Introduction:

This lab demonstrates the proper use of RTOS concepts such as semaphores and mutexes. The source code is provided in order of the lab_8 header file, the main routine source code, and the threads in ascending order (as documented). The Serial.c, Serial.h, and RTOS-specific files have been omitted due to no modification deviation from the vendor-provided code.

Appendix: Source code

```
#ifndef example_lab_8
     #define example_lab_8


     #include "stdint.h"
     #include "Board_Joystick.h"
  #include "Board_GLCD.h"


     #include "cmsis_os2.h"
     #include "rtx_os.h"


     /**
     * Modified from assignment provided sources:
     * Author of modifications: Peter A. Dranishnikov
     * Lab #: 8
     * Course: EEL4685C
     * Due date: April 23, Spring 2019
     */


     // shared variables
     extern uint32_t treceiveChar;
     extern uint32_t ttimerSec;
     extern uint32_t tgear;
     extern uint32_t joychar;
```

```
	// mutexes
	// Note, actual variable declarations are in main.c, since exactly one file must create the space.
	// The extern statements tell other files that the variable already exists, and they are allowed to access it.
	extern osMutexId_t mut1Display;
	extern osMutexId_t mut2Ser; //mutex for serial output

	// flag semaphores
	extern osSemaphoreId_t semTick;
	extern osSemaphoreId_t semChar;
	extern osSemaphoreId_t semSM;
	//extern osSemaphoreId_t semDisp; //not needed, semTick handles display thread

	//threads
	extern osThreadId_t tid_thdDisplay;
	int Init_thdDisplay (void);

	extern osThreadId_t tid_thdTick;
	int Init_thdTick (void);

	extern osThreadId_t tid_thdChar;
	int Init_thdChar (void);

	extern osThreadId_t tid_thdJoystick;
	int Init_thdJoystick (void);
```

```
	//New: sm thread declaration
	extern osThreadId_t tid_thdStateMachine;
	int Init_thdStateMachine (void);

#endif

/**
	* Modified from assignment provided sources:
	* Author of modifications: Peter A. Dranishnikov
	* Lab #: 8
	* Course: EEL4685C
	* Due date: April 23, Spring 2019
	*/
/*----------------------------------------------------------------------
-------------
 * CMSIS-RTOS 'main' function
 * Contains all thread initialization (with support function),
mutexes,
 * semaphores, and an IRQHandler for serial input
 *----------------------------------------------------------------------
-------------*/



#include "RTE_Components.h"
#include  CMSIS_device_header
#include "serial.h"
#include "GLCD_Config.h"
#include "stm32f2xx_hal.h"

#include "lab_8.h"    // specific to this project
```

```
extern GLCD_FONT GLCD_Font_16x24;


// Note that the main file declares the space for all the
system's variables (someone has to),

// and "extern" declarations are in the lab_8.h file so other
files can find them


osMutexId_t mut1Display;

osMutexId_t mut2Ser;

osSemaphoreId_t semTick;

osSemaphoreId_t semChar;

osSemaphoreId_t semSM;

//osSemaphoreId_t semDisp;


uint32_t treceiveChar;


void serial_first_message(void);


/*****************************************/

// The RTOS and HAL need the SysTick for timing. The RTOS wins
and gets control

// of SysTick, so we need to route the HAL's tick call to the
RTOS's tick.

// Don't mess with this code.

uint32_t HAL_GetTick(void) {

  return osKernelGetTickCount();

}
```

```
/*-------------------------------------------------------------
-------------

 * HW Init - since the HAL depends on a periodic timer, we need
the RTOS

 * in order for several HW devices to initialize correctly, like
the GLCD

 *-------------------------------------------------------------
------------*/

void app_hw_init (void *argument) {


     GLCD_Initialize();

     GLCD_SetBackgroundColor(GLCD_COLOR_PURPLE);

     GLCD_SetForegroundColor(GLCD_COLOR_WHITE);

     GLCD_ClearScreen();

     GLCD_SetFont(&GLCD_Font_16x24);


     Joystick_Initialize();  // Note: Joystick now uses HAL
functions


     SER_Init(115200);   // 115200 baud, 8 data bits, 1 stop
bits, no flow control

     // configures and enables the interrupt for the USART 3
serial port.

     USART3->CR1 |= USART_CR1_RXNEIE;

     NVIC->ISER[ USART3_IRQn/32] =  (1UL << (USART3_IRQn%32));

     NVIC->IP[USART3_IRQn] = 0x80;

     serial_first_message();//print first serial message


     // Create other threads here so that all initialization is
done before others get scheduled.

     Init_thdStateMachine();
```

```
	Init_thdDisplay();
	Init_thdTick();
	Init_thdChar();
	Init_thdJoystick();

  osThreadExit(); // job is done, thread suicide. There better
be other threads created above...
}

int main (void) {

	SystemCoreClockUpdate(); // always first, make sure the
clock freq. is current
	osKernelInitialize();     // Initialize CMSIS-RTOS
	HAL_Init();

	mut1Display = osMutexNew(NULL);
		if (mut1Display==NULL) while(1){} // failed, scream
and die
	mut2Ser = osMutexNew(NULL);
		if (mut2Ser == NULL) while(1){} //surrender on failure

	semTick = osSemaphoreNew(1000, 1, NULL);
		if (semTick==NULL) while(1){} // failed, scream and
die
	semChar = osSemaphoreNew(1000, 0, NULL);
		if (semChar==NULL) while(1){} // failed, scream and
die
	semSM = osSemaphoreNew(1000, 1, NULL);
		if (semSM == NULL) while(1){} // surrender on failure
```

```
  osThreadNew(app_hw_init, NULL, NULL); // Create application's
main thread to init HW now that HAL is running

  osKernelStart();                      // Start thread
execution

  for (;;) {}
               // should never get here

}



// Interrupt serv routine for the serial port. Triggered upon
receiving any character or space

void USART3_IRQHandler(void)

{


     treceiveChar = SER_GetChar();

     if (treceiveChar == 'D' || treceiveChar == 'd' ||
treceiveChar == 'U' || treceiveChar == 'u')

     {

          osSemaphoreRelease(semSM);

     }

     else; //do nothing

}


void serial_first_message(void)

{

     //"Time\tGear\r\n"; equivalent null-terminated string

     const uint32_t init_message[] = {'T', 'i', 'm', 'e', '\t',
'G', 'e', 'a', 'r', '\r', '\n', '\0'};
```

```
	osMutexAcquire(mut2Ser, osWaitForever);
	for(uint32_t i = 0; init_message[i] != '\0'; i++)
	{
		SER_PutChar(init_message[i]);
	}
	osMutexRelease(mut2Ser);
}

#include "lab_8.h"

/**
	* Modified from assignment provided sources:
	* Author of modifications: Peter A. Dranishnikov
	* Lab #: 8
	* Course: EEL4685C
	* Due date: April 23, Spring 2019
	*/
/*-----------------------------------------------------------------
--------------
 *	Thread 1 'Thread_Char': Display character thread
 * Displays the current gear to the main onboard display.
Initializes on boot
 * to display the current gear (at initialization, 0)
 * Triggers on the character semaphore
 *-----------------------------------------------------------------
-------------*/

void thdChar (void *argument);
// thread function
```

```
osThreadId_t tid_thdChar;
// thread id


int Init_thdChar (void) {
  tid_thdChar = osThreadNew (thdChar, NULL, NULL);
  if (!tid_thdChar) return(-1);

  osMutexAcquire(mut1Display, osWaitForever);
     GLCD_DrawChar(100,100,(tgear + 48));
  osMutexRelease(mut1Display);

  return(0);
}

void thdChar (void *argument)
{

     while (1)
     {
          osSemaphoreAcquire(semChar, osWaitForever);


          osMutexAcquire(mut1Display, osWaitForever);
               GLCD_DrawChar(100,100,(tgear + 48));
          osMutexRelease(mut1Display);

     }
}
```

```
#include "lab_8.h"
#include "serial.h"
 /**
     * Modified from assignment provided sources:
     * Author of modifications: Peter A. Dranishnikov
     * Lab #: 8
     * Course: EEL4685C
     * Due date: April 23, Spring 2019
     */
/*-----------------------------------------------------------------
--------------
 *      Thread 2 'Thread_Display': Serial log thread
 * This may sound like a misnomer, but it indeed "displays" to
the serial port.
 * Data displayed include the thread timer's value and the
current gear
 * Triggers on the tick semaphore
 *-----------------------------------------------------------------
-------------*/


void thdDisplay (void *argument);
// thread function
osThreadId_t tid_thdDisplay;
// thread id


int Init_thdDisplay (void) {
  tid_thdDisplay = osThreadNew(thdDisplay, NULL, NULL);
  if (!tid_thdDisplay) return(-1);
  return(0);
}
```

```
void thdDisplay (void *argument)
{
     //placeholder text, first 8 characters overritten
     uint32_t text[11] = {'0','0','0','.','0','0',' 
','g','\r','\n','\0'};
     uint32_t timerDecomp[6];
     while (1)
     {
          osSemaphoreAcquire(semTick, osWaitForever);

          //isolate decimal digits for display
          timerDecomp[5] = ttimerSec % 10;
          timerDecomp[4] = (ttimerSec % 100) / 10;
          timerDecomp[2] = (ttimerSec % 1000) / 100;
          timerDecomp[1] = (ttimerSec % 10000) / 1000;
          timerDecomp[0] = (ttimerSec % 100000) / 10000; //don't 
know of a better way

          //convert to ascii
          for(uint32_t i = 0; i < 6; i++)
          {
               if(i != 3)
               {
                    text[i] = timerDecomp[i] + 48;
               }
          }
          text[7] = tgear + 48;

          //dump to serial
```

```
          osMutexAcquire(mut2Ser, osWaitForever);
          for(uint32_t i = 0; text[i] != '\0'; i++)
               SER_PutChar(text[i]);
          osMutexRelease(mut2Ser);

     }
}

#include "lab_8.h"
/**
     * Modified from assignment provided sources:
     * Author of modifications: Peter A. Dranishnikov
     * Lab #: 8
     * Course: EEL4685C
     * Due date: April 23, Spring 2019
     */
/*-------------------------------------------------------------
--------------
 *      Thread 3 'Thread_Joystick': Joystick thread
 * This thread is on a perpetual poll for new joystick positions
 * This thread does not trigger on any semaphore
 * This file used to have a drunk indentation format, which
should be corrected
 * However, there may be a remaining combination of tabs and
spaces
 * I apologize for any inconvenience this may cause when parsing
 *-------------------------------------------------------------
-------------*/
```

```
void thdJoystick (void *argument);
// thread function

osThreadId_t tid_thdJoystick;
// thread id

uint32_t joychar = ' ';


int Init_thdJoystick (void) {


  tid_thdJoystick = osThreadNew (thdJoystick, NULL, NULL);

  if (!tid_thdJoystick) return(-1);


  return(0);

}


void thdJoystick (void *argument) {


     uint32_t newjoystick, joystick;


     while (1)

     {

          osDelay(osKernelGetTickFreq()/10);


          newjoystick = Joystick_GetState();

          if (joystick != newjoystick)

          {

               switch (newjoystick)

               {

                    case JOYSTICK_UP:

                    {
```

```
                    joychar = 'U';
                    osSemaphoreRelease(semSM);
                    break;
                }
                case JOYSTICK_DOWN:
                {
                    joychar = 'D';
                    osSemaphoreRelease(semSM);
                    break;
                }
                default:
                    break;
            }


        }
        joystick = newjoystick;
    }
}

#include "lab_8.h"
/**
    * Modified from assignment provided sources:
    * Author of modifications: Peter A. Dranishnikov
    * Lab #: 8
    * Course: EEL4685C
    * Due date: April 23, Spring 2019
    */
```

```
/*-------------------------------------------------------------
-------------

 *      Thread 4 'Thread_StateMachine': State Machine thread

 * This thread is the primary state machine of a supposed gear
shifter

 * Triggers on the statemachine semaphore

 *

 *-------------------------------------------------------------
-------------*/


void thdStateMachine(void *argument);

void fsm_reaction(bool up, bool down);

osThreadId_t tid_thdStateMachine;

uint32_t tgear;


int Init_thdStateMachine(void)

{

     tid_thdStateMachine = osThreadNew(thdStateMachine, NULL,
NULL);

     if(!tid_thdStateMachine) return(-1);

     tgear = 0;

     return(0);

}


void thdStateMachine(void *argument)

{

     while(1)

     {

          osSemaphoreAcquire(semSM, osWaitForever);
```

```
          bool u = joychar == 'D' || (treceiveChar == 'U' ||
treceiveChar == 'u');

          bool d = joychar == 'U' || (treceiveChar == 'D' ||
treceiveChar == 'd');

          fsm_reaction(u, d);

          //IMPORTANT: these values must be set to zero for the
state machine to transition correctly

          treceiveChar = 0x00;

          joychar = 0x00;

     }

}


void fsm_reaction(bool up, bool down)

{


     switch (tgear)

     {


          case 0:

               if (up && !down)

               {

                    tgear = 1;

                    osSemaphoreRelease(semChar);

                    osSemaphoreRelease(semTick);

               }

               else if (up && down)

               {

                    tgear = 0;

                    osSemaphoreRelease(semChar);

                    osSemaphoreRelease(semTick);
```

```
            }
            else if (!up && down)
            {
                  tgear = 0;
                  osSemaphoreRelease(semChar);
                  osSemaphoreRelease(semTick);

            }
            else
            {
                  osSemaphoreRelease(semChar);
                  osSemaphoreRelease(semTick);
            }
            break;
      case 1:
            if (up && !down)
            {
                  tgear = 2;
                  osSemaphoreRelease(semChar);
                  osSemaphoreRelease(semTick);

            }
            else if (up && down)
            {
                  osSemaphoreRelease(semChar);
                  osSemaphoreRelease(semTick);
                  tgear = 1;
                  osSemaphoreRelease(semChar);
```

```
                osSemaphoreRelease(semTick);
            }
            else if (!up && down)
            {
                tgear = 0;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else;
            break;
        case 2:
            if (up && !down)
            {
                tgear = 3;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (up && down)
            {
                tgear = 2;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (!up && down)
            {
                tgear = 1;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
```

```
            }
            else;
            break;
        case 3:
            if (up && !down)
            {
                tgear = 4;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (up && down)
            {
                tgear = 3;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (!up && down)
            {
                tgear = 2;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else;
            break;
        case 4:
            if (up && !down)
            {
                tgear = 5;
```

```
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (up && down)
            {
                tgear = 4;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (!up && down)
            {
                tgear = 3;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else;
            break;
        case 5:
            if (up && !down)
            {
                tgear = 6;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (up && down)
            {
                tgear = 5;
                osSemaphoreRelease(semChar);
```

```
                osSemaphoreRelease(semTick);
            }
            else if (!up && down)
            {
                tgear = 4;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else;
            break;
        case 6:
            if (up && !down)
            {
                tgear = 6;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (up && down)
            {
                tgear = 6;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
            }
            else if (!up && down)
            {
                tgear = 5;
                osSemaphoreRelease(semChar);
                osSemaphoreRelease(semTick);
```

```
                }
                else;
                break;
          default:
                for(;;){}
     }
}


#include "cmsis_os2.h"                                          //
CMSIS RTOS header file
#include "lab_8.h"
/**
     * Modified from assignment provided sources:
     * Author of modifications: Peter A. Dranishnikov
     * Lab #: 8
     * Course: EEL4685C
     * Due date: April 23, Spring 2019
     */
/*-------------------------------------------------------------
-------------
 *      Thread 5 'Thread_tick': Ticker thread
 * This thread ticks. Thats it.
 * This thread functions autonomously after init, but the delay
can be adjusted
 * at compile time using the tickTime constant
 *-------------------------------------------------------------
------------*/


void thdTick (void *argument);
// thread function
```

```
osThreadId_t tid_thdTick;
// thread id

const uint32_t tickTime = 100;

uint32_t ttimerSec;


int Init_thdTick (void) {
     tid_thdTick = osThreadNew (thdTick, NULL, NULL);
     if (!tid_thdTick) return(-1);
     ttimerSec = 0;
     return(0);
}


void thdTick (void *argument) {


     while (1)
     {
          osSemaphoreRelease(semTick);
          ttimerSec++;
          osDelay(osKernelGetTickFreq()/tickTime);


     }
}
```