Data logger for Automotive and Aerial Vehicles

Sanish Kharade
Vishal Raj

Final Project Report
ECEN 5613 Embedded System Design
December 11, 2021

Table of contents

1. Introduction

In our project, we have designed a data logging system by storing sensor data on a storage device. Our intent is to study a different type of storage device and its corresponding device controller. We have chosen a Class-10, 16GB Secure Digital High Capacity (SDHC) card mainly because of its popularity in many embedded applications and the ability to communicate with it using the SPI protocol.

Our application consists of three input sensors namely an accelerometer, a gyroscope and a vibration sensor which provide data to the microcontroller. The microcontroller processes the data and stores it in the SD card in suitable format. We have also used an LCD and a piezoelectric buzzer as the output components of the system. We are using the host PC for system debugging.

For the requirements of our application, we need a controller which has a higher processing power and can perform multiple tasks in less amount of time for which we have chosen the MSP432P401R which acts as the main storage device controller, the sensor data logger and implements a file system to write data on the SD card along with printing the output data on the LCD and the buzzer.

2. Technical Description

2.1 Hardware Design

2.1.1 Hardware Components

We have used the following hardware components in our project.

1. MSP432P401R

The MSP432 is a mixed-signal microcontroller family from Texas Instruments. It has 48MHz 32-bit ARM Cortex M4F with Floating Point Unit and DSP acceleration. It is a low power consumption board. It supports various communication protocols like UART, I2C, SPI etc. which can be used to interface different sensors. This microcontroller controls the entire functionality of the product. MSP432P401R is programmable with Code Composer Studio via a micro USB cable.
Reference : link

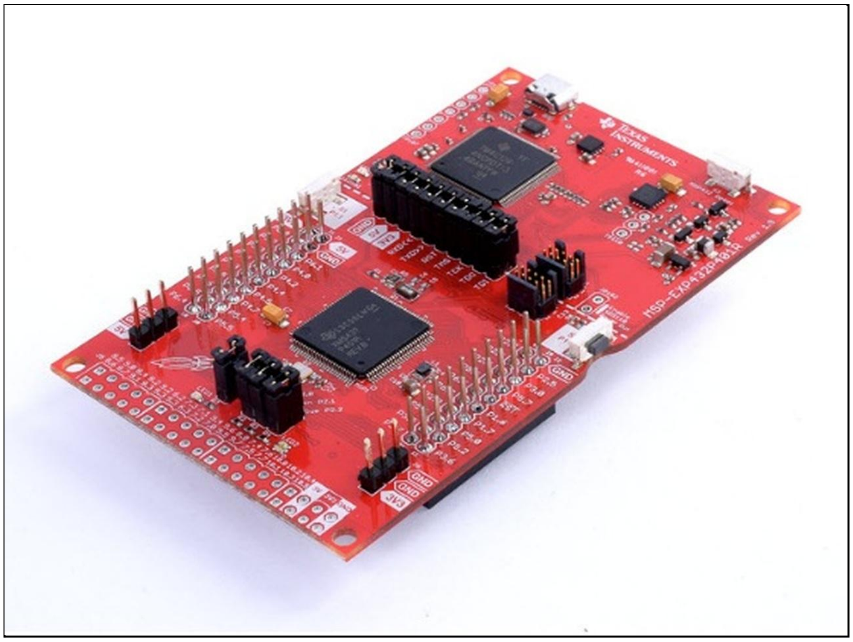


Figure 1: MSP432P401R
Reference : source

2. MPU6050

The accelerometer is the MPU6050 sensor which has a 3 axis accelerometer and gyroscope making a total of 6 channels. Each channel has its own 16-bit ADC. The output of all ADCs are given to a common signal conditioning block and the final values are stored in the sensor registers. In our application, the sensor communicates with MSP432 as an I2C slave and is operated at 3.3V.

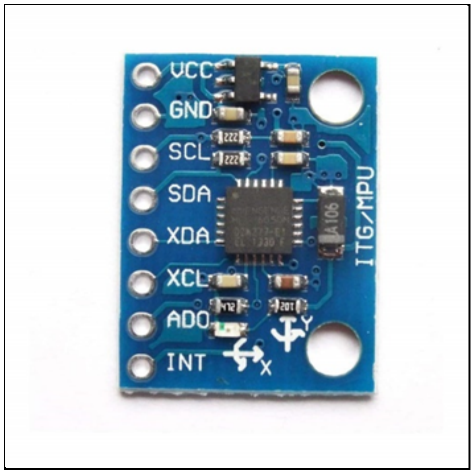


Figure 2: MPU6050.
Reference : [source]

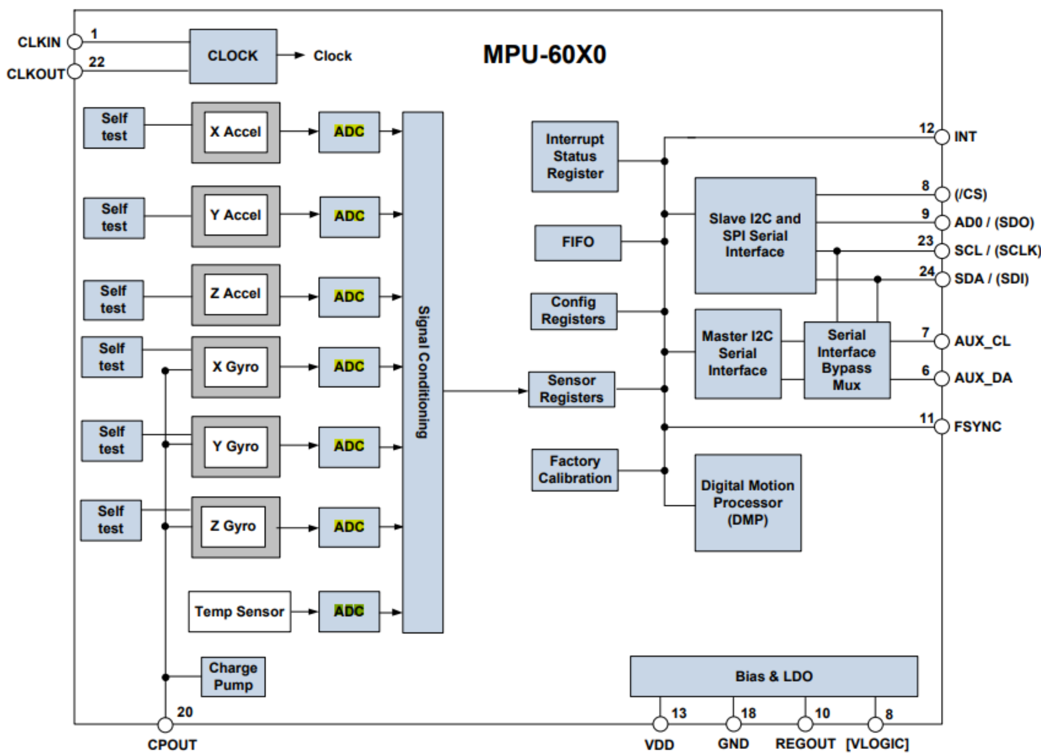


Figure 3: MPU6050 internal block diagram.
Reference : MPU6050 datasheet/Pg 24

3. 16x2 LCD

This LCD display is a 16x2 panel which communicates with the main controller via the I2C protocol. This I2C interface is achieved with an IO expander PCF8574T which is present on the bottom side of the board. The main LCD controller is the Hitachi HD44780U. It operates on 5V and the control and data lines are operated at 3.3v in our application. This LCD is operated in four bit data mode as only 8 lines are provided by the IO expander of which the four are control signals-backlight, Enable, RS, RW.

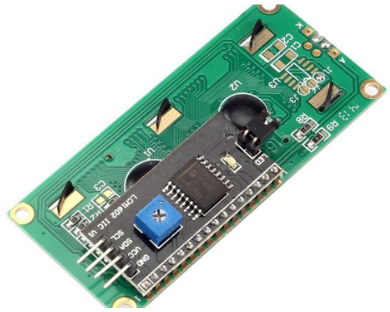


Figure 4: 16x2 LCD display.
Reference : [source]

4. Vibration Sensor

SW 420 is a vibration sensor which provides a low GPIO output when a vibration is detected. We developed the hardware circuit for this sensor by ourselves as you can see from Figure 6.

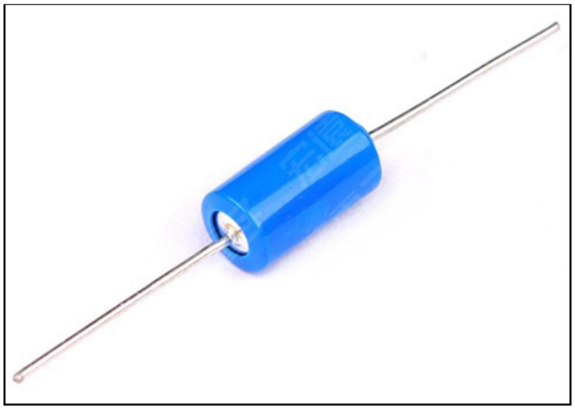


Figure 5: SW420 vibration sensor
Reference : [source]

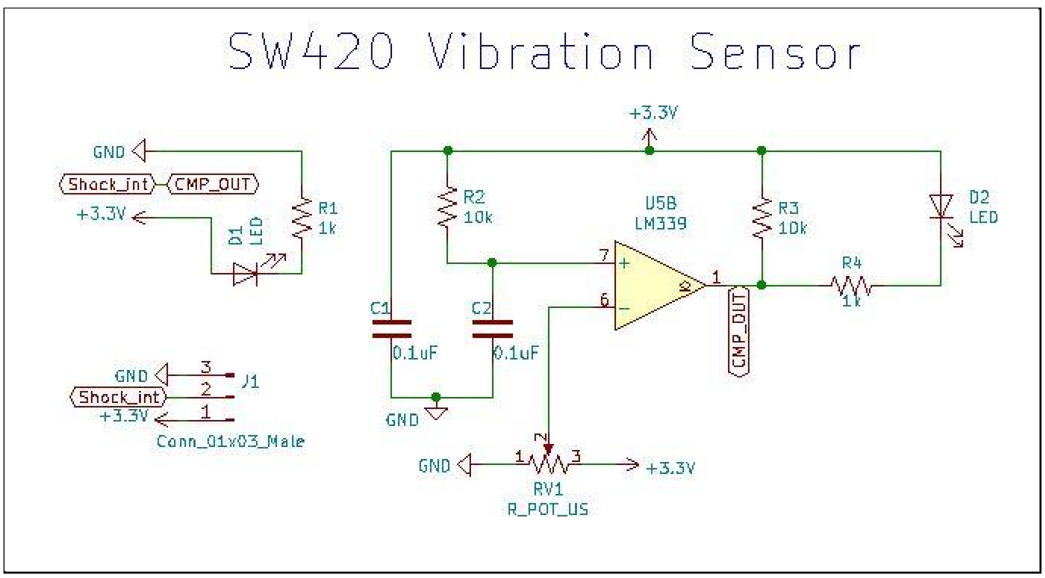


Figure 6: SW420 vibration sensor hardware circuit
Reference: Project Schematic

It consists of the vibration sensor, LM339 comparator, a 10k potentiometer, a power LED, an LED to indicate shock detection and a few resistors and capacitors. Whenever a shock is detected the sw420 generates a GPIO signal which is given to one input of the comparator. The potentiometer is used as the other input of the comparator and is used to control the threshold of vibrations. We adjusted the potentiometer such that no false vibrations are detected. There is a power LED on the circuit and another LED to indicate the detection of vibrations. This circuit needs to be mounted in a confined environment inside the vehicle so that it doesn't get triggered easily by small vibrations which might be caused due to normal circumstances. In case of aerial vehicles, the vibrations might happen due to jerks caused by sudden acceleration changes

5. Buzzer

The alarm which we have used in our project is a 90dB Active Piezo electric buzzer which has an operating voltage range of 3-24 VDC. The buzzer consists of two terminals positive and negative which are used to control the buzzer. According to the V-I characteristics, at 3.3v given by the GPIO of the MSP, the current required to drive the buzzer would be around 5mA and the sound level would be 67dB. Hence a normal IO pin is used to drive the buzzer as the current requirement is very less.

Figure 7: 16x2 LCD display.
Reference : Source

6. SD Card

The SD card is the main data storage element of our system. Since this is a data logger that will be used in automotive and aerospace vehicles, the amount of data gathered over the years will be huge. We have used a 16GB, class 10 micro SD card for our application so that we never fall short of memory for many years. The storage size of the SD card can be reduced which has been discussed in the future scope of this project.

Figure 8: SD Card

Reference : source

7. SD Card breakout Board

This module provides a 6 pin interface to communicate with the SD card using the SPI protocol. The module provides a spring loaded push/push mechanism to insert and remove the SD card. It has a 3.3V voltage regulator circuit and hence can be used at voltage level of 5V as well.

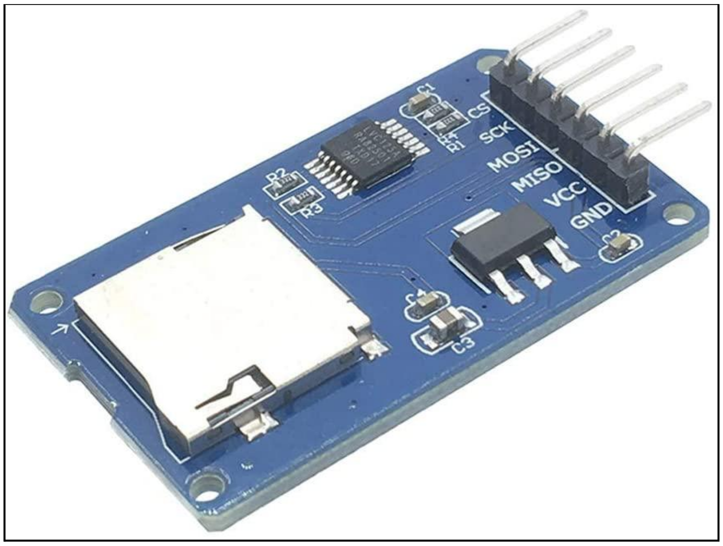


Figure 9: SD Card Breakout Board
Reference : source

2.1.2 System Design

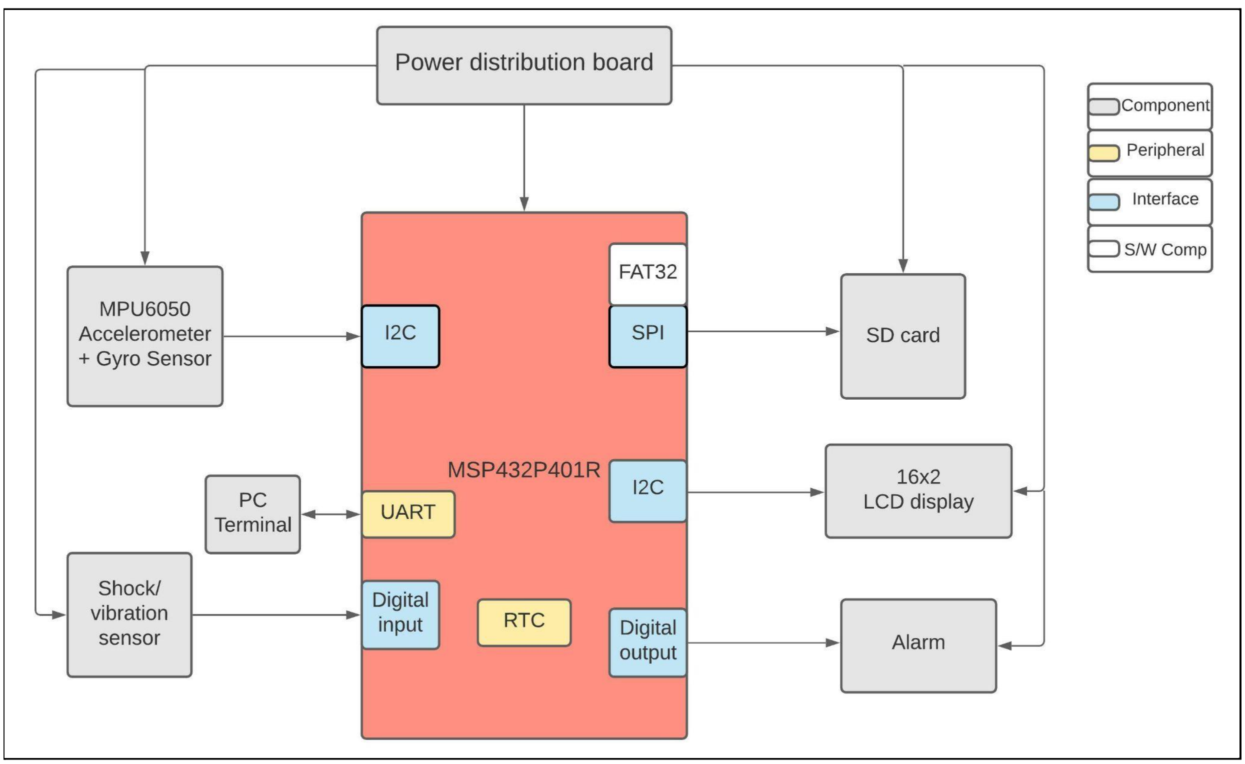


Figure 10: Block diagram of system

As you can see from the block diagram, MSP432 acts as the brain of our system. The input modules include the MPU6050 sensor which has an accelerometer and a gyroscope. These provide values of acceleration in m/s^2 and angular velocity in degrees/second respectively. MPU6050 is interfaced with the MSP432 using the I2C protocol. The other input module is the shock/vibration sensor. This provides GPIO interrupt to the MSP432 when a shock is detected.

The output of the sensor is displayed on the 16x2 LCD display along with the current time. This LCD display also communicates over I2C with the MSP432. We also have a buzzer (alarm) which directly corresponds to the shock sensor. It beeps twice whenever a shock is detected. The main storage element of our system is the SD card which is interfaced with the MSP432 using the SPI protocol. We have used a FAT32 file system to organise the files on the SD card. Finally the entire system gets power from the MSP432 itself. We have ensured that no peripheral is drawing more current than the rated parameters of each pin of MSP432.

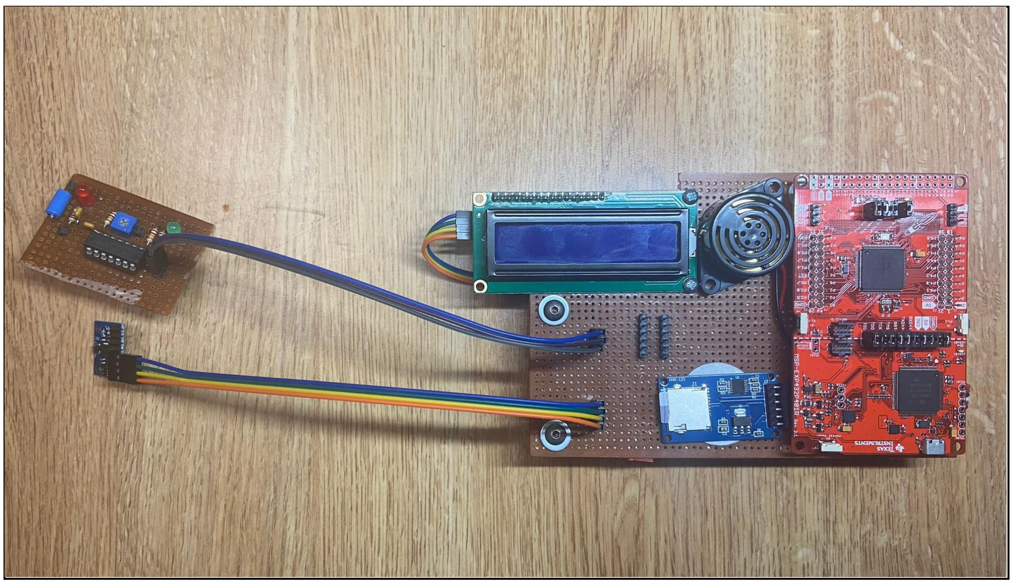


Figure 11: Final project top view.

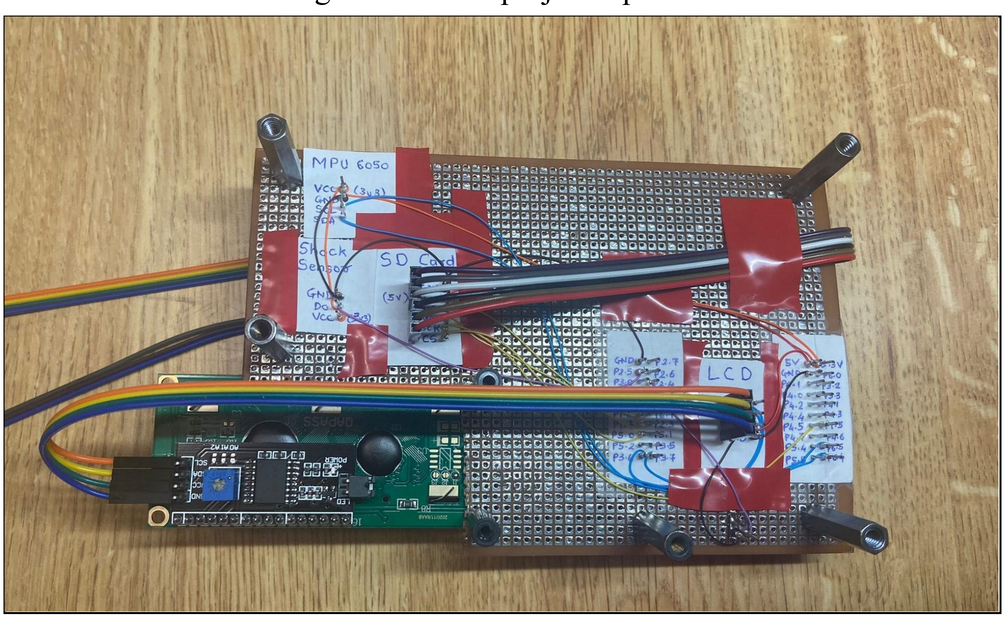


Figure 12: Final project bottom view.

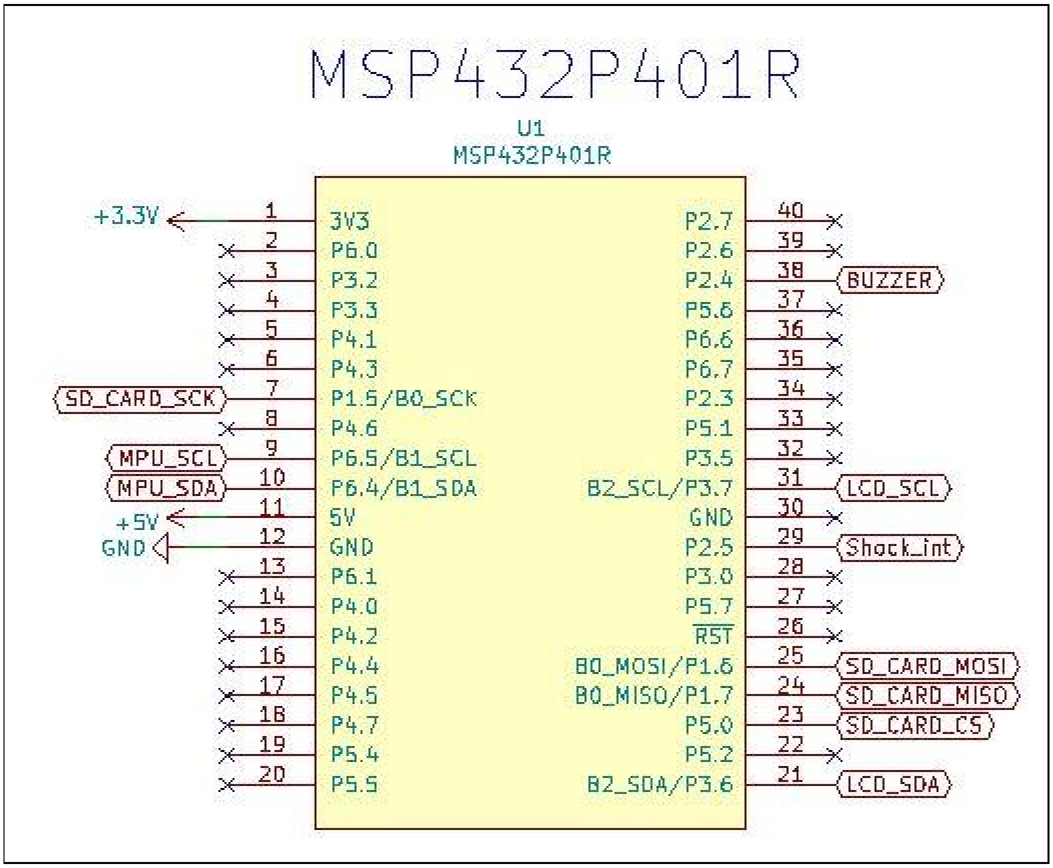


Figure 13: MSP432 connections
Reference : Schematic

2.2 Firmware Design

We worked on the following firmware modules

2.2.1 I2C protocol : MPU6050 Inertial measurement unit sensor

- The sensor gives 3 axis acceleration from its accelerometer and 3 axis angular velocity value from the gyroscope. These values need to be read from the sensor in a required format. These values are decided by the Gyroscope and accelerometer sensitivity factors given by the datasheet as below.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		º/s	
	FS_SEL=1		±500		º/s	
	FS_SEL=2		±1000		º/s	
	FS_SEL=3		±2000		º/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(º/s)	
	FS_SEL=1		65.5		LSB/(º/s)	
	FS_SEL=2		32.8		LSB/(º/s)	
	FS_SEL=3		16.4		LSB/(º/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			±2		%	
Nonlinearity	Best fit straight line; 25°C		0.2		%	
Cross-Axis Sensitivity			±2		%	

Figure 14: Gyroscope sensitivity specification
Reference : MPU6050 datasheet/Pg12

- For our application, we have chosen the Full-Scale range as ±500 degree per second i.e FS_SEL=1. In that case, the sensitivity scale factor is 65.5 LSB degrees per second. Hence, one degree of movement in any axis will give us a value of 65.5 in that axis.
- Similarly, for the accelerometer, the full scale range is selected as AFS_SEL = 2 i.e ±8g which gives us a Sensitivity Scale factor of 4096LSB per g(gravitational constant of 9.8m/sec^2).

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		*g*	
	AFS_SEL=1		±4		*g*	
	AFS_SEL=2		±8		*g*	
	AFS_SEL=3		±16		*g*	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*	
	AFS_SEL=1		8,192		LSB/*g*	
	AFS_SEL=2		4,096		LSB/*g*	
	AFS_SEL=3		2,048		LSB/*g*	
Initial Calibration Tolerance			±3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			±2		%	

Figure 15: Accelerometer sensitivity specification.

Reference : MPU6050 datasheet/Pg13

- All these values are written to the sensor register during the initialization sequence of the sensor.
- For reading the sensor values a single byte I2C write and multi-byte I2C read operation is performed. The frame of a typical transaction is given below.

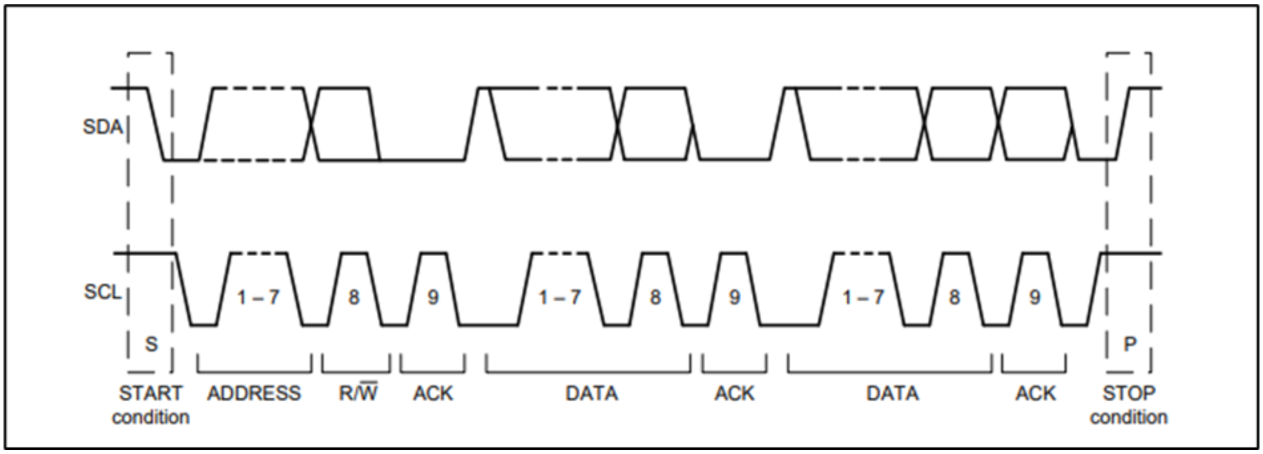


Figure 16: I2C transaction for MPU6050.

Reference : MPU6050 datasheet/Pg35

- As it can be seen above, the start sequence is followed by the 7-bit address of the register which needs to be read or written on. The next bit is the Read/Write(bar) bit. On receiving this first byte, the sensor sends an acknowledgement by pulling the SDA line high. For the write operation, the next byte is the data after which the sensor again sends an acknowledgement. For a multi byte read operation like reading the acceleration values, the sensor sends a byte which the master(MSP432)acknowledges after which the sensor sends the next byte of data. This process continues until the master sends a STOP condition and the transaction ends. This I2C bus can be operated at a maximum bit-rate of 400Khz.

- The gyroscope values are read from the sensor by reading from the registers at address 0x43 to 0x48. The corresponding 3 axis upper and lower bytes are stored in variables in the formats as given below.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

Figure 17: Gyroscope registers.

Reference : MPU6050 register map/P31

- Similarly for the accelerometer, the values from register 0x3B to 0x40 are read for getting 3 axis acceleration values. These are read in the variables as given in the below format.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Figure 18: Accelerometer registers.

Reference : MPU6050 register map/P29

- These values received from the I2C are in raw format, hence conversion is performed over them.

2.2.2 Gyroscope & accelerometer calculation

- The first step of the conversion is getting the raw sensor value of all the three axes in degrees per second. This can be done by dividing the raw sensor value by the gyroscope sensitivity factor of 65.5 discussed earlier. Now, there is an option to convert these values to angle, required in most applications. This is done by integrating the value over time, i.e multiplying the previous value by loop time of the application and accumulating all the samples in every cycle. For our application, we simply need the instantaneous angular velocity values in degrees per second. This gives us: sensor value on an axis / 65.5 = sensor value on an axis * 0.0152671.

```
//Gyro angle calculations
angle_pitch  = gyro_x * 0.0152671;
angle_roll   = gyro_y * 0.0152671
angle_yaw = gyro_z * 0.0152671;
```

Reference : src/mpu.c

- The accelerometer value can also be found for three axes by dividing the raw value by the sensitivity scale factor of the accelerometer of 4096 discussed above. This value is then multiplied by the earth's gravitational constant of 9.8m/sec^2 rounded off to 10 to get acceleration values in m/sec^2. During our testing, slight error was observed in the acceleration value of the z axis. The value on this axis was slightly less than 4096 even when the sensor was at rest and in this case, it should have been 4096 i.e 1g should always be present in this axis due to earth's gravitational force. Hence we added a factor of 1 to offset the sensor value in the correct range.

```
//Calculating acceleration values
accl_x = (int16_t) (acc_x/4096) * 10;//in m/s^2
accl_y = (int16_t) (acc_y/4096) * 10;
accl_z = (int16_t) ((acc_z/4096)+1) * 10;
```

Reference : src/mpu.c

```c
void process_raw_values(int16_t accelerometer[3], int16_t gyro[3])
{
    int16_t accl_x,accl_y,accl_z;

    /*Output of gyro is in degree/sec, hence for FS=1, Gyro sensitivity scale factor = 65.5
     * therefore 1 degree/sec = 65.5 or rawGyroVal/65.5 = degree/sec*/
    //reading gyro raw values
    gyro_x = gyro[0];
    gyro_y = gyro[1];
    gyro_z = gyro[2];

    /*Output of accelerometer for AFS_SEL=2 is 4096/g, i.e 1g=4096, therefore rawAcclVal/4096=9.8m/s^2(1g)*/
    //reading acceleromter values
    acc_x = accelerometer[0];
    acc_y = accelerometer[1];
    acc_z = accelerometer[2];

    //Compensating current values with the calibration values
    gyro_x -= gyro_x_cal;
    gyro_y -= gyro_y_cal;
    gyro_z -= gyro_z_cal;

   //Gyro angle calculations
   angle_pitch  = gyro_x * 0.0152671;
   angle_roll   = gyro_y * 0.0152671;
   angle_yaw = gyro_z * 0.0152671;

   //Calculating acceleration values
   accl_x = (int16_t) (acc_x/4096) * 10;//in m/s^2
   accl_y = (int16_t) (acc_y/4096) * 10;
   accl_z = (int16_t) ((acc_z/4096) + 1) * 10;

   //storing the values
   gAccelero_t.gGyroVal[0] = angle_roll;//x - roll
   gAccelero_t.gGyroVal[1] = angle_pitch;//y - pitch
   gAccelero_t.gGyroVal[2] = angle_yaw;//z - yaw.
   gAccelero_t.gAcclVal[0] = accl_x;//x - roll
   gAccelero_t.gAcclVal[1] = accl_y;//y - pitch
   gAccelero_t.gAcclVal[2] = accl_z;//z - yaw

}
```

Figure 19: Complete sensor value conversion code.

Reference : src/mpu.c

- In the above code, the current reading gyroscope values are getting subtracted from the calibration values and in the end the 6 values of gyroscope and accelerometer reading are getting stored in the sensor value structure.

2.2.3 Calibration process

```
P2->OUT |= BIT0;//to indicate start of calibration

for (i = 0; i < 2000 ; i++ ){
    MPU6050_ReadData(accelerometer, gyro, &temp);
    gyro_x_cal += gyro[0];
    gyro_y_cal += gyro[1];
    gyro_z_cal += gyro[2];
}

gyro_x_cal /= 2000;
gyro_y_cal /= 2000;
gyro_z_cal /= 2000;


P2->OUT ^= BIT0;//turn the led off, to indicate end of calibration
```

Figure 20: Calibration code.
Reference : src/mpu.c

- For the calibration process, the gyroscope readings are first integrated or accumulated for two-thousand samples. These values are then averaged by dividing from 2000. These calibration constants are saved and later compensated with current values as shown in the previous section. It is necessary to perform the calibration process every time as the initial position of the sensor is going to be different at every startup and it is necessary to set a reference at every axis. Hence it is not required to save these calibration constants in the non-volatile memory in our application.

2.2.4 SPI protocol: SD Card interface

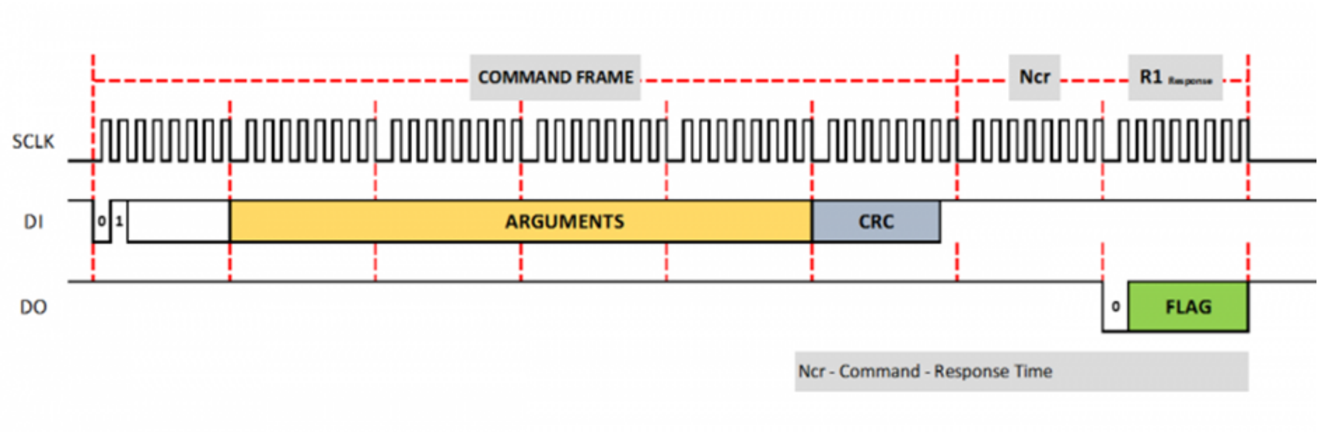


Figure 21: An SPI command frame for SD card.

Reference : source.

- The total SD card frame consists of 6 bytes of which the first byte in the command name(for example, CMD0 or CMD55) followed by 4 bytes of argument which could be the address of the location to be read or written on. The last byte is the CRC(cyclic redundancy check byte).

- The response from the SD card can be of different types like R1, R3, R7, R1b. An R1 response frame is given below.

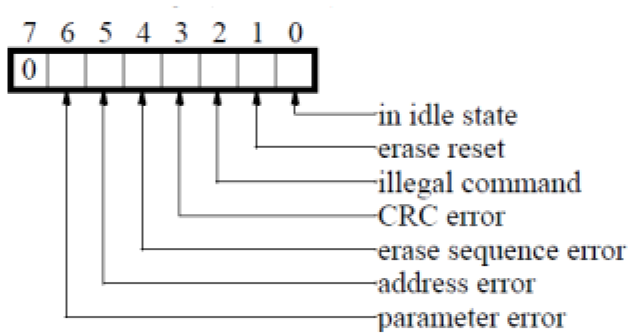


Figure 22: An R1 response packet.

Reference : [source].

- For an R1 response if a 0x01 is received means that the card has gone into idle state after receiving the previous command. A 0x00 indicates that the command has been accepted with no error and if any values other than 0x00 or 0x01 has been received it indicates the error condition specified by the bit set in the R1 bits 1 to 6.

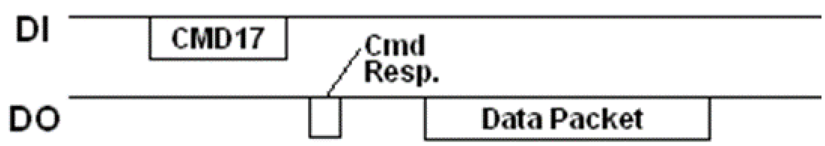


Figure 23: A single byte read frame on SD card.

Reference : [source].

- For reading a single byte from the SD card, a command 17 is sent for which the card provides a response after accepting the command followed by the read block which is received on the host.

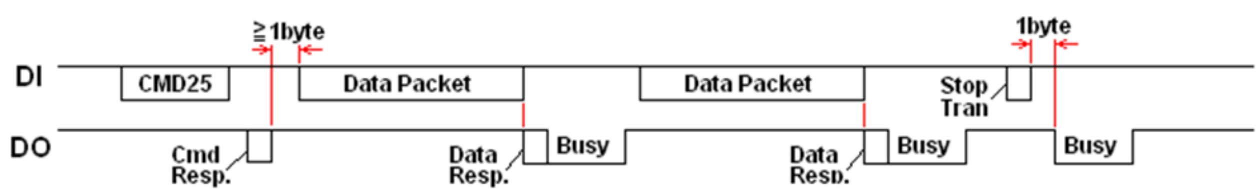


Figure 24: A multi byte write frame on SD card.

Reference : [source].

- For writing multiple blocks of data to the SD card, a command 25 is sent to the card and after the SD card accepts the command by sending a response, the consecutive data blocks are sent. The write operation continues until a stop token is sent. After every data block and stop token, a busy flag is output indicated by the DO line being pulled low by the SD card data line DO. The number of bytes to be written can be pre-defined by sending a CMD23 prior to CMD25 and the writing operation stops after the SD card has received the last byte without the stop token.

2.2.5 FAT File System

FatFs is a generic FAT/exFAT filesystem module for small embedded systems. The FatFs module is written in compliance with ANSI C (C89) and completely separated from the disk I/O layer. Therefore it is independent of the platform on which it is used. It can be incorporated into small microcontrollers with limited resources, thus is an asset to the embedded system domain.

It is compatible with Windows and DOS operating systems and is easy to port from one system to another. It has a very small footprint for program code. It has numerous configuration options which are present in the ffconf.h file.

Examples

- When FF_USE_STRFUNC macro is defined to a non zero value, the f_gets function is enabled
- Multiple volumes can be enabled on the SD Card
- Long file names can be used in ANSI or Unicode

The driver library for FATFS commands can be downloaded from here. All the FAT commands are present in the ff.c file which we have not included as a part of this report.

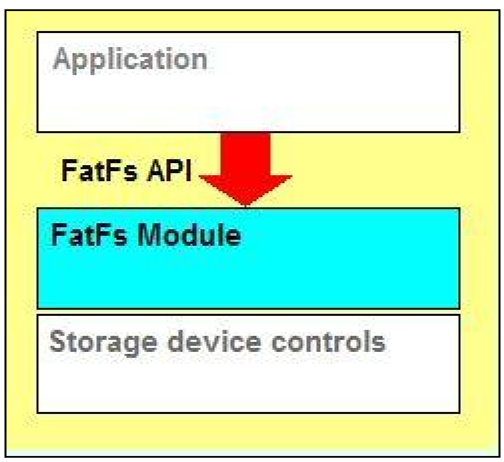


Figure 25: FATFs

Reference : source

2.3 Software Design

2.3.1 Software Components

We have used the following software tools for this project.

1. **Code Composer Studio**

Code Composer Studio 6.2.0 is an integrated development environment (IDE) that supports TI's Microcontroller(MSP432P401R) and Embedded Processors portfolio. Code Composer Studio comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow. Code Composer Studio combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers.
Reference : link

2. **KiCad**

KiCad is a free software suite for electronic design automation. It facilitates the design and simulation of electronic hardware. It features an integrated environment for schematic capture, PCB layout, manufacturing file viewing, SPICE simulation, and engineering calculation. We can design our own symbols and footprints for components using KiCad.
Reference : link

2.3.2 Program Flow

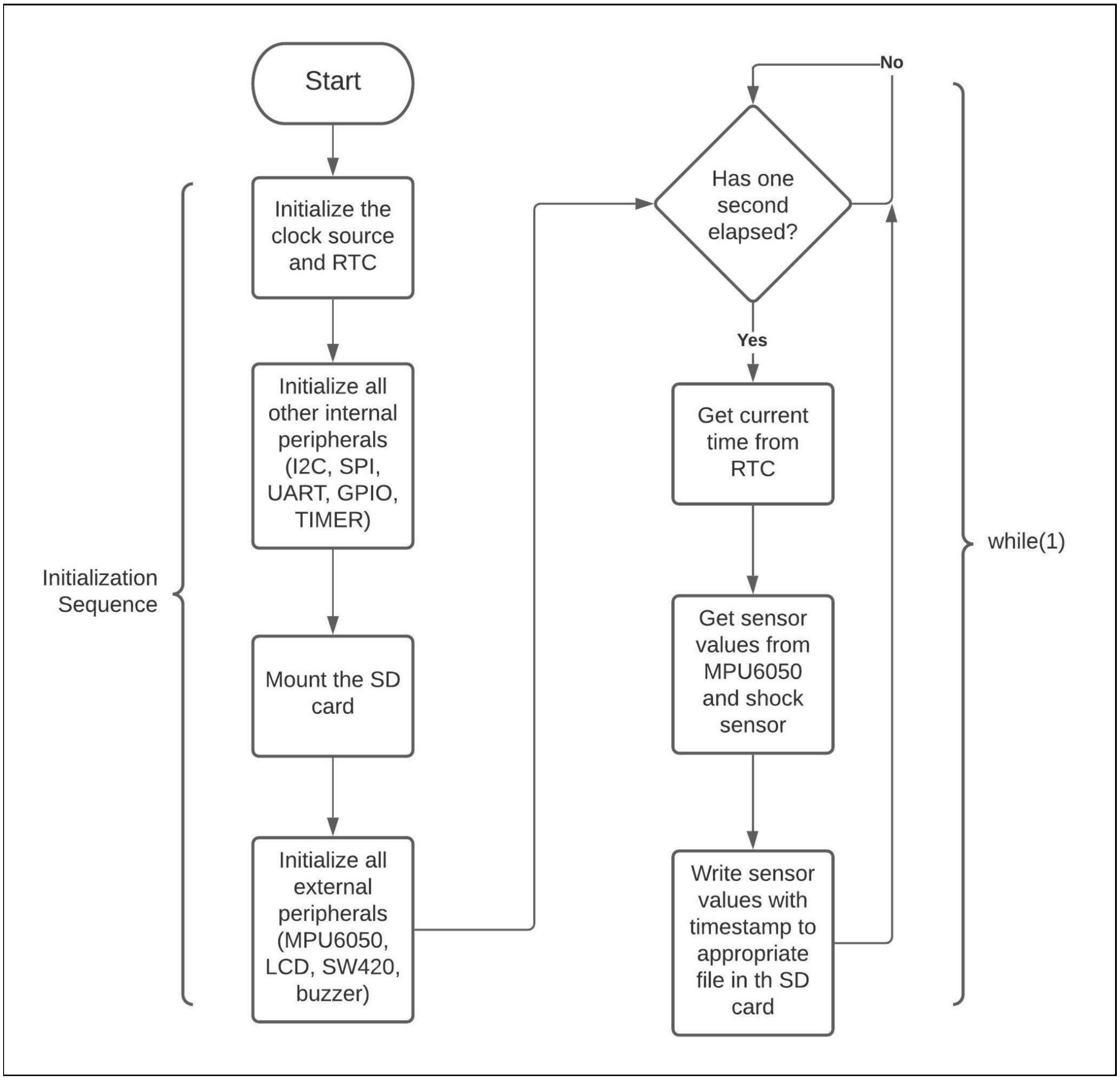


Figure 26: Software Flowchart of the application

```c
DWORD str=0;
str = get_fattime();
RTC_C_Calendar start_time =
{
     (str & 0x0000001F)*2,
     (str & 0x000007E0)>>5,
     (str & 0x0000F800)>>11,
     3,
     (str & 0x001F0000)>>16,
     (str & 0x01E00000)>>21,
     ((str & 0xFE000000)>>21)/16 + 1980 // epoch has been set as 1980

};

CS_Init();
RTC_init(start_time);

/*Initialize all hardware required for the SD Card*/
SPI_Init(EUSCI_B0_BASE, SPI0MasterConfig);
UART_Init(EUSCI_A0_BASE, UART0Config);
GPIO_Init(GPIO_PORT_P5, GPIO_PIN0);
TIMERA_Init(TIMER_A1_BASE, UP_MODE, &upConfig, disk_timerproc);

Interrupt_enableMaster();
```

Figure 27: Code snippet for peripheral initialization

Reference: src/main.c

Just like any other programs, our application starts by initializing the clock for the system. We take the system's current time to initialize the RTC module. Hence we don't have to synchronize the RTC clock with the world clock manually every single time. We then initialize the internal peripherals of the MSP432 including SPI, UART, GPIO, TimerA and I2C. We enable the global interrupts to make efficient use of the system's CPU.

```c
void my_file_write(FIL fp, char* filename, const void* buff, UINT btw)
{
    FRESULT fr;
    UINT bw;
    fr = f_open(&fp, filename, FA_WRITE);
    if(fr != FR_OK)
    {
        MSPrintf(EUSCI_A0_BASE, "Error opening file/directory\r\n");
        while(1);
    }

    fr = f_write(&fp, buff, btw, &bw);
    if(fr != FR_OK)
    {
        MSPrintf(EUSCI_A0_BASE, "Error writing to  file/directory\r\n");
        while(1);
    }

    f_close(&fp);
}
```

Figure 28: Code snippet for wrapper function of f_write

Reference: src/my_file_func.c

The next step is to mount the SD Card. For all SD card operations, we have used the FATFs commands like (f_mount, f_read, f_write, f_stat etc) After these commands we always check if the command worked successfully or not. This is done using the return value of the command. We have written wrapper functions for the above FATFs commands which are present in the “my_file_func.c” file. This is done to avoid duplicating the code statements in the main loop.

Once the SD card is mounted properly, we run a directory scan to go through all the directories and print the folder structure. This shows us how much data space is already used in the SD card. We then initialize all the external sensors and setup their interrupts wherever needed. Calibration is performed on the MPU6050 module to set its initial position. We print a welcome string “ESD Project by Sanish and Vishal” on the LCD during its initialization sequence. This completes the initialization.

```
while(1)
{
    if(one_second_elapsed() == true)
    {

        time = RTC_get_current_time();

        sensor = get_mpu_values();
        update_SD_card(time,sensor);
        clear_one_second_elapsed_flag();
        clear_shock_detected_flag();

    }
}
```

Figure 29: Code snippet for while(1)

Reference: src/main.c

In the main loop we continuously monitor if one second has elapsed. This is done because we are recording data into the SD card at an interval of one second. Every one second we get the current time and read the sensor values of MPU6050. These values are then passed to the update SD card function The vibration sensor works on an interrupt mechanism and hence if a vibration was detected in that one second, a variable has already been updated and ready to write into the SD card.

Inside the update SD card function, we have a static variable to keep track of the old time. This is done to check which parameter from the RTC (year, month, date, hour, minute, second) has changed from last time. If a shock was detected in the last second, the buzzer beeps twice. Since this is a data logger that will be used in automotive and aerospace vehicles, the amount of data gathered over the years would be huge. Hence it is important that we store it in a systematic way. We have used a folder structure based on the timestamp of the data being recorded. A new folder will be created for each year and sub folders for each month, date, day and hour will be created subsequently.

```c
/* If date changed, create a new directory in the correct path */
if(ntime.dayOfmonth != old_time.dayOfmonth)
{

    sprintf(final_path, "/%s/%s/",path[YEAR], path[MONTH]);

    f_chdir(final_path);
    f_mkdir(path[DATE]);
    f_chdir(path[DATE]);

    f_getcwd(read_buffer, sizeof(read_buffer));
    MSPrintf(EUSCI_A0_BASE, read_buffer);

    memset(final_path, 0, sizeof(final_path));

}
```

Figure 30: Code snippet when a day changes

Reference: src/sd_card.c

For example-

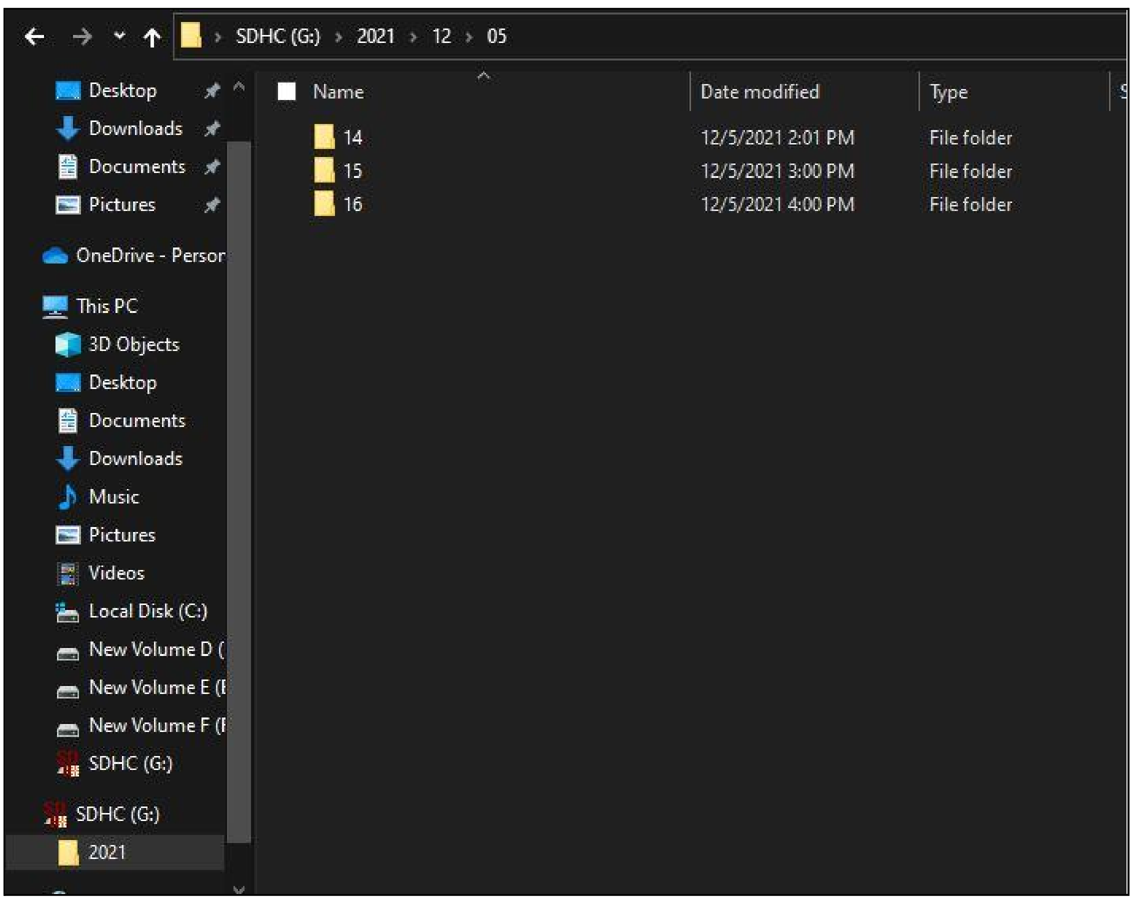


Figure 31: date folder on SD card

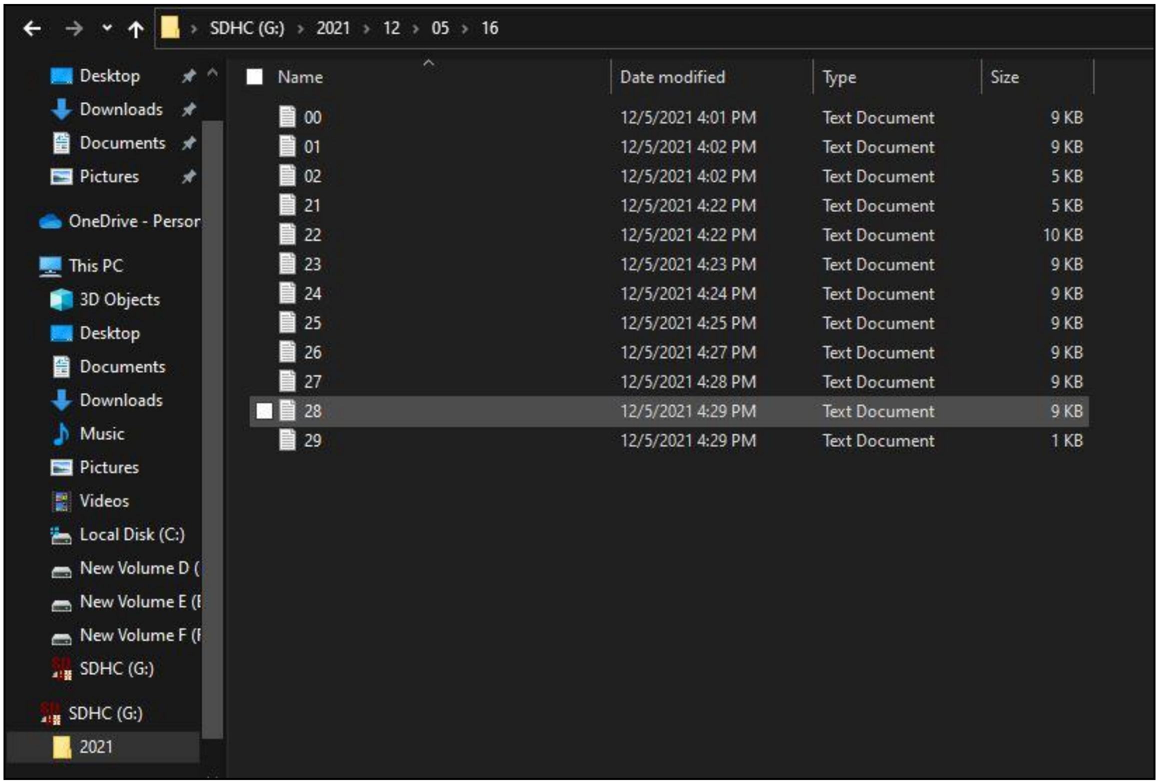
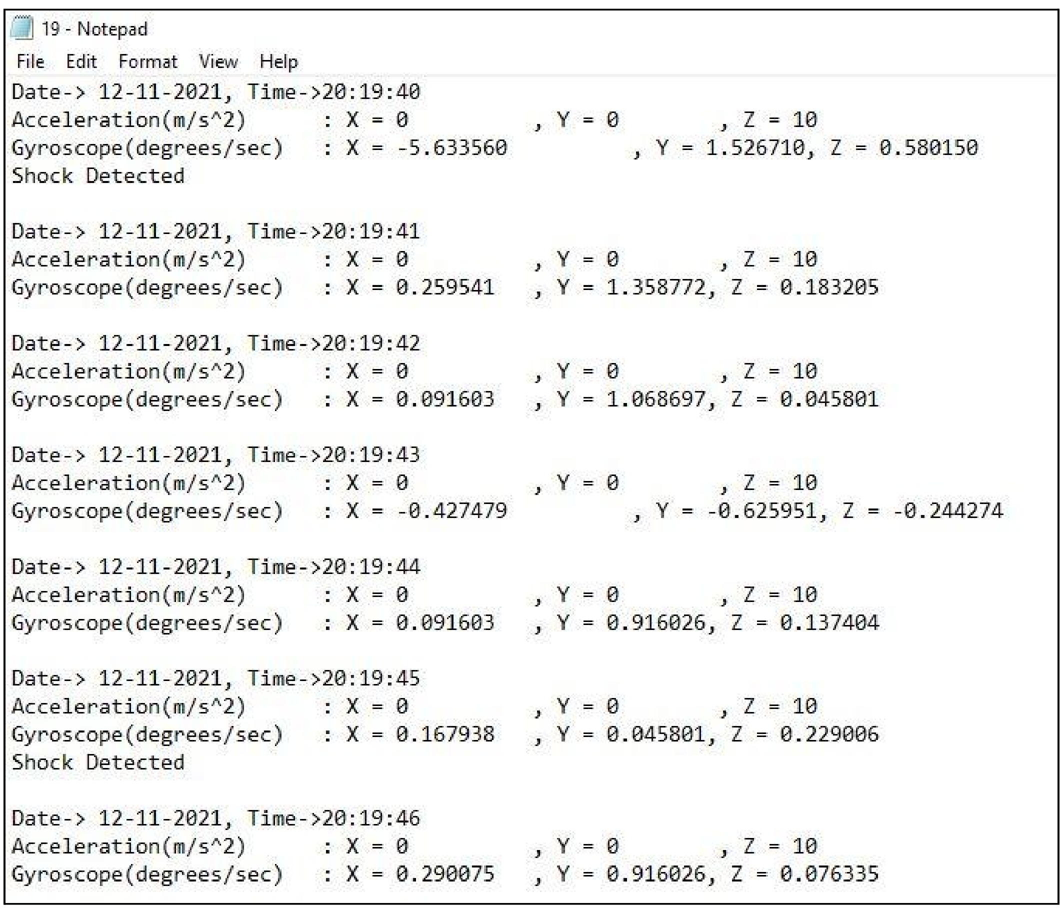


Figure 32: hour folder on SD card


19 - Notepad

File Edit Format View Help

```
Date-> 12-11-2021, Time->20:19:40
Acceleration(m/s^2)      : X = 0          , Y = 0          , Z = 10
Gyroscope(degrees/sec)   : X = -5.633560          , Y = 1.526710, Z = 0.580150
Shock Detected

Date-> 12-11-2021, Time->20:19:41
Acceleration(m/s^2)      : X = 0          , Y = 0          , Z = 10
Gyroscope(degrees/sec)   : X = 0.259541   , Y = 1.358772, Z = 0.183205

Date-> 12-11-2021, Time->20:19:42
Acceleration(m/s^2)      : X = 0          , Y = 0          , Z = 10
Gyroscope(degrees/sec)   : X = 0.091603   , Y = 1.068697, Z = 0.045801

Date-> 12-11-2021, Time->20:19:43
Acceleration(m/s^2)      : X = 0          , Y = 0          , Z = 10
Gyroscope(degrees/sec)   : X = -0.427479          , Y = -0.625951, Z = -0.244274

Date-> 12-11-2021, Time->20:19:44
Acceleration(m/s^2)      : X = 0          , Y = 0          , Z = 10
Gyroscope(degrees/sec)   : X = 0.091603   , Y = 0.916026, Z = 0.137404

Date-> 12-11-2021, Time->20:19:45
Acceleration(m/s^2)      : X = 0          , Y = 0          , Z = 10
Gyroscope(degrees/sec)   : X = 0.167938   , Y = 0.045801, Z = 0.229006
Shock Detected

Date-> 12-11-2021, Time->20:19:46
Acceleration(m/s^2)      : X = 0          , Y = 0          , Z = 10
Gyroscope(degrees/sec)   : X = 0.290075   , Y = 0.916026, Z = 0.076335
```

Figure 33:sensor data in minute file

If the month changed from November 2021 to December 2021, a new folder named “12” will be created inside the “2021” directory. Within this folder a new folder named “00” will be created initially and a new folder for each hour will be created subsequently. Each hour folder will consist of 60 files one for each minute and each minute file will consist of 60 entries one for each second. Each such entry has the timestamp with date and time, and the data of accelerometer, gyroscope and shock sensor.

We are continuously displaying the RTC time along with the sensor readings on the LCD display. We have configured the button on the MSP432 (P1.4) to switch the display mode from accelerometer to gyroscope. The RTC time is always updated every second. After updating the sensor readings in the SD card, we clear the flags and repeat the cycle every second. In this way real time data gets logged into the SD card.

2.4 Testing Process

For validating the working of our system, we have executed various testing strategies like individual module testing and complete system testing. In individual module testing, we have tested the the individual software modules and the various hardware modules to verify their correctness. Various test cases and inputs are given to the specific module. For example, for the accelerometer various values of angular motion and acceleration are given across different axis and its values are verified on the uart terminal. For the different protocols, the individual frames for read and write are analyzed on the logic analyzer and the logic port utility and verfied against the specific component datasheet. For the SD card, the whole system is made to run for extended period of time and then the logged outputs are checked if they are stored in their right format, in specific folder according to their date and time for every second.

The complete system is tested by integrating all the software and hardware modules and re-testing the functionality of multiple or a combination of modules like testing the LCD for printing the current sensor value every 1 second tick given by the RTC and also verifying the trigger of buzzer when the impact reaches more than a set threshold. All the logged data is again verified by analysing the folder structure and the various files written on the SD card.

3. Results & Error Analysis

3.1 Results

We successfully developed the hardware for the vibration sensor circuit. We were able to adjust the threshold according to our requirement so that the sensor doesn't give false alarms. We interfaced the MPU6050 sensor via the I2C protocol with MSP432. We obtained accurate readings for 3 axes acceleration and 3 axes angular velocity from this sensor. We thoroughly tested the sensor for different kinds of movements to ensure it works in all cases. We successfully interfaced the SD card with MSP432 using the SPI protocol. We used a FAT32 file system on the SD card for file management. We used MSP432's RTC to keep track of time and save it into the SD card along with the sensor readings. We displayed the sensor readings and the live time on a 16x2 LCD display. We used interrupt mechanisms for all the modules thus making efficient use of the system's CPU. Finally, we conducted thorough testing of all the components individually and as a system to ensure that there are no failures. In this way we were able to achieve all the goals proposed at the beginning of the project.

3.2 Error Analysis

One of the errors we encountered during the project was that we were missing out on the last byte of data during I2C read of MPU6050. We realized that this byte was sent over the wire but was being recorded in the next reading cycle. Hence we modified the software so that we read the last byte just before the stop sequence of the I2C.

4. Conclusions & Lessons Learned

In this project we were able to design a data logging system for automotive and aerial vehicles which consisted of the SD card as the main storage device. We were able to understand and implement the working of an SD card on the SPI protocol and after the communication link was established between the SD card and the host controller, we implemented a file system necessary to store files on the storage device in the required folder structure such that it is easy to fetch the sensor data according to date and time from the various folders. Two sensors were interfaced necessary to detect jerks or sudden change in movement experienced in any vehicle with one being the accelerometer/gyroscope and the other being the vibration sensor. On implementing the sensor interface we were able to understand the methods of reading the sensor values and converting these in their expected units. By making one of the sensors ourselves we understood how the hardware circuit can be tuned for the sensor to give an output in the required range and how calibration of these sensor values can be performed. The various output elements of the project were the LCD display and the piezoelectric buzzer. Here, the main learning was the implementation of code for communicating with the LCD using I2C and also choosing and analyzing the various drive capabilities of MSP432 for driving these output peripherals.

The other challenges we faced helped us understand the various debugging techniques both in the hardware and software aspect. For instance, on the software side, not being able to communicate with a peripheral helped us understand the possible problems which might exist with a communication protocol and how we can use various analysis techniques to identify the problem. On the hardware side it can be done by using an oscilloscope or a logic analyzer to see the signal parameters like voltage & timing characteristics.

Overall, we had a great learning experience as we were able to design a complete system consisting of multiple hardware and software modules and also integrate and test them such that all these elements interact with each other synchronously.

5. Future Scope

There are multiple ways in which this project can be taken forward.

5.1 Real-time feedback for guidance

Currently the system is logging the data received from sensors into the SD card at an interval of 1 second. This data can be used to reduce accidents. We can check at what accelerometer and gyroscope values shocks were detected previously and provide real time guidance to the user if he's approaching the numbers where the risk increases.

5.2 Cloud Storage

Since the SD card could be damaged in case of an accident, we can regularly backup the SD card data onto the cloud. This data can then be deleted from the SD card thus allowing us to use an SD card of smaller size. This will ensure the data is securely stored and can be used for further analysis and guidance.

5.3 CAN bus protocol

We can use the CAN bus protocol to interface this system with other components of the vehicle. The buzzer can be replaced by a live notification system which prompts the emergency departments in case of an accident. Thus help can be provided instantly without any delays.

6. Acknowledgement

We feel privileged to thank our professor Dr. Linden McClure, for providing all the support needed to successfully complete the project. His persistent encouragement, support, efforts and guidance has helped us throughout our project.
We would also like to thank the teaching assistants Sundar Krishnakumar, Alex Fritz and Venkat Tata for helping us debug our hardware and software problems throughout the course of this project.

7. Division of Labour

Vishal Raj	Sanish Kharade
MPU6050 (accelerometer and gyroscope) interfacing over I2C	Hardware circuit development and GPIO interrupt program for vibration sensor
16x2 LCD interfacing over I2C	SD Card interfacing using SPI
Buzzer interfacing using GPIO	Data storage using FATFs file system commands and RTC
Software code integration and final hardware board development	
Final Project Video Demonstration and Report	

Table 1: Work Distribution

8. References

References for hardware, written content and images have been given at their respective places above. Reference for any leveraged code has been mentioned in the respective file.

1. Accelerometer value conversion.
2. I2C LCD interfacing.
3. SD card SPI protocol.
4. FATFs interfacing
5. Datasheets of MPU6050, SD Card, LCD, MSP432P401R

9. Appendices

9.1 Bill of Materials

Qty	Description	Part Number	Price
Vibration Sensor			
1	LM339	296-39009-5-ND	$1.72
1	10k Potentiometer	3362P-1-304LF-ND	$0.48
2	Resistor CFR 1k 0.25W	CFR-25JB-52-1K	$0.1
2	Resistor CFR 10k 0.25W	CFR-25JB-52-10K	$0.1
1	Led, Green	754-1731-ND	$0.1
1	Led, Red	754-1870-ND	$0.1
1	SW-420 sensor	A19042700UX1073	$0.42
2	0.1uF Monolithic Ceramic Capacitor(>=10V)	C322C104K3G5TA	$0.2
Peripherals			
1	Weichuang DC 3-24V 90dB buzzer	Amazon	$1.66
1	Sunfounder I2C TWI 1602 serial LCD	CN0295	$8.99
1	Sandisk Ultra 16GB Class 10 Micro SDHC.	SDSQUNS-016G-GN3MN	$6.39
1	Hiletgo Micro SD TF Card Reader 6Pin SPI	3-01-0038-5PCS	$1.39

1	Hiletgo GY-521 MPU-6050	3-01-0122	$3.33
MCU & Board			
1	MSP-EXP432P401R	296-39653-ND	$23.99
1	4x6 Inches General Purpose Zero PCB	SKU:PCB011	$0.59
Total			$49.56

Table 2: Bill of Materials

9.2 Schematic

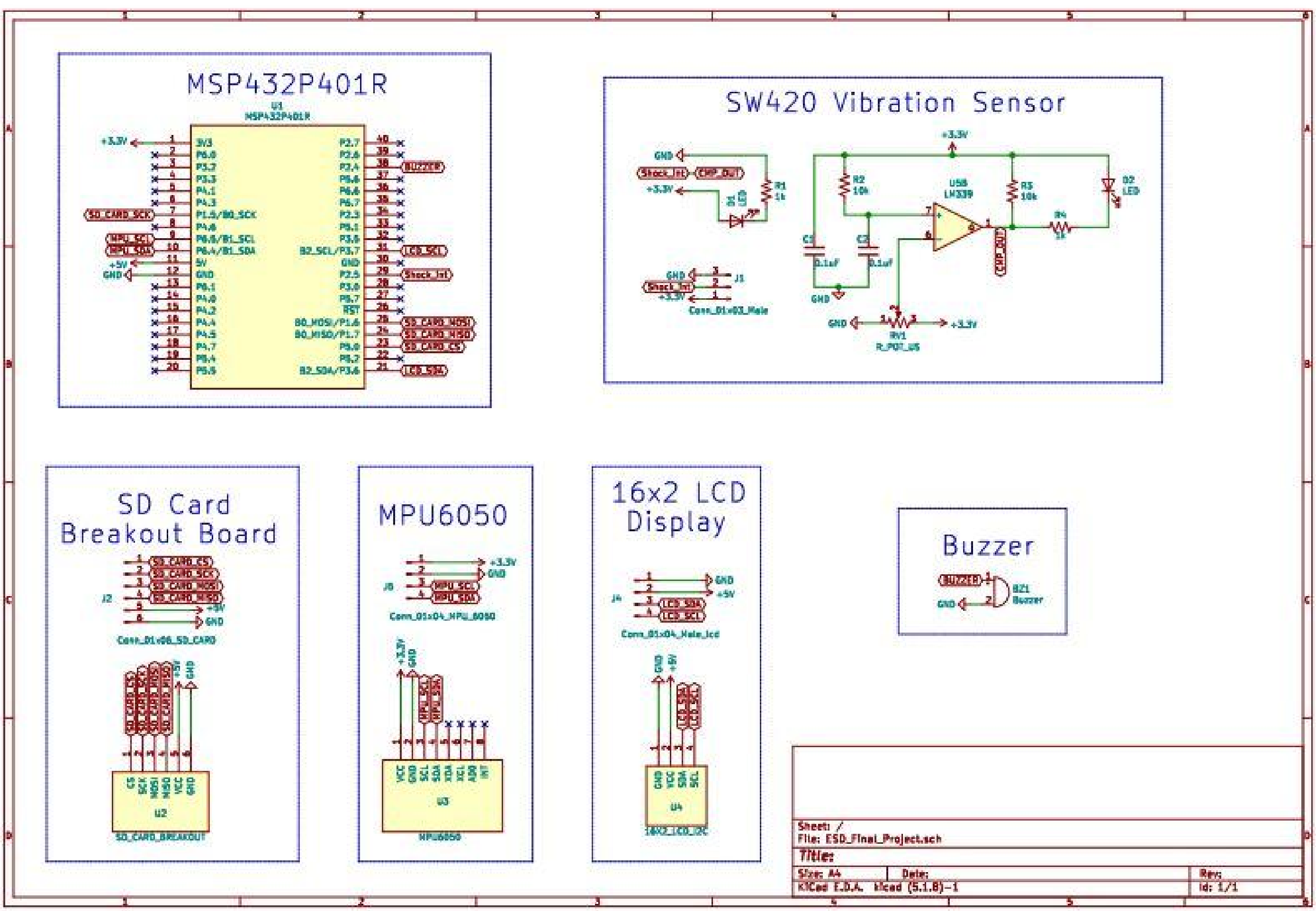


Figure 34: Schematic

9.3 Code

Github URL - https://github.com/sanishkharade/Data-Logger-for-automotive-and-aerial-veihicles

Code Contribution by Sanish Kharade - 1088 lines

Code Contribution by Vishal Raj - 1044 lines

9.3.1 main.c

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2017, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
```

```
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * --/COPYRIGHT--*/
/******************************************************************************** */
/*
 * Project  - Data Logger for Automotive and Aerial Vehicles
 * Members  - Sanish Kharade and Vishal Raj
 *          University of Colorado Boulder
 * Course   - Embedded System Design
 *
 * Comments    - All function descriptions are present in header files.
 *
 * */
/**
 * @file    :   main.c
 *
 * @brief   :   Main file
```

```
 *
 * @author  :   Sanish Kharade
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   -
*/
/* Standard Defines */
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
/* DriverLib Defines */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#include <Hardware/SPI_Driver.h>
#include <Hardware/GPIO_Driver.h>
#include <Hardware/CS_Driver.h>
#include <Hardware/TIMERA_Driver.h>
#include <fatfs/ff.h>
#include <fatfs/diskio.h>
#include <Devices/MSPIO.h>

#include <inc/my_file_func.h>
```

```
#include <inc/rtc.h>

#include <inc/sd_card.h>

#include <inc/sw420.h>

#include <inc/lcd.h>

#include <inc/buzzer.h>

#include <inc/lcd.h>

#include <inc/mpu.h>

#include <inc/mode.h>


/* Slave Address for I2C Slave */

#define SLAVE_ADDRESS       0x68


/* Variables */

const uint8_t TXData[] = {0x04};

uint8_t RXData[NUM_OF_REC_BYTES];

static volatile uint32_t xferIndex;

static volatile bool stopSent;

volatile bool read_done = false;

/* I2C Master Configuration Parameter */

const eUSCI_I2C_MasterConfig i2cConfig =

{

    EUSCI_B_I2C_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source

    3000000,                                // SMCLK = 3MHz

    EUSCI_B_I2C_SET_DATA_RATE_100KBPS,      // Desired I2C Clock of 100khz

    0,                                      // No byte counter threshold
```

```
    EUSCI_B_I2C_NO_AUTO_STOP            // No Autostop
};
/*Huge thanks to bluehash @ https://github.com/bluehash/MSP432Launchpad/tree/master/MSP432-Launchpad-FatFS-SDCard*/


/* UART Configuration Parameter. These are the configuration parameters to
 * make the eUSCI A UART module to operate with a 115200 baud rate. These
 * values were calculated using the online calculator that TI provides
 * at:
 * http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html
 */
eUSCI_UART_ConfigV1 UART0Config =
{
    EUSCI_A_UART_CLOCKSOURCE_SMCLK,
    13,
    0,
    37,
    EUSCI_A_UART_NO_PARITY,
    EUSCI_A_UART_LSB_FIRST,
    EUSCI_A_UART_ONE_STOP_BIT,
    EUSCI_A_UART_MODE,
    EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION
};
/* SPI Configuration Parameter. These are the configuration parameters to
 * make the eUSCI B SPI module to operate with a 500KHz clock.*/
eUSCI_SPI_MasterConfig SPI0MasterConfig =
```

```
{
    EUSCI_B_SPI_CLOCKSOURCE_SMCLK,
    3000000,
    500000,
    EUSCI_B_SPI_MSB_FIRST,
    EUSCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT,
    EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH,
    EUSCI_B_SPI_3PIN
};
/* Timer_A UpMode Configuration Parameters */
Timer_A_UpModeConfig upConfig =
{
      TIMER_A_CLOCKSOURCE_SMCLK,              // SMCLK Clock Source
      TIMER_A_CLOCKSOURCE_DIVIDER_64,         // SMCLK/1 = 3MHz
      30000,                                  // 1 ms tick period
      TIMER_A_TAIE_INTERRUPT_DISABLE,         // Disable Timer interrupt
      TIMER_A_CCIE_CCR0_INTERRUPT_ENABLE ,    // Enable CCR0 interrupt
      TIMER_A_DO_CLEAR                        // Clear value
};
FATFS FS;
DIR DI;
FILINFO FI, FILEINFO;
/*******************************************************************************
 * @function:   main
 *
```

```
 * @brief  :  Main entry point to the application
 *
 * @param  :  none
 *
 * @return :  void
 *
*********************************************************************************/
void main(void)
{
   WDT_A_holdTimer();


   DWORD str=0;
   str = get_fattime();
   RTC_C_Calendar start_time =
   {
       (str & 0x0000001F)*2,
       (str & 0x000007E0)>>5,
       (str & 0x0000F800)>>11,
       3,
       (str & 0x001F0000)>>16,
       (str & 0x01E00000)>>21,
       ((str & 0xFE000000)>>21)/16 + 1980 // epoch has been set as 1980
       //>>25 was not allowed for year field. hence >>21 and divide by 16
   };
   CS_Init();
```

```
RTC_init(start_time);


/*Initialize all hardware required for the SD Card*/
SPI_Init(EUSCI_B0_BASE, SPI0MasterConfig);
UART_Init(EUSCI_A0_BASE, UART0Config);
GPIO_Init(GPIO_PORT_P5, GPIO_PIN0);
TIMERA_Init(TIMER_A1_BASE, UP_MODE, &upConfig, disk_timerproc);


Interrupt_enableMaster();
char startup_buff[100];
 sprintf(startup_buff, "Startup - Date-> %02d-%02d-%04d, Time-> %02d:%02d:%02d",  start_time.month,
start_time.dayOfmonth, start_time.year,
                                        start_time.hours, start_time.minutes, start_time.seconds);
MSPrintf(EUSCI_A0_BASE, startup_buff);
FRESULT r;
/* Mount the SD Card into the Fatfs file system*/
r = f_mount(&FS, "0", 1);
if(r != FR_OK)
{
   MSPrintf(EUSCI_A0_BASE, "Error mounting SD Card, check your connections\r\n");
   while(1);
}
/* Open the root directory on the SD Card*/
r = f_opendir(&DI, "/");
if(r != FR_OK)
{
```

```
    MSPrintf(EUSCI_A0_BASE, "Could not open root directory\r\n");
    while(1);
}
/*Read everything inside the root directory*/
do
{
    /*Read a directory/file*/
    r = f_readdir(&DI, &FI);
    if(r != FR_OK)
    {
        MSPrintf(EUSCI_A0_BASE, "Error reading file/directory\r\n");
        while(1);
    }
    /*Print the file to the serial terminal*/
    MSPrintf(EUSCI_A0_BASE, "%c%c%c%c%c %s\r\n",
            (FI.fattrib & AM_DIR) ? 'D' : '-',
            (FI.fattrib & AM_RDO) ? 'R' : '-',
            (FI.fattrib & AM_HID) ? 'H' : '-',
            (FI.fattrib & AM_SYS) ? 'S' : '-',
            (FI.fattrib & AM_ARC) ? 'A' : '-',
            ((char*)FI.fname));

}while(FI.fname[0]);
RTC_C_Calendar time;
char read_buf[70]={0};
```

```
//f_getcwd(read_buf, sizeof(read_buf));
MSPrintf(EUSCI_A0_BASE, read_buf);
SW420_gpio_init();
mode_gpio_init();
/////////////////////////////////////MPU6050/////////////////////////////////////

/* Select Port 6 for I2C - Set Pin 4, 5 to input Primary Module Function,
 *   (UCB0SIMO/UCB0SDA, UCB0SOMI/UCB0SCL).
 */
MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6,
        GPIO_PIN4 + GPIO_PIN5, GPIO_PRIMARY_MODULE_FUNCTION);
stopSent = false;
memset(RXData, 0x00, NUM_OF_REC_BYTES);

/* Initializing I2C Master to SMCLK at 100khz with no autostop */
MAP_I2C_initMaster(EUSCI_B1_BASE, &i2cConfig);

/* Specify slave address */
MAP_I2C_setSlaveAddress(EUSCI_B1_BASE, SLAVE_ADDRESS);
/* Enable I2C Module to start operations */
MAP_I2C_enableModule(EUSCI_B1_BASE);
MAP_Interrupt_enableInterrupt(INT_EUSCIB1);
/////////////////////////////////////MPU6050/////////////////////////////////////
gAcc_gyro sensor;
i2c_init();
```

```
    lcd_i2c_init();
    MPU6050_Reset();
    while(1)
    {
        if(one_second_elapsed() == true)
        {
            time = RTC_get_current_time();
            sensor = get_mpu_values();
            update_SD_card(time,sensor);
            clear_one_second_elapsed_flag();
            clear_shock_detected_flag();
        }
    }
}
/******************************************************************************
 * eUSCIB0 ISR. The repeated start and transmit/receive operations happen
 * within this ISR.
 ******************************************************************************/
void EUSCIB1_IRQHandler(void)
{
    uint_fast16_t status;
    status = MAP_I2C_getEnabledInterruptStatus(EUSCI_B1_BASE);
    /* Receives bytes into the receive buffer. If we have received all bytes,
     * send a STOP condition */
    if (status & EUSCI_B_I2C_RECEIVE_INTERRUPT0)
```

```
{
   if (read_idx == NUM_OF_REC_BYTES - 2)
   {
      MAP_I2C_disableInterrupt(EUSCI_B1_BASE,
            EUSCI_B_I2C_RECEIVE_INTERRUPT0);
      MAP_I2C_enableInterrupt(EUSCI_B1_BASE, EUSCI_B_I2C_STOP_INTERRUPT);//dont disable I2c
      /*
       * Switch order so that stop is being set during reception of last
       * byte read byte so that next byte can be read.
       */
      MAP_I2C_masterReceiveMultiByteStop(EUSCI_B1_BASE);
      RXData[read_idx++] = MAP_I2C_masterReceiveMultiByteNext(
            EUSCI_B1_BASE);
      if(read_idx == NUM_OF_REC_BYTES -1)//last byte received
         read_done = true;
   }
   else
   {
      RXData[read_idx++] = MAP_I2C_masterReceiveMultiByteNext(
      EUSCI_B1_BASE);
   }
}
else if (status & EUSCI_B_I2C_STOP_INTERRUPT)
{
   RXData[read_idx++] = MAP_I2C_masterReceiveMultiByteNext(
```

```
            EUSCI_B1_BASE);
        MAP_Interrupt_disableSleepOnIsrExit(); //dont sleep on exit
        MAP_I2C_disableInterrupt(EUSCI_B1_BASE,EUSCI_B_I2C_STOP_INTERRUPT);//dont disable I2C!
    }
}
```

9.3.2 buzzer.h

```
/**
 * @file    :   buzzer.h
 * @brief   :   An abstraction for buzzer functions
 *
 *              This header file provides an abstraction of buzzer functions
 *              which are used to initialize and play the buzzer
 *
 * @author  :   Vishal Raj
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   MSP432 Reference Manual
*/
#ifndef BUZZER_H_
#define BUZZER_H_
#include <stdbool.h>
/* Define MACROS for buzzer timings */
```

```
#define ON_TIME       (150000)
#define OFF_TIME      (100000)
#define BUZZ_CNT      (2)
/*******************************************************************************
 * @function:   play_buzzer
 *
 * @brief   :   Beeps the buzzer count times
 *
 * @param   :   buzz_count   - No of beeps of buzzer.
 *
 * @return  :   void
 *
********************************************************************************/
void play_buzzer(int buzz_count);
/*******************************************************************************
 * @function:   buzzer_state_change
 *
 * @brief   :   Changes the state of the buzzer
 *
 * @param   :   state   - true  - turn ON the buzzer
 *                      - flase - turn OFF the buzzer
 *
 * @return  :   void
 *
********************************************************************************/
```

```
void buzzer_state_change(bool state);

#endif /* BUZZER_H_ */
```

9.3.3 buzzer.c

```
/**

 * @file    :   buzzer.c

 * @brief   :   An abstraction for buzzer functions

 *

 *              This source file provides buzzer functions

 *              which are used to initialize and play the buzzer

 *

 * @author  :   Vishal Raj

 * @date    :   November 28, 2021

 * @version :   1.0

 *

 * @tools   :   Code Composer Studio

 *

 * @link    :   MSP432 Reference Manual

*/

/* DriverLib Includes */

#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */

#include <stdint.h>

#include <inc/buzzer.h>

bool enable_buzzer = false;

void play_buzzer(int buzz_count)
```

```
{
   if(enable_buzzer ==  true)
   {
      volatile int i,j;
      /* Configure P2.4 as output */
      MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN4);
      /* Turn on the buzzer */
      MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN4);
      for(i = 0; i < buzz_count ; i++)
      {
         /* ON Time */
         for(j = 0; j < ON_TIME; j++);
         /* Turn off the buzzer */
         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4);
         /* OFF Time */
         for(j = 0; j < OFF_TIME; j++);
         /* Turn on the buzzer */
         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN4);
      }
      /* Turn off the buzzer */
      MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4);
   }
}
void buzzer_state_change(bool state)
{
```

```
    enable_buzzer = state;
}
```

9.3.4 lcd.h

```
/**
 * @file    :   lcd.h
 * @brief   :   Header for LCD driver
 *
 *              This header file declares LCD functions needed
 *              to write data to the LCD.
 *
 * @author  :   Vishal Raj
 * @date    :   November 26, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   LCD command sequence,values referred from-
 *              https://github.com/johnrickman/LiquidCrystal_I2C/blob/master/LiquidCrystal_I2C.cpp.
*/

#ifndef LCD_H_
#define LCD_H_

#include <stdint.h>
```

```
/**********************************************************************************
 * @function:   i2c_init
 *
 * @brief   :   Initializes the I2C for the LCD
 *              using EUSCIB2   :   SDA - pin 3.6
 *                                  SCL - pin 3.7
 *
 * @param   :   none
 *
 * @return  :   void
 *
 **********************************************************************************/
void i2c_init(void);
/**********************************************************************************
 * @function:   lcd_i2c_init
 *
 * @brief   :   Initializes the I2C for LCD
 *
 * @param   :   none
 *
 * @return  :   void
 *
 **********************************************************************************/
void lcd_i2c_init(void);    //redundant function
```

```
/*******************************************************************************
 * @function:   delay_ms
 *
 * @brief   :   Creates a delay in milliseconds using a hard spin loop
 *
 * @param   :   delay   - delay in  milliseconds
 *
 * @return  :   void
 *
 *******************************************************************************/
void delay_ms(uint32_t delay);
/*******************************************************************************
 * @function:   lcd_init
 *
 * @brief   :   Initializes the LCD
 *
 * @param   :   none
 *
 * @return  :   void
 *
 *******************************************************************************/
void lcd_init(void);
/*******************************************************************************
 * @function:   i2c_write
 *
```

```
 * @brief   :   Writes data to the I2C of LCD
 *
 * @param   :   data    - byte of data to be written
 *
 * @return  :   void
 *
*******************************************************************************/
void i2c_write(uint8_t data);
/*******************************************************************************
 * @function:   lcd_cmd
 *
 * @brief   :   Send a command to the LCD
 *
 * @param   :   data    - command to be sent
 *
 * @return  :   void
 *
*******************************************************************************/
void lcd_cmd(uint8_t data);
/*******************************************************************************
 * @function:   lcd_4bit_write
 *
 * @brief   :   writes 4 bits to the LCD
 *
 * @param   :   data    - data to be written
```

```
 *
 * @return  :   void
 *
********************************************************************************/
void lcd_4bit_write(uint8_t data);
/********************************************************************************
 * @function:   lcd_clear_home
 *
 * @brief   :   clears the LCD and returns to the start
 *
 * @param   :   none
 *
 * @return  :   void
 *
********************************************************************************/
void lcd_clear_home(void);
/********************************************************************************
 * @function:   lcd_home
 *
 * @brief   :   returns to the start
 *
 * @param   :   none
 *
 * @return  :   void
 *
```

```
*********************************************************************************/
void lcd_home();
/*********************************************************************************
 * @function:   lcd_display
 *
 * @brief   :   turns on the LCD display
 *
 * @param   :   none
 *
 * @return  :   void
 *
*********************************************************************************/
void lcd_display();
/*********************************************************************************
 * @function:   lcd_clear
 *
 * @brief   :   clears the LCD screen
 *
 * @param   :   none
 *
 * @return  :   void
 *
*********************************************************************************/
void lcd_clear();
/*********************************************************************************
```

```
 * @function:   lcd_send
 *
 * @brief   :   clears the LCD screen
 *
 * @param   :   data - data to be sent
 *              mode - mode of the LCD
 *
 * @return  :   void
 *
*******************************************************************************/
void lcd_send(uint8_t data, uint8_t mode);
/******************************************************************************
 * @function:   pulse_enable
 *
 * @brief   :   enables the pulse (blinking), to enable the LCD
 *
 * @param   :   data - data to be sent
 *
 * @return  :   void
 *
*******************************************************************************/
void pulse_enable(uint8_t data);
/******************************************************************************
 * @function:   lcd_set_cursor
 *
```

```
 * @brief   :   sets the LCD cursor
 *
 * @param   :   col - column value
 *              row - row value
 *
 * @return  :   void
 *
**************************************************************************************/
void lcd_set_cursor(uint8_t col, uint8_t row);
/*************************************************************************************
 * @function:   lcd_print
 *
 * @brief   :   prints a string on to the LCD
 *
 * @param   :   data    - string to be printed
 *
 * @return  :   void
 *
**************************************************************************************/
void lcd_print(char data[]);
#endif /* LCD_H_ */
```

9.3.5 lcd.c

```
/**
 * @file    :   lcd.c
 * @brief   :   An abstraction for lcd functions
 *
 *              This source file provides the LCD functions and
 *              to write data at specific locations of the LCD.
 *
 * @author  :   Vishal Raj
 * @date    :   November 26, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   LCD command sequence,values referred from-
 *              https://github.com/johnrickman/LiquidCrystal_I2C/blob/master/LiquidCrystal_I2C.cpp.
*/
#include <stdbool.h>
#include <stdio.h>
#include  <math.h>
#include <stdint.h>
#include <stddef.h>
#include "lcd.h"
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#define I2C_WRITE_VAL           0x27
#define I2C_READ_VAL            0x41
```

```
#define DATA_LINES        (0x20 | 0x00)

#define ROW_DISP          (0x20 | 0x08)

#define DOTS              (0x20 | 0x00)

#define DISP_ON           (0x08 | 0x04)

#define CURSOR_OFF        (0x08 | 0x00)

#define BLINK_OFF         (0x08 | 0x00)

#define CLR_DISP          (0x01)

#define GO_HOME           0x02

#define CURSOR_INC        (0x04 | 0x02)

#define DISP_NO_SHIFT     (0x04 | 0x00)

#define DELAY             (1)


// commands

#define LCD_CLEARDISPLAY 0x01

#define LCD_RETURNHOME 0x02

#define LCD_ENTRYMODESET 0x04

#define LCD_DISPLAYCONTROL 0x08

#define LCD_CURSORSHIFT 0x10

#define LCD_FUNCTIONSET 0x20

#define LCD_SETCGRAMADDR 0x40

#define LCD_SETDDRAMADDR 0x80


// flags for display entry mode

#define LCD_ENTRYRIGHT 0x00

#define LCD_ENTRYLEFT 0x02
```

```
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00
// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

// flags for backlight control
```

```
#define LCD_BACKLIGHT 0x08

#define LCD_NOBACKLIGHT 0x00

#define En 0b00000100  // Enable bit

#define Rw 0b00000010  // Read/Write bit

#define Rs 0b00000001  // Register select bit


//lcd variables

uint8_t data_val;

uint8_t backlight_val;

uint8_t display_func,display_mode,display_control;


#define SLAVE_ADDRESS       0x27//0x48

#define NUM_OF_REC_BYTES    1


/* Variables */

const uint8_t TXData_lcd[] = {0x08,0x08};

/* I2C Master Configuration Parameter */

const eUSCI_I2C_MasterConfig i2cConfig_lcd =

{

        EUSCI_B_I2C_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source

        3000000,                                // SMCLK = 3MHz (default)

        EUSCI_B_I2C_SET_DATA_RATE_100KBPS,      // Desired I2C Clock of 100khz

        0,                                      // No byte counter threshold

        EUSCI_B_I2C_NO_AUTO_STOP                // No Autostop

};
```

```
void i2c_init(void)
{

    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3,
            GPIO_PIN6 + GPIO_PIN7, GPIO_PRIMARY_MODULE_FUNCTION);
    /* Initializing I2C Master to SMCLK at 100khz with no autostop */
    MAP_I2C_initMaster(EUSCI_B2_BASE, &i2cConfig_lcd);

    /* Specify slave address */
    MAP_I2C_setSlaveAddress(EUSCI_B2_BASE, SLAVE_ADDRESS);

    /* Enable I2C Module to start operations */
    MAP_I2C_enableModule(EUSCI_B2_BASE);
    MAP_Interrupt_enableInterrupt(INT_EUSCIB2);

    // enable RX interrupts
    MAP_I2C_enableInterrupt(EUSCI_B2_BASE, EUSCI_B_I2C_RECEIVE_INTERRUPT0);
}
void lcd_i2c_init(void)
{
    lcd_init();
    lcd_set_cursor(3,0);
    lcd_print("ESD Project");
    lcd_set_cursor(0,1);
    lcd_print("Vishal & Sanish");
```

```
}
//Used to position the LCD cursor
void lcd_set_cursor(uint8_t col, uint8_t row)
{
    int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
    lcd_cmd(LCD_SETDDRAMADDR | (col + row_offsets[row]));
}
void lcd_init(void)
{
    backlight_val = LCD_BACKLIGHT;
    display_func = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;
    display_func |= LCD_2LINE;
    display_func |= LCD_5x10DOTS;
    delay_ms(50);//Test delay
    i2c_write(backlight_val);
    delay_ms(1000);
    //write data in 8 bit mode and shift to 4 bit mode.
    lcd_4bit_write(0x03 << 4);
    delay_ms(5);
    lcd_4bit_write(0x03 << 4);
    delay_ms(5);
    lcd_4bit_write(0x03 << 4);
    delay_ms(5);
    //set to 4 bit.
    lcd_4bit_write(0x02 <<4);
```

```
    //set no of lines and font size, etc.
    lcd_cmd(LCD_FUNCTIONSET | display_func);


    //turn cursor on with blinking
    display_control = LCD_DISPLAYON | LCD_CURSORON | LCD_BLINKOFF;//changed
    lcd_display();
    lcd_clear();
    display_mode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
    //set entry mode
    lcd_cmd(LCD_ENTRYMODESET | display_mode);
    lcd_home();
}
void lcd_home()
{
    lcd_cmd(LCD_RETURNHOME);
    delay_ms(2);
}
void lcd_display()
{
    display_control |= LCD_DISPLAYON;
    lcd_cmd(LCD_DISPLAYCONTROL | display_control);
}
void lcd_clear()
{
```

```
    lcd_cmd(LCD_CLEARDISPLAY);
    delay_ms(2);
}
void i2c_write(uint8_t data)
{
    data = data | backlight_val;
    //Write one byte of data to I2C slave from master
    while (MAP_I2C_masterIsStopSent(EUSCI_B2_BASE));
    MAP_I2C_masterSendSingleByte(EUSCI_B2_BASE,data);
}
void delay_ms(uint32_t delay)
{
    uint32_t i;
    delay = delay*300;
    for(i = 0; i < delay ; i++)
        __asm("  nop");
}
void lcd_cmd(uint8_t data)
{
    lcd_send(data,0);//RS=0;
}
void lcd_print(char data[])
{
    char *str_ptr = data;
    while(*str_ptr)
```

```
        lcd_send(*str_ptr++,Rs);
}
void lcd_send(uint8_t data, uint8_t mode)
{
    uint8_t high_nibble = data & 0xF0;
    uint8_t low_nibble = (data << 4) & 0xF0;
    lcd_4bit_write(high_nibble | mode);
    lcd_4bit_write(low_nibble | mode);
}
void lcd_4bit_write(uint8_t data)
{
   i2c_write(data);
   pulse_enable(data);
}
void pulse_enable(uint8_t data)
{
    i2c_write(data | En);
    delay_ms(1);
    i2c_write(data & ~En);
    delay_ms(1);
}
void lcd_clear_home(void)
{
    lcd_cmd(CLR_DISP);
    lcd_cmd(GO_HOME);
```

```
}
```

9.3.6 mode.h

```
/**
 * @file    :   mode.h
 * @brief   :   An abstraction for mode functions
 *
 *              This header file provides abstraction of mode functions which are used to
 *              initialize the mode gpio and get the current mode of the system
 *
 * @author  :   Vishal Raj
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   MSP432 Reference Manual
*/
#ifndef INC_MODE_H_
#define INC_MODE_H_
//#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
/* Enum for state of the LCD display */
typedef enum
{
   ACCELEROMETER,
```

```
    GYROSCOPE
}mode_t;
/*******************************************************************************
 * @function:   mode_gpio_init
 *
 * @brief   :   Initializes the GPIO pin used to set the mode of LCD display
 *
 * @param   :   none
 *
 * @return  :   void
 *
 ********************************************************************************/
void mode_gpio_init(void);
/*******************************************************************************
 * @function:   get_mode
 *
 * @brief   :   Returns the current mode
 *
 * @param   :   mode_t - current mode
 *              ACCELEROMETER   - accelerometer values are displayed on the LCD
 *              GYROSCOPE       - gyroscope values are displayed on the LCD
 *
 * @return  :   void
 *
 ********************************************************************************/
```

```
mode_t get_mode(void);

#endif /* INC_MODE_H_ */
```

9.3.7 mode.c

```
**

 * @file    :   mode.c

 * @brief   :   An abstraction for mode functions

 *

 *              This source file provides mode functions which are used to

 *              initialize the mode gpio and get the current mode of the system

 *

 * @author  :   Vishal Raj

 * @date    :   November 28, 2021

 * @version :   1.0

 *

 * @tools   :   Code Composer Studio

 *

 * @link    :   MSP432 Reference Manual

*/

/* DriverLib Includes */

#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */

#include <stdbool.h>

#include <inc/buzzer.h>

#include <inc/mode.h>
```

```
/* Initialize mode to gyroscope */
mode_t mode = GYROSCOPE;
void mode_gpio_init(void)
{
   /* Configuring P1.4 as an input and enabling interrupts */
   MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN4);
   MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN4);
   MAP_GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN4);
   MAP_Interrupt_enableInterrupt(INT_PORT1);
   /* Enabling MASTER interrupts */
   MAP_Interrupt_enableMaster();
}
/******************************************************************************
 * @function:   PORT1_IRQHandler
 *
 * @brief   :   ISR for Port 1
 *
 * @param   :   none
 *
 * @return  :   void
 *
******************************************************************************/
void PORT1_IRQHandler(void)
{
   uint32_t status;
```

```
    static uint8_t counter = 0;
    status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P1);
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, status);
    /* Toggling the output on the LED */
    if(status & GPIO_PIN4)
    {
        counter++;
        /*Enable the buzzer after 2 changes of state so that accelerometer can be
         * tested properly without the buzzer beeping continuously
         * */
        if(counter == 2)
            buzzer_state_change(true);
        if(mode == ACCELEROMETER)
            mode = GYROSCOPE;
        else if(mode == GYROSCOPE)
            mode = ACCELEROMETER;
    }
}
mode_t get_mode(void)
{
    return mode;
}
```

9.3.8 mpu.h

```
/**
 * @file    :   mpu.h
```

```
 * @brief   :   Hearder for MPU6050 driver
 *
 *           This header file declares MPU6050 functions needed
 *           to read and write data from the sensor.
 *
 * @author  :   Vishal Raj
 * @date    :   November 24, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   Some implementation referred from http://www.brokking.net/imu.html.
*/
#ifndef MPU_H_
#define MPU_H_
#include <stdint.h>
#define NUM_OF_REC_BYTES    6
struct mpu_values{
   float gGyroVal[3];//order x y z axis
   int16_t gAcclVal[3];//order x y z axis
};
//externs
extern volatile bool read_done;
extern uint8_t RXData[NUM_OF_REC_BYTES];
extern volatile uint8_t read_idx;
```

```
typedef struct mpu_values gAcc_gyro;

/*******************************************************************************

 * @function:   MPU6050_Reset

 *

 * @brief   :   Initializes the MPU6050 sensor

 *

 * @param   :   none

 *

 * @return  :   void

 *

*******************************************************************************/

void MPU6050_Reset(void);//add void

/*******************************************************************************

 * @function:   mpu6050

 *

 * @brief   :   A test function for MPU6050 to test the module independently.

 *

 * @param   :   none

 *

 * @return  :   void

 *

*******************************************************************************/

void mpu6050(void);

/*******************************************************************************

 * @function:   get_mpu_values
```

```
 *
 * @brief   :   Reads values from MPU6050 and returns them in a structure
 *
 * @param   :   None
 *
 * @return  :    gAcc_gyro - structure containing 2 arrays of 3 elements each
 *                        for accelerometer and gyroscope readings
 *
*******************************************************************************/
gAcc_gyro get_mpu_values(void);
#endif /* MPU_H_ */
```

9.1.9 mpu.c

```
/**
 * @file    :   mpu.c
 * @brief   :   An I2C driver file for MPU6050
 *
 *            This source codes provides the data from MPU-6050 using I2C to
 *            get three axis gyroscope and calibration values each.
 *
 * @author  :   Vishal Raj
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
```

```
 * @link    :   Some implementation referred from http://www.brokking.net/imu.html.
*/
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <stdbool.h>
#include <stdio.h>
#include  <math.h>
#include <stdint.h>
#include <stddef.h>
#include "lcd.h"
#include "mpu.h"
#define MPU6050_ADDRESS 0x68
void MPU6050_Reset();
void MPU6050_ReadData(int16_t accelerometer[3], int16_t gyro[3], int16_t *temp);
void process_raw_values(int16_t accelerometer[3], int16_t gyro[3]);
//SCL - P6.5
//SDA - P6.4
//***Value processing code***
int gyro_x, gyro_y, gyro_z;
long acc_x, acc_y, acc_z, acc_total_vector;
int temperature;
long gyro_x_cal, gyro_y_cal, gyro_z_cal;
long loop_timer;
int lcd_loop_counter;
float angle_pitch, angle_roll,angle_yaw;
int angle_pitch_buffer, angle_roll_buffer;
```

```
bool set_gyro_angles;

float angle_roll_acc, angle_pitch_acc, angle_yaw_acc;

float angle_pitch_output, angle_roll_output;

uint8_t angle_pitch_int;

int16_t accelerometer[3], gyro[3], temp;

volatile uint8_t read_idx;

static gAcc_gyro gAccelero_t;

//***Value processing code***

gAcc_gyro get_mpu_values(void)//getter for MPU values

{

   MPU6050_ReadData(accelerometer, gyro, &temp);

   process_raw_values(accelerometer,gyro);

   return gAccelero_t;

}

void mpu6050(void)

{

   MPU6050_Reset();

   int i = 0;

   char s[10];

   lcd_home();

   while(1)

   {

      MPU6050_ReadData(accelerometer, gyro, &temp);

      process_raw_values(accelerometer,gyro);
```

```
printf("Pitch : %f^ , Roll : %f^ Yaw: %d\r\n",gAccelero_t.gGyroVal[0],gAccelero_t.gGyroVal[1],gAccelero_t.gGyroVal[2]);

printf("Pitch : %dg , Roll : %dg Yaw: %dg ",gAccelero_t.gAcclVal[0],gAccelero_t.gAcclVal[1],gAccelero_t.gAcclVal[2]);

    sprintf(s, "%d", i++);

    delay_ms(100);

    char buff[20];

    angle_pitch_int = (int)angle_pitch_output;

    sprintf(buff, "%d", angle_pitch_int);

  }

}

void MPU6050_Reset()//Init function

{

  int16_t accelerometer[3], gyro[3], temp;

  int i,j=0;

  //Use P2.0 for calibration indication

  P2->DIR |= BIT0;

  //Internal 8Mhz clock

  uint8_t writeData[] = {0x6B, 0x00};

  //Check for last transaction to complete

  while (MAP_I2C_masterIsStopSent(EUSCI_B1_BASE));

  //send bytes from master to slave

  MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,writeData[0]);

  MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,writeData[1]);

  //Configure accelerometer for +-8g

  writeData[0] = 0x1C;
```

```
    writeData[1] = 0x10;
    while (MAP_I2C_masterIsStopSent(EUSCI_B1_BASE));
    //send bytes from master to slave
    MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,writeData[0]);
    MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,writeData[1]);
    //Selecting full scale range as 500 degree per second
    writeData[0] = 0x1B;
    writeData[1] = 0x08;
    while (MAP_I2C_masterIsStopSent(EUSCI_B1_BASE));
    //send bytes from master to slave
    MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,writeData[0]);
    MAP_I2C_masterSendMultiByteFinish(EUSCI_B1_BASE,writeData[1]);
    P2->OUT |= BIT0;//to indicate start of calibration
    for (i = 0; i < 2000 ; i++ ){
        MPU6050_ReadData(accelerometer, gyro, &temp);
        gyro_x_cal += gyro[0];
        gyro_y_cal += gyro[1];
        gyro_z_cal += gyro[2];
    }
    gyro_x_cal /= 2000;
    gyro_y_cal /= 2000;
    gyro_z_cal /= 2000;
    P2->OUT ^= BIT0;//turn the led off, to indicate end of calibration
}
void MPU6050_ReadData(int16_t accelerometer[3], int16_t gyro[3], int16_t *temp)
```

```
{
    uint8_t writeData;//to write register address to read from
    // reading the accelerometer data
    writeData = 0x3B;
    //**send read address
    while (MAP_I2C_masterIsStopSent(EUSCI_B1_BASE));
    MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,writeData);
    MAP_I2C_masterSendMultiByteStop(EUSCI_B1_BASE);
    //**start reading
    read_idx = 0;
    MAP_I2C_masterReceiveStart(EUSCI_B1_BASE);
    MAP_I2C_enableInterrupt(EUSCI_B1_BASE, EUSCI_B_I2C_RECEIVE_INTERRUPT0);
    while(!read_done);//dont process until readind done
    if(read_done){
        read_done = false;
        accelerometer[0] = ((RXData[0] << 8) | RXData[1]);
        accelerometer[1] = ((RXData[2] << 8) | RXData[3]);
        accelerometer[2] = ((RXData[4] << 8) | RXData[5]);
    }
    //Gyro data
    writeData = 0x43;
    //**send read address
    while (MAP_I2C_masterIsStopSent(EUSCI_B1_BASE));
    MAP_I2C_masterSendMultiByteStart(EUSCI_B1_BASE,writeData);
    MAP_I2C_masterSendMultiByteStop(EUSCI_B1_BASE);
```

```
    //**start reading
    read_idx = 0;
    MAP_I2C_masterReceiveStart(EUSCI_B1_BASE);
    MAP_I2C_enableInterrupt(EUSCI_B1_BASE, EUSCI_B_I2C_RECEIVE_INTERRUPT0);
    while(!read_done);//dont process until reading done
    if(read_done){
        read_done = false;
        gyro[0] = ((RXData[0] << 8) | RXData[1]);
        gyro[1] = ((RXData[2] << 8) | RXData[3]);
        gyro[2] = ((RXData[4] << 8) | RXData[5]);
    }
}
void process_raw_values(int16_t accelerometer[3], int16_t gyro[3])
{
    int16_t accl_x,accl_y,accl_z;
    /*Output of gyro is in degree/sec, hence for FS=1, Gyro sensitivity scale factor = 65.5
     * therefore 1 degree/sec = 65.5 or rawGyroVal/65.5 = degree/sec*/
    //reading gyro raw values
    gyro_x = gyro[0];
    gyro_y = gyro[1];
    gyro_z = gyro[2];
            /*Output of accelerometer for AFS_SEL=2 is 4096/g, i.e 1g=4096, therefore
rawAcclVal/4096=9.8m/s^2(1g)*/
    //reading acceleromter values
    acc_x = accelerometer[0];
    acc_y = accelerometer[1];
```

```
    acc_z = accelerometer[2];
    //Compensating current values with the calibration values
    gyro_x -= gyro_x_cal;
    gyro_y -= gyro_y_cal;
    gyro_z -= gyro_z_cal;
   //Gyro angle calculations
   angle_pitch  = gyro_x * 0.0152671;
   angle_roll   = gyro_y * 0.0152671;
   angle_yaw = gyro_z * 0.0152671;
   //Calculating acceleration values
   accl_x = (int16_t) (acc_x/4096) * 10;//in m/s^2
   accl_y = (int16_t) (acc_y/4096) * 10;
   accl_z = (int16_t) ((acc_z/4096) + 1) * 10;
   //storing the values
   gAccelero_t.gGyroVal[0] = angle_roll;//x - roll
   gAccelero_t.gGyroVal[1] = angle_pitch;//y - pitch
   gAccelero_t.gGyroVal[2] = angle_yaw;//z - yaw.
   gAccelero_t.gAcclVal[0] = accl_x;//x - roll
   gAccelero_t.gAcclVal[1] = accl_y;//y - pitch
   gAccelero_t.gAcclVal[2] = accl_z;//z - yaw
}
```

9.3.10 my_file_func.h

```
/**
 * @file    :   my_file_func.h
 * @brief   :   An abstraction for my file functions
```

```
 *
 *          This header file provides abstraction of my file functions which are wrapper functions
 *          around the FAT file functions. They help in reducing the code duplication in the main logic
 *
 * @author  :   Sanish Kharade
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   http://elm-chan.org/fsw/ff/00index_e.html
*/
#ifndef INC_MY_FILE_FUNC_H_
#define INC_MY_FILE_FUNC_H_
#include <fatfs/ff.h>
/***************************************************************************************
 * @function:   file_write
 *
 * @brief   :   wrapper function for f_write
 *          writes a string to the file mentioned
 *
 * @param   :   fp          - file object
 *          filename    - name of the file
 *          buff        - string to be written to the file
 *          btw         - no of bytes to be written
```

```
 *
 * @return  :   void
 *
*********************************************************************************/
void my_file_write(FIL fp, char* filename, const void* buff, UINT btw);
/********************************************************************************
 * @function:   file_append
 *
 * @brief   :   appends a string to the file mentioned
 *
 * @param   :   fp          - file object
 *              filename    - name of the file
 *              buff        - string to be appended to the file
 *              btw         - no of bytes to be appended
 *
 * @return  :   void
 *
*********************************************************************************/
void my_file_append(FIL fp, char* filename, const void* buff, UINT btw);
/********************************************************************************
 * @function:   file_read
 *
 * @brief   :   wrapper function for f_read
 *              reads full contents of a file line by line
 *
```

```
 * @param   :   fp          - file object
 *             filename    - name of the file
 *
 * @return  :   void
 *
**********************************************************************************************/
void my_file_read(FIL fp, char* filename);
/*********************************************************************************************
 * @function:   file_stat
 *
 * @brief   :   wrapper function for f_stat
 *             gets the info of the file into the fileinfo parameter
 *
 * @param   :   fp          - file object
 *             filename    - name of the file
 *
 * @return  :   void
 *
**********************************************************************************************/
void my_file_stat(char* filename, FILINFO *fileinfo);
/*********************************************************************************************
 * @function:   print_file_attributes
 *
 * @brief   :   prints the attributes of the file
 *
```

```
 * @param   :   fileinfo - FILINFO parameter
 *
 * @return  :   void
 *
********************************************************************************/
void my_file_print_attributes(FILINFO fileinfo);
#endif /* INC_MY_FILE_FUNC_H_ */
```

9.3.11 my_file_func.c

```
/**
 * @file    :   my_file_func.c
 * @brief   :   An abstraction for my file functions
 *
 *              This source file provides abstraction of my file functions which are wrapper functions
 *              around the FAT file functions. They help in reducing the code duplication in the main logic
 *
 * @author  :   Sanish Kharade
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   http://elm-chan.org/fsw/ff/00index_e.html
*/
#include <string.h>
#include <fatfs/ff.h>
```

```
#include <fatfs/diskio.h>

#include <Devices/MSPIO.h>

#include "my_file_func.h"

void my_file_write(FIL fp, char* filename, const void* buff, UINT btw)

{

   FRESULT fr;

   UINT bw;

   fr = f_open(&fp, filename, FA_WRITE);

   if(fr != FR_OK)

   {

      MSPrintf(EUSCI_A0_BASE, "Error opening file/directory\r\n");

      while(1);

   }

   fr = f_write(&fp, buff, btw, &bw);

   if(fr != FR_OK)

   {

      MSPrintf(EUSCI_A0_BASE, "Error writing to  file/directory\r\n");

      while(1);

   }

   f_close(&fp);

}

void my_file_append(FIL fp, char* filename, const void* buff, UINT btw)

{

   FRESULT fr;

   UINT bw;
```

```
    fr = f_open(&fp, filename, (FA_OPEN_APPEND | FA_WRITE) );
    if(fr != FR_OK)
    {
        MSPrintf(EUSCI_A0_BASE, "Error opening %s file/directory\r\n", filename);
        /* Uncomment below line if you want to halt the code at the first error in opening file */
        //while(1);
    }
    fr = f_write(&fp, buff, btw, &bw);
    if(fr != FR_OK)
    {
        MSPrintf(EUSCI_A0_BASE, "Error writing to  file/directory\r\n");
        /* Uncomment below line if you want to halt the code at the first error in opening file */
        //while(1);
    }
    f_close(&fp);
}
void my_file_read(FIL fp, char* filename)
{
    FRESULT fr;
    //UINT bw;
    char line[50] = {0};
    fr = f_open(&fp, filename, FA_READ );
    if(fr != FR_OK)
    {
        MSPrintf(EUSCI_A0_BASE, "Error opening file/directory\r\n");
```

```
        while(1);
    }
    MSPrintf(EUSCI_A0_BASE, "\n\rReading output4.txt :\n\r");
    while (f_gets(line, sizeof line, &fp))
    {
        /* Data is printed out on the UART */
        MSPrintf(EUSCI_A0_BASE, line);
        memset(line, 0, sizeof(line));
    }
    f_close(&fp);
}
void my_file_stat(char* filename, FILINFO *fileinfo)
{
    char buf[70]={0};
    FRESULT fr;
    fr = f_stat(filename, fileinfo);
    if(fr == FR_NO_FILE)
    {
        MSPrintf(EUSCI_A0_BASE, "%s - File not found\r\n", filename);
    }
    else if(fr == FR_OK)
    {
        /* Data is printed out on the UART */
        sprintf(buf, "size = %d, date = %d, time = %d, attribute = %c, name= %s \n\r"
                ,(*fileinfo).fsize, (*fileinfo).fdate,(*fileinfo).ftime, (*fileinfo).fattrib, (*fileinfo).fname);
```

```
        MSPrintf(EUSCI_A0_BASE, buf);
    }
}
void my_file_print_attributes(FILINFO fileinfo)
{
    char buf[70]={0};
    /* Data is printed out on the UART */
    sprintf(buf, "size = %d, date = %d, time = %d, attribute = %c, name= %s \n\r"
            ,fileinfo.fsize, fileinfo.fdate,fileinfo.ftime, fileinfo.fattrib, fileinfo.fname);
    MSPrintf(EUSCI_A0_BASE, buf);
}
```

9.3.12 rtc.h

```
/**
 * @file    :   rtc.h
 * @brief   :   An abstraction for RTC functions
 *
 *              This header file provides abstraction of RTC functions which are
 *              used to initialize and keep track of time
 *
 * @author  :   Sanish Kharade
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
```

```
 * @link    :   MSP432 Reference Manual

*/

#ifndef INC_RTC_H_

#define INC_RTC_H_

#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#include <stdio.h>

/********************************************************************************

 * @function:   RTC_init

 *

 * @brief   :   Initializes the RTC

 *

 * @param   :   start_time - start time for the RTC

 *

 * @return  :   void

 *

********************************************************************************/

void RTC_init(RTC_C_Calendar start_time);

/********************************************************************************

 * @function:   RTC_get_current_time

 *

 * @brief   :   Returns the current time as managed by the RTC

 *

 * @param   :   void

 *

 * @return  :   RTC_C_Calendar - current RTC calendar time
```

```
 *
*********************************************************************************/
RTC_C_Calendar RTC_get_current_time(void);
/*********************************************************************************
 * @function:   one_second_elapsed
 *
 * @brief   :   Checks if one second has elapsed
 *
 * @param   :   void
 *
 * @return  :   true    - if one second has elapsed
 *              false   - if one second has not elapsed
 *
*********************************************************************************/
bool one_second_elapsed(void);
/*********************************************************************************
 * @function:   clear_one_second_elapsed_flag
 *
 * @brief   :   Clears the one second elapsed flag
 *
 * @param   :   void
 *
 * @return  :   void
 *
*********************************************************************************/
```

```
void clear_one_second_elapsed_flag(void);

#endif /* INC_RTC_H_ */
```

9.3.13 rtc.c

```
/**
 * @file    :   rtc.c
 * @brief   :   An abstraction for RTC functions
 *
 *              This source file provides abstraction of RTC functions which are
 *              used to initialize and keep track of time
 *
 * @author  :   Sanish Kharade
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   MSP432 Reference Manual
*/

#include <stdbool.h>
#include <inc/rtc.h>

static volatile RTC_C_Calendar current_time;
static volatile bool one_second_tracker;

RTC_C_Calendar RTC_get_current_time(void)
{
    return current_time;
```

```
}
//check logic
bool one_second_elapsed(void)
{
    return one_second_tracker;
}
void clear_one_second_elapsed_flag(void)
{
    one_second_tracker = false;
}
void RTC_init(RTC_C_Calendar start_time)
{
    /* Initialize the RTC, some functions are done for debugging purposes */
    /* Initializing RTC with current time */
    MAP_RTC_C_initCalendar(&start_time, RTC_C_FORMAT_BINARY);

    /* Setup Calendar Alarm for 10:04pm (for the flux capacitor) */
    MAP_RTC_C_setCalendarAlarm(0x04, 0x22, RTC_C_ALARMCONDITION_OFF,
            RTC_C_ALARMCONDITION_OFF);
    /* Specify an interrupt to assert every minute */
    MAP_RTC_C_setCalendarEvent(RTC_C_CALENDAREVENT_MINUTECHANGE);
    /* Enable interrupt for RTC Ready Status, which asserts when the RTC
     * Calendar registers are ready to read.
     * Also, enable interrupts for the Calendar alarm and Calendar event. */
    MAP_RTC_C_clearInterruptFlag(
```

```
        RTC_C_CLOCK_READ_READY_INTERRUPT | RTC_C_TIME_EVENT_INTERRUPT
                | RTC_C_CLOCK_ALARM_INTERRUPT);
    MAP_RTC_C_enableInterrupt(
        RTC_C_CLOCK_READ_READY_INTERRUPT | RTC_C_TIME_EVENT_INTERRUPT
                | RTC_C_CLOCK_ALARM_INTERRUPT);
    /* Start RTC Clock */
    MAP_RTC_C_startClock();
    /* Enable interrupts and go to sleep. */
    MAP_Interrupt_enableInterrupt(INT_RTC_C);
    MAP_Interrupt_enableMaster();
}
/*******************************************************************************
 * @function:   RTC_C_IRQHandler
 *
 * @brief   :   ISR for RTC
 *
 * @param   :   none
 *
 * @return  :   void
 *
*******************************************************************************/
void RTC_C_IRQHandler(void)
{
    uint32_t status;
    one_second_tracker = true;
```

```
    /*Update current time */
    current_time = MAP_RTC_C_getCalendarTime();
    status = MAP_RTC_C_getEnabledInterruptStatus();
    MAP_RTC_C_clearInterruptFlag(status);
    if (status & RTC_C_CLOCK_READ_READY_INTERRUPT)
    {
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
    }
    if (status & RTC_C_TIME_EVENT_INTERRUPT)
    {
        /* Interrupts every minute - Set breakpoint here */
        __no_operation();
        current_time = MAP_RTC_C_getCalendarTime();
    }
    if (status & RTC_C_CLOCK_ALARM_INTERRUPT)
    {
        /* Interrupts at 10:04pm */
        __no_operation();
    }
}
```

9.3.14 sd_card.h

```
/**
 * @file    :   sd_card.h
 * @brief   :   An abstraction for RTC functions
 *
```

```
*       This header file provides abstraction of SD Card function
*       which is used to update the SD Card
*
* @author  :   Sanish Kharade
* @date    :   November 28, 2021
* @version :   1.0
*
* @tools   :   Code Composer Studio
*
* @link    :   http://elm-chan.org/fsw/ff/00index_e.html
*/
#ifndef INC_SD_CARD_H_
#define INC_SD_CARD_H_
#include <inc/mpu.h>
#include <inc/rtc.h>
/*****************************************************************************
* @function:   update_SD_card
*
* @brief   :   updates the SD card with the timestamp and sensor reading
*
* @param   :   none
*
* @return  :   void
*
*****************************************************************************/
```

```
void update_SD_card(RTC_C_Calendar ntime, gAcc_gyro sensor);

#endif /* INC_SD_CARD_H_ */
```

9.3.15 sd_card.c

```
/**

 * @file    :   sd_card.c

 * @brief   :   An abstraction for RTC functions

 *

 *              This source file provides abstraction of SD Card function

 *              which is used to update the SD Card

 *

 * @author  :   Sanish Kharade

 * @date    :   November 28, 2021

 * @version :   1.0

 *

 * @tools   :   Code Composer Studio

 *

 * @link    :   -

*/

#include <string.h>

#include <stdbool.h>

#include <stdint.h>

#include <fatfs/ff.h>

#include <fatfs/diskio.h>

#include <Devices/MSPIO.h>

#include <inc/sd_card.h>
```

```
#include <inc/my_file_func.h>
#include <inc/sw420.h>
#include <inc/lcd.h>
#include <inc/buzzer.h>
#include <inc/mode.h>
/* Enum for time */
enum time_e
{
    YEAR,
    MONTH,
    DATE,
    HOUR,
    MINUTE,
    SECOND
};
void update_SD_card(RTC_C_Calendar ntime, gAcc_gyro sensor)
{
    static RTC_C_Calendar old_time = {0,0,0,0,0,0,0};
    FIL fp;
    //,fp1;
    /* Create buffers */
    char final_path[100] = {0};
    char write_buffer[200] = {0};
    char path[10][5] = {0};
    char read_buffer[20];
```

```
/* Set values of RTC for the path */
sprintf(path[YEAR],"%04d", ntime.year);
sprintf(path[MONTH],"%02d", ntime.month);
sprintf(path[DATE],"%02d", ntime.dayOfmonth);
sprintf(path[HOUR],"%02d", ntime.hours);
sprintf(path[MINUTE],"%02d", ntime.minutes);
sprintf(path[SECOND],"%02d", ntime.seconds);
/* If year changed, create a new directory in the correct path */
if(ntime.year != old_time.year)
{
   sprintf(final_path, "/");
   f_chdir(final_path);
   f_mkdir(path[YEAR]);
   f_chdir(path[YEAR]);
   f_getcwd(read_buffer, sizeof(read_buffer));
   MSPrintf(EUSCI_A0_BASE, read_buffer);
   memset(final_path, 0, sizeof(final_path));
}
/* If month changed, create a new directory in the correct path */
if(ntime.month != old_time.month)
{
   sprintf(final_path, "/%s/",path[YEAR]);

   f_chdir(final_path);
   f_mkdir(path[MONTH]);
```

```
    f_chdir(path[MONTH]);
    f_getcwd(read_buffer, sizeof(read_buffer));
    MSPrintf(EUSCI_A0_BASE, read_buffer);
    memset(final_path, 0, sizeof(final_path));
}
/* If date changed, create a new directory in the correct path */
if(ntime.dayOfmonth != old_time.dayOfmonth)
{
    sprintf(final_path, "/%s/%s/",path[YEAR], path[MONTH]);
    f_chdir(final_path);
    f_mkdir(path[DATE]);
    f_chdir(path[DATE]);
    f_getcwd(read_buffer, sizeof(read_buffer));
    MSPrintf(EUSCI_A0_BASE, read_buffer);
    memset(final_path, 0, sizeof(final_path));
}
/* If hour changed, create a new directory in the correct path */
if(ntime.hours != old_time.hours)
{
    sprintf(final_path, "/%s/%s/%s",path[YEAR], path[MONTH],path[DATE]);
    f_chdir(final_path);
    f_mkdir(path[HOUR]);
    f_chdir(path[HOUR]);
    f_getcwd(read_buffer, sizeof(read_buffer));
    MSPrintf(EUSCI_A0_BASE, read_buffer);
```

```
        memset(final_path, 0, sizeof(final_path));
    }
    char filename[10]={0};
    char date[20]={0};
    char timestamp[20]={0};
    char sensor_data[200] = {0};
    char sensor_data_disp[20] = {0};
    memset(read_buffer, 0, sizeof(read_buffer));
    sprintf(date, "Date-> %s-%s-%s", path[MONTH], path[DATE], path[YEAR]);
    sprintf(timestamp, "Time->%s:%s:%s ", path[HOUR], path[MINUTE], path[SECOND]);
    /* Print the time on the LCD */
    lcd_home();
    lcd_print(timestamp);
    /* Check the mode and print correct sensor's data */
    if(get_mode() == ACCELEROMETER)
    {
        lcd_print("A");
        lcd_set_cursor(0,1);
              sprintf(sensor_data_disp,"X%02dg  Y%02dg  Z%02dg    ",sensor.gAcclVal[0],  sensor.gAcclVal[1],
sensor.gAcclVal[2]);
        lcd_print(sensor_data_disp);
    }
    else
    {
        lcd_print("G");
        lcd_set_cursor(0,1);
```

```
        sprintf(sensor_data_disp,"X%02d   Y%02d   Z%02d            ",(int16_t)sensor.gGyroVal[0],(int16_t) sensor.gGyroVal[1],(int16_t) sensor.gGyroVal[2]);

      lcd_print(sensor_data_disp);

    }

    /* If shock was detectedpplay the buzzer and add a note along with sensor data */

    if(shock_detected() == true)

    {

      play_buzzer(BUZZ_CNT);

      sprintf(sensor_data,  "Acceleration(m/s^2)      : X = %d\t  , Y = %d\t , Z = %d\n"

                    "Gyroscope(degrees/sec)   : X = %f\t  , Y = %f, Z = %f\n"

                    "Shock Detected\n\r",

                    sensor.gAcclVal[0], sensor.gAcclVal[1], sensor.gAcclVal[2],

                    sensor.gGyroVal[0], sensor.gGyroVal[1], sensor.gGyroVal[2]);

    }

    else

    {

      sprintf(sensor_data,  "Acceleration(m/s^2)      : X = %d\t  , Y = %d\t , Z = %d\n"

                    "Gyroscope(degrees/sec)   : X = %f\t  , Y = %f, Z = %f\n\r",

                    sensor.gAcclVal[0], sensor.gAcclVal[1], sensor.gAcclVal[2],

                    sensor.gGyroVal[0], sensor.gGyroVal[1], sensor.gGyroVal[2]);

    }

    sprintf(write_buffer, "%s, %s \n%s", date, timestamp, sensor_data);

    sprintf(filename, "%s.txt", path[MINUTE]);

    MSPrintf(EUSCI_A0_BASE, write_buffer);

    my_file_append(fp, filename, write_buffer, strlen(write_buffer));

    /* Update old time */
```

```
    old_time = ntime;

    memset(final_path, 0, sizeof(final_path));

}
```

9.3.16 sw420.h

```
/**

 * @file    :   sw420.h

 * @brief   :   An header for abstraction for vibration functions

 *

 *              This header file provides abstraction of vibration sensor functions

 *              which are used to initialize it and track shocks detected

 *

 * @author  :   Sanish Kharade

 * @date    :   November 28, 2021

 * @version :   1.0

 *

 * @tools   :   Code Composer Studio

 *

 * @link    :   MSP432 Reference Manual

*/

#ifndef INC_SW420_H_

#define INC_SW420_H_

#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/*******************************************************************************

 * @function:   SW420_gpio_init

 *
```

```
 * @brief   :   Initializes the SW420 sensor's GPIO pin
 *
 * @param   :   none
 *
 * @return  :   void
 *
********************************************************************************/
void SW420_gpio_init(void);
/*******************************************************************************
 * @function:   shock_detected
 *
 * @brief   :   Checks if shock was detected by the SW420 sensor
 *
 * @param   :   none
 *
 * @return  :   void
 *
********************************************************************************/
bool shock_detected(void);
/*******************************************************************************
 * @function:   shock_detected
 *
 * @brief   :   Clears the shock detected flag
 *
 * @param   :   none
```

```
 *
 * @return  :   void
 *
*******************************************************************************/
void clear_shock_detected_flag(void);
#endif /* INC_SW420_H_ */
```

9.3.17 sw420.c

```
/**
 * @file    :   sw420.c
 * @brief   :   An abstraction for RTC functions
 *
 *              This header file provides abstraction of vibration sensor functions
 *              which are used to initialize it and track shocks detected
 *
 * @author  :   Sanish Kharade
 * @date    :   November 28, 2021
 * @version :   1.0
 *
 * @tools   :   Code Composer Studio
 *
 * @link    :   MSP432 Reference Manual
*/
#include <stdbool.h>
#include <inc/sw420.h>
bool shock_tracker = false;
```

```
void SW420_gpio_init(void)
{
    /* Configuring P2.5 as an input and enabling interrupts */
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P2, GPIO_PIN5);
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P2, GPIO_PIN5);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P2, GPIO_PIN5);
    MAP_Interrupt_enableInterrupt(INT_PORT2);
    /* Enabling MASTER interrupts */
    MAP_Interrupt_enableMaster();
}
/******************************************************************************
 * @function:   PORT2_IRQHandler
 *
 * @brief   :   ISR for Port 2
 *
 * @param   :   none
 *
 * @return  :   void
 *
 ******************************************************************************/
void PORT2_IRQHandler(void)
{
    uint32_t status;
    status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P2);
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P2, status);
```

```
    /* Toggling the output on the LED */
    if(status & GPIO_PIN5)
    {
        shock_tracker = true;
    }
}
bool shock_detected(void)
{
    return shock_tracker;
}
void clear_shock_detected_flag(void)
{
    shock_tracker = false;
}
```

Note -

Leveraged code has not been added to the report. Please find the complete source code in the github URL mentioned above