**------------------------------------------------------------------------------------**

**Task 09: Date Submitted: 11/30/19**

Youtube Link: <https://www.youtube.com/watch?v=WMLS4m43H6Q>

**Modified Code:**

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------------

// Project: Blink TM4C BIOS Using Mailbox/Queue (SOLUTION)

// Author: Eric Wilbur

// Date: June 2014

//

// Note: The function call TimerIntClear(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT) HAS

// to be in the ISR. This fxn clears the TIMER's interrupt flag coming

// from the peripheral - it does NOT clear the CPU interrupt flag - that

// is done by hardware. The author struggled figuring this part out - hence

// the note. And, in the Swi lab, this fxn must be placed in the

// Timer\_ISR fxn because it will be the new ISR.

//

// Follow these steps to create this project in CCSv6.0:

// 1. Project -> New CCS Project

// 2. Select Template:

// - TI-RTOS for Tiva-C -> Driver Examples -> EK-TM4C123 LP -> Example Projects ->

// Empty Project

// - Empty Project contains full instrumentation (UIA, RTOS Analyzer) and

// paths set up for the TI-RTOS version of MSP430Ware

// 3. Delete the following files:

// - Board.h, empty.c, EK\_TM4C123GXL.c/h, empty\_readme.txt

// 4. Add main.c from TI-RTOS Workshop Solution file for this lab

// 5. Edit empty.cfg as needed (to add/subtract) BIOS services, delete given Task

// 6. Build, load, run...

//

// FYI - Part B solution for Queues is actually shown working. Part A solution

// (Mailbox) is populated below but commented out.

//----------------------------------------------------------------------------------

//----------------------------------------

// BIOS header files

//----------------------------------------

**#include** <xdc/std.h> //mandatory - have to include first, for BIOS types

**#include** <ti/sysbios/BIOS.h> //mandatory - if you call APIs like BIOS\_start()

**#include** <xdc/runtime/Log.h> //needed for any Log\_info() call

**#include** <xdc/cfg/global.h> //header file for statically defined objects/handles

//------------------------------------------

// TivaWare Header Files

//------------------------------------------

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_types.h"

**#include** "inc/hw\_memmap.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/gpio.h"

**#include** "inc/hw\_ints.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/timer.h"

//----------------------------------------

// Prototypes

//----------------------------------------

**void** **hardware\_init**(**void**);

**void** **ledToggle**(**void**);

**void** **Timer\_ISR**(**void**);

//---------------------------------------

// Globals

//---------------------------------------

**volatile** int16\_t i16ToggleCount = 0;

//------------------------

// for Mailbox - Part A

//------------------------

//typedef struct MsgObj {

// Int val; // message value

//} MsgObj, \*Msg;

//------------------------

// for Queue - Part B

//------------------------

**typedef** **struct** MsgObj {

Queue\_Elem elem;

Int val; // message value

} MsgObj, \*Msg; // Use Msg as pointer to MsgObj

//---------------------------------------------------------------------------

// main()

//---------------------------------------------------------------------------

**void** **main**(**void**)

{

hardware\_init(); // init hardware via Xware

BIOS\_start(); // start BIOS Scheduler

}

//---------------------------------------------------------------------------

// hardware\_init()

//

// inits GPIO pins for toggling the LED

//---------------------------------------------------------------------------

**void** **hardware\_init**(**void**)

{

uint32\_t ui32Period;

//Set CPU Clock to 40MHz. 400MHz PLL/2 = 200 DIV 5 = 40MHz

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_XTAL\_16MHZ|SYSCTL\_OSC\_MAIN);

// ADD Tiva-C GPIO setup - enables port, sets pins 1-3 (RGB) pins for output

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

// Turn on the LED

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 4);

// Timer 2 setup code

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER2); // enable Timer 2 periph clks

**TimerConfigure**(TIMER2\_BASE, TIMER\_CFG\_PERIODIC); // cfg Timer 2 mode - periodic

ui32Period = (**SysCtlClockGet**() /2); // period = CPU clk div 2 (500ms)

**TimerLoadSet**(TIMER2\_BASE, TIMER\_A, ui32Period); // set Timer 2 period

**TimerIntEnable**(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT); // enables Timer 2 to interrupt CPU

**TimerEnable**(TIMER2\_BASE, TIMER\_A); // enable Timer 2

}

//---------------------------------------------------------------------------

// mailbox\_queue Task() - Run by BIOS\_Start(), then unblocked by Timer ISR

//

// Places state of LED (msg.val) into a mailbox for ledToggle() to use

//---------------------------------------------------------------------------

**void** **mailbox\_queue**(**void**)

{

//---------------------------------

// msg used for Mailbox and Queue

//---------------------------------

MsgObj msg; // create an instance of MsgObj named msg

//---------------------------------

// msgp used for Queue only

//---------------------------------

Msg msgp; // Queues pass POINTERS, so we need a pointer of type Msg

msgp = &msg; // init message pointer to address of msg

msg.val = 1; // set initial value of msg.val (LED state)

**while**(1){

msg.val ^= 1; // toggle msg.val (LED state)

Semaphore\_pend(mailbox\_queue\_Sem, BIOS\_WAIT\_FOREVER); // wait on semaphore from Timer ISR

//------------------------------

// MAILBOX CODE follows...

//------------------------------

// Mailbox\_post (LED\_Mbx, &msg, BIOS\_WAIT\_FOREVER); // post msg containing LED state into the MAILBOX

//------------------------------

// QUEUE CODE follows...

//------------------------------

Queue\_put(LED\_Queue, (Queue\_Elem\*)msgp); // pass pointer to Message object via LED\_Queue

Semaphore\_post (QueSem); // unblock Queue\_get to get msg

}

}

//---------------------------------------------------------------------------

// ledToggle() - called by BIOS\_Start(), then unblocked by mailbox\_queue()

//

// toggles LED on Tiva-C LaunchPad

//---------------------------------------------------------------------------

**void** **ledToggle**(**void**)

{

//---------------------------------

// msg used for Mailbox and Queue

//---------------------------------

MsgObj msg; //define msg using MsgObj struct created earlier

//---------------------------------

// msgp used for Queue only

//---------------------------------

Msg msgp; //define pointer to MsgObj to use with queue put/get

msgp = &msg; //init msgp to point to address of msg (used for put/get)

**while**(1)

{

//------------------------------

// MAILBOX CODE follows...

//------------------------------

// Mailbox\_pend(LED\_Mbx, &msg, BIOS\_WAIT\_FOREVER); // wait/block until post of msg, get msg.val

//------------------------------

// QUEUE CODE follows...

//------------------------------

Semaphore\_pend(QueSem, BIOS\_WAIT\_FOREVER); // unblocked by mailbox\_queue() when Queue has msg

msgp = Queue\_get(LED\_Queue); // read contents of queue to get value of LED state

// LED values - 0=OFF, 2=RED, 4=BLUE, 8=GREEN

// if (msg.val) // MAILBOX "if" - msg.val contains LED state

**if**(msgp->val) // QUEUE "if" - mspg->val contains LED state for QUEUE's the use pointers

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 8); // turn LED on

}

**else**

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 0); // turn LED off

}

i16ToggleCount += 1; // keep track of #toggles

Log\_info1("LED TOGGLED [%u] TIMES",i16ToggleCount); // send toggle count to UIA

}

}

//---------------------------------------------------------------------------

// Timer\_ISR()

//

// Called by Hwi when timer hits zero

//

// TimerIntClear is needed here because THIS fxn is the ISR now

//---------------------------------------------------------------------------

**void** **Timer\_ISR**(**void**)

{

**TimerIntClear**(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT); // must clear timer flag FROM timer

Semaphore\_post(mailbox\_queue\_Sem); // post Sem to unblock mailbox-queue-task

}



