Department of Engineering

EKG MANUAL/WRITEUP

SUBJECT: Embedded Systems Design

ENGR 355

2021-2022

By: Joshua Mularczyk

For: Dr. Aamodt

Table of Contents

Abstract

For this project, the task was to design a hand-held portable instrument that would analyze a periodic analog input signal i.e. an EKG (electrocardiogram). I was tasked with determining the rate of the analog input signal (heart rate) as well as producing an output analog signal, voltage, that could be stored in Microcontroller memory. Over the three months, I learned and designed the software for the PIT, Switches, ADC (analog to digital converter), DAC (digital to analog converter), LCD display, and my menu. I assembled the hardware for my NXP FRDM-KL25Z board attachments (circuit board with switches, LEDs, and LCD), and connected them to the NXP Microcontroller. At the end of the project, I assembled the hardware for a complete circuit board including the MKL25Z processor, 25LC128 memory, switches, and LCD display. I completed about 90% of the project. I had 5 out of the 7 working modes. I was not able to accomplish storing a previously sampled data amount and downloading anything to external memory due to lack of time and I was not able to communicate with the all-in-one designed circuit board for unknown reasons.

Device Description

The device created in this project is a portable handheld electrocardiogram. This analyzed an analog input signal, specifically from the heart, and allows a user to do multiple things with this data. Data can be stored in customized amounts and at different rates. The analog signal can also be output to a device such as an oscilloscope. The device currently needs to be powered by a USB port, but has potential for being battery powered in the future. The handheld device is 2.03 in x 1.02 in x 6.5 in (LxWxH). The device can operate at low frequencies to frequencies at about 15 Hz.

Department of Engineering

EKG USER MANUAL

SUBJECT: Embedded Systems Design

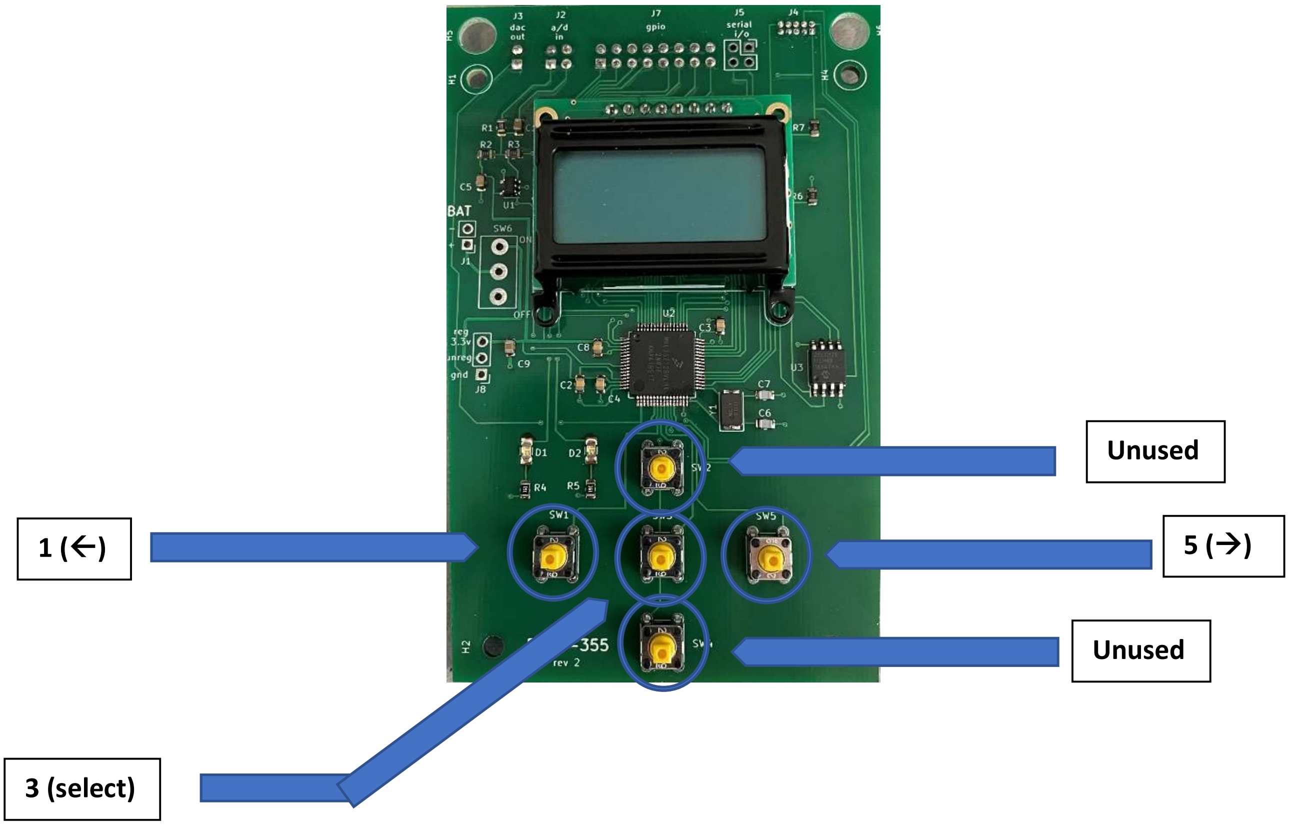
ENGR 355

2021-2022

EDWARD F. CROSS SCHOOL OF
ENGINEERING
WALLA WALLA UNIVERSITY

By: Joshua Mularczyk

For: EKG User



Buttons

Three out of the five buttons are used for this device. The buttons are used to navigate through the device menu.

Button 1: Moves left through the menu

Button 2: Unused

Button 3: Used to select choices/enter modes/exit modes

(Press once to select mode. Press again to return to menu)

Button 4: Unused

Button 5: Moves right through the menu

Menu Navigation

The menu for this device consists of seven different modes: EKG Mode, Samp.ADC (STOR), Samp.ADC (FUTR), sampRate, set.Data, DAC.outp, and Download. The LCD start up in EKG mode, then allows the user to navigate through the other modes.

Startup

To power on the EKG, make sure the batteries have charge and flip the switch to "ON" (this hasn't been implemented yet, so the other way to power this is via USB cable).

EKG.MODE

- EKG.MODE is the default display for the EKG and outputs the BPM (beats per minute) of the input analog signal (machine or heart).
- The lower half of the screen will change as input varies. It will settle on its final value after a couple of seconds.
- There is no further selection for this mode.

Samp.ADC (STOR)

- Samp.ADC (STOR) samples the analog input (at a previously set rate) and when the user presses the specified button (either button 2 or 4, but not yet determined) and stores the preceding number of data samples in RAM memory for use by the DAC.
- This mode has not been set up yet due to the fact that RAM memory has not been communicated with yet.
- There is no further selection for this mode.

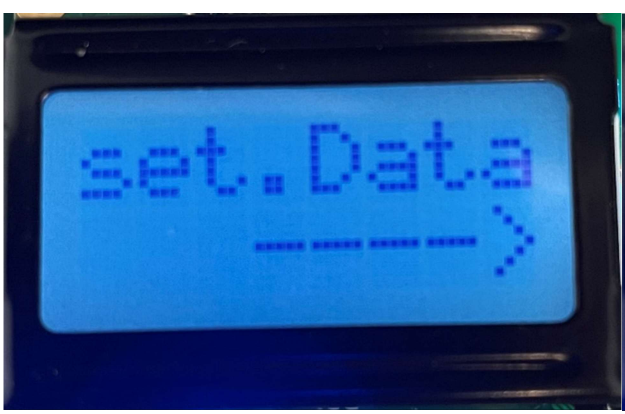
Samp.ADC (Futr)

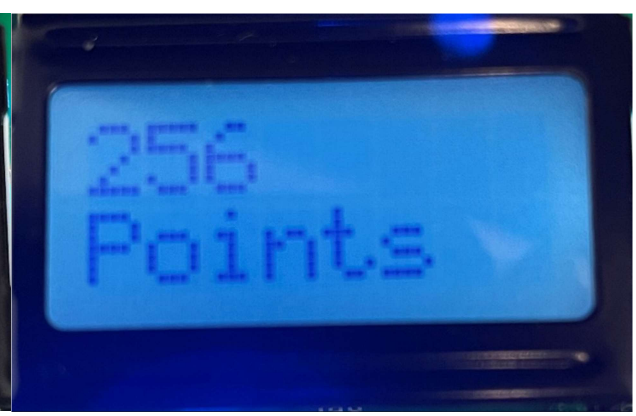
- Samp.ADC (Futr) has the same function as Samp.ADC (STOR) but instead of storing previously sampled points at the press of the button, it stores the point from the button press until the specified number of data points (selected in set.Data mode).
- This mode has not been set up yet due to the fact that RAM memory has not been communicated with yet.
- There is no further selection for this mode.

sampRate

- sampRate is a mode that is used to select the sample rate of the ADC. Rates available range from 50 to 1,000 samples per second (Hz).
- To enter this mode press the select button (3). Use buttons 1 and 5 to move left and right through the options. When desired option is reached, press the select button (3) one more time to return to the main menu and this will update the sample rate.

set.Data



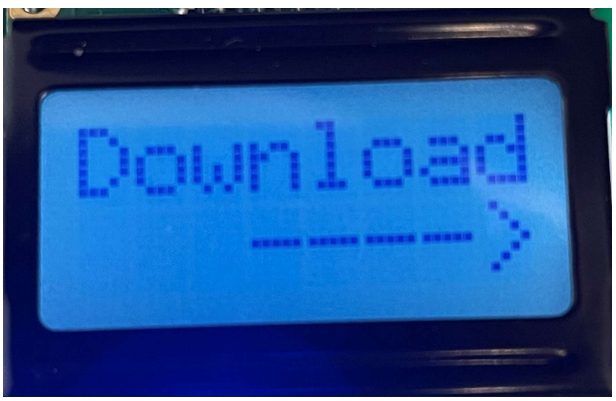


- Set.Data is uset to set the number of data points in a waveform. Number of data points available are 128, 256 (default), 512, and 1024.
- To enter this mode press the select button (3). Use buttons 1 and 5 to move left and right through the options. When desired option is reached, press the select button (3) one more time to return to the main menu and this will update the number of data points in the waveform.
- This should be chosen before using the Samp.ADC modes or else it will just use the default 256 points.

DAC.outp

- DAC.outp mode creates an output waveform using data currently being input from the ADC (analog digital converter) or DAC memory (this is not used since memory has not been implemented yet).
- To display the waveform connect to a electronic test instrument (oscilloscope), press the select button while in DAC.outp mode.
- There is no further selection for this mode.

Download



*This mode is currently not in use

Hardware Design Notes

While dealing with the hardware, there were a few things that needed to be kept in mind:

- When soldering, bridges between connections could not be formed
- To power the device, the USB cord had to be connected to the SDA port
- The waveform generator was connected to the two ADC ports
- The oscilloscope was connected to the two DAC ports
- I needed to make sure that there was an offset in the analog signal voltage of around 1.5 volts so that my board did not heat up and destroy itself

Software Design Notes

I seperated my software into eight different files: Main.c, ADC, DAC, PIT, LCD functions, debug, EKG Functions, and Switches. I will step through them below.

Main.c

- Initialize the PIT
- Set clocks for A, B, C, D (E is set in the ADC and DAC)
- Enable and initiate everything used from the other files
- While loop

- Menu for EKG using a Switch statement
    - Case 1: Displays EKG mode and runs the ekgMode function (described in the EKG functions file below)
    - Case 2: Displays ADC sampling mode
        - If the select switch is pressed then it runs the outputPrev function (hasn't been created yet)
    - Case 3: Displays the second ADC sampling mode
        - If the select switch is pressed then it sets flag "Future" which will allow the filling of an array of specified data points at that time.
    - Case 4: Displays the rate select mode
        - Runs the setRateDisplay functions (which is desbribed in the EKG function section)
    - Case 5: Displays the set data mode
        - If the select button is pressed then the dataSelect Function is run to select data amount.
    - Case 6: Displays DAC output onto external device
        - If select button is pressed than the DAC is displayed
    - Case 7: Not implimented yet.

Switches.c

- Defined the five switches being used
- Initialized switches
- Switch IRQ handler
    - Clear pending interrupts
    - If swtich 5 is pressed
        - Increment positively through screens, rates, or datas depending on what mode you are in.
    - If switch 1 is pressed
        - Decrement through screens, rates, or datas depending on what mode you are in.
    - If switch 3 is pressed
        - Increment flags for the six different modes that contol entering and exiting the modes.
    - Switches 4 and 2 havent been used yet
    - Clear status flags

PIT.c

- Function to initialize the PIT

  - Enable clock to PIT module
  - Enable module, freeze timers in debug mode
  - Initialize PIT0 to count down from arugument
  - No chaining
  - Generate interrupts
- Function to Start PIT
  - Enable counter using |=
- Function to stop PIT
  - Enable counter using &=~
- Function for PIT IRQ
  - Clear pending IRQ
  - Check to see which channel triggered interrupt
  - Clear status flag for timer channel 0
  - Do ISR work
  - Clear status flags
- Function to change input period to user selected value
  - Use the same line of code from the initializeing PIT0

ADC.c

- Function to initialize ADC
  - Enable clock to ADC and Port E
  - Set pin signal type to analog
  - Set pins (ADCO alternate trigger enable, ADCO pretrigger select, ADCO trigger select)
  - Set pins (Normal power config, divide ratio of 1, short sample time, 12 bit sigle-ended conversion, (Bus clock)/2 input)
  - Set pins (hardware tigger selected, compare functin disabled, DMA disabled, Default voltage reference pair pin)
  - Set pins (one conversion or sets of conversions, hardware average function enabled, 4 samples avereaged)
  - Diff mode enable, input chanel select
- Function for ADC interrrupt fuction initialization
  - Enable interrupts
- Function for ADC IRQ work
  - Read result from ADC
  - Read analog value directly to dac when dac mode from main is selected
  - Determine the amplitude of the waveform to set a max point

  - Store the input from the ADC directly into an array
  - Store the input from the ADC into an array at the press of a button.
  - Two if statements used to figure out when a wave was rising and dropping.

DAC.c

- Function for initialization of the DAC
  - Enable clock to Dac and Port E
  - Set pin signal type to analog
  - Disable buffer mode
  - Enable DAC, select VDDA as reference voltage

EKG_Functions.c

- Function to display different options for rate select
  - Switch statement with 11 cases to choose values between 50 and 1000. Variable is set that changes period in ADC.
- Function to display different options for data select
  - Switch statement with 4 cases to choose values 128, 256, 513, and 1024.
- Function to display EKG BPM

  - Takes the saveBPM array created in the ADC and averages the last 4 reads and displays that to the LCD

Debug_signals.c

- Function to initialize the debug signals
  - Enable clock for port b
  - Select GPIO for pins connected to debug signals
  - Set bits to outputs
  - Clear output signals initially

Lcd_lib_4bit_20B.c

- *this was a library consisting of functions for the LCD display that was provided by the instructor. See full code in the appendix

Build Output Window

Build Output

```
Build started: Project: Project
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
".\Objects\Project.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:00
```

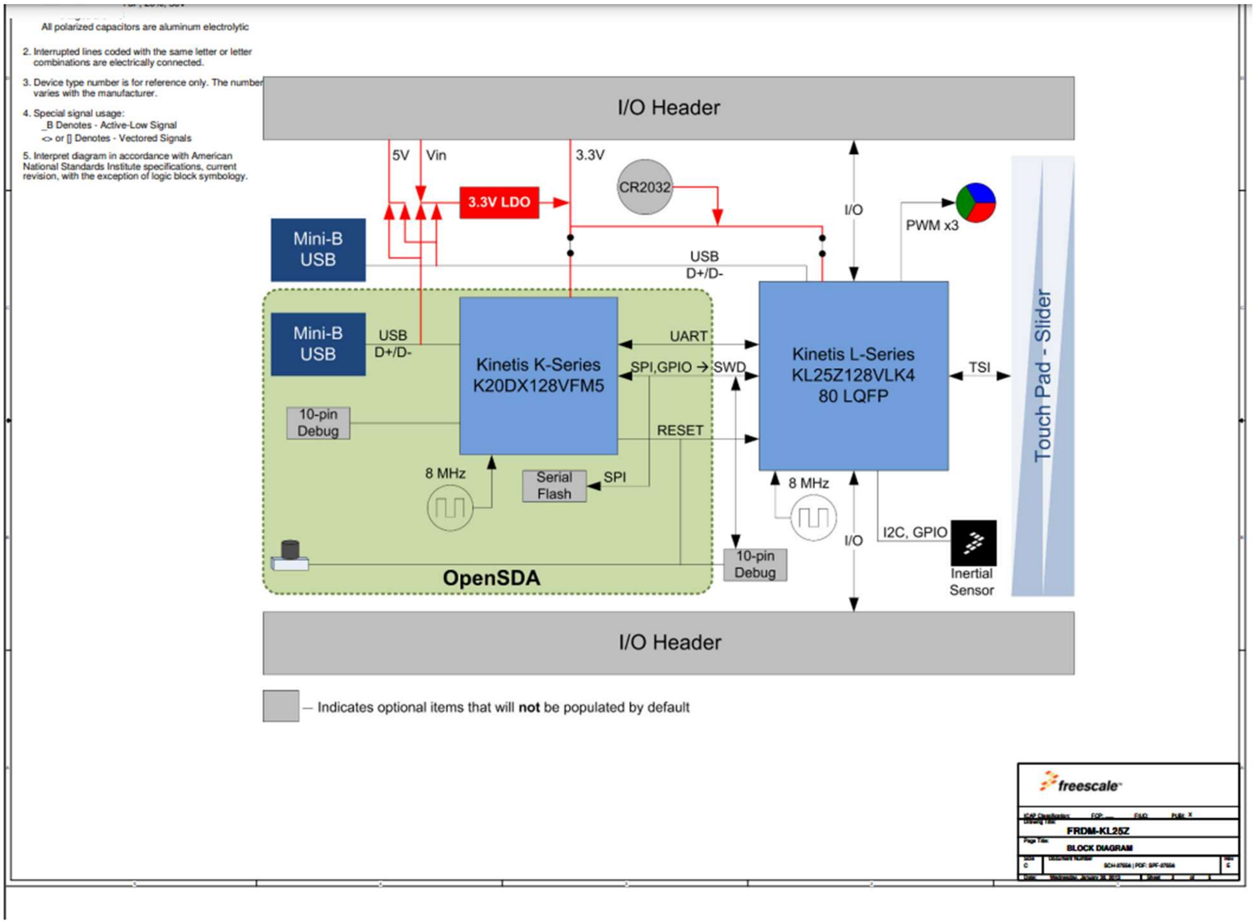
Comments

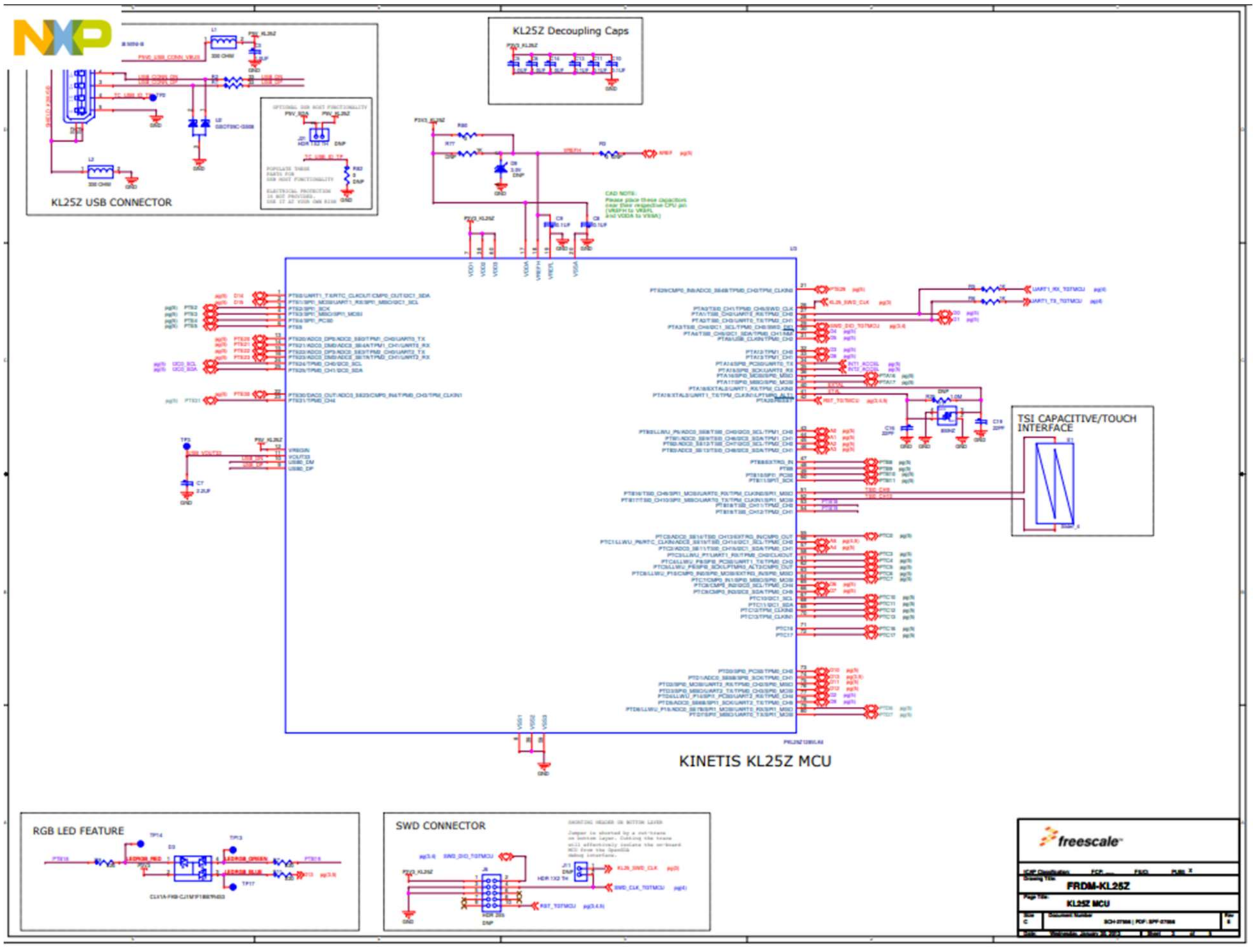
I really enjoyed this project. This project allowed me to learn about embedded systems working with both hardware and software. I was able to retouch up on my skills in C as well as work with soldering a little bit more. I was happy with my results since my EKG does the main tasks that were sought out at the beginning. I would like to spend some extra time getting my samp.ADC (Stor) mode completed as well as figuring out how to get my computer to talk to the circuit that I created at the end of the project.

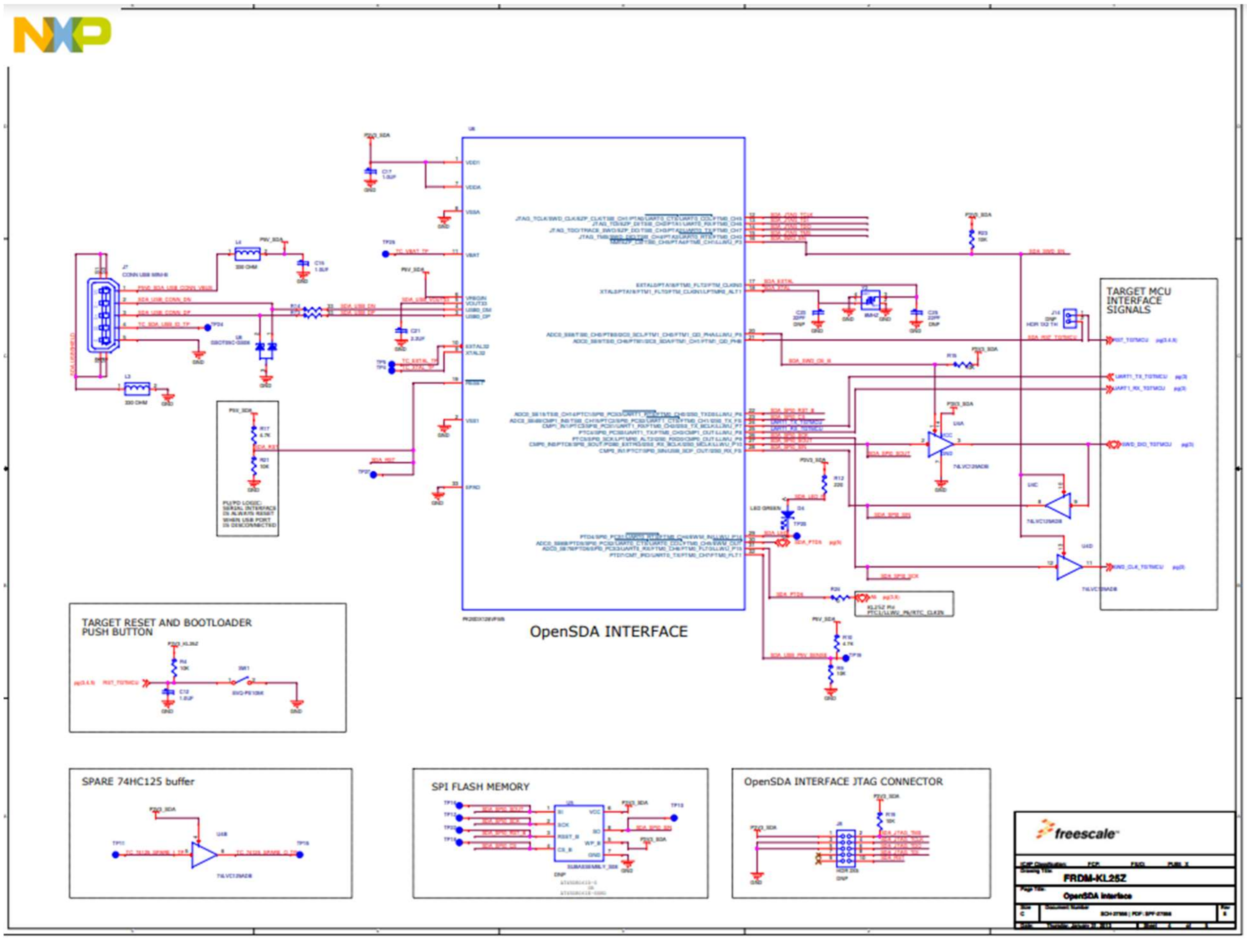
Appendix

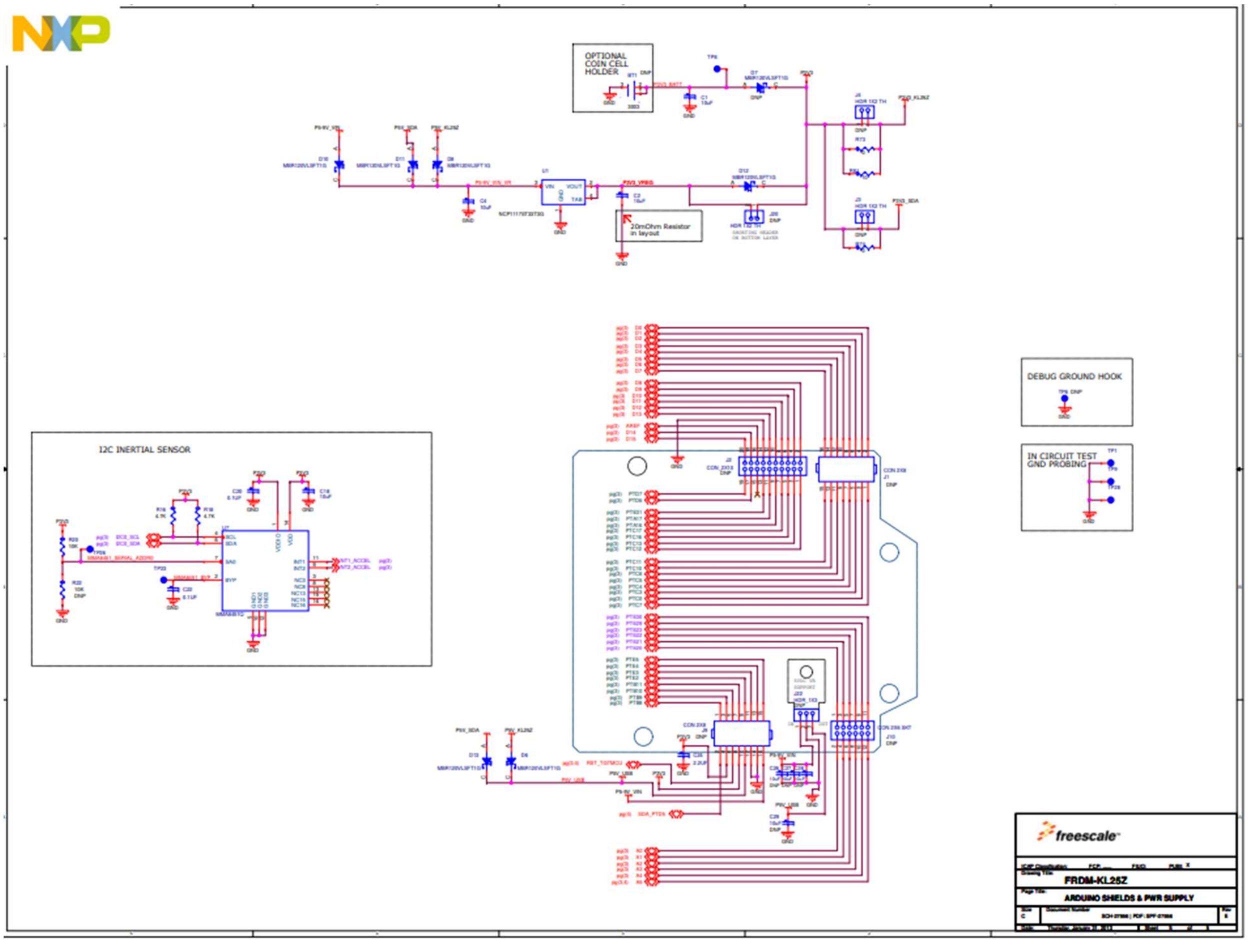
Schematics

FRDM-KL25Z development board




KL25Z Decoupling Caps
KL25Z USB CONNECTOR
KINETIS KL25Z MCU
TSI CAPACITIVE/TOUCH INTERFACE
RGB LED FEATURE
SWD CONNECTOR
freescale
FRDM-KL25Z
KL25Z MCU


OpenSDA INTERFACE
TARGET MCU INTERFACE SIGNALS
TARGET RESET AND BOOTLOADER PUSH BUTTON
SPARE 74HC125 buffer
SPI FLASH MEMORY
OpenSDA INTERFACE JTAG CONNECTOR
freescale
FRDM-KL25Z
OpenSDA Interface



OPTIONAL COIN CELL HOLDER
20mOhm Resistor in layout
I2C INERTIAL SENSOR
DEBUG GROUND HOOK
IN CIRCUIT TEST GND PROBING
freescale
FRDM-KL25Z
ARDUINO SHIELDS & PWR SUPPLY

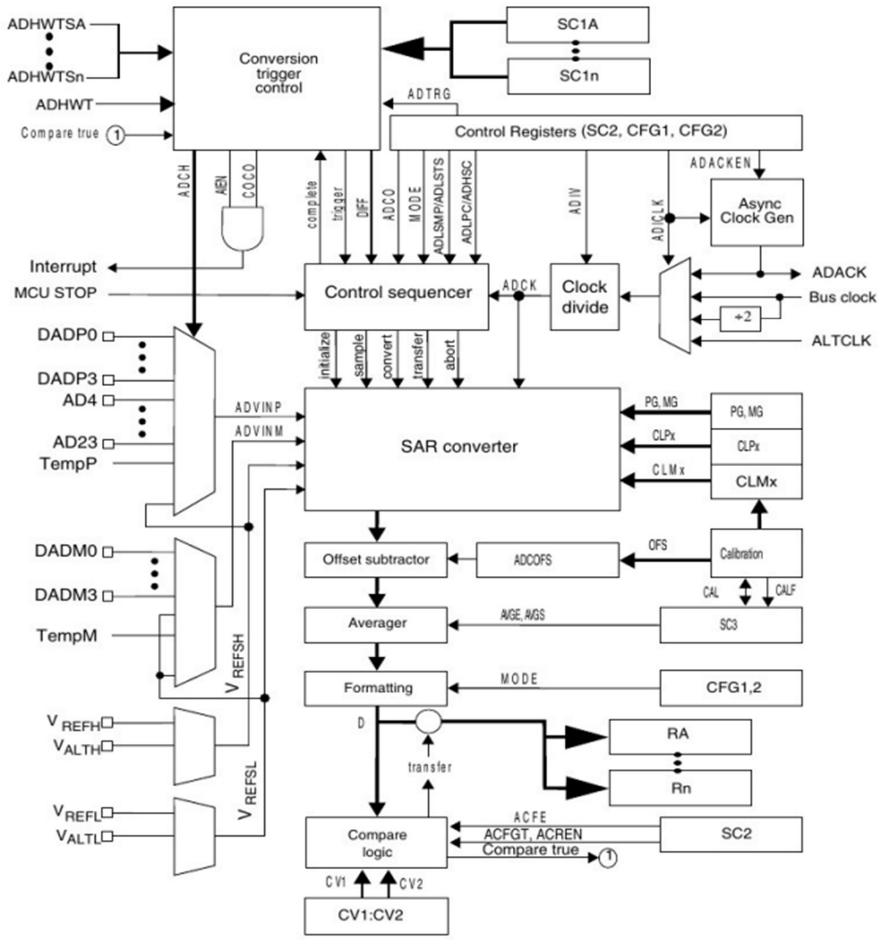
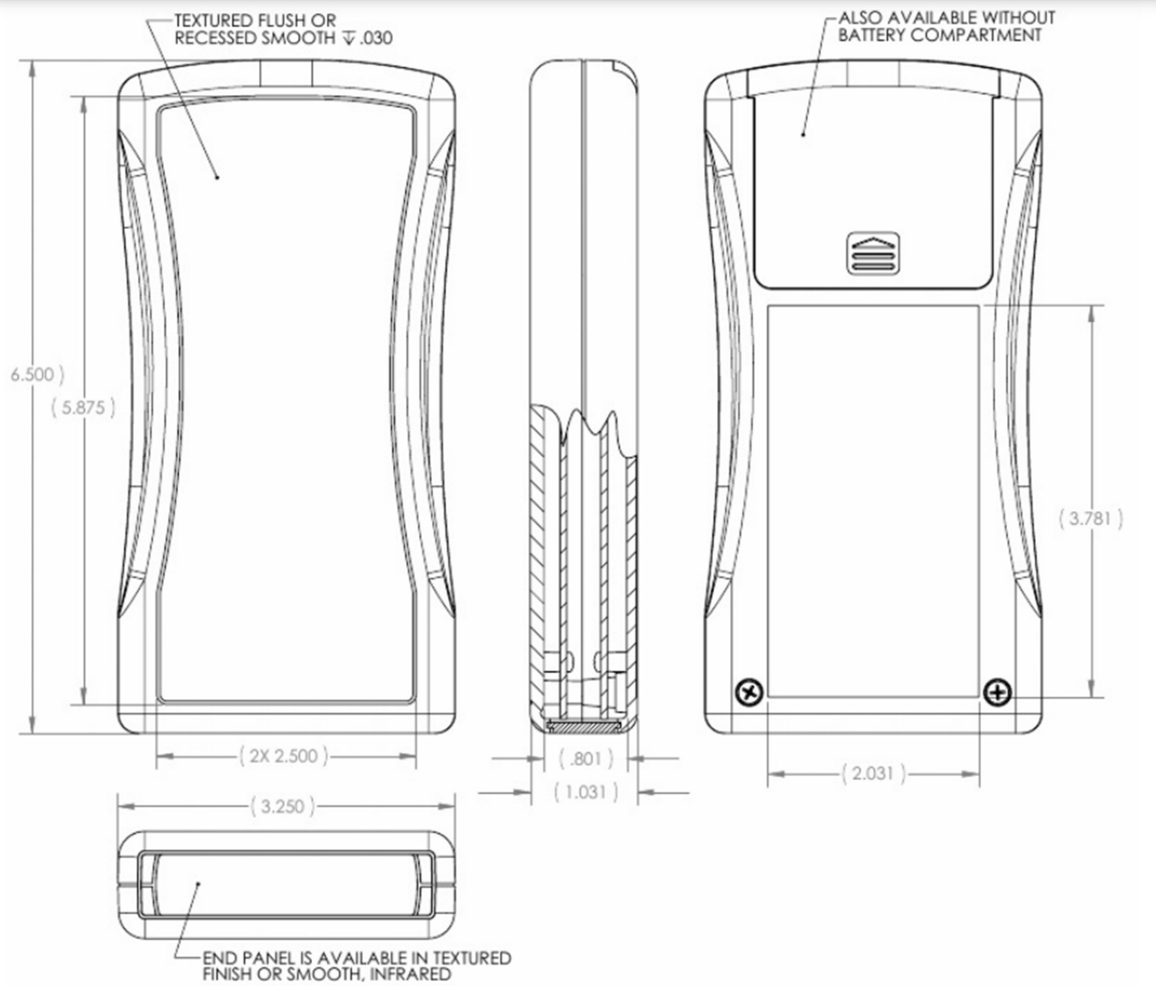
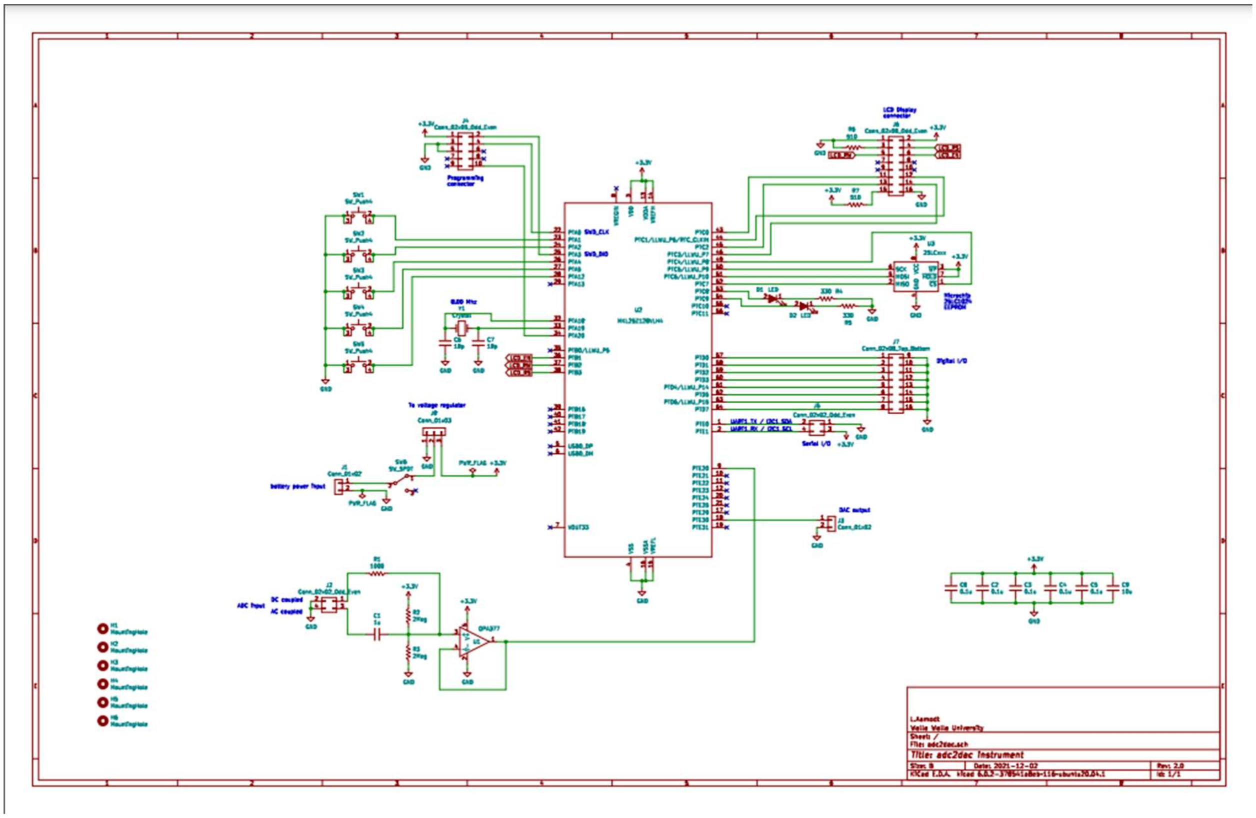


Figure 28-1. ADC block diagram

Casing for the EKG



Instructor created PCB



Source Files

```
//
***************************************************************************/
/
// Program for EKG utilization
                                                                        //
//
//
// Filename: Main.c


//
// Author: Joshua Mularczyk

                                                     //
// Version: 03/2/22 written

                                               //
```

```
// Processor: NXP MKL25Z4

                                                   //
// Compiler: Keil uVision5
//
// Library: CMSIS core and device startup
//
// also needs lcd_lib_4bit_20b.c
//
//                                         debug_signals.c


              //
//                                         switches.c


                     //
//                                         DAC.c


                                    //
//                                         Pit.c


                                    //
//                                         ADC.c


                                    //
//           EKG_Functions.c
// Hardware: NXP Freedom board connected to a 16x2 LCD display
//
// Software note: This program is a "bare metal" application since it
//
// doesn't use an operating system.
//
// Operation: Uses a timer to create a periodic inerrupt 10 times a second
//
//
**********************************************************************/
/

#include "MKL25Z4.h"
#include <stdint.h>
#define MASK(x) (1UL << (x))
#define DEBUG_PORT PTD

void delayMs(uint32_t n);
void LCD_init(void);
void LCD_command(uint32_t command);
void LCD_send_data(uint32_t data);
void init_debug_signals(void);
void init_switch(void);
void Init_DAC(void);
```

```
void Init_PIT(uint32_t period);
void Start_PIT(void);
void PIT_IRQHandler(void);
void Init_ADC(void);
void Init_ADC_Interrupts(void);
void display_string(char string[]);
void setRateDisplay(void);
void dataSelect(void);
void ekgMode(void);
void outputPrev(void);
void outputFuture(void);

extern unsigned int period;
volatile unsigned int ratesel = 0;
extern unsigned int Future;
int dataSel = 0;
int screen = 1;
int DacOn = 0;
int rate = 1;
int datas = 2;
extern int mode2;
extern int mode3;
extern int mode6;
extern int ekgselect;
int storedValues[1024];
int FutureValues[1024];
volatile unsigned int sampleRate = 0;

//**************************************************************************
***
//  Main function

                              Joshua Mularczyk
//**************************************************************************
***
int main (void) {

       // initialization of the PIT

       Init_PIT(24000);
       Start_PIT();

       // Set clock

       SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;
       SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;
       SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK;
       SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

       __enable_irq();
       init_debug_signals();
       Init_DAC();
       Init_ADC();
       Init_ADC_Interrupts();
```

```
LCD_init();
LCD_command(0x01);
init_switch();

while(1){

        delayMs(500);
        LCD_command(0x80);

        //Menu for EKG

        switch (screen){
                case 1:
                        display_string("EKG.MODE");
                        LCD_command(0xC0);
                        ekgMode();
                        break;
                case 2:
                        display_string("Samp.ADC");
                        LCD_command(0xC0);
                        display_string("Stor--->");
                if(mode2 == 1){
                        outputPrev();
                };
                break;
                case 3:
                        display_string("Samp.ADC");
                        LCD_command(0xC0);
                        display_string("Futr--->");
                if(mode3 == 1){
                        Future = 1;
                };
                break;
                case 4:
                        display_string("sampRate");
                        LCD_command(0xC0);
                        display_string("   ---->");
                if(ratesel == 1){
                                setRateDisplay();
                        };
                break;
                case 5:
                        display_string("set.Data");
                        LCD_command(0xC0);
                        display_string("   ---->");
                if(dataSel == 1){
                                dataSelect();
                        };
                break;
                case 6:
                        display_string("DAC.outp");
                        LCD_command(0xC0);
                        display_string("   ---->");
                if(mode6 == 1){
```

```
                                DacOn = 1;
                        }
                        else {
                                DacOn = 0;
                        };
                        break;
                        case 7:
                                display_string("Download");
                                LCD_command(0xC0);
                                display_string("   ---->");
                        break;
                }
        }
};
```

```
//**********************************************************************
***
//  file:  switches.c
//**********************************************************************
***

#include <MKL25Z4.H>
#include <stdint.h>

#define MASK(x) (1UL << (x))

// Debug status bits
#define DBG_ISR_POS (0)
#define DBG_MAIN_POS (1)
#define DEBUG_PORT PTD

// Switches is on port D for interrupt support
#define SW1_POS (1)
#define SW2_POS (2)
#define SW3_POS (4)
#define SW4_POS (5)
#define SW5_POS (12)

extern void init_switch(void);
extern int ratesel;
extern int dataSel;
extern int screen;
extern int rate;
extern int datas;
int mode2 = 0;
int mode3 = 0;
int mode6 = 0;
int ekgselect;
```

```
//**********************************************************************
***
//  switch initialization function
                                                                 Larry
Aamodt
//**********************************************************************
***
void init_switch(void) {
	SIM->SCGC5 |=  SIM_SCGC5_PORTA_MASK; /* enable clock for port A
(was D) */

	/* Select GPIO and enable pull-up resistors and interrupts
		on falling edges for pins connected to switches */

	// Initialize button 1
	PORTA->PCR[SW1_POS] |= PORT_PCR_MUX(1) | PORT_PCR_PS_MASK |
PORT_PCR_PE_MASK | PORT_PCR_IRQC(0x0a);
	/* Set port A switch bit to inputs */
	PTA->PDDR &= ~MASK(SW1_POS);


	// Initialize button 2
	PORTA->PCR[SW2_POS] |= PORT_PCR_MUX(1) | PORT_PCR_PS_MASK |
PORT_PCR_PE_MASK | PORT_PCR_IRQC(0x0a);
	/* Set port A switch bit to inputs */
	PTA->PDDR &= ~MASK(SW2_POS);


	// Initialize button 3
	PORTA->PCR[SW3_POS] &= ~PORT_PCR_MUX_MASK;
	PORTA->PCR[SW3_POS] |= PORT_PCR_MUX(1) | PORT_PCR_PS_MASK |
PORT_PCR_PE_MASK | PORT_PCR_IRQC(0x0a);
	/* Set port A switch bit to inputs */
	PTA->PDDR &= ~MASK(SW3_POS);


	// Initialize button 4

	PORTA->PCR[SW4_POS] |= PORT_PCR_MUX(1) | PORT_PCR_PS_MASK |
PORT_PCR_PE_MASK | PORT_PCR_IRQC(0x0a);

	/* Set port A switch bit to inputs */
	PTA->PDDR &= ~MASK(SW4_POS);


	// Initialize button 5

	PORTA->PCR[SW5_POS] |= PORT_PCR_MUX(1) | PORT_PCR_PS_MASK |
PORT_PCR_PE_MASK | PORT_PCR_IRQC(0x0a);
	/* Set port A switch bit to inputs */
	PTA->PDDR &= ~MASK(SW5_POS);

	/* Enable Interrupts */
```

```
        NVIC_SetPriority(PORTA_IRQn, 128); // 0, 64, 128 or 192
        NVIC_ClearPendingIRQ(PORTA_IRQn);
        NVIC_EnableIRQ(PORTA_IRQn);
}
//**********************************************************************
***
//  Switch IRQ handler
                                                                  Joshua
Mularczyk
//**********************************************************************
***
void PORTA_IRQHandler(void) {
        DEBUG_PORT->PSOR = MASK(DBG_ISR_POS);

        // clear pending interrupts

        NVIC_ClearPendingIRQ(PORTA_IRQn);

        // code to determine what occurs if switch 5 is pressed

        if ((PORTA->ISFR & MASK(SW5_POS))){
                if((ratesel == 0) && (dataSel == 0)){
                        screen++;
                if (screen > 7){
                        screen = 1;
                        };
                };
                if (ratesel == 1){
                        rate++;
                if(rate > 11){
                        rate = 1;
                };
        };
                if (dataSel == 1){
                datas++;
                if(datas > 4){
                        datas = 1;
                        };
                };
        };

        // code to determine what occurs if switch 1 is pressed

        if ((PORTA->ISFR & MASK(SW1_POS))){
                if((ratesel == 0) && (dataSel == 0)){
                        screen--;
                if (screen == 0){
                        screen = 7;
                        };
                };
                if (ratesel == 1){
                        rate--;
                if(rate == 0){
                        rate = 11;
```

```
                };
        };
                if (dataSel == 1){
                datas--;
                if(datas == 0){
                        datas = 4;
                        };
                };
        };

// code to determine what occurs if switch 3 is pressed

        if ((PORTA->ISFR & MASK(SW3_POS))) {
                if(screen == 1){
                ekgselect++;
                if(ekgselect > 1){
                        ekgselect = 0;
                        };
                };
                if(screen == 2){
                mode2++;
                if(mode2 > 1){
                        mode2 = 0;
                };
        };
                if(screen == 3){
                mode3++;
                if(mode3 > 1){
                        mode3 = 0;
                };
        };
                if(screen == 4){
                ratesel++;
                if(ratesel > 1){
                        ratesel = 0;
                };
        };
                if(screen == 5){
                dataSel++;
                if(dataSel > 1){
                        dataSel = 0;
                };
        };
                if(screen == 6){
                mode6++;
                if(mode6 > 1){
                        mode6 = 0;
                };
        };
}
/*
// code to determine what occurs if switch 2 is pressed

        if ((PORTA->ISFR & MASK(SW2_POS))) {
```

```
        };

        // code to determine what occurs if switch 4 is pressed

        if ((PORTA->ISFR & MASK(SW4_POS))) {
        };
        */

        // clear status flags

        PORTA->ISFR = 0xffffffff;
        DEBUG_PORT->PCOR = MASK(DBG_ISR_POS);
}
```

```
//*********************************************************************
***
//  file:  ADC.c
//*********************************************************************
***

#include <MKL25Z4.H>
#define MASK(x) (1UL << (x))
#define ADC_POS (20)

unsigned int Future = 0;
volatile unsigned count_time=0;
volatile unsigned count=0;
volatile unsigned int flag = 0;
volatile int saveBPM[5] = {0,0,0,0,0};
extern int storedValues[1024];
extern int FutureValues[1024];
volatile unsigned int countIndex = 0;
unsigned int max = 0;
extern volatile unsigned int chosenData;
extern int DacOn;
//*********************************************************************
***
//  ADC initialization function
                                                                  Joshua
Mularczyk
//*********************************************************************
***

void Init_ADC(void){
```

```
        // Enable clock to ADC and Port E

        SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

        // Set pin signal type to analog

        PORTE->PCR[ADC_POS]&=~PORT_PCR_MUX_MASK;
        PORTE->PCR[ADC_POS]|=PORT_PCR_MUX(0);

        // ADCO alternate trigger enable, ADC0 pretriger select, ADC0
trigger select

        SIM->SOPT7 |= SIM_SOPT7_ADC0ALTTRGEN(1);
        SIM->SOPT7 |= SIM_SOPT7_ADC0PRETRGSEL(0);
        SIM->SOPT7 |= SIM_SOPT7_ADC0TRGSEL(4);

        // Normal power config, divide ratio of 1, short sample time, 12
bit sigle-ended conversion,
        // (Bus clock)/2 input

        ADC0_CFG1 = ADC_CFG1_ADLPC(0) | ADC_CFG1_ADIV(0) |
ADC_CFG1_ADLSMP(0) | ADC_CFG1_MODE(1) | ADC_CFG1_ADICLK(1);

        //hardware tigger selected, compare functin disabled, DMA disabled,
Default voltage reference pair pin

        ADC0_SC2 |= ADC_SC2_ADTRG(1) | ADC_SC2_ACFE(0) | ADC_SC2_DMAEN(0) |
ADC_SC2_REFSEL(0);

        // one conversion or sets of conversions, hardware average function
enabled, 4 samples averaged

        ADC0_SC3 |= ADC_SC3_ADCO(0) | ADC_SC3_AVGE(1) | ADC_SC3_AVGS(0);

        // Diff mode enable, input chanel select

        ADC0_SC1A |= ADC_SC1_DIFF(0);

        ADC0_SC1A &= ~ADC_SC1_ADCH_MASK;
};

//*******************************************************************
***
//  ADC initialization of inerrupt function
                             Joshua Mularczyk
//*******************************************************************
***

void Init_ADC_Interrupts(void){

        // interrupt enable
```

```
        ADC0_SC1A |= ADC_SC1_AIEN(1);


        NVIC_SetPriority(ADC0_IRQn, 128); // 0, 64, 128 or 192
        NVIC_ClearPendingIRQ(ADC0_IRQn);
        NVIC_EnableIRQ(ADC0_IRQn);

};

//**********************************************************************
***
//  ADC IRQ initialization

        Joshua Mularczyk
//**********************************************************************
***

void ADC0_IRQHandler(void){

        NVIC_ClearPendingIRQ(ADC0_IRQn);
        unsigned int analogValue;
        unsigned int high;
        unsigned int low;
        // Read result from ADC

        analogValue =  ADC0_RA;

        //read analog value directly to dac when dac mode is enabled

        if (DacOn == 1){

        DAC0->DAT[0].DATL = DAC_DATL_DATA0(analogValue);
        DAC0->DAT[0].DATH = DAC_DATH_DATA1(analogValue >> 8);

        ADC0_SC1A |= ADC_SC1_AIEN(0);
        ADC0_SC1A |= ADC_SC1_AIEN(1);

        };

        //Determine the amplitude of our waveform

        if (analogValue > max){
                max = analogValue;
        };

        high = 0.8*(max);
        low = 0.7*(max);

// Storing the input from ADC directly into an array to be called for
later

        storedValues[count] = analogValue;
        count++;
        if(count > 1023){
```

```
        count = 0;
        };


// Stores the input from the ADC into an array once a button in mode 3 has
been pressed

        while (Future == 1){
                              FutureValues[count] = analogValue;
                              count++;
                              if(count > chosenData){
                                     Future = 0;
                              };
                      }

// Do code here for accessing previously saved data at the press of button
in mode 2

        count_time++;

        if((analogValue > high)&&(flag == 0)){
               saveBPM[countIndex] = count_time;
               count_time = 0;
               countIndex++;
               if (countIndex == 5){
                      countIndex = 0;
               };
               flag = 1;
        };

        if((analogValue < low)&&(flag == 1)){
               flag = 0;
        };
};
```

```
//**************************************************************************
***
//  file:  DAC.c
//**************************************************************************
***
```

```
#include <MKL25Z4.H>
#define MASK(x) (1UL << (x))
#define DAC_POS (30)

//**********************************************************************
***
//  DAC initialization function
                                                                  Joshua
Mularczyk
//**********************************************************************
***

void Init_DAC(void){

        // Enable clock to Dac and Port E

        SIM->SCGC6 |= SIM_SCGC6_DAC0_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

        // Set pin signal type to analog

        PORTE->PCR[DAC_POS]&=~PORT_PCR_MUX_MASK;
        PORTE->PCR[DAC_POS]|=PORT_PCR_MUX(0);

        //Disable buffer mode

        DAC0->C1 = 0;
        DAC0->C2 = 0;

        //Enable DAC, select VDDA as reference voltage

        DAC0->C0 = DAC_C0_DACEN_MASK | DAC_C0_DACRFS_MASK;
}
```

```
//**********************************************************************
***
//   File:   debug_signals.c
//**********************************************************************
***

// #include "gpio_defs.h"
#include <MKL25Z4.h>

#define MASK(x) (1UL << (x))
```

```
// Debug status bits
#define DBG_ISR_POS (3)
#define DBG_MAIN_POS (1)
#define DEBUG_PORT PTD

//**********************************************************************
***
//  debug initialization function

	Larry Aamodt
//**********************************************************************
***

void init_debug_signals(void) {
	SIM->SCGC5 |=  SIM_SCGC5_PORTD_MASK; /* enable clock for port B */

	/* Select GPIO for pins connected to debug signals*/
	PORTD->PCR[DBG_ISR_POS] |= PORT_PCR_MUX(1);
	PORTD->PCR[DBG_MAIN_POS] |= PORT_PCR_MUX(1);

	/* Set bits to outputs */
	PTD->PDDR |= MASK(DBG_ISR_POS) | MASK(DBG_MAIN_POS);

	/* Clear output signals initially */
	PTD->PCOR |= MASK(DBG_ISR_POS) | MASK(DBG_MAIN_POS);
}
```

```
//**********************************************************************
***
//  file:  EKG_Functions
//**********************************************************************
***

#include <MKL25Z4.H>
#include <stdint.h>
#define MASK(x) (1UL << (x))

extern int rate;
extern int datas;
extern volatile int saveBPM[5];
extern volatile unsigned int sampleRate;
volatile unsigned int chosenData = 255;
volatile unsigned int origPit = 24000;
```

```
/*-------------------------------------------------------------------------
----
   functions
 *------------------------------------------------------------------------
----*/
void LCD_command(uint32_t command);       // use to send LCD commands
void LCD_send_data(uint32_t data);        // use to send one char to
screen
void setRateDisplay(void);
void dataSelect(void);
void display_string(char string[]);
void outputPrev(void);
void outputFuture(void);
void ekgMode(void);
void PIT_change(uint32_t period);

//************************************************************************
***
//  Function to set up menu to choose rates
                              Joshua Mularczyk
//************************************************************************
***

void setRateDisplay(void){
       switch (rate){
                      case 1:
                             LCD_command(0x01);
                             display_string("50");
                             LCD_command(0xC0);
                             display_string("Hz      ");
                             sampleRate = origPit*2.9;
                             PIT_change(sampleRate);

                      break;
                      case 2:
                             LCD_command(0x01);
                             display_string("100");
                             LCD_command(0xC0);
                             display_string("Hz      ");
                      sampleRate = origPit*2.8;
                             PIT_change(sampleRate);
                      break;
                      case 3:
                             LCD_command(0x01);
                             display_string("200");
                             LCD_command(0xC0);
                             display_string("Hz      ");
                      sampleRate = origPit*2.6;
                             PIT_change(sampleRate);
                      break;
                      case 4:
                             LCD_command(0x01);
                             display_string("300");
```

```
        LCD_command(0xC0);
        display_string("Hz      ");
sampleRate = origPit*2.4;
        PIT_change(sampleRate);
break;
case 5:
        LCD_command(0x01);
        display_string("400");
        LCD_command(0xC0);
        display_string("Hz      ");
sampleRate = origPit*2.2;
        PIT_change(sampleRate);
break;
case 6:
        LCD_command(0x01);
        display_string("500");
        LCD_command(0xC0);
        display_string("Hz      ");
sampleRate = origPit*2;
        PIT_change(sampleRate);
break;
case 7:
        LCD_command(0x01);
        display_string("600");
        LCD_command(0xC0);
        display_string("Hz      ");
sampleRate = origPit*1.8;
        PIT_change(sampleRate);
break;
case 8:
        LCD_command(0x01);
        display_string("700");
        LCD_command(0xC0);
        display_string("Hz      ");
sampleRate = origPit*1.6;
        PIT_change(sampleRate);
break;
case 9:
        LCD_command(0x01);
        display_string("800");
        LCD_command(0xC0);
        display_string("Hz      ");
sampleRate = origPit*1.4;
        PIT_change(sampleRate);
break;
case 10:
        LCD_command(0x01);
        display_string("900");
        LCD_command(0xC0);
        display_string("Hz      ");
sampleRate = origPit*1.2;
        PIT_change(sampleRate);
break;
case 11:
```

```
                        LCD_command(0x01);
                        display_string("1000");
                        LCD_command(0xC0);
                        display_string("Hz      ");
                sampleRate = origPit;
                        PIT_change(sampleRate);
                break;
        }
    };

//**********************************************************************
***
//  Function to set up data menu
                                                      Joshua Mularczyk
//**********************************************************************
***

void dataSelect(void){
    switch (datas){
                case 1:
                        LCD_command(0x01);
                        display_string("128");
                        LCD_command(0xC0);
                        display_string("Points  ");
                        chosenData = 127;
                break;
                case 2:
                        LCD_command(0x01);
                        display_string("256");
                        LCD_command(0xC0);
                        display_string("Points  ");
                        chosenData = 255;
                break;
                case 3:
                        LCD_command(0x01);
                        display_string("512");
                        LCD_command(0xC0);
                        display_string("Points  ");
                        chosenData = 511;
                break;
                case 4:
                        LCD_command(0x01);
                        display_string("1024");
                        LCD_command(0xC0);
                        display_string("Points  ");
                        chosenData = 1023;
                break;

        }
    };

//**********************************************************************
***
```

```
//  Function to display EKG BPM
                                                                   Joshua
Mularczyk
//*********************************************************************
***

void ekgMode(void){
              int    averageBPM =  ((saveBPM[0] + saveBPM[1] + saveBPM[2]
+ saveBPM[3] + saveBPM[4])/5);
              averageBPM = 60000/averageBPM;
              LCD_command(0xC0);
              LCD_send_data((averageBPM/100)+48);
              LCD_send_data(((averageBPM/10)%10)+48);
              LCD_send_data((averageBPM%10)+48);
              display_string(" BPM ");
};
void outputPrev(void){
//code to save the array to mem
};
void outputFuture(void){
//code to save the array to mem
};
```

```
/*---------------------------------------------------------------------
----*/
/*  LCD initialization and data transfer routines
*/
/*       4-bit bus version
*/
/*  Filename: lcd_lib_4bit_20b.c
*/
/*  Author:   Larry Aamodt
*/
/*  Version:  1/25/19 written
*/
/*            1/28/19 updated
*/
/*            1/29/20 updated for wtr 2020 project
*/
/*            2/12/20 revised LCD control signal port & pins
*/
/*            3/10/21 added defines for cursor commands
*/
```

```
/*             1/19/22 updated comments re clock turn on. No code changes
*/
/*             3/15/22 updated by Joshua Mularczyk
*/
/*  Compiler: Keil uVision5
*/
/*  Hardware: NXP Freedom board & a 2x8 or 2x16 LCD display w/parallel
interfc*/
/*             PORT B  bits 1,2,3 used for control
*/
/*             PORT C  bits 0,1,2,3 used for data
*/
/*  Software note:  A software loop is used for time delay generation
*/
/*         Port B and C clocks must be turned on before calling these
routines */
/*  Function use:   call LCD_command to move the cursor, clear screen,
etc.   */
/*                  call LCD_send_data to send one ASCII character code
*/
/*---------------------------------------------------------------------------
----*/
#include <stdint.h>
#include <MKL25Z4.h>

#define MASK(x) (1UL << (x))
#define LCD_EN              0x00000002    // PTB1 LCD enable
#define LCD_RW              0x00000004    // PTB2 LCD read/write
#define LCD_RS              0x00000008    // PTB3 LCD RS
#define LCD_LOW4_MASK       0x0000000F    // low 4 bits of a command
#define LCD_UPPER4_MASK     0x000000F0    // upper 4 bits of a command
#define LCD_MASK            0x0000000F    // PortC bits 0-3
#define LCD_DATA_PINS       0x0000000F    // PortC LCD data pins
#define LCD_CNTRL_PINS      0x0000000E    // PortB LCD control pins

#define clear_screen 0x01
#define cursor_left 0x10
#define cursor_right 0x14
#define cursor_line1 0x80
#define cursor_line2 0xC0


/*---------------------------------------------------------------------------
----
  LCD functions
 *--------------------------------------------------------------------------
----*/
void LCD_command(uint32_t command);       // use to send LCD commands
void LCD_send_data(uint32_t data);        // use to send one char to
screen
void delayMs(uint32_t n);
void pulse_the_LCD_enable(void);
void LCD_init(void);
void converttohex(uint32_t input);
```

```
uint32_t converttoASCII(uint32_t input);


/*----------------------------------------------------------------------------
----
  Initialize the LCD in 4-bit bus mode.         L.Aamodt
 *----------------------------------------------------------------------------
----*/
void LCD_init(void)
{
  // Note: you need to turn on Port B and C clocks prior to calling this
routine

       uint32_t k;

     // First set up bits in GPIO Port C used by the LCD
       for (k=0; k<4; k++) {                   // make ports GPIO
       PORTC->PCR[k] &= ~PORT_PCR_MUX_MASK;    // 4 LCD data bits
       PORTC->PCR[k] |= PORT_PCR_MUX(1);
       }

       for (k=1; k<4; k++) {                    // make ports GPIO
       PORTB->PCR[k] &= ~PORT_PCR_MUX_MASK;     //   LCD control 3-bits
       PORTB->PCR[k] |= PORT_PCR_MUX(1);
       }

       PTC->PDDR |= LCD_DATA_PINS;              // set ports to output
       PTB->PDDR |= LCD_CNTRL_PINS;

       PTB->PCOR = LCD_RW | LCD_RS | LCD_EN;    // clear R/W, RS, and EN
              // Now initialize the LCD itself
       delayMs(00);
       PTC->PCOR = LCD_MASK;                    // clear output data bits
to 0
       PTC->PSOR = (0x3);                    // put a wake-up value on bus
       delayMs(10);
       pulse_the_LCD_enable();
       delayMs(1);
       pulse_the_LCD_enable();
       delayMs(1);
       pulse_the_LCD_enable();
       delayMs(1);
       PTC->PCOR = LCD_MASK;                    // clear output data bits
to 0
       PTC->PSOR = (0x2);                    // initialize to 4-bit bus
mode
       delayMs(10);
       pulse_the_LCD_enable();
       LCD_command(0x28);                       // Set to 4-bit/2-
line/5x7pixels
       LCD_command(0x10);                       //
       LCD_command(0x0F);                       // Display on, cursor on
and blink
       LCD_command(0x06);                       //
```

```
}
/*----------------------------------------------------------------------
----
  Send a command to the LCD                     L.Aamodt
 *----------------------------------------------------------------------
----*/
void LCD_command(uint32_t command)
{
       PTB->PCOR = LCD_RW | LCD_RS | LCD_EN;    // clear R/W, RS, and EN
       PTC->PCOR = LCD_MASK;                    // clear output data bits
to 0
       PTC->PSOR = (command & LCD_UPPER4_MASK)>>4; // output upper 4 bits
of command
       pulse_the_LCD_enable();
       PTC->PCOR = LCD_MASK;                    // clear output data bits
       PTC->PSOR = (command & LCD_LOW4_MASK);    // output lower 4 bits
       pulse_the_LCD_enable();
       if (command < 4)
              delayMs(3);                            // command 1 and 2
need 1.64ms
       else
              delayMs(1);                            // all others 40us
}
/*----------------------------------------------------------------------
----
   Pulse the LCD enable line                    L.Aamodt
 *----------------------------------------------------------------------
----*/
void pulse_the_LCD_enable(void)
{
       PTB->PSOR = LCD_EN;                      // assert enable
       delayMs(1);
       PTB->PCOR = LCD_EN;                      // de-assert enable
}

/*----------------------------------------------------------------------
----
   Send data (one character, using ASCII code) to the LCD      L.Aamodt
 *----------------------------------------------------------------------
----*/
void LCD_send_data(uint32_t data)
{
       PTB->PCOR = LCD_RW | LCD_EN;             // clear R/W, RS, and EN
       PTB->PSOR = LCD_RS;                      // set RS high
       PTC->PCOR = LCD_MASK;                    // clear output data bits
to 0
       PTC->PSOR = (data & LCD_UPPER4_MASK)>>4;   // output upper 4 bits
of command
       pulse_the_LCD_enable();
       PTC->PCOR = LCD_MASK;                    // clear output data bits
       PTC->PSOR = (data & LCD_LOW4_MASK);  // output lower 4 bits
       pulse_the_LCD_enable();
}
```

```
/*------------------------------------------------------------------------
----
  Delay, used with the LCD routines.  Delay specified in units of
milliseconds
      The inner loop max count should be 3500 for 20.97 MHZ system clock
rate
      or 8000 for 48 MHZ system clock rate                      L.Aamodt
 *-----------------------------------------------------------------------
----*/
void delayMs(uint32_t n)
{
uint32_t i;
uint32_t j;
for(i=0; i < n; i++)
              for(j=0; j < 3500; j++) {}
}
//**********************************************************************
***
//  Function to convert a binary input to ASCII
                      Joshua Mularczyk
//**********************************************************************
***
uint32_t converttoASCII(uint32_t input){
if (input < 10){
       input = input + 48;
}
else {
       input = input + 55;
}
return input;
}


//**********************************************************************
***
//  Function to convert to Hexadecimal
                                            Joshua Mularczyk
//**********************************************************************
***
void converttohex(uint32_t input) {

uint32_t value1, value2, value3, value4;


value1 = input%16;
input = input/16;


value2 = input%16;
input = input/16;


value3 = input%16;
input = input/16;
```

```
value4 = input%16;

// converting the hex representation to ASCII

value1 = converttoASCII(value1);
value2 = converttoASCII(value2);
value3 = converttoASCII(value3);
value4 = converttoASCII(value4);

//printing out the values

LCD_send_data(value4);
LCD_send_data(value3);
LCD_send_data(value2);
LCD_send_data(value1);
}

//**********************************************************************
***
//  Function to display a string on the LCD
                             Mikhail Beresnev
//**********************************************************************
***
void display_string(char string[]){

       int i=0;
       while (string[i] != 0){
              LCD_send_data(string[i]);
              i++;
       }
}
```

```
//**********************************************************************
***
//  file:  Pit.c
//**********************************************************************
***

#include "MKL25Z4.h"

#define MASK(x) (1UL << (x))
#define DEBUG_PORT PTD
```

```
//**********************************************************************
***
//  Function to initialize the PIT
//**********************************************************************
***
void Init_PIT(uint32_t period) {

// Enable clock to PIT module

SIM->SCGC6 |= SIM_SCGC6_PIT_MASK;

// Enable module, freeze timers in debug mode

PIT->MCR &= ~PIT_MCR_MDIS_MASK;
PIT->MCR |= PIT_MCR_FRZ_MASK;

// Initialize PIT0 to count down from argument

PIT->CHANNEL[0].LDVAL = PIT_LDVAL_TSV(period);

// No chaining

PIT->CHANNEL[0].TCTRL &= PIT_TCTRL_CHN_MASK;

// Generate interrupts

PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TIE_MASK;

/* Enable Interrupts */

NVIC_SetPriority(PIT_IRQn, 128); // 0, 64, 128 or 192
NVIC_ClearPendingIRQ(PIT_IRQn);
NVIC_EnableIRQ(PIT_IRQn);
}
//**********************************************************************
***
//  Function to start PIT
//**********************************************************************
***
void Start_PIT(void) {

// Enable counter

PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TEN_MASK;
}
//**********************************************************************
***
//  Function to stop PIT
//**********************************************************************
***
void Stop_PIT(void) {
```

```
// Enable counter

PIT->CHANNEL[0].TCTRL &= ~PIT_TCTRL_TEN_MASK;
}
//*********************************************************************
***
//  Function for Pit interrupt handler
//*********************************************************************
***
void PIT_IRQHandler(void) {

//clear pending IRQ

NVIC_ClearPendingIRQ(PIT_IRQn);

// check to see which channel triggered interrupt

if (PIT->CHANNEL[0].TFLG & PIT_TFLG_TIF_MASK) {

// clear status flag for timer channel 0

PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;

        // Do ISR work
        //on
        //     DEBUG_PORT->PSOR = MASK(3);
        //off
        //DEBUG_PORT->PCOR = MASK(3);


} else if (PIT->CHANNEL[1].TFLG & PIT_TFLG_TIF_MASK) {

// clear status flag for timer channel 1

PIT->CHANNEL[1].TFLG &= PIT_TFLG_TIF_MASK;
}
}
//*********************************************************************
***
//  Function to change input period to user selected valueJoshua Mularczyk
//*********************************************************************
***
void PIT_change(uint32_t period){
        PIT->CHANNEL[0].LDVAL = PIT_LDVAL_TSV(period);
};
```