Microprocessor Systems Laboratory (ELCE333)

Laboratory Experiment No.4

MARKING SHEET

Student Names and ID No's: Amal AlQasimi 100036830
Shaikha AlBesher 100037064

Laboratory Section: Wednesday **Experiment Date: March 4, 2015**

No.	Criteria	Description	Weight %	Mark	Comments
1	Pre-lab	A mark will be allocated to each student that reflects his preparations for the lab.	20		
2	Performance In the lab	A mark will be allocated to each student individually that reflects his performance in the lab	10		
3	Results and Analysis	Documentation and analysis of the results for each task performed in the lab	30		
4	Summary/ Conclusions	Conclusions for each task performed in the lab	10		
5	Assignment Questions	Answers to assignment questions	20		
6	Report Presentation	Overall presentation of the report including proper layout and clarity of figures, tables, and graphs. Correct use of English language.	10		
	Total		**100**		

KHALIFA UNIVERSITY

ELCE 333: Microprocessor Systems Laboratory

Lab Report- Experiment No.6

Experiment Title: HCS12 Interrupts

Completed by:

Amal Al Qasimi (100036830)

Shaikha Al Besher (100037064)

Lab instructors:

Mahmoud Khonji

Mohammed Al Zaabi

Spring 2015

Table of Contents

Summary

The following lab report will include the details of the six Microprocessor System's Laboratory lab session. It will first start off by stating the aims and objectives of the tasks assigned to us throughout the lab session. Following the aims and objectives, the report will discuss the purpose of each task separately with the results obtained upon the completion of the task. Based on the observations of the tasks completed, the analysis and interpretation section will include the explanation of why we obtained the results we did. Finally the report will end with a brief conclusion. The assignment questions given to us as part of the lab are answered after the lab report.

.

1. Introduction

Interrupts in embedded systems are much like subroutines, but they are generated by hardware events rather than software calls. When a hardware condition generates an interrupt, such as a Port H interrupt when SW2 is pushed, it tells the CPU to halt normal operation and jump to a specific location in memory called an Interrupt Service Routine (ISR). The ISR is just like a subroutine and contains code that is executed once the ISR is entered. Once the code in the ISR is completed, control is passed back to the main program.

The HCS12 has a specific way to locate interrupt service routines once an interrupt occurs. First, when a particular interrupt is activated, the program jumps to the Interrupt Vector Table, which is a region of memory that contains specific interrupt vectors. There is one interrupt vector for each type of interrupt that can occur, and each interrupt vector is located at a unique memory address in the table. Each interrupt vector contains the address of the start of the ISR that will run for that interrupt. The CPU will then load the address of the ISR in the Program Counter Register (PC), so that the program then starts executing at the beginning of the ISR. So, on an interrupt, the HCS12 first goes to one address (the interrupt vector address for that interrupt) which in turn contains another address which is that of the actual ISR itself - it's like a pointer to a pointer in C jargon. To be noted that the interrupt vector addresses are fixed by the hardware, whereas the ISR addresses in each interrupt vector are changeable values determined by the programmer.

Interrupt can be enabled to act on a low-to-high event (rising edge) or high-to-low event (falling edge) by setting the polarity in the relevant interrupt register. For instance, Port H can be set to interrupt polarity to "falling edge", meaning that the interrupt is activated as we go from high to low (logical 1 to 0). The switch buttons are tied to Port H, so a Port H pin tied to a switch will sit at +5V (high or logical 1) until we push it. Once we push it, this take the Port H pin to ground (low or logical 0) and thus activate our interrupt.

There are several types of interrupts. The ones we use most often are those triggered by peripherals, such as a timer overflow interrupt or the Port H interrupt. These are built in to the HCS12 chip. There are also external interrupts that can be used by devices external to the HCS12. The IRQ pin is an external interrupt that can be used for general purpose cases. For instance, if a device connected to the IRQ pin sends out a falling-edge logic pulse (i.e. takes the IRQ pin from high to low), it will set off an IRQ interrupt which will then cause the IRQ

interrupt service routine to be executed. The XIRQ pin is a non-maskable interrupt. It is called non-maskable because, in general, it can't be turned off or masked, so it is always armed (with a few minor exceptions). It is used to service interrupts that have extremely urgent service requests.

1.1 Aim:

To introduce the concept of interrupt and their purpose in embedded system.

1.2 Objectives:

1. To understand interrupts and their function in embedded system.
2. To write a program that deal with external interrupts.
3. To develop simple programs for an embedded system.
4. To 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+ Trainer board.

2. Design and Results

TASK-1: Up Counter using C Language

In this task we were asked to write an 8-bit binary counter that counts from $00 to $0F and repeats. The counter has to increment at approximately a quarter Hertz meaning the 4000 ms delay. The count has to be displayed on LED7-LED0.

The code is as following:

```
#include <hidef.h>
#include "derivative.h"
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
 DDRP = 0xFF; //PORTT as output
 PTJ = 0x0;
 PTP=0x0F;
 PORTB=0x00;

for(;;){
PORTB++;
delay (1000);


if (PORTB == 0x0F)

{
  PORTB = 0x00;
delay (1000);//to Display 0
}}


}
```

After running the code, the count appears on the LEDs and change according to the counter. As seen, the delay had been reduced to 1000 for a better display.

TASK-2: Interrupt Based Counter

This task requires modifying the previous task code to represents interrupt service using IRQ switch. 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 $06, the counter should go from $06 $07 . . . to $0F and repeat $06 $07 . . . $0F.

The code is implemented as follows:

```
#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++);
}
intnum= 0;
void main(void)
{
 INTCR = 0xC0;
 DDRB = 0xFF; //PTB as output for LEDs
 DDRJ = 0xFF; //PTJ as output
 DDRP = 0xFF; //PORTT as output
 PTJ = 0x0;   //Let PTB show data. Needed by Dragon12+ board
//PORTH interrupt setup
 PTP=0x0F;  //  To disable the 7-segment
 PORTB=0x00;
 DDRH = 0x00;     //  PTH input
  __asm CLI;     // enable interrupts globally

for(;;)
{
PORTB++;
delay (1500);
if (PORTB == 0x0F)
{
  PORTB = num;
delay (1000);//to Display 0
}}}
#pragma CODE_SEG NON_BANKED
interrupt 6 void mySubroutine (void) {
   PORTB =  PTH&0x0F;
num = PORTB;

}
```

Running the code gave correct results, and the counter started the counter from the value of the switches, and started counting from the begging according to the entered value.

Task-3: Interrupt Based Buzzer

In this task, we had to modify the ISR code provided in the introduction to sound the buzzer (PT5) for short period of time at different frequency for each bit of PTH. Each bit of PTH can represent a door (or window) and as any door gets opened, the buzzer will make a unique sound. The code will start by a flashing LED, port H switches will be used to introduce an interrupt which is buzzing sound. The aim is to have different buzzing sounds from each switch, accordingly the delay time interval had been changed, each switch with different delay value.

The code is adjusted asfollow:

```
#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 (200);
} //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;
```

```
}
```

The code worked correctly, the buzzing sound differs between switches particularly in the buzzing speed.

Task-4: Interrupt Based LED Flashing

The aim of this task is to write an interrupt program that flashes all LEDs with a delay of 100 ms and changes the flashing pattern displayed with a different pattern each time you press SW5 and stops the flashing each time you press SW2.

The code used for this program is displayed below:

```
#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
intnum= 0xFF;
void delay(unsigned intms)
{ unsignedinti,j;
for(i=0;i<ms;i++)for(j=0;j<4000;j++);
}
void main(void) {
 INTCR = 0xC0;
 DDRB = 0xFF; //PTB as output for LEDs
 DDRJ = 0xFF; //PTJ as output
 DDRP = 0xFF; //PORTT as output
 PTJ = 0x0;   //Let PTB show data. Needed by Dragon12+ board
//PORTH interrupt setup
 PTP=0x0F;  //  To disable the 7-segment
 PORTB=0x00;
 DDRH = 0x00;     //  PTH input
   __asm CLI;     // enable interrupts globally
for(;;)
{  PORTB = 0x00;
delay (100);
  PORTB = num;
delay (100);//to Display 0
}}
#pragma CODE_SEG NON_BANKED
interrupt 6 void mySubroutine (void) {
if (PIFH_PIFH0==1) {
num=num+100;
PIFH=PIFH |0xFF;
 }
if (PIFH_PIFH3==1) {
num=0;
PIFH=PIFH |0xFF;
 }
}
```

3. Analysis and Interpretation

This laboratory experiment consists of four tasks related to interrupts.

- Task 1: A C code was designed to form a counter that counts from $00 to $0F and repeats with a 4,000 ms delay. The result was displayed on the LEDs.
- Task 2: Creating an interrupt using IRQ switch, by modifying the code designed in the first task. The new code has to start the counter from the value of SW4-SW1 and continue whenever IRQ switch is pressed.
- Task 3: An ISR code to sound the buzzer (PT5) for short period of time at different frequency for each bit of PTH.
- Task 4: a program for flashing LEDs with 100 ms delay. If SW5 pressed, the pattern has to change. Furthermore, if SW2 is pressed the flashing has to stop.

4. Conclusion and Recommendations

Laboratory 6 experiment aims to provide us with understanding the interrupts, how to enable them and their various usage in embedded system. The requirements were completely achieved by the end of the lab session.

Interrupts in counters was experimented in different ways, like re-initializing counters using IRQ switch interrupt, and changing patterns or stopping the flashes on the LED using two switches. Another usage of interrupts was defining different sounds of the Buzzer (PT5) for different frequencies where interrupts occurs when moving the DIP switches from high to low. All of the previously mentioned tasks and concepts were clear and easy to implement using the DRAGON12. No problems were detected when using the software and the hardware. Also, no risks were faced in laboratory experiment.

5. 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"
intnum=0;
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
DDRP = 0xFF; //PTP as output
PTJ = 0x0F;
 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
LCD_Init();
for(;;)  { //do something
Num++;
PORTB = num; //Toggle PB0 while
waiting for Interrupt
delay (1000);  } //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 25 void PORTH_ISR(void)
{
if(PIFH_PIFH0==1)  {
for(;;){
LCDWriteInt(num);
PIFH= PIFH_PIFH0_MASK;
LCD_Init()  }  }
if(PIFH_PIFH2==1)
{ { for(;;){
Num--;
delay (1000);
PORTB=num;}
PIFH= PIFH_PIFH3_MASK; }}
```

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

- Interrupt latency definition:

  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 may be affected by microprocessor design, interrupt controllers, interrupt masking, and the operating system's (OS) interrupt 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 address.

References

1. CourseTextBook
2. http://www.evbplus.com/
3. MC9S12DP256User'sManual
4. https://learn.sparkfun.com/tutorials/serial-communication/all.pdf