Author: Peter A. Dranishnikov
Lab #: 7
Course: EEL4768C
Due date: March 25th, Spring 2019
Below is lab7.c source code:

```
/* Author: Peter A. Dranishnikov
 * Lab #: 7
 * Course: EEL4768C
 * Due date: March 25th, Spring 2019
 */


// These include headers refer to files supplied with the Keil uVision installation
#include "stm32f2xx_hal.h"  // when using HAL for control of HW config/interface
#include "Board_GLCD.h"            // functions for GLCD
#include "GLCD_Config.h"           // constants needed as inputs to GLCD functions


// These include headers are supplied by the professor (earlier less-integrated files from Keil)
// that will temporarily allow easier access to some board peripherals
#include "serial.h"      // RS-232 driver
#include "I2C.h"         // required for communication with the joystick hardware
#include "JOY.h"         // the joystick driver the Keil board


extern GLCD_FONT GLCD_Font_16x24;    // Used by GLCD_SetFont, references the 16 pixel wide by 24 pixels tall font


// We humans don't care what the numbers are, so let C's "enum" statement determine it - also helps
// with warnings when compiling
```

Author: Peter A. Dranishnikov
Lab #: 7
Course: EEL4768C
Due date: March 25th, Spring 2019

```
enum STATE {GEAR_0, GEAR_1, GEAR_2, GEAR_3, GEAR_4, GEAR_5,
GEAR_6};


enum STATE state = GEAR_0;    // sets initial state



void fsm_reaction(bool, bool);


int main(void)

{

     SystemCoreClockUpdate();  // Makes sure that the variable
tracking the cycles/second (declared in

     // system_stm32f2xx.h) is consistent with the initialized
clock rate - i.e. always call this first


     JOY_Init();

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


     HAL_Init();              // Uses provided functions/config
values to setup the HAL, also enables the SysTick Interrupt.


     GLCD_Initialize();

     GLCD_SetBackgroundColor(GLCD_COLOR_PURPLE);

     GLCD_ClearScreen();

     GLCD_SetFont(&GLCD_Font_16x24);


     // 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;


	//set inital state to screen & serial (for user
friendliness)

	fsm_reaction(false,false);


	for(;;)

	{

		// infinite loop - background code


	}

}



// The SysTick Handler is configured by the HAL to interrupt
every millisecond.

// One way to create periodic actions is to count the desired
number of interrupts.

// The example below generates two periodic actions.


void SysTick_Handler(void)

{


	//const uint32_t PERIOD_IN_MSEC1 = 1000;

	//const uint32_t PERIOD_IN_MSEC2 = 5000;


	static int32_t joystick;

	int32_t newjoystick;
```

Author: Peter A. Dranishnikov
Lab #: 7
Course: EEL4768C
Due date: March 25th, Spring 2019

```
	//static uint32_t text = 'A';

	//static uint32_t count1 = 0, count2 = 0;


	HAL_IncTick(); // This increments the internal clock of the
micro- THIS *MUST* be in the SysTick_Handler!!!


	// periodic process 1 - prints a character to the upper
left of the LCD display on the board

	/*

	count1++;

	if (count1 >= PERIOD_IN_MSEC1)

	{

		count1 = 0;

		GLCD_DrawChar(0,0,text);

		text++;

		if (text > 'Z') text = 'A';

	}


	// periodic process 2 - triggers a reaction based on the
tick input

	// A tick is generated periodically and by manually
pressing the joystick (sometimes useful for debugging)

	count2++;

	if (count2 >= PERIOD_IN_MSEC2)

	{

		count2 = 0;

		fsm_reaction(true, false);

	}
```

Author: Peter A. Dranishnikov
Lab #: 7
Course: EEL4768C
Due date: March 25th, Spring 2019

```
	*/
	newjoystick = JOY_GetKeys();
	if (joystick != newjoystick && joystick == JOY_UP)
	{
		fsm_reaction(true, false);
	}
	else if (joystick != newjoystick && joystick == JOY_DOWN)
	{
		fsm_reaction(false, true);
	}
	else;
	joystick = newjoystick;
}



void fsm_reaction(bool up, bool down)
{
	/*
	Lab Spec:
	Alter the program to implement the finite state machine [image].

	The upshift input (up) to the FSM must come from receiving a ‘U’ or ‘u’ on the serial port, or detecting that the joystick was pressed downward relative to the screen.
	The downshift input (dn) to the FSM must come from receiving a ‘D’ or ‘d’ on the serial port, or detecting that the joystick was pressed upward relative to the screen.
```

Author: Peter A. Dranishnikov
Lab #: 7
Course: EEL4768C
Due date: March 25th, Spring 2019

```
     The output gear must write the corresponding character to
both the middle of the graphic LCD screen and to the serial
port.


     Inputs: up, dn : pure

     Output: gear : character


     */

     switch (state)

     {

          //NOTE: for readability purposes, all transitions
(including same state) are mapped as if statements

          case GEAR_0:

               if (up && !down) // check guard on transition
arrow.

               {

                    SER_PutChar('1');   // generate output

                    GLCD_DrawChar(0,0,'1');

                                             // Perform action
if an extended FSM

                    state = GEAR_1;     // set new state

               }

               else if (up && down)

               {

                    SER_PutChar('0');

                    GLCD_DrawChar(0,0,'0');

                    state = GEAR_0;

               }

               else if (!up && down)
```

```
                {
                    SER_PutChar('0');
                    GLCD_DrawChar(0,0,'0');
                    state = GEAR_0;
                }
                else
                {
                    SER_PutChar('0');
                    GLCD_DrawChar(0,0,'0');
                }
                break;
            case GEAR_1:
                if (up && !down)
                {
                    SER_PutChar('2');
                    GLCD_DrawChar(0,0,'2');

                    state = GEAR_2;
                }
                else if (up && down)
                {
                    SER_PutChar('1');
                    GLCD_DrawChar(0,0,'1');
                    state = GEAR_1;
                }
                else if (!up && down)
                {
                    SER_PutChar('0');
```

```
                    GLCD_DrawChar(0,0,'0');
                    state = GEAR_0;
               }
               else;
               break;
          case GEAR_2:
               if (up && !down)
               {
                    SER_PutChar('3');
                    GLCD_DrawChar(0,0,'3');

                    state = GEAR_3;
               }
               else if (up && down)
               {
                    SER_PutChar('2');
                    GLCD_DrawChar(0,0,'2');

                    state = GEAR_2;
               }
               else if (!up && down)
               {
                    SER_PutChar('1');
                    GLCD_DrawChar(0,0,'1');
                    state = GEAR_1;
               }
               else;
               break;
```

```
            case GEAR_3:
                 if (up && !down)
                 {
                      SER_PutChar('4');
                      GLCD_DrawChar(0,0,'4');

                      state = GEAR_4;
                 }
                 else if (up && down)
                 {
                      SER_PutChar('3');
                      GLCD_DrawChar(0,0,'3');

                      state = GEAR_3;
                 }
                 else if (!up && down)
                 {
                      SER_PutChar('2');
                      GLCD_DrawChar(0,0,'2');
                      state = GEAR_2;
                 }
                 else;
                 break;
            case GEAR_4:
                 if (up && !down)
                 {
                      SER_PutChar('5');
                      GLCD_DrawChar(0,0,'5');
```

```
                    state = GEAR_5;
               }
               else if (up && down)
               {
                    SER_PutChar('4');
                    GLCD_DrawChar(0,0,'4');

                    state = GEAR_4;
               }
               else if (!up && down)
               {
                    SER_PutChar('3');
                    GLCD_DrawChar(0,0,'3');
                    state = GEAR_3;
               }
               else;
               break;
          case GEAR_5:
               if (up && !down)
               {
                    SER_PutChar('6');
                    GLCD_DrawChar(0,0,'6');

                    state = GEAR_6;
               }
               else if (up && down)
               {
```

```
                    SER_PutChar('5');
                    GLCD_DrawChar(0,0,'5');

                    state = GEAR_5;
               }
               else if (!up && down)
               {
                    SER_PutChar('4');
                    GLCD_DrawChar(0,0,'4');
                    state = GEAR_4;
               }
               else;
               break;
          case GEAR_6:
               if (up && !down)
               {
                    SER_PutChar('6');
                    GLCD_DrawChar(0,0,'6');

                    state = GEAR_6;
               }
               else if (up && down)
               {
                    SER_PutChar('6');
                    GLCD_DrawChar(0,0,'6');

                    state = GEAR_6;
               }
```

```
                else if (!up && down)
                {
                     SER_PutChar('5');
                     GLCD_DrawChar(0,0,'5');
                     state = GEAR_5;
                }
                else;
                break;
          default:
                for(;;){}  // infinite loop to catch incorrect
execution
     }
}



// Interrupt service routine for the serial port. It is
triggered upon receiving any character
void USART3_IRQHandler(void)
{
          int32_t treceiveChar;
          treceiveChar = SER_GetChar();

          if (treceiveChar == 'd' || treceiveChar == 'D')
          {
                fsm_reaction(false, true);
          }
          else if(treceiveChar == 'u' || treceiveChar == 'U')
```

Author: Peter A. Dranishnikov
Lab #: 7
Course: EEL4768C
Due date: March 25th, Spring 2019

```
            {
                  fsm_reaction(true, false);
            }
            else;
}
```