Laboratory Exercise 5

Using Interrupts with Assembly Code

The purpose of this exercise is to investigate the use of interrupts for the Nios II processor, using assembly-language code. To do this exercise you need to be familiar with the exceptions processing mechanisms for the Nios II processor, which are discussed in the tutorial *Nios II Introduction*, available in the Intel PSG University Program website. You should also read the information on exceptions and interrupts of the pre-built computer system documentation corresponding to which DE-series board you own.

Part I

Consider the main program shown in Figure 1. The main program needs to set up the stack pointer, configure the pushbutton KEYs port to generate interrupts, and then enable interrupts in the Nios II processor. You are to fill in the code that is not shown in the figure.

The function of your program is to show the numbers 0 to 3 on the *HEX*0 to *HEX*3 displays, respectively, when a corresponding pushbutton *KEY* is pressed. Since the main program simply "idles" in an endless loop, as shown in Figure 1, you have to control the displays by using an interrupt service routine for the pushbutton KEYs port.

Perform the following:

1. Create a new folder to hold your files for this part. Create a file, such as *part1.s*, and copy the assembly language code for the main program, given in Figure 1, into this file. Create a file *exception_handler.s*, and copy the code given in Figure 2 into it. Create any other source-code files that you need.

2. Figure 2 gives the code required for the Nios II reset and exceptions handlers. The exception handler calls a subroutine *KEY_ISR* to handle interrupts from the *KEY* pushbuttons. Create a file *key_isr.s* and write the code for the *KEY_ISR* interrupt service routine. Your code should show the digit 0 on the *HEX*0 display when KEY_0 is pressed, and then if KEY_0 is pressed again the display should be "blank". You should toggle the *HEX*0 display between 0 and "blank" in this manner each time KEY_0 is pressed. Similarly, toggle between "blank" and 1, 2, or 3 on the *HEX*1 to *HEX*3 displays each time KEY_1, KEY_2, or KEY_3 is pressed, respectively. If you are using a DE10-Lite, the toggling between 2/3 and "blank" is not possible as the DE10-Lite Computer has only two pushbutton KEYs .

3. Make a new Monitor Program project in the folder where you stored your source-code files. In the Monitor Program screen illustrated in Figure 3, make sure to choose Exceptions in the *Linker Section Presets* drop-down menu. Compile, download, and test your program.

```
          .text
          .global    _start
_start:
          /* set up the stack */
          · · · code not shown

          /* write to the pushbutton port interrupt mask register */
          · · · code not shown

          /* enable Nios II processor interrupts */
          · · · code not shown

IDLE:     br         IDLE                  /* main program simply idles */
          .end
```

Figure 1: Main program for Part 1.

```
/************************ RESET SECTION ******************************/
          .section  .reset, "ax"
          movia     r2, _start
          jmp       r2                      /* branch to main program */

/************************ EXCEPTIONS SECTION *************************/
          .section  .exceptions, "ax"
          .global   EXCEPTION_HANDLER
EXCEPTION_HANDLER:
          subi      sp, sp, 16              /* make room on the stack */
          stw       et, 0(sp)
          rdctl     et, ctl4
          beq       et, r0, SKIP_EA_DEC     /* interrupt is not external */
          subi      ea, ea, 4               /* must decrement ea by one instruction */
                                            /* for external interrupts, so that the */
                                            /* interrupted instruction will be re-run */
SKIP_EA_DEC:
          stw       ea, 4(sp)               /* save all used registers on the Stack */
          stw       ra, 8(sp)               /* needed if call inst is used */

          stw       r22, 12(sp)
          rdctl     et, ctl4
          bne       et, r0, CHECK_LEVEL_1   /* interrupt is an external interrupt */
```

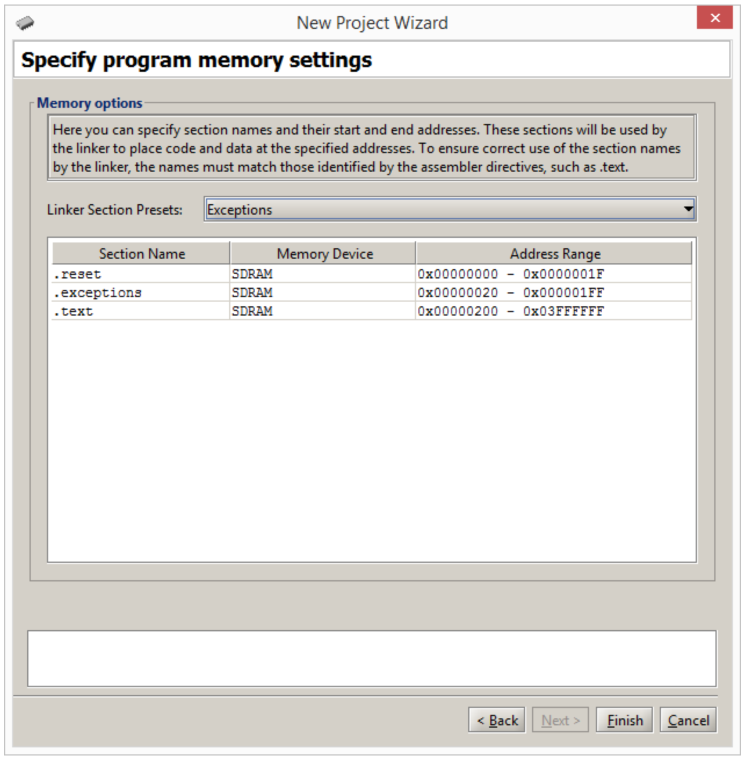
Figure 2: Exception handlers (Part a).

```
NOT_EI:     br      END_ISR             /* must be unimplemented instruction or TRAP */
                                        /* instruction; ignored in this code */
CHECK_LEVEL_1:                          /* pushbutton port is interrupt level 1 */
            andi    r22, et, 0b10
            beq     r22, r0, END_ISR    /* other interrupt levels are not handled in this code */
            call    KEY_ISR

END_ISR:    ldw     et, 0(sp)           /* restore all used register to previous values */
            ldw     ea, 4(sp)
            ldw     ra, 8(sp)           /* needed if call inst is used */
            ldw     r22, 12(sp)
            addi    sp, sp, 16
            eret

            .end
```

Figure 2. Exception handlers (Part *b*).


New Project Wizard

Specify program memory settings

Memory options

Here you can specify section names and their start and end addresses. These sections will be used by the linker to place code and data at the specified addresses. To ensure correct use of the section names by the linker, the names must match those identified by the assembler directives, such as .text.

Linker Section Presets: Exceptions

Section Name	Memory Device	Address Range
.reset	SDRAM	0x00000000 - 0x0000001F
.exceptions	SDRAM	0x00000020 - 0x000001FF
.text	SDRAM	0x00000200 - 0x03FFFFFF

Figure 3: Selecting the Exceptions linker section.

Part II

Consider the main program shown in Figure 4. The code is required to set up the Nios II stack pointer and to enable interrupts from two sources: the pushbutton KEYs and the Interval Timer. The main program calls the subroutines *CONFIG_TIMER* and *CONFIG_KEYS* to set up the two ports. You are to write each of these subroutines. Set up the Interval Timer to generate one interrupt every 0.25 seconds.

In Figure 4 the main program executes an endless loop writing the value of the global variable *COUNT* to the red lights LEDR. In the interrupt service routine for the Interval Timer you are to increment the variable *COUNT* by the value of the *RUN* global variable, which should be either 1 or 0. You are to toggle the value of the *RUN* global variable in the interrupt service routine for the pushbutton KEYs, each time a KEY is pressed. When $RUN = 0$, the main program will display a static count on the red lights, and when $RUN = 1$, the count shown on the red lights will increment every 0.25 seconds.

Make a new Monitor Program project for this part, and assemble, download, and test your code.

Part III

Modify your program from Part II so that you can vary the speed at which the counter displayed on the red lights is incremented. All of your changes for this part should be made in the interrupt service routine for the pushbutton KEYs. The main program and the rest of your code should not be changed.

Implement the following behavior. When KEY_0 is pressed, the value of the *RUN* variable should be toggled, as in Part I. Hence, pressing KEY_0 stops/runs the incrementing of the *COUNT* variable. When KEY_1 is pressed, the rate at which *COUNT* is incremented should be either increased or decreased depending on the value of SW_0. If SW_0 is 1, then the rate should be doubled, otherwise the rate should be halved. You should implement this feature by stopping the Interval Timer within the pushbutton KEYs interrupt service routine, modifying the load value used in the timer, and then restarting the timer.

Part IV

For this part you are to create a real-time clock that is shown on the seven-segment displays $HEX3-0$. Set up an interval timer to provide an interrupt every 1/100 of a second. Use this timer to increment a global variable called *TIME*. You should use the *TIME* variable as your real time clock. Use the format SS:DD, where *SS* are seconds and *DD* are hundredths of a second. When the clock reaches 59:99, it should wrap around to 00:00.

Make a new folder to hold your Monitor Program project for this part. Write the program for the real-time clock. To show the *TIME* variable in the real-time clock format SS:DD, you can use the same approach that was followed for Part 4 of Lab Exercise 4. In that previous exercise you used polled I/O with the Interval Timer, whereas now you are using interrupts. One possible way to structure your code is illustrated in Figure 5. The endless loop in this code writes the value of a variable named *HEX_code* to the $HEX3-0$ displays.

Using the scheme in Figure 5, the interrupt service routine for the second Interval Timer has to increment the *TIME* variable, and also update the *HEX_code* variable that is being written to the 7-segment displays by the main program.

Make a new Monitor Program project and test your program.

```
            .text
            .global     _start
_start:
            /* set up the stack */
            · · · code not shown

            call        CONFIG_TIMER
            call        CONFIG_KEYS

            /* enable Nios II processor interrupts */
            · · · code not shown

            movia       r8, /* insert red lights LEDR base address */
LOOP:       ldw         r9, COUNT(r0)           /* global variable */
            stw         r9, (r8)
            br          LOOP

/* Configure the interval timer to create interrupts at 0.25 second intervals */
CONFIG_TIMER:
            · · · code not shown
            ret

/* Configure the pushbutton KEYS to generate interrupts */
CONFIG_KEYS:
            · · · code not shown
            ret

/* Global variables */
            .global     COUNT
COUNT:      .word       0x0                     # used by timer
            .global     RUN                     # used by pushbutton KEYs
RUN:        .word       0x1                     # initial value to increment COUNT

            .end
```

Figure 4: Main program for Part II.

```
            .text
            .global   _start
_start:
            /* set up the stack */
            ··· code not shown

            call      CONFIG_TIMER

            /* enable Nios II processor interrupts */
            ··· code not shown

            movia     r9, /* insert HEX3-0 base address here */
LOOP:       ldw       r10, HEX_code(r0)      /* global variable */
            stw       r10, (r9)              /* show the time in format SS:DD */
            br        LOOP

/* Configure the interval timer to create interrupts at 1/100 second intervals */
CONFIG_TIMER:
            ··· code not shown
            ret

/* Global variables */
            .global   COUNT
TIME:       .word     0x0                    # used for real-time clock
            .global   HEX_code
HEX_code:   .word     0x0                    # used for 7-segment displays
```

Figure 5: Main program for Part IV.