Final Project Report: Music Light Display

ENGR 478 Design with Microcontrollers
Electrical Engineering Department
San Francisco State University
Fall 2022

Christie Lai - clai2@mail.sfsu.edu - 917158767
Jennifer Elias - jelias1@mail.sfsu.edu - 916265212
Elizabeth Kirwan - ekirwan@mail.sfsu.edu - 918611608

Date report submitted: May 22, 2022

Table of Contents Title page……1
Table of Contents……...2
Introduction……....3
Hardware……3
Design and Implementation……3
Experiments……5
Results and Discussion……..6
TeamWork……7
Conclusion and Future Work……..7
References……8
Source Code……9

Introduction

For our final project in the course, Design with Microcontrollers, we built a device that analyzes the peak amplitudes of signals within the coded ranges of our analog microphone. This device ultimately uses those coded ranges to activate specific LED colors. This project utilizes the Tiva C Launchpad microcontroller, and the LED colors would come from the board itself using Port Function F. Using the Analog to Digital Converter, Direct Memory Access, Timers, and the Fast Fourier Transform algorithm, we were able to create a functioning device that could a trigger red, blue, green, purple, yellow, cyan, or white light to turn on or off. The motivation of our project came from learning about the ADC in lab and gaining inspiration from the temperature sensing portion of the code. We also gained inspiration from the LED display projects from lab, and we incorporated those inspirations to have a goal to create a device that would have the amplitude of sound trigger different LED outputs.

Hardware

- 1x TI Launchpad TM4C123G
- 1x SparkFun Analog MEMS Microphone Breakout ICS-40180
- 3x Alligator Clips

Design and Implementation

Our project was relatively simple in terms of design, since we only used a single microphone, the TI Launchpad, and 3 alligator clips.

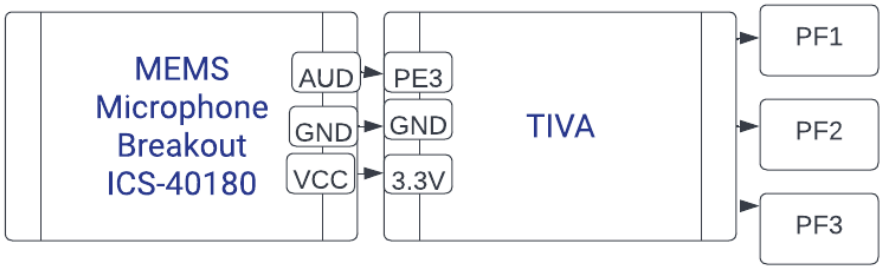

Figure 1: Music Light Display Project Design

Shows an example of how the microphone and LEDS connect to the TI Launchpad (TIVA)

The Microphone

The reason we chose the SparkFun Analog MEMS Microphone Breakout ICS-40180 is because we needed to make sure it was not simply a sound detector. We wanted an actual microphone, and most breakout boards have preamps built into them. Our last condition was that it needed to run with 3.3V, and not 5V since the latter would be the choice for an arduino. The microphone we chose had an audio, ground, and vcc port that connected to our TI Launchpad, using the PE3, ground, and 3.3V ports.

Figure 2: SparkFun Analog MEMS Microphone Breakout ICS-40180

The LEDS

The TI Launchpad's port PF1 correlated to the red LED, PF2 correlated to a blue LED, and PF3 correlated to the green LED. We used these to also create purple, yellow, cyan, and white. The way we did this was by turning on two LED colors at the same time.

The Sampling Rate

To implement a system that would analyze frequency, we picked our sample rate. We used a 40kHz sampling rate, which considers the 20kHz frequency range of human hearing. We needed to invert 400 samples on the tiva board, using the math of 40,000Hz/100s and using the DMA in ping pong mode. However, instead of storing 400 samples, we stored 512 samples so that we could run with a number that uses powers of 2, since digital Fast Fourier Transform works better in powers of 2's.

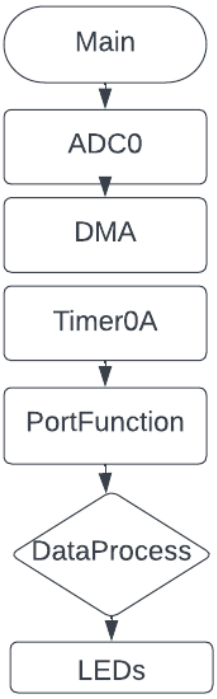

Figure 3: Main Flowchart

Experiments

To evaluate the system performance we started by testing the ADC and timer interrupt. The way we tested them was by using what we did in our lab assignment, ADC lab, where we tested the temperature sensor. When checking to see if the sensor was functional, we opened the watch window and included the formula for testing the temperature value. While verifying if the numbers we were getting were correct was by matching what we had in our lab to the result we were getting in our design code. The period we entered for the timer was 40,000 samples per second. We then used a variable to determine how many times a second we were filling in our 512 buffers. In our watch window, we saw that the variable was increasing by 100 times a second, so the period used in the timer was correct since 400 multiplied by 100 equals to 40khz. We then went ahead and used a code example that was given to us on iLearn, called the DMAandUARTbuffer, to add the DMA in the code to help us with memory access when using the ADC. The way we would implement it was by having the ADC value be stored into the DMA, in which then the DMA would reference the memory which then reads the digital signals. We then opened the watch window again and tested the values for the temperature sensor again to see if the ADC was working well with the DMA. Whenever the mic was picking up sound the values on the watch window would start to change. After verifying the ADC, DMA, and timer was picking up the mic sound, we then implemented the design code for the Fast Fourier Transform (FFT), for the digital signal processing. We used the arm function for FFT to help us with running the FFT in our design. In our FFT design, we tested it to find the peak value of the amplitude. To verify that the FFT was working correctly we opened the watch window again and tested the maxTest to find peak values from the sound we were inputting through the mic. After we concluded that it was working correctly, we then went ahead and tested the condition for the maxTest and added the LED colors to each condition. When we tested this, we opened the watch window just to see if the maxTest was matching up with the conditional statements we added for the LEDs and when the lights were emitted correctly, we then verified that it was working. We adjusted our conditional statements through trial and error depending on the LEDs' responses.

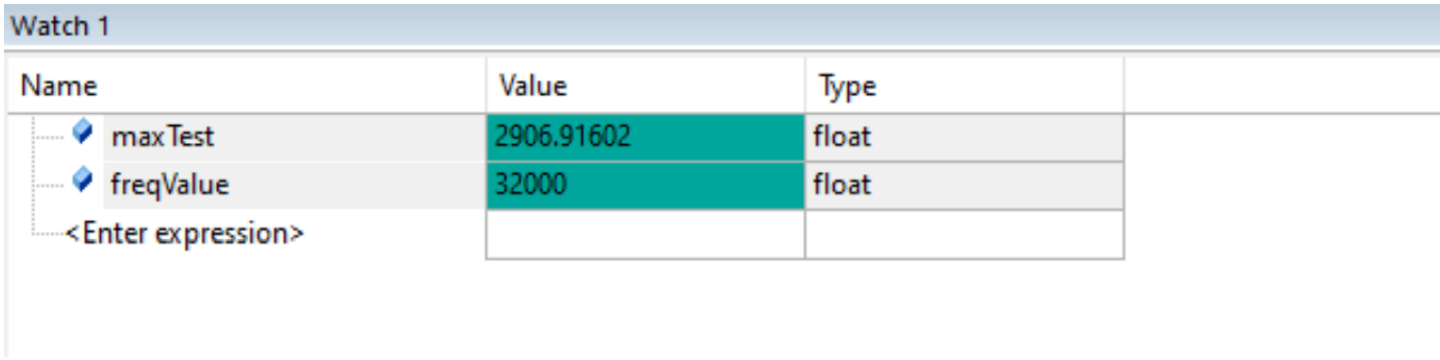

Figure 4: Sound and LED Are Both Off

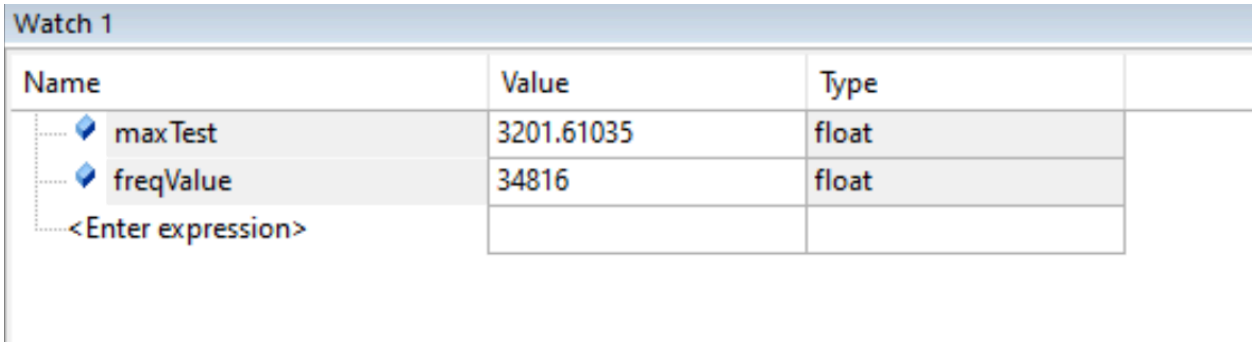

Figure 5: Sound and Yellow LED Are Both On

Results and Discussion

The original idea for the project was to compute the Fast Fourier Transform (FFT) of a given sound and find the frequency. Each music note on a scale corresponds to a given frequency, or the amount of wave cycles in a set amount of time. Performing an fft on our waveform would transform the data from the time domain to the frequency domain. Once we found the frequency value of the sound, we could use it for tuning an instrument like a piano or just for finding the notes of a favorite song. However, the project was a lot more complicated than we thought it was going to be. We first had to research how to use DMA in Ping Pong mode with our ADC. We looked at many different examples for the project, but many of them were not for the tiva board. Finally, we saw that iLearn had a DMA and UART example file which was the most helpful for us to use. Getting the right period was also tricky, but once we were able to determine that we were filling our buffers at the right rate, we could move on to actually performing our fft calculations on our data.

We did a lot of research on which library to use. There were many options to choose from, but we found that since the CMSIS DSP library from ARM was already installed for our tiva board, it would probably be the best to use. However, actually getting the CMSIS DSP library functions to work also took a good portion of our time. We got many error messages saying that the functions we were trying to use were undefined. At first we thought that we could use “#include arm_cfft_f32.c” to include our fft functions. When that didn’t work, we tried adding more paths in the “Options for Target 1.” Finally, we figured out that we could just “add existing items” to our project’s source group. So, we found the location of the ARM DSP library, found the Transform Functions folder, and added each function we needed. Since these functions also relied on other functions in the library, we also added “arm_common_tables.c,” “arm_const_structs.c,” “arm_cfft_radix8_f32.c,” and “arm_bitreversal.c.” Unfortunately, this wasn’t enough to solve all our errors because we didn’t have the correct “bit reversal” function. Since arm_cfft_32.c is a newer function, it needed to use “arm_bitreversal2.c.” However, our arm_bitreversal2 was not a c file, it was a source code(.s) file and we weren’t sure how to use it. Luckily, many other people on the internet had the same problem. One person(Koumoto) converted the source code into C language and posted it to the ARM community forum. We created a new c file in our uVision, gave it a new name like “arm_myBitReversal.c,” and used his C code. Finally, we could run our project with our three CMSIS DSP functions without causing any errors.

Using ADC and DMA functions in Ping Pong mode, we could now get audio data from our pin and set our data array through our “data process” function. In this function we computed the fft on our values and found the magnitude of these complex values. These values were stored in our testOutput data array. Then, we found the peak value of the array and the index

where the peak occurred. We assumed the amplitude of the peak would correspond with the amplitude of the sound, while the index where the peak occurred corresponded with the frequency. However, the frequency values seemed to be giving us fairly random values. With two of our group members catching COVID in the last month, time was running out more quickly than we thought it would, and we decided to switch our project idea from being a frequency detecting sound device to a music LED display. We took the Peak Value of our frequency array and used it to determine which LED color to light up. While it wasn't what we originally had intended to do, the result of the project was still something we were proud of. Whenever we played a song and put our microphone near our speaker, we got a really interesting light show. Watching the LED colors changing with the rhythm of the music was mesmerizing, and overall it was a very fun project to implement.

If we had more time or we continued working on the project in the future, I think there is a lot we could add to our code. Firstly, I think with more time, we could figure out how to use the FFT functions better and find out why our frequency values were off. I think we could also find the RMS value of our data in the time domain to help us find the amplitude instead of using the peak frequency value. We could use a button switch to loop through these three different sound values so that we can see the pattern of each. Finally, we could also add external LED lights to our tiva board to make the display more interesting.

Team Work

The whole group contributed to the research of the project's design. Elizabeth focused on discovering the resources that would provide the best codes and algorithms for the Fast Fourier Transform, Jennifer and Christie did a lot of basic research to best discover how Direct Memory Access works with Ping Pong Mode. The entire group would meet on Discord voice chat every week during the time of project designs and would stay on chat for hours at a time updating each other on resources and new information that we learned. We all contributed to a Google Document titled "Notes/References" which contained notes we took from Professor Donavan, new information, ideas, examples, other links, and resources so that we could try to learn at the same pace.

Conclusion and Future Work

In the future, we plan on continuing our project individually, considering updating each other on any successful attempts at improving this same project to be what our original intended project was supposed to be. Our intended project was a frequency detector device which would not just trigger LEDs at peak amplitudes, but it would instead trigger LEDs at specific frequencies or frequency ranges. We succeeded in getting the system's signal to translate from an analog signal to a digital signal, we succeeded in getting it from a time domain to a frequency domain, however we failed at retrieving the right idea on how to code for this exact idea.

References

Texas Instruments. Analog | Embedded Processing | Semiconductor Company | Ti.com. CMSIS DSP Software Library. Real FFT functions. (2022, May). Retrieved May 19, 2022, from https://www.keil.com/pack/doc/CMSIS/DSP/html/group__RealFFT.html

Convert arm_bitreversal2.S file to c code - Keil forum - Support forums - Arm Community. (n.d.). Retrieved May 22, 2022, from https://community.arm.com/support-forums/f/keil-forum/5986/convert-arm_bitreversal2-s-file-to-c-code

DMAandUARTBuffer. iLearn at San Francisco State University: Log in to the site. (n.d.). Retrieved May 22, 2022, from https://ilearn.sfsu.edu/ay2122/pluginfile.php/1087062/mod_resource/content/1/MidtermProject.c

Embedded signals. (n.d.). Retrieved May 22, 2022, from http://www.embeddedsignals.com/ARM.htm

Frequency Bin Example. (n.d.). Retrieved May 22, 2022, from https://www.keil.com/pack/doc/CMSIS/DSP/html/group__FrequencyBin.html

Getting started with STM32 - working with ADC and DMA. Digikey. (n.d.). Retrieved May 22, 2022, from https://www.digikey.com/en/maker/projects/getting-started-with-stm32-working-with-adc-and-dma/f5009db3a3ed4370acaf545a3370c30c

H., Y. L. (2020, October 13). Music to led strip tutorial (using Fourier transform). Medium. Retrieved May 22, 2022, from https://medium.com/@yolandaluqueh/music-to-led-strip-tutorial-using-fourier-transform-3d203a48fe14

Mathworks. (n.d.). Hann (Hanning) window - MATLAB. Retrieved May 22, 2022, from https://www.mathworks.com/help/signal/ref/hann.html

Saiyam, & Instructables. (2017, October 1). Music reactive multicolor LED LIGHTS. Instructables. Retrieved May 22, 2022, from https://www.instructables.com/Music-Reactive-Multicolor-LED-Lights/

Tiva C series TM4C1233H6PM microcontroller data sheet ... - ti.com. (n.d.). Retrieved May 23, 2022, from https://www.ti.com/lit/ds/symlink/tm4c1233h6pm.pdf

user486191. (2018, May 2). Automatic Guitar Tuner. Digilent Projects. Retrieved May 22, 2022, from https://projects.digilentinc.com/user486191/automatic-guitar-tuner-0b0438

Source Code

finalProject.c

```
#include <stdint.h>
#include <stdbool.h>
#define ARM_MATH_CM4 1
#include "arm_math.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_adc.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/udma.h"


#include "inc/tm4c123gh6pm.h"

#define RED_MASK 0x02
#define BLUE_MASK 0x04
#define GREEN_MASK 0x08

// The standard udma control table
#if defined(ewarm)
#pragma data_alignment=1024
uint8_t ui8ControlTable[1024];
#elif defined(ccs)
#pragma DATA_ALIGN(ui8ControlTable, 1024)
uint8_t ui8ControlTable[1024];
#else
uint8_t ui8ControlTable[1024] __attribute__ ((aligned(1024)));
#endif



// a and b buffers
#define ADC_BUF_SIZE       512
```

```
static uint32_t ui32BufA[ADC_BUF_SIZE];
static uint32_t ui32BufB[ADC_BUF_SIZE];


static uint8_t ui8Flags = 0;
static float32_t maxTest = 0;
static float32_t freqValue;

void ADC0_Init(void)
{


SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTA
L_16MHZ); // configure the system clock to be 40MHz
                SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);//activate the clock of ADC0
                SysCtlDelay(2);         //insert a few cycles after enabling the peripheral to allow
the clock to be fully activated.
                ADCSequenceDisable(ADC0_BASE, 1); //disable ADC0 before the configuration
is complete
                ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_TIMER, 1); //Use
with TimerControlTrigger(uint32_t ui32Base,uint32_t ui32Timer,bool bEnable)
        ADCSequenceStepConfigure(ADC0_BASE, 1, 0,
ADC_CTL_CH0|ADC_CTL_IE|ADC_CTL_END); //ADC0 SS1 Step 0, sample from channel 0
   IntPrioritySet(INT_ADC0SS1, 0x00);        // configure ADC0 SS1 interrupt priority as 0
                IntEnable(INT_ADC0SS1);                                 // enable interrupt 31 in NVIC
(ADC0 SS1)
                ADCIntEnableEx(ADC0_BASE, ADC_INT_SS1);     // arm interrupt of ADC0 SS1
                ADCSequenceEnable(ADC0_BASE, 1); //enable ADC0
}


void Timer0A_Init(void) //ADC 10KHz trigger

{
        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
        TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
        TimerLoadSet(TIMER0_BASE,TIMER_A,999);//40,000,000/40,000 - 1
        TimerControlTrigger(TIMER0_BASE, TIMER_A,true);
        IntPrioritySet(INT_TIMER0A,0x00);
        IntEnable(INT_TIMER0A);
        TimerEnable(TIMER0_BASE,TIMER_A);
}

void DMA_Init(void)
```

```
{
   SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);// Enable the clock
   uDMAEnable(); // Enable the uDMA
   uDMAControlBaseSet(ui8ControlTable); //use the ui8ControlTable

         //disable attributes of udma channel
   uDMAChannelAttributeDisable(UDMA_CHANNEL_ADC1,
                        UDMA_ATTR_ALTSELECT |
                        UDMA_ATTR_HIGH_PRIORITY |
                        UDMA_ATTR_REQMASK);

//set channel paramters of primary data structure
   uDMAChannelControlSet(UDMA_CHANNEL_ADC1 | UDMA_PRI_SELECT,
                  UDMA_SIZE_32 | UDMA_SRC_INC_NONE | UDMA_DST_INC_32 |
                  UDMA_ARB_1);

//set channel paramters of alternate data structure
   uDMAChannelControlSet(UDMA_CHANNEL_ADC1 | UDMA_ALT_SELECT,
                  UDMA_SIZE_32 | UDMA_SRC_INC_NONE | UDMA_DST_INC_32 |
                  UDMA_ARB_1);

//set transfer paramters of primary data structure. uses buffer A
   uDMAChannelTransferSet(UDMA_CHANNEL_ADC1 | UDMA_PRI_SELECT,
                        UDMA_MODE_PINGPONG,
                        (void *)(ADC0_BASE + ADC_O_SSFIFO1),
                        ui32BufA, ADC_BUF_SIZE);


//set transfer paramters of alternate data structure. uses buffer B
                  uDMAChannelTransferSet(UDMA_CHANNEL_ADC1 | UDMA_ALT_SELECT,
                        UDMA_MODE_PINGPONG,
                        (void *)(ADC0_BASE + ADC_O_SSFIFO1),
                        ui32BufB, ADC_BUF_SIZE);
                  //enables the channel
   uDMAChannelEnable(UDMA_CHANNEL_ADC1);

}



void ADC0_Handler(void)
{
         //ADC ISR
```

```
        //clear SS1
                ADCIntClear(ADC0_BASE, 1);
        //create int variable to store current DMA mode
                uint32_t ui32Mode;

        //get current mode (ping pong mode) of primary data structure
    ui32Mode = uDMAChannelModeGet(UDMA_CHANNEL_ADC1 | UDMA_PRI_SELECT);

        //when transfer is complete, mode is STOP
    if(ui32Mode == UDMA_MODE_STOP)
    {
                                ui8Flags |= 0x01;

                                //reload DMA and set mode to pingpong
       uDMAChannelTransferSet(UDMA_CHANNEL_ADC1 | UDMA_PRI_SELECT,
                                                UDMA_MODE_PINGPONG,
                                                (void *)(ADC0_BASE + ADC_O_SSFIFO1),
                                                ui32BufA, ADC_BUF_SIZE);

                                //enable the DMA channel

                                uDMAChannelEnable(UDMA_CHANNEL_ADC1);
    }


                //get current mode (ping pong mode) of alternate data structure
    ui32Mode = uDMAChannelModeGet(UDMA_CHANNEL_ADC1 | UDMA_ALT_SELECT);


                //when transfer is complete, mode is STOP
    if(ui32Mode == UDMA_MODE_STOP)
    {
                                ui8Flags |= 0x02;

                                //reload DMA and set mode to pingpong
                                uDMAChannelTransferSet(UDMA_CHANNEL_ADC1 |
UDMA_ALT_SELECT,
                                                UDMA_MODE_PINGPONG,
                                                (void *)(ADC0_BASE + ADC_O_SSFIFO1),
                                                ui32BufB, ADC_BUF_SIZE);

                                //enable the uDMA channel

                                uDMAChannelEnable(UDMA_CHANNEL_ADC1);
```

```
    }


}


void dataProcess(uint32_t *data){
        //create array of floats because ARM functions only accept float values
        static float32_t testInput[512];

        float32_t ave = 0;
                for (uint16_t i = 0; i<512; i++){
                        ave+=data[i];
                        //multiply by hanning window
                        //https://www.mathworks.com/help/signal/ref/hann.html
                        //data[i] = (0.5*(1- cos ( 2.0 * PI * i/511)))*data[i];


                        //fill float array with int data
                        testInput[i] = data[i];



                }

        //calculate average in case its needed
        ave = (ave)/512;

        //create output array data for values in frequency domain
        static float32_t testOutput[512];
        uint32_t ifftFlag = 0;
        uint32_t doBitReverse = 1;
        arm_cfft_instance_f32 varInstCfftF32;
        uint32_t testIndex = 0;


        //arm_status status;
  float32_t maxValue;

        //find fft of values to change to frequency domain
  arm_cfft_f32(&varInstCfftF32, testInput, ifftFlag, doBitReverse);

        //find magnitude of the complex values
  arm_cmplx_mag_f32(testInput, testOutput, 256);
```

```
        //find the peak value and the index where the peak occurs
  arm_max_f32(testOutput, 256, &maxValue, &testIndex);

        //store peak value
        //set global value so we can check it in the main function
        maxTest = maxValue;

        //frequency value should be determined by index of peak, multiplied by bin length but it
doesn't seem to be working?
        freqValue = testIndex*256;


}

void PortFunctionInit(void) //GPIOF and PWMs
{
                SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
                SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = 0x1;

                GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_3);

                GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);

    //
    // Enable pin PF1 for GPIOOutput
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);

    //
    // Enable pin PF3 for GPIOOutput
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3);



}



int main(void)
{
```

```
        //400*100 =40khz
    ADC0_Init();
    DMA_Init();
        Timer0A_Init();
    PortFunctionInit();

        while(1)
        {
            if ((ui8Flags&0x01) != 0) {dataProcess(ui32BufA);ui8Flags&=~0x01;}
            if ((ui8Flags&0x02) != 0) {dataProcess(ui32BufB);ui8Flags&=~0x02;}


                                    if (maxTest<3000) // LEDs off
    {
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0x00);
    }
    else if (maxTest<3050) // Red
    {
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, RED_MASK);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0x00);
    }
    else if (maxTest<3100) // Blue
    {

GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, BLUE_MASK);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0x00);

    }
    else if (maxTest<3150) // Purple
    {
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, RED_MASK);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, BLUE_MASK);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0x00);
    }
                                    else if (maxTest<3200) // Green
    {

GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GREEN_MASK);
```

```
        }
        else if (maxTest<3250) // Yellow (Red and Green)
        {
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, RED_MASK);
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GREEN_MASK);

        }
        else if (maxTest<3300) // Cyan (Blue and Green)
        {
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, BLUE_MASK);
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GREEN_MASK);

        }
        else if (maxTest<3350) // White (all LED on)
        {
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, RED_MASK);
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, BLUE_MASK);
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GREEN_MASK);
        }

                }

}
```