Laboratory Exercise 6

Using Interrupts with Assembly Language Code

The purpose of this exercise is to investigate the use of interrupts for the ARM* processor, using assembly-language code. To do this exercise you need to be familiar with the exceptions processing mechanisms for the ARM processor, and with the operation of the ARM Generic Interrupt Controller (GIC). These concepts are discussed in the tutorials *Introduction to the ARM Processor*, and *Using the ARM Generic Interrupt Controller*. We assume that you are using the DE1-SoC Computer to implement the solutions to this exercise. It may be useful to read the parts of the documentation for those computer systems that pertain to the use of exceptions and interrupts.

Part I

Consider the main program shown in Figure 1. The code first sets the exceptions vector table for the ARM processor using a code section called *.vectors*. Then, in the *.text* section the main program needs to set up the stack pointers (for both interrupt mode and supervisor mode), initialize the generic interrupt controller (GIC), configure the KEY pushbuttons port to generate interrupts, and finally enable interrupts in the 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 KEY pushbutton 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 KEY pushbuttons port.

Perform the following:

1. Create a new folder to hold your solution for this part. Create a file, such as *part1.s*, and type the assembly language code for the main program into this file.
2. Create any other source code files you may want, and write the code for the *CONFIG_GIC* subroutine that initializes the GIC. Set up the GIC to send interrupts to the ARM processor from the KEY pushbuttons port.
3. The bottom part of Figure 1 gives the code required for the interrupt handler, *SERVICE_IRQ*. You have to 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.

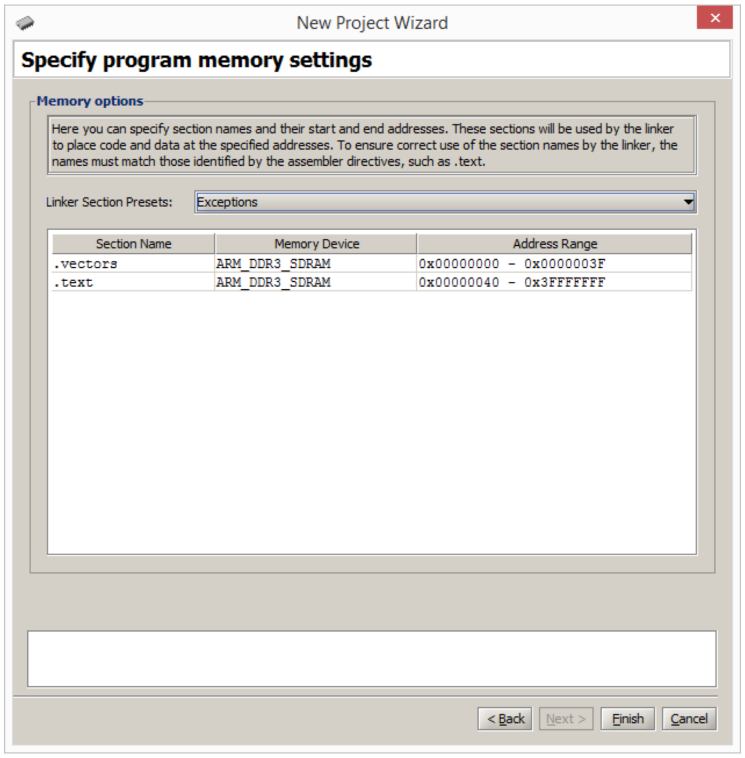
   Figure 2 provides code, using just simple loops, which can be used for the other ARM exception handlers.
4. Make a new Monitor Program project in the folder where you stored your source-code files. In the Monitor Program screen illustrated in Figure 3, make sure to choose Exceptions in the *Linker Section Presets* drop-down menu. Compile, download, and test your program.

```
                .section .vectors, "ax"
                B        _start                // reset vector
                B        SERVICE_UND           // undefined instruction vector
                B        SERVICE_SVC           // software interrupt vector
                B        SERVICE_ABT_INST      // aborted prefetch vector
                B        SERVICE_ABT_DATA      // aborted data vector
                .word    0                     // unused vector
                B        SERVICE_IRQ           // IRQ interrupt vector
                B        SERVICE_FIQ           // FIQ interrupt vector

                .text
                .global  _start

_start:
/* Set up stack pointers for IRQ and SVC processor modes */
                ... code not shown

                BL       CONFIG_GIC        // configure the ARM generic
                                           // interrupt controller
/* Configure the KEY pushbuttons port to generate interrupts */
                ... code not shown

/* Enable IRQ interrupts in the ARM processor */
                ... code not shown
IDLE:
                B        IDLE              // main program simply idles

/* Define the exception service routines */

SERVICE_IRQ:    PUSH     {R0-R7, LR}
                LDR      R4, =0xFFFEC100 // GIC CPU interface base address
                LDR      R5, [R4, #0x0C] // read the ICCIAR in the CPU
                                         // interface

FPGA_IRQ1_HANDLER:
                CMP      R5, #73           // check the interrupt ID

UNEXPECTED:     BNE      UNEXPECTED        // if not recognized, stop here
                BL       KEY_ISR

EXIT_IRQ:       STR      R5, [R4, #0x10] // write to the End of Interrupt
                                         // Register (ICCEOIR)
                POP      {R0-R7, LR}
                SUBS     PC, LR, #4        // return from exception
```

Figure 1: Main program and interrupt service routine.

```
/* Undefined instructions */
SERVICE_UND:
                    B    SERVICE_UND
/* Software interrupts */
SERVICE_SVC:
                    B    SERVICE_SVC
/* Aborted data reads */
SERVICE_ABT_DATA:
                    B    SERVICE_ABT_DATA
/* Aborted instruction fetch */
SERVICE_ABT_INST:
                    B    SERVICE_ABT_INST
SERVICE_FIQ:
                    B    SERVICE_FIQ

                    .end
```

Figure 2: Exception handlers.



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
.vectors	ARM_DDR3_SDRAM	0x00000000 – 0x0000003F
.text	ARM_DDR3_SDRAM	0x00000040 – 0x3FFFFFFF

Figure 3: Selecting the Exceptions linker section.

Part II

Consider the main program shown in Figure 4. The code has to set up the ARM stack pointers for interrupt and supervisor modes, and then enable interrupts. The subroutine *CONFIG_GIC* has to configure the GIC to send interrupts to the ARM processor from two sources: an Interval Timer and the KEY pushbuttons port. 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. In *CONFIG_TIMER* set up the *Interval Timer*, which is implemented in the FPGA and has the base address `0xFF202000`, 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 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 folder and 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 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 doubled, and when KEY_2 is pressed the rate should be halved. You should implement this feature by stopping the Interval Timer within the 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 add a third source of interrupts to your program, using the A9 Private Timer. Set up the 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 a real-time clock that is shown on the seven-segment displays $HEX3-0$. Use the format SS:DD, where *SS* are seconds and *DD* are hundredths of a second. You should be able to stop/run the clock by pressing pushbutton KEY_3. When the clock reaches 59:99, it should wrap around to 00:00.

Make a new folder to hold your solution for this part. Modify the main program from Part III to call a new subroutine, named *CONFIG_PRIV_TIMER*, which sets up the A9 Private Timer to generate the required interrupts. 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 private timer, whereas now you are using interrupts. One possible way to structure your code is illustrated in Figure 5. In this version of the code, the endless loop in the main program writes the value of a variable named *HEX_code* to the $HEX3-0$ displays.

```
                  .section .vectors, "ax"
                  ... code not shown

                  .text
                  .global  _start
_start:
/* Set up stack pointers for IRQ and SVC processor modes */
                  ... code not shown

                  BL       CONFIG_GIC       // configure the ARM generic
                                            // interrupt controller
                  BL       CONFIG_TIMER     // configure the Interval Timer
                  BL       CONFIG_KEYS      // configure the pushbutton
                                            // KEYs port

/* Enable IRQ interrupts in the ARM processor */
                  ... code not shown
                  LDR      R5, =0xFF200000  // LEDR base address
LOOP:
                  LDR      R3, COUNT        // global variable
                  STR      R3, [R5]         // write to the LEDR lights
                  B        LOOP

/* Configure the Interval Timer to create interrupts at 0.25 second intervals */
CONFIG_TIMER:
                  ... code not shown
                  BX       LR

/* Configure the pushbutton KEYS to generate interrupts */
CONFIG_KEYS:
                  ... code not shown
                  BX       LR

/* 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.

Using the scheme in Figure 5, the interrupt service routine for the private 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 stack pointers for IRQ and SVC processor modes */
                    ... code not shown
                    BL      CONFIG_GIC          // configure the ARM generic
                                                // interrupt controller
                    BL      CONFIG_PRIV_TIMER   // configure the private timer
                    BL      CONFIG_TIMER        // configure the Interval Timer
                    BL      CONFIG_KEYS         // configure the pushbutton
                                                // KEYs port
/* Enable IRQ interrupts in the ARM processor */
                    ... code not shown
                    LDR     R5, =0xFF200000     // LEDR base address
                    LDR     R6, =0xFF200020     // HEX3-0 base address
LOOP:
                    LDR     R4, COUNT           // global variable
                    STR     R4, [R5]            // light up the red lights
                    LDR     R4, HEX_code        // global variable
                    STR     R4, [R6]            // show the time in format
                                                // SS:DD
                    B       LOOP

/* Configure the MPCore private timer to create interrupts every 1/100 seconds */
CONFIG_PRIV_TIMER:
                    ... code not shown
                    BX      LR
/* Configure the Interval Timer to create interrupts at 0.25 second intervals */
CONFIG_TIMER:
                    ... code not shown
                    BX      LR
/* Configure the pushbutton KEYS to generate interrupts */
CONFIG_KEYS:
                    ... code not shown
                    BX      LR

/* Global variables */
                    .global COUNT
COUNT:              .word   0x0         // used by timer
                    .global RUN         // used by pushbutton KEYs
RUN:                .word   0x1         // initial value to increment COUNT
                    .global TIME
TIME:               .word   0x0         // used for real-time clock
                    .global HEX_code
HEX_code:           .word   0x0         // used for 7-segment displays

                    .end
```

Figure 5: Main program for Part IV.