Khalifa University of Science, Technology and Research

Electronic Engineering Department

ELCE333Microprocessor Systems laboratory

Laboratory Experiment 6

HCS12 INTERRUPTS

Lab Partners

Leena ElNeel	100037083
Muna Darweesh	100035522
Shamsah AlNabooda	100036984

Date Experiment Performed: 4/3/2015
Date Lab Report Submitted: 18/3/2015

Lab Instructors:

Mahmoud Khonji

Mohammed Ali Saif Al Zaabi

Spring 2015

Table of Contents

List of Figures and Tables

Figures:

Summary

This experiment introduces the students to the concept of interrupt and their usage in embedded system. The first task is Up Counter using C Language. The second task is Interrupt Based Counter. Thirdly, the task is about Interrupt Based Buzzer. And finally, the last task is about Interrupt Based LED Flashing. Also, this experiment required us to answer two assignment questions, one is practical regarding using the both LCD and LED to show an increment/decrement of a counter. The second was a theoretical question about interrupt latency.

1. Introduction

An interrupt is a special event that requires the CPU to stop normal program execution and perform some service related to the event. An example is a timer time-out. Interrupts coordinating I/O activities and preventing CPU from being tied up, they provide a graceful way to exit from errors and they remind the CPU to perform routine tasks. Interrupts that can be ignored by the CPU are called maskable interrupts. Interrupts that can't be ignored by the CPU are called non-maskable interrupts. Interrupt priority allows multiple pending interrupt requests and resolves the order of service for multiple pending interrupts. CPU executes a program called the interrupt service routine and works in the following way:

1. Save the program counter value in the stack
2. Save the CPU status
3. Identify the cause of interrupt
4. Resolve the starting address of the corresponding interrupt service routine
5. Execute the interrupt service routine
6. Restore the CPU status and the program counter from the stack
7. Restart the interrupted program

1.1 Aim

To introduce the students to the concept of interrupt and their usage in embedded system**.**

1.2 Objectives

On completion of this experiment the student should be able to:

1- Understand interrupts and their use in embedded system.
2- Write a program that handles external interrupts.
3- Develop simple programs for an embedded system.
4- Use the CodeWarrior IDE for the development of HCS12 microcontroller C programs.
5- To compile, download and debug/test a C program using CodeWarrior C compiler and Dragon12 Plus Trainer board.

2. Design and Results

Task 1: Up counter using C language

In the first task of this lab we were asked to write a code for an 8 binary counter and display the count on the LEDs. The counter should counts from $00 to $0F and repeats incrementally at ¼ Hz; hence the delay will be set at 4000 ms. The following code is for this task:

```
#include <hidef.h>
#include "derivative.h"
#include "lcd.h" // to be able to use delay function
void main(void) {
// initializing LEDs to display output
DDRB= 0xFF;
DDRJ=0xFF;
DDRP=0x0F;
PTP= 0x0F;
PTJ= 0x00;
PORTB= 1;
          EnableInterrupts;
 for(;;) {
          PORTB ++; // to increment counter
          delay (4000);// increment at ¼ Hz
           if (PORTB ==0x0F)
          // when reach $0F restart counter and redisplay
            {
                    PORTB =0x00;
            }
          } //end of loop
}//end of main
```

Task 2: Interrupt Based Counter

In this task, it is required to write an interrupt service routine using the IRQ switch. This program should count and keep incrementing as long as there are no interrupts. The program should run until 15 and then start from zero. When the IRQ switch is pressed, the current contents of switches SW4-SW1 must be used to initialize the starting count of the counter. For example, if the switches are set to $04, the counter should go from $05 $06 up to $0F and repeat $05 $06 . . . $0F. When the IRQ switch is pressed the counter must be immediately set to the new initial and keeps counting as long as the IRQ interrupt is pressed. As an initialization, the Interrupt Control Register (INTCR) to $C0 and set PTB and PTJ and PTP as output port with value FF. For the interrupt setting, for the interrupt port we assign the PTH as input. The 7-segment display is disabling by assigning PTP to $0F. Then the loop runs until $0F is reached the counter starts again. For the interrupt, the Subroutine is defined with IRQ vector number or the port notification of 6, with this we can set the counter to the number that at which the interrupt occurs and keeps repeating as the long as the interrupt is present, this done by changing the jumper in IRQ pins.

```
#include <hidef.h>
#include "derivative.h"
void delay(unsigned int ms){
 int i,j;
for(i=0;i<ms;i++)for(j=0;j<4000;j++)
;
}
int num= 0;
void main(void)
{
 INTCR = 0xC0;
 DDRB = 0xFF;
 DDRJ = 0xFF;
 DDRP = 0xFF;
 PTJ = 0x0;
 PTP=0x0F;
PORTB=0x00;
 DDRH = 0x00;
  __asm CLI;
```

```
for(;;)
  {
        PORTB++;
        delay (1000);
        if (PORTB == 0x0F)
        {
          PORTB = num;
          delay (1000);
        }
  } //end of loop
}//end of main

#pragma CODE_SEG NON_BANKED
// allow CW to access interrupt vector table
// interrupt service routine
interrupt 6 void mySubroutine (void) {
  PORTB =  PTH&0x0F;
  num = PORTB;
}//end of void
```

Task 3: Interrupt Based Buzzer

```
#include <hidef.h> #include "derivative.h"
void delay(unsigned int ms)
{ int i,j;
for(i=0;i<ms;i++)for(j=0;j<4000;j++);
}
void main(void)
{
  DDRB = 0xFF;  DDRJ = 0xFF;  DDRT = 0xFF;  DDRH = 0x00; PTJ = 0x0;
  PIEH = 0xFF; //enable PTH interrupt
  PPSH = 0x0; //Make it Falling Edge-Trig.
  __asm CLI; //Enable interrupts globally
for(;;)
  {
  PORTB = PORTB ^ 0b00000001; //Toggle PB0 while waiting for Interrupt
  delay (100);
  } //stay here until interrupt come.
} //end of main
//PTH Interrupt, Moving any of DIP Switches of PORTH from High-to-Low,
will //sound the buzzer for short period of time
#pragma CODE_SEG NON_BANKED
interrupt (((0x10000-Vporth)/2)-1) void PORTH_ISR(void)
// interrupt 25 void PORTH_ISR(void)
{
unsigned char x;
//Upon PTH Interrupt (any of DIP SWitch going from H-to-L) will sound
the //buzzer for a short period
for (x=0;x<1000/(PTH*PTH/5);x++)
{
PTT = PTT ^ 0b00100000; //toggle PT5 for Buzzer
//how long the Buzzer should sound. During the buzzer sound PB0 stops
//toggling. Why? 0
delay (PTH*PTH/5);
}
//clear PTH Interupt Flags for the next round. Writing HIGH will clear the
//Interrupt flags
PIFH = PIFH | 0xFF;}
```

In the last task we were given a buzzer code to modify it such that the each bit of PTH represents a different sound of a buzzer. The code starts off with an initializing of ports required for LEDs to flash, enabling PTH bits interrupt, selecting a falling edge trigger and enabling interrupts globally. Then, in the loop section, PORTB_BIT0 was toggled waiting for interrupts to come.

In order to have different buzzer sound with different frequencies we used the equation *PTH*PTH/5* to generate a sound with noticeable difference. However, to avoid having a huge delay between the changes and the sound the same function was used as a delay function.

Task 4: Interrupt Based LED Flashing

In the last task of this lab we were asked to write an interrupt based C# code that flashed all LEDs with a delay of 100 ms with additional functionality implemented for SW5 and SW2. In case SW5 is pressed a different pattern is displayed on the LEDs and in case SW2 is pressed the flashing stops.

The first step in writing the code was defining the delay and initializing LEDs for displaying the output. After that, PIEH was set t0 0x09 to enable the interrupt that can be caused by each of the PTH 8 bits. Also, the port polarity select register PPSH was set to 0 for falling edge trigger. Then, in the loop section, the value of PORTB was set to 0 and then changed to contain the value of the variable x (which is initially set to 1 to turn on all LEDs). In case SW5 is pressed the value of x is set to a random number each time by using the function *x=x+5* and in case SW2 was pressed *x=0* to turn off all LEDs. In addition to that, at the end of each case implementation the PTH interrupt flags are cleared for the next round by setting them to 1's using *PIFH = PIFH | 0xFF*.

```
#include <hidef.h>
#include "derivative.h"
int x=0xFF;
void delay(unsigned int ms)
 {
   unsigned int i; unsigned int j;
  for(i=0;i<ms;i++)
    for(j=0;j<4000;j++);
 }
void main(void)
{ // initializing LEDs
  DDRB = 0xFF;
  DDRJ = 0xFF;
  DDRP = 0xFF;
  DDRT = 0xFF;
  PTP  = 0x0F;
  PTJ  = 0x00;
  DDRH = 0x00;
  PIEH = 0x09;
//enable PTH interrupt
  PPSH = 0x00;
  EnableInterrupts;
  for(;;)
  {
  PORTB =0x00;
  delay(100);
  PORTB = x;
  delay(100);
  } //end of loop
}//end of main
```

```
#pragma CODE_SEG NON_BANKED
// allow CW to access interrupt vector table
interrupt 25 void PORTH_ISR(void)
 {
 if(PIFH_PIFH0==1)
  {
   x=x+5;
  PIFH = PIFH | 0xFF;
  }
 if(PIFH_PIFH3==1) {
  x=0;
  PIFH = PIFH | 0xFF;
 } } //end of function
```

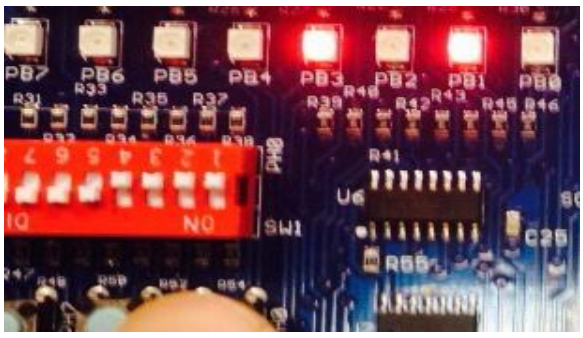

Figure 1: Task 4: case SW5 is pressed

3. Assignment Questions

1- Write an interrupt based binary counter that will display the reached count on the LCD if SW 3 is pressed and reverses the counter if SW4 is pressed (keep your interrupt routine optimized).

```
#include <hidef.h>
#include "derivative.h"
#include "lcd.h"
void main(void)
{
// initializing LEDs for display
  DDRB = 0xFF;
  DDRJ = 0xFF;
  DDRP = 0xFF;
  DDRT = 0xFF;
  PTP  = 0x0F;
  PTJ  = 0x00;
 DDRH = 0x00;
  PIEH = 0xFF;   // Enable PTH
interrupt
  PPSH = 0x00;   // Make it Edge-Trig.
  __asm CLI;

  for(;;)
  {
  PORTB++;
  delay (100*30);
  if ( PORTB == 0x0F)
  PORTB =  0x00 ;

  }
}
```

```
#pragma CODE_SEG NON_BANKED  //
to access the interrupt vector table
void PORTH_ISR(void)
interrupt 25 void PORTH_ISR(void)
 {
  LCD_Init();
   if(PIFH_PIFH2==1)
    {
     delay (10);
     LCDWriteLine(1, "the number= ");
     LCDWriteFloat(PORTB);
     delay (10);
     PIFH=PIFH_PIFH2_MASK;
    }
  if(PIFH_PIFH1==1)
  {
    PORTB--;
    delay (10);
    delay (10);
       LCDWriteLine(1, "the number=
");
       LCDWriteFloat(PORTB);
       delay (1000);
    if ( PORTB == 0x00)
    PORTB =  0x0F ;
   }
    PIFH=PIFH_PIFH1_MASK;
  }     //end of fucntion
```

2- Define interrupt latency and give at least two components of interrupt latency in HCS12.

Interrupt latency is the time that elapses from when an interrupt is generated to when the source of the interrupt is serviced. For many operating systems, devices are serviced as soon as the device's interrupt handler is executed. Interrupt latency is affected by interrupt controllers, interrupt masking and handling methods.

Components of interrupt latency in HCS12:

- Time to complete the current instruction
- Time to push the registers on the stack
- Time to fetch the ISR addres

4. Conclusion

In this laboratory experiment, the four tasks were accomplished in order to understand the functionality and usage of interrupts.

In the first task, a C code for counter was written and it is designed to be implemented and show the counting results on the LEDs of the DRAGON 12 taking into account that each number is incremented with ¼ Hz, so the delay will be 4000. In the second task, the counter was modified to be an interrupt based counter, which keep counting from the number that the interrupt happen at. The interrupt is IRQ which is makeable one and defined by the subroutine and with the port notification number. In task 3, we were able to use the ISR to sound the buzzer for different frequency for a short duration when moving the DIP switches from high to low. In task four, we flash the LEDs depending on the switch pressed. If switch 5 is pressed then the results in

LEDs is changed, but if switch 2 is pressed no flashing LEDs. This task allows us to understand the use of PTH as edge-triggered interrupt.