Khalifa University of Science, Technology and Research
Electronic Engineering Department

Microprocessor Systems laboratory
ELCE333

Laboratory Experiment 6

HCS12 INTERRUPTS

Lab Partners
Dana Bazazeh, 100038654
Moza Al Ali 100038604

Date Experiment Performed: 04/03/2015
Date Lab Report Submitted:18/03/2015

Lab Instructor: Dr.Mohamed Al Zaabi/ Dr.MahmoudKhonji

SPRING 2015

Table of Contents

List of Figures and Tables

Summary

In this lab report students were introduced to techniques of writing a program that handles Interrupts which is the main topic of this experiment. Mainly this session contains three tasks that are analyzed in details in the report. The first task was about writing a binary counter with 8-bits that counts from $00 to $0F and repeats the procedure. Apart from that, the second task focuses on how to write an interrupt service routine using the IRQ switch. Moreover, the third task introduces the technique to sound the buzzer (PT5) for a short period of time with different frequencies controlled by PTH. The third part of the report analyses the results of each task. Finally, there is a conclusion to discuss the main achievements and the recommendations in this laboratory session.

1. Introduction

In systems programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.[1] There are two types of interrupts: hardware interrupts and software interrupts.

Hardware interrupts are used by devices to communicate that they require attention from the operating system. Internally, hardware interrupts are implemented using electronic alerting signals that are sent to the processor from an external device, which is either a part of the computer itself, such as a disk controller, or an external peripheral. For example, pressing a key on the keyboard or moving the mouse triggers hardware interrupts that cause the processor to read the keystroke or mouse position. Unlike the software type (described below), hardware interrupts are asynchronous and can occur in the middle of instruction execution, requiring additional care in programming. The act of initiating a hardware interrupt is referred to as an interrupt request (IRQ).

A software interrupt is caused either by an exceptional condition in the processor itself, or a special instruction in the instruction set which causes an interrupt when it is executed. The former is often called a trap or exception and is used for errors or events occurring during program execution that is exceptional enough that they cannot be handled within the program itself. For example, if the processor's arithmetic logic unit is commanded to divide a number by zero, this impossible demand will cause a divide-by-zero exception, perhaps causing the computer to abandon the calculation or display an error message. Software interrupt instructions function similarly to subroutine calls and are used for a variety of purposes, such as to request services from low-level system software such as device drivers. For example, computers often use software interrupt instructions to communicate with the disk controller to request data be read or written to the disk.

Each interrupt has its own interrupt handler. The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts. Interrupts are a commonly used technique for computer multitasking, especially in real-time computing. Such a system is said to be interrupt-driven.

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

This task asks us to write a program for an 8-bit binary counter that counts from $00 to $0F and repeats. It should increment at about ¼ Hz (4secs delay) and the count should be displayed on LED7-LED0. The code is shown below:

The code of this task is shown as follows:

```
#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
void delay(byte ms)
{   inti,j;
for(i=0;i<(ms*100);i++)for(j=0;j<14;j++) asm("nop\n");
}

void main(void)
{
DDRB = 0xFF; //initialize PORTB as output
DDRJ = 0xFF;
PTJ = 0x0;
PORTB = 0; //initialize counter

for(;;)
{
  PORTB++;
delay (4000); //delay of 4 secs
if ( PORTB == 0x0F)
  PORTB = 0x00  ;
}
}
```

Figure 1: C program for an Up counter

As usual, we begin by initializing the data direction register of PORTB as output (0xFF). Next, we initialize the counter(PORTB) to 0. Inside the infinite for loop, we increment PORTB and then delay 4 secs. We also check whether the counter reached the value 0x0F, and if it did, then we reset it to 0.

Task 2: Interrupt Based Counter

Task 2 asks us to write an interrupt service routine that depends on the IRQ switch. We have to write a program for a counter like in task1, but now the initial value of the counter is taken from the contents of switches SW4-SW1 and this happen when the IRQ switch is pressed, causing an interrupt. The counter must be set to the new value immediately when the interrupt occuts. The full program code is shown below:

```
#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
void delay(unsigned intms)
{ inti,j;
for(i=0;i<ms;i++)for(j=0;j<4000;j++);
}
int x;
void main(void)
{
DDRB = 0xFF; //output
DDRP = 0xFF;
DDRJ = 0xFF;
PTP = 0x0F;
PTJ = 0x0;

DDRH = 0x00; //input
INTCR = 0xC0; //
PORTB = 0;
__asm CLI;
for(;;)
{
delay (1000);
  PORTB++;
if ( PORTB == 0x0F)
    PORTB =  x ;
}
}
#pragma CODE_SEG NON_BANKED
interrupt 6
void PORTH_ISR(void){
  x=PTH & 0x0F;
  PORTB=x;
}
```

Figure 2: C program for interrupt based counter

Again, we begin by initializing PORTB as output and PORTH as input. In addition, we must set the Interrupt Control Register (INTCR) to 0XC0. This means setting bits 7 and 6, where bit 7 is used to set IRQ for falling edge triggered interrupt and bit 6 is the IRQEN bit used to enable IRQ specifically, Next, in order to globally enable interrupts the assembly command clear global interrupt flag has to be called "__asm CLI;". We initialize an integer x to hold the value of the switches contents. As before, we increment the counter in PORTB, and when it reaches 0x0F, we initialize it to x. In the interrupt routine, After looking at the mc9s12dg256.h file, we obtained the interrupt vector number of IRQ, which is 6 and then we defined the interrupt type using "interrupt 6". Then , inside the interrupt function for PORTH ISR interrupt, we set x to the value of SW4-SW1 using bit masking PTH & 0X0F to only store the value of the 4 least significant bits of PORTH. Also, to immediately start the counter using this new value, we set PORTB = x.

Finally, we loaded the program on the dragonboard and tested the IRQ interrupt and it worked as needed.

Task 3: Interrupt Based Buzzer

This task asks us to modify the ISR interrupt routine shown in the introduction in order to sound the buzzerat different frequency for each bit of the PTH. Therefore, for each PTH bit, we should have a unique sound.

Using the code provided in the labscript, we can see that now we are dealing with a PORTH ISR interrupt. The ISR interrupt vector number is 25. Initially, in the interrupt function definition, bit 5 of PTT, which is the buzzer, is toggled a 100 times using a for loop, each with a fixed delay of 10ms. After that, the interrupt is cleared for the next round by writing 0XFF to the port interrupt flag register PIFH. Now, in order to get a unique sound for each switch, we can manipulate the value of the delay. Using delay(PTH) instead of delay(10) the delay time will depend on the value of PTH and will be different for every switch.The complete modified program is shown below:

```
#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
void delay(unsigned intms)
{ inti,j;
for(i=0;i<ms;i++)for(j=0;j<4000;j++);
}
void main(void)
{
DDRB = 0xFF; //PTB as output for LEDs
DDRJ = 0xFF; //PTJ as output
DDRT = 0xFF; //PORTT as output
PTJ = 0x0; //Let PTB show data. Needed by Dragon12+ board
//PORTH interrupt setup
DDRH = 0x00; //PTH as input
PIEH = 0xFF; //enable PTH interrupt
PPSH = 0x0; //Make it Falling Edge-Trig.
__asm CLI; //Enable interrupts globally
for(;;)
{ //do something
PORTB = PORTB ^ 0b00000001; //Toggle PB0 while waiting for Interrupt
delay (100);
} //stay here until interrupt come.
}
//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<100;x++)
{
PTT = PTT ^ 0b00100000; //toggle PT5 for Buzzer
//how long the Buzzer should sound. During the buzzer sound PB0 stops //toggling. Why?
delay (PTH);
}
//clear PTH Interupt Flags for the next round. Writing HIGH will clear the //Interrupt flags
PIFH = PIFH | 0xFF;
}
```

Figure 3: C program for interrupt based buzzer

Finally, we loaded the program on the dragonboard and verified that it gave us different sounds when flipping different switches.

Task-4: Interrupt Based LED Flashing

This task asks us to write an interrupt based program that should flash all of the LEDs with a delay of 100 ms.The flashing pattern should change each time SW5 is pressed and the flashing should stop each time SW2 is pressed.

The complete code is shown below:

```
#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
int x=0;
void delay(unsigned intms)
  {
unsignedinti; unsigned int j;
for(i=0;i<ms;i++)
for(j=0;j<4000;j++);
  }
void main(void)
{
  DDRB = 0xFF;    // Set PORTB as output since LEDs are connected to it
  DDRJ = 0xFF;    // Set PORTJ as output to control Dragon12+ LEDs
  DDRP = 0xFF;    // Set PORTP as output
  DDRT = 0xFF;    // Set PORTT as output
PTP  = 0x0F;    // Disable the 7-segment display
PTJ  = 0x00;    // Turn off PTJ1 to allow the LEDs to show data on PORTB
  //PORTH interrupt setup
  DDRH = 0x00;   // Set PORTH as input
  PIEH = 0x09;   // Enable PTH interrupt
  PPSH = 0x00;   // Make it Edge-Trig:falling edge
__asm CLI; //Enable interrupts globally
for(;;)
  {                           //do something
  PORTB =0x00;
delay(100);
  PORTB = x;
delay(100);
  } //stay here until interrupt come.
}
#pragma CODE_SEG NON_BANKED  // to access the interrupt vector table

interrupt 25 void PORTH_ISR(void)
  {
if(PIFH_PIFH0==1)   //Check if SW5 is pressed
   {
    x=x+5;
    PIFH = PIFH | 0xFF;
   }
if(PIFH_PIFH3==1) {   //Check if SW2 is pressed
   x=0;
  PIFH = PIFH | 0xFF;
 }
  }
```

Figure 4: C program for interrupt based LED flashing

We initialize PORTB as output and PTH as input. Since SW2 and SW5 are connected to PTH, we need to enable PTH interrupts by setting the port interrupt enable regitserr PIEH to 0x09 as well as set the port polarity select register PPSH to 0x00 to make the interrupt edge triggered ; falling edge. We initialize PORTB with a value of 0. In the interrupt function definition, we check whether SW5 is pressed (PIFH_PIFH0==1) in which we increment the current PORTB value by 5, to give a new flashing pattern. On the other hand, if SW2 is pressed (PIFH_PIFH3==1), then we simply clear PORTB. In both cases, we have to clear the interrupt flags to allow new interrupts to be accounted for using PIFH = PIFH | 0xFF.

We then loaded the program onto the dragonboard and verified that it functioned properly as needed.

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>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
#include "lcd.h"
void main(void)
{
  DDRB = 0xFF;    // Set PORTB as output since LEDs are connected to it
  DDRJ = 0xFF;    // Set PORTJ as output to control Dragon12+ LEDs
  DDRP = 0xFF;    // Set PORTP as output
  DDRT = 0xFF;    // Set PORTT as output
PTP  = 0x0F;    // Disable the 7-segment display
PTJ  = 0x00;    // Turn off PTJ1 to allow the LEDs to show data on PORTB

//PORTH interrupt setup

 DDRH = 0x00;   // Set PORTH as input
  PIEH = 0xFF;   // Enable PTH interrupt
  PPSH = 0x00;   // Make it Edge-Trig.
  __asm CLI;

for(;;)
  {
  PORTB++;
delay (500);
if ( PORTB == 0x0F)
  PORTB =  0x00 ;
  }  }
#pragma CODE_SEG NON_BANKED  // to access the interrupt vector table
interrupt 25 void PORTH_ISR(void)
  {
LCD_Init();
if(PIFH_PIFH2==1) //SW3 is pressed
   {
for(;;){PIFH = PIFH | 0xFF;
LCDWriteLine(1, "counter is  ");
LCDWriteFloat(PORTB);
   }


   }
if(PIFH_PIFH1==1) { //SW4 is pressed
PIFH = PIFH | 0xFF;
for(;;)
  {
   PORTB--;
delay (500);
if ( PORTB == 0x00)
  PORTB =  0x0F ;
  }
}  }
```

Figure 5: C program for interrupt based counter with LCD

Here, we need to use the LCD, therefore we include the lcd.h file. We begin by using the same code for the counter as in task 1 where we increment PORTB in an infinite loop. If a PTH button is pressed, the PORTH_ISR(void) function will be called. Inside this function, we first initialize the connection with the LCD and then based on whether SW3 (PIFH_PIFH2==1) or SW4 (PIFH_PIFH1==1) is pressed, we carry out the needed instruction..If SW3 is pressed, we write the value of PORTB to the LCD and then clear the flags. However, if SW4 is pressed, we clear the flags and then reverse the counter by decrementing PORTB in an infinite loop.

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

Interrupt latency is the time that elapses between the processor's initiation of interrupt servicing and the execution of the first byte of the specified service routine.

1- Time needed to save the processor status and CPU registers onto the stack.
2- Time needed to push current microcontroller status and registers onto the stack.
3- Time needed to fetch the ISR address.

4. Conclusions and Recommendations

At the end of this session students conclude, that the purpose and the objectives are successfully accomplished and well-understood. The three practical tasks of this laboratory session taught the students the ways that enables the interrupts in the microcontroller and how to use them. It also showed the students how to let the dip switches affect the criteria of the program as pre-programmed by adding the interrupt service routine to the software. Moreover, students could control the frequency of the sound generated by the buzzer via a function of an interrupt. All of the previously mentioned concepts are clear and easy to be implemented and reused in multiple purposes in microcontroller programs as a result of the well explanation of the laboratory instructor and the assistant. Not any problems or risks were faced in laboratory experiment.

After performing the required tasks of this experiment properly and that can be interpreted by the results that we got, we can say confidently that the theoretical knowledge that we own from the class courses matches the implementation results that we achieved in the laboratory.