ENCM 511 - Assignment 2

Kyle Derby MacInnis - 10053959

December 7, 2014

Abstract

The following report covers the necessary material required for laboratory 3, and excepting the parts which have been taken from laboratory materials provided in class, all work contained in this report is our own. The Smith 20 Minute Rule was adhered to, and we have not copied any work or received help which I did not then build off of ourself.

The purpose of this assignment was to expand on the design of the coffee pot device, such that it utilizes a Watchdog Timer and adjusts automatically for changes in temperature, water level, and overal progress. The methods of implementing these features will be covered in more detail within the report.

Kyle Derby MacInnis | Date

Note to Marker:

For this assignment I chose to write my own library for many of the functions from the uTTCOS, and as a result I have chosen to include myBlackfin.h as my library header file. There will be no use of any of the uTTCOS functions except what I have rewritten. Please see Appendix A for myBlackfin.c and myBlackfin.h.

Section 1 - Controlling Multiple Coffee Pots at Once

The main purpose of this assignment was learn to use interrupts and figure out how to automate and control the coffee pot via their use. Because of this fact, I chose to control multiple coffee pots using the interrupts rather than writing a separate function to do it using methods similar to Assignment 1. I felt nothing could be learned from rewriting my assignment 1 code to adapt it for two coffee pots. In order to implement it, the following would need to be done:

- Rewrite *PrepareTheCoffeePot()* to allow for two coffee pots in the arguments.
- Adjust the heater, water, and coffee functions to allow for two coffee pots.
- Adjust the constant values empirically (trial and error) in order to loop the functions the appropriate times.

Once again I would like to state that I chose not to do this portion of the assignment as I felt it redundant and obselete compared to the code involving the interrupts as shown in section 4. Please see section 4 for overview of my code which controls multiple coffee pots at once.

Section 2 - Controlling Coffee Pot with Switches

As the main purpose of this assignment was to get a better understanding of interrupts, I chose to forgo this portion of the assignment as it is expanded on later in the report under section 4. The implementation I would have done, had I not already implemented the functionality using interrupts would be to setup a super loop very similar to the one from Lab 1, and within it, I would have a periodic check for which switches were pressed. This would involved initializing the switches to be level based rather than edge based as is needed for interrupts. Alongside these periodic checks there would be periodic flashing of LED5, and the loop would update periodically as well. The only major changes would need to be done to the super loop as found in section 4, and a small addition of the Super Loop struct defining my periodic intervals as well as a *Tic()* function. Once again, I wish to state that I forgo this part of the assignment because I felt that I had an understanding of the material, and instead wished to focus on the new material regarding interrupts.

Section 3 - Testing Software and Hardware Interrupts

Overview

In order to test my interrupt functions were working correctly I created a new EUNIT project and linked it to my Blackfin library, and included myBlackfin.h.

Interrupt Testing - A common ISR Function

I chose to write a single ISR to be used to test both the Core Timer interrupt, and the GPIO PF interrupt. The ISR code is as follows in my Test file:

```
// Count
volatile unsigned char count =  0x0;

// ISR for Core Timer
#pragma interrupt
void TestISR(void)
{
    // If less than
    if(count<0x3F)
```

```
        count++;
    else
        count = 0;

    // Write count to LEDS
    Write_LED(count);

    // Clear Switch Interrupt (GPIO)
    *pFIO_FLAG_D = 0x0;

    // Core Timer Reset
    // No Need to ---
}
```

As can be seen, the ISR is designed to increment a counter and then write the current value onto the Blackfin LEDs. At the end, the ISR resets the PF interupt values by setting the FIO_FLAG_D register to 0x0 to clear out the bits.

Core Timer Interrupt Test Function

The Test for my Core Timer Interrupt is as follows:

```
// Test Core Timer Interrupts
TEST(CoreTimer)
{

    // Setup Interrupt with ISR (Core Timer)
    #define TEST_PERIOD 0x02FFFFFF
    Setup_CoreTimer_Int(&TestISR, TEST_PERIOD);
    // Start Core Timer Interrupt
    Start_CoreTimer_Int();

    printf("\n\nBeginning to run test for the Core Timer Interrupt.\n");
    printf("The First part of this test will look at initializing an interrupt.\n");
    printf("---------- Let the Test Begin:\n");

    char readChar[250];

    printf("Look at the LEDS.\n Are they flashing? (y or n):\n");

    while(count < 0x3F)
    {
    }
    gets(readChar);

    // Check for Correct
    CHECK_EQUAL((char)(*readChar),'y');

    printf("Test #1 is now finished.\n");

    // Stop Interrupt
    Stop_CoreTimer_Int();

}
```

Essentially, the test starts the interrupt and then waits until the interrupt counts up to 0x3F, at which point it then checks for the user input to see if the counter was in fact working.

GPIO Interrupt Test Function

The test for the GPIO interrupt was done as follows:

```
// Test GPIO PF Interrupts
TEST(GPIO_Interrupt)
{
    printf("\n\nBeginning to run test for the GPIO Interrupt for the PF Switches.\n");
    printf("This Test will look at the ability for the switch to affect variables.\n");
```

```
    printf("---------- Let the Test Begin:\n");

    Init_LED();

    // CLear LED Count
    count = 0x0;

    // Setup interrupt for GPIO
    Setup_GPIO_Int(&TestISR);

    // Start GPIO_Interrupt
    Start_GPIO_Int();

    while(count != 0x0F)
    {
        // Do Nothing
    }

    printf("\n\nThe Test is now Finished.\n");

    // Passed if makes it
    CHECK_EQUAL(true, true);

    // Stop Interrupt
    Stop_GPIO_Int();
}
```

As can be seen, this test is very similar to the Core Timer test, except this time, the counter is incremented whenever someone presses one of the PF8-11 switches. Once the counter reaches 15 presses, it finishes the test.

Results

Following the execution of the test program, the tests proved successful. Both the Core Timer and the GPIO interrupts functioned as expected and incremented the counter as necessary.

```
Output

Beginning to run test for the Core Timer Interrupt.
The First part of this test will look at initializing an interrupt.
--------- Let the Test Begin:
Look at the LEDS.
 Are they flashing? (y or n):
..\src\CoreTime_WDog_Interrupts_Test.cpp(70): Success in CoreTimer: y == y
Test #1 is now finished.
..\src\CoreTime_WDog_Interrupts_Test.cpp(47): Error: Failure in CoreTimer: Test Memory leak detected : 520 bytes leaked.


Beginning to run test for the GPIO Interrupt for the PF Switches.
This Test will look at the ability for the switch to affect variables.
---------- Let the Test Begin:


The Test is now Finished.
..\src\CoreTime_WDog_Interrupts_Test.cpp(104): Success in GPIO_Interrupt: 1 == 1
Succesful link to test file CoreTime_WDog_Interrupts_Test_cpp.
FAILURE: 1 out of 4 blackbox tests failed.
Blackbox Assert statistics: 1 Failures, 0 Expected Failures, 2 Successes.
Whitebox Assert statistics: 0 Failures, 0 Expected Failures, 0 Successes. (Includes C Test statistics)
Test time: 16.61932750 seconds.
```

Figure 1: Following the completion of the test, the test was successful. The only failure was due to the known memory leak problem.

Section 4 - Control Coffee Pots with just Interrupts

Overview

For the main portion of this Assignment I created a project called *Assignment2_Custom* and linked it to the CoffeePot Library provided. Following that, I then added two heaps for memory banks to allow for Plug-and-Play cababilities of the CoffeePots, and set the system for 2 Pots.

Main Code Loop - *Assignment2_Main.cpp*

My main Program Code including my main Super Loop is as follows:

```
/**********************************\
*   ENCM 511 - Assignment 2        *
*                                  *
*    Author: Kyle Derby MacInnis   *
*    Date:   November 5, 2014      *
*                                  *
************************************
*       Assignment2_Main.cpp       *
\**********************************/

#include <stdio.h>
#include <sys\exception.h>
#include <cdefbf533.h>
#include "myBlackfin.h"
#include "Updated_CoffeePot_SimulatorFunctions.h"
#include "myCoffeePotFunctions.h"
#include "Assignment2_Main.h"
#include "Assignment2_ISR.h"

// Semaphores for Control Signals
volatile bool SW1_isPressed = false;
volatile bool SW2_isPressed = false;
volatile bool SW3_isPressed = false;
volatile bool SW4_isPressed = false;

// Semaphores for Coffee States
volatile bool WaterOn1      = false;
volatile bool WaterOn2      = false;
volatile bool HeatOn1       = false;
volatile bool HeatOn2       = false;

// Global Coffee Pot Array
COFFEEPOT_DEVICE* CoffeePots1 = NULL;
COFFEEPOT_DEVICE* CoffeePots2 = NULL;

// Semaphores for Coffee Pot Devices
volatile int HeaterLevel1   = HEATLEVEL1;
volatile int HeaterLevel2   = HEATLEVEL2;
volatile int WaterRate1         = WATERRATE1;
volatile int WaterRate2         = WATERRATE2;
volatile int BoostRate1         = BOOSTRATE1;
volatile int BoostRate2         = BOOSTRATE2;

// Semaphores for Coffee Pot Status
volatile bool DeviceReady1  = false;
volatile bool DeviceReady2  = false;
volatile bool LEDPower1     = false;
volatile bool LEDPower2     = false;
volatile bool CoffeePower1  = false;
volatile bool CoffeePower2  = false;
volatile bool WaterPower1   = false;
volatile bool WaterPower2   = false;
volatile bool HeaterPower1  = false;
volatile bool HeaterPower2  = false;
volatile int Temperature1   = NULL;
volatile int Temperature2   = NULL;
volatile int WaterLevel1    = NULL;
volatile int WaterLevel2    = NULL;


// Enumerate Coffee Pots
enum{COFFEEPOT1=1, COFFEEPOT2};

// Main Coffee Pot Function
void main(void)
{
    //Setup Coffee Pot Devices
    int numCoffeePots = 2;

    // Select Display Type
```

```
    int whichDisplay = USE_TEXT_GUI | USE_CCES_GUI; // USE_TEXT_GUI | USE_SPI_GUI |
    USE_CCES_GUI;

    // Prepare Simulation for # of Coffee Pots
    Init_CoffeePotSimulation(numCoffeePots, (WHICHDISPLAY) whichDisplay);

    //Initialize CoffeePots
    COFFEEPOT_DEVICE* baseAddress1 = Add_CoffeePotToSystem_PlugAndPlay(COFFEEPOT1, "
    Kyle42");
    COFFEEPOT_DEVICE* baseAddress2 = Add_CoffeePotToSystem_PlugAndPlay(COFFEEPOT2, "
    LzmfDr.Who");

    //Initialize Coffee Pot Array
    CoffeePots1 = baseAddress1;
    CoffeePots2 = baseAddress2;

    // Print Welcome Statement
    printf("Thank you for choosing Blackfin Coffee: available exclusively at the '
    Restaurant at the End of the Universe'.\n\n");

    printf("About to Start Brewing Coffee from out of this world!\n\n");

    //Initialize Blackfin Flash Memory
    Init_Flash();

    //Initialize Blackfin LEDS
    Init_LED();
    // LED Reader
    unsigned char currentLEDS;

    //Initialize Blackfin Switches
    Init_Switches();

//## Setup Interrupts (Switches)

    //Stop Switch Interrupts
    Stop_GPIO_Int();
    //Setup Switch Interrupts
    Setup_GPIO_Int(&ISR_GPIO_Function);
    //register_handler(ik_ivg7, ISR_GPIO_Function);
    //Start Switch Interrupts
    Start_GPIO_Int();

//## Setup Interrupts (Timer)

    // Stop Core Timer Interrupts
    Stop_CoreTimer_Int();
    // Setup Core Timer Interrupts
    Setup_CoreTimer_Int(&ISR_CoreTimer_Function, TIMER_PERIOD);
    //register_handler(ik_timer, ISR_CoreTimer_Function);
    // Start Core Timer Interrupts
    Start_CoreTimer_Int();

//## Main Super Loop
    while(!SW4_isPressed)
    {
        // If SW1 is Pressed, Initialize Devices
        if(SW1_isPressed)
        {
            if(!DeviceReady1 || !DeviceReady2)
            {
                // Conditional
                bool finished = false;

                // Initialize Coffee Pots
                TurnOnCoffeePot(baseAddress1);
                TurnOnCoffeePot(baseAddress2);

                // Wait until Devices Initialized
                while(!finished)
                {
                    // Check for Device Ready Bits to be Set
                    if(((baseAddress1->controlRegister & DEVICEREADY) == DEVICEREADY) &&
```

```
((baseAddress1->controlRegister & DEVICEREADY) == DEVICEREADY))
                {
                    finished = true;
                }
                else
                {
                    finished = false;
                }

            }
            // Set Devices to Ready
            DeviceReady1 = true;
            DeviceReady2 = true;

            // Turn On LED 1
            currentLEDS = Read_LED();
            currentLEDS |= LED1_BITMASK;
            Write_LED(currentLEDS);
        }
    }
    else
    {
        // If SW1 is Released, Turn off Coffee Pot
        if(DeviceReady1 || DeviceReady2)
        {
            TurnOffCoffeePot(baseAddress1);
            TurnOffCoffeePot(baseAddress2);
            // Device Turned Off, So Not Ready
            DeviceReady1 = false;
            DeviceReady2 = false;

            // Turn Off LED 1
            currentLEDS = Read_LED();
            currentLEDS &= (~LED1_BITMASK);
            Write_LED(currentLEDS);
        }
    }

    // If SW2 is Pressed, Start Water Flowing
    if((DeviceReady1 && DeviceReady2) && (SW2_isPressed))
    {
        WaterLevel1 = WaterLevelRequest(baseAddress1);
        WaterLevel2 = WaterLevelRequest(baseAddress2);

        // If POT 1 IS BELOW MAX, KEEP FILLING
        if ((WaterLevel1 < MAXVOLUME1))
        {
            SetWaterRate(baseAddress1, WATERRATE1);
            TurnOnWater(baseAddress1);
            WaterOn1 = true;
        }
        else
        {
            // Reduce Flow if Full
            SetWaterRate(baseAddress1, 0);
        }

        // If POT 2 IS BELOW MAX, KEEP FILLING
        if ((WaterLevel2 < MAXVOLUME2))
        {
            SetWaterRate(baseAddress2, WATERRATE2);
            TurnOnWater(baseAddress2);
            WaterOn2 = true;
        }
        else
        {
            // Reduce Flow if Full
            SetWaterRate(baseAddress2, 0);
        }
        // Turn On LED 2
        currentLEDS = Read_LED();
        currentLEDS |= LED2_BITMASK;
        Write_LED(currentLEDS);
```

```
        }
        else
        {
            // If SW2 is released, Turn off Water
            if(WaterOn1 || WaterOn2)
            {
                TurnOffWater(baseAddress1);
                TurnOffWater(baseAddress2);

                WaterOn1 = false;
                WaterOn2 = false;

                // Turn Off LED 2
                currentLEDS = Read_LED();
                currentLEDS &= (~LED2_BITMASK);
                Write_LED(currentLEDS);
            }
        }

        // If SW3 is Pressed, Start Heater Going
        if((DeviceReady1 && DeviceReady2) && (SW3_isPressed))
        {
            Temperature1 = CurrentTemperatureRequest(baseAddress1);
            Temperature2 = CurrentTemperatureRequest(baseAddress2);

            // If POT 1 IS BELOW MAX, KEEP FILLING
            if ((Temperature1 < MAXTEMP))
            {
                SetHeaterRate(baseAddress1, HEATLEVEL1, BOOSTRATE1);
                TurnOnHeater(baseAddress1);
                HeatOn1 = true;
            }
            else
            {
                // Set Heat To Low
                SetHeaterRate(baseAddress1, 0, 1);
            }

            // If POT 2 IS BELOW MAX, KEEP FILLING
            if ((Temperature2 < MAXTEMP))
            {
                SetHeaterRate(baseAddress2, HEATLEVEL2, BOOSTRATE2);
                TurnOnHeater(baseAddress2);
                HeatOn2 = true;
            }
            else
            {
                // Set Heat to Low
                SetHeaterRate(baseAddress2, 0, 1);
            }

            // Turn On LED 3
            currentLEDS = Read_LED();
            currentLEDS |= LED3_BITMASK;
            Write_LED(currentLEDS);
        }
        else
        {
            // If SW3 is released, Turn off Heater
            if(HeatOn1 || HeatOn2)
            {
                TurnOffHeater(baseAddress1);
                TurnOffHeater(baseAddress2);
                HeatOn1 = false;
                HeatOn2 = false;

                // Turn Off LED 3
                currentLEDS = Read_LED();
                currentLEDS &= (~LED3_BITMASK);
                Write_LED(currentLEDS);
            }
        }
```

```
        // If Temperature and Water are at Proper Levels
        // Start the Coffee Pod and Make Coffee
        if((WaterLevel1 >= MAXVOLUME1) && (Temperature1 >= MAXTEMP))
        {
            // Start Coffee Pod for Coffee Pot 1
            Insert_Coffee(baseAddress1);
        }

        if((WaterLevel2 >= MAXVOLUME2) && (Temperature2 >= MAXTEMP))
        {
            // Start Coffee Pod for Coffee Pot 2
            Insert_Coffee(baseAddress2);
        }
    }

    // STOP CORE TIMER INTERRUPT
    Stop_CoreTimer_Int();
    // STOP GPIO INTERRUPT
    Stop_GPIO_Int();

    // Eject Coffee Pots From System after SW4 is Pressed
    Remove_CoffeePotFromSystem(baseAddress1);
    Remove_CoffeePotFromSystem(baseAddress2);

    // Clear LEDS
    Write_LED(0x0);
}

// CCES GUI DELAY FUNCTION
void USE_CCES_GUI_Delay(void) {
    int delay = 0xFFFF;
        for ( int count = 0; count < delay; count++) {    // 0x1FFFFFF Seemed to work
    abeit slowly
            count = count + 1;
        }
}
```

As can be seen, this function sets a bunch of global semaphores for use with ISR's, it then proceeds to initialize the coffee pot devices with their necessary parameters. Following this, the function initializes the flash memory, blackfin LEDs, and the GPIO Switches, and once they are initialized, it then prepares the interrupts for the Core Timer and the GPIO. Finally after all of this has been initialized, it proceeds to go into the Super Loop.

Inside the super Loop, the program does the following:

- Checks for Switches 1, 2, 3, and 4 via the GPIO ISR.
- If Switch 1 is pressed, it Turns on and Prepares the Coffee Pot.
- If Switch 2 is pressed, and the device is ready (see above), it turns on and monitors the water level.
- If Switch 3 is pressed, and device is ready, it proceeds to turn on and monitor the heat.
- If Switch 4 is pressed, it proceeds to exit the loop, eject the coffeepots and stop the interrupts.
- Additionally, once the heat and water levels reach appropriate amounts, the coffee pots are then set to make coffee by turning on the coffee pod.

Main Code Header File - *Assignment2_Main.h*

The following is my code for my main header file:

```
/*********************************\
*   ENCM 511 - Assignment 2       *
*                                 *
*   Author: Kyle Derby MacInnis   *
*   Date:   November 5, 2014      *
*                                 *
***********************************
*       Assignment2_Main.h        *
```

```
\*********************************/

#include "Updated_CoffeePot_SimulatorFunctions.h"
#include "myCoffeePotFunctions.h"

#ifndef _ASSN2_MAIN_H
#define _ASSN2_MAIN_H

// Figure Out Empirically
    // MAX VOLUME
#define MAXVOLUME1        300
#define MAXVOLUME2        325
    // MAX TEMP
#define MAXTEMP           85
    // HEAT
#define HEATLEVEL1        150
#define HEATLEVEL2        200
    // WATER
#define WATERRATE1        25
#define WATERRATE2        50
    // BOOST
#define BOOSTRATE1        10
#define BOOSTRATE2        15

// Semaphores for Control Signals
extern volatile bool SW1_isPressed;
extern volatile bool SW2_isPressed;
extern volatile bool SW3_isPressed;
extern volatile bool SW4_isPressed;

// Semaphores for Coffee States
extern volatile bool WaterOn1;
extern volatile bool WaterOn2;
extern volatile bool HeatOn1;
extern volatile bool HeatOn2;

// Global Coffee Pot Array
extern COFFEEPOT_DEVICE* CoffeePots1;
extern COFFEEPOT_DEVICE* CoffeePots2;

// Semaphores for Coffee Pot Devices
extern volatile int HeaterLevel1;
extern volatile int HeaterLevel2;
extern volatile int WaterRate1;
extern volatile int WaterRate2;
extern volatile int BoostRate1;
extern volatile int BoostRate2;

// Semaphores for Coffee Pot Status
extern volatile bool DeviceReady1;
extern volatile bool DeviceReady2;
extern volatile bool LEDPower1;
extern volatile bool LEDPower2;
extern volatile bool CoffeePower1;
extern volatile bool CoffeePower2;
extern volatile bool WaterPower1;
extern volatile bool WaterPower2;
extern volatile bool HeaterPower1;
extern volatile bool HeaterPower2;
extern volatile int Temperature1;
extern volatile int Temperature2;
extern volatile int WaterLevel1;
extern volatile int WaterLevel2;

#endif
```

As can be seen, this header file declares some constants for use as well as declaring the global semaphores within the main CPP file for use in other files.

Interrupt Service Routines - *Assignment2_ISR.cpp*

The following is my code for the ISR source file in which all my ISR's used are defined:

```
/*********************************\
*   ENCM 511 - Assignment 2       *
*                                 *
*   Author: Kyle Derby MacInnis   *
*   Date:   November 5, 2014      *
*                                 *
***********************************
*       Assignment2_ISR.cpp       *
\*********************************/

#include "myBlackfin.h"
#include "Assignment2_ISR.h"
#include "Assignment2_Main.h"
#include "Updated_CoffeePot_SimulatorFunctions.h"
#include "myCoffeePotFunctions.h"

// TODO - ISR Function for WDOG
//#pragma interrupt
//void ISR_WDog_Function(void)
EX_INTERRUPT_HANDLER(ISR_WDog_Function)
{
    // TODO
}

// Core Timer Interrupt Service Routine
#pragma interrupt
void ISR_CoreTimer_Function(void)
//EX_INTERRUPT_HANDLER(ISR_CoreTimer_Function)
{
    // Flash LED5 each timer it runs
    unsigned char currentLEDS = Read_LED();

    if((currentLEDS & LED5_BITMASK) == LED5_BITMASK)
    {
        currentLEDS &= (~LED5_BITMASK);
    }
    else
    {
        currentLEDS |= LED5_BITMASK;
    }

    Write_LED(currentLEDS);

    // Measure Temperature of Pots
    Temperature1 = CurrentTemperatureRequest(CoffeePots1);
    Temperature2 = CurrentTemperatureRequest(CoffeePots2);

    // Measure Water Level of Pots
    WaterLevel1 = WaterLevelRequest(CoffeePots1);
    WaterLevel2 = WaterLevelRequest(CoffeePots2);

    // Update Simulation
    UpdateSimulationDisplay();
}

// GPIO Interrupt Service Routine
#pragma interrupt
void ISR_GPIO_Function(void)
//EX_INTERRUPT_HANDLER(ISR_GPIO_Function)
{
    // Read Switches
    unsigned short currentSwitchValue = Read_Switches();

    // Acknowledge Interrupt
    *pFIO_FLAG_D &= (~0x0F00);
    // Force Write
    Ssync();

    // SW1 - Toggle Bool
    if((currentSwitchValue & (SW1_BITMASK)) == (SW1_BITMASK))
```

```
    {
        SW1_isPressed = (!SW1_isPressed);
    }
    //else
        //SW1_isPressed = false;

    // SW2 - Toggle Bool
    if((currentSwitchValue & (SW2_BITMASK)) == (SW2_BITMASK))
    {
        SW2_isPressed = (!SW2_isPressed);
    }
    //else
        //SW2_isPressed = false;

    // SW3 - Toggle Bool
    if((currentSwitchValue & (SW3_BITMASK)) == (SW3_BITMASK))
    {
        SW3_isPressed = (!SW3_isPressed);
    }
    //else
        //SW3_isPressed = false;

    // SW4 - Toggle Bool
    if((currentSwitchValue & (SW4_BITMASK)) == (SW4_BITMASK))
    {
        SW4_isPressed = (!SW4_isPressed);
    }
    //else
        //SW4_isPressed = false;
}
```

As shown above, there are two functions which have been fleshed out as well as another ISR, *void ISR_WDog_Function(void)* which is empty. I was having trouble getting the Watchdog timer to properly reset itself, so I gave up on using it, and decided to use the Core Timer instead, hence the ISR for it. The Core Timer ISR acts as a constant updator. It runs about once every second or so, and when it runs it launches the update display function provided for the lab. It also makes an updated reading on the current temperatures and water levels of the pots to ensure that these values remain accurate.

Interrupt Service Routines - *Assignment2_ISR.h*

The following is my code for the header file declaring the ISR functions:

```
/**********************************\
*   ENCM 511 - Assignment 2        *
*                                  *
*   Author: Kyle Derby MacInnis    *
*   Date:   November 5, 2014       *
*                                  *
**********************************
*       Assignment2_ISR.h        *
\**********************************/
#include <sys\exception.h>  // Need if using MACROs
#include <cdefbf533.h>      // Same as above
#ifndef _ASSN2_ISR_H
#define _ASSN2_ISR_H

// WDOG/CORE TIMER PERIOD (FIGURE OUT)
#define TIMER_PERIOD          0x02FFFFFF

//EX_INTERRUPT_HANDLER(ISR_WDog_Function);
// WDOG ISR Prototype
#pragma interrupt
void ISR_WDog_Function(void);

//EX_INTERRUPT_HANDLER(ISR_GPIO_Function);
// GPIO ISR Prototype
#pragma interrupt
void ISR_GPIO_Function(void);
```

```
//EX_INTERRUPT_HANDLER(ISR_CoreTimer_Function);
// Core Timer ISR Prototype
#pragma interrupt
void ISR_CoreTimer_Function(void);

#endif
```

This file just merely declared my ISR functions using the pragma *interrupt* which tells the preprocessor that these are ISR functions. I chose not to use the MACRO for my own personal learning, but the MACRO does the exact same thing as I wrote beneath it.

Coffee Pot Functions - *myCoffeePotFunctions.cpp*

The Following is my code written up for the coffee pot functions used in this assignment:

```
/*********************************\
*   ENCM 511 - Assignment 2       *
*                                 *
*   Author: Kyle Derby MacInnis   *
*   Date:   November 5, 2014      *
*                                 *
***********************************
*       myCoffeePotFunctions.cpp  *
\*********************************/

#include <stdio.h>
#include "myBlackfin.h"
#include "myCoffeePotFunctions.h"

// Turn On Coffee Pot
void TurnOnCoffeePot(COFFEEPOT_DEVICE* baseAddress)
{
    // Read Register Value
    unsigned short int currentRegisterValue = baseAddress->controlRegister;
    // Set to New Control Value (Initialize and Power On)
    unsigned short int updatedRegisterValue = currentRegisterValue | INIT_COFFEEPOT |
    LED_POWER;
    updatedRegisterValue = updatedRegisterValue | POWER_LED | LEDPOWER_LED;
    // Update Register
    baseAddress->controlRegister = updatedRegisterValue;
    // Force Write
    Ssync();
}

// Turn Off Coffee Pot
void TurnOffCoffeePot(COFFEEPOT_DEVICE* baseAddress)
{
    // Read in Current Register Value
    unsigned short int currentRegisterValue = baseAddress->controlRegister;
    // Turn Off Power to Coffee Pot
    unsigned short int updatedRegisterValue = currentRegisterValue & (~INIT_COFFEEPOT) &
     (~LED_POWER);
    updatedRegisterValue = updatedRegisterValue & (~POWER_LED) & (~LEDPOWER_LED);
    // Update Register
    baseAddress->controlRegister = updatedRegisterValue;
    // Force Write
    Ssync();
}

// Set Water Flow Rate
void SetWaterRate(COFFEEPOT_DEVICE* baseAddress, unsigned char Rate)
{
    // Set Water Flow Rate
    baseAddress->waterInFlowRegister = Rate;
    // Force Write
    Ssync();
}

// Turn On Water
void TurnOnWater(COFFEEPOT_DEVICE* baseAddress)
```

```
{
    // Turn on the power to water and LED
     unsigned short int currentRegisterValue = baseAddress->controlRegister;
     unsigned short int updatedRegisterValue = currentRegisterValue | WATERPOWER;
     updatedRegisterValue = updatedRegisterValue | WATER_LED ;
     // Update Control Register
     baseAddress->controlRegister = updatedRegisterValue ;
     // Force Write
     Ssync();
}

// Turn Off Water
void TurnOffWater(COFFEEPOT_DEVICE* baseAddress)
{
    // Turn off the power to water and LED
     unsigned short int currentRegisterValue = baseAddress->controlRegister;
     unsigned short int updatedRegisterValue = currentRegisterValue & (~WATERPOWER);
     updatedRegisterValue = updatedRegisterValue & (~WATER_LED) ;
     // Update Control Register
     baseAddress->controlRegister = updatedRegisterValue ;
     // Force Write
     Ssync();
}

// Set Heater Rate
void SetHeaterRate(COFFEEPOT_DEVICE* baseAddress, unsigned char Rate, unsigned char
    Boost)
{
    // Set Heater Flow Rate
    baseAddress->heaterRegister = Rate;
    // Set Heater Boost Rate
    baseAddress->heaterBoostRegister = Boost;
    // Force Write
    Ssync();
}

// Turn On Heater
void TurnOnHeater(COFFEEPOT_DEVICE* baseAddress)
{
    // Turn on the power to Heater and LED
     unsigned short int currentRegisterValue = baseAddress->controlRegister;
     unsigned short int updatedRegisterValue = currentRegisterValue | HEATERPOWER;
     updatedRegisterValue = updatedRegisterValue | HEATER_LED ;
     // Update Control Register
     baseAddress->controlRegister = updatedRegisterValue ;
     // Force Write
     Ssync();
}

// Turn Off Heater
void TurnOffHeater(COFFEEPOT_DEVICE* baseAddress)
{
    // Turn off the power to heater and LED
     unsigned short int currentRegisterValue = baseAddress->controlRegister;
     unsigned short int updatedRegisterValue = currentRegisterValue & (~HEATERPOWER);
     updatedRegisterValue = updatedRegisterValue & (~HEATER_LED) ;
     // Update Control Register
     baseAddress->controlRegister = updatedRegisterValue ;
     // Force Write
     Ssync();
}

// Insert Coffee
void Insert_Coffee(COFFEEPOT_DEVICE* baseAddress)
{
    // Insert Coffee to the pot
    unsigned short int currentRegisterValue = baseAddress->controlRegister;
    unsigned short int updatedRegisterValue = currentRegisterValue | COFFEE_INSERT;
    // Turn Heat Down Low
    SetHeaterRate(baseAddress, 0, 1);
    // Turn Water Flow Low
    SetWaterRate(baseAddress, 0);
    // Update Control Register
```

```
    baseAddress->controlRegister = updatedRegisterValue;
    // Force Write
    Ssync();
}
```

The only modifications that I really made between my first assignment and this one was that I expanded on a couple functions, and disregarded others. I did not decide to use a *PrepareTheCoffeePot()* function as I merely placed that code directly into my Super Loop, additionally, I added a *TurnOfCoffeePot()* function. I also changed the water and heat functions to be a bit more specific. Such as making "turn-off" versions and setting the rate via separate functions. This was just done so that I had more control and ease of writing my code as it would align better with my pseudo-code.

Coffee Pot Functions - *myCoffeePotFunctions.h*

The following is my code for the Coffee Pot function header file:

```
/*********************************\
*   ENCM 511 - Assignment 2       *
*                                 *
*   Author: Kyle Derby MacInnis   *
*   Date:   November 5, 2014      *
*                                 *
***********************************
*       myCoffeePotFunctions.h    *
\*********************************/

#include <stdio.h>
#include "Updated_CoffeePot_SimulatorFunctions.h"

#ifndef _MY_COFFEEPOT_FUNC_H
#define _MY_COFFEEPOT_FUNC_H

// COFFEEPOT CONTROL BITMASKS
#define INIT_COFFEEPOT          (1<<0)              //  (0x0001)
#define LED_POWER               (1<<1)              //  (0x0002)
#define WATERPOWER              (1<<2)              //  (0x0004)
#define HEATERPOWER         (1<<3)              //  (0x0008)
#define DEVICEREADY             (1<<4)              //  (0x0010)
#define COFFEE_INSERT           (1<<11)             //  (0x0800)

// COFFEEPOT LED CONTROL BITMASKS
#define LED_OFFSET              12                  //  Bit 12
#define COFFEE_LED1             (1<<(LED_OFFSET+0)) //  (0x1000)
#define COFFEE_LED2             (1<<(LED_OFFSET+1)) //  (0x2000)
#define COFFEE_LED3             (1<<(LED_OFFSET+2)) //  (0x4000)
#define COFFEE_LED4             (1<<(LED_OFFSET+3)) //  (0x8000)

// DEFINE ALIAS FOR LEDS
#define POWER_LED               COFFEE_LED1
#define LEDPOWER_LED            COFFEE_LED2
#define WATER_LED               COFFEE_LED3
#define HEATER_LED              COFFEE_LED4

// FUNCTION DECLARATIONS

// Init Coffee Pot
void Init_CoffeePot(COFFEEPOT_DEVICE* baseAddress);

// Turn On Coffee Pot
void TurnOnCoffeePot(COFFEEPOT_DEVICE* baseAddress);

// Turn Off Coffee Pot
void TurnOffCoffeePot(COFFEEPOT_DEVICE* baseAddress);

// Set Water Flow Rate
void SetWaterRate(COFFEEPOT_DEVICE* baseAddress, unsigned char Rate);

// Turn On Water
void TurnOnWater(COFFEEPOT_DEVICE* baseAddress);
```

```
// Turn Off Water
void TurnOffWater(COFFEEPOT_DEVICE* baseAddress);

// Set Heater Rate
void SetHeaterRate(COFFEEPOT_DEVICE* baseAddress, unsigned char Rate, unsigned char
    Boost);

// Turn On Heater
void TurnOnHeater(COFFEEPOT_DEVICE* baseAddress);

// Turn Off Heater
void TurnOffHeater(COFFEEPOT_DEVICE* baseAddress);

// Insert Coffee
void Insert_Coffee(COFFEEPOT_DEVICE* baseAddress);

#endif
```

Again, this header file merely defines some constants and bitmasks, and then declares the functions defined within the source file.

Results

The fourth portion of this assignment was developed after arduous labour, but alas it worked as expected. Once the program is loaded and run, the processor does nothing but loop through the super loop until a button it pressed. Whilst it loops it is constantly running the timer interrupt which flashes the fifth LED on the blackfin. Once a button is pressed, it is mirrored on the LEDs and then the corresponding globabl semaphore is then set to reflect the current state of affairs regarding the switches. This then sets off a variety of conditionals within the main super loop.

Following are some of the screen captures taken, showcasing the progress of the code as it fills up and prepares the two coffee pots:

Before SW1 is Pressed - Looping

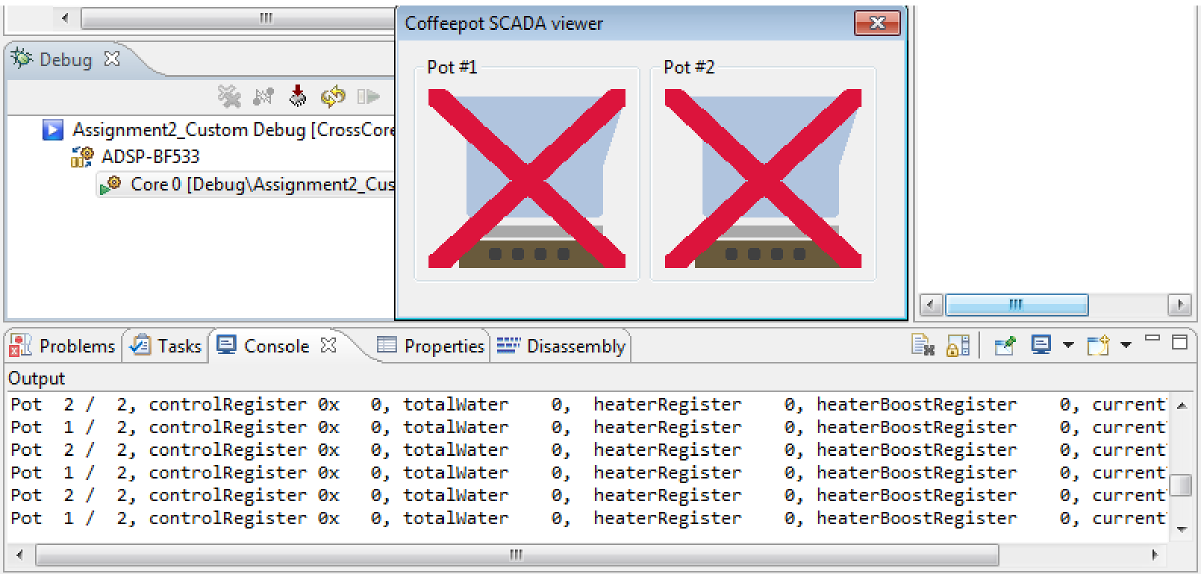

Figure 2: This figure shows that Coffee pot prior to SW1 being pressed.

After SW1 is Pressed - Coffee Pot Initializes

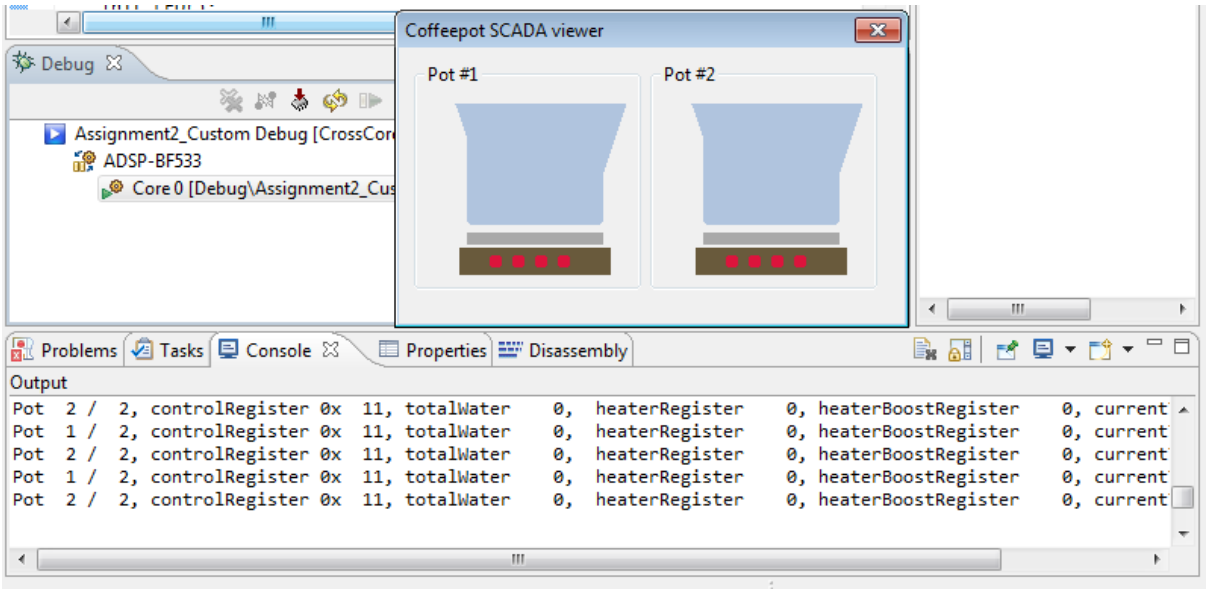

Figure 3: After SW1 is pressed, the coffee pot goes into the loop and turns on. (Some error with CCES GUI though - LEDs)

After Sw2 is Pressed - Water Fills

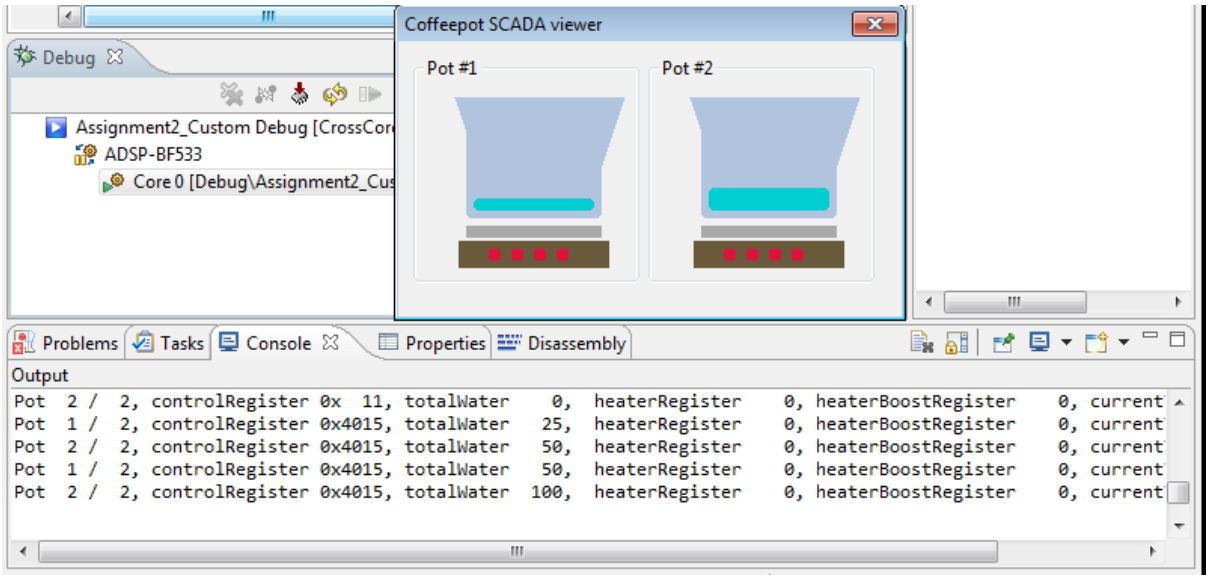

Figure 4: As can be seen the water has begun to fill in the pots, but they are filling at different rates.

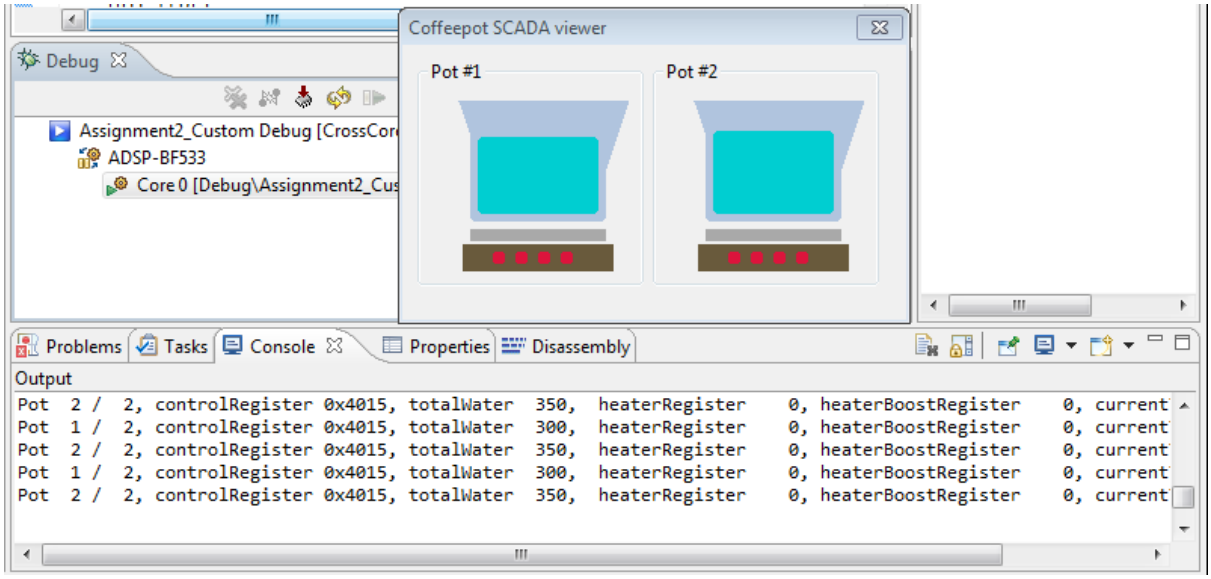

Figure 5: Once the water reaches the intended maximum, it stops and holds the volume. Notice the difference in amount.

After SW3 is Pressed - Heater Starts

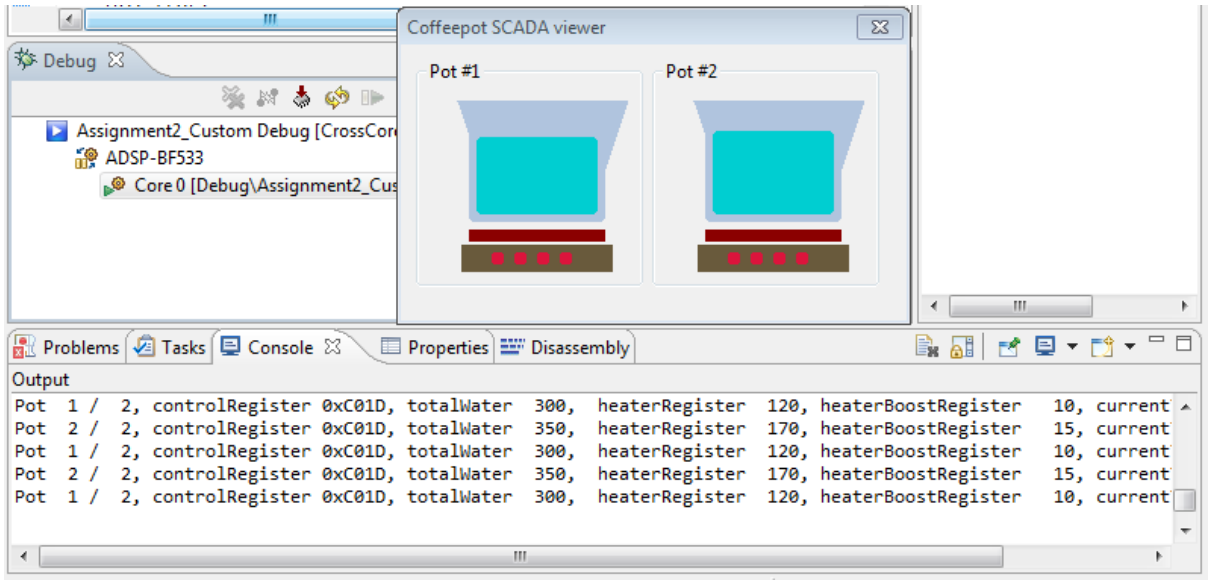

Figure 6: The heater turns on and the water begins to heat up.

Once Water and Heat is Ready - Insert Coffee

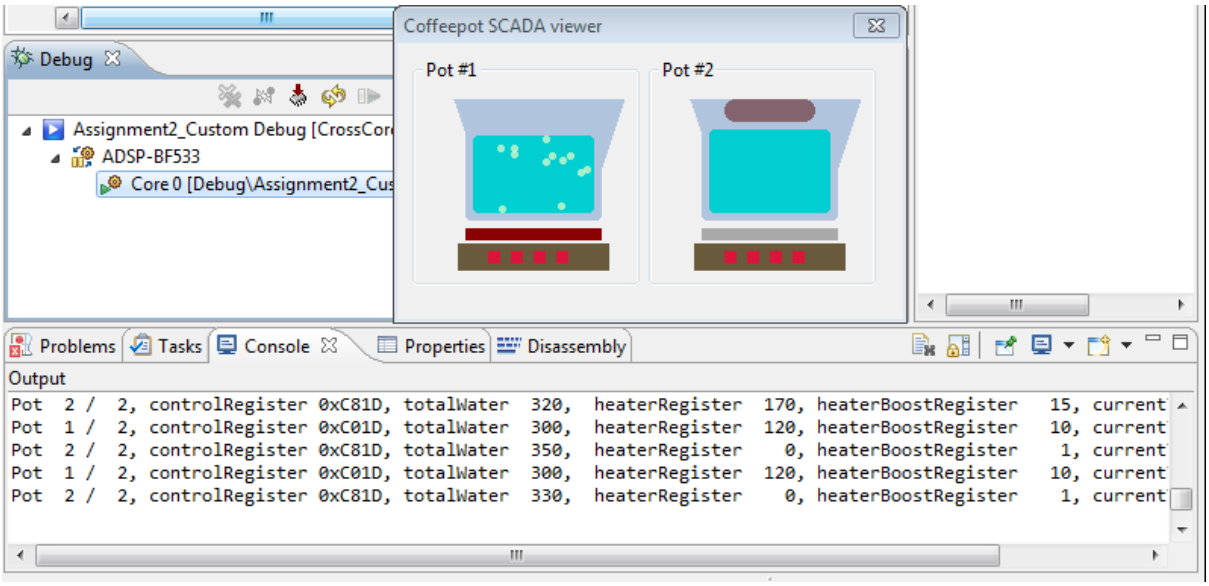

Figure 7: Once the water starts to get hot enough, it begins to boil. If water and heat are ready, coffee gets inserted. Notice only one has coffee ready.

After SW4 is pressed - Power Off

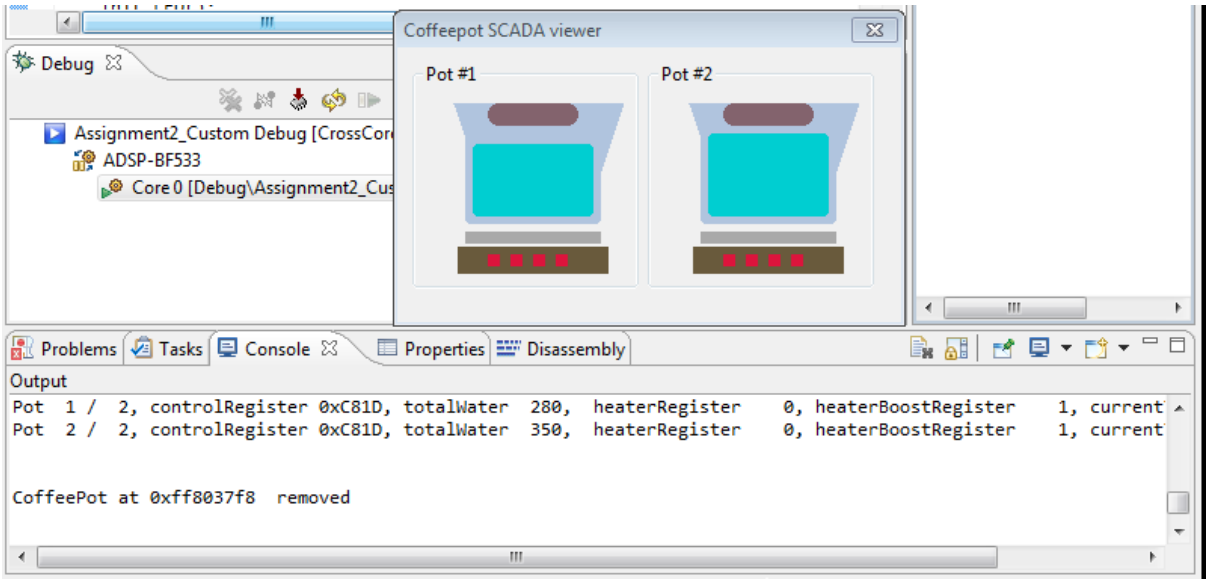

Figure 8: Once SW4 is pressed or Power is removed, the coffee pot turns off and is then ejected.

Conclusions

This assignment was difficult in comparison to the first assignment. The amount of referencing I had to do in regards to the programmer and hardware references was much higher than any previous part of this course. This lab despite its difficulties did help me have a much better understanding of how interrupts function and their importance. The most difficult part of this lab was remember quirks about the semaphores and the ensuring that the ISR was properly loaded into the Event Vector Table (EVT). All in all, I found this assignment difficult, but once I had progressed through the mud and I had a fleshed out program, I was able to debug it effectively and bring it into working fruition. I chose to forgo certain portions of the assignment due to time constraints and the fact that I felt it would be better

understood and expressed in later portions of the lab rather than writing redundant and obselete code at the beginning. Essentially, I made an executive decision based on the necessary final requirements of the projects and chose to ignore some important steps because I felt I could manage to skip them. In a real world project this could come with consequences, but for this assignment I felt that the consequences were worth the risk.

Appendix A - My Blackfin Library

Blackfin Library Header - *myBlackfin.h*

The following is the header file for my Blackfin Library:

```
/**********************************\
*   ENCM 511 - Assignment 2        *
*                                  *
*    Author: Kyle Derby MacInnis   *
*    Date:   November 5, 2014      *
*                                  *
************************************
*        myBlackfin.h              *
\**********************************/

#ifndef _MY_BLACKFIN_H
#define _MY_BLACKFIN_H

//########################################
//#                                      #
//#    REGISTER ADDRESS DECLARATIONS     #
//#                                      #
//#                                      #
//########################################

//## IO PROGRAMMABLE REGISTERS

// BLACKFIN LED CONTROL REGISTER ADDRESSES (FLASH)
#define LED_IO_STAT       0x20270001
#define LED_IO_SET        0x20270005
#define LED_IO_DIR        0x20270007

//## PROGRAMMABLE FLAG REGISTERS

// FIO_FLAG REGISTERS
#define FIO_FLAG_D              0xFFC00700
#define FIO_FLAG_C              0xFFC00704
#define FIO_FLAG_S              0xFFC00708
#define FIO_FLAG_T              0xFFC0070C
// FIO_MASKA REGISTERS
#define FIO_MASKA_D             0xFFC00710
#define FIO_MASKA_C             0xFFC00714
#define FIO_MASKA_S             0xFFC00718
#define FIO_MASKA_T             0xFFC0071C
// FIO_MASKB REGISTERS
#define FIO_MASKB_D             0xFFC00720
#define FIO_MASKB_C             0xFFC00724
#define FIO_MASKB_S             0xFFC00728
#define FIO_MASKB_T             0xFFC0072C
// FIO
#define FIO_DIR                 0xFFC00730
#define FIO_POLAR               0xFFC00734
#define FIO_EDGE                0xFFC00738
#define FIO_BOTH                0xFFC0073C
#define FIO_INEN                0xFFC00740

//## PHASE LOCK LOOP REGISTERS
#define PLL_CTL                 0xFFC00000
#define PLL_DIV                 0xFFC00004
#define VR_CTL                  0xFFC00008
#define PLL_STAT                0xFFC0000C
#define PLL_LOCKCNT             0xFFC00010

//## SYSTEM INTERRUPT REGISTERS

#define SWRST                   0xFFC00100
#define SYSCR                   0xFFC00104
#define SIC_IMASK               0xFFC0010C
#define SIC_IAR0                0xFFC00110
#define SIC_IAR1                0xFFC00114
#define SIC_IAR2                0xFFC00118
#define SIC_ISR                 0xFFC00120
```

```
#define SIC_IWR                 0xFFC00124

//## PPI CONTROLLER REGISTERS

#define PPI_CONTROL             0xFFC01000
#define PPI_STATUS              0xFFC01004
#define PPI_COUNT               0xFFC01008
#define PPI_DELAY               0xFFC0100C
#define PPI_FRAME               0xFFC01010

//## SPI CONTROLLER REGISTERS

#define SPI_CTL                 0xFFC00500
#define SPI_FLG                 0xFFC00504
#define SPI_STAT                0xFFC00508
#define SPI_TDBR                0xFFC0050C
#define SPI_RDBR                0xFFC00510
#define SPI_BAUD                0xFFC00514
#define SPI_SHADOW              0xFFC00518

//## GP TIMER PROGRAMMABLE REGISTERS

#define TIMER_ENABLE            0xFFC00640
#define TIMER_DISABLE           0xFFC00644
#define TIMER_STATUS            0xFFC00648
// TIMER0 REGISTERS
#define TIMER0_CONFIG           0xFFC00600
#define TIMER0_COUNTER          0xFFC00604
#define TIMER0_PERIOD           0xFFC00608
#define TIMER0_WIDTH            0xFFC0060C
// TIMER1 REGISTERS
#define TIMER1_CONFIG           0xFFC00610
#define TIMER1_COUNTER          0xFFC00614
#define TIMER1_PERIOD           0xFFC00618
#define TIMER1_WIDTH            0xFFC0061C
// TIMER2 REGISTERS
#define TIMER2_CONFIG           0xFFC00620
#define TIMER2_COUNTER          0xFFC00624
#define TIMER2_PERIOD           0xFFC00628
#define TIMER2_WIDTH            0xFFC0062C

//## WATCHDOG TIMER PROGRAMMABLE REGISTERS

#define WDOG_CTL                0xFFC00200
#define WDOG_CNT                0xFFC00204
#define WDOG_STAT               0xFFC00208

//## REAL-TIME CLOCK PROGRAMMABLE REGISTERS

#define RTC_STAT                0xFFC00300
#define RTC_ICTL                0xFFC00304
#define RTC_ISTAT               0xFFC00308
#define RTC_SWCNT               0xFFC0030C
#define RTC_ALARM               0xFFC00310
#define RTC_PREN                0xFFC00314

//## EBIU PROGRAMMABLE REGISTERS

#define EBIU_AMGCTL             0xFFC00A00
#define EBIU_AMBCTL0            0xFFC00A04
#define EBIU_AMBCTL1            0xFFC00A08
#define EBIU_SDGCTL             0xFFC00A10
#define EBIU_SDBCTL             0xFFC00A14
#define EBIU_SDSTAT             0xFFC00A1C
#define EBIU_SDRRC              0xFFC00A18

//## CORE EVENT VECTOR TABLE REGISTERS
#define EVT0                    0xFFE02000
#define EVT1                    0xFFE02004
#define EVT2                    0xFFE02008
#define EVT3                    0xFFE0200C
#define EVT4                    0xFFE02010
#define EVT5                    0xFFE02014
```

```
#define EVT6                    0xFFE02018
#define EVT7                    0xFFE0201C
#define EVT8                    0xFFE02020
#define EVT9                    0xFFE02024
#define EVT10                   0xFFE02028
#define EVT11                   0xFFE0202C
#define EVT12                   0xFFE02030
#define EVT13                   0xFFE02034
#define EVT14                   0xFFE02038
#define EVT15                   0xFFE0203C

//## CORE EVENT CONTROLLER REGISTERS

#define IMASK                   0xFFE02104
#define IPEND                   0xFFE02108
#define ILAT                    0xFFE0210C
#define IPRIO                   0xFFE02110

//## CORE TIMER PROGRAMMABLE REGISTERS

#define TCNTL                   0xFFE03000
#define TPERIOD                 0xFFE03004
#define TSCALE                  0xFFE03008
#define TCOUNT                  0xFFE0300C


//##########################################
//#                                        #
//#    REGISTER POINTER DECLARATIONS       #
//#                                        #
//#                                        #
//##########################################

// LED CONTROL REGISTER POINTERS
#define pLED_IO_STAT            ((volatile unsigned char *)LED_IO_STAT)
#define pLED_IO_SET             ((volatile unsigned char *)LED_IO_SET)
#define pLED_IO_DIR             ((volatile unsigned char *)LED_IO_DIR)

// FIO_FLAG REGISTERS
#define pFIO_FLAG_D             ((volatile unsigned short *)FIO_FLAG_D)
#define pFIO_FLAG_C             ((volatile unsigned short *)FIO_FLAG_C)
#define pFIO_FLAG_S             ((volatile unsigned short *)FIO_FLAG_S)
#define pFIO_FLAG_T             ((volatile unsigned short *)FIO_FLAG_T)
// FIO_MASKA REGISTERS
#define pFIO_MASKA_D            ((volatile unsigned short *)FIO_MASKA_D)
#define pFIO_MASKA_C            ((volatile unsigned short *)FIO_MASKA_C)
#define pFIO_MASKA_S            ((volatile unsigned short *)FIO_MASKA_S)
#define pFIO_MASKA_T            ((volatile unsigned short *)FIO_MASKA_T)
// FIO_MASKB REGISTERS
#define pFIO_MASKB_D            ((volatile unsigned short *)FIO_MASKB_D)
#define pFIO_MASKB_C            ((volatile unsigned short *)FIO_MASKB_C)
#define pFIO_MASKB_S            ((volatile unsigned short *)FIO_MASKB_S)
#define pFIO_MASKB_T            ((volatile unsigned short *)FIO_MASKB_T)
// FIO
#define pFIO_DIR                ((volatile unsigned short *)FIO_DIR)
#define pFIO_POLAR              ((volatile unsigned short *)FIO_POLAR)
#define pFIO_EDGE               ((volatile unsigned short *)FIO_EDGE)
#define pFIO_BOTH               ((volatile unsigned short *)FIO_BOTH)
#define pFIO_INEN               ((volatile unsigned short *)FIO_INEN)

//## PHASE LOCK LOOP REGISTERS
#define pPLL_CTL                ((volatile unsigned short *)PLL_CTL)
#define pPLL_DIV                ((volatile unsigned short *)PLL_DIV)
#define pVR_CTL                 ((volatile unsigned short *)VR_CTL)
#define pPLL_STAT               ((volatile unsigned short *)PLL_STAT)
#define pPLL_LOCKCNT            ((volatile unsigned short *)PLL_LOCKCNT)

//## SYSTEM INTERRUPT REGISTERS

#define pSWRST                  ((volatile unsigned short *)SWRST)
#define pSYSCR                  ((volatile unsigned short *)SYSCR)
#define pSIC_IMASK              ((volatile unsigned long *)SIC_IMASK)
#define pSIC_IAR0               ((volatile unsigned long *)SIC_IAR0)
```

```
#define pSIC_IAR1               ((volatile unsigned long *)SIC_IAR1)
#define pSIC_IAR2               ((volatile unsigned long *)SIC_IAR2)
#define pSIC_ISR                ((volatile unsigned long *)SIC_ISR)
#define pSIC_IWR                ((volatile unsigned long *)SIC_IWR)

//## PPI CONTROLLER REGISTERS

#define pPPI_CONTROL            ((volatile unsigned short *)PPI_CONTROL)
#define pPPI_STATUS             ((volatile unsigned short *)PPI_STATUS)
#define pPPI_COUNT              ((volatile unsigned short *)PPI_COUNT)
#define pPPI_DELAY              ((volatile unsigned short *)PPI_DELAY)
#define pPPI_FRAME              ((volatile unsigned short *)PPI_FRAME)

//## SPI CONTROLLER REGISTERS

#define pSPI_CTL                ((volatile unsigned short *)SPI_CTL)
#define pSPI_FLG                ((volatile unsigned short *)SPI_FLG)
#define pSPI_STAT               ((volatile unsigned short *)SPI_STAT)
#define pSPI_TDBR               ((volatile unsigned short *)SPI_TDBR)
#define pSPI_RDBR               ((volatile unsigned short *)SPI_RDBR)
#define pSPI_BAUD               ((volatile unsigned short *)SPI_BAUD)
#define pSPI_SHADOW             ((volatile unsigned short *)SPI_SHADOW)

//## GP TIMER PROGRAMMABLE REGISTERS

#define pTIMER_ENABLE           ((volatile unsigned short *)TIMER_ENABLE)
#define pTIMER_DISABLE          ((volatile unsigned short *)TIMER_DISABLE)
#define pTIMER_STATUS           ((volatile unsigned short *)TIMER_STATUS)
// TIMER0 REGISTERS
#define pTIMER0_CONFIG          ((volatile unsigned short *)TIMER0_CONFIG)
#define pTIMER0_COUNTER         ((volatile unsigned long *)TIMER0_COUNTER)
#define pTIMER0_PERIOD          ((volatile unsigned long *)TIMER0_PERIOD)
#define pTIMER0_WIDTH           ((volatile unsigned long *)TIMER0_WIDTH)
// TIMER1 REGISTERS
#define pTIMER1_CONFIG          ((volatile unsigned short *)TIMER1_CONFIG)
#define pTIMER1_COUNTER         ((volatile unsigned long *)TIMER1_COUNTER)
#define pTIMER1_PERIOD          ((volatile unsigned long *)TIMER1_PERIOD)
#define pTIMER1_WIDTH           ((volatile unsigned long *)TIMER1_WIDTH)
// TIMER2 REGISTERS
#define pTIMER2_CONFIG          ((volatile unsigned short *)TIMER2_CONFIG)
#define pTIMER2_COUNTER         ((volatile unsigned long *)TIMER2_COUNTER)
#define pTIMER2_PERIOD          ((volatile unsigned long *)TIMER2_PERIOD)
#define pTIMER2_WIDTH           ((volatile unsigned long *)TIMER2_WIDTH)

//## WATCHDOG TIMER PROGRAMMABLE REGISTERS

#define pWDOG_CTL               ((volatile unsigned short *)WDOG_CTL)
#define pWDOG_CNT               ((volatile unsigned long *)WDOG_CNT)
#define pWDOG_STAT              ((volatile unsigned long *)WDOG_STAT)

//## REAL-TIME CLOCK PROGRAMMABLE REGISTERS

#define pRTC_STAT               ((volatile unsigned long *)RTC_STAT)
#define pRTC_ICTL               ((volatile unsigned short *)RTC_ICTL)
#define pRTC_ISTAT              ((volatile unsigned short *)RTC_ISTAT)
#define pRTC_SWCNT              ((volatile unsigned short *)RTC_SWCNT)
#define pRTC_ALARM              ((volatile unsigned long *)RTC_ALARM)
#define pRTC_PREN               ((volatile unsigned short *)RTC_PREN)

//## EBIU PROGRAMMABLE REGISTERS

#define pEBIU_AMGCTL            ((volatile unsigned short *)EBIU_AMGCTL)
#define pEBIU_AMBCTL0           ((volatile unsigned long *)EBIU_AMBCTL0)
#define pEBIU_AMBCTL1           ((volatile unsigned long *)EBIU_AMBCTL1)
#define pEBIU_SDGCTL            ((volatile unsigned long *)EBIU_SDGCTL)
#define pEBIU_SDBCTL            ((volatile unsigned short *)EBIU_SDBCTL)
#define pEBIU_SDSTAT            ((volatile unsigned short *)EBIU_SDSTAT)
#define pEBIU_SDRRC             ((volatile unsigned short *)EBIU_SDRRC)

//## CORE EVENT CONTROLLER REGISTERS

#define pIMASK                  ((volatile unsigned long *)IMASK)
#define pIPEND                  ((volatile unsigned long *)IPEND)
```

```
#define pILAT                   ((volatile unsigned long *)ILAT)
#define pIPRIO                  ((volatile unsigned long *)IPRIO)

//## CORE EVENT VECTOR TABLE REGISTERS
#define pEVT0                   ((void * volatile *)EVT0)
#define pEVT1                   ((void * volatile *)EVT1)
#define pEVT2                   ((void * volatile *)EVT2)
#define pEVT3                   ((void * volatile *)EVT3)
#define pEVT4                   ((void * volatile *)EVT4)
#define pEVT5                   ((void * volatile *)EVT5)
#define pEVT6                   ((void * volatile *)EVT6)
#define pEVT7                   ((void * volatile *)EVT7)
#define pEVT8                   ((void * volatile *)EVT8)
#define pEVT9                   ((void * volatile *)EVT9)
#define pEVT10                  ((void * volatile *)EVT10)
#define pEVT11                  ((void * volatile *)EVT11)
#define pEVT12                  ((void * volatile *)EVT12)
#define pEVT13                  ((void * volatile *)EVT13)
#define pEVT14                  ((void * volatile *)EVT14)
#define pEVT15                  ((void * volatile *)EVT15)

//## CORE TIMER PROGRAMMABLE REGISTERS

#define pTCNTL                  ((volatile unsigned long *)TCNTL)
#define pTPERIOD                ((volatile unsigned long *)TPERIOD)
#define pTSCALE                 ((volatile unsigned long *)TSCALE)
#define pTCOUNT                 ((volatile unsigned long *)TCOUNT)

//########################################
//#                                      #
//#    SELECTED BITMASK DECLARATIONS     #
//#                                      #
//#                                      #
//########################################

// LED BITMASKS
#define LED1_BITMASK            0x1
#define LED2_BITMASK            0x2
#define LED3_BITMASK            0x4
#define LED4_BITMASK            0x8
#define LED5_BITMASK            0x10
#define LED6_BITMASK            0x20

// SWITCH BITMASKS

#define SW1_BITMASK             0x1
#define SW2_BITMASK             0x2
#define SW3_BITMASK             0x4
#define SW4_BITMASK             0x8

// OTHERS CAN BE DEFINED BELOW IF NECESSARY


//########################################
//#                                      #
//#    LIBRARY FUNCTION DECLARATIONS     #
//#                                      #
//#                                      #
//########################################

// TTCOS FUNCTION DECLARATIONS WILL BE PLACED BELOW

// Initialize Flash Memory
void Init_Flash();

// Initialize LEDS
void Init_LED();

// Read LEDS
unsigned char Read_LED();

// Write LEDS
void Write_LED(unsigned char MASK);
```

```
// Initialize GPIO Switches
void Init_Switches();

// Enable GPIO Switches
void Enable_Switches();

// Disable GPIO Switches
void Disable_Switches();

// Read Switches
unsigned short int Read_Switches();

// Stop Watchdog Interrupts
void Stop_WDOG_Int();
// Setup Watchdog Interrupts (Passes Function, and Period)
void Setup_WDOG_Int(void (*isr_func)(void), unsigned long period );
// Start Watchdog Interrupts
void Start_WDOG_Int();

// Stop Core Timer Interrupts
void Stop_CoreTimer_Int();
// Setup Watchdog Interrupts (Passes Function, and Period)
void Setup_CoreTimer_Int(void (*isr_func)(void), unsigned long period );
// Start Watchdog Interrupts
void Start_CoreTimer_Int();

// Stop Switch Interrupts
void Stop_GPIO_Int();
// Setup Switch Interrupts
void Setup_GPIO_Int(void (*isr_func)(void));
// Start Switch Interrupts
void Start_GPIO_Int();

// SSYNC() Declaration
#define Ssync() \
    asm("ssync;")

//CSYNC() Declaration
//#define csync() \
    asm("csync;")

//IDLE() Declaration
//#define idle() \
    asm("idle")


#endif
```

Blackfin Library Source - *myBlackfin.cpp*

The following is my Blackfin Library source file outline all of my library functions. Many of which are based off the uTTCOS functions required for this assignment.

```
/*********************************\
*   ENCM 511 - Assignment 2       *
*                                 *
*    Author: Kyle Derby MacInnis  *
*    Date:   November 5, 2014     *
*                                 *
***********************************
*       myBlackfin.cpp      *
\*********************************/

#include "myBlackfin.h"

// Initialize Blackfin Flash Memory (For LEDS)
void Init_Flash()
{
    // Set Read and Write Times for Memory Banks 0 and 1
```

```
    *pEBIU_AMBCTL0 = (0x7bb07bb0);
    // Set Read and Write Times for Memory Banks 2 and 3
    *pEBIU_AMBCTL1 = (0x7bb07bb0);
    // Enable all Memory Banks
    *pEBIU_AMGCTL = (0x000F);
    // Set LEDS 1-6 for Output
    *pLED_IO_DIR = 0x3F;
    // Force Write
    Ssync();
}

// Initialize Blackfin LEDS
void Init_LED()
{
    // Set LEDS Off
    *pLED_IO_SET = 0x0;
    //Force Write
    Ssync();
}

// Read Blackfin LEDS
unsigned char Read_LED()
{
    // Read in Value from Register
    unsigned char currentValue = (*pLED_IO_STAT & 0x3F);
    // Return Value
    return currentValue;
}

// Write Blackfin LEDS
void Write_LED(unsigned char MASK)
{
    // Clear LEDS
    *pLED_IO_SET = 0x0;
    // Write Given MASK to LEDS
    *pLED_IO_SET = MASK & 0x3F;
    // Force Write
    Ssync();
}

// Initialize Blackfin Switches
void Init_Switches()
{
    // Set Polarity of Switches to Active High
    *pFIO_POLAR &= (~0x0F00);
    // Set Sensitivity to Level
    *pFIO_EDGE &= (~0x0F00);
    // Set Direction of Switches to Inputs
    *pFIO_DIR &= (~0x0F00);
    // Set MASKA and MASKB Interrupts Off
    *pFIO_MASKA_S &= (~0x0F00);
    *pFIO_MASKB_S &= (~0x0F00);
    // Set Input Enabled
    *pFIO_INEN |= (0x0F00);
    // Force Write
    Ssync();
}

// Enable Switches
void Enable_Switches()
{
    // Set Switch Input Enabled
    *pFIO_INEN |= (0x0F00);
    // Force Write
    Ssync();
}

// Disable Switches
void Disable_Switches()
{
    // Set Switch Input Disabled
    *pFIO_INEN &= (~0x0F00);
    // Force Write
```

```
    Ssync();
}

// Read Blackfin Switches
unsigned short int Read_Switches()
{
    // Read in Switches from GPIO and Bitshift 8 Bits
    unsigned short currentValue = ((*pFIO_FLAG_D & 0x0F00) >> 8);
    // Return Value
    return currentValue;
}

// Stop Watchdog Interrupts
void Stop_WDOG_Int()
{
    // Stop WDOG Counter
    *pWDOG_CTL = (0x0AD0) | (0x0004);
    *pWDOG_CNT &= (0xFFFF);
    // Force Write
    Ssync();
}

// Setup Watchdog Interrupts (Passes Function, and Period)
void Setup_WDOG_Int(void (*isr_func)(), unsigned long period )
{
    // Reset Watchdog Timer - And Map GP interrupt
    *pWDOG_CTL = 0x0AD0 | (0x0004);
    // Set Period of Interrupt
    *pWDOG_CNT = period;
    // Enable WDOG Interrupt
    *pSIC_IMASK |= (0x00800000);
    // Enable Corresponding EVT Interrupt (IVG8)
    *pIMASK |= (0x00000100);
    // Enable Correspondong SIC_IAR Register (IVG8)
    *pSIC_IAR2 |= (0x10000000);
    // Set given ISR to EVT
    *pEVT8 = isr_func;
    // Force Write
    Ssync();
}

// Start Watchdog Interrupts
void Start_WDOG_Int()
{
    // Turn on WDOG Timer
    *pWDOG_CTL &= (~0x0FF0);
    // Force Write
    Ssync();
}

// Stop Watchdog Interrupts
void Stop_CoreTimer_Int()
{
    // Stop WDOG Counter
    *pTCNTL &= (~0x00000002);
    // Force Write
    Ssync();
}

void Setup_CoreTimer_Int(void (*isr_func)(), unsigned long period )
{
    // Set Auto Reload and Set Active State
    *pTCNTL = 0x00000005;
    // Set Period of Interrupt
    *pTPERIOD = period;
    // Set Scale to 1 Cycle Decrement
    *pTSCALE = 0x0;
    // Enable Corresponding EVT Interrupt (IVTMR)
    *pIMASK |= (0x00000040);
    // Set given ISR to EVT
    *pEVT6 = isr_func;
    // Force Write
    Ssync();
```

```
}

// Start Core Timer Interrupts
void Start_CoreTimer_Int()
{
    // Turn on Core Timer
    *pTCNTL |= 0x00000002;
    // Force Write
    Ssync();
}

// Stop Switch Interrupts
void Stop_GPIO_Int()
{
    // Disable Interrupts for PF8-PF11
    *pFIO_MASKA_S &= (~0x0F00);
    // Disable SIC_IMASK Interrupt
    *pSIC_IMASK &= (~0x00080000);
    // Disable IMASK Interrupt
    *pIMASK &= (~0x00000080);
    // Force Write
    Ssync();
}

// Setup Switch Interrupts
void Setup_GPIO_Int(void (*isr_func)(void))
{
    // Set GPIO to be Edge Triggered
    *pFIO_EDGE |= 0x0F00;
    // Set Interrupts for Both Rising/Falling
    *pFIO_BOTH |= 0x0F00;
    // Enable GPIO Interrupt
    *pSIC_IMASK |= 0x00080000;
    // Enable IVG7 Interrupt
    *pIMASK |= (0x00000080);
    // Enable Corresponding SIC_IAR2 Register (EVT7)
    *pSIC_IAR2 &= (~0x0000F000);
    // Set Given ISR to EVT7
    *pEVT7 = isr_func;
    // Force Write
    Ssync();
}

// Start Switch Interrupts
void Start_GPIO_Int()
{
    // Enable Interrupts for PF8-PF11
    *pFIO_MASKA_S |= (0x0F00);
    // Force Write
    Ssync();
}
```