Weight Lifting Performance Monitor

David Stoddard Electrical Engineering

Project Advisor: Dr. Marc Mitchell

April 26, 2019

Evansville, Indiana

Table of Contents

List of Figures

List of Tables

I. Introduction

A person who regularly engages in weight lifting oftentimes deals with a large degree of uncertainty in their workouts. Every time a weight lifter attempts a lift, that exercise may feel a little different. Based on what the lifter eats the day before, how much sleep they got the night before and the intensity of the previous day's work out, the feel of a workout a lifter engages in can change on a day to day basis. What this means is that a lifter cannot trust his own senses when trying to engage in precise workouts and needs assistance from an electronic device to receive accurate information about his workout. To this end I created a device that can be used by a lifter during his workout that will give that lifter as much information as possible about the lifts he is attempting. The impetus for this device came out of a request from Assistant Athletic Director Sonny Park who is in charge of the weight lifting program for athletes here at UE. He was looking for an inexpensive device capable of measuring four key components of an average lift, that could be used by the variety of teams in the weight room. I decided that this project had a high degree of relevance to me personally as I have been a member of the Swim and Dive team for four years now and after speaking more with Sonny we determined the main requirements for such a device would be to measure the peak velocity, peak force, peak power and average velocity of a standard Olympic lift such as a clean. Additional considerations for this device were to constrain the size as much as possible and for it to be able to be placed at different locations on the bar. I determined that the easiest way to achieve the measurements I need was to use a combination of an outside accelerometer sensor and the onboard timers on a microcontroller. With the acceleration and time values that I received from these components I was able to use several basic physic equations to calculate the measurements I needed and was able to accurately display them on my LCD screen. To this end my approach for this project had four main parts, ensure accurate communication between all devices, determine an appropriate way to find the length of time the lift took, correctly implement the equations I had to use and allow for user control for the display and input. I was able to successfully accomplish all portions of my initial approach and I successfully created a device that was able to measure all four of the components of a lift that were required. Additionally, I was able to constrain the size of the final device to a box measuring 5x2x2.5 inches and was able to implement a way for the device to easily be relocated to different parts of the weight bar.

II. Background

A. Impetus of Project

Over the summer of 2018, the Assistant Athletic Director for Sports Performance Sunny Park purchased a production quality device that monitors lifting performance for use in the weight room. I spoke to AAD Park and he expressed an interest in having more devices for use in the weight room. My main goal for this project was to produce a device of similar functionality for significantly less cost, allowing more athletes access to this information. This device was required to measure four key components to any Olympic style lift. These four components were peak velocity, peak force, peak power and average velocity. With this information the lifter is able set specific goals for different components and then adjust their weight to better hit those goals. A lifter is also able to keep track of his performance and analyze it over time. Additionally, this extra information can help to make a workout safer for a lifter by eliminating some of the guesswork from trying to decide on the proper amount of weight to put on the bar for each lift. There are many other ways that this kind of information can prove beneficial to a weight lifter, but they all go back to the concept that the more you know, the better you can do.

B. Technical Background

In this project the three devices communicate via I2C with the microcontroller acting as the Master and the sensor and LCD acting as slaves. Additionally, the microcontrollers onboard timers were used as a way to measure time. There were three major and several minor, technical hurdles that had to be addressed to allow this project to function properly. These major hurdles were communication between the three devices involved in this project (the accelerometer, the LCD screen and the microcontroller), ensuring a rapid enough sample rate of the accelerometer to produce accurate results, and finding an efficient way to use the onboard timers to measure the overall length of time of the lift. These three issues did not exist in a vacuum and are related to each other in their implementation. For example, to ensure a fast-enough sampling time from the sensor requires that the I2C bus used be available almost all the time to receive data from the sensor. This impacts the ability of the LCD screen to be communicated with as there is a limited time when the I2C bus is free to be used. Additionally, the starting and stopping of the onboard timers for use in measuring time are reliant on what values the sensor is outputting at a given time. The minor issues involved with this project are also tied back into the major issues, namely the availability of the I2C bus and when the LCD screen is communicated with causing problems with the ability of the user to change what is being displayed. A more in depth look at these problems can be found in section III B. In addition to the technical problems associated with this project, there were also several non-technical issues that had to be addressed such as the overall size of the device and its portability.

III. Project Design

A. Hardware

The first step in the design of this project was to determine which components I would use in the creation of this device. I needed to pick a microcontroller that was able to communicate via I2C, had accurate onboard timers, had several available pins to act as user input and was ideally small in size. I determined that the STM32L432, as seen in **Figure A**, met all my requirements and had the added benefit of being similar in function to the STM32F446, a device that I had used extensively my Junior year. This made it far easier for me to use the L432 and meant that I was already familiar with several troubleshooting techniques that would work for this microcontroller.

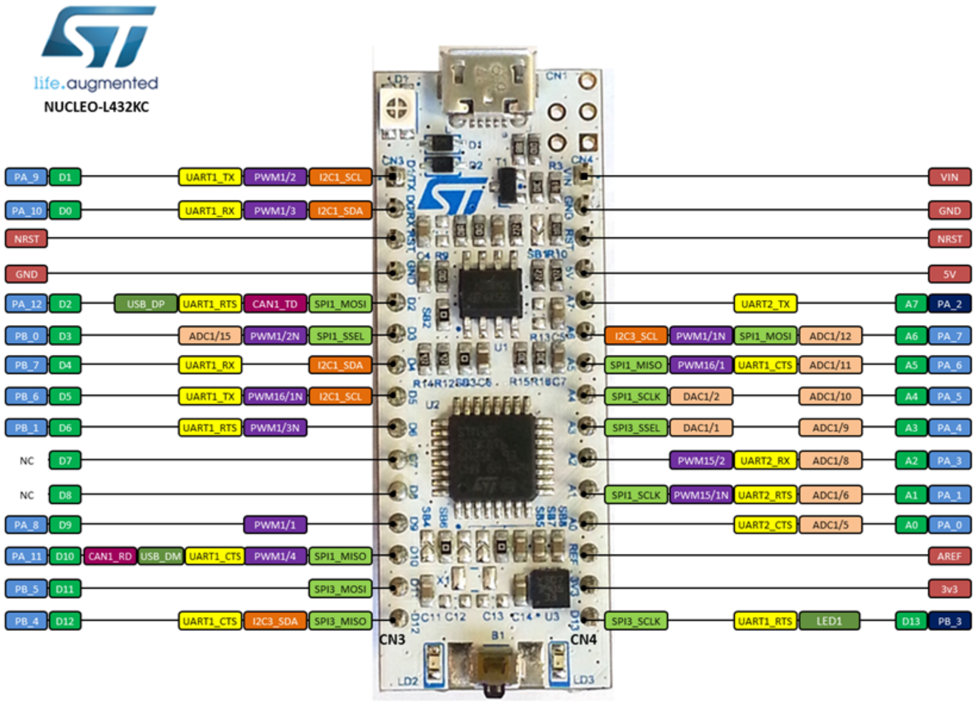

Figure A: STM32L432KC

My initial requirements for a sensor that I had to pick for this project needed to have a 6-axis accelerometer/gyroscope and be available to be communicated with via I2C. I realized later on in my design process that while the gyroscope was a useful tool to ensure further accuracy in my project, it was not a necessary inclusion. Before I made that determination however, I had already purchased two MPU-6050 6-axis accelerometer/gyroscopes for use in this project, as seen in **Figure B**. I picked these devices due to them meeting all of my above requirements and doing so for several dollars less than equivalent sensors. I initially purchased two sensors because at the beginning of my design process I had decided that I would use two sensors to increase the accuracy of my project (you can see where the second sensor was supposed to be in figure ?????????). However, as the design process went on I determined that constraints put on my system by trying to use the I2C bus to communicate with two sensors, outweighed the benefits and so I cut the second sensor from the final design.

Figure B: GY-521 MPU 6050 6-Axis Accelerometer/Gyroscope

The primary requirement I initially had for my LCD screen was simply that it be available for I2C communication with my master device. After further research into the function of LCD devices I determined that the LCD having its own processing chip was an additional requirement. With this feature I would be able to just send the LCD characters, and have it display them rather than having to determine what the hexadecimal value was for each character I wanted to display. I decided on purchasing the SERLCD, pictured in **Figure C,** as it met both of my requirements for this component and had been used in the pass by several of my peers with great success.

Figure C: SERLCD

The next step after determining my components was to design a circuit board using PCB artist to connect them all, pictured in **Figure D**. This circuit board had to two purposes, the first was to ensure a proper electrical connection between all the devices used in this project, and the second was to reduce the required space the device would take up by eliminating connecting wires. The design of the circuit board was straightforward as the majority of the design was connecting pins on the three devices. Addition pieces on the circuit board include pullup resistors for the I2C bus and two voltage regulators to step down the 9-volt power source down to 5 volts to power the L432 and down to 3.3 volts for the sensor and LCD. The board was designed so that female to male header pins could be soldered into it and all the components could be simply slotted in. This was done to allow for the components to be easily switched out in case one of them suffered a catastrophic failure.

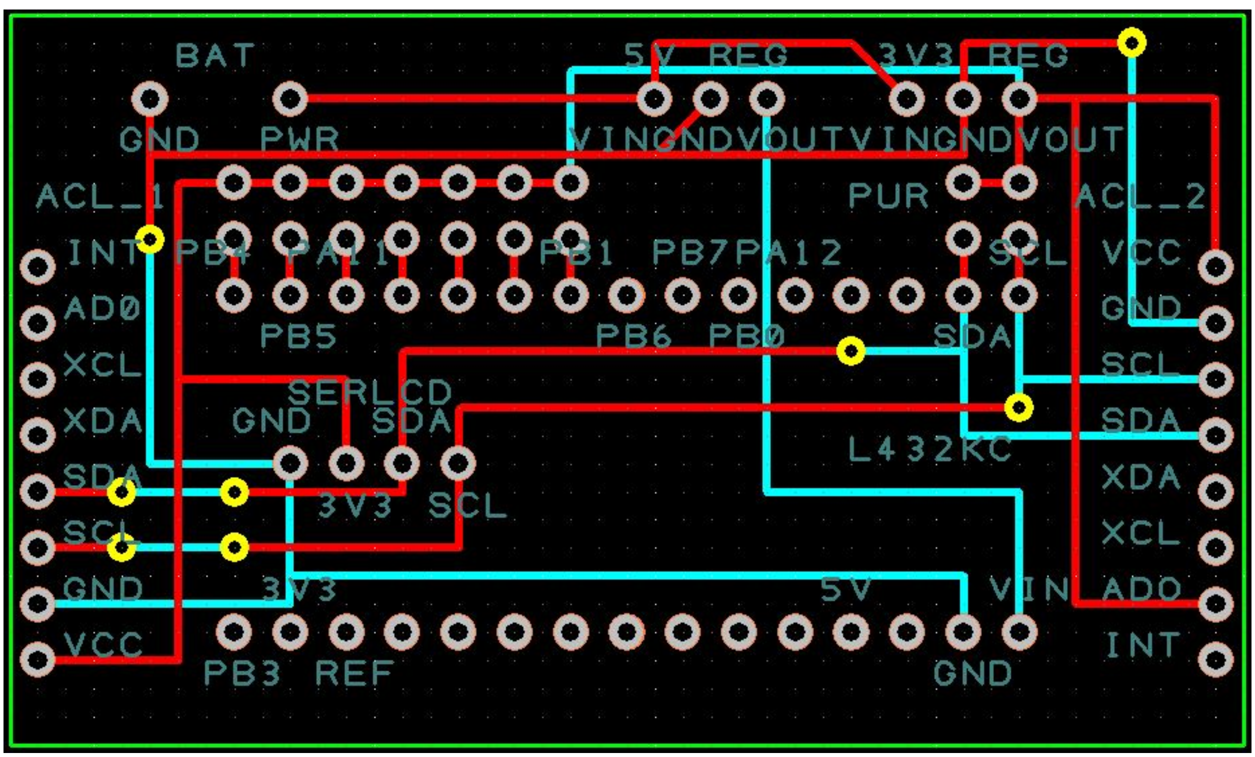

Figure D: PCB Artist Circuit Board

The main requirement for the housing of this was to sit easily on a weight bar and to be as small as possible while still having sufficient space to allow the components to sit inside. Due to the highly specific requirements for this casing I opted to design and 3-D print a custom box, as pictured in **Figure E**, and **Figure F**. measuring 5x2x2.5 inches with space for the LCD screen to be seen, holes for the user input buttons to be inserted, and a curved bottom to allow the device to sit on a weight bar.

Figure E: Top of 3-D Box

Figure F: Side View of 3-D Box

In addition to the specific components, three generic push buttons and a 9-volt battery were used to allow for user input and to power the overall device.

B. Software

The three main technical hurdles that had to be over come for this project were all solved with software. In this section I will detail the overall function of the code used in this project and then address how that code solved the three problems detailed in Section II B. Full code is located in Appendix A.

1. Code Overview

The first several functions in this code are the setup for the various protocols used such as setting up the I2C bus, preparing the timers used, and enabling the correct functions on the L432's pins. Additionally, there is a function at the beginning that sends values to the sensor to configure it properly and another one to clear the LCD display. The only other functions that appear in the main code of this project are the functions used to read values from the sensor and to display data to the LCD screen. All the remaining functions are either used to accomplish these tasks or are called in the interrupt that the code uses. The program works by continuously receiving data from the sensor via the function Update_Sensors. This function works by using I2C protocols to read the values of each component of acceleration from the sensors internal registers and then loads these values into their associated variables. The sensor produces 16-bit values for each component of acceleration, so it is necessary to read the most significant bits first, store them in a temporary variable, and then combine then with the least significant bits to get the final value. The Display_Data function is set up so that every 500 iterations through the main code it will be called. When the device is first turned on it will display the bar weight to allow the user to select what value they wish to input for the total weight of the bar. Then after the user has finished their selection and pressed the mode button once, the function will only display one of the four measurements until the reset button is pushed. The reason this function is located here and setup like this is detailed in the next section as it is related to the availability of the I2C bus. While the sensor is continuously updating there is an interrupt that is setup to trigger off every 1 millisecond. When this interrupt triggers off, the average of the most recent 5 acceleration values is taken using the function Average_Arrays. That average is then used in the function Do_Math to resolve the three components into a single magnitude of acceleration. This single acceleration value is compared to the previous acceleration value to produce an acceleration difference value (an absolute value of this variable is taken to keep it a positive number). Back in the interrupt handler this difference value is compared to a pre-determined threshold value (this threshold value was determined through testing) to determine if the lift has started, to start the other timer if it has been and to set a flag. If this threshold is hit and the flag had been set the function Equations is called, where the math to determine the final measurements is performed using the collected variables. This method is explained in the next section as it is related to the technical hurdle of accurately measuring the time of a lift. Also located in the interrupt handler is a check to see if the user input buttons have been depressed. This was done to address the issue of buttons "bouncing". In the event that the reset button has been pressed, all variables are cleared, and the display is set back to the bar weight.

2. Hurdles Addressed

Even though there are three distinct hurdles they need to be discussed as one connected issue as they are all heavily influenced by one another. The most effective method that I could come up with to determine the overall time of the lift was to use a threshold value and compare that to the magnitude of acceleration to determine when the bar begins and ends its movements. While sitting stationary with no one moving the device, the accelerometer will have a magnitude of acceleration of 1g. Setting this as the base or "0" value I could then determine that whenever the sensor read a value significantly above this base, the lift must be beginning. When this threshold was hit I would be able to start a timer to measure the lift and set a flag indicating the lift had begun. When this threshold change was hit again, it would indicate that the bar had stopped moving and the lift had ended, allowing me to stop the timer, get the time variable I need, and reset the flag for the lift. This solves one of my primary hurdles for this project, however in order for this method to work the sensor values must be continuously updated as rapidly as possible. For this to occur I decided to eliminate the second sensor, as I would be able to receive double the number of samples in the same time period with only one sensor. This allowed me to solve my hurdle about ensuring a sufficiently fast sample rate. However, this presented a significant issue in addressing the hurdle of sufficient communication between all three devices. Due to the fact that the I2C bus was nearly always being used by the sensor I had to work out a way to allow my LCD screen to be communicated with in such a way so as to not interrupt the sensor data transmission any more than necessary. The solution I arrived at for that was the simply have the display continuously updated every 500 times the sensors were read (approximately ever half second). Attempting to place the Display_Data function anywhere else in my code cause the data transmission from the sensor to fail and would result in the code becoming stuck waiting for transmission. These solutions addressed the main technical challenges I faced and were successful in allowing the project to function.

C. Impact, Safety, and Manufacturing Concerns

The overall impact from this project was low. Environmentally, the device is very small, so the amount of plastic used in the construction of the 3-D box is very low. Additionally, the plastic is biodegradable in approximately 20 years, so it won't be sitting in a landfill until the end of time. Aside from the plastic the device is made of, there is the usual minor environmental impact from the creation and shipping of small electronic components, and batteries. This is however, a onetime impact in the case of the electronics, and a rare impact in the case of the battery. Ideally this impact could be further mitigated by the introduction of a rechargeable power source over a disposable battery. The political impact of this project is absolutely zero. Unless politicians develop an interest in powerlifting, there is not much chance of this device having any sort of political impact. There is a potential for a social impact, if the device's accuracy is improved. Producing a device that allows an average person to afford a way to optimize their workout could be patented and used as publicity for the University.

The main safety issues involved in this project come on the user end of the equation and are only a problem is the user is inexperienced in lifting. As this device is to be used while someone is engaged in weight lifting exercises, there is the potential for someone to cause harm to themselves due to improper lifting technique. There is nothing that this project can do to address this issue however, as the device is simply a measuring tool, not an instruction tool. Ideally the only people using this device would be individuals who are at least competent in lifting enough to not harm themselves. Aside from this consideration, this project adhered to all IEEE ethical standards, and is designed to not cause direct harm to anyone. From a health viewpoint, this project has the potential to positively affect the health of a user. An optimized lifting workout can help a user maximize their effectiveness while at a gym and as a result, achieve better results in the same time.

From a manufacturing standpoint, since the initial work is completed this project should be easy to replicate. The 3-D box the device is housed in will be able to be copied by any 3-D printer that accepts its file type. The sensors, the microcontroller, and the LCD screen are available wholesale and are cheaper to buy in bulk than singles. The custom circuit board can be reordered with ease. The only portion of the manufacturing that requires semi-skilled worked is the assembly of the circuit board with the components, and the loading of the program onto the microcontroller. As far as sustainability goes, the device is secured against the type of sudden violent movement associated with weight lifting exercises and is able to have its power source easily recharged. These two factors allow for very minimal maintenance of the device and allow it to survive for use for a number of years.

IV. Results

This project performed as it was supposed to and was able to successfully measure the peak velocity, peak power, peak force and average velocity of a standard Olympic lift such as a clean. It was housed in a box 5x2x2.5 inches in dimensions and was able to be easily moved from one part of a weight bar to another. The user is able to input the total amount of weight on the weight bar, is able to cycle between the four measurements and is able to reset the all the variables on the device with the push of a button. It weighs approximately 6 ounces, so it does not unduly affect the overall lift of a bar weighing between 45-300 pounds.

In the future there are some additional features I would like to add to this device to make it more useful. I would like to add in the ability to store user data on the device so a lifter can track their progress over time. This would require the inclusion of some data storage device to keep all of this information even when the device is powered off. I would also like to add in the ability to store the measurements for multiple lifts in a set. As it stands right now after each repetition the display is updated and the previous rep is lost. I would like to have a used be able to perform their entire set and then go back and look at the values for each individual repetition. This would again require the inclusion of a data storage device to hold onto this information. Finally, I would like to separate the data collection and display portion of this device is two separate devices that communicate wirelessly to further reduce the size of the device that is placed on the weight bar.

V. Conclusion

The Weight Lifting Performance Monitor met the minimum requirements that were set forth in the project proposal submitted last semester. It successfully measured the four components for an Olympic lift that were set out in that proposal. Additionally, the project was completed for less than the requested budget, as can be seen in **Table 1** and **Table 2**. Furthermore, this project successfully met some of its additional constraints to make it more useful, including its size requirements and ability to have a user interact with it. In speaking with Sonny Park, he has indicated an intent to test it over the Summer of 2019 and to see if he will be able to implement it in weight lifting workouts in the Fall Semester of 2019. The information this device can provide has the potential to allow the University of Evansville athletes to further optimize their workouts and allow them to compete at a higher level than they have before.

Table 1: Estimated Costs

Item	Units	Cost Per Unit	Total
Arduino Acceleration + Gyroscopic Sensor	2	$25 USD	$50 USD
STM32L432KC Microcontroller	1	$25 USD	$25 USD
LCD Display	1	$10 USD	$10 USD
Spool of 3D Printer Material	1	$25 USD	$25 USD
Custom Circuit Board	3	$160 USD	$160 USD
Total Costs	$270 USD		

Table 2: Final Costs

Item	Units	Cost Per Unit	Total
Arduino Acceleration + Gyroscopic Sensor	2	$20 USD	$40 USD
STM32L432KC Microcontroller	1	$20 USD	$20 USD
LCD Display	1	$20 USD	$20 USD
Spool of 3D Printer Material	1	$20 USD	$20 USD
Custom Circuit Board	3	$120 USD	$120 USD
Total Costs	$220 USD		

Appendix A

This appendix details the code used for this project.

```
#include "STML432KC.h"
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>


void I2C_setup();
void I2C_clock_setup();
void LED_setup();
void pin_setup();
void I2C_start(int SenAdd, int Size, int Dir);
void I2C_stop();
void I2C_wait();
int I2C_SendData(int SlaveAdd, int Data[], int Size);
int I2C_ReceiveData(int SlaveAdd, int Data[], int Size);
int Read_Sensor(int SlaveAdd, int RegAdd[], int Size);
int I2C_SendData_Sensor(int SlaveAdd, int Data[], int Size);
void Configure_Sensors();
void Update_1_Sensor(int number);
void Update_Sensors();
void Timer_2_Setup();
void Timer_3_Setup();
void Timers_Init();
void Display_Data();
void Clear_LCD();
```

```
int I2C_SendChar(int SlaveAdd, char Char[], int Size);
void Average_Arrays();
void Do_Math();
void Equations();
void button_pin_setup();

int acc_x_1[5];int acc_y_1[5];int acc_z_1[5];
int acc_x_avg = 0; int acc_y_avg = 0; int acc_z_avg = 0;

int acc_total;
int acc_diff = 0;
int acc_check[2];
double real_time;
double real_acc;
double real_acc_max;
double distance;
double velocity; int vel_int; int vel_dub;
double force; int force_int; int force_dub;
double power; int pow_int; int pow_dub;
double peak_velocity; int vel_peak_int; int vel_peak_dub;
double acc_max = 0;
int flag = 0;
int time;
int mode = 0;
double mass;

char LCD[100];
```

```
int RXDT[16];
int TXDT[16];
int tmp[1];


int c = 0;
char char_array[5];
int cycle = 0;
int check = 0;
int bar_weight = 45;
int counter1 = 0;
int counter2 = 0;


int main()
{
LED_setup();
pin_setup();
button_pin_setup();
I2C_clock_setup();
I2C_setup();
Timers_Init();
Configure_Sensors();
Clear_LCD();
        while(1)
        {
        Update_Sensors();
        if(check > 500)
```

```
        {
        Display_Data();
        check = 0;
        GPIOB_BSRR |= LED_OFF;
        }
        check++;
        }
}

void Do_Math()
{
        acc_total = sqrt((acc_x_avg * acc_x_avg) + (acc_y_avg * acc_y_avg) + (acc_z_avg *
acc_z_avg));//Magnitude of acceleration
        if(acc_total > acc_max)
        {
        acc_max = acc_total;//determining the max acceleration
        }
        acc_check[cycle] = acc_total;//comparing the previous magnitude to the most recent one
to find
        acc_diff = (acc_check[0] - acc_check[1]);//the difference between the two
        acc_diff = abs(acc_diff);
        cycle++;
        if(cycle > 1)//oscilating between 1 and 0
        {
        cycle = 0;
        }
}

void Average_Arrays()
```

```
{
        for(int loop = 0; loop < 4; loop++)//takes the aveage of the components of acceleration
        {
                                        //over the course of 5 measurements
    acc_x_avg = acc_x_avg + acc_x_1[loop];
        }
        acc_x_avg = (acc_x_avg / 5);

        for(int loop = 0; loop < 4; loop++)
        {
    acc_y_avg = acc_y_avg + acc_y_1[loop];
        }
        acc_y_avg = (acc_y_avg / 5);

        for(int loop = 0; loop < 4; loop++)
        {
    acc_z_avg = acc_z_avg + acc_z_1[loop];
        }
        acc_z_avg = (acc_z_avg / 5);
}

void Clear_LCD()
{
int LCD[2] = {LCD_set,LCD_clear};//sends the SERLCD the command to
I2C_SendData(0x72,LCD, 2);      //clear its screen
}

void Display_Data()
{
```

```
Clear_LCD();
	if(mode == 0)//checks for the mode and displays the associated value
	{
	sprintf(LCD, "Bar Weight:     %d lbs", bar_weight);
	}
	else if(mode == 1)
	{
	sprintf(LCD, "AverageVelocity:%d.%dm/s", vel_int, vel_dub);
	}
	else if(mode == 2)
	{
	sprintf(LCD, "Average Force:  %d.%dN", force_int, force_dub);
	}
	else if(mode == 3)
	{
	sprintf(LCD, "Average Power:  %d.%dNm", pow_int, pow_dub);
	}
	else if(mode == 4)
	{
	sprintf(LCD, "Peak Velocity:  %d.%dm/s", vel_peak_int, vel_peak_dub);
	}
if(mode > 4)//cycles between the four measurements to be displayed
{
mode = 1;
}



int length = strlen(LCD);
```

```
I2C_SendChar(LCD_ADD, LCD, length);//actual function that send the characters to be displayed

}


int I2C_SendChar(int SlaveAdd, char Char[], int Size)//same as Send_Data but with char not int

{

int i;


if( Size <= 0 || Char == NULL)

{

	return -1;

}


I2C_wait();//wait until the lines are availible


I2C_start(SlaveAdd, Size, 0);//writing data to the slave


for(i = 0; i < Size; i++)

{

while( ( I2C1_ISR & I2C_ISR_TXE) == 0);//waits until TXDR register is empty (data has been sent)


//TXIS is cleared by writing to the TXDR register

I2C1_TXDR = Char[i] & I2C_TXDR_TXDATA;//only first 8bits


}


while( (I2C1_ISR & I2C_ISR_TC) == 0 && (I2C1_ISR & I2C_ISR_NACKF) == 0);
```

```
if( (I2C1_ISR & I2C_ISR_NACKF) != 0 ) //Error checking. Not sure what the deal is
{                                //with the return and why this function doesn't
return -1;                       //have a return.
}

I2C_stop();

return 0;

}

void Timers_Init()//intializes the two timers used
{
Timer_2_Setup();
Timer_3_Setup();
}

void Timer_3_Setup()
{
RCC_APB1ENR1 |= TIM6_CLK_EN;//enables TIM6

TIM6_CR1 &= ~TIM_CR1_CEN;//disables timer 6

TIM6_CR1 &= ~TIM_CR1_DIR;//sets upcount mode
TIM6_PSC = 999;
TIM6_ARR = 65535;//Overflow = ARR * (PRSC + 1) / Fsys => Time = CNT * (PRSC + 1)
/Fsys

}
```

```
void Timer_2_Setup()
{
NVIC_ISER0 |= TIM2_Interrupt_Enable;

RCC_APB1ENR1 |= TIM2_CLK_EN;//enables TIM2

TIM2_CR1 &= ~TIM_CR1_CEN;//disables timer 2

TIM2_CR1 &= ~TIM_CR1_DIR;//sets upcount mode
TIM2_PSC = 399;
TIM2_ARR = 10000;//Overflow = ARR * (PRSC + 1) / Fsys => 40,000 = 10,000 * (4)
TIM2_DIER |= TIM_DIER_UIE;
TIM2_DIER |= TIM_DIER_TIE;
TIM2_CR1 |= TIM_CR1_CEN;

}

void TIM2_IRQHandler()
{
Average_Arrays();
Do_Math();
TIM2_CNT = 0;//resets the count

if((acc_diff > 2040) && (flag == 0))//checks if threshold is reached, and flag isn't set
{
flag = 1;
TIM6_CR1 |= TIM_CR1_CEN;//starts timer
```

```
}

if((acc_diff < 2040) && (flag == 1))//checks for threshold and flag
{
time = TIM6_CNT;//loads timer value into time variable
TIM6_CR1 &= ~TIM_CR1_CEN;//resets timer and flag
TIM6_CNT = 0;
flag = 0;
Equations();//does math
}

GPIOB_BSRR |= LED_ON;

if( ((GPIOB_IDR & (1 << 5)) == (1 << 5)))//PB5 is Mode button
{//checks to see if button is pressed for 2ms, debounces the button
counter1++;
	if(counter1 >= 2)
	{
	mode++;//increments mode
	counter1 = 0;
	}
		if(mode > 4)
		{
		mode = 1;
		}
}

if((GPIOB_IDR & (1 << 1)) == (1 << 1))//PB1 is the Increment button
```

```
{//checks to see if button is pressed for 2ms, debounces the button
counter2++;
        if(counter2 >=1)
        {
        bar_weight = bar_weight + 10;
        counter2 = 0;
        }
}

        if((GPIOA_IDR & (1 << 8)) == (1 << 8))//PA8 is the reset button
        {//when pressed resets all variable and returns display to "bar weight"
        mode = 0;
        bar_weight = 45;
        velocity = 0;
        force = 0;
        power = 0;
        peak_velocity = 0;
        }

TIM2_SR &= ~(1 << 0);
}

void Equations()
{//basic kinematic equations used to find final values
real_time = ((time) / 100000.0);
real_acc = ((acc_total / 2040.0) * 9.81);
real_acc_max = ((acc_max    / 2040.0) * 9.81);
distance = 0.5 * (real_acc * (real_time * real_time));
```

```
mass = (bar_weight / 2.20462);

velocity  = (real_time * real_acc);
vel_int = floor(velocity);
vel_dub = ((velocity - vel_int) * 100);

peak_velocity = (real_time * real_acc_max)/10;
vel_peak_int = floor(peak_velocity);
vel_peak_dub = ((peak_velocity - vel_peak_int) * 100);

force = (mass * real_acc);
force_int = floor(force);
force_dub = ((force - force_int) * 100);

power = (force * distance);
pow_int = floor(power);
pow_dub = ((power - pow_int) * 100);

}

void button_pin_setup()
{
RCC_AHB2ENR |= 1 << 0;//GPIOA clock enable
RCC_AHB2ENR |= 1 << 1;//GPIOB clock enable
RCC_AHB2ENR |= 1 << 2;//GPIOC clock enable

GPIOA_MODER &= ~(3 << (2*8));//clears PA8 function
GPIOA_MODER |= (0 << (2*8));//clears PA8 function
```

```
GPIOB_MODER &= ~(3 << (2*1));//clears PB1 function
GPIOB_MODER |= (0 << (2*1));//clears PB1 function
GPIOB_MODER &= ~(3 << (2*5));//clears PB5 function
GPIOB_MODER |= (0 << (2*5));//clears PB5 function

}

void Update_Sensors()
{
Update_1_Sensor(1);
//Update_1_Sensor(2);//unused second sensor
}

void Update_1_Sensor(int number)
{
	int RegAdd[1];
	int ADD = 0;

	if(number == 1)
	{
	ADD = 0x68;
	}
	else
	{
	ADD = 0x69;
	}
```

```
RegAdd[0] = 0x3B;//acc_x_H internal register
Read_Sensor(ADD, RegAdd, 1);
int acc_x_h;
acc_x_h = RXDT[0];

RegAdd[0] = 0x3C;//acc_x_L internal register
Read_Sensor(ADD, RegAdd, 1);
tmp[0] = RXDT[0];
acc_x_1[c] = (acc_x_h << 8) + tmp[0];//allows for MSBs and LSBs to combine for 16-bit
value

RegAdd[0] = 0x3D;//acc_y_H internal register
Read_Sensor(ADD, RegAdd, 1);
int acc_y_h;
acc_y_h = RXDT[0];

RegAdd[0] = 0x3E;//acc_y_L internal register
Read_Sensor(ADD, RegAdd, 1);
tmp[0] = RXDT[0];
acc_y_1[c] = (acc_y_h << 8) + tmp[0];

RegAdd[0] = 0x3F;//acc_z_H internal register
Read_Sensor(ADD, RegAdd, 1);
int acc_z_h;
acc_z_h = RXDT[0];

RegAdd[0] = 0x40;//acc_z_L internal register
Read_Sensor(ADD, RegAdd, 1);
tmp[0] = RXDT[0];
```

```
        acc_z_1[c] = (acc_z_h << 8) + tmp[0];

        if(c > 98)//allows storage of up to 100 values
        {
                c = -1;
        }
        c++;

}

void Configure_Sensors()
{
int RegInfo[2];
int count = 0;
int ADD = 0;

//while(count <= 1)
{
if(count == 0)
{
ADD = 0x68;
}
else
{
ADD = 0x69;
}

RegInfo[0] = 0x6B;//power management register
```

```
RegInfo[1] = 0x00;//disables sleep mode for sensor
I2C_SendData(ADD, RegInfo, 2);

RegInfo[0] = 0x1B;//gyro setup register
RegInfo[1] = 0x00;//check settings to change sensistivity (this base should work though)
I2C_SendData(ADD, RegInfo, 2);

RegInfo[0] = 0x1C;//accel setup register
RegInfo[1] = (0x03 << 3);//+- 16g's
I2C_SendData(ADD, RegInfo, 2);

//count ++;
}

}

int Read_Sensor(int SlaveAdd, int RegAdd[], int Size)
{
//Write Protocol
//send start sequence
//send I2C address of the slave w/ R/W bit low
//send internal register adress to write to
//send the data byte
//send the stop sequence

//Read Protocol
//send start sequence
//send I2C address of the slave w/ R/W bit low
```

```
//send internal register address
//send start sequence (repeated start)
//send the I2C address of the slave w/ R/W bit high
//send data byte
//send stop sequence
I2C_SendData(SlaveAdd, RegAdd, Size);

I2C_ReceiveData(SlaveAdd, RegAdd, Size);
}



int I2C_ReceiveData(int SlaveAdd, int RegAdd[], int Size)
{
int i;

if( Size <= 0 || RegAdd == NULL)
{
return -1;
}

I2C_wait();//waits for bus to be clear

I2C_start(SlaveAdd, Size, 1);//reading from the slave

for(i = 0; i < Size; i ++)
{
	while( (I2C1_ISR & I2C_ISR_RXNE) == 0);//wait until RXNE flag is set
	RXDT[i] = I2C1_RXDR & I2C_RXDR_RXDATA;//only first 8bits
```

```
while((I2C1_ISR & I2C_ISR_TC) == 0);//wait until transfer is complete

I2C_stop();

return 0;

}
}

int I2C_SendData(int SlaveAdd, int Data[], int Size)
{
int i;

if( Size <= 0 || Data == NULL)
{
	return -1;
}

I2C_wait();//wait until the lines are availible

I2C_start(SlaveAdd, Size, 0);//writing data to the slave

for(i = 0; i < Size; i++)
{
while( ( I2C1_ISR & I2C_ISR_TXE) == 0);//waits until TXDR register is empty (data has been sent)

//TXIS is cleared by writing to the TXDR register
```

```
I2C1_TXDR = Data[i] & I2C_TXDR_TXDATA;//only first 8bits

}

while( (I2C1_ISR & I2C_ISR_TC) == 0 && (I2C1_ISR & I2C_ISR_NACKF) == 0);

if( (I2C1_ISR & I2C_ISR_NACKF) != 0 ) //Error checking. Not sure what the deal is
{                                   //with the return and why this function doesn't
return -1;                          //have a return.
}

I2C_stop();

return 0;

}

void I2C_wait()
{
while( (I2C1_ISR & I2C_ISR_BUSY) == I2C_ISR_BUSY);//waits until busy flag is cleared
}

void I2C_stop()
{
I2C1_CR2 |= I2C_CR2_STOP;//generate STOP bit after current byte transfer

while( (I2C1_ISR & I2C_ISR_STOPF) == 0);//wait until STOPF flag is reset
```

```
I2C1_ISR |= I2C_ISR_STOPF;//clears the STOPF flag
}

void I2C_start(int SenAdd, int Size, int Dir)
{
//Dir = 0; Master requesting write transfer
//Dir = 1; Master requesting read transfer
int tmpreg = I2C1_CR2;

tmpreg &= ~(I2C_CR2_SADD | I2C_CR2_NBYTES | I2C_CR2_RELOAD |
I2C_CR2_AUTOEND |
                                    I2C_CR2_RD_WRN | I2C_CR2_START
|I2C_CR2_STOP);

if(Dir == 1)
{
	tmpreg |= I2C_CR2_RD_WRN; //read from slave
}
else
{
	tmpreg &= ~I2C_CR2_RD_WRN; //write to slave
}

tmpreg |= ((SenAdd << 1) & (I2C_CR2_SADD)) | ((Size << 16) & (I2C_CR2_NBYTES));

tmpreg |= I2C_CR2_START;
I2C1_CR2 = tmpreg;
```

```
}

void pin_setup()
{
RCC_AHB2ENR |= 1 << 0;//GPIOA clock enable
GPIOA_OTYPER |= 1 << 9;//sets PA9 to open drain
GPIOA_OTYPER |= 1 << 10;//sets PA10 to open drain
GPIOA_MODER &= ~(3 << (2*9));//clears PA9 function
GPIOA_MODER &= ~(3 << (2*10));//clears PA10 function
GPIOA_MODER |= (2 << (2*9));//sets PA9 to alt. funct.
GPIOA_MODER |= (2 << (2*10));//sets PA10 to al. funct.
GPIOA_PUPDR &= ~(3 << (2*9));//clears PA9 PUPDR
GPIOA_PUPDR &= ~(3 << (2*10));//clears PA10 PUPDR
//AF4 for PA9 PA10
GPIOA_AFRH |= (4 << (4 * 1));//sets PA9 to alt. funct. 4 (SCL)
GPIOA_AFRH |= (4 << (4 * 2));//sets PA10 to alt. funct. 4 (SDA)

}

void LED_setup()
{
RCC_AHB2ENR |= 1 << 1;
GPIOB_MODER &= ~(3 << (2*3));
GPIOB_MODER |= (1 << (2*3));
}

void I2C_clock_setup()
{
```

```
RCC_APB1ENR1 |= (1 << 21);//turns on I2C1 clock

RCC_CCIPR &= ~(4 << (2*6));//clears bits (may not be necessary, reset is 0x0000 0000)
RCC_CCIPR |= (1 << (2*6));//selects SYSCLK for I2C1 (4MHz)

RCC_APB1RSTR1 |= 1 << 21;//resets I2C1
RCC_APB1RSTR1 &= ~(1 <<21);//Completes reset (not sure why this but book said to)
}

void I2C_setup()
{
//I2C1 Control Register 1 Setup
I2C1_CR1 &= ~(1 << 0);//disables I2C1
I2C1_CR1 &= ~(1 << 12);//0:Turns on analog noise filter
I2C1_CR1 &= ~(15 << 8);//disables digital filter
I2C1_CR1 |= (1 << 7);//errors interupts enabled
I2C1_CR1 &= ~(1 << 16);//disables SMBus and sets I2C mode (I think)
I2C1_CR1 &= ~(1 << 17);//enable clock stretching

//I2C1 TIMINGR Setup
I2C1_TIMINGR = 0;
//SYSCLK = 4MHz, PRESC = 3, 4MHz/(1+3) = 1 MHz (easy math from this base)
I2C1_TIMINGR &= ~(15U << 28);//clears prescaler
I2C1_TIMINGR |= 3U << 28;//sets the prescaler to 3
I2C1_TIMINGR |= 4U;//sets SCL Low period. (master mode > 4.7 us) ((1+4)*tI2C_CLK(1us) = 5us)
I2C1_TIMINGR |= 4U << 8;//sets the SCL High period. (master mode > 4.0 us)((1+4)*tI2C_CLK(1us) = 5us)
I2C1_TIMINGR |= 1U << 20;//data setup time > 1.0 us((1+1)*tI2C_CLK(1us) = 2us)
```

```
I2C1_TIMINGR |= 1U << 16;//data hold time > 1.25 us((1+1)*tI2C_CLK(1us) = 2us)

//I2C1 Own address setup (Only in Slave Mode)
//I2C1_OAR1 &= ~(0 << 15);//disables own adress 1 to allow for writing
//I2C1_OAR1 |= (0x0000 << 1);//sets the slave adress
//I2C1_OAR1 |= 1 << 15;//enables own adress 1

I2C1_OAR2 &= ~(1 << 15);//disable own adress 2

//I2C1 Control Register 2 Setup
I2C1_CR2 &= ~(1 << 11);//7 bit adressing mode
I2C1_CR2 |= 1 << 25;//AutoEnd enabled

I2C1_CR1 |= 1 << 0;//Enables 12C1
//GPIOB_BSRR |= LED_ON;

}
```

Appendix B

This appendix details the header file used for this project as well as the masks used throughout the code detail in **Appendix A**.

```
//David Stoddard

//STML432KC Header File for Weight Lifting Performance Monitor Senior Project

//Updated April 15, 2019


//https://www.st.com/content/ccc/resource/technical/document/reference_manual/group0/b0/ac/3e/8f/6d/21/47/af/DM00151940/files/DM00151940.pdf/jcr:content/translations/en.DM00151940.pdf


#define TIM2_Interrupt_Enable (1 << 28)

#define TIM_DIER_TIE (1 << 6);

#define TIM_DIER_UIE (1 << 0);

#define LED_ON (1 << 3)

#define LED_OFF (1 << (3 + 16))

#define I2C_CR2_SADD (0x7F << 1)

#define I2C_CR2_NBYTES (0x7F << 16)

#define I2C_CR2_RELOAD (1 << 24)

#define I2C_CR2_AUTOEND (1 << 25)

#define I2C_CR2_RD_WRN (1 << 10)//read transfer

#define I2C_CR2_START (1 << 13)

#define I2C_CR2_STOP (1 << 14)

#define I2C_ISR_STOPF (1 << 5)

#define I2C_ISR_BUSY (1 << 15)

#define I2C_ISR_TXIS (1 << 1)

#define I2C_ISR_TXE (1 << 0)

#define I2C_ISR_TC (1 << 6)

#define I2C_ISR_NACKF (1 << 4)
```

```c
#define NULL 0
#define I2C_ISR_RXNE (1 << 2)
#define I2C_TXDR_TXDATA 0xFF
#define I2C_RXDR_RXDATA 0xFF
#define AHB2ENR_SPI1_ENR (1 << 12)
#define APB2RSTR_SPI1_RESET (1 << 12)
#define SPI_CR1_SPE (1 << 6)
#define SPI_CR1_RXONLY (1 << 10)
#define SPI_CR1_BIDIMODE (1 << 15)
#define SPI_CR1_BIDIOE (1 << 14)
#define SPI_CR1_LSBFIRST (1 << 7)
#define SPI_CR1_CPHA (1 << 0)
#define SPI_CR1_CPOL (1 << 1)
#define SPI_CR1_CREN (1 << 13)
#define SPI_CR1_SSM (1 << 9)
#define SPI_CR1_MSTR (1 << 2)
#define SPI_CR1_SSI (1 << 8)
#define SPI_CR2_DS (0x7 << 8)//8 bit mode 0111
#define SPI_CR2_FRF (1 << 4)
#define SPI_CR2_NSSP (1 << 3)
#define SPI_CR2_FRXTH (1 << 12)
#define SPI_SR_TXE (1 << 1)
#define SPI_SR_RXNE (1 << 0)
#define SPI_SR_BSY (1 << 7)
#define TIM2_CLK_EN (1 << 0)
#define TIM3_CLK_EN (1 << 1)
#define TIM6_CLK_EN (1 << 4)
#define TIM_CR1_CEN (1 << 0)
```

```
#define TIM_CR1_DIR (1 << 4)


#define LCD_set 0x7C
#define LCD_clear 0x2D
#define LCD_ADD 0x72
#define Sensor_1 0x68
#define Sensor_2 0x69


//TIM2 Adresses 0x 4000 0000
#define TIM2_CR1        (*((volatile unsigned long *) 0x40000000))//Contorl Register 1
#define TIM2_CR2        (*((volatile unsigned long *) 0x40000004))//Control Register 2
#define TIM2_SMCR       (*((volatile unsigned long *) 0x40000008))//Slave Mode Control
Register
#define TIM2_DIER       (*((volatile unsigned long *) 0x4000000C))//DMA/Interupt Enable
Register
#define TIM2_SR         (*((volatile unsigned long *) 0x40000010))//Status Register
#define TIM2_EGR        (*((volatile unsigned long *) 0x40000014))//Event Generation
Register
#define TIM2_CCMR1      (*((volatile unsigned long *) 0x40000018))//Capture/Compare
Mode Register 1
#define TIM2_CCMR2      (*((volatile unsigned long *) 0x4000001C))//Capture/Compare
Mode Register 2
#define TIM2_CCER       (*((volatile unsigned long *) 0x40000020))//Capture/Compare
Enable Register
#define TIM2_CNT        (*((volatile unsigned long *) 0x40000024))//Counter
#define TIM2_PSC        (*((volatile unsigned long *) 0x40000028))//Prescaler
#define TIM2_ARR        (*((volatile unsigned long *) 0x4000002C))//Auto-Reload Register
#define TIM2_CCR1       (*((volatile unsigned long *) 0x40000034))//Capture/Compare
Register 1
#define TIM2_CCR2       (*((volatile unsigned long *) 0x40000038))//Capture/Compare
Register 2
```

#define TIM2_CCR3 (*((volatile unsigned long *) 0x4000003C))//Capture/Compare Register 3

#define TIM2_CCR4 (*((volatile unsigned long *) 0x40000040))//Capture/Compare Register 4

#define TIM2_DCR (*((volatile unsigned long *) 0x40000048))//DMA Control Register

#define TIM2_DMAR (*((volatile unsigned long *) 0x4000004C))//DMA Address For Full Transfer

#define TIM2_OR1 (*((volatile unsigned long *) 0x40000050))//Option Register 1

#define TIM2_OR2 (*((volatile unsigned long *) 0x40000060))//Option Register 2

//TIM3 Adresses 0x 4000 0400 0x40000400

#define TIM3_CR1 (*((volatile unsigned long *) 0x40000400))//Contorl Register 1

#define TIM3_CR2 (*((volatile unsigned long *) 0x40000404))//Control Register 2

#define TIM3_SMCR (*((volatile unsigned long *) 0x40000408))//Slave Mode Control Register

#define TIM3_DIER (*((volatile unsigned long *) 0x4000040C))//DMA/Interupt Enable Register

#define TIM3_SR (*((volatile unsigned long *) 0x40000410))//Status Register

#define TIM3_EGR (*((volatile unsigned long *) 0x40000414))//Event Generation Register

#define TIM3_CCMR1 (*((volatile unsigned long *) 0x40000418))//Capture/Compare Mode Register 1

#define TIM3_CCMR2 (*((volatile unsigned long *) 0x4000041C))//Capture/Compare Mode Register 2

#define TIM3_CCER (*((volatile unsigned long *) 0x40000420))//Capture/Compare Enable Register

#define TIM3_CNT (*((volatile unsigned long *) 0x40000424))//Counter

#define TIM3_PSC (*((volatile unsigned long *) 0x40000428))//Prescaler

#define TIM3_ARR (*((volatile unsigned long *) 0x4000042C))//Auto-Reload Register

#define TIM3_CCR1 (*((volatile unsigned long *) 0x40000434))//Capture/Compare Register 1

#define TIM3_CCR2 (*((volatile unsigned long *) 0x40000438))//Capture/Compare Register 2

#define TIM3_CCR3 (*((volatile unsigned long *) 0x4000043C))//Capture/Compare Register 3

#define TIM3_CCR4 (*((volatile unsigned long *) 0x40000440))//Capture/Compare Register 4

#define TIM3_DCR (*((volatile unsigned long *) 0x40000448))//DMA Control Register

#define TIM3_DMAR (*((volatile unsigned long *) 0x4000044C))//DMA Address For Full Transfer

#define TIM3_OR1 (*((volatile unsigned long *) 0x40000450))//Option Register 1

#define TIM3_OR2 (*((volatile unsigned long *) 0x40000460))//Option Register 2

//TIM6 Adresses 0x 4000 1000

#define TIM6_CR1 (*((volatile unsigned long *) 0x40001000))//Contorl Register 1

#define TIM6_CR2 (*((volatile unsigned long *) 0x40001004))//Control Register 2

#define TIM6_SMCR (*((volatile unsigned long *) 0x40001008))//Slave Mode Control Register

#define TIM6_DIER (*((volatile unsigned long *) 0x4000100C))//DMA/Interupt Enable Register

#define TIM6_SR (*((volatile unsigned long *) 0x40001010))//Status Register

#define TIM6_EGR (*((volatile unsigned long *) 0x40001014))//Event Generation Register

#define TIM6_CCMR1 (*((volatile unsigned long *) 0x40001018))//Capture/Compare Mode Register 1

#define TIM6_CCMR2 (*((volatile unsigned long *) 0x4000101C))//Capture/Compare Mode Register 2

#define TIM6_CCER (*((volatile unsigned long *) 0x40001020))//Capture/Compare Enable Register

#define TIM6_CNT (*((volatile unsigned long *) 0x40001024))//Counter

#define TIM6_PSC (*((volatile unsigned long *) 0x40001028))//Prescaler

#define TIM6_ARR (*((volatile unsigned long *) 0x4000102C))//Auto-Reload Register

```
#define TIM6_CCR1        (*((volatile unsigned long *) 0x40001034))//Capture/Compare Register 1

#define TIM6_CCR2        (*((volatile unsigned long *) 0x40001038))//Capture/Compare Register 2

#define TIM6_CCR3        (*((volatile unsigned long *) 0x4000103C))//Capture/Compare Register 3

#define TIM6_CCR4        (*((volatile unsigned long *) 0x40001040))//Capture/Compare Register 4

#define TIM6_DCR         (*((volatile unsigned long *) 0x40001048))//DMA Control Register

#define TIM6_DMAR        (*((volatile unsigned long *) 0x4000104C))//DMA Address For Full Transfer

#define TIM6_OR1         (*((volatile unsigned long *) 0x40001050))//Option Register 1

#define TIM6_OR2         (*((volatile unsigned long *) 0x40001060))//Option Register 2


//NVIC

#define NVIC_ISER0       (*((volatile unsigned long *) 0xE000E100))//Interrupt Set-Enable Register


//Clock Adresses 0x4002 1000

#define RCC_CR           (*((volatile unsigned long *) 0x40021000))//Clock Control Register

#define RCC_CFGR         (*((volatile unsigned long *) 0x40021008))//Clock Configuration Registetr

#define RCC_AHB2ENR      (*((volatile unsigned long *) 0x4002104C))//GPIO Ports Enable

#define RCC_APB1RSTR1    (*((volatile unsigned long *) 0x40021038))//APB1 Peripheral Reset Register

#define RCC_APB1ENR1     (*((volatile unsigned long *) 0x40021058))//I2C Clock enable

#define RCC_APB2RSTR     (*((volatile unsigned long *) 0x40021040))//APB2 Peripehral Reset Register

#define RCC_APB2ENR      (*((volatile unsigned long *) 0x40021060))//SPI Clock enable

#define RCC_CCIPR        (*((volatile unsigned long *) 0x40021088))//Peripherals Independent Clock Configuration Register
```

//SP1 0x4001 3000

#define SPI1_CR1 (*((volatile unsigned long *) 0x40013000))//SPI Control Register 1

#define SPI1_CR2 (*((volatile unsigned long *) 0x40013004))//SPI Control Register 2

#define SPI1_SR (*((volatile unsigned long *) 0x40013008))//SPI Status Register

#define SPI1_DR (*((volatile unsigned long *) 0x4001300C))//SPI Data Register

#define SPI1_CRCPR (*((volatile unsigned long *) 0x40013010))//SPI CRC Polynomial Register

#define SPI1_RXCRCR (*((volatile unsigned long *) 0x40013014))//SPI Rx CRC Register

#define SPI1_TXCRCR (*((volatile unsigned long *) 0x40013018))//SPI Tx CRC Register

//GPIOA 0x4800 0000 (pg 261)

#define GPIOA_MODER (*((volatile unsigned long *) 0x48000000))//Port Mode Register

#define GPIOA_OTYPER (*((volatile unsigned long *) 0x48000004))//Port Output Type Register

#define GPIOA_OSPEEDR (*((volatile unsigned long *) 0x48000008))//Port Output Speed Register

#define GPIOA_PUPDR (*((volatile unsigned long *) 0x4800000C))//Port Pull-Up/Pull-Down

#define GPIOA_IDR (*((volatile unsigned long *) 0x48000010))//Port Input Data Register

#define GPIOA_ODR (*((volatile unsigned long *) 0x48000014))//Port Output Data Register

#define GPIOA_BSRR (*((volatile unsigned long *) 0x48000018))//Port Bit Set/Reset Register

#define GPIOA_LCKR (*((volatile unsigned long *) 0x4800001C))//Port Config Lock Register

#define GPIOA_AFRL (*((volatile unsigned long *) 0x48000020))//Alt. Function Low Register

#define GPIOA_AFRH (*((volatile unsigned long *) 0x48000024))//Alt. Function High Register

#define GPIOA_BRR (*((volatile unsigned long *) 0x48000028))//Port Bit Reset Register

```
//GPIOB 0x4800 0400

#define GPIOB_MODER      (*((volatile unsigned long *) 0x48000400))//Port Mode Register

#define GPIOB_OTYPER     (*((volatile unsigned long *) 0x48000404))//Port Output Type Register

#define GPIOB_OSPEEDR    (*((volatile unsigned long *) 0x48000408))//Port Output Speed Register

#define GPIOB_PUPDR      (*((volatile unsigned long *) 0x4800040C))//Port Pull-Up/Pull-Down

#define GPIOB_IDR        (*((volatile unsigned long *) 0x48000410))//Port Input Data Register

#define GPIOB_ODR        (*((volatile unsigned long *) 0x48000414))//Port Output Data Register

#define GPIOB_BSRR       (*((volatile unsigned long *) 0x48000418))//Port Bit Set/Reset Register

#define GPIOB_LCKR       (*((volatile unsigned long *) 0x4800041C))//Port Config Lock Register

#define GPIOB_AFRL       (*((volatile unsigned long *) 0x48000420))//Alt. Function Low Register

#define GPIOB_AFRH       (*((volatile unsigned long *) 0x48000424))//Alt. Function High Register

#define GPIOB_BRR        (*((volatile unsigned long *) 0x48000428))//Port Bit Reset Register


//GPIOC 0x4800 0800                        0x48000800

#define GPIOC_MODER      (*((volatile unsigned long *) 0x48000800))//Port Mode Register

#define GPIOC_OTYPER     (*((volatile unsigned long *) 0x48000804))//Port Output Type Register

#define GPIOC_OSPEEDR    (*((volatile unsigned long *) 0x48000808))//Port Output Speed Register

#define GPIOC_PUPDR      (*((volatile unsigned long *) 0x4800080C))//Port Pull-Up/Pull-Down

#define GPIOC_IDR        (*((volatile unsigned long *) 0x48000810))//Port Input Data Register
```

```
#define GPIOC_ODR       (*((volatile unsigned long *) 0x48000814))//Port Output Data Register
#define GPIOC_BSRR      (*((volatile unsigned long *) 0x48000818))//Port Bit Set/Reset Register
#define GPIOC_LCKR      (*((volatile unsigned long *) 0x4800081C))//Port Config Lock Register
#define GPIOC_AFRL      (*((volatile unsigned long *) 0x48000820))//Alt. Function Low Register
#define GPIOC_AFRH      (*((volatile unsigned long *) 0x48000824))//Alt. Function High Register
#define GPIOC_BRR       (*((volatile unsigned long *) 0x48000828))//Port Bit Reset Register

//I2C2  0x4000 5800 (pg 1177)
#define I2C2_CR1        (*((volatile unsigned long *) 0x40005800))//I2C Control Register
#define I2C2_CR2        (*((volatile unsigned long *) 0x400058C4))//I2C Control Register 2
#define I2C2_OAR1       (*((volatile unsigned long *) 0x40005808))//Own Adress 1 Register
#define I2C2_OAR2       (*((volatile unsigned long *) 0x4000580C))//Own Adress 2 Register
#define I2C2_TIMINGR    (*((volatile unsigned long *) 0x40005810))//Timing Register
#define I2C2_TIMEOUTR   (*((volatile unsigned long *) 0x40005814))//Timeout Register
#define I2C2_ISR        (*((volatile unsigned long *) 0x40005818))//Interrupt and Status Register
#define I2C2_ICR        (*((volatile unsigned long *) 0x4000581C))//Interrupt Clear Register
#define I2C2_PECR       (*((volatile unsigned long *) 0x40005820))//PEC Register
#define I2C2_RXDR       (*((volatile unsigned long *) 0x40005824))//Receive Data Register
#define I2C2_TXDR       (*((volatile unsigned long *) 0x40005828))//Transmit Data Register

//I2C1  0x4000 5400
#define I2C1_CR1        (*((volatile unsigned long *) 0x40005400))//I2C Control Register
#define I2C1_CR2        (*((volatile unsigned long *) 0x40005404))//I2C Control Register 2
#define I2C1_OAR1       (*((volatile unsigned long *) 0x40005408))//Own Adress 1 Register
#define I2C1_OAR2       (*((volatile unsigned long *) 0x4000540C))//Own Adress 2 Register
```

```c
#define I2C1_TIMINGR     (*((volatile unsigned long *) 0x40005410))//Timing Register
#define I2C1_TIMEOUTR    (*((volatile unsigned long *) 0x40005414))//Timeout Register
#define I2C1_ISR         (*((volatile unsigned long *) 0x40005418))//Interrupt and Status Register
#define I2C1_ICR         (*((volatile unsigned long *) 0x4000541C))//Interrupt Clear Register
#define I2C1_PECR        (*((volatile unsigned long *) 0x40005420))//PEC Register
#define I2C1_RXDR        (*((volatile unsigned long *) 0x40005424))//Receive Data Register
#define I2C1_TXDR        (*((volatile unsigned long *) 0x40005428))//Transmit Data Register
```