/\*

\* KEYPAD AND SEVEN-SEGEMENT DECODER IMPLEMENTATION FOR PART-1 LAB-1

\* The code will read the key pressed from the PMOD kypd module and the corresponding value will be displayed on both the SSDs (i.e., left as well as right)

\* Initially, the delay of 250 ms is used between left and right segment switching so user can clearly see that both segment do not appear to lit at the same time.

\* If user decreases this time to say, 10 ms, it will appear as if both the left and right segments lits at the same time.

\*

\* ECE- 315 WINTER 2021 - COMPUTER INTERFACING COURSE

\*

\* Created on: Date:::5 February, 2021

\* Author: Shyama M. Gandhi

\*/

//Include FreeRTOS Library

#include "FreeRTOS.h"

#include "task.h"

#include "queue.h"

#include "xparameters.h"

#include "xgpio.h"

#include "xscugic.h"

#include "xil\_exception.h"

#include "xil\_printf.h"

#include "pmodkypd.h"

#include "sleep.h"

#include "xil\_cache.h"

// Parameter definitions

#define SSD\_DEVICE\_ID XPAR\_AXI\_GPIO\_PMOD\_SSD\_DEVICE\_ID

#define KYPD\_DEVICE\_ID XPAR\_AXI\_GPIO\_PMOD\_KEYPAD\_DEVICE\_ID

//Button Variable

XGpio SSDInst, KYPDInst;

/\* The Tx described at the top of this file. \*/

static void prvTxTask( void \*pvParameters );

void DemoInitialize();

u32 SSD\_decode(u8 key\_value,u8 cathode);

PmodKYPD myDevice;

/\*-----------------------------------------------------------\*/

static TaskHandle\_t xTxTask;

// keytable is determined as follows (indices shown in Keypad position below)

// 12 13 14 15

// 8 9 10 11

// 4 5 6 7

// 0 1 2 3

#define DEFAULT\_KEYTABLE "0FED789C456B123A"

void DemoInitialize() {

KYPD\_begin(&myDevice, XPAR\_AXI\_GPIO\_PMOD\_KEYPAD\_BASEADDR);

KYPD\_loadKeyTable(&myDevice, (u8\*) DEFAULT\_KEYTABLE);

}

u32 SSD\_decode(u8 key\_value, u8 cathode){

switch(key\_value){

case 48: if(cathode==0) return 0b00111111; else return 0b10111111;

case 49: if(cathode==0) return 0b00000110; else return 0b10000110;

case 50: if(cathode==0) return 0b01011011; else return 0b11011011;

case 51: if(cathode==0) return 0b01001111; else return 0b11001111;

case 52: if(cathode==0) return 0b01100110; else return 0b11100110;

case 53: if(cathode==0) return 0b01101101; else return 0b11101101;

case 54: if(cathode==0) return 0b01111101; else return 0b11111101;

case 55: if(cathode==0) return 0b00000111; else return 0b10000111;

case 56: if(cathode==0) return 0b01111111; else return 0b11111111;

case 57: if(cathode==0) return 0b01101111; else return 0b11101111;

case 65: if(cathode==0) return 0b01110111; else return 0b11110111;

case 66: if(cathode==0) return 0b01111100; else return 0b11111100;

case 67: if(cathode==0) return 0b00111001; else return 0b10111001;

case 68: if(cathode==0) return 0b01011110; else return 0b11011110;

case 69: if(cathode==0) return 0b01111001; else return 0b11111001;

case 70: if(cathode==0) return 0b01110001; else return 0b11110001;

default: if(cathode==0) return 0b00000000; else return 0b00000000;

}

}

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

// MAIN FUNCTION

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

int main (void)

{

int status;

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

// INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO

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

// Initialize SSD

status = XGpio\_Initialize(&SSDInst, SSD\_DEVICE\_ID);

if(status != XST\_SUCCESS){

xil\_printf("GPIO Initialization for SSD unsuccessful.\r\n");

return XST\_FAILURE;

}

// Set SSD direction to output

XGpio\_SetDataDirection(&SSDInst, 1, 0x00);

xil\_printf("Initialization Complete, System Ready!\n");

/\* Create the two tasks. The Tx task is given a lower priority than the

Rx task, so the Rx task will leave the Blocked state and pre-empt the Tx

task as soon as the Tx task places an item in the queue. \*/

xTaskCreate( prvTxTask, /\* The function that implements the task. \*/

( const char \* ) "Tx", /\* Text name for the task, provided to assist debugging only. \*/

configMINIMAL\_STACK\_SIZE, /\* The stack allocated to the task. \*/

NULL, /\* The task parameter is not used, so set to NULL. \*/

tskIDLE\_PRIORITY, /\* The task runs at the idle priority. \*/

&xTxTask );

DemoInitialize();

vTaskStartScheduler();

while(1);

return 0;

}

/\*-----------------------------------------------------------\*/

static void prvTxTask( void \*pvParameters )

{

for( ;; ) {

u16 keystate;

XStatus status, last\_status = KYPD\_NO\_KEY;

u8 key, last\_key = 'x';

u32 ssd\_value=0;

// Initial value of last\_key cannot be contained in loaded KEYTABLE string

Xil\_Out32(myDevice.GPIO\_addr, 0xF);

xil\_printf("Pmod KYPD demo started. Press any key on the Keypad.\r\n");

while (1) {

// Capture state of each key

keystate = KYPD\_getKeyStates(&myDevice);

// Determine which single key is pressed, if any

status = KYPD\_getKeyPressed(&myDevice, keystate, &key);

// Print key detect if a new key is pressed or if status has changed

if (status == KYPD\_SINGLE\_KEY

&& (status != last\_status || key != last\_key)) {

xil\_printf("Key Pressed: %c\r\n", (char) key);

last\_key = key;

}

else if (status == KYPD\_MULTI\_KEY && status != last\_status)

xil\_printf("Error: Multiple keys pressed\r\n");

last\_status = status;

XGpio\_DiscreteWrite(&SSDInst, 1, 0b10000000);

ssd\_value = SSD\_decode(key, 0);

XGpio\_DiscreteWrite(&SSDInst, 1, ssd\_value );

vTaskDelay(pdMS\_TO\_TICKS(10));

XGpio\_DiscreteWrite(&SSDInst, 1, 0b00000000);

ssd\_value = SSD\_decode(key, 1);

XGpio\_DiscreteWrite(&SSDInst, 1, ssd\_value);

vTaskDelay(pdMS\_TO\_TICKS(10));

}

}

}

/\*-----------------------------------------------------------\*/