|  |  |
| --- | --- |
| Pulse Width Modulation.  30th October 2019.  Prof. Randall Brouwer | Daniel Ackuaku |

**#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

**#define** AUTO 1

**#define** MANUAL 0

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

**volatile** alt\_32 p\_Edge\_Capture;

**volatile** alt\_u64 Counter = 0;

**static** **void** HandleButtonInterrupts(**void**\* context, alt\_u32 id)

{

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

**volatile** alt\_32\* p\_Edge\_Capture\_Ptr = (**volatile** alt\_32\*) context;

/\*

\* Read the edge capture register on the button PIO.

\* Store value.

\*/

\*p\_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** InitKeyPio()

{

/\* Recast the p\_Edge\_Capture pointer to match the alt\_irq\_register() function prototype. \*/

**void**\* p\_Edge\_Capture\_Ptr = (**void**\*) &p\_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, p\_Edge\_Capture\_Ptr, HandleButtonInterrupts);

}

/\* Initializes the timer and sets the period to 100us \*/

**static** **void** TimerInit()

{

IOWR(TIMER\_0\_BASE, ALTERA\_AVALON\_TIMER\_PERIODL\_REG, 0x1388);

IOWR(TIMER\_0\_BASE, ALTERA\_AVALON\_TIMER\_PERIODH\_REG, 0x0);

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

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

}

/\* Updates the display every 100ms \*/

**static** **void** RefreshDisplay(alt\_8 mode, alt\_up\_character\_lcd\_dev \* char\_lcd\_dev, alt\_8 duty\_cycle)

{

alt\_u8 first\_row[15] = "\0";

alt\_u8 second\_row[15] = "\0";

alt\_u8 operating\_mode[7] = "\0";

alt\_u8 number[3] = "";

strcpy(first\_row, "");

strcat(first\_row, "Dutycycle = ");

number[2] = '0' + (duty\_cycle / 1) % 10;

number[1] = '0' + (duty\_cycle / 10) % 10;

**if** (duty\_cycle >= 100)

{

number[0] = '1';

} **else**

{

number[0] = ' ';

}

strcat(first\_row, number);

strcat(first\_row, "%");

// Output to the LCDS

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

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

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

**if** (mode == AUTO)

{

strcpy(operating\_mode, "Auto");

strcpy(second\_row, "");

strcat(second\_row, "Mode = ");

strcat(second\_row, operating\_mode);

}

**else** **if** (mode == MANUAL)

{

strcpy(operating\_mode, "Manual");

strcpy(second\_row, "");

strcat(second\_row, "Mode = ");

strcat(second\_row, operating\_mode);

}

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

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

}

/\* Flashes the HEX displays in response to the dutyCycle \*/

**static** **void** PwmFunction(alt\_8 duty\_cycle, alt\_up\_parallel\_port\_dev \* hexL)

{

//Flashing the HEX displays

**if** ((alt\_nticks() % 100) < duty\_cycle) {

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

}

**else**

{

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

}

}

**int** main(**void**)

{

//Declaration of constants and variables

TimerInit();

InitKeyPio();

**volatile** alt\_8 duty\_cycle = 0; // The dutycyle did not update untill it was made volatile.

alt\_8 mode = AUTO;

alt\_8 uptick = 1;

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

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

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

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

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

**while** (1)

{

PwmFunction(duty\_cycle, hexL);

/\* Interupt Service routine for the KEY1 press \*/

**if** (p\_Edge\_Capture & KEY1\_MASK)

{

**if** (mode == AUTO)

{

mode = MANUAL;

} **else** **if** (mode == MANUAL)

{

mode = AUTO;

}

p\_Edge\_Capture = 0;

}

// Condition that determines counting up or down

**if** (duty\_cycle == 100)

{

uptick = 0;

}

**else** **if** (duty\_cycle == 0)

{

uptick = 1;

}

**if** ((Counter - alt\_nticks()) <= 0)

{

**if** (mode == AUTO)

{

**if** (uptick == 1)

{

duty\_cycle++;

}

**else** **if** (uptick == 0)

{

duty\_cycle -= 2;

}

}

// Calls for the LCD to be updated

RefreshDisplay(mode, char\_lcd\_dev, duty\_cycle);

Counter = alt\_nticks() + 1000;

}

**if** (mode == MANUAL)

{

/\* Interupt Service routine for the KEY2 press \*/

**if** (p\_Edge\_Capture & KEY2\_MASK)

{

//Increasing the duty\_cycle value

**if** (duty\_cycle <= 5)

{

duty\_cycle = 0;

}

**else**

{

duty\_cycle -= 5;

p\_Edge\_Capture = 0;

}

}

/\* Interupt Service routine for the KEY3 press \*/

**else** **if** (p\_Edge\_Capture & KEY3\_MASK)

{

//Decreasing the duty\_cycle value

**if** (duty\_cycle >= 95)

{

duty\_cycle = 100;

}

**else**

{

duty\_cycle += 5;

p\_Edge\_Capture = 0;

}

}

}

}

**return** 0;

}