Code for Lab 0/1

```
/*****************************************************************************
 * Lab1_BF609_Core0.cpp
 *****************************************************************************/

#include <sys/platform.h>
#include <sys/adi_core.h>
#include <ccblkfn.h>
#include "adi_initialize.h"
#include "Lab1_BF609_Core0.h"

/**
 * If you want to use command program arguments, then place them in the following
string.
 */
char __argv_string[] = "";

int main(int argc, char *argv[])
{
	/**
	 * Initialize managed drivers and/or services that have been added to
	 * the project.
	 * @return zero on success
	 */
	adi_initComponents();

	/**
	 * The default startup code does not include any functionality to allow
	 * core 0 to enable core 1. A convenient way to enable
	 * core 1 is to use the adi_core_enable function.
	 */
	adi_core_enable(ADI_CORE_1);

	/* Begin adding your custom code here */


	#ifdef __ADSPBF533__
		printf("Start BF533 Lab 1\n");
		Start_Lab1();
	#endif

	#ifdef __ADSPBF609__
		printf("Start BF609 Lab 1\n");
		Start_Lab1();
	#endif

return 0;
}
```

```
/*****************************************************************************
 * Lab1_BF609_Core0.h
 *****************************************************************************/

#ifndef __LAB1_BF609_CORE0_H__
#define __LAB1_BF609_CORE0_H__

/* Add your custom header content here */

#include <stdio.h>
#include "../../ENCM511_SpecificFiles/ENCM511_include/FrontPanel_LED_Switches.h"
#include "../../ENCM511_SpecificFiles/ENCM511_include/ADSP_BF609_Utilities_Library.h"
#include "../../ENCM511_SpecificFiles/ENCM511_include/REB_GPIO_Input_Library.h"
#include "../../ENCM511_SpecificFiles/ENCM511_include/REB_GPIO_Output_Library.h"

#define GARBAGE_VALUE static_cast<unsigned char>(-1) //The garbage value is unsigned
-1
#define GARBAGE_VALUE1 static_cast<unsigned short int>(-1) //The Garbage value is
unsigned short int -1
#define MASK_KEEP_BITS_11_TO_8 0x0f00 //This is masking the bit values so we can only
have PF8-11
#define MaskBits15to12And7to0 0xf0ff
#define MaskBits11to0 0x0fff

//These variables are all declared and initialized to be false to begin until the
respective function is called
//to initialize the respective equipment
static bool My_Init_SwitchInterface_Done = false;
static bool My_Init_LEDInterface_Done = false;
static bool My_Init_GPIO_REB_Input_Done = false;
static bool My_Init_GPIO_REB_Output_Done = false;
static bool My_Init_GPIO_REB_Done = false;
static bool reset = false; //This variable is going to reset the operations back to
choosing between Lab1 or Lab0

//Initialization Prototypes
void My_Init_LEDInterface(void);
void My_Init_SwitchInterface(void);

void My_Init_GPIO_REB_Input(void);
void My_Init_GPIO_REB_Output(void);
void My_Init_GPIO_REB_InputCpp(void); //Our own function for initializing the REB
void My_Init_GPIO_REB_OutputCpp(void); //Our own function for initializing the REB

//Read Prototypes
unsigned char My_ReadSwitches(void);
unsigned short int My_Read_REB_Switches(void);
unsigned short int My_Read_GPIO_REB_Input(void);

//Write Prototypes
void My_WriteLED(unsigned char);
void My_Write_REB_LED(unsigned short int);
void My_Write_GPIO_REB_Output(unsigned short int);

//Other Prototypes for Lab 1
```

```
void Start_Lab0(void);
void Start_Lab1(void);
void Start_PreLab1(void);
void charToBinary(unsigned char, unsigned char*);
void WaitTillSwitchREB1PressedAndReleased(void);
void WaitTillSwitchREB2PressedAndReleased(void);
void WaitTillSwitchREB3PressedAndReleased(void);
void WaitTillSwitch1PressedAndReleased(void);
void WaitTillSwitch2PressedAndReleased(void);
void WaitTillSwitch3PressedAndReleased(void);

//extern "C" means that you are declaring these functions and they can be used in a
different file as the compiler knows they are in an external file
//Pretty much they act in this case as global functions and are declared below

extern "C" unsigned long long int ReadProcessorCyclesASM(void);
extern "C" void My_Write_GPIO_REB_OutputASM(unsigned short int);
extern "C" unsigned short int My_Read_GPIO_REB_InputASM(void);
extern "C" void My_Init_GPIO_REB_InputASM(void);
extern "C" void My_Init_GPIO_REB_OutputASM(void);


#endif /* __LAB1_BF609_CORE0_H__ */
```

```
/*
 * Lab1_GeneralCode.cpp
 *
 *  Created on: Oct 8, 2019
 *      Author: Aidan and Michele
 */

#include <sys/platform.h>
#include "adi_initialize.h"
#include "Lab1_BF609_Core0.h"

void Start_Lab1(void) //Code stub for Start Lab1
{
        printf("Here in Start_Lab1\n"); //This is declaring it is the start of Lab 1

        My_Init_SwitchInterface();  //This function is initiating the switches on the panel
        My_Init_GPIO_REB_Input(); //This function is initiating the switches on the board
        My_Init_LEDInterface();  //This function is initiating the LEDS on the panel
        My_Init_GPIO_REB_Output(); //This function is initiating the LEDs on the board

        printf("Please Press Switch 1 to Begin the Lab\n"); //Pressing Switch 1 will initiate the Start of the Lab

        WaitTillSwitchREB1PressedAndReleased(); //This function is in place to make sure that switch 1 was pressed and then released

        int count = 0; //Creating a counter value
        int i = 0;  //Creating a counter variable for the for loop
        unsigned short int hardWareArray [100]; //This is the hardWareArray that will hold the SW values that are pressed
        unsigned char switchValue = 0; //Creating a value to hold the switch Value
        unsigned short int switchREBValue = 0; //Creating a value to hold the switch REB Value

        //Switch 1 has a value of: 0x01
        //Switch 2 has a value of: 0x02
        //Switch 3 has a value of: 0x04
        //Switch 4 has a value of: 0x08

        unsigned long long int initialTime; //This variable will hold the initial Time
        unsigned long long int WaitTime = 480000000; //The wait time was selected to be 1 second which is equal to 480000000 processor cycles
        unsigned long long int time; //This variable will hold the time

        while(1)
        {
                reset = false;
                printf("Press Switch 1 for Lab0, Press Switch 2 for PreLab1 and Press Switch 3 for Lab 1 \n");
                switchValue = My_ReadSwitches();
                switchREBValue = My_Read_REB_Switches();

                if(switchValue == 0x1 || switchREBValue == 0x1)
                {
                    Start_Lab0();
                }
                else if(switchValue == 0x2 || switchREBValue == 0x2)
                {
                    Start_PreLab1();
                }
                else if(switchValue == 0x4 || switchREBValue == 0x4)
                {
                    WaitTillSwitchREB3PressedAndReleased();
                    while(!reset)
                    {
                        printf("Starting HardWare Fill \n");
                        i = 0;
                        count = 0;
                        while(!reset) //This loop is accumulating all the switches
pressed and recorded to fill the hardWareArray
                        {
                            switchREBValue = My_Read_REB_Switches(); //This is
reading the switch value pressed

                            switchValue = My_ReadSwitches();  //This is reading
if a front panel switch was pressed

                            if (switchValue == 0x10)
                            {
                                reset = true;
                            }

                            if (switchValue == 0x01) //0x01 is switch 1 on front
panel which when pressed will record the value
                            {
                                WaitTillSwitch1PressedAndReleased();
                                hardWareArray[i] = switchREBValue; //Filling
the hardWareArray with the switch value
                                printf("Filling HardwareArray \n");
                                count++;
                                i++;
                            }
                            else if (switchValue == 0x08) //0x08 is switch 4 on
front panel which when pressed will record the value
                            {
                                count = 0;
                                break;
                            }

                            if(i > 99)
                            {
                                printf("The hardWareArray has now been filled
\n");

                                count = 0;
                                break;
                            }
                        }
                        while(!reset)
                        {
                            switchREBValue = My_Read_REB_Switches();
                            switchValue = My_ReadSwitches();

                            if (switchValue == 0x10)
                            {
                                reset = true;
                            }

                            initialTime = ReadProcessorCyclesASM();
                            My_Write_REB_LED(hardWareArray[count]);

                            count = count + 1; //incrementing the counter by 1

                            if(switchREBValue == 1)
                            {
                                WaitTillSwitchREB1PressedAndReleased();
                                WaitTime = WaitTime / 2; //decreasing the
time to wait
                                if(WaitTime == 1)
                                {
                                    WaitTime = WaitTime * 2; //This is here
to make sure the wait time does not get too fast
                                }
                            }
                            else if(switchREBValue == 2)
                            {
                                WaitTillSwitchREB2PressedAndReleased();
                                WaitTime = WaitTime * 2; //increasing the
time to wait
                            }

                            time = ReadProcessorCyclesASM();
                            while(time < initialTime + WaitTime)
                            {
                                time = ReadProcessorCyclesASM();
                            }

                            //This is making sure the count does not go past the
amount of indexes in the hardWare array
                            if(count > i)
                            {
                                count = 0;
                            }
                        }

                    }
                }
            }
}

unsigned char My_ReadSwitches(void) //This function is reading the switches from the
panel
{
            //printf("Stub for My_ReadSwitches()\n");

            #ifdef __ADSPBF609__
                    if (My_Init_SwitchInterface_Done == false)
                    {
                        printf("Switch hardware not ready \n");
                        return GARBAGE_VALUE;
                    }

                    FRONTPANEL_SWITCH_5BIT_VALUE activeLowValues =
Read_GPIO_FrontPanelSwitches();
                    FRONTPANEL_SWITCH_5BIT_VALUE activeHighValues = ~activeLowValues;


                    #define MASK_KEEP_LOWER_FIVE_BITS 0x1F // use bit-wise
                    FRONTPANEL_SWITCH_5BIT_VALUE wantedSwitchValueActiveHigh =
activeHighValues & MASK_KEEP_LOWER_FIVE_BITS;
                    return wantedSwitchValueActiveHigh;
            #else
                return 0x55;
            #endif

}

unsigned short int My_Read_REB_Switches(void)
{
                if(My_Init_GPIO_REB_Input_Done == false) //My_Init_GPIO_REB_Input_Done
for his function
                {
                    printf("Switch hardware not ready \n");
                    return GARBAGE_VALUE1;
                }

                REB_BITS16 wantedSwitchOnBoardValueActiveHigh =
My_Read_GPIO_REB_InputASM(); //The board is active high
                return wantedSwitchOnBoardValueActiveHigh;
}

void My_Write_REB_LED(unsigned short int LEDValue) //This function is writing the
values for the REB LEDs
{
            //printf("Stub for My_Write_REB_LED() \n");

            if (My_Init_GPIO_REB_Output_Done == false) // My_Init_GPIO_REB_Output_Done for
his function
            {
                printf("LED hardware not ready \n");
                return;
            }

            #ifdef __ADSPBF609__
                My_Write_GPIO_REB_OutputASM(LEDValue);
            #endif
}
```

```
void My_WriteLED(unsigned char neededLEDValue) //This function is writing the values
to be displayed by the LEDs
{
    //printf("Stub for My_WriteLED() \n");

    if (My_Init_LEDInterface_Done == false)
    {
        printf("LED hardware not ready \n");
        return;
    }

#ifdef __ADSPBF609__
    Write_GPIO_FrontPanelLEDS(neededLEDValue); //Writing the value to the panel of
LEDs

#else //This is for the 533 emulator
    //Char array holding the values to print out
    unsigned char binaryArray[9];
    charToBinary(neededLEDValue, binaryArray); //Function converting the char to
binary using the array holding the values
    printf("LED value - decimal %3d; hex 0x%2x; bit pattern %s \n",
neededLEDValue, neededLEDValue, &binaryArray);
#endif

}

void My_Init_SwitchInterface(void) //This function is initializing the Switches on
the Panel
{
    //printf("Stub for My_Init_SwitchInterface() \n");
    My_Init_SwitchInterface_Done = true;

#ifdef __ADSPBF609__
    Init_GPIO_FrontPanelSwitches();
#endif
}

void My_Init_LEDInterface(void) //This function is initializing the LEDs on the Panel
{
    //printf("Stub for My_Init_LEDInterface() \n");
    My_Init_LEDInterface_Done = true;

    #ifdef __ADSPBF609__
        Init_GPIO_FrontPanelLEDS();
    #endif

}

void My_Init_GPIO_REB_Input(void) //This function is initializing the switches on the
Board
{
    //printf("Stub for My_Init_GPIO_REB_Input() \n");
    My_Init_GPIO_REB_Input_Done = true;

    #ifdef __ADSPBF609__
```

```
#if 0
                My_Init_GPIO_REB_InputCpp();
#else
                My_Init_GPIO_REB_InputASM();
#endif

        #endif
}

void My_Init_GPIO_REB_Output(void) //This function is initializing the LEDs on the
Board
{
        //printf("Stub for My_Init_GPIO_REB_Output");
        My_Init_GPIO_REB_Output_Done = true;

        #ifdef __ADSPBF609__
#if 0
                My_Init_GPIO_REB_OutputCpp();
#else
                My_Init_GPIO_REB_OutputASM();
#endif
        #endif
}

void charToBinary(unsigned char charValue, unsigned char* array) //This is my
function to convert the char value into a binary number for the operation to display
my initials
{
        unsigned char numberValueDuplicate = charValue;
        char i;
        for (i = 7; i >= 0; i--)
        {
                if (numberValueDuplicate & 0x01)
                        array[i] = '1';
                else
                        array[i] = ' '; // change to '0' to get binary, I have the space
to properly print initials

                numberValueDuplicate = numberValueDuplicate >> 1; //Shifting the bits by
1
        }
        array[8] = 0; //To end the string with a null character
}

void WaitTillSwitchREB1PressedAndReleased() //This function is making sure that
Switch REB 1 is pressed and released
{
        unsigned short int switchValue = 0;
        while(1)
        {
                switchValue = My_Read_REB_Switches();
                if(switchValue == 0x1)
                {
                        while(1)
                        {
```

```
                switchValue = My_Read_REB_Switches();
                unsigned char bitValue = switchValue & 0x1;
                if(bitValue == 0x0)
                {
                    break;
                }
            }
            break;
        }
    }
}

void WaitTillSwitchREB2PressedAndReleased() //This function is making sure that
Switch REB 2 is pressed and released
{
    unsigned short int switchValue = 0;
    while(1)
    {
        switchValue = My_Read_REB_Switches();
        if(switchValue == 0x2)
        {
            while(1)
            {
                switchValue = My_Read_REB_Switches();
                unsigned char bitValue = switchValue & 0x2;
                if(bitValue == 0x0)
                {
                    break;
                }
            }
            break;
        }
    }
}

void WaitTillSwitchREB3PressedAndReleased() //This function is making sure that
Switch REB 3 is pressed and released
{
    unsigned short int switchValue = 0;
    while(1)
    {
        switchValue = My_Read_REB_Switches();
        if(switchValue == 0x4)
        {
            while(1)
            {
                switchValue = My_Read_REB_Switches();
                unsigned char bitValue = switchValue & 0x4;
                if(bitValue == 0x0)
                {
                    break;
                }
            }
            break;
        }
        }
}

void WaitTillSwitch1PressedAndReleased() //This function is making sure that Switch 1
is pressed and released
{
        unsigned char switchValue = 0;
        while(1)
        {
                switchValue = My_ReadSwitches();
                if(switchValue == 0x1)
                {
                        while(1)
                        {
                                switchValue = My_ReadSwitches();
                                unsigned char bitValue = switchValue & 0x1;
                                if(bitValue == 0x0)
                                {
                                        break;
                                }
                        }
                        break;
                }
        }
}

void WaitTillSwitch2PressedAndReleased() //This function is making sure that Switch 2
is pressed and released
{
        unsigned char switchValue = 0;
        while(1)
        {
                switchValue = My_ReadSwitches();
                if(switchValue == 0x2)
                {
                        while(1)
                        {
                                switchValue = My_ReadSwitches();
                                unsigned char bitValue = switchValue & 0x2;
                                if(bitValue == 0x0)
                                {
                                        break;
                                }
                        }
                        break;
                }
        }
}

void WaitTillSwitch3PressedAndReleased() //This function is making sure that Switch 3
is pressed and released
{
        unsigned char switchValue = 0;
        while(1)
        {
                switchValue = My_ReadSwitches();
                if(switchValue == 0x4)
                {
                        while(1)
                        {
                                switchValue = My_ReadSwitches();
                                unsigned char bitValue = switchValue & 0x4;
                                if(bitValue == 0x0)
                                {
                                        break;
                                }
                        }
                        break;
                }
        }
}

void Start_Lab0()
{
        printf("Here in Start_Lab0\n");
        printf("Press Switch 1\n");

        WaitTillSwitch1PressedAndReleased();
        unsigned char intials[15] = {0x00, 0xe0, 0x1c, 0x13, 0x1c, 0xe0, 0x00, 0xc0,
0x00, 0xe0, 0xc3, 0xff, 0x03, 0x00, 0xc0};
        int count = 0;
        unsigned char switchValue = 0;

        //Variables to Control time
        unsigned long long int intialTime;
        unsigned long long int WaitTime = 480000000;
        unsigned long long int time;


        while(!reset)
        {
                intialTime = ReadProcessorCyclesASM();
                My_WriteLED(intials[count]); //printing intials line by line
                count = count + 1; //incrementing the counter

                switchValue = My_ReadSwitches();

                if (switchValue == 0x10)
                {
                        reset = true;
                }

                if(switchValue == 1)
                {
                        WaitTillSwitch1PressedAndReleased();
                        WaitTime = WaitTime / 2; //decreasing the time to wait
                        if(WaitTime == 0)
                        {
                                WaitTime = 480000000;
                        }
                }
                else if(switchValue == 2)
                {
                        WaitTillSwitch2PressedAndReleased();
                        WaitTime = WaitTime * 2; //increasing the time to wait
                }

                time = ReadProcessorCyclesASM();
                while(time < intialTime + WaitTime)
                {
                        time = ReadProcessorCyclesASM();
                }

                //This is making sure the count does not go past the amount of indexes
in my intials array
                if(count == 16)
                {
                        count = 0;
                }
        }
}

void Start_PreLab1(void) //Code stub for Start Lab1
{
        printf("Here in Start_PreLab1\n"); //This is declaring it is the start of Lab
1
        printf("Please Press Switch 1 to Begin the PreLab\n"); //Pressing Switch 1
will initiate the Start of the Lab

        WaitTillSwitchREB1PressedAndReleased(); //This function is in place to make
sure that switch 1 was pressed and then released

        //The array below holds random short integer value to display the LED lights
        unsigned short int softwarearray[4] = {0x0008, 0x0004, 0x0002, 0x0001};
//Array to test the LEDs

        int count = 0; //Creating a counter value
        unsigned short int switchREBValue = 0; //Creating a value to hold the switch
REB Value
        unsigned char switchValue = 0; //Creating a value to read FP switch for the
reset

        unsigned long long int initialTime; //This variable will hold the initial Time
        unsigned long long int WaitTime = 480000000; //The wait time was selected to
be 1 second which is equal to 480000000 processor cycles
        unsigned long long int time; //This variable will hold the time

        while(!reset)
        {
                switchREBValue = My_Read_REB_Switches();
                switchValue = My_ReadSwitches();

                initialTime = ReadProcessorCyclesASM();
                My_Write_REB_LED(softwarearray[count]);
                count = count + 1; //incrementing the counter

                if (switchValue == 0x10)
                {
                        reset = true;
                }

                if(switchREBValue == 1)
                {
                        WaitTillSwitchREB1PressedAndReleased();
                        WaitTime = WaitTime / 2; //decreasing the time to wait
                        if(WaitTime == 1)
                        {
                            WaitTime = WaitTime * 2; //This is here to make sure the wait time does not get too fast
                        }
                }
                else if(switchREBValue == 2)
                {
                        WaitTillSwitchREB2PressedAndReleased();
                        WaitTime = WaitTime * 2; //increasing the time to wait
                }

                time = ReadProcessorCyclesASM();
                while(time < initialTime + WaitTime)
                {
                        time = ReadProcessorCyclesASM();
                }

                //This is making sure the count does not go past the amount of indexes in my intials array
                if(count == 4) //was 16 for initials
                {
                        count = 0;
                }
        }
}
```

Code for My Init Functions ASM

```
/*
 * My_Init_GPIO_REB_InputASM.asm
 *
 *  Created on: Oct 12, 2019
 *      Author: aidan
 */
#include <blackfin.h>

	.section L1_data;

	.section program;
	.global _My_Init_GPIO_REB_InputASM;

	#define returnValue_R0 R0
	#define MASK_KEEP_BITS_11_TO_8 0x0f00
	#define SETTING_TO_ALL_ZEROS 0x0000
	#define MASK_KEEP_BITS_15_TO_12_AND_7_TO_0 0xf0ff
	#define SETTING_BITS_11_TO_8_ALL_ONES 0x0f00

_My_Init_GPIO_REB_InputASM:
	LINK 20;

	//This code is storing the value in the port F data register into the pointer
register P0
	P0.L = lo(REG_PORTF_DATA);
	P0.H = hi(REG_PORTF_DATA);

	R1 = SETTING_TO_ALL_ZEROS;
	[P0] = R1; //This is intializing the data register with all zeros to begin
with

	R2 = MASK_KEEP_BITS_15_TO_12_AND_7_TO_0(Z);

	//This code is storing the value in the port F enabled register into the
pointer register P0
	P0.L = lo(REG_PORTF_INEN);
	P0.H = hi(REG_PORTF_INEN);

	R3 = W[P0](Z);
	R0 = R3 & R2; //This is making sure we only zero the bits 11-8
	[P0] = R0;

	R3 = W[P0](Z);

	R1 = SETTING_BITS_11_TO_8_ALL_ONES;

	R0 = R3 | R1; //This putting in the correct enabled values into the enabled
bits part
	[P0] = R0;
```

```
        //This code is storing the value in the port F polarity register into the
pointer register P0
        P0.L = lo(REG_PORTF_POL);
        P0.H = hi(REG_PORTF_POL);

        R1 = SETTING_TO_ALL_ZEROS;
        [P0] = R1;

        UNLINK;

_My_Init_GPIO_REB_InputASM.END:
        RTS;
```

```
/*
```

```
 * My_Init_GPIO_REB_OutputASM.asm
 *
 *  Created on: Oct 12, 2019
 *      Author: aidan
 */
#include <blackfin.h>

      .section L1_data;

      .section program;
      .global _My_Init_GPIO_REB_OutputASM;

      #define returnValue_R0 R0
      #define MASK_KEEP_BITS_11_TO_0 0x0fff
      #define MAKING_DIRECTION_ALL_ONES 0xf000

_My_Init_GPIO_REB_OutputASM:
      LINK 20;

      //This code is storing the value in the port F data register into the pointer
register P0
      P0.L = lo(REG_PORTF_DIR);
      P0.H = hi(REG_PORTF_DIR);

      R0 = W[P0](Z);
      R1 = MASK_KEEP_BITS_11_TO_0;
      R2 = MAKING_DIRECTION_ALL_ONES(Z);

      R0 = R0 & R1;
      R0 = R0 | R2;

      [P0] = R0;

      UNLINK;

_My_Init_GPIO_REB_OutputASM.END:
      RTS;
```

My Read and Write REB functions ASM

```
/*
 * My_Read_GPIO_REB_InputASM.asm
 *
 *  Created on: Oct 12, 2019
 *      Author: aidan
 */
#include <blackfin.h>

       .section L1_data;

       .section program;
       .global _My_Read_GPIO_REB_InputASM;

       #define returnValue_R0 R0
       #define MASK_KEEP_BITS_11_TO_8 0x0f00

_My_Read_GPIO_REB_InputASM:
       LINK 20;

       R1 = MASK_KEEP_BITS_11_TO_8; //Putting the masks into the registers

       //This code is storing the value in the port F register into the pointer
register P0
       P0.L = lo(REG_PORTF_DATA);
       P0.H = hi(REG_PORTF_DATA);

       returnValue_R0 = W[P0](Z); //Putting the value for the switches into the R0
register (this is reading the values)

       returnValue_R0 = returnValue_R0 & R1; //This is selecting only bits 11-8 which
are the input pins
       returnValue_R0 = returnValue_R0 >> 8; //Shifting the 4bit input down to the
bottom to be able to read as a switche value

       UNLINK;

_My_Read_GPIO_REB_InputASM.END:
       RTS;
```

```
/*
 * My_Write_GPIO_REB_OutputASM.asm
 *
 *  Created on: Oct 11, 2019
 *      Author: aidan
 */

#include <blackfin.h>

       .section L1_data;

       .section program;
       .global _My_Write_GPIO_REB_OutputASM;

       #define returnValue_R0 R0
       #define MaskBitValues11to0 0x0fff

_My_Write_GPIO_REB_OutputASM:
       LINK 20;

       R1 = MaskBitValues11to0; //Storing the mask value into R1 register

       //This code is storing the value in the port F register into the pointer
register P0
       P0.L = lo(REG_PORTF_DATA);
       P0.H = hi(REG_PORTF_DATA);

       R2 = W[P0](Z); //Putting the value for the port F register into the R2
register (this is reading the values)

       returnValue_R0 = returnValue_R0 << 12; //Shifting the value for the LEDs up to
the output pins

       R2 = R2 & R1; //Masking the port F register

       returnValue_R0 = returnValue_R0 | R2; //Oring the port F register with the
correct value for the LED outputs

       [P0] = returnValue_R0; //Storing theses new output values into the port F
register so it will display the correct LED orientation

       UNLINK;

_My_Write_GPIO_REB_OutputASM.END:
       RTS;
```

Code for My Init Functions CPP

```
/*
 * My_Init_GPIO_REB.cpp
 *
 *  Created on: Oct 10, 2019
 *      Author: aidan
 */
#include <blackfin.h>
#include "Lab1_BF609_Core0.h"

void My_Init_GPIO_REB_InputCpp(void)
{
        #ifdef __ADSPBF609__


                *pREG_PORTF_DATA = 0x0000; //Setting the port F data register to all 0's 
to begin with

                *pREG_PORTF_INEN = *pREG_PORTF_INEN & MaskBits15to12And7to0; //This is 
making sure we keep whatever values are in the other bits

                *pREG_PORTF_INEN = 0x0f00 | *pREG_PORTF_INEN;  //Making the enable point 
to the input pins 11-8

                *pREG_PORTF_POL = 0x0000; //Setting the port F polarity register to all 
0's to begin with

        #endif
}
```

```
/*
 * My_Init_GPIO_REB_Output.cpp
 *
 *  Created on: Oct 10, 2019
 *      Author: aidan
 */
#include <blackfin.h>
#include "Lab1_BF609_Core0.h"

void My_Init_GPIO_REB_OutputCpp(void)
{
      #ifdef __ADSPBF609__
             *pREG_PORTF_DIR = *pREG_PORTF_DIR & MaskBits11to0; //This is masking the
port F direction register to make sure we are not wiping the other bits out
             *pREG_PORTF_DIR = 0xf000 | *pREG_PORTF_DIR; //Making the direction point
to the output pins 15-12
      #endif
}
```

Code for My Read and Write REB CPP

```
/*
 * My_Write_GPIO_REB_Output.cpp
 *
 *  Created on: Oct 10, 2019
 *      Author: aidan
 */
#include <blackfin.h>
#include "Lab1_BF609_Core0.h"

void My_Write_GPIO_REB_Output(unsigned short int LEDValueToDisplay)
{
	unsigned short int TempValue;
	LEDValueToDisplay = LEDValueToDisplay << 12; //Shifting the bits by 12 to output to the Port for the LEDS as they are pins 12-15
	TempValue = *pREG_PORTF_DATA & 0x0fff; //Creating a Temporary value to hold the masked values of the portf data
	*pREG_PORTF_DATA = LEDValueToDisplay | TempValue; //Placing the LED Value into the PORTF register
}
```

```
/*
 * My_Read_GPIO_REB_Input.cpp
 *
 *  Created on: Oct 10, 2019
 *      Author: aidan
 */
#include <blackfin.h>
#include "Lab1_BF609_Core0.h"

unsigned short int My_Read_GPIO_REB_Input(void)
{
	unsigned short int switchREBValue; //Declaring a short integer (16bits)
switchREBValue
	switchREBValue = *pREG_PORTF_DATA; //This statement is storing the value from
the pointer holding port F register data
	switchREBValue = switchREBValue & MASK_KEEP_BITS_11_TO_8; //This statement is
just selecting the bits from PF8-11
	switchREBValue = switchREBValue >> 8; //Shift the bits down by 8 to get the
correct correlated switch values
	return switchREBValue;
}
```

Read Cycles ASM

```
/*
 * ReadProcessorCyclesASM.asm
 *
 *  Created on: Sep 26, 2019
 *      Author: aidan
 */
       .section L1_data;

       .section program;
       .global _ReadProcessorCyclesASM;

       #define returnValue_R0 R0
       #define returnValue_R1 R1

_ReadProcessorCyclesASM:
       LINK 20;

       returnValue_R0 = CYCLES;
       returnValue_R1 = CYCLES2;

       UNLINK;

_ReadProcessorCyclesASM.END:
       RTS;
```