TO: Drs. Victor P. Nelson and John Y. Hung

FROM: “Uncovered Warriors,” Stephen Taylor, Brian Arnberg

SECTION: 3

DATE: August 23, 2011

SUBJECT: Interrupt Processing and Keypad Interface

OBJECTIVES

The primary objectives of labs 4 and 5 were to introduce the students to processing interrupts and interfacing with keypads. Lab 4 focused on processing interrupts, specifically XIRQ and IRQ. Lab 5 focused on interfacing with the keypad. The secondary objective of the above labs was to allow the students to practice communication skills while interpreting and executing design criteria.

PROCEDURE

The lab procedures were read in and used to design the code for each lab experiment.

The code for lab 4 was based on code from lab 3. In it the first count updated LEDs at Port T 0-3 twice a second with a decade counter. The second count, which was also a decade counter, updated LEDs at Port AD 0-3 once a second, and either incremented or decremented, depending on which interrupt had last been triggered. The second count initially incremented, but would decrement when XIRQ was triggered (Port E 0). If IRQ (Port E 1) was triggered, the count would increment. Whether the count would increment or decrement was controlled by the value of the variable “dir,” with “dir=0” counting up and “dir=1” counting down. The value of “dir” was set by the interrupt routines. Triggering the interrupts worked as expected, with an XIRQ causing the count to decrement and an IRQ causing the count to increment. Attachment 1a lists the code and Attachment 1b shows screen captures of the count direction being changed by the interrupts.

The code for lab 5 was based on code from lab 4. Port T was used to interface with the keypad (Bits 7-4 being output to the columns and bits 3-0 being inputs from rows). The column lines were pulled-down and the rows pulled up. The input to the interrupt was the AND of all of the rows (implemented with a 7411 chip), so whenever a row voltage went low, it caused the interrupt to be triggered. Row voltages when low whenever a key was pressed; therefore, key presses triggered the interrupt. When the interrupt was triggered, an interrupt routine determined which key was pressed. In order to cause the system to display the key number to the display (Port AD) for 10 seconds, the interrupt routine also set a global countdown variable which was used to display the key press for 10 seconds. Either the count, or the key press, was displayed to Port AD, depending on whether the countdown variable was set. Initially, the program could not correctly detect key presses on the first row. This was because of a delay in the voltage propagation across the keypad following a pattern write. A small delay between when the voltage pattern is written and when the row pattern is checked allowed keys on the first row to be read correctly.

CONCLUSION

The primary objectives were met: Interrupts were correctly processed and the keypad was correctly interfaced. The secondary objective was met: Design criteria were used to determine how to write the code for both experiments, and both sets of code executed correctly. Lab 5 also introduced students to the hardware limitations of a system and coding around those limitations.

Attachment 1a: Lab 4 Program

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

\* Brian Arnberg and Stephen Taylor \*

\* ELEC 3040 - Lab 4, main.c \*

\* Two decade counters, \*

\* PortT (count1) always up \*

\* PortAD (count2) variable direction \*

\* up - XIRQ activated \*

\* down - IRQ activated \*

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

#include <hidef.h> /\* common defines and macros \*/

#include <MC9S12C32.h> /\* derivative information \*/

#pragma LINK\_INFO DERIVATIVE "MC9S12C32"

static unsigned char count1; //Count 1 (Port T)

static unsigned char count2; //Count 2 (Port AD)

char sw1; //state of S1

char sw2; //state of S2

char dir; //count direction (1 is down, 0 is up)

/\* Define the XIRQ service routine \*/

interrupt void XIRQ\_ISR(void) {

dir = 1;//dir = 1 counts down

}

/\* Define the IRQ service routine \*/

interrupt void IRQ\_ISR(void) {

dir = 0;//dir = 0 counts down

}

/\* Delay function - In lab tests show i<17 ~ .5s \*/

void delay (void) {

int i,j;

for (i=0; i<17; i++) { //outer loop, "i<50" seems too slow

for (j=0; j<20000; j++) { //inner loop

}

}

}

/\* Function for the first count \*/

void count\_1 (void) {

if ( count1 == 9 ) { count1 = 0; }

else { count1 +=1; }

PTT = count1; // output count1 to Port T

}

/\* Function for the second count. \*/

void count\_2 (char dir) {

if (dir == 0){//count up

if ( count2 == 9 ) {//if 9 has been reached, go to 0

count2 = 0;

}

else { //else increment by 1

count2 += 1;

}

}

else if (dir == 1){//count down

if (count2 == 0 ) { //if 0 has been reached, go to 9

count2 = 9 ;

}

else { //else decrement by 1

count2 -= 1;

}

}

PTAD = count2;//output count2 to Port AD

}

void main (void) {

/\* DDRA = 0; //set PA0 to 0

DDRB = 0; //set PB4 to 0 \*/

DDRE = 0; //set Port E to read

DDRT = 0xFF; //set PortT to output

DDRAD = 0xFF; //set PortAD to output

count1 = 0; //initialize count1 to 0

count2 = 0; //initialize count2 to 0

dir = 0; //initial direction is up (1 is down)

INTCR\_IRQEN = 1; /\*enable IRQ# interrupts \*/

INTCR\_IRQE = 1; /\*IRQ# interrupts edge-triggered \*/

EnableInterrupts; /\*clear I mask to enable interrupts \*/

\_\_asm ANDCC #0xBF /\*clear X mast to enable XIRQ# \*/

while (1){

delay();//delay for 0.5s

count\_1();//First Count

delay();//delay for 0.5s

count\_1();//First Count

count\_2(dir);//Second Count

} /\* repeat forever \*/

}

Attachment 1b: Lab 4 screenshots.

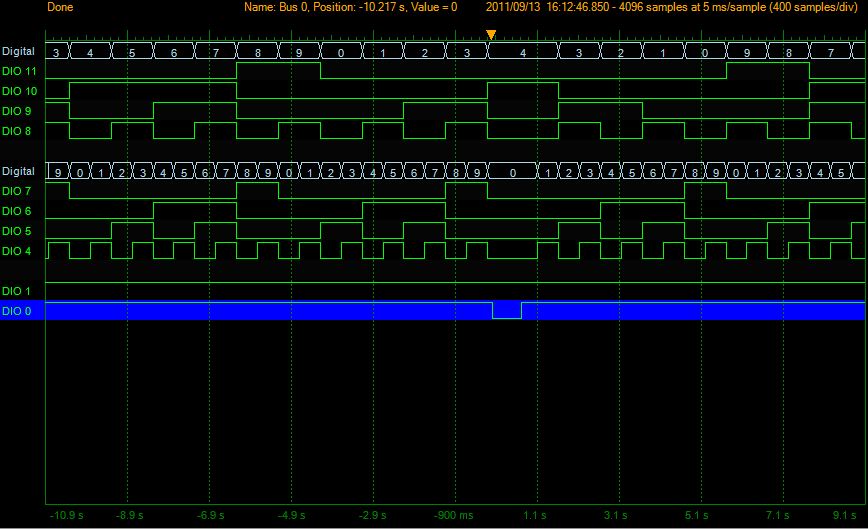


Figure 1: Logic Analyzer screen capture showing the count change direction with an XIRQ interrupt.



Figure 2: Logic Analyzer screen capture showing the count direction change due to an IRQ interrupt.

Fullcount

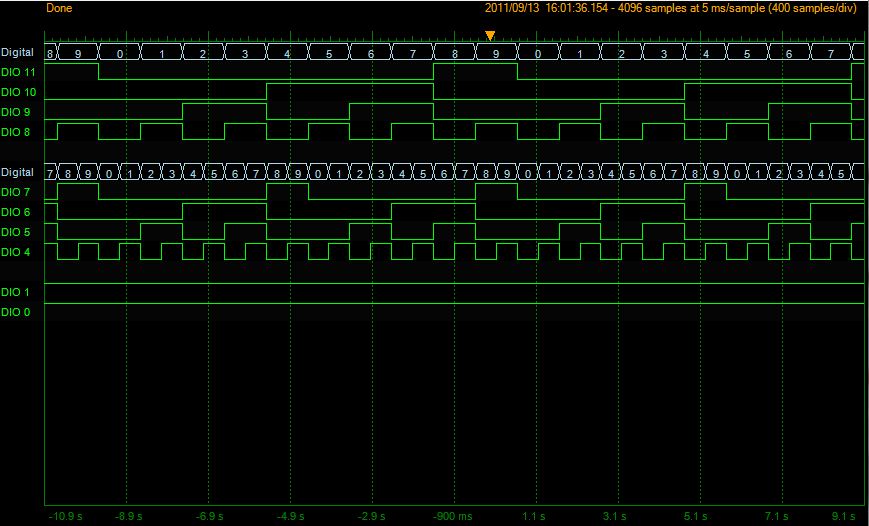


Figure 3: Logic Analyzer screen capture showing a full, uninterrupted count up.

Attachment 2a: Lab 5 Program

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

\* Brian Arnberg and Stephen Taylor

\* ELEC 3040 - Lab 5, main.c

\* Keypad programming

\* Always count up

\* Output Port AD

\* Port T 7-4 - keypad columns, (output)

\* Port T 3-0 - keypad rows, (input)

\* Port E 1 - IRQ

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

#include <hidef.h> /\* common defines and macros \*/

#include <MC9S12C32.h> /\* derivative information \*/

#pragma LINK\_INFO DERIVATIVE "MC9S12C32"

static unsigned char count1; //Count 1 - Constant up, output to AD

static unsigned char count2; //Count 2 - used to display keynumber for 10s

/\*the keynumber is in hex, hex 0-D makes sense

\* hex E is \* and hex F is # \*/

static unsigned char keynumber;

/\* Define the IRQ service routine \*/

interrupt void IRQ\_ISR(void) {

/\*This is really ugly, but it should work\*/

keynumber = PTAD;

PTT = 0xF0;

if (PTT\_PTT7 == 0) {

}

else {

PTT = 0xE0; // PTT\_PTT4 driven low, C1 low, Celse high

PTT = 0xE0; // PTT\_PTT4 driven low, C1 low, Celse high

PTT = 0xE0; // PTT\_PTT4 driven low, C1 low, Celse high

PTT = 0xE0; // PTT\_PTT4 driven low, C1 low, Celse high

if (PTT\_PTT0 == 0) {//row 1

PTAD = 0x01;//key 1

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT1 == 0){//row 2

PTAD = 0x04;//key 4

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT2 == 0){//row 3

PTAD = 0x07;//key 7

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT3 == 0){//row 4

PTAD = 0x0E;//key \*

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

} else {

PTT = 0xD0; // PTT\_PTT5 driven low, C2 low, Celse high

PTT = 0xD0; // PTT\_PTT5 driven low, C2 low, Celse high

PTT = 0xD0; // PTT\_PTT5 driven low, C2 low, Celse high

PTT = 0xD0; // PTT\_PTT5 driven low, C2 low, Celse high

if (PTT\_PTT0 == 0) {//row 1

PTAD = 0x02;//key 2

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT1 == 0){//row 2

PTAD = 0x05;//key 5

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT2 == 0){//row 3

PTAD = 0x08;//key 8

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT3 == 0){//row 4

PTAD = 0x00;//key 0

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

} else {

PTT = 0xB0; // PTT\_PTT6 driven low, C3 low, Celse high

PTT = 0xB0; // PTT\_PTT6 driven low, C3 low, Celse high

PTT = 0xB0; // PTT\_PTT6 driven low, C3 low, Celse high

PTT = 0xB0; // PTT\_PTT6 driven low, C3 low, Celse high

if (PTT\_PTT0 == 0) {//row 1

PTAD = 0x03;//key 3

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT1 == 0){//row 2

PTAD = 0x06;//key 6

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT2 == 0){//row 3

PTAD = 0x09;//key 9

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT3 == 0){//row 4

PTAD = 0x0F;//key #

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

} else {

PTT = 0x70; // PTT\_PTT7 driven low, C4 low, Celse high

PTT = 0x70; // PTT\_PTT7 driven low, C4 low, Celse high

PTT = 0x70; // PTT\_PTT7 driven low, C4 low, Celse high

PTT = 0x70; // PTT\_PTT7 driven low, C4 low, Celse high

if (PTT\_PTT0 == 0) {//row 1

PTAD = 0x0A;//key A

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT1 == 0){//row 2

PTAD = 0x0B;//key B

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT2 == 0){//row 3

PTAD = 0x0C;//key C

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

else if (PTT\_PTT3 == 0){//row 4

PTAD = 0x0D;//key D

count2 = 10;

PTT = 0x0F; //reset PTT columns to 0

\_\_asm rti //use assembly code to exit the interupt

}

}

}

}

}

}

/\* Delay function - In lab tests show i<17 ~ .5s \*/

void delay (void) {

int i,j;

for (i=0; i<17; i++) { //outer loop, "i<50" seems too slow

for (j=0; j<20000; j++) { //inner loop

}

}

}

/\* Function for the first count \*/

void count\_1 (void) {

if ( count1 == 9 ) { count1 = 0; }

else { count1 +=1; }

//PTAD = count1; // output count1 to Port AD

}

void main (void) {

DDRE = 0; //set Port E to read

DDRT = 0xF0; //set PortT bits 7-4 output, 3-0 input

DDRAD = 0xFF; //set PortAD to output

count1 = 0; //initialize count1 to 0 to start count up

count2 = 0; //initialize count2 to 0 to prevent countdown

PERT = 1; //Enable Port T's pull device

PPST = 0xF0; // Port T: 7-4 pull low; 3-0 pull up

PTT = 0x0F;

INTCR\_IRQEN = 1; /\*enable IRQ# interrupts \*/

INTCR\_IRQE = 1; /\*IRQ# interrupts edge-triggered \*/

EnableInterrupts; /\*clear I mask to enable interrupts \*/

//\_\_asm ANDCC #0xBF /\*clear X mast to enable XIRQ# \*/

while (1){

delay();//delay for 0.5s

delay();//delay for 0.5s

count\_1();//increment count 1

/\* if the countdown is greater than 0, decrement by one

\* and do nothing;

\* else output the "count up" to AD \*/

if ( count2 > 0 ) {

count2 -= 1; //decrement countdown

PTT = 0x0F;

}

else {

PTAD = count1; //display the current count to AD

PTT = 0x0F;

}

} /\* repeat forever \*/

}

Attachment 2b: Lab 5 Screenshot



Figure 4: Logic Analyzer Screen Capture showing the count changing to a key number (8).