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

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;

//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);

}

\*/

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.

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')

{

fsm\_reaction(true, false);

}

else;

}