Laboratory Exercise 6

Timer/Counter Circuits

Timer/counter circuits are used in many embedded systems. This exercise introduces the *Interval Timer* module that is included in Altera's DE-series Media Computer. It shows how the timer can be used to trigger an action at specified time intervals. It also illustrates how the module's counter can be used to estimate the performance of an arbitrary application program, in terms of the total number of clock cycles needed to execute the program.

The Interval Timer has an internal counter which is set to a specified starting value and then decremented in each clock cycle. When the counter reaches 0, a "timeout" event is said to have occurred. At this point the Interval Timer can raise an interrupt request and the counter can be reset to the starting value. The Interval Timer has a set of 16-bit registers that can be accessed as memory locations. These registers are shown in Figure 1. The address of the *Status* register is 0x10002000, which is the base address assigned to the Interval Timer. The *Control* register is at address 0x10002004. The starting value for the counter is specified in registers at addresses 0x10002008 (low-order 16 bits of the value) and 0x1000200C (high-order 16 bits of the value). As the counter value is decremented in each clock cycle, it is possible to capture a snapshot of the value at any time by performing a write to the address 0x10002010. This write operation causes the current 32-bit counter value to be stored into the two 16-bit registers at addresses 0x10002010 and 0x10002014. These registers can then be read to obtain the count value.

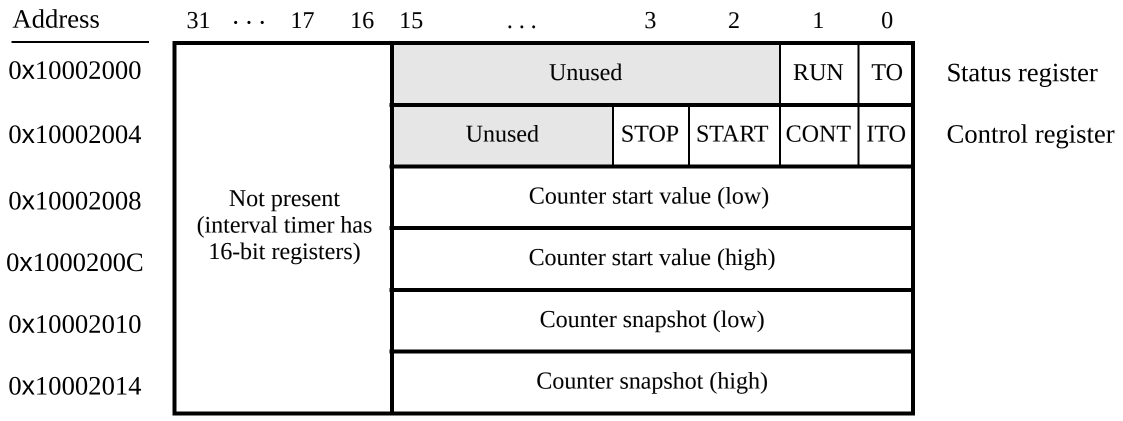

Figure 1. Registers in the Interval Timer

The bits in the *Status* register are used as follows:

- b_0 (TO) is the timeout bit. It is set to 1 when the internal counter in the Interval Timer reaches 0. It remains set until explicitly cleared by the processor writing a 0 to it, which must be done to clear an existing interrupt request.
- b_1 (RUN) is equal to 1 when the internal counter is running; otherwise, it is equal to 0. This bit is not changed by a write operation to the *Status* register.

The bits in the *Control* register are used as follows:

- b_0 (ITO) enables the Interval Timer interrupts when set to 1.
- b_1 (CONT) determines how the internal counter behaves when it reaches 0. If CONT = 1, the counter runs continuously by reloading the specified starting count value; otherwise, it stops when it reaches 0.

- b_2 (START) causes the internal counter to start running when set to 1 by a write operation.
- b_3 (STOP) stops the internal counter when set to 1 by a write operation.

To enable interrupts from the Interval Timer, the bit b_0 of the Nios II control register *ctl3* must be set to 1. The control register *ctl4*, also referred to as *ipending*, is used to determine which interrupt has occurred (if multiple I/O devices are enabled to raise interrupt requests). If an interrupt is disabled using the control register *ctl3*, an interrupt request from the corresponding device will be ignored and it will not show as having occurred in the control register *ctl4*.

Part I

To illustrate the convenience of using a timer, we wish to flash a light on a DE-series board in one-second intervals. Write an assembly-language program that continuously turns the green light LED_0 on for half a second and off for half a second. The program is to run on a DE-series Media Computer that you will download into the FPGA device on the board. Use the Altera Monitor Program to assemble, download and run your program to demonstrate that it works correctly.

Part II

In real-time embedded systems it may be necessary to know how much time is spent on execution of an application program. This time can be determined if we know how many clock cycles are needed to execute the program. The Interval Timer module in the Media Computer has a counter that can be used for this purpose. The counter can be set to some (large enough) value before the execution of the application program is started. Then, the count can be decremented in each clock cycle until the application program is finished.

Write an assembly-language program that determines the number of cycles needed to execute an application program and displays this number (in decimal form) on the seven-segment displays on the DE-series board. Run your program to demonstrate its validity.

A very simple application program, which computes the largest number in a list of integers, is provided as a test design file with this exercise. Use this application to test your program. Then, try your program on some larger application program that you have written.

Part III

Suppose that we wish to determine how long it takes to execute a specific portion, such as a particular loop. This can be done by inserting a **trap** instruction before the loop is entered, and another one at the exit from the loop. The software trap instruction causes the Nios II processor to raise an exception when it encounters this instruction.

Write an assembly-language program that determines the number of cycles needed to execute a loop in an application program and displays this number on the seven-segment displays on the DE-series board. Test your program on the loop in the example program mentioned in Part II. Then, use it on one of your own application programs.

Part IV

Implement the task in Part I using a C-language program.

Part V

Implement the task in Part II using a C-language program.

Part VI

Implement the task in Part III using a C-language program.

Preparation

As a part of your preparation you should do the following:

1. Write the assembly-language programs for Parts I to III.
2. Write the C-language programs for Parts IV to VI.