Embedded Challenge "The Need for Speed"

RTES Project Report

New York University - Tandon School of Engineering

Author: Cavin Roger Nunes crn2400

Fall 2021

Table of Contents

Introduction

Motion capture can be best described as being able to analyze and use the measured movement experienced by or forced upon objects, as well as the orientation and position of those objects. Many devices and technologies have been using motion analysis for quite some time, however as technology continues to advance faster we are seeing motion capture being featured in common consumer devices as well. The most common example of a well-known device which utilizes accelerometer and gyroscope sensors to capture motion is the smartphone. There are plenty of applications used on smartphones which utilize motion capture, such as changing the screen orientation for the user, using the device's motion to play interactive games, and being able to utilize the accelerometer and gyroscope together as a compass. Other common applications for accelerometers and gyroscopes are determining aircraft orientation for pilot controls, interpreting human gestures in software to act similarly to how we tend to use a computer mouse (such as the Nintendo Wii using a motion controller), and camera stabilization to optimize image capturing by offsetting the camera's detected movement. Overall, there are many modern day applications that use motion capture already, and as technological innovation continues it is likely that we will continue to see motion capture being utilized.

Problem Statement

Embedded system design focuses on gathering useful data, processing that data, and providing a useful representation of information. In the past decade we have seen an explosion of wearable health devices ranging from heart rate sensors to step counters to distance trackers. These devices are designed to help us meet our fitness goals and help us keep in good physical shape. The objective of this semester's embedded challenge is to build a wearable speedometer which can calculate velocity by measuring angular velocities available from our built-in gyroscope (L3GD20) - without a GPS. Our gyroscope is able to measure 3-axis angular velocity. Strategically placing the sensor on the legs or feet can capture the angular velocities and with a bit of processing can convert those angular velocitiesto linear forward velocity and calculate distance traveled (using only a gyro!)

Theory behind the Gyroscope

Gyroscope

Gyroscopes are used to measure the rotational velocity of an object, and the ones used for most embedded system sensors utilize a vibrating structure gyroscope rather than the traditional rotary gyroscope that most people are familiar with. Vibrating structure gyroscopes are MEMS devices that utilize the Coriolis force. The Coriolis force represents the force of an object whose relative angular position is maintained while the object experiences a lateral speed causing angular velocity. Vibrating structure gyroscopes contain small masses which are fixed to set array of springs that can then resonate in a capacitive outer shell structure as an object rotates, and the Coriolis force is then detected by the outer shell [5]. This implementation of a gyroscope allows for creating miniaturized sensors that are still able to experience and record the rotational motion experienced by devices.

Below, Figure below depicts the MEMS structure of a gyroscope with the spring array structure described. The encapsulating structure contains a block that can make rotational movement in the open space permitted. This block is composed of an array of springs that allow the centerpiece mass to move in various directions. The movement of the mass is sensed by the capacitive sensors surrounding the spring array in order to detect the relative rotational movement as felt due to the Coriolis force previously discussed. The Coriolis force is measured from the differences in capacitance measured by the capacitive plates; as the mass is pushed by the Coriolis force in different directions while the structure rotates, the capacitance measured by the sensors changes and reads the angular velocity as a result.

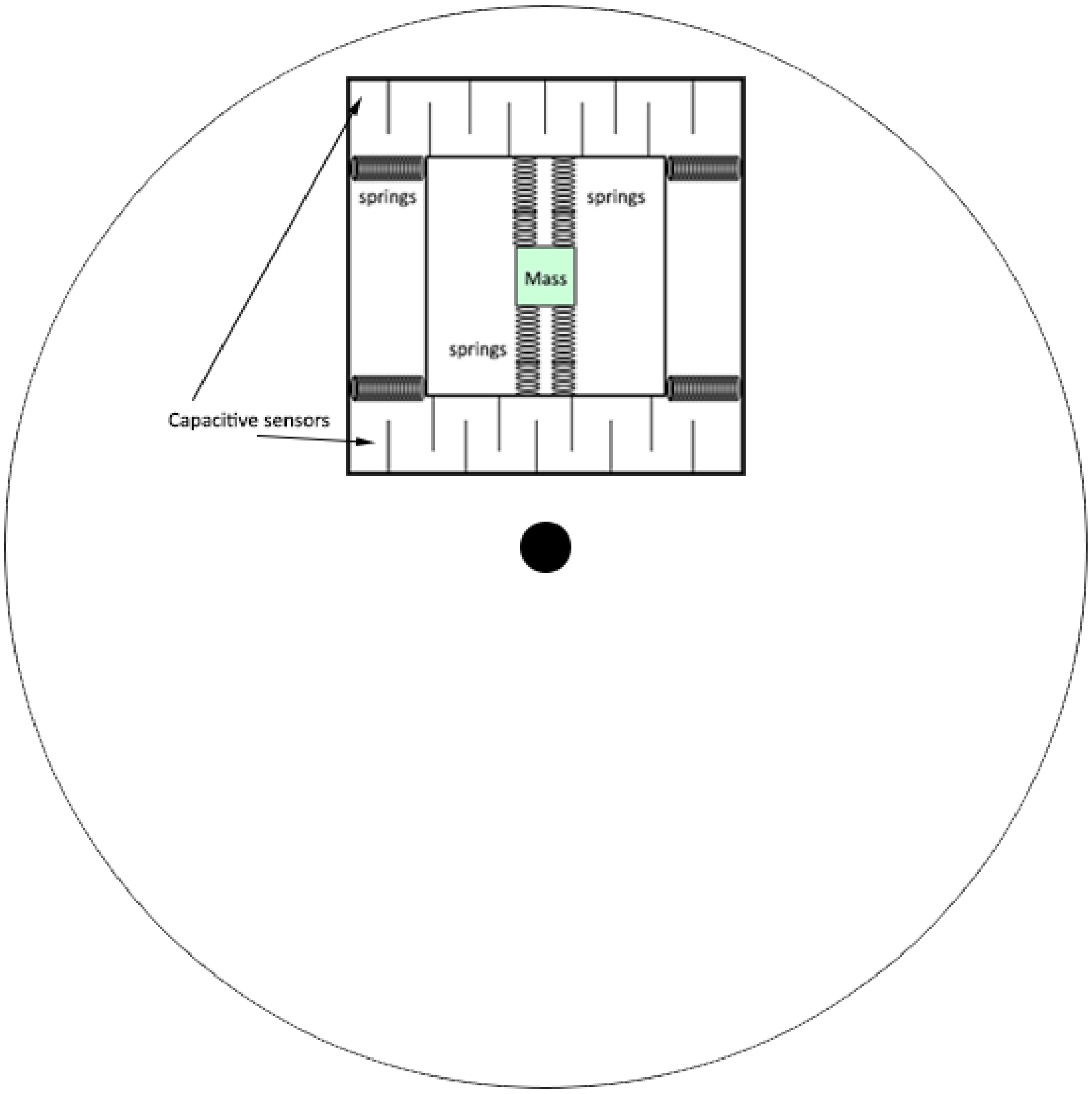


Figure: Diagram of MEMS gyroscope using Coriolis force

Hardware

The 32F429IDISCOVERY Discovery kit leverages the capabilities of the STM32F429 high-performance microcontrollers, to allow users to develop rich applications easily with advanced graphic user interfaces.

Figure: Top View of STM32F429 Discovery Board

STM32F429 Microcontroller Features

STM32F429ZIT6 microcontroller featuring 2 Mbytes of Flash memory, 256 Kbytes of RAM in an LQFP144 package

2.4" QVGA TFT LCD

USB OTG with Micro-AB connector

I3G4250D, ST MEMS motion sensor 3-axis digital output gyroscope

Six LEDs:

LD1 (red/green) for USB communication

LD2 (red) for 3.3 V power-on

Two user LEDs: LD3 (green), LD4 (red)

Two USB OTG LEDs: LD5 (green) VBUS and LD6 (red) OC (over-current)

Two push-buttons (user and reset)

64-Mbit SDRAM

Extension header for LQFP144 I/Os for a quick connection to the prototyping board and an easy probing

On-board ST-LINK/V2-B

USB functions:

Debug port

Virtual COM port

Mass storage

Mbed Enabled™

Board power supply: through the USB bus or from an external 3 V or 5 V supply voltage

Triple-Axis I3G4250D Gyroscope

The I3G4250D gyroscope sensor reads rotational motion at as accurate as ±250 degrees-per-second and provides 16 bits of data output for each axis. The gyroscope also uses SPI as its interface, therefore only requiring four wires and the use of pull-up resistors on the MOSI, MISO, CS and SCK lines. The I3G4250D chip has a Wide supply voltage: 2.4 V to 3.6 V input, however the full breakout circuit includes a voltage regulator to allow for 5V voltage supplies as well in order to expand compatibility. The gyroscope interfaces the STM32F429 microcontroller to transmit the current rate of rotation of the system in all three directions.

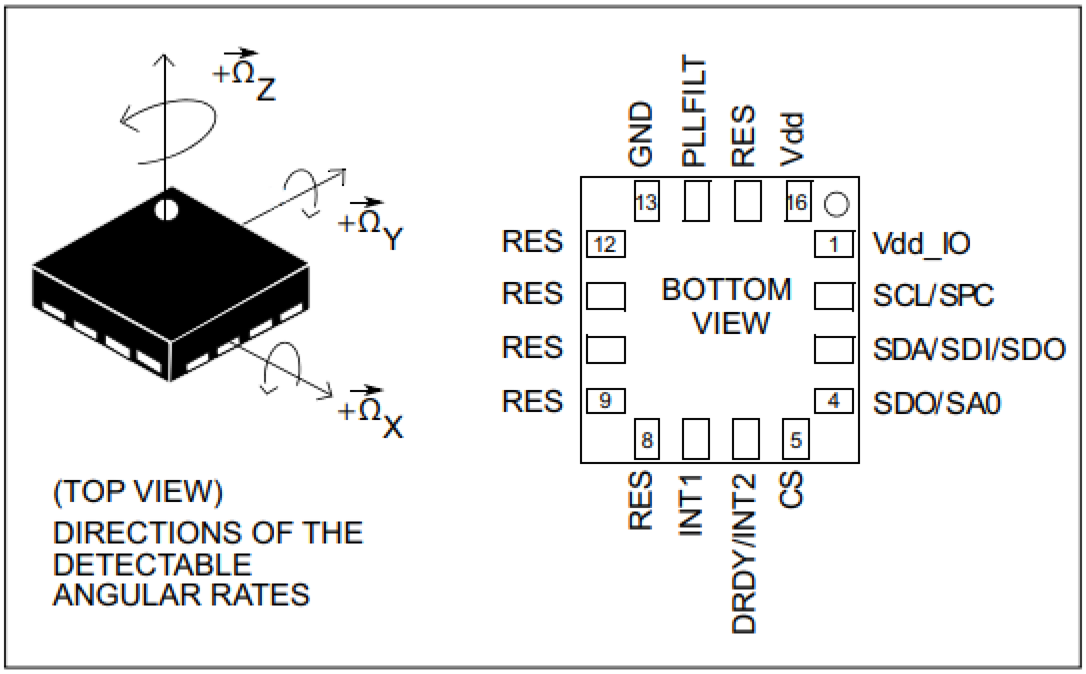


Figure: Gyroscope Pin Diagram

Software

All software for the STM32F429 microcontroller is written and compiled on PlatformIO mbed’s Compiler IDE, which provides a workspace for organizing and writing code, then compiling programs and flashing them to a USB-connected microcontroller. The mbed API provides a wealthy resource of available functions and libraries to use for projects. The mbed API includes a kernel level SPI library and provides additional libraries for interfacing I3G4250D gyroscope, all code for interfacing with these sensors is written by us, including the SPI library being written at the user level. All of the code is written in C++ with an object-oriented approach; classes are used to represent each of the sensors, and utilize a custom SPI class we created in order to cleanly organize the program’s architectural structure.

Software Main Program Flow

1. Attach your STM32F429 and power supply to any suitable part of your body

2. Interface the gyro using SPI to capture the angular velocities

3. Write the code to sample the angular velocities every 0.5s.

4. Convert those measurements to linear forward velocity.

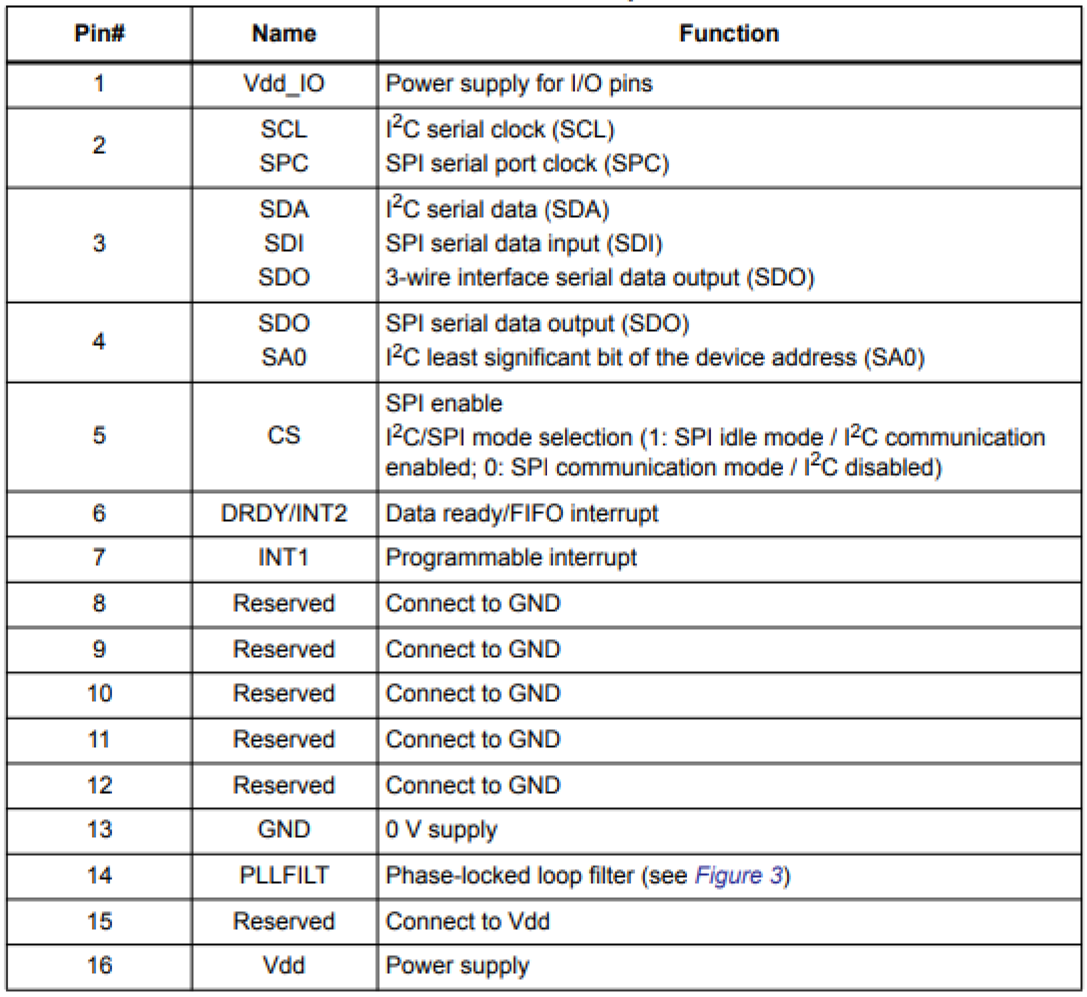
5. Record 20 seconds worth of data that can be exported through the USB or extracted from memory somehow.

6. Calculate the overall distance traveled during the 20 seconds of measurements.

Software Architecture

The main complexity of the program comes into play when actually having to interface with the sensors using SPI.

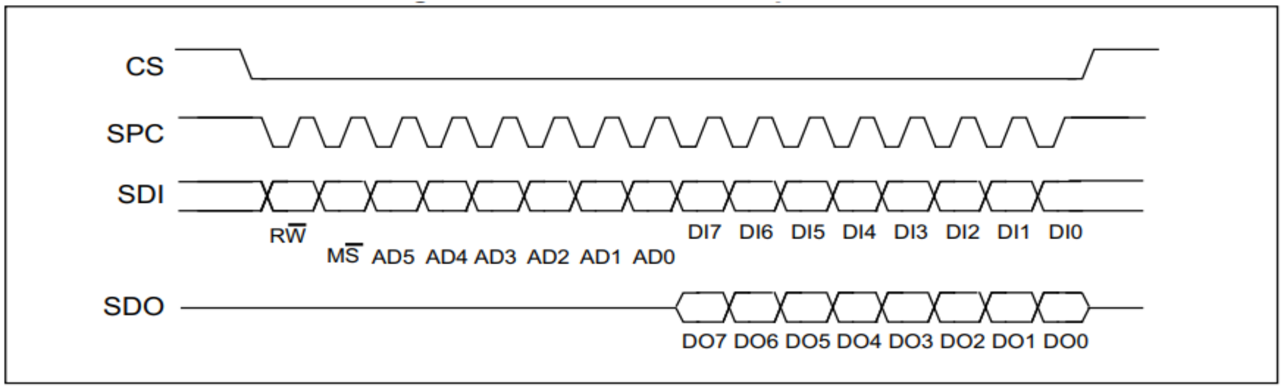
PIN DESCRIPTION OF THE GYROSCOPE I3G4250D



Pin#	Name	Function
1	Vdd_IO	Power supply for I/O pins
2	SCL SPC	I^2C serial clock (SCL) SPI serial port clock (SPC)
3	SDA SDI SDO	I^2C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
4	SDO SA0	SPI serial data output (SDO) I^2C least significant bit of the device address (SA0)
5	CS	SPI enable I^2C/SPI mode selection (1: SPI idle mode / I^2C communication enabled; 0: SPI communication mode / I^2C disabled)
6	DRDY/INT2	Data ready/FIFO interrupt
7	INT1	Programmable interrupt
8	Reserved	Connect to GND
9	Reserved	Connect to GND
10	Reserved	Connect to GND
11	Reserved	Connect to GND
12	Reserved	Connect to GND
13	GND	0 V supply
14	PLLFILT	Phase-locked loop filter (see *Figure 3*)
15	Reserved	Connect to Vdd
16	Vdd	Power supply

Class: SPI BUS

SPI bus interface - The SPI is a bus slave. The SPI allows writing to and reading from the device registers. The serial interface interacts with the application through 4 wires: CS, SPC, SDI, and SDO.


CS
SPC
SDI
RW
MS AD5 AD4 AD3 AD2 AD1 AD0
DI7 DI6 DI5 DI4 DI3 DI2 DI1 DI0
SDO
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0

CS is the serial port enable and is controlled by the SPI master. It goes low at the start of the transmission and returns to high at the end. SPC is the serial port clock and is controlled by the SPI master. It is stopped high when CS is high (no transmission). SDI and SDO are, respectively, the serial port data input and output. These lines are driven at the falling edge of SPC and should be captured at the rising edge of SPC.

Both the read register and write register commands are completed in 16 clock pulses, or in multiples of 8 in case of multiple read/write bytes. Bit duration is the time between two falling edges of SPC. The first bit (bit 0) starts at the first falling edge of SPC after the falling edge of CS while the last bit (bit 15, bit 23, etc.) starts at the last falling edge of SPC just before the rising edge of CS.

Bit 0: RW bit. When 0, the data DI(7:0) is written to the device. When 1, the data DO(7:0) from the device is read. In the latter case, the chip drives SDO at the start of bit 8.

Bit 1: MS bit. When 0, the address remains unchanged in multiple read/write commands. When 1, the address is auto-incremented in multiple read/write commands.

Bit 2-7: address AD(5:0). This is the address field of the indexed register.

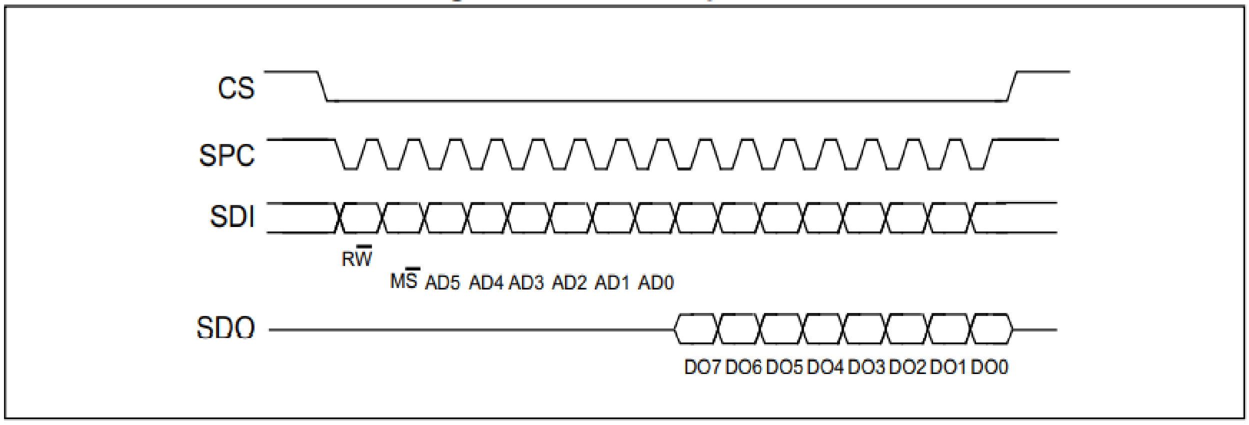
Bit 8-15: data DI(7:0) (write mode). This is the data that is written to the device (MSb first).

Bit 8-15: data DO(7:0) (read mode). This is the data that is read from the device (MSb first).

In multiple read/write commands, further blocks of 8 clock periods are added. When the MS bit is 0, the address used to read/write data remains the same for every block. When the MS bit is 1, the address used to read/write data is incremented at every block.

The function and the behavior of SDI and SDO remain unchanged

SPI READ PROTOCOL



The SPI read command is performed with 16 clock pulses. A multiple byte read command is performed by adding blocks of 8 clock pulses to the previous one.

Bit 0: READ bit. The value is 1.

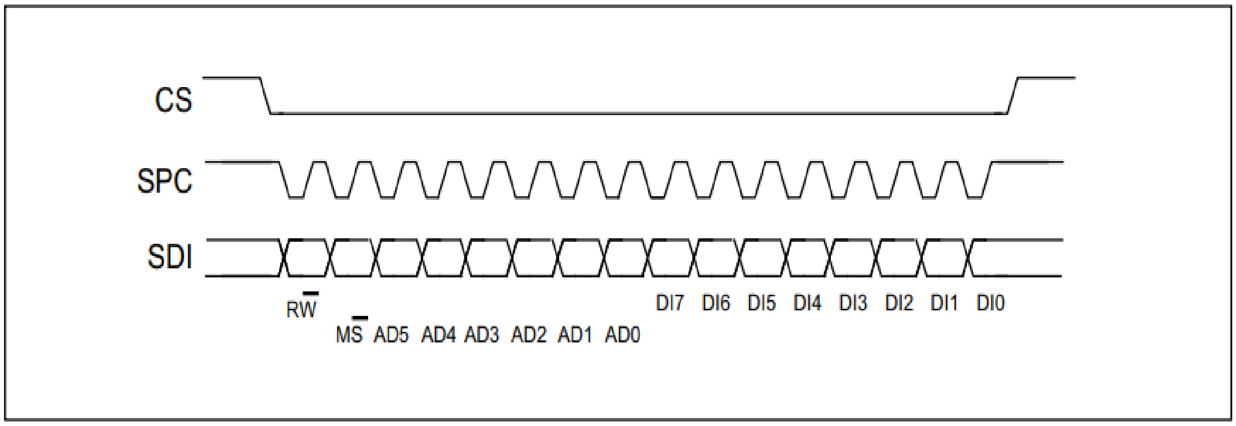
Bit 1: MS bit. When 0, does not increment address; when 1, increments address in multiple reads.

Bit 2-7: address AD(5:0). This is the address field of the indexed register.

Bit 8-15: data DO(7:0) (read mode). This is the data that is read from the device (MSb first).

Bit 16-...: data DO(...-8). Further data in multiple byte reads.

SPI WRITE PROTOCOL



The SPI write command is performed with 16 clock pulses. A multiple byte write command is performed by adding blocks of 8 clock pulses to the previous one.

Bit 0: WRITE bit. The value is 0.

Bit 1: MS bit. When 0, does not increment address; when 1, increments address in multiple writes.

Bit 2 -7: address AD(5:0). This is the address field of the indexed register.

Bit 8-15: data DI(7:0) (write mode). This is the data that is written to the device (MSb first).

Bit 16-...: data DI(...-8). Further data in multiple byte writes.

OUTPUT REGISTER MAPPING

Name	Type	Register Address		Default
		Hex	Binary	
CTRL_REG1	rw	20	010 0000	00000111
OUT_X_L	r	28	010 1000	Output
OUT_X_H	r	29	010 1001	Output
OUT_Y_L	r	2A	010 1010	Output
OUT_Y_H	r	2B	010 1011	Output
OUT_Z_L	r	2C	010 1100	Output
OUT_Z_H	r	2D	010 1101	Output

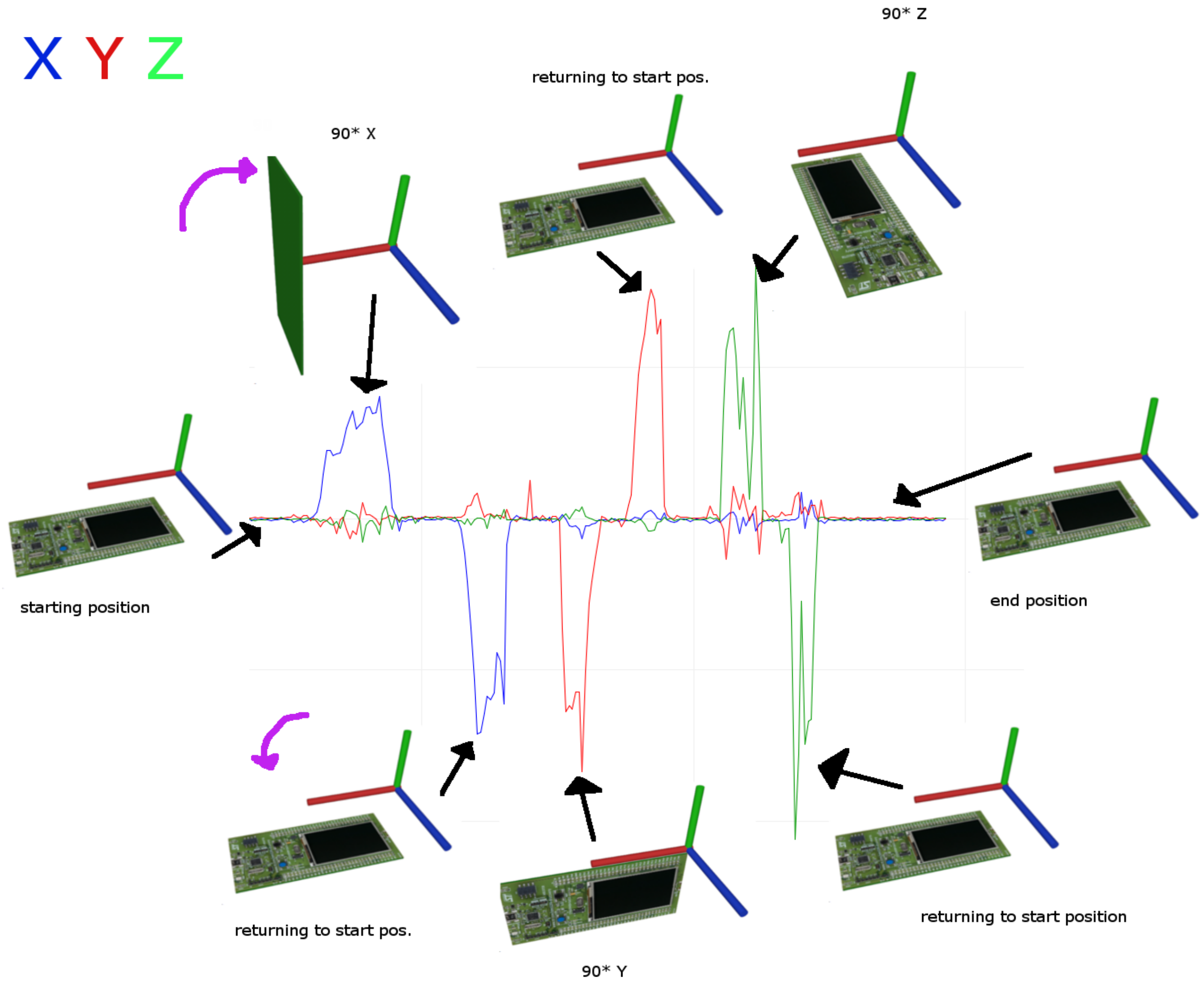
CTRL_REG1

DR1	DR0	BW1	BW0	PD	Zen	Yen	Xen

DR1-DR0	Output data rate selection. Refer to *Table 22*
BW1-BW0	Bandwidth selection. Refer to *Table 22*
PD	Power-down mode enable. Default value: 0 (0: power-down mode, 1: normal mode or sleep mode)
Zen	Z-axis enable. Default value: 1 (0: Z-axis disabled; 1: Z-axis enabled)
Yen	Y-axis enable. Default value: 1 (0: Y-axis disabled; 1: Y-axis enabled)
Xen	X-axis enable. Default value: 1 (0: X-axis disabled; 1: X-axis enabled)

In order to begin Reading the values from the Gyroscope, we first have to make sure that the values in the control register1 keep the Device ON and the x, y, z axis enabled.This we do by writing to the register at 20H address location. Then we read each of the 16 bit pair output registers along x, y and z axis.

Gyroscope Angular Velocity with respect to x, y, z axis



Conclusion

The most important lesson learned from this project would be to program incrementally with hardware testing devices. Most of the issues and debugging involved resulted from not using an oscilloscope or logic analyzer to look at the actual signals being generated by the program at an early stage. Shortcomings were seen in the procedure as a lot of values had to be assumed to convert the angular velocity to linear velocity. Outputs for walking and Running Cases are attached as shown along with the code (main.cpp).

Future scope would be to interface with an LCD to obtain the values on the screen rather than serial monitor as debugging would be easier.

Need For Speed – Code

main.cpp

```
/*
Embedded Challenge - Need for Speed
Author: Cavin Roger Nunes
        Neil Francisco Araujo
*/

#include "mbed.h"
#include <stdlib.h>
#include <string.h>

SPI spi(PF_9, PF_8, PF_7); // mosi, miso, sclk
DigitalOut cs(PC_1); //CS

// Documents
// Manual for dev board: https://www.st.com/resource/en/user_manual/um1670-discovery-kit-with-
stm32f429zi-mcu-stmicroelectronics.pdf
// Gyroscope datasheet: https://www.mouser.com/datasheet/2/389/dm00168691-1798633.pdf

int main() {
    int i=0;
    // Chip must be deselected
    cs = 1;
    // Setup the spi for 8 bit data, high steady state clock,
    // second edge capture, with a 1MHz clock rate
    spi.format(8,3);
    spi.frequency(1000000);
    cs = 0;                         //chip select = 0
      spi.write(0x20);              //SPI Access the Control_Register1 at address 20H and writing to it
      spi.write(0x0F);              //SPI Setting last 4 bits (0000 1111) to enable the x, y, z axis on
the gyro
    cs = 1;                         //Deselecting the chip after initialization chip select = 1
    cs = 0;                         //chip select = 0
      spi.write(0xA0);              //SPI Reading from the Control_Register1.
      int Controlreg_1 = spi.write(0x00);     //Passing 0000 values As no data is passed in SPI read
operation
      printf("Control Register 1  = 0x%X\n", Controlreg_1);  //Printing Control_Register1 values to
terminal
    cs = 1;                         //Deselecting the chip. chip select = 1

    //Initialize variables
    int step=0;                     // No of steps travelled by the User
    int cumsum =0;                  // Cumulative angular velocity for the duration of a step
    int count=0;                    // No of readings collected per step
    int firststep=0;                // Identify  when the user goes from stationary to a motion and vice
versa
    int distance=0;                 // Total distance travelled by the user

    while (i<40)                    // Assume i from 0 to 40. Each loop is 0.5s. Hence total runtime
would be 20s
    {
      cs = 0;                       // Select the device by setting chip select low
      spi.write(0xE8);              // Send 0xE8, the command to read X_L register
      short X_L = spi.write(0x00);  // Send a dummy byte to receive the contents register pair
      short X_H = spi.write(0x00);  // Send a dummy byte to receive the contents register pair
      short X_axis = X_H << 8 | X_L;// Shifting values 8 bit left to combine both high and low reg values
      float scaler = 8.75 * 0.001;  // Scalar multiplier with offset to normalize the angular velocity
      short final_X=(short)((float)X_axis*scaler); // Normalizing gyroscope values along x axis
      cs = 1;                       // Deselect the device
      wait_us(166666);              // Delay between reading the axis
      cs = 0;                       // Select the device by setting chip select low
      spi.write(0xEA);              // Send 0xEA, the command to read Y_L register
      short Y_L = spi.write(0x00);  // Send a dummy byte to receive the contents register pair
      short Y_H = spi.write(0x00);  // Send a dummy byte to receive the contents register pair
      short Y_axis = Y_H << 8 | Y_L;// Shifting values 8 bit left to combine both high and low reg values
```

```
        scaler = 8.75 * 0.001;          // Scalar multiplier with offset to normalize the angular velocity
        short final_Y=(short)((float)Y_axis*scaler); // Normalizing gyroscope values along x axis
        cs = 1;                         // Deselect the device
        wait_us(166666);                // Delay between reading the axis
        cs = 0;                         // Select the device by setting chip select low
        spi.write(0xEC);                // Send 0xEC, the command to read Z_L register
        short Z_L = spi.write(0x00);    // Send a dummy byte to receive the contents register pair
        short Z_H = spi.write(0x00);    // Send a dummy byte to receive the contents register pair
        short Z_axis = Z_H << 8 | Z_L;// Shifting values 8 bit left to combine both high and low reg values
        scaler = 8.75 * 0.001;          // Scalar multiplier with offset to normalize the angular velocity
        short final_Z=(short)((float)Z_axis*scaler); // Normalizing gyroscope values along x axis
        cs = 1;                         // Deselect the device
        wait_us(166666);                // Delay between reading the axis
        //Printing the output of the Gyroscope along all x,y,z axis at every 0.5s
        printf(" Final X_axis = %d\n", final_X);
        printf(" Final Y_axis = %d\n", final_Y);
        printf(" Final Z_axis = %d\n", final_Z);
        i++;                            //Counter for while loop condition (i<40)
        // If the value of Z is within these threshold then the user is stationary or between
        // the motion of 2 steps
        if(final_Z<=100 && final_Z>=-100){
          if(count!=0){
            int avgangvel=cumsum/count;
            printf("\nAverage angular velocity of the step: %d radians/sec\n",avgangvel);
            // Distance from the board to the center or the top of the leg = 19.68 inches or 0.5 meters
            // Therefore the average Linear forward moment is angular velocity by 0.5
            int avglinvel= (abs(avgangvel)* 0.5)/10;
            // distance = linear velocity X time which we consider as 0.2 seconds the intervals we are
taking
            // we add all distances of each step to get the total distance
            distance=distance + (avglinvel *0.5);
            printf("Average Linear forward moment: %d metres/sec\n",avglinvel);
            printf("Distance travelled: %d metres\n\n",distance);
          }
          // Reset values for the next step
          cumsum=0;
          count=0;
        }

        // If the value of Z is outside the threshold of 50 then the user is in the motion of a step
        // and cumulative velocity over a step is taken
        if(!(final_Z<=50 && final_Z>=-50)){
          cumsum=cumsum + abs(final_Z);
          //If the value of firststep changes from 0 to 1 then we count it as a step other velocity values
          // in a step thus do not affect the no of steps taken
          if(firststep==0){
             firststep=1;
             step=step+1;
             printf("\nNo of steps taken: %d\n\n", step);
          }
          step=step+1;
             printf("\nNo of steps taken: %d\n\n", step);
          // Increment count to reflect the number of reading
          count=count+1;
        }
      }
}
```

Need For Speed – Code Output
Walking case

```
Final X_axis = -10
 Final Y_axis = -2
 Final Z_axis = -77

Average angular velocity of the step: 52 radians/sec
Average Linear forward moment: 2 metres/sec
Distance travelled: 1 metres


No of steps taken: 3

 Final X_axis = 6
 Final Y_axis = -24
 Final Z_axis = -81

Average angular velocity of the step: 77 radians/sec
Average Linear forward moment: 3 metres/sec
Distance travelled: 2 metres


No of steps taken: 4

 Final X_axis = 7
 Final Y_axis = 31
 Final Z_axis = 130

No of steps taken: 5

 Final X_axis = -10
 Final Y_axis = 9
 Final Z_axis = -40

Average angular velocity of the step: 105 radians/sec
Average Linear forward moment: 5 metres/sec
Distance travelled: 4 metres

 Final X_axis = 19
 Final Y_axis = -77
 Final Z_axis = -36
 Final X_axis = -1
 Final Y_axis = 69
 Final Z_axis = 114

No of steps taken: 6

 Final X_axis = -21
 Final Y_axis = 24
 Final Z_axis = 1

Average angular velocity of the step: 114 radians/sec
Average Linear forward moment: 5 metres/sec
Distance travelled: 6 metres

 Final X_axis = 10
 Final Y_axis = -6
 Final Z_axis = -81

No of steps taken: 7

 Final X_axis = 3
 Final Y_axis = 7
 Final Z_axis = 18

Average angular velocity of the step: 81 radians/sec
Average Linear forward moment: 4 metres/sec
Distance travelled: 8 metres

 Final X_axis = -11
 Final Y_axis = 20
 Final Z_axis = 74

No of steps taken: 8

 Final X_axis = -1
 Final Y_axis = -4
 Final Z_axis = -42

Average angular velocity of the step: 74 radians/sec
Average Linear forward moment: 3 metres/sec
Distance travelled: 9 metres

 Final X_axis = 10
 Final Y_axis = 6
 Final Z_axis = -29
 Final X_axis = 7
 Final Y_axis = 37
 Final Z_axis = 15
 Final X_axis = -4
 Final Y_axis = 6
 Final Z_axis = 62

No of steps taken: 9

 Final X_axis = -18
 Final Y_axis = 12
 Final Z_axis = 16

Average angular velocity of the step: 62 radians/sec
Average Linear forward moment: 3 metres/sec
Distance travelled: 10 metres

 Final X_axis = 5
 Final Y_axis = 8
 Final Z_axis = -44
 Final X_axis = 14
 Final Y_axis = 5
 Final Z_axis = -35
 Final X_axis = 0
 Final Y_axis = -7
 Final Z_axis = 1
 Final X_axis = -6
 Final Y_axis = -15
 Final Z_axis = 104

No of steps taken: 10

 Final X_axis = -18
 Final Y_axis = 6
 Final Z_axis = 32

Average angular velocity of the step: 104 radians/sec
Average Linear forward moment: 5 metres/sec
Distance travelled: 12 metres

 Final X_axis = 8
 Final Y_axis = -6
 Final Z_axis = -126

No of steps taken: 11

 Final X_axis = 11
 Final Y_axis = -66
 Final Z_axis = -45

Average angular velocity of the step: 126 radians/sec
Average Linear forward moment: 6 metres/sec
Distance travelled: 15 metres

 Final X_axis = -8
 Final Y_axis = 15
 Final Z_axis = 111

No of steps taken: 12

 Final X_axis = -8
 Final Y_axis = -22
 Final Z_axis = -8

Average angular velocity of the step: 111 radians/sec
Average Linear forward moment: 5 metres/sec
Distance travelled: 17 metres

 Final X_axis = 23
 Final Y_axis = 44
 Final Z_axis = -60

No of steps taken: 13

 Final X_axis = 22
 Final Y_axis = -18
 Final Z_axis = -22

Average angular velocity of the step: 60 radians/sec
Average Linear forward moment: 3 metres/sec
Distance travelled: 18 metres

 Final X_axis = -12
 Final Y_axis = 50
 Final Z_axis = 104

No of steps taken: 14

 Final X_axis = -40
 Final Y_axis = -17
 Final Z_axis = -1

Average angular velocity of the step: 104 radians/sec
Average Linear forward moment: 5 metres/sec
Distance travelled: 20 metres

 Final X_axis = 26
 Final Y_axis = 22
 Final Z_axis = -55

No of steps taken: 15

 Final X_axis = 17
 Final Y_axis = -61
 Final Z_axis = -25

Average angular velocity of the step: 55 radians/sec
Average Linear forward moment: 2 metres/sec
Distance travelled: 21 metres

 Final X_axis = -15
 Final Y_axis = 80
 Final Z_axis = 68

No of steps taken: 16

 Final X_axis = -27
 Final Y_axis = -13
 Final Z_axis = 35

Average angular velocity of the step: 68 radians/sec
Average Linear forward moment: 3 metres/sec
Distance travelled: 22 metres

 Final X_axis = 5
 Final Y_axis = 0
 Final Z_axis = -80

No of steps taken: 17

 Final X_axis = 32
 Final Y_axis = -43
 Final Z_axis = -74

Average angular velocity of the step: 80 radians/sec
Average Linear forward moment: 4 metres/sec
Distance travelled: 24 metres


No of steps taken: 18
```

Need For Speed – Code Output

Running/ Speed Walking case

```
 Final X_axis = -3
 Final Y_axis = -2
 Final Z_axis = 0
 Final X_axis = -1
 Final Y_axis = 0
 Final Z_axis = -9
 Final X_axis = -53
 Final Y_axis = -31
 Final Z_axis = -160

No of steps taken: 1


No of steps taken: 2

 Final X_axis = 31
 Final Y_axis = -45
 Final Z_axis = -25

Average angular velocity of the step: 160 radians/sec
Average Linear forward moment: 8 metres/sec
Distance travelled: 4 metres

 Final X_axis = -46
 Final Y_axis = -84
 Final Z_axis = -69

No of steps taken: 3

 Final X_axis = 44
 Final Y_axis = -82
 Final Z_axis = -111

No of steps taken: 4

 Final X_axis = -7
 Final Y_axis = 111
 Final Z_axis = 93

Average angular velocity of the step: 90 radians/sec
Average Linear forward moment: 4 metres/sec
Distance travelled: 6 metres


No of steps taken: 5

 Final X_axis = 42
 Final Y_axis = -98
 Final Z_axis = -122

No of steps taken: 6

 Final X_axis = -15
 Final Y_axis = 60
 Final Z_axis = 151

No of steps taken: 7

 Final X_axis = 16
 Final Y_axis = -43
 Final Z_axis = -185

No of steps taken: 8

 Final X_axis = -6
 Final Y_axis = 23
 Final Z_axis = 277

No of steps taken: 9

 Final X_axis = -20
 Final Y_axis = -4
 Final Z_axis = -167

No of steps taken: 10

 Final X_axis = 6
 Final Y_axis = 21
 Final Z_axis = 89

Average angular velocity of the step: 165 radians/sec
Average Linear forward moment: 8 metres/sec
Distance travelled: 10 metres
```

```
No of steps taken: 11

 Final X_axis = -58
 Final Y_axis = -23
 Final Z_axis = -131

No of steps taken: 12

 Final X_axis = 19
 Final Y_axis = 71
 Final Z_axis = 77

Average angular velocity of the step: 110 radians/sec
Average Linear forward moment: 5 metres/sec
Distance travelled: 12 metres


No of steps taken: 13

 Final X_axis = -65
 Final Y_axis = 16
 Final Z_axis = -119

No of steps taken: 14

 Final X_axis = 44
 Final Y_axis = -94
 Final Z_axis = -24

Average angular velocity of the step: 98 radians/sec
Average Linear forward moment: 4 metres/sec
Distance travelled: 14 metres

 Final X_axis = -52
 Final Y_axis = -47
 Final Z_axis = -24
 Final X_axis = 67
 Final Y_axis = -124
 Final Z_axis = -106

No of steps taken: 15

 Final X_axis = -10
 Final Y_axis = 89
 Final Z_axis = 194

No of steps taken: 16

 Final X_axis = 8
 Final Y_axis = -8
 Final Z_axis = -181

No of steps taken: 17

 Final X_axis = -10
 Final Y_axis = 7
 Final Z_axis = 183

No of steps taken: 18

 Final X_axis = -73
 Final Y_axis = -19
 Final Z_axis = -89

Average angular velocity of the step: 166 radians/sec
Average Linear forward moment: 8 metres/sec
Distance travelled: 18 metres


No of steps taken: 19

 Final X_axis = -23
 Final Y_axis = 89
 Final Z_axis = 172

No of steps taken: 20

 Final X_axis = -85
 Final Y_axis = -2
 Final Z_axis = -91

Average angular velocity of the step: 130 radians/sec
Average Linear forward moment: 6 metres/sec
Distance travelled: 21 metres
```

```
No of steps taken: 21

 Final X_axis = -14
 Final Y_axis = -22
 Final Z_axis = 209

No of steps taken: 22

 Final X_axis = -70
 Final Y_axis = 31
 Final Z_axis = -148

No of steps taken: 23

 Final X_axis = 40
 Final Y_axis = 19
 Final Z_axis = 119

No of steps taken: 24

 Final X_axis = -69
 Final Y_axis = 33
 Final Z_axis = -59

Average angular velocity of the step: 141 radians/sec
Average Linear forward moment: 7 metres/sec
Distance travelled: 24 metres


No of steps taken: 25

 Final X_axis = 15
 Final Y_axis = -54
 Final Z_axis = 228

No of steps taken: 26

 Final X_axis = -84
 Final Y_axis = -32
 Final Z_axis = -128

No of steps taken: 27

 Final X_axis = 70
 Final Y_axis = 2
 Final Z_axis = 122

No of steps taken: 28

 Final X_axis = -99
 Final Y_axis = -26
 Final Z_axis = -53

Average angular velocity of the step: 134 radians/sec
Average Linear forward moment: 6 metres/sec
Distance travelled: 27 metres


No of steps taken: 29

 Final X_axis = 46
 Final Y_axis = -129
 Final Z_axis = 192

No of steps taken: 30
```