARISON OF COMMUNICATION PROTOCOL USED

After having used all the three protocols in my design, I tried to perform some experiment to know which protocol is best for which application

We know that Energy = Power * Time. Let us do some engineering calculation,

3.1.1 Comparing the Time of Single Byte transfer:

1) I2C:



A total of 30 – 40 cycles are required for a single byte transfer in I2C. Assuming that bus is operating at speed of 1000 KHz, time required for single byte transfer will be 40 useconds.

2) SPI

A total of 16 cycles are required for single byte transfer. Note that the number of cycles required are very less as compared to I2C as there is no overhead of ACK/NACK involved in case of SPI protocol. SO, the total time required for single byte transfer at speed of 10,000KHz is 1.6 useconds

3) UART

A total of 11 cycles are required for single byte transfer. Assuming a Baud rate of 115200, the time required for single byte transfer is 95.5 useconds.

3.1.2 Comparing the Power of Single Byte transfer

1) I2C:

I2c bus can run in lowest energy mode with current consumption of 60 uA.

Assuming that it is running at crystal frequency of 48MHz. Active Current = 60 uA * 48 MHz = 2.89 mA

Assume 50% of the time, the pullups are active(2 pull-ups: SCL and SDA) Quiescent current = 2 *0.50 *(3.3/4.7K) = 0.6875 mA

Total current = Active current + Quiescent current(pull ups) = 3.5775 mA

Total Power = V * I = 3.3V * 3.5775 mA = 11.805775 mW

2) SPI:

Considering SPI bus, there are no pull ups involved so the total power consumption is only due to active current consumption,

Total Power = V*I = 3.3 * 2.89mA= 9.537 Mw

3) UART:

UART almost consumes same amount of power as SPI and its calculation is almost the same like since there are no Pull ups involved, so total power consumption will be 3.3 * 2.89mA = 9.537mW

3.1.3 Comparing the Energy consumption:

Digital Serial Interface	Power	Time	Energy
I2C @10,000 KHz	11.805775 mW	40 useconds	472.231 nJ
SPI @ 10,000 KHz	9.537 mW	1.6 useconds	15.2592 nJ
UART @115200	9.537 mW	95.5 useconds	910.7835 nJ

3.1.4 Other considerations for Digital Serial Bus

1) I2C
   Advantages: Addressable address bus upto 128 devices
2) SPI
   Disadvantages: Requires additional GPIO for chip select for each addressable device
3) UART
   Disadvantage: Requires additional UART resource and GPIO pins for each device.

3.2 Performance metrics:

3.2.1 Response time

The response time was found out by profiling the program using the Systick timer

Response time = 2.82 msecnds

3.2.2 Memory loading

The total memory was calculated based on memory map:

```
MODULE SUMMARY

       Module                          code   ro data   rw data
       ------                          ----   -------   -------
    .\
       system_msp432p401r.obj          820    0         8
       startup_msp432p401r_ccs.obj     14     228       0
    +--+------------------------------+------+---------+---------+
       Total:                          834    228       8

    .\src\
       heartbeat.obj                   1516   24        180
       rtc.obj                         1080   0         0
       i2c.obj                         688    0         0
       main.obj                        624    0         0
       uart.obj                        484    0         16
       lcd.obj                         352    0         0
       circularbuffer.obj              344    0         0
       spi.obj                         268    0         0
       Systick.obj                     144    0         112
       console.obj                     256    0         0
       gpio.obj                        228    0         8
       shiftregister.obj               224    0         0
       pedometer.obj                   200    0         0
       clock.obj                       76     0         0
       watchdog.obj                    16     0         0
    +--+------------------------------+------+---------+---------+
       Total:                          6500   24        316

    C:\ti\ccsv8\tools\compiler\ti-cgt-arm_18.1.3.LTS\lib\rtsv7M4_T_le_v4SPD16_eabi.lib
       fd_add_t2.asm.obj               438    0         0
       memory.c.obj                    276    0         24
       fd_mul_t2.asm.obj               252    0         0
       memcpy_t2.asm.obj               156    0         0
       fd_tos_t2.asm.obj               110    0         0
       copy_decompress_lzss.c.obj      104    0         0
       autoinit.c.obj                  68     0         0
       boot_cortex_m.c.obj             56     0         0
       i_tofd_t2.asm.obj               46     0         0
       _lock.c.obj                     2      0         16
       copy_zero_init.c.obj            18     0         0
       copy_decompress_none.c.obj      14     0         0
       exit.c.obj                      4      0         0
       pre_init.c.obj                  4      0         0
    +--+------------------------------+------+---------+---------+
       Total:                          1548   0         40

       Heap:                           0      0         2048
       Stack:                          0      0         1024
       Linker Generated:               0      70        0
    +--+------------------------------+------+---------+---------+
       Grand Total:                    8882   322       3436
```

3.3 LESSONS LEARNT

1) Check for CPOL and CPHA parameter of SPI protocol for every sensor. The value of those bits may be different for every sensor and hence needs to be taken into consideration to make SPI bus operational.

2) Check the value of pull up resistors on I2c. They are function of Bus capacitance and Rise time and when they are not taken into consideration, the I2C bus may not work.

3) The Bit rate of each protocols must be within the given range in datasheet. Not following those values may result in dysfunctional protocol.

4) Always ensure the voltage on GPIO pins before connecting to sensor. Some microcontroller output 5V on their GPIO and hence they cannot be connected to any sensor that operates at 3.3V. A level converter is required in those case.

5) It is a good idea to keep ISR for each of the peripheral used rather than opting for polling mode. We can allow the microcontroller to enter power down mode in main function thus making the system energy efficient.

CHAPTER 4 CONCLUSION

The fitness tracker was designed with the idea to measure the parameters associated with body such as temperature, Heartbeat and Steps taken by user. One important thing that I learned while building the project was how to use various protocols such as I2C, SPI and UART.

I took risk of using four new hardware elements in a single project considering the amount of time I had. I feel that the project has been a success for me and taught me how to select a project considering the skillset that I have.

Although the embedded system that I have developed is not competitive with similar products available in the market, designing this embedded system from scratch gave me a good idea of how hardware and firmware should be developed in incremental approach and then how they should be integrated to get a complete robust system.

The strategies I used in my design, right from debugging the hardware with oscilloscopes and logic analyzer gave me some confidence that my design was correct and this skills were something that were lacking in my engineering toolbox.

CHAPTER 5 FUTURE DEVELOPMENTS

Below are some points which I think I would add in my project in my future iteration

1) Scheduling with RTOS: Scheduling the individual task with RTOS can achieve higher efficiency and improve the execution , response time. As an example, I can use FreeRTOS that can schedule individual task, and it would be a viable choice since TI MSP432 supports FreeRTOS.

2)Making the device low power: Since the device would be a wearable technology and possibly a IOT device, I think it would be a good idea to make the device consume low power since it will increase the lifetime of the product and will sustain for long period of time.

3) Use of Energy Harvesting mechanism: It makes sense to add a energy harvesting mechanism that can act as a source of energy for a energy reservoir such as battery or supercapacitor. For eg, adding a solar charging would make the device very energy efficient

4) Making a two-layer Printed Circuit board: A printed circuit board would certainly reduce the footprint and space thus making it sitable wearable technology.

5) Addition of other sensors: Addition of other sensors such as humidity, altimeter, calorie tracker would make a sophisticated fitness tracker. This is another consideration that I would keep in mind while developing future iteration.

CHAPTER 6 ACKNOWLEDGEMENTS

Firstly, I would like to thank Dr. Linden McClure for such great course that taught me Hardware and Software skills that I think would be useful throughout my career in Embedded Systems.

I would also like to thank Professor Keith Graham for giving me this idea of Fitness tracker mentioned in one of his lecture slides. It would have been possible to select this idea without Professor's lecture slides

I would also like to thank all the TA's for being there to solve all my doubts whenever I got stuck in a particular part of my implementation.

I would also like to thank my Parents for giving this opportunity to study in United States of America. Without their backing, it would have not been possible for me to get through all this.

I would like to thank all my friends back in India as well as United States of America for being patient enough to adjust with my schedule.

Lastly, I would like to thanks TI forum community without which it would not have been possible to resolve my doubts specific to TI MSP432.

CHAPTER 7 REFERENCES

1) http://ecee.colorado.edu/~mcclurel/Final_Project_Report_Example_Craner.pdf
2) https://www.gme.cz/data/attachments/dsh.772-148.1.pdf
3) http://www.ti.com/lit/an/slaa655/slaa655.pdf
4) https://fruct.org/publications/fruct13/files/Lau.pdf
5) http://www.techforfuture.nl/fjc_documents/mitrabaratchi-measuringheartratewithopticalsensor.pdf
6) https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library/blob/master/src/heartRate.cpp
7) https://www.explainthatstuff.com/how-pedometers-work.html
8) https://learn.adafruit.com/adafruit-arduino-lesson-4-eight-leds/the-74hc595-shift-register
9) https://www.sparkfun.com/products/13699
10) http://www.ti.com/lit/ds/symlink/sn74hc595.pdf
11) http://www.ti.com/lit/ug/slau356h/slau356h.pdf

CHAPTER 8 APPENDICES

8.1 Bill of Materials

Part Description	Manufacturer	Cost
TI MSP432	Texas Instruments	Free(Tool's kit)
HC-05 Bluetooth module	Gomcu	Free (from TA's)
Particle Sensor MAX30105	Sparkfun	Free (from my Friend)
Accelerometer LSM6DS3	Sparkfun	Free(from my friend)
Wires	Unknown	Free(from ITLL)
Shift Register 74HC595	Texas instruments	Free (from my friend)
LCD(Dot Matrix Display)	Unknown	Free (Tool's kit)
Header pins	Unknown	2 $
Misc(Breadboard, female,male wires)	Unknown	Free(from ITLL)

8.2 Schematics

Submitted separately as well

8.3 Firmware Source code

Submitted separately.

8.4 Software Source code

Software for Bluetooth was not written, but an already written Bluetooth application was used whose link is as follows:

8.5 Application notes and Datasheets

Submitted separately