Exercise 2: Keyboard Interrupt

EG-252 Group Design Exercise – Microcontroller Laboratory

Dr K. S. (Joseph) Kim Dr Chris P. Jobling

September 2014

Exercise 2: Keyboard Interrupt

For this exercise you are provided sample keyboard interrupt programs in both C and Assembly in the appendices; make sure that you have downloaded electronic versions of the program from the GitHub repository before the exercise. The program uses the interrupt generated by push buttons to switch on/off the LEDs on the MC9S08AW60 evaluation board.

You are to carry out the following three tasks with this exercise:

- Use the sample program to practice using interrupt mechanism to interface peripheral devices with the evaluation board.
- Answer the questions related to the CPU interrupt procedure (3 marks given on the final report).
- Adjust the sample program to use the onboard switches to control the display of your student number (7 marks, with 4 marks given on demonstration and 3 marks given on the final report).

The demonstration of your program on the board needs to be done no later than *Monday, 10 November 2014*.

You can view this document as a web page HTML, PDF or as a Word Document .docx

I. Experiment with the Sample Program

The sample program, "kbi_interrupt.c" and "kbi_interrupt.asm", flashes onboard LEDs upon the interrupt requests generated by the keyboard inputs, which are `SW3` and `SW4` used as keyboard inputs 6 and 5, respectively. The flowchart

of the program is shown in Figure 1. Enter it or copy the electronic file onto your CodeWarrior project, which is created targeting on HCS08 CPU family and MC9S08AW60 MCU.

```
Start
Port initialization:
PTFDD=$FF
PTDPE=$FF
LED_on=$0F
KBI initialization:
KBI1PE=$60;
Set KBI1SC bits 1 and 2
Unmask CPU interrupts:
CLI
Read LED_on
Mainloop
Main Program
Start
Clear KBI flag:
Set bit 2 of KBI1SC
Toggle bits of LED_on:
LED_on=LED_on XOR $FF
Output to light LEDs:
PTFD = LED_on
Interrupt service routine
RTI
```

Figure 1. Flowchart of keyboard interrupt program

After you create your own keyboard interrupt project, you can use the evaluation board to debug the sample keyboard interrupt program. Next we will introduce how to use evaluation board to configure peripheral device inputs and generate interrupts.

A. Interrupt Generation with the Evaluation Board

The evaluation board includes four pushbutton switches (`SW1-SW4`) which can provide momentary active low input for user applications. `SW3` and `SW4` are connected to MCU Port `PTD3` and `PTD2` respectively. If the pins of `PTD3` and `PTD2` are configured as KBI inputs, they will be read with "1" if the corresponding pushbutton is not pressed and "0" if pressed. Edge events and edge/level events can be easily generated by pressing `SW3` and `SW4`. The KBI inputs can detect the selected events as configured by the programmer and generate interrupt requests if keyboard interrupt for the inputs is enabled.

B. Experiment with the Interrupt Mechanism

Once we know how to generate keyboard interrupt events, we can use the sample program and evaluation board kits to experiment with interrupt mechanism.

According to the interrupt procedure, CPU will leave the main program and run the interrupt service routine (ISR) if interrupt for the keyboard module is enabled. You can use "Single step" or set breakpoint(s) to observe how the CPU responds to interrupt requests. After the CPU finishes the ISR, it returns to the main program. You can generate as many keyboard interrupts as you like by pushing down the switches.

You can also experiment with detection of rising edge and rising edge/high level events and correspondingly generating keyboard interrupt events by configuring the keyboard interrupt status and control register.

II. Questions

By doing experiments with either slightly modified sample C or assembly program, you are required to answer the following questions.

1) Upon the generation of keyboard interrupt requests, the ISR will be run by the CPU. Experiment with evaluation board, record and compare the content in the stack just before and after the CPU enter the ISR, just before and after the CPU leaves the ISR corresponding to a keyboard interrupt. You can get stack pointer value from register panel in real-time debugger window.

2) Explain why the stack content may change as you have observed.

III. Adjust the Sample Program to Display your Student Numbers

All the eight KBI inputs share the KBI source. In this task you are required to adjust the ISR shown in the Appendix, in order to determine which of the two pushbuttons `SW3` and `SW4` triggered a KBI interrupt and correspondingly display a student number of a member in your team.

Suppose the student numbers for your team are 43210 and 12345, respectively. On reset, if pushbutton `SW3` is pressed down, in your ISR you should light up the leftmost nonzero digit 4 in the first student number over LEDs (i.e., in a binary format using four LEDs). Then the CPU should leave the ISR and return to the main program. If `SW3` is released and then pressed down again, the next digit 3 in the first student should be displayed over LED in your ISR. With more pressing of SW3, the following digits in the first student number are displayed

sequentially. Note that after the rightmost digit 0 was displayed over LEDs upon the last pressing of SW3, the leftmost digit 4 will be displayed over LEDs upon new SW3 pressing. Similar operations are performed for the second student number 12345 if pushbutton SW4 is pressed.

You are required to design and debug the ISR program (using either Assembly or C language), and demonstrate the application to Dr Chris Jobling or Dr Tim Davies.

Appendix A

Sample Program in C

```
/* kbi_interrupt.c */

#include <hidef.h>    /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

typedef unsigned char muint8;
typedef unsigned short muint16;
typedef unsigned long muint32;

typedef char mint8;
typedef short mint16;
typedef long mint32;

/* to clear or set single bits in a byte variable */
#define b_SetBit(bit_ID, varID)   (varID |= (muint8)(1<<bit_ID))
#define b_ClearBit(bit_ID, varID) (varID &= ~(muint8)(1<<bit_ID))
#define b_XorBit(bit_ID, varID)   (varID ^= (muint8)(1<<bit_ID))

muint8 LED_onseq;

void main(void) {
  EnableInterrupts; /* enable interrupts */
  SOPT = 0x00;    /* disable COP */

  /* begin LED/switch test */

  PTDPE = 0xFF;   /* enable port D pullups for push button switch interrupt */

  /* Init_GPIO init code */
  PTFDD = 0xFF;   /* set port F as outputs for LED operation */
  LED_onseq = 0x0F; /* initialize LED_onseq */

```

```
  /* enable interrupt for keyboard input */
  b_ClearBit(1, KBI1SC);  /* KBI1SC: KBIE=0, disable KBI interrupt request */
  KBI1PE = 0x60;      /* KBI1PE: KBIPE7=1, enable KBI function for pins 5 and 6 only */
  b_ClearBit(0, KBI1SC);  /* KBI1SC: KBIMOD=0, select edge-only detection */

  /* in defaut only falling edge events to be detected */
  b_SetBit(2, KBI1SC);  /* KBI1SC: KBACK=1, to clear KBI flag */
  b_SetBit(1, KBI1SC);  /* KBI1SC: KBIE=1, enable KBI */

  for(;;) {
    __RESET_WATCHDOG(); /* feeds the dog */
  } /* loop forever */
    /* please make sure that you never leave main */
}

interrupt 22 void intKBI_SW(){
  KBI1SC_KBACK = 1; /*acknowledge interrupt*/
  PTFD = LED_onseq;
  LED_onseq ^= 0xFF;  /* toggle LED_onseq bits */
}
```

View on GitHub

Appendix B

sample Program in Assembly

```
;*******************************************************************
;*  kbi_interrupt.asm                                              *
;*                                                                 *
;*  MC9S08AW60 Evaluation board keyboard interrupt example         *
;*  - Switch SW3 onboard connected to Port D pin 3, KBI pin6;      *
;*  - Switch SW4 onboard connected to Port D pin 2, KBI pin5       *
;*                                                                 *
;*  Function:                                                      *
;*  on reset all LEDs will light on. If SW3 or SW4 pressed,        *
;*  an interrupt is generated, which set LEDs 0:3 to light on.     *
;*  More interrupts are genereated if SW3 or SW4 are pressed.      *
;*******************************************************************

; Include derivative-specific definitions
    INCLUDE 'derivative.inc'

FLASH EQU   $2000
RAM   EQU   $0070
```

```
19 WATCH EQU   $1802
20
21     ORG   RAM
22 LED_on  DS.B  1               ; Define a variable VAR_D with a size of 1 byte
23
24 ;Start program after reset
25     ORG   FLASH
26 START_UP
27     LDA   #$00
28     STA   WATCH     ; Turn off the watchdog timer
29
30 ;Init_GPIO init code
31     LDA     #$FF
32     STA     PTFDD
33     MOV     #$0F, LED_on    ; Initialize VAR_D, used to control the LEDs
34     LDA     #$FF
35     STA     PTDPE           ; Port D is enabled with pull-up
36     RSP           ; Reset stack pointer
37
38 ;Enable interrupt for Keyboard input
39     LDA     #$60
40     STA     KBI1PE           ; KBI1PE: KBIPE7=1, enable KBI function for pins 5 and 6 only
41     BSET    $02, KBI1SC      ; KBI1SC: KBACK=1, to clear KBI flag
42     BSET    $01, KBI1SC      ; KBI1SC: KBIE=1, enable KBI
43
44     CLI                      ; Enable interrupt
45
46 MAINLOOP
47     LDA     LED_on           ; Simple routine
48     BRA   MAINLOOP
49
50 ;Interrupt service routine for a keyboard interrupt generated upon the press of a pushbutto
51 ;with a falling edge (transition from high logic level "1" to low logic level "0")
52 LED_SWITCH
53     BSET    $02, KBI1SC     ; clear KBI flag
54     LDA     LED_on
55     EOR     #$FF            ; Toggle bits in VAR_D
56     STA     PTFD            ; Output to light LEDs (port F)
57     STA     LED_on          ; Store the new value to VAR_D
58
59     RTI
60
61 ;INT_VECTOR
62     ORG     $FFD2
63     DC.W    LED_SWITCH
64
```

```
65          ORG     $FFFE
66          DC.W    START_UP
```

View on GitHub