**Coffee Pot Assignment 2 Code:**

**Aidan Johnson**

**EUNIT Code**

**EUNIT 533 Code:**

**EUNITAssignment2533\_EUNIT2017\_main.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* AUTOMATICALLY GENERATED COMMENT -- DO NOT MODIFY

\* Author: aidan.johnson1

\* Date: Mon 2019/11/25 at 02:15:13 PM

\* File Type: EUNIT Main File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <EmbeddedUnit2017/EmbeddedUnit2017.h>

#include "TestingAssignment2533.h"

void UpdateEunitGui(void);

extern volatile int useLongFileFormat;

extern void AutomatedTestLevelControl(void);

void RestartEunitGui(void);

void UpdateEunitGui(void);

int main(void)

{

int failureCount;

RestartEunitGui( );

UpdateEunitGui();

UnitTest::ProcessorSpecificStartup();

AutomatedTestLevelControl();

UnitTest::Test::GetTestList().ReverseListDirection();

bool showFail = true; bool showXFail = true;

bool showSuccesses = true;

// TODO You can adjust UnitTest::RunAllTests( ) parameters to show only failures -- Wed 2018/09/26 at 08:14:10 PM

// TODO by setting bool showSuccesses = false;;

failureCount = UnitTest::RunAllTests(showFail, showXFail, showSuccesses);

UpdateEunitGui();

return failureCount;

}

**TestingAssignment2533.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* AUTOMATICALLY GENERATED COMMENT -- DO NOT MODIFY

\* Author: aidan.johnson1

\* Date: Mon 2019/11/25 at 02:15:13 PM

\* File Type: EUNIT Test File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define EMBEDDEDUNIT\_LITE

#include <EmbeddedUnit2017/EmbeddedUnit2017.h>

#include "TestingAssignment2533.h"

TEST\_CONTROL(TestingAssignment2533\_cpp);

#if 1

void UpdateEunitGui(void);

TEST(TestingAssignment2533\_cpp\_GUIUpdate) {

UpdateEunitGui(); // Conditionally compile this line (use #if 0) to stop an GUI update based on last completed test

}

#endif

unsigned short int TestBitwiseAND(unsigned short int bitPattern, unsigned short int bitMask);

unsigned short int TestBitwiseOR(unsigned short int bitPattern, unsigned short int bitMask);

#if 0

TEST(TestingAllFourFunctionsForCoreTimer\_MoreComplexTest)

{

#warning 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:15:13 PM '

// TODO -- 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:15:13 PM '

printf("Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:15:13 PM \n");

unsigned long int value = 0x01FF01FF;

unsigned long int ORmask = 0x0F000F0F;

unsigned long int expectedORResult = 0x0100010F;

unsigned long int resultOR = TestBitwiseOR(value, ORmask);

CHECK(expectedORResult == resultOR);

CHECK\_EQUAL(expectedORResult, resultOR);

#error("You insert the 'wrong' test for TestBitwiseAND";

}

unsigned short int TestBitwiseAND(unsigned short int bitPattern, unsigned short int bitMask) {

return bitPattern && bitMask;

}

unsigned short int TestBitwiseOR(unsigned short int bitPattern, unsigned short int bitMask) {

return bitPattern || bitMask;

}

TEST(TestingAllFourFunctionsForCoreTimer\_Successes)

{

#warning 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:15:13 PM '

// TODO -- 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:15:13 PM '

printf("Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:15:13 PM \n");

CHECK(false == false);

CHECK\_EQUAL(false, false);

XF\_CHECK(false == true); // Expected failure occurs

XF\_CHECK\_EQUAL(false, true); // Expected failure occurs

XF\_CHECK(false == false); // Expected failure does not occur

XF\_CHECK\_EQUAL(false, false); // Expected failure does not occur

#error("You insert the 'wrong' test for TestBitwiseAND";

}

#endif

TEST(TestingAllFourFunctionsForCoreTimer)

{

//Checking if the core timer was properly initialized

myInit\_CoreTimer(PERIOD, COUNT);

unsigned long int coreTimerControlRegister = CORETIMEPOWERBIT;

unsigned long int coreTimerPeriodRegister = PERIOD;

unsigned long int coreTimerCountRegister = COUNT;

unsigned long int coreTimerScaleRegister = 0x0;

#if CHECKFORSUCCESS

CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

CHECK\_EQUAL(coreTimerPeriodRegister, \*pTPERIOD);

CHECK\_EQUAL(coreTimerCountRegister, \*pTCOUNT);

#else

XF\_CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

XF\_CHECK\_EQUAL(coreTimerPeriodRegister, \*pTPERIOD);

XF\_CHECK\_EQUAL(coreTimerCountRegister, \*pTCOUNT);

#endif

//Checking if the core timer was enabled properly

myControl\_CoreTimer((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT)); //Enabling the Core Timer

coreTimerControlRegister = ((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

#if CHECKFORSUCCESS

CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#else

XF\_CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#endif

//Checking if the core timer went off after the count

myTimedWaitOnCoreTimer();

coreTimerControlRegister = ((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

#if CHECKFORSUCCESS

CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#else

XF\_CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#endif

myControl\_CoreTimer(TCNTLBIT2TO0);

\*pTCOUNT = PERIOD;

//Checking if the Core timer is complete

bool isCoreTimerComplete = false;

while(!myCompleted\_CoreTimer())

{}

isCoreTimerComplete = myCompleted\_CoreTimer();

#if CHECKFORSUCCESS

CHECK\_EQUAL(true, isCoreTimerComplete);

#else

XF\_CHECK\_EQUAL(true, isCoreTimerComplete);

#endif

//Checking if the core timer is for the array

unsigned long int beforeReadCyclesValue;

unsigned long int differenceInCyclesValue;

unsigned long int afterReadCyclesValue;

unsigned long int cyclesArray[ARRAYLENGTH];

unsigned long int theoreticalCyclesArray[ARRAYLENGTH];

unsigned long int sum = 0;

unsigned long int average;

myControl\_CoreTimer((TCNTLTAUTORLDBIT | TCNTLTMRENBIT | CORETIMEPOWERBIT)); //Turning on the AutoReload bit

stopCoreTimer();

resetCoreTimerInterrupt();

myControl\_CoreTimer(TCNTLBIT2TO0);

\*pTCOUNT = PERIOD;

for(int i = 0; i < ARRAYLENGTH; i++)

{

myControl\_CoreTimer((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

beforeReadCyclesValue = ReadProcessorCyclesASM();

myTimedWaitOnCoreTimer();

myControl\_CoreTimer(TCNTLBIT2TO0);

resetCoreTimerInterrupt();

afterReadCyclesValue = ReadProcessorCyclesASM();

differenceInCyclesValue = afterReadCyclesValue - beforeReadCyclesValue;

cyclesArray[i] = differenceInCyclesValue;

sum += differenceInCyclesValue;

}

#if CHECKFORSUCCESS

average = (sum/ARRAYLENGTH);

for(int i = 0; i < ARRAYLENGTH; i++)

{

theoreticalCyclesArray[i] = average;

}

#endif

for(int i = 0; i < ARRAYLENGTH; i++)

{

#if CHECKFORSUCCESS

CHECK\_CLOSE(theoreticalCyclesArray[i], cyclesArray[i], (average/100));

#else

XF\_CHECK\_EQUAL(PERIOD, cyclesArray[i]);

#endif

}

//Checking the Polling

unsigned long int coreTimerControlValue = \*pTCNTL;

unsigned long int pollingArray[ARRAYLENGTH];

unsigned long int theoreticalPollingArray[ARRAYLENGTH];

sum = 0;

myControl\_CoreTimer(TCNTLBIT2TO0);

\*pTCOUNT = PERIOD;

for (int i = 0; i < ARRAYLENGTH; i++)

{

myControl\_CoreTimer((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

beforeReadCyclesValue = ReadProcessorCyclesASM();

while(!myCompleted\_CoreTimer())

{

}

myControl\_CoreTimer(TCNTLBIT2TO0);

afterReadCyclesValue = ReadProcessorCyclesASM();

differenceInCyclesValue = afterReadCyclesValue - beforeReadCyclesValue;

pollingArray[i] = differenceInCyclesValue;

sum += differenceInCyclesValue;

}

#if CHECKFORSUCCESS

average = (sum/ARRAYLENGTH);

for(int i = 0; i < ARRAYLENGTH; i++)

{

theoreticalPollingArray[i] = average;

}

#endif

for(int i = 0; i < ARRAYLENGTH; i++)

{

#if CHECKFORSUCCESS

CHECK\_CLOSE(theoreticalPollingArray[i], pollingArray[i], (average/100));

#else

XF\_CHECK\_EQUAL(PERIOD, pollingArray[i]);

#endif

}

}

void myInit\_CoreTimer(unsigned long int period, unsigned long int count)

{

#if NOTSTUBS

\*pTCNTL = CORETIMEPOWERBIT;

\*pTPERIOD = period;

\*pTCOUNT = count;

\*pTSCALE = 0x00000000;

#else

//printf("Stub for myInit\_CoreTimer \n");

#endif

}

void myControl\_CoreTimer(unsigned short int cntrl\_value)

{

#if NOTSTUBS

cntrl\_value = cntrl\_value & 0x0000000f;

\*pTCNTL = cntrl\_value;

#else

//printf("Stub for myControl\_CoreTimer \n");

#endif

}

bool myCompleted\_CoreTimer(void)

{

#if NOTSTUBS

bool returnValue = false;

unsigned long int coreTimerControlValue = \*pTCNTL;

if((coreTimerControlValue & TCNTLBIT3TO0) == TCNTLBIT3TO0)

{

returnValue = true;

}

return returnValue;

#else

//printf("Stub for myCompleted\_CoreTimer \n");

#endif

}

void myTimedWaitOnCoreTimer(void)

{

#if NOTSTUBS

bool coreTimerInitialized = false;

while(!coreTimerInitialized)

{

unsigned long int stickyBitTCNTL = (\*pTCNTL & TCNTLDONEBIT);

if(TCNTLDONEBIT == stickyBitTCNTL)

{

\*pTCNTL = \*pTCNTL & (~TCNTLTINTBIT);

coreTimerInitialized = true;

}

}

return;

#else

//printf("Stub for myTimedWaitOnCoreTimer \n");

#endif

}

void startCoreTimer(void)

{

unsigned long int cntrl\_value = TCNTLTMRENBIT;

\*pTCNTL = (\*pTCNTL | cntrl\_value);

}

void stopCoreTimer(void)

{

unsigned long int cntrl\_value = (~TCNTLTMRENBIT);

\*pTCNTL = (\*pTCNTL & (cntrl\_value));

}

void resetCoreTimerInterrupt(void)

{

\*pTCNTL = \*pTCNTL & (~TCNTLTINTBIT);

}

TEST\_FILE\_RUN\_NOTIFICATION(TestingAssignment2533\_cpp);

**TestingAssignment2533.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* AUTO-GENERATED COMMENT - DO NOT MODIFY

\* Author: aidan.johnson1

\* Date: Mon 2019/11/25 at 02:15:13 PM

\* File Type: EUNIT Test Header File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef TESTINGASSIGNMENT2533\_H

#define TESTINGASSIGNMENT2533\_H

#include <blackfin.h>

void UpdateEunitGui(void); // Update EUNIT GUI with results from previous test

//MACROS

#define CORETIMEPOWERBIT 0x00000001

#define CORETIMERAUTORLD 0x00000008

#define TCNTLBITS3AND1TO0 0x0000000b

#define TCNTLDONEBIT 0x00000008

#define NOTSTUBS 0

#define PERIOD 100000

#define COUNT 100000

#define TCNTLTMRENBIT 0x00000002

#define TCNTLTINTBIT 0x00000008

#define TCNTLTAUTORLDBIT 0x00000004

#define ARRAYLENGTH 10

#define CHECKFORSUCCESS 0

#define TCNTLBIT2TO0 0x00000007

#define TCNTLBIT3TO0 0x0000000f

//Prototypes for CoreTimer Functions for BF533

void myInit\_CoreTimer(unsigned long int period, unsigned long int count);

void myControl\_CoreTimer(unsigned short int cntrl\_value);

bool myCompleted\_CoreTimer(void);

void myTimedWaitOnCoreTimer(void);

void startCoreTimer(void);

void stopCoreTimer(void);

void resetCoreTimerInterrupt(void);

extern "C" unsigned long long int ReadProcessorCyclesASM(void);

#endif

**ReadProcessorCyclesASM.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;

**EUNIT 609 Code:**

**EUNITAssignment2\_609\_Core0\_EUNIT2017\_main.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* AUTOMATICALLY GENERATED COMMENT -- DO NOT MODIFY

\* Author: aidan.johnson1

\* Date: Mon 2019/11/25 at 02:25:33 PM

\* File Type: EUNIT Main File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <EmbeddedUnit2017/EmbeddedUnit2017.h>

#include "EUNITAssignment2609.h"

void UpdateEunitGui(void);

extern volatile int useLongFileFormat;

extern void AutomatedTestLevelControl(void);

void RestartEunitGui(void);

void UpdateEunitGui(void);

int main(void)

{

int failureCount;

RestartEunitGui( );

UpdateEunitGui();

UnitTest::ProcessorSpecificStartup();

AutomatedTestLevelControl();

UnitTest::Test::GetTestList().ReverseListDirection();

bool showFail = true; bool showXFail = true;

bool showSuccesses = true;

// TODO You can adjust UnitTest::RunAllTests( ) parameters to show only failures -- Wed 2018/09/26 at 08:14:10 PM

// TODO by setting bool showSuccesses = false;;

failureCount = UnitTest::RunAllTests(showFail, showXFail, showSuccesses);

UpdateEunitGui();

return failureCount;

}

**EUNITAssignment2609.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* AUTOMATICALLY GENERATED COMMENT -- DO NOT MODIFY

\* Author: aidan.johnson1

\* Date: Mon 2019/11/25 at 02:25:33 PM

\* File Type: EUNIT Test File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define EMBEDDEDUNIT\_LITE

#include <EmbeddedUnit2017/EmbeddedUnit2017.h>

#include "EUNITAssignment2609.h"

TEST\_CONTROL(EUNITAssignment2609\_cpp);

#if 1

void UpdateEunitGui(void);

TEST(EUNITAssignment2609\_cpp\_GUIUpdate) {

UpdateEunitGui(); // Conditionally compile this line (use #if 0) to stop an GUI update based on last completed test

}

#endif

unsigned short int TestBitwiseAND(unsigned short int bitPattern, unsigned short int bitMask);

unsigned short int TestBitwiseOR(unsigned short int bitPattern, unsigned short int bitMask);

#if 0

TEST(TestingAllFourCoreTimerFunctions\_MoreComplexTest)

{

#warning 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:25:33 PM '

// TODO -- 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:25:33 PM '

printf("Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:25:33 PM \n");

unsigned long int value = 0x01FF01FF;

unsigned long int ORmask = 0x0F000F0F;

unsigned long int expectedORResult = 0x0100010F;

unsigned long int resultOR = TestBitwiseOR(value, ORmask);

CHECK(expectedORResult == resultOR);

CHECK\_EQUAL(expectedORResult, resultOR);

#error("You insert the 'wrong' test for TestBitwiseAND";

}

unsigned short int TestBitwiseAND(unsigned short int bitPattern, unsigned short int bitMask) {

return bitPattern && bitMask;

}

unsigned short int TestBitwiseOR(unsigned short int bitPattern, unsigned short int bitMask) {

return bitPattern || bitMask;

}

TEST(TestingAllFourCoreTimerFunctions\_Successes)

{

#warning 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:25:33 PM '

// TODO -- 'Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:25:33 PM '

printf("Dummy test has been inserted -- replace with your own -- Mon 2019/11/25 at 02:25:33 PM \n");

CHECK(false == false);

CHECK\_EQUAL(false, false);

XF\_CHECK(false == true); // Expected failure occurs

XF\_CHECK\_EQUAL(false, true); // Expected failure occurs

XF\_CHECK(false == false); // Expected failure does not occur

XF\_CHECK\_EQUAL(false, false); // Expected failure does not occur

#error("You insert the 'wrong' test for TestBitwiseAND";

}

#endif

TEST(TestingAllFourCoreTimerFunctions)

{

//Checking if the core timer was properly initialized

Init\_CoreTimer(PERIOD, COUNT);

unsigned long int coreTimerControlRegister = CORETIMEPOWERBIT;

unsigned long int coreTimerPeriodRegister = PERIOD;

unsigned long int coreTimerCountRegister = COUNT;

unsigned long int coreTimerScaleRegister = 0x0;

#if CHECKFORSUCCESS

CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

CHECK\_EQUAL(coreTimerPeriodRegister, \*pTPERIOD);

CHECK\_EQUAL(coreTimerCountRegister, \*pTCOUNT);

#else

XF\_CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

XF\_CHECK\_EQUAL(coreTimerPeriodRegister, \*pTPERIOD);

XF\_CHECK\_EQUAL(coreTimerCountRegister, \*pTCOUNT);

#endif

//Checking if the core timer was enabled properly

Control\_CoreTimer((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT)); //Enabling the Core Timer

coreTimerControlRegister = ((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

\*pTCOUNT = PERIOD;

#if CHECKFORSUCCESS

CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#else

XF\_CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#endif

//Checking if the core timer went off after the count

TimedWaitOnCoreTimer();

ssync();

coreTimerControlRegister = ((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

#if CHECKFORSUCCESS

CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#else

XF\_CHECK\_EQUAL(coreTimerControlRegister, \*pTCNTL);

#endif

Control\_CoreTimer(TCNTLBIT2TO0);

\*pTCOUNT = PERIOD;

//Checking if the Core timer is complete

bool isCoreTimerComplete = false;

while(!Completed\_CoreTimer())

{}

isCoreTimerComplete = Completed\_CoreTimer();

ssync();

Control\_CoreTimer(TCNTLBIT2TO0);

#if CHECKFORSUCCESS

CHECK\_EQUAL(true, isCoreTimerComplete);

#else

XF\_CHECK\_EQUAL(true, isCoreTimerComplete);

#endif

//Checking if the core timer is for the array

unsigned long int beforeReadCyclesValue;

unsigned long int differenceInCyclesValue;

unsigned long int afterReadCyclesValue;

unsigned long int cyclesArray[ARRAYLENGTH];

unsigned long int theoreticalCyclesArray[ARRAYLENGTH];

unsigned long int sum = 0;

unsigned long int average;

\*pTCOUNT = PERIOD;

for(int i = 0; i < ARRAYLENGTH; i++)

{

Control\_CoreTimer((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

beforeReadCyclesValue = ReadProcessorCyclesASM();

TimedWaitOnCoreTimer();

Control\_CoreTimer(TCNTLBIT2TO0);

afterReadCyclesValue = ReadProcessorCyclesASM();

differenceInCyclesValue = afterReadCyclesValue - beforeReadCyclesValue;

cyclesArray[i] = differenceInCyclesValue;

sum += differenceInCyclesValue;

}

#if CHECKFORSUCCESS

average = (sum/ARRAYLENGTH);

for(int i = 0; i < ARRAYLENGTH; i++)

{

theoreticalCyclesArray[i] = average;

}

#endif

for(int i = 0; i < ARRAYLENGTH; i++)

{

#if CHECKFORSUCCESS

CHECK\_CLOSE(theoreticalCyclesArray[i], cyclesArray[i], (average/100));

#else

XF\_CHECK\_EQUAL(PERIOD, cyclesArray[i]);

#endif

}

//Checking the Polling

unsigned long int coreTimerControlValue = \*pTCNTL;

unsigned long int pollingArray[ARRAYLENGTH];

unsigned long int theoreticalPollingArray[ARRAYLENGTH];

sum = 0;

Control\_CoreTimer(TCNTLBIT2TO0);

\*pTCOUNT = PERIOD;

for (int i = 0; i < ARRAYLENGTH; i++)

{

Control\_CoreTimer((TCNTLTMRENBIT | CORETIMEPOWERBIT | TCNTLTAUTORLDBIT));

beforeReadCyclesValue = ReadProcessorCyclesASM();

while(!Completed\_CoreTimer())

{

}

Control\_CoreTimer(TCNTLBIT2TO0);

afterReadCyclesValue = ReadProcessorCyclesASM();

differenceInCyclesValue = afterReadCyclesValue - beforeReadCyclesValue;

pollingArray[i] = differenceInCyclesValue;

sum += differenceInCyclesValue;

}

#if CHECKFORSUCCESS

average = (sum/ARRAYLENGTH);

for(int i = 0; i < ARRAYLENGTH; i++)

{

theoreticalPollingArray[i] = average;

}

#endif

for(int i = 0; i < ARRAYLENGTH; i++)

{

#if CHECKFORSUCCESS

CHECK\_CLOSE(theoreticalPollingArray[i], pollingArray[i], (average/100));

#else

XF\_CHECK\_EQUAL(PERIOD, pollingArray[i]);

#endif

}

}

TEST\_FILE\_RUN\_NOTIFICATION(EUNITAssignment2609\_cpp);

**EUNITAssignment2609.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* AUTO-GENERATED COMMENT - DO NOT MODIFY

\* Author: aidan.johnson1

\* Date: Mon 2019/11/25 at 02:25:33 PM

\* File Type: EUNIT Test Header File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef EUNITASSIGNMENT2609\_H

#define EUNITASSIGNMENT2609\_H

#include <blackfin.h>

#include "../../ENCM511\_SpecificFiles/ENCM511\_include/CoreTimer\_Library.h"

void UpdateEunitGui(void); // Update EUNIT GUI with results from previous test

//MACROS

#define CORETIMEPOWERBIT 0x00000001

#define CORETIMERAUTORLD 0x00000008

#define TCNTLBITS3AND1TO0 0x0000000b

#define TCNTLDONEBIT 0x00000008

#define NOTSTUBS 0

#define PERIOD 100000

#define COUNT 100000

#define TCNTLTMRENBIT 0x00000002

#define TCNTLTINTBIT 0x00000008

#define TCNTLTAUTORLDBIT 0x00000004

#define ARRAYLENGTH 10

#define CHECKFORSUCCESS 1

#define TCNTLBIT2TO0 0x00000007

extern "C" unsigned long long int ReadProcessorCyclesASM(void);

#endif

**Non-EUNIT Assignment 2 Code**

**609 Code:**

**Assignment2CoffeePot\_BF609\_Core0.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Assignment2CoffeePot\_BF609\_Core0.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <sys/platform.h>

#include <sys/adi\_core.h>

#include <ccblkfn.h>

#include "adi\_initialize.h"

#include "Assignment2CoffeePot\_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);

#ifdef \_\_ADSPBF533\_\_

printf("Starting the Coffee Pot Assignment\n");

startCoffeePot();

#endif

#ifdef \_\_ADSPBF609\_\_

printf("Starting the Coffee Pot Assignment\n");

startCoffeePot();

#endif

return 0;

}

**Assignment2CoffeePot\_BF609\_Core0.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Assignment2CoffeePot\_BF609\_Core0.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_ASSIGNMENT2COFFEEPOT\_BF609\_CORE0\_H\_\_

#define \_\_ASSIGNMENT2COFFEEPOT\_BF609\_CORE0\_H\_\_

/\* Add your custom header content here \*/

#include "../../ENCM511\_SpecificFiles/ENCM511\_include/CoffeePot\_SimulatorFunctions2017.h"

#include <stdio.h>

#include <blackfin.h>

#include <sys/exception.h>

#include "Lab1And0FrontPanelFunctions.h"

#include "../../ENCM511\_SpecificFiles/ENCM511\_include/FrontPanel\_LED\_Switches.h"

#include "../../ENCM511\_SpecificFiles/ENCM511\_include/REB\_GPIO\_Input\_library.h"

#include "../../ENCM511\_SpecificFiles/ENCM511\_include/REB\_GPIO\_Output\_library.h"

//Prototypes for Coffee Pot Assignment 1 Functions

void startCoffeePot(void);

void showNameOfProcessorUsed(void);

void DemonstrateLEDControl\_CPP(void);

void initializingCoffeePot(COFFEEPOT\_DEVICE\* coffeePot[]);

void activateLEDControl(COFFEEPOT\_DEVICE \*coffeePot[]);

void activateWaterControl(COFFEEPOT\_DEVICE \*coffeePot[]);

void activateHeatControl(COFFEEPOT\_DEVICE \*coffeePot[]);

void LEDControlDemo(COFFEEPOT\_DEVICE \*coffeePot);

void fillCoffeePotToWaterLEvel(COFFEEPOT\_DEVICE \*coffeePot);

void heatWaterToTemperature(COFFEEPOT\_DEVICE \*coffeePot);

void checkForCoffeePod(COFFEEPOT\_DEVICE \*coffeePot);

//Prototypes for CoreTimer Functions for BF533

void myInit\_CoreTimer(unsigned long int period, unsigned long int count);

void myControl\_CoreTimer(unsigned short int cntrl\_value);

bool myCompleted\_CoreTimer(void);

void myTimedWaitOnCoreTimer(void);

void startCoreTimer(void);

void stopCoreTimer(void);

void resetCoreTimerInterrupt(void);

//Prototypes for Interrupt Functions

void interruptServiceRoutineFastForward(void);

void startCoreTimerInterrupts(void);

void stopCoreTimerInterrupts(void);

//Prototypes for Requirement1

void showingCoffeePot1LEDSOnFrontPanelAndSimulationLEDS(COFFEEPOT\_DEVICE \*coffeePot);

void showingCoffeePot2LEDSOnREBAndSimulationLEDS(COFFEEPOT\_DEVICE \*coffeePot);

//Prototypes for Requirement2

void coffeePot1OverWritingControlRegisterBits(COFFEEPOT\_DEVICE \*coffeePot);

void coffeePot2OverWritingControlRegisterBits(COFFEEPOT\_DEVICE \*coffeePot);

//These variables are all declared and initialized to be false to begin until the respective function is called

//to initialize the respective equipment

extern bool My\_Init\_SwitchInterface\_Done;

extern bool My\_Init\_LEDInterface\_Done;

extern bool My\_Init\_GPIO\_REB\_Input\_Done;

extern bool My\_Init\_GPIO\_REB\_Output\_Done;

extern bool My\_Init\_GPIO\_REB\_Done;

//Function Prototypes

void My\_Init\_SwitchInterface(void);

void My\_Init\_LEDInterface(void);

void My\_Init\_GPIO\_REB\_Input(void);

void My\_Init\_GPIO\_REB\_Output(void);

void My\_Write\_REB\_LED(unsigned short int);

unsigned char My\_ReadSwitches(void);

unsigned short int My\_Read\_REB\_Switches(void);

void WaitTillSwitch5PressedAndReleased(void);

//Prototypes for Coffee Pot Assignment 1 Assembly Functions as well as from Lab 1

extern "C" void demonstrateLEDASM(COFFEEPOT\_DEVICE \*coffeePot);

extern "C" void checkForCoffeePodASM(COFFEEPOT\_DEVICE \*coffeePot);

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 /\* \_\_ASSIGNMENT2COFFEEPOT\_BF609\_CORE0\_H\_\_ \*/

**checkForCoffeePodASM.asm**

/\*

\* checkForCoffeePodASM.asm

\*

\* Created on: Oct 30, 2019

\* Author: aidan

\*/

#include <blackfin.h>

.section L1\_data;

.section program;

#define INCOMING\_PAR R0

#define TEMPERATURE 95

#define COFFEEPOD 0x0800(Z)

.global \_checkForCoffeePodASM;

\_checkForCoffeePodASM:

[--SP] = R4;

[--SP] = P4;

LINK 16;

P4 = INCOMING\_PAR;

R4 = W[P4](Z);

R0 = P4;

.extern \_\_Z22CurrentTemperature\_CPPP16COFFEEPOT\_DEVICE;

CALL \_\_Z22CurrentTemperature\_CPPP16COFFEEPOT\_DEVICE;

R2 = R0;

R3 = TEMPERATURE;

CC = R3 <= R2;

If CC jump second;

jump Out;

second:

R1 = COFFEEPOD;

R2 = R4 & R1;

CC = R2 == 0;

If CC jump coffeePod;

coffeePod:

R1 = COFFEEPOD;

R0 = R4 | R1;

[P4] = R0;

Out:

UNLINK;

P4 = [SP++];

R4 = [SP++];

\_checkForCoffeePodASM.end:

RTS;

**MainCoffeePotAssignment2.cpp**

/\*

\* MainCoffeePotAssignment2.cpp

\*

\* Created on: Nov 25, 2019

\* Author: aidan

\*/

#include "Assignment2CoffeePot\_BF609\_Core0.h"

#define MAX\_WATER\_LEVEL 600

#define MAX\_TEMPERATURE 100

#define NUMBEROFCOFFEEPOTS 2

#define SHOW\_FUNCTION\_STUB\_INFORMATION 1

#define MASK\_BITS\_15TO2\_AND\_BIT0 0xfffd

#define MASK\_LED 0xf000

#define MASK\_BITS\_15TO3\_AND\_BITS1TO0 0xfffb

#define MASK\_BITS\_15TO4\_AND\_BITS2TO0 0xfff7

#define WATERINFLOW 30

#define FULLHEATBOOST 15

#define HEATVALUE 190

#define TRUE 1

#define ZERO 0

#define TEMPERATURETOINSERTCOFFEEPOD 95

#define PERIOD 480000000

#define COUNT 480000000

#define DEBUG 0

#define TCNTLENABLETIMERBIT 0x00000002

#define TCNTLAUTORELOADBIT 0x00000004

#define TCNTLPOWERBIT 0x00000001

#define CONTROLREGISTERCONTRILBITSMASK 0x000F

//ENUM for Coffeepot\_ID to make it an array

COFFEEPOT\_ID COFFEEPOTIDS[] = {COFFEEPOT1, COFFEEPOT2, COFFEEPOT3, COFFEEPOT4};

extern volatile bool doFastForward = false;

bool My\_Init\_LEDInterface\_Done = false;

bool My\_Init\_SwitchInterface\_Done = false;

bool My\_Init\_GPIO\_REB\_Input\_Done = false;

bool My\_Init\_GPIO\_REB\_Output\_Done = false;

bool My\_Init\_GPIO\_REB\_Done = false;

static bool FPSW5PressedAndReleased = false;

void startCoffeePot(void)

{

showNameOfProcessorUsed();

Init\_CoffeePotSimulation(NUMBEROFCOFFEEPOTS, USE\_TEXT\_AND\_GRAPHICS\_GUIS); //Initializing the simulation

char uniqueCoffeePot1Name[] = "Aidan";

char uniqueCoffeePot2Name[] = "Aidanj";

COFFEEPOT\_DEVICE\* coffeePotsBaseAddress[NUMBEROFCOFFEEPOTS];

coffeePotsBaseAddress[0] = Add\_CoffeePotToSystem\_PlugAndPlay(COFFEEPOTIDS[0], uniqueCoffeePot1Name);

coffeePotsBaseAddress[1] = Add\_CoffeePotToSystem\_PlugAndPlay(COFFEEPOTIDS[1], uniqueCoffeePot2Name); //Making the second coffeepot have a different capacity

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

myInit\_CoreTimer(PERIOD, COUNT); //This is initializing the Core Timer

register\_handler(ik\_timer, interruptServiceRoutineFastForward);

myControl\_CoreTimer((TCNTLENABLETIMERBIT | TCNTLAUTORELOADBIT |TCNTLPOWERBIT)); //This is enabling the Core Timer

startCoreTimerInterrupts();

initializingCoffeePot(coffeePotsBaseAddress);

activateLEDControl(coffeePotsBaseAddress);

activateWaterControl(coffeePotsBaseAddress);

activateHeatControl(coffeePotsBaseAddress);

bool isCoreTimerComplete;

isCoreTimerComplete = myCompleted\_CoreTimer();

#if DEBUG

printf("%d \n", isCoreTimerComplete);

#endif

while(TRUE)

{

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

//LEDControlDemo(coffeePotsBaseAddress);

//demonstrateLEDASM(coffeePotsBaseAddress[j]); //Using the ASM function instead

fillCoffeePotToWaterLEvel(coffeePotsBaseAddress[j]);

heatWaterToTemperature(coffeePotsBaseAddress[j]);

checkForCoffeePodASM(coffeePotsBaseAddress[j]); //Using the ASM function instead

//checkForCoffeePod(coffeePotsBaseAddress[j]);

}

WaitTillSwitch5PressedAndReleased();

if(FPSW5PressedAndReleased == true)

{

coffeePot1OverWritingControlRegisterBits(coffeePotsBaseAddress[0]);

coffeePot2OverWritingControlRegisterBits(coffeePotsBaseAddress[1]);

}

showingCoffeePot1LEDSOnFrontPanelAndSimulationLEDS(coffeePotsBaseAddress[0]);

showingCoffeePot2LEDSOnREBAndSimulationLEDS(coffeePotsBaseAddress[1]);

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePotsBaseAddress);

doFastForward = false;

}

}

}

inline void Show\_Function\_Stub\_Information(char \*functionNameInformation)

{

if(SHOW\_FUNCTION\_STUB\_INFORMATION)

printf("%s \n", functionNameInformation);

}

void showNameOfProcessorUsed(void) //This function is displaying what processor is being used

{

#if defined(\_\_ADSPBF533\_\_)

char processor[] = "\_\_ADSPBF533\_\_";

#else

char processor[] = "\_\_ADSOBF609\_\_";

#endif

printf("Aidan's Coffee pots are running on %s system \n", processor);

}

void initializingCoffeePot(COFFEEPOT\_DEVICE \*coffeePot[]) //This function will be activating the 4 different coffeepots

{

unsigned short int currentControlRegister; //declaring a currentControlRegister variable

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = ReadControlRegister\_CPP(coffeePot[j]);

unsigned short int newControlRegister = INITandSTAYPOWEREDON\_BIT;

coffeePot[j] -> controlRegister = newControlRegister;

}

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

while((currentControlRegister & DEVICE\_READY\_BIT\_RO)!= DEVICE\_READY\_BIT\_RO)

{

if(doFastForward)

{

FastForward\_OneSimulationTIC(coffeePot[j]);

doFastForward = false;

}

currentControlRegister = ReadControlRegister\_CPP(coffeePot[j]);

}

}

}

void activateLEDControl(COFFEEPOT\_DEVICE \*coffeePot[])

{

unsigned short int currentControlRegister;

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = coffeePot[j] -> controlRegister;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO2\_AND\_BIT0;

currentControlRegister = currentControlRegister | LED\_DISPLAY\_ENABLE\_BIT;

coffeePot[j] -> controlRegister = currentControlRegister;

}

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePot);

doFastForward = false;

}

}

void activateWaterControl(COFFEEPOT\_DEVICE \*coffeePot[])

{

unsigned short int currentControlRegister;

for (int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = coffeePot[j] -> controlRegister;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO3\_AND\_BITS1TO0;

currentControlRegister = currentControlRegister | WATER\_ENABLE\_BIT;

coffeePot[j] -> controlRegister = currentControlRegister;

}

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePot);

doFastForward = false;

}

}

void activateHeatControl(COFFEEPOT\_DEVICE \*coffeePot[])

{

unsigned short int currentControlRegister;

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = coffeePot[j] -> controlRegister;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO4\_AND\_BITS2TO0;

currentControlRegister = currentControlRegister | HEATER\_ENABLE\_BIT;

coffeePot[j] -> controlRegister = currentControlRegister;

}

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePot);

doFastForward = false;

}

}

void LEDControlDemo(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentLEDState;

currentLEDState = (currentControlRegister & MASK\_LED) >> 12;

switch(currentLEDState)

{

case 0:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED1\_BIT;

break;

case 1:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED2\_BIT;

break;

case 2:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED3\_BIT;

break;

case 4:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED4\_BIT;

break;

case 8:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED1\_BIT;

break;

}

coffeePot -> controlRegister = currentControlRegister;

}

void fillCoffeePotToWaterLEvel(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentWaterLevel = CurrentWaterLevel\_CPP(coffeePot);

if (currentWaterLevel < MAX\_WATER\_LEVEL - (MAX\_WATER\_LEVEL/3))

{

coffeePot -> waterInFlowRegister = WATERINFLOW;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO3\_AND\_BITS1TO0;

currentControlRegister = currentControlRegister | WATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

else

{

coffeePot -> waterInFlowRegister = ZERO;

currentControlRegister = currentControlRegister & ~WATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

}

void heatWaterToTemperature(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentTemperature = CurrentTemperature\_CPP(coffeePot);

if(currentTemperature < MAX\_TEMPERATURE)

{

coffeePot -> heaterRegister = HEATVALUE;

coffeePot -> heaterBoostRegister = FULLHEATBOOST;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO4\_AND\_BITS2TO0;

currentControlRegister = currentControlRegister | HEATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

else

{

coffeePot -> heaterRegister = ZERO;

coffeePot -> heaterBoostRegister = ZERO;

currentControlRegister = currentControlRegister & ~HEATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

}

void checkForCoffeePod(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentTemperature = CurrentTemperature\_CPP(coffeePot);

if (currentTemperature >= TEMPERATURETOINSERTCOFFEEPOD)

{

if((currentControlRegister & COFFEEPOT\_INSERTED) == ZERO)

{

currentControlRegister = (currentControlRegister | COFFEEPOT\_INSERTED);

coffeePot -> controlRegister = currentControlRegister;

}

}

}

#define CORETIMEPOWERBIT 0x00000001

#define CORETIMERAUTORLD 0x00000008

void myInit\_CoreTimer(unsigned long int period, unsigned long int count)

{

\*pTCNTL = CORETIMEPOWERBIT; //Might have to include the autorelod bit or with the power bit

\*pTPERIOD = period; //Setting period after count to make sure count does not get overwritten

\*pTCOUNT = count; //Setting the count registers which counts down to 0 every cycle and causes the control register bit 3 to turn on when 0

\*pTSCALE = 0x00000000; //Setting the scale register to 0

#if DEBUG

printf("Core Timer Control Register %lu \n", \*pTCNTL);

printf("Core Timer Period Register %lu \n", \*pTPERIOD);

printf("Core Timer Count Register %lu \n", \*pTCOUNT);

printf("Core Timer Scale Register %lu \n", \*pTSCALE);

#endif

}

#define TCNTLBIT3TO0 0x0000000f

void myControl\_CoreTimer(unsigned short int cntrl\_value)

{

cntrl\_value = cntrl\_value & TCNTLBIT3TO0;

\*pTCNTL = cntrl\_value;

}

#define TCNTLBITS3AND1TO0 0x0000000b

#define TCNTLDONEBIT 0x00000008

bool myCompleted\_CoreTimer(void)

{

bool returnValue = false;

unsigned long int coreTimerControlValue = \*pTCNTL;

if((coreTimerControlValue & TCNTLBITS3AND1TO0) == TCNTLBITS3AND1TO0)

{

\*pTCNTL = \*pTCNTL & (~TCNTLDONEBIT);

returnValue = true;

}

return returnValue;

}

void myTimedWaitOnCoreTimer(void)

{

bool coreTimerInitialized = false;

int count = 0;

while(!coreTimerInitialized)

{

unsigned long int stickyBitTCNTL = (\*pTCNTL & TCNTLDONEBIT);

if(TCNTLDONEBIT == stickyBitTCNTL)

{

coreTimerInitialized = true;

}

count++;

#if DEBUG

printf("Time through loop %d \n", count);

#endif

}

return;

}

#define TCNTLTINTBIT 0x00000008

#define TCNTLTMRENBIT 0x00000002

void startCoreTimer(void)

{

unsigned long int cntrl\_value = TCNTLTMRENBIT;

\*pTCNTL = (\*pTCNTL | cntrl\_value);

}

void stopCoreTimer(void)

{

unsigned long int cntrl\_value = (~TCNTLTMRENBIT);

\*pTCNTL = (\*pTCNTL & (cntrl\_value));

}

void resetCoreTimerInterrupt(void)

{

\*pTCNTL = \*pTCNTL & (~TCNTLTINTBIT);

}

#define CORETIMERIMASKBITINTERRUPT 0x00000040

#pragma interrupt

void interruptServiceRoutineFastForward(void)

{

#if DEBUG

printf("In interrupt Function \n");

#endif

unsigned long int coreTimerControlValue = \*pTCNTL;

\*pTCNTL = coreTimerControlValue & (~TCNTLDONEBIT); //Reseting the Core Timer

//unsigned long int coreInteruptMaskRegister = \*pIMASK;

//\*pIMASK = coreInteruptMaskRegister & (~CORETIMERIMASKBITINTERRUPT); //Clearing the interrupt coretimer bit

doFastForward = true;

ssync(); //Making sure all the instructions have been flushed

}

void startCoreTimerInterrupts(void)

{

unsigned long int coreInteruptMaskRegister = \*pIMASK;

//This is sotring the address of the function that the interrupt will go to

//in the vector event table

//\*pEVT6 = (void\*) interruptServiceRoutineFastForward;

\*pIMASK = coreInteruptMaskRegister | CORETIMERIMASKBITINTERRUPT; //This is enabling the core timer interupt bit in the IMASK register

//OR

//register\_handler(ik\_timer,interruptServiceRoutineFastForward);

}

void stopCoreTimerInterrupts(void)

{

unsigned long int coreInteruptMaskRegister = \*pIMASK;

\*pIMASK = coreInteruptMaskRegister & (~CORETIMERIMASKBITINTERRUPT); //Disabling the coretimer IMASK interrupt Bit

}

void showingCoffeePot1LEDSOnFrontPanelAndSimulationLEDS(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int coffeePot1LEDS;

coffeePot1LEDS = (currentControlRegister & CONTROLREGISTERCONTRILBITSMASK);

Write\_GPIO\_FrontPanelLEDS(coffeePot1LEDS);

coffeePot1LEDS = coffeePot1LEDS << 12;

currentControlRegister = ((~MASK\_LED) & currentControlRegister);

coffeePot -> controlRegister = (coffeePot1LEDS | currentControlRegister);

}

void showingCoffeePot2LEDSOnREBAndSimulationLEDS(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int coffeePot2LEDS;

coffeePot2LEDS = (currentControlRegister & CONTROLREGISTERCONTRILBITSMASK);

My\_Write\_REB\_LED(coffeePot2LEDS);

coffeePot2LEDS = coffeePot2LEDS << 12;

currentControlRegister = ((~MASK\_LED) & currentControlRegister);

coffeePot -> controlRegister = (coffeePot2LEDS | currentControlRegister);

}

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

}

#define CONTROLREGISTERCONTROLBITSMASK 0x000f

void coffeePot1OverWritingControlRegisterBits(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int switchValue = My\_ReadSwitches();

currentControlRegister = (currentControlRegister & (~CONTROLREGISTERCONTROLBITSMASK));

coffeePot -> controlRegister = (currentControlRegister | switchValue);

}

void coffeePot2OverWritingControlRegisterBits(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int switchValue = My\_Read\_REB\_Switches();

currentControlRegister = (currentControlRegister & (~CONTROLREGISTERCONTROLBITSMASK));

coffeePot -> controlRegister = (currentControlRegister | switchValue);

}

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 -1;

}

REB\_BITS16 wantedSwitchOnBoardValueActiveHigh = My\_Read\_GPIO\_REB\_InputASM(); //The board is active high

return wantedSwitchOnBoardValueActiveHigh;

}

unsigned char My\_ReadSwitches(void) //This function is reading the switches from the panel

{

if (My\_Init\_SwitchInterface\_Done == false)

{

printf("Switch hardware not ready \n");

return -1;

}

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;

}

#define FRONTPANELSW5 0x10

void WaitTillSwitch5PressedAndReleased() //This function is making sure that Switch 3 is pressed and released

{

unsigned char switchValue;

switchValue = My\_ReadSwitches();

if((switchValue == FRONTPANELSW5) && (!FPSW5PressedAndReleased))

{

while(1)

{

switchValue = My\_ReadSwitches();

unsigned char bitValue = switchValue & FRONTPANELSW5;

if(bitValue == 0x0)

{

FPSW5PressedAndReleased = true;

return;

}

}

}

else

{

return;

}

}

**Lab0And1InitFunctions.cpp**

/\*

\* Lab0And1InitFunctions.cpp

\*

\* Created on: Nov 21, 2019

\* Author: aidan

\*/

#include "Assignment2CoffeePot\_BF609\_Core0.h"

void My\_Init\_SwitchInterface(void) //This function is initializing the Switches on the Front Panel

{

My\_Init\_SwitchInterface\_Done = true;

#ifdef \_\_ADSPBF609\_\_

Init\_GPIO\_FrontPanelSwitches();

#endif

}

void My\_Init\_LEDInterface(void) //This function is initializing the LEDs on the Front Panel

{

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 REB

{

My\_Init\_GPIO\_REB\_Input\_Done = true;

#ifdef \_\_ADSPBF609\_\_

Init\_GPIO\_REB\_Input();

#endif

}

void My\_Init\_GPIO\_REB\_Output(void) //This function is initializing the LEDs on the REB

{

My\_Init\_GPIO\_REB\_Output\_Done = true;

#ifdef \_\_ADSPBF609\_\_

Init\_GPIO\_REB\_Output();

#endif

}

**Lab1And0FrontPanelFunctions.cpp**

/\*

\* Lab1And0FrontPanelFunctions.cpp

\*

\* Created on: Nov 21, 2019

\* Author: aidan

\*/

#include "Lab1And0FrontPanelFunctions.h"

void myWriteFrontPanelLEDs(unsigned char neededLEDValue) //This function is writing the values to be displayed by the LEDs

{

if (My\_Init\_LEDInterface\_Done == false)

{

return;

}

Write\_GPIO\_FrontPanelLEDS(neededLEDValue); //Writing the value to the panel of LEDs

}

unsigned char myReadFrontPanelSwitches(void) //This function is reading the switches from the panel

{

if (My\_Init\_SwitchInterface\_Done == false)

{

return GARBAGEVALUE;

}

FRONTPANEL\_LED\_8BIT\_VALUE activeLowValues = Read\_GPIO\_FrontPanelSwitches();

FRONTPANEL\_LED\_8BIT\_VALUE activeHighValues = ~activeLowValues;

FRONTPANEL\_LED\_8BIT\_VALUE wantedSwitchValueActiveHigh = activeHighValues & MASK\_KEEP\_LOWER\_FIVE\_BITS;

return wantedSwitchValueActiveHigh;

}

unsigned char myReadFrontPanelLEDs(void)

{

if (My\_Init\_LEDInterface\_Done == false)

{

return GARBAGEVALUE;

}

return Read\_GPIO\_FrontPanelLEDS();

}

**Lab1And0FrontPanelFunctions.h**

/\*

\* Lab1And0FrontPanelFunctions.h

\*

\* Created on: Nov 21, 2019

\* Author: aidan

\*/

#ifndef LAB1AND0FRONTPANELFUNCTIONS\_H\_

#define LAB1AND0FRONTPANELFUNCTIONS\_H\_

#include "Assignment2CoffeePot\_BF609\_Core0.h"

#define GARBAGEVALUE static\_cast<unsigned char>(-1)

#define MASK\_KEEP\_LOWER\_FIVE\_BITS 0x1F

//Function Prototypes used in Lab 1 for FrontPanel

void myWriteFrontPanelLEDs(unsigned char);

unsigned char myReadFrontPanelSwitches(void);

unsigned char myReadFrontPanelLEDs(void);

#endif /\* LAB1AND0FRONTPANELFUNCTIONS\_H\_ \*/

**My\_Init\_GPIO\_REB\_INPUTASM.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**

/\*

\* 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\_GPIO\_REB\_INPUTASM.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**

/\*

\* 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;

**533 Code:**

**Assignment1WithCoreTimer\_Sim533.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Assignment1WithCoreTimer\_Sim533.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <sys/platform.h>

#include "adi\_initialize.h"

#include "Assignment1WithCoreTimer\_Sim533.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();

#ifdef \_\_ADSPBF533\_\_

printf("Starting the Coffee Pot Assignment\n");

startCoffeePot();

#endif

#ifdef \_\_ADSPBF609\_\_

printf("Starting the Coffee Pot Assignment\n");

startCoffeePot();

#endif

return 0;

}

**Assignment1WithCoreTimer\_Sim533.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Assignment1WithCoreTimer\_Sim533.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_ASSIGNMENT1WITHCORETIMER\_SIM533\_H\_\_

#define \_\_ASSIGNMENT1WITHCORETIMER\_SIM533\_H\_\_

/\* Add your custom header content here \*/

#include "../../ENCM511\_SpecificFiles/ENCM511\_include/CoffeePot\_SimulatorFunctions2017.h"

#include <stdio.h>

#include <blackfin.h>

#include <sys/exception.h>

//Prototypes for Coffee Pot Assignment 1 Functions

void startCoffeePot(void);

void showNameOfProcessorUsed(void);

void DemonstrateLEDControl\_CPP(void);

void initializingCoffeePot(COFFEEPOT\_DEVICE\* coffeePot[]);

void activateLEDControl(COFFEEPOT\_DEVICE \*coffeePot[]);

void activateWaterControl(COFFEEPOT\_DEVICE \*coffeePot[]);

void activateHeatControl(COFFEEPOT\_DEVICE \*coffeePot[]);

void LEDControlDemo(COFFEEPOT\_DEVICE \*coffeePot);

void fillCoffeePotToWaterLEvel(COFFEEPOT\_DEVICE \*coffeePot);

void heatWaterToTemperature(COFFEEPOT\_DEVICE \*coffeePot);

void checkForCoffeePod(COFFEEPOT\_DEVICE \*coffeePot);

//Prototypes for CoreTimer Functions for BF533

void myInit\_CoreTimer(unsigned long int period, unsigned long int count);

void myControl\_CoreTimer(unsigned short int cntrl\_value);

bool myCompleted\_CoreTimer(void);

void myTimedWaitOnCoreTimer(void);

void startCoreTimer(void);

void stopCoreTimer(void);

void resetCoreTimerInterrupt(void);

//Prototypes for Interrupt Functions

void interruptServiceRoutineFastForward(void);

void startCoreTimerInterrupts(void);

void stopCoreTimerInterrupts(void);

//Prototypes for Coffee Pot Assignment 1 Assembly Functions

extern "C" void demonstrateLEDASM(COFFEEPOT\_DEVICE \*coffeePot);

extern "C" void checkForCoffeePodASM(COFFEEPOT\_DEVICE \*coffeePot);

#endif /\* \_\_ASSIGNMENT1WITHCORETIMER\_SIM533\_H\_\_ \*/

**MainCoffeePot.cpp**

/\*

\* MainCoffeePot.cpp

\*

\* Created on: Oct 23, 2019

\* Author: aidan

\*/

#include "Assignment1WithCoreTimer\_Sim533.h"

#define MAX\_WATER\_LEVEL 600

#define MAX\_TEMPERATURE 100

#define NUMBEROFCOFFEEPOTS 2

#define SHOW\_FUNCTION\_STUB\_INFORMATION 1

#define MASK\_BITS\_15TO2\_AND\_BIT0 0xfffd

#define MASK\_LED 0xf000

#define MASK\_BITS\_15TO3\_AND\_BITS1TO0 0xfffb

#define MASK\_BITS\_15TO4\_AND\_BITS2TO0 0xfff7

#define WATERINFLOW 30

#define FULLHEATBOOST 15

#define HEATVALUE 190

#define TRUE 1

#define ZERO 0

#define TEMPERATURETOINSERTCOFFEEPOD 95

#define PERIOD 100000

#define COUNT 100000

#define DEBUG 0

#define TCNTLENABLETIMERBIT 0x00000002

#define TCNTLAUTORELOADBIT 0x00000004

#define TCNTLPOWERBIT 0x00000001

//ENUM for Coffeepot\_ID to make it an array

COFFEEPOT\_ID COFFEEPOTIDS[] = {COFFEEPOT1, COFFEEPOT2, COFFEEPOT3, COFFEEPOT4};

extern volatile bool doFastForward = false;

void startCoffeePot(void)

{

showNameOfProcessorUsed();

Init\_CoffeePotSimulation(NUMBEROFCOFFEEPOTS, USE\_TEXT\_AND\_GRAPHICS\_GUIS); //Initializing the simulation

char uniqueCoffeePot1Name[] = "Aidan";

char uniqueCoffeePot2Name[] = "Aidanj";

COFFEEPOT\_DEVICE\* coffeePotsBaseAddress[NUMBEROFCOFFEEPOTS];

coffeePotsBaseAddress[0] = Add\_CoffeePotToSystem\_PlugAndPlay(COFFEEPOTIDS[0], uniqueCoffeePot1Name);

coffeePotsBaseAddress[1] = Add\_CoffeePotToSystem\_PlugAndPlay(COFFEEPOTIDS[1], uniqueCoffeePot2Name); //Making the second coffeepot have a different capacity

myInit\_CoreTimer(PERIOD, COUNT); //This is initializing the Core Timer

register\_handler(ik\_timer, interruptServiceRoutineFastForward);

myControl\_CoreTimer((TCNTLENABLETIMERBIT | TCNTLAUTORELOADBIT |TCNTLPOWERBIT)); //This is enabling the Core Timer

startCoreTimerInterrupts();

initializingCoffeePot(coffeePotsBaseAddress);

activateLEDControl(coffeePotsBaseAddress);

activateWaterControl(coffeePotsBaseAddress);

activateHeatControl(coffeePotsBaseAddress);

bool isCoreTimerComplete;

isCoreTimerComplete = myCompleted\_CoreTimer();

#if DEBUG

printf("%d \n", isCoreTimerComplete);

#endif

while(TRUE)

{

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

//LEDControlDemo(coffeePotsBaseAddress);

demonstrateLEDASM(coffeePotsBaseAddress[j]); //Using the ASM function instead

fillCoffeePotToWaterLEvel(coffeePotsBaseAddress[j]);

heatWaterToTemperature(coffeePotsBaseAddress[j]);

checkForCoffeePodASM(coffeePotsBaseAddress[j]); //Using the ASM function instead

//checkForCoffeePod(coffeePotsBaseAddress[j]);

}

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePotsBaseAddress);

doFastForward = false;

}

}

}

inline void Show\_Function\_Stub\_Information(char \*functionNameInformation)

{

if(SHOW\_FUNCTION\_STUB\_INFORMATION)

printf("%s \n", functionNameInformation);

}

void showNameOfProcessorUsed(void) //This function is displaying what processor is being used

{

#if defined(\_\_ADSPBF533\_\_)

char processor[] = "\_\_ADSPBF533\_\_";

#else

char processor[] = "\_\_ADSOBF609\_\_";

#endif

printf("Aidan's Coffee pots are running on %s system \n", processor);

}

void initializingCoffeePot(COFFEEPOT\_DEVICE \*coffeePot[]) //This function will be activating the 4 different coffeepots

{

unsigned short int currentControlRegister; //declaring a currentControlRegister variable

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = ReadControlRegister\_CPP(coffeePot[j]);

unsigned short int newControlRegister = INITandSTAYPOWEREDON\_BIT;

coffeePot[j] -> controlRegister = newControlRegister;

}

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

while((currentControlRegister & DEVICE\_READY\_BIT\_RO)!= DEVICE\_READY\_BIT\_RO)

{

if(doFastForward)

{

FastForward\_OneSimulationTIC(coffeePot[j]);

doFastForward = false;

}

currentControlRegister = ReadControlRegister\_CPP(coffeePot[j]);

}

}

}

void activateLEDControl(COFFEEPOT\_DEVICE \*coffeePot[])

{

unsigned short int currentControlRegister;

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = coffeePot[j] -> controlRegister;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO2\_AND\_BIT0;

currentControlRegister = currentControlRegister | LED\_DISPLAY\_ENABLE\_BIT;

coffeePot[j] -> controlRegister = currentControlRegister;

}

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePot);

doFastForward = false;

}

}

void activateWaterControl(COFFEEPOT\_DEVICE \*coffeePot[])

{

unsigned short int currentControlRegister;

for (int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = coffeePot[j] -> controlRegister;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO3\_AND\_BITS1TO0;

currentControlRegister = currentControlRegister | WATER\_ENABLE\_BIT;

coffeePot[j] -> controlRegister = currentControlRegister;

}

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePot);

doFastForward = false;

}

}

void activateHeatControl(COFFEEPOT\_DEVICE \*coffeePot[])

{

unsigned short int currentControlRegister;

for(int j = ZERO; j < NUMBEROFCOFFEEPOTS; j++)

{

currentControlRegister = coffeePot[j] -> controlRegister;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO4\_AND\_BITS2TO0;

currentControlRegister = currentControlRegister | HEATER\_ENABLE\_BIT;

coffeePot[j] -> controlRegister = currentControlRegister;

}

if(doFastForward)

{

FastForward\_OneSimulationTIC(\*coffeePot);

doFastForward = false;

}

}

void LEDControlDemo(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentLEDState;

currentLEDState = (currentControlRegister & MASK\_LED) >> 12;

switch(currentLEDState)

{

case 0:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED1\_BIT;

break;

case 1:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED2\_BIT;

break;

case 2:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED3\_BIT;

break;

case 4:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED4\_BIT;

break;

case 8:

currentControlRegister = (currentControlRegister & ~MASK\_LED) | LED1\_BIT;

break;

}

coffeePot -> controlRegister = currentControlRegister;

}

void fillCoffeePotToWaterLEvel(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentWaterLevel = CurrentWaterLevel\_CPP(coffeePot);

if (currentWaterLevel < MAX\_WATER\_LEVEL - (MAX\_WATER\_LEVEL/8))

{

coffeePot -> waterInFlowRegister = WATERINFLOW;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO3\_AND\_BITS1TO0;

currentControlRegister = currentControlRegister | WATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

else

{

coffeePot -> waterInFlowRegister = ZERO;

currentControlRegister = currentControlRegister & ~WATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

}

void heatWaterToTemperature(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentTemperature = CurrentTemperature\_CPP(coffeePot);

if(currentTemperature < MAX\_TEMPERATURE)

{

coffeePot -> heaterRegister = HEATVALUE;

coffeePot -> heaterBoostRegister = FULLHEATBOOST;

currentControlRegister = currentControlRegister & MASK\_BITS\_15TO4\_AND\_BITS2TO0;

currentControlRegister = currentControlRegister | HEATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

else

{

coffeePot -> heaterRegister = ZERO;

coffeePot -> heaterBoostRegister = ZERO;

currentControlRegister = currentControlRegister & ~HEATER\_ENABLE\_BIT;

coffeePot -> controlRegister = currentControlRegister;

}

}

void checkForCoffeePod(COFFEEPOT\_DEVICE \*coffeePot)

{

unsigned short int currentControlRegister = coffeePot -> controlRegister;

unsigned short int currentTemperature = CurrentTemperature\_CPP(coffeePot);

if (currentTemperature >= TEMPERATURETOINSERTCOFFEEPOD)

{

if((currentControlRegister & COFFEEPOT\_INSERTED) == ZERO)

{

currentControlRegister = (currentControlRegister | COFFEEPOT\_INSERTED);

coffeePot -> controlRegister = currentControlRegister;

}

}

}

#define CORETIMEPOWERBIT 0x00000001

#define CORETIMERAUTORLD 0x00000008

void myInit\_CoreTimer(unsigned long int period, unsigned long int count)

{

\*pTCNTL = CORETIMEPOWERBIT; //Might have to include the autorelod bit or with the power bit

\*pTPERIOD = period; //Setting period after count to make sure count does not get overwritten

\*pTCOUNT = count; //Setting the count registers which counts down to 0 every cycle and causes the control register bit 3 to turn on when 0

\*pTSCALE = 0x00000000; //Setting the scale register to 0

#if DEBUG

printf("Core Timer Control Register %lu \n", \*pTCNTL);

printf("Core Timer Period Register %lu \n", \*pTPERIOD);

printf("Core Timer Count Register %lu \n", \*pTCOUNT);

printf("Core Timer Scale Register %lu \n", \*pTSCALE);

#endif

}

#define TCNTLBIT3TO0 0x0000000f

void myControl\_CoreTimer(unsigned short int cntrl\_value)

{

cntrl\_value = cntrl\_value & TCNTLBIT3TO0;

\*pTCNTL = cntrl\_value;

}

#define TCNTLBITS3AND1TO0 0x0000000b

#define TCNTLDONEBIT 0x00000008

bool myCompleted\_CoreTimer(void)

{

bool returnValue = false;

unsigned long int coreTimerControlValue = \*pTCNTL;

if((coreTimerControlValue & TCNTLBITS3AND1TO0) == TCNTLBITS3AND1TO0)

{

\*pTCNTL = \*pTCNTL & (~TCNTLDONEBIT);

returnValue = true;

}

return returnValue;

}

void myTimedWaitOnCoreTimer(void)

{

bool coreTimerInitialized = false;

int count = 0;

while(!coreTimerInitialized)

{

unsigned long int stickyBitTCNTL = (\*pTCNTL & TCNTLDONEBIT);

if(TCNTLDONEBIT == stickyBitTCNTL)

{

coreTimerInitialized = true;

}

count++;

#if DEBUG

printf("Time through loop %d \n", count);

#endif

}

return;

}

#define TCNTLTINTBIT 0x00000008

#define TCNTLTMRENBIT 0x00000002

void startCoreTimer(void)

{

unsigned long int cntrl\_value = TCNTLTMRENBIT;

\*pTCNTL = (\*pTCNTL | cntrl\_value);

}

void stopCoreTimer(void)

{

unsigned long int cntrl\_value = (~TCNTLTMRENBIT);

\*pTCNTL = (\*pTCNTL & (cntrl\_value));

}

void resetCoreTimerInterrupt(void)

{

\*pTCNTL = \*pTCNTL & (~TCNTLTINTBIT);

}

#define CORETIMERIMASKBITINTERRUPT 0x00000040

#pragma interrupt

void interruptServiceRoutineFastForward(void)

{

#if DEBUG

printf("In interrupt Function \n");

#endif

unsigned long int coreTimerControlValue = \*pTCNTL;

\*pTCNTL = coreTimerControlValue & (~TCNTLDONEBIT); //Reseting the Core Timer

//unsigned long int coreInteruptMaskRegister = \*pIMASK;

//\*pIMASK = coreInteruptMaskRegister & (~CORETIMERIMASKBITINTERRUPT); //Clearing the interrupt coretimer bit

doFastForward = true;

ssync(); //Making sure all the instructions have been flushed

}

void startCoreTimerInterrupts(void)

{

unsigned long int coreInteruptMaskRegister = \*pIMASK;

//This is sotring the address of the function that the interrupt will go to

//in the vector event table

//\*pEVT6 = (void\*) interruptServiceRoutineFastForward;

\*pIMASK = coreInteruptMaskRegister | CORETIMERIMASKBITINTERRUPT; //This is enabling the core timer interupt bit in the IMASK register

//OR

//register\_handler(ik\_timer,interruptServiceRoutineFastForward);

}

void stopCoreTimerInterrupts(void)

{

unsigned long int coreInteruptMaskRegister = \*pIMASK;

\*pIMASK = coreInteruptMaskRegister & (~CORETIMERIMASKBITINTERRUPT); //Disabling the coretimer IMASK interrupt Bit

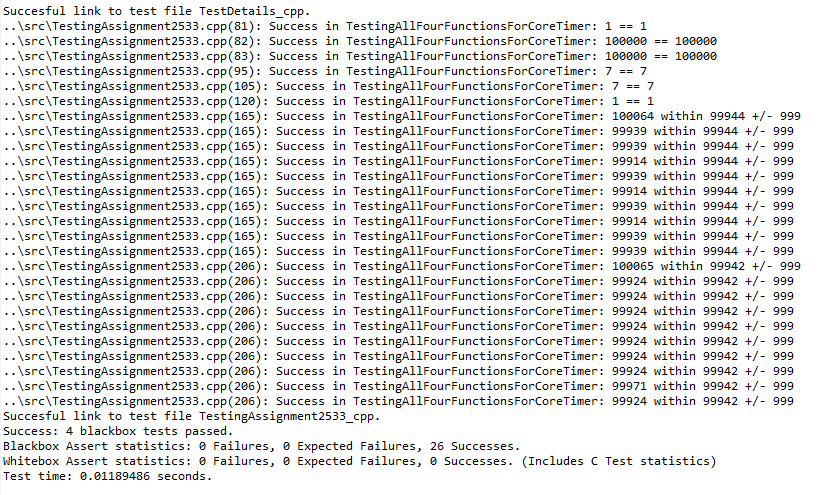
}

**EUNIT Screen Shots:**

**533 Testing with just Function Stubs:**



**533 Testing the Four Core Timer Functions:**



**609 Testing the Four Core Timer Functions:**

