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 at the Intel FPGA University Program website. You should also read the information on exceptions and interrupts of the DE0-Nano-SoC or DE-Nano Computer, depending on the board you are using.

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 turn on/off the green lights LED_1 and LED_0 when a corresponding pushbutton KEY_1 or KEY_0 is pressed. Since the main program simply "idles" in an endless loop, as shown in Figure 1, you have to control the LEDs by using an interrupt service routine for the pushbutton KEYs.

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 turn on *LED*1 display when KEY_1 is pressed, and then if KEY_1 is pressed again, *LED*1 should be turned off . You should toggle *LED*1 on and off each time KEY_1 is pressed. If you are using the DE0-Nano-SoC board, you should toggle *LED*0 when KEY_0 is pressed, the same as when *KEY*1 is pressed. Note that *KEY*0 is not available on the DE0-Nano board as it is hardwired to reset the processor.
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*).

Part II

Consider the main program shown in Figure 4. The code is required to set up the Nios II stack pointers and then enable interrupts. 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 green lights LED. In the interrupt service routine for 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 green lights, and when *RUN* = 1, the count shown on the green 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 green 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 II. Hence, pressing KEY_0 stops/runs the incrementing of the *COUNT* variable. Note that this can

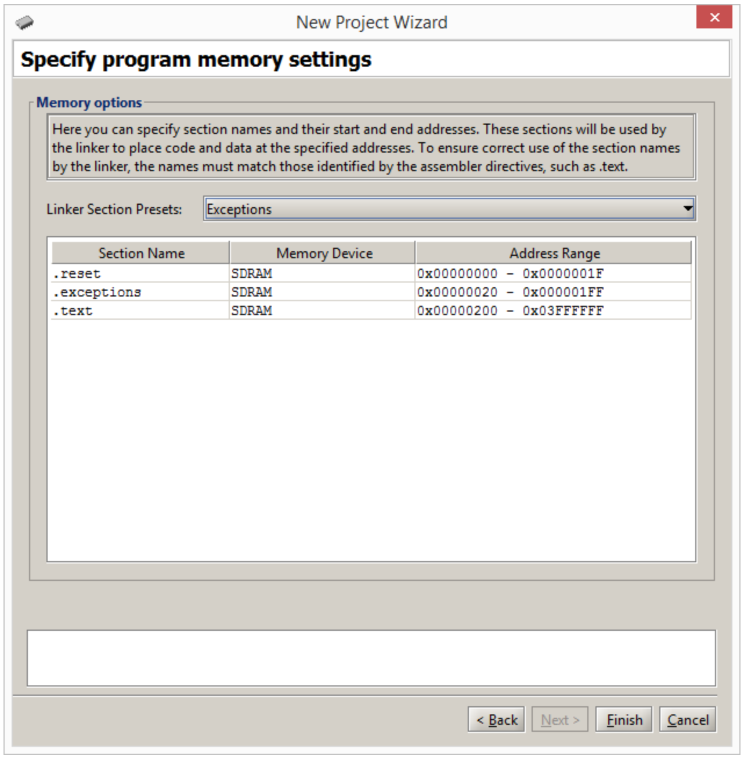

Figure 3: Selecting the Exceptions linker section.

only be done on the DE0-Nano-SoC board, and not on the DE0-Nano board. If you are using the DE0-Nano board, ignore this instruction. For either then DE0-Nano and De0-Nano-SoC boards, when SW_0 is high and KEY_1 is pressed, the rate at which *COUNT* is incremented should be doubled, and when SW_0 is low and KEY_1 is pressed 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 display the current count as a clock. Set up the timer to provide one interrupt each second. Use this timer to increment a global variable called *COUNT*. You should use the *COUNT* variable as a real-time clock that is shown on the Monitor Program Terminal window. Use the format MM:SS, where *MM* are minutes and *SS* are seconds. You should be able to stop/run the clock by pressing KEY_1. When the clock reaches 59:59, it should wrap around to 00:00.

Make a new folder to hold your Monitor Program project for this part. To show the *TIME* variable in the real-time clock format MM:SS, you can use the approach that was followed for Part IV of Lab Exercise 4. In Lab Exercise 4 you used polled I/O with the interval timer, whereas now you are using interrupts. The interrupt service routine for the timer should display the real-time clock on the Terminal window.

```
          .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 green lights LED 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.

Make a new Monitor Program project and test your program. In the screen shown in Figure 5, make sure to select JTAG_UART as the *Terminal device*. Without this setting no character output will appear on the Terminal window when your code writes to the JTAG UART.

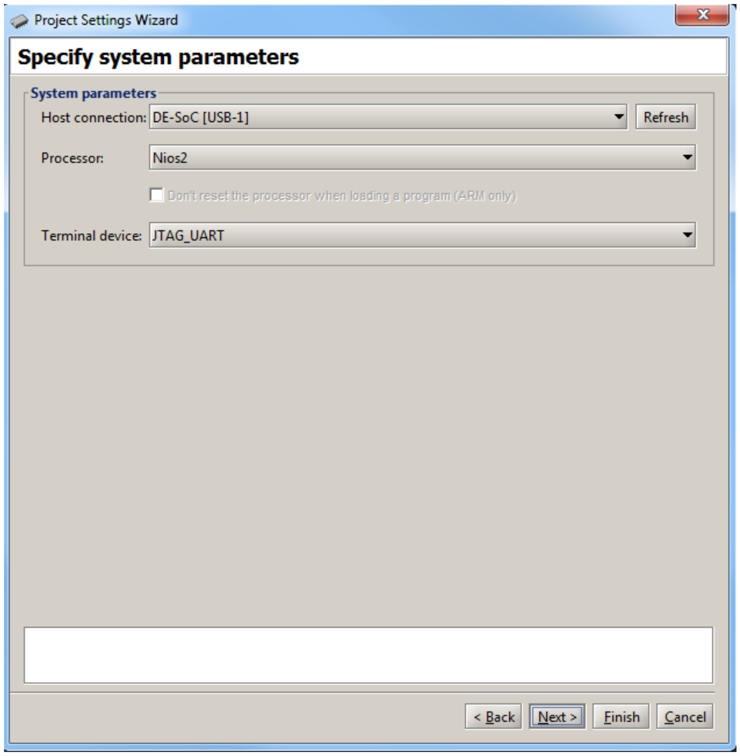

Figure 5: Specifying the *Terminal device*.

As a final exercise, add to your program the ability to slow down/speed up the timer, in the same way that you implemented this capability for the Interval Timer in Part III of this exercise. Observe the behavior of the Terminal window as it displays the real-time clock value at various timer rates. Discuss any anomolous behavior that you observe.

Copyright © 1991-2016 Intel Corporation. All rights reserved. Intel, The Programmable Solutions Company, the stylized Intel logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Intel Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Intel products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Intel warrants performance of its semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel Corporation. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.