|  |  |
| --- | --- |
| I/O and Interrupts.  24th October 2019.  Prof. Randall Brouwer | Daniel Ackuaku |

*C Code for the lab*

#include "altera\_up\_avalon\_character\_lcd.h"

#include "altera\_avalon\_timer\_regs.h"

#include "altera\_up\_avalon\_parallel\_port.h"

#include "altera\_up\_avalon\_parallel\_port\_regs.h"

#include "altera\_avalon\_timer.h"

#include <stdio.h>

#include "sys/alt\_timestamp.h"

#include "alt\_types.h"

#include "system.h"

#include "altera\_avalon\_pio\_regs.h"

#include "alt\_types.h"

#include "sys/alt\_irq.h"

#define KEY3\_MASK 0x08 //obaitned from memeory tab in altera by pressing keys

#define KEY2\_MASK 0x04

#define KEY1\_MASK 0x02

/\* Declare a global variable to hold the edge capture value. \*/

volatile int edge\_capture;

static void handle\_button\_interrupts(void\* context, alt\_u32 id) {

/\* cast the context pointer to an integer pointer. \*/

volatile int\* edge\_capture\_ptr = (volatile int\*) context;

/\* Read the edge capture register on the button PIO. Store value. \*/

\*edge\_capture\_ptr = IORD\_ALTERA\_AVALON\_PIO\_EDGE\_CAP(KEY\_BASE);

/\* Write to the edge capture register to reset it. \*/

IOWR\_ALTERA\_AVALON\_PIO\_EDGE\_CAP(KEY\_BASE, 0x0);

/\* reset interrupt capability for the Button PIO. \*/

IOWR\_ALTERA\_AVALON\_PIO\_IRQ\_MASK(KEY\_BASE, 0xf);

}

/\* Initialize the button\_pio. \*/

static void init\_KEY\_pio() {

/\* Recast the edge\_capture pointer to match the alt\_irq\_register() function prototype. \*/

void\* edge\_capture\_ptr = (void\*) &edge\_capture;

/\* Enable all 4 button interrupts. \*/

IOWR\_ALTERA\_AVALON\_PIO\_IRQ\_MASK(KEY\_BASE, 0xf);

/\* Reset the edge capture register. \*/

IOWR\_ALTERA\_AVALON\_PIO\_EDGE\_CAP(KEY\_BASE, 0x0);

/\* Register the ISR. \*/

alt\_irq\_register( KEY\_IRQ,edge\_capture\_ptr,handle\_button\_interrupts );

}

static void timerInit() {

alt\_avalon\_timer\_sc\_init (TIMER\_0\_BASE, TIMER\_0\_IRQ\_INTERRUPT\_CONTROLLER\_ID,

TIMER\_0\_IRQ, TIMER\_0\_FREQ);

}

int main(void)

{

//Declaration of constants and variables

timerInit();

init\_KEY\_pio();

int flashCount;

int y;

int hexOn;

char time[8];

char copyTime[8] = "\0";

unsigned long ticks = 0;

unsigned long resetTicks = 0;

// declaration of global vairables for the LCD, Timer and Key ports.

alt\_up\_parallel\_port\_dev \* hexL;

alt\_up\_parallel\_port\_dev \* hexH;

hexL = alt\_up\_parallel\_port\_open\_dev ("/dev/SevenSeg30");

hexH = alt\_up\_parallel\_port\_open\_dev ("/dev/SevenSeg74");

alt\_up\_character\_lcd\_dev \* char\_lcd\_dev;

char\_lcd\_dev = alt\_up\_character\_lcd\_open\_dev ("/dev/LCD");

while (1) {

// 1 sec = 100 ticks; 1 min = 6000 ticks; 10 mins = 60000 ticks

ticks = (alt\_nticks() - resetTicks)/1;

copyTime[7] = '\0';

time[7] = '\0';

time[6] = '0'+((ticks / 10) % 10);

time[5] = '.' ;

time[4] = '0'+((ticks / 100 ) % 10);

time[3] = '0'+((ticks / 1000 ) % 6);

time[2] = ':';

time[1] = '0'+((ticks / 6000 ) % 10);

time[0] = '0'+((ticks / 60000 ) % 10);

for ( y = 0; y < 150000 ; y++) {

asm("nop;nop;nop;");

}

alt\_up\_character\_lcd\_init (char\_lcd\_dev);

alt\_up\_character\_lcd\_string(char\_lcd\_dev, time);

alt\_up\_character\_lcd\_set\_cursor\_pos(char\_lcd\_dev, 0, 1);

alt\_up\_character\_lcd\_string(char\_lcd\_dev, copyTime);

alt\_up\_character\_lcd\_cursor\_blink\_on(char\_lcd\_dev);

/\* Interupt Service routine for the KEY presses \*/

if (edge\_capture & KEY1\_MASK) {

//Reseting the timer

resetTicks = alt\_nticks();

ticks = 0;

copyTime[0] = 0;

edge\_capture = 0;

}

else if (edge\_capture & KEY2\_MASK) {

//Duplicating the time on the second line

strcpy(copyTime, time);

edge\_capture = 0;

}

else if (edge\_capture & KEY3\_MASK){

//flash three times

flashCount = alt\_nticks() + 300;

edge\_capture = 0;

}

//Flashing the HEX displays

if (flashCount >= alt\_nticks()) {

if ((flashCount - alt\_nticks()) % 100 > 50){

if (hexOn == 1) {

alt\_up\_parallel\_port\_write\_data(hexL, 0xffffffff);

alt\_up\_parallel\_port\_write\_data(hexH, 0xffffffff);

hexOn = 0;

}

}

else {

if (hexOn == 0){

alt\_up\_parallel\_port\_write\_data(hexL, 0x00000000);

alt\_up\_parallel\_port\_write\_data(hexH, 0x00000000);

hexOn = 1;

}

}

}

}

return 0;

}