Laboratory Exercise 7

Implementation of UART and Timer Circuits

This is a more challenging version of Laboratory Exercise 6, in which we investigated the use of UART and Timer circuits as implemented in the DE-series Basic Computer. In this exercise, we will use the Quartus II and Qsys Tool to generate the hardware that implements the UART and Timer functionality in a Nios II system (to correspond to a subset of the DE-series Basic Computer). This exercise requires sufficient knowledge of Verilog or VHDL hardware description languages, to enable the user to instantiate a Nios II module generated by the Qsys Tool into a system that can be implemented on an Altera DE-series Board. A user who is not familiar with the Qsys Tool should read the tutorial *Introduction to the Altera Qsys System Integration Tool*, which can be found in the University Program section of the Altera website.

The purpose of the exercise is to learn how to send and receive data to/from I/O devices by using a UART circuit, and how to time events by using a timer circuit.

Background

A simple and commonly used scheme for transferring data between a processor and an I/O device is known as the *Universal Asynchronous Receiver Transmitter (UART)*. A UART interface (circuit) is placed between the processor and the I/O device. It handles data one 8-bit character at a time. The transfer of data between the UART and the processor is done in parallel, where all bits of a character are transferred at the same time using separate wires. However, the transfer of data between the UART and the I/O device is done in bit-serial fashion, transferring the bits one at a time.

Altera's Quartus II software includes the Qsys Tool which can be used to implement Nios II systems on FPGA devices. The tool includes many commonly used components that may be included in a system the user wants to design. One component provides an interface of the UART type, called JTAG UART, which establishes a connection between a Nios II processor and the host computer connected to the DE-series board. Figure 1 shows a block diagram of the JTAG UART circuit. On one side the JTAG UART connects to the Avalon switch fabric, which interconnects the Nios II processor, the memory chips, and the I/O interfaces. On the other side it connects to the host computer via the USB-Blaster interface. The JTAG UART core contains two registers: *Data* and *Control*, which are accessed by the processor as memory locations. The address of the *Control* register is 4 bytes higher than the address assigned to the *Data* register. The core also contains two FIFOs that serve as storage buffers, one for queueing up the data to be transmitted to the host and the other for queueing up the data received from the host. Figure 2 gives the format of the registers.

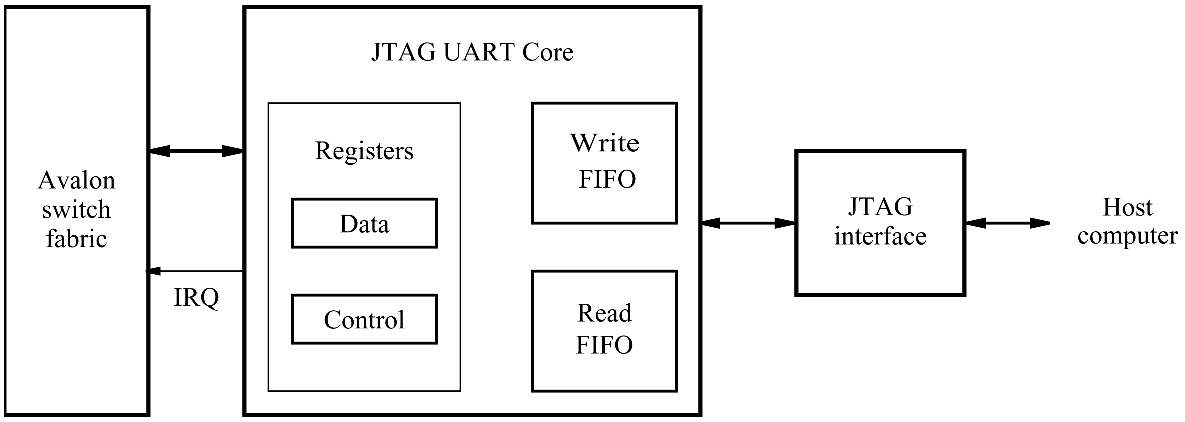


Figure 1. A block diagram of the JTAG UART circuit

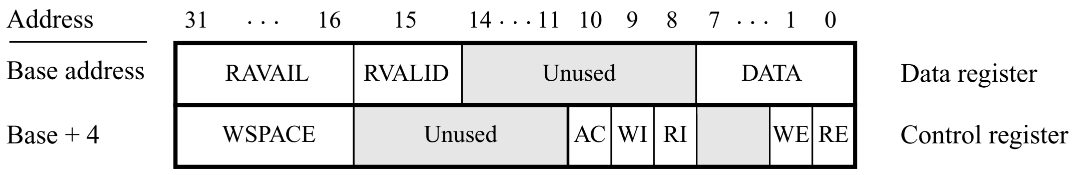


Figure 2. Registers in the JTAG UART.

The fields in the *Data* register are used as follows:

- b_{7-0} (DATA) is an 8-bit character to be placed into the Write FIFO when a Store operation is performed by the processor, or it is a character read from the Read FIFO when a Load operation is performed.
- b_{15} (RVALID) indicates whether the DATA field contains a valid character that may be read by the processor. This bit is set to 1 if the DATA field is valid; otherwise it is cleared to 0.
- b_{31-16} (RAVAIL) indicates the number of characters remaining in the Read FIFO (after this read).

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

- b_0 (RE) enables the read interrupts when set to 1.
- b_1 (WE) enables the write interrupts when set to 1.
- b_8 (RI) indicates that a read interrupt is pending if the value is 1. Reading the *Data* register clears the bit to 0.
- b_9 (WI) indicates that a write interrupt is pending if the value is 1.
- b_{10} (AC) indicates that there has been JTAG activity (such as the host computer polling the JTAG UART to verify that a connection exists) since the bit was cleared. Writing a 1 to AC clears it to 0.
- b_{31-16} (WSPACE) indicates the number of spaces available in the Write FIFO.

More information on the JTAG UART may be found in Chapter 5 of the *Altera Embedded Peripherals Handbook*.

In this exercise, we will use the JTAG UART to transfer ASCII-encoded characters between a Nios II processor implemented on the DE-series board and the host computer. We will also make use of an "interval timer" circuit to provide fixed delays. A block diagram of the hardware that we want use is shown in Figure 3. We will design this hardware in Part I. Then, in Parts II to V, we will use the designed system to implement some programming tasks.

Part I

Use the Qsys Tool to create the system in Figure 3, which consists of a Nios II/s processor, an on-chip memory block, a JTAG UART and an Interval Timer.

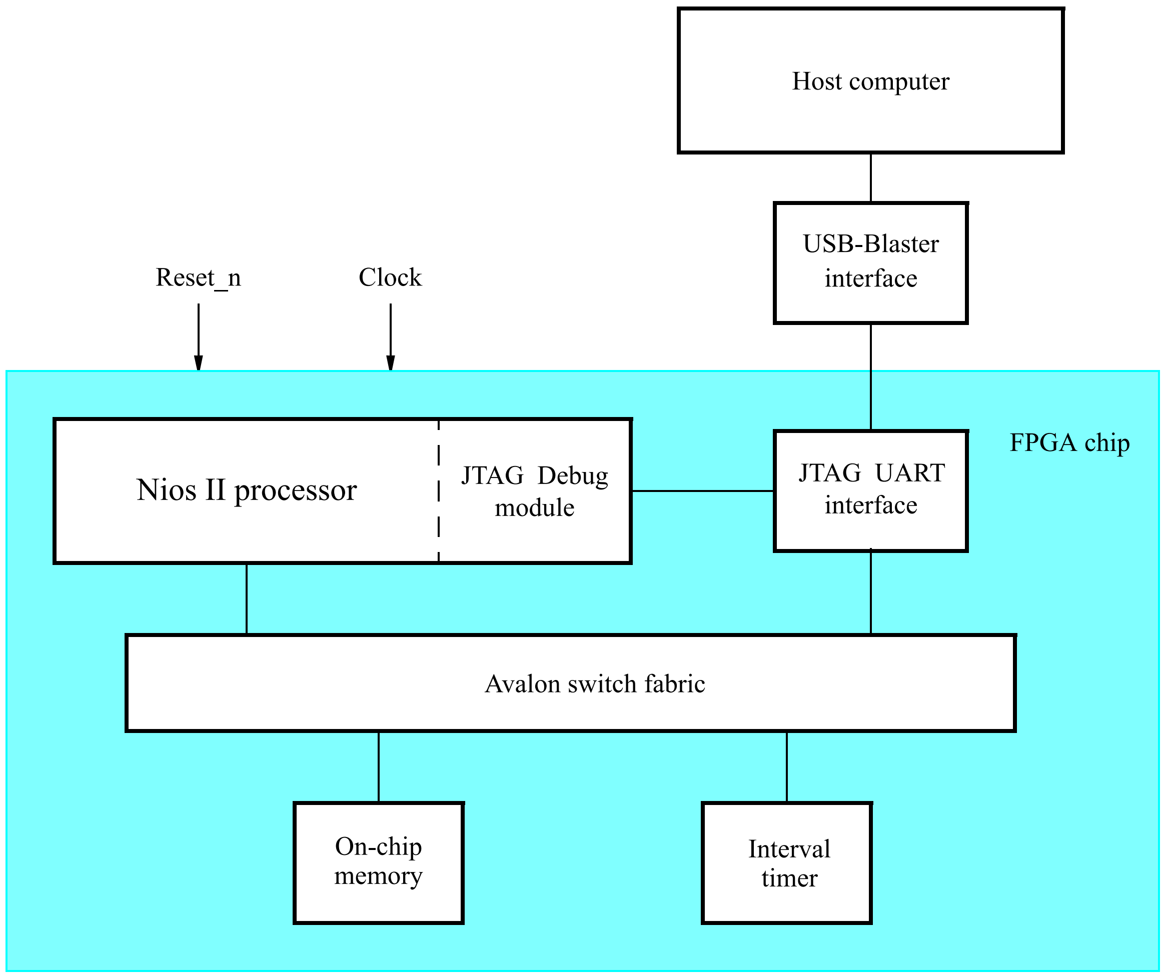


Figure 3. The desired Nios II system on an FPGA chip.

Implement the system as follows:

1. Create a new Quartus II project. Select the Cyclone-series device family for your DE-series board. From the list of available devices, choose the appropriate device name for your DE-series board. A list of devices names on DE-series boards can be found in Table **??**.

Board	Device Name
DE0	Cyclone III EP3C16F484C6
DE1	Cyclone II EP2C20F484C7
DE2	Cyclone II EP2C35F672C6
DE2-70	Cyclone II EP2C70F896C6
DE2-115	Cyclone IVE EP4CE115F29C7

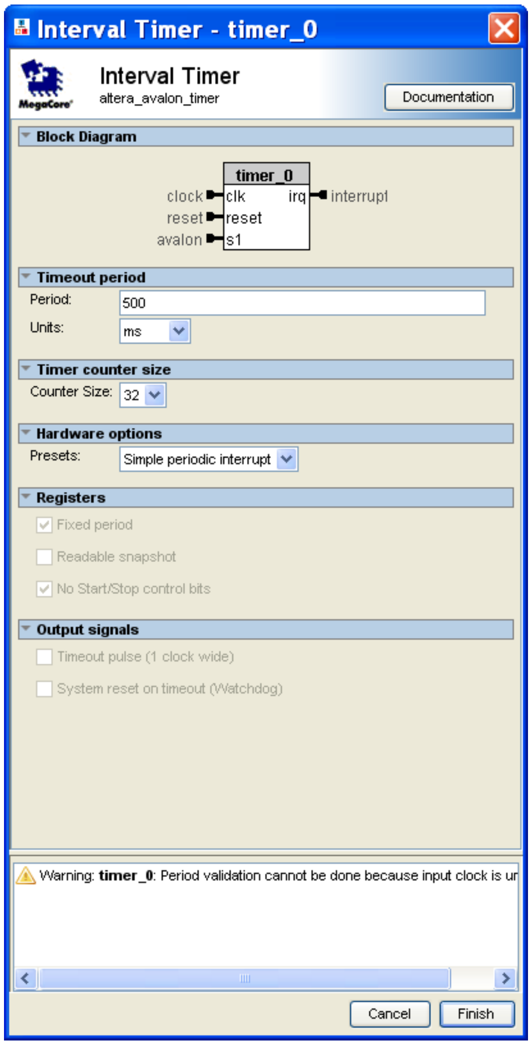
Table 1: DE-series FPGA device names.

2. Use the Qsys Tool to create a system named *nios_system*, which includes the following components:

   - Nios II/s processor with JTAG Debug Module Level 1
     - Do not choose Hardware Multiply and Hardware Divide options
     - Choose on-chip memory for both Reset and Exception vectors

- On-chip memory - RAM mode and 8 Kbytes in size
- JTAG UART - Located in the component section Interface Protocols > Serial; use the default settings
- Interval Timer - Located in Peripherals > Microcontroller Peripherals
  - For the Hardware Options - Presets choose Simple periodic interrupt
  - For the Timeout Period choose a Fixed Period of 500 msec

  as shown in Figure 4.


Interval Timer - timer_0

Interval Timer
altera_avalon_timer

Block Diagram

Timeout period
Period: 500
Units: ms

Timer counter size
Counter Size: 32

Hardware options
Presets: Simple periodic interrupt

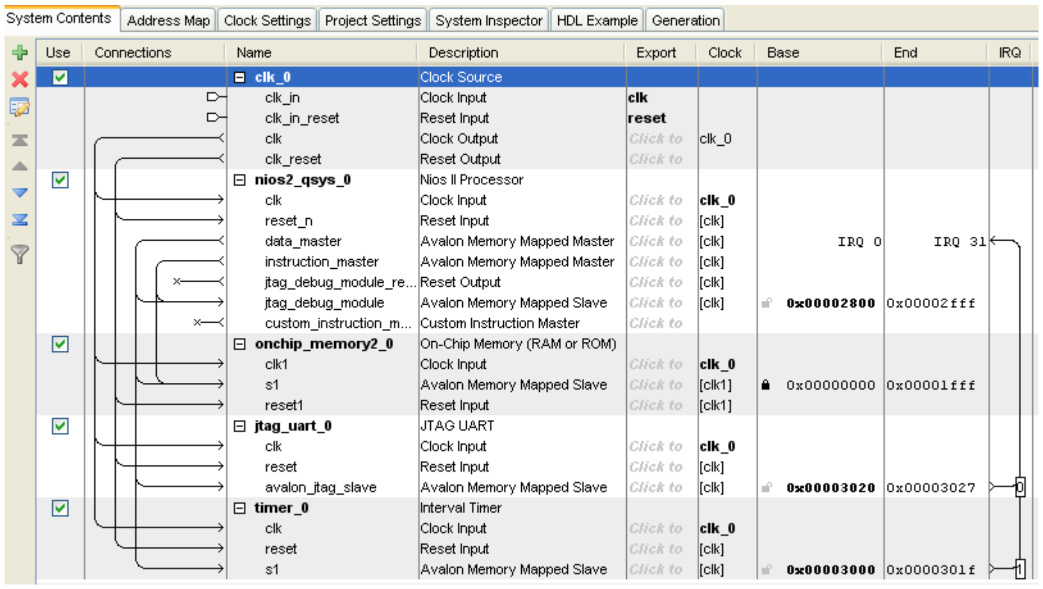
Registers
- [✓] Fixed period
- [ ] Readable snapshot
- [✓] No Start/Stop control bits

Output signals
- [ ] Timeout pulse (1 clock wide)
- [ ] System reset on timeout (Watchdog)

Warning: timer_0: Period validation cannot be done because input clock is ur

Figure 4. Specification for the Interval Timer.

3. Assign the base address of zero to the on-chip memory, and lock this address by clicking on the adjacent lock symbol. Then, let Qsys assign the rest of the addresses.
4. From the System menu, select Assign Interrupt Numbers, which will assign IRQ 0 to the JTAG UART and IRQ 1 to the timer. You should now have the system shown in Figure 5.



System Contents | Address Map | Clock Settings | Project Settings | System Inspector | HDL Example | Generation

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		clk_0	Clock Source					
		clk_in	Clock Input	clk				
		clk_in_reset	Reset Input	reset				
		clk	Clock Output	Click to	clk_0			
		clk_reset	Reset Output	Click to				
✓		nios2_qsys_0	Nios II Processor					
		clk	Clock Input	Click to	clk_0			
		reset_n	Reset Input	Click to	[clk]			
		data_master	Avalon Memory Mapped Master	Click to	[clk]	IRQ 0	IRQ 31	
		instruction_master	Avalon Memory Mapped Master	Click to	[clk]			
		jtag_debug_module_re...	Reset Output	Click to	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Click to	[clk]	0x00002800	0x00002fff	
		custom_instruction_m...	Custom Instruction Master	Click to				
✓		onchip_memory2_0	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Click to	clk_0			
		s1	Avalon Memory Mapped Slave	Click to	[clk1]	0x00000000	0x00001fff	
		reset1	Reset Input	Click to	[clk1]			
✓		jtag_uart_0	JTAG UART					
		clk	Clock Input	Click to	clk_0			
		reset	Reset Input	Click to	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	Click to	[clk]	0x00003020	0x00003027	0
✓		timer_0	Interval Timer					
		clk	Clock Input	Click to	clk_0			
		reset	Reset Input	Click to	[clk]			
		s1	Avalon Memory Mapped Slave	Click to	[clk]	0x00003000	0x0000301f	1

Figure 5. The Nios II system implemented by the Qsys Tool on an FPGA chip.

5. Generate the system, exit the Qsys Tool and return to the Quartus II software.

6. Write a Verilog/VHDL module that instantiates the generated Nios II system and establishes the connections between the clock and reset ports of the *nios_system* module and the CLOCK_50 and pushbutton KEY[0] pins on the DE-series board.

7. Assign the pin connections by referring to Table **??**.

Port	DE0	DE1	DE2	DE2-70	DE2-115
CLOCK_50	PIN_G21	PIN_L1	PIN_N2	PIN_AD15	PIN_Y2
KEY[0]	PIN_H2	PIN_R22	PIN_G26	PIN_T29	PIN_M23

Table 2: Pin assignments.

Alternatively, you can simply import the pin assignments from a Quartus II Setting File with Pin Assignments. For example, the pin assignments for the DE2-115 board are provided in the *DE2-115.qsf* file, which can be found on Altera's DE2-115 web pages.

8. Add the *nios_system.qip* file (IP Variation file) to your Quartus II project.

9. Compile your Quartus II project.

Part II

The JTAG UART can send ASCII characters to the Altera Monitor Program, which will display these characters in its terminal window. When the WSPACE field in the *Control* register of the JTAG UART has a non-zero value, the JTAG UART can accept a new character to be written to the Altera Monitor Program. To write a character to the Monitor Program, poll (continuously read) this register until space is available. Once space is available the ASCII character can be written into the *Data* register of the JTAG UART.

Write a Nios II assembly-language program to display the letter Z approximately every half second in the terminal window of the Monitor Program. Create and execute the program as follows:

1. Using the Nios II assembly language, write a loop which reads the *Control* register in the JTAG UART and keeps looping until there is some write space available. Then, write the letter Z to the *Data* register.
2. Using the Monitor Program, create a new project for this part. Select Custom System and under System Details give the *.qsys* and *.sof* files for the system that you designed in Part I. Download your system into the FPGA chip on the DE-series Board.
3. Compile and load your assembly-language program.
4. Run this program using the single step feature only. If you run this program using the Continue mode, the character will be sent to the terminal window faster than the Monitor Program can handle.
5. In the assembly-language code, create a delay loop so that characters are only printed approximately every half of a second.
6. Recompile, load and run the program.

Part III

The JTAG UART can also receive ASCII characters from the terminal window. The RVALID bit, b_{15}, in the *Data* register indicates whether a value in the DATA field is a valid received ASCII character. If more characters are still waiting to be read, the RAVAIL field will have a non-zero value.

Write a program that implements a “typewriter-like” task; that is, read each character received by the JTAG UART from the host computer and then display this character in the terminal window of the Monitor Program. Use polling to determine if a new character is available from the JTAG UART.

Note: The cursor must be in the terminal window of the Monitor Program to write characters to the JTAG UART’s receive port.

Part IV

Polling the JTAG UART to determine its state is inefficient. The overhead of determining if a new character is available significantly impacts the performance of the program. Instead of polling, it is possible to use the interrupt mechanism, which allows the processor to do useful work while it is waiting for an I/O transfer to take place.

Create an interrupt-service routine to read characters recieved by the JTAG UART from the host computer. Place the interrupt-service routine at the hex address 0x20, which is the default location assigned for the *exception handler*. The exception return address in the *ea* register must be decremented by 4 for external interrupts.

Nios II’s control register *ctl3*, also referred to as *ienable*, enables interrupts on an individual basis. If in your system the JTAG UART is assigned the interrupt level 0, then in the control register *ctl3*, bit $ctl3_0$ must be set to 1 to enable the JTAG UART’s interrupts. In addition, control register *ctl0*, which is also referred to as *status*, must have its bit $ctl0_0$ set to 1. This is the processor interrupt-enable bit - setting it to 1 causes the processor to accept external interrupts.

Perform the following:

1. Write an interrupt-service routine to read a character from the JTAG UART. Note that
   - The interrupt service routine must be placed at the memory address 0x20.
   - To enable interrupts, appropriate values must be written to the *Control* register of the JTAG UART, and the Nios II’s control registers *ctl0* and *ctl3*.

2. In your interrupt-service routine, use the polling approach to display the characters received from the host computer in the terminal window of the Monitor Program.
3. Compile, load and run your program.

If your program does not work at a first try, you will have to debug it and fix the errors. One aid in debugging is the single-step feature of the Altera Monitor Program, which allows the user to observe the flow of execution and the contents of Nios II registers as each instruction is being executed. However, this approach cannot be used when interrupts are involved, because interrupts are automatically disabled when single stepping through a program. Therefore, use breakpoints as a debugging aid.

Note also that interrupts are automatically disabled when the execution of an interrupt-service routine begins and re-enabled upon exit from this routine. This means that if some application requires nested interrupts, the interrupts have to be re-enabled within the interrupt-service routine.

Part V

In this part we wish to write a program that uses interrupts to read characters received by the JTAG UART from the host computer and displays the last character received repeatedly every 500 milliseconds. In Part II we used a delay loop to generate an approximate time interval of this length. Now, we want to use the Interval Timer circuit for this purpose. The Interval Timer should interrupt the processor every 500 ms, at which point a character should be written to the Monitor Program's terminal window.

The Interval Timer has an internal counter which is set to a specified 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 specified value. The Interval Timer has a set of 16-bit registers that can be accessed as memory locations, similar to the JTAG UART. These registers are shown in Figure 6. The address of the *Status* register is the base address assigned to the Interval Timer when you designed your system in Part I. The address of the *Control* register is four bytes higher. The next two word addresses are assigned to registers that specify the starting value for the counter, as indicated in the figure. Note that one of these registers holds the low-order 16 bits of the value and the other register holds the high-order 16 bits of the value.

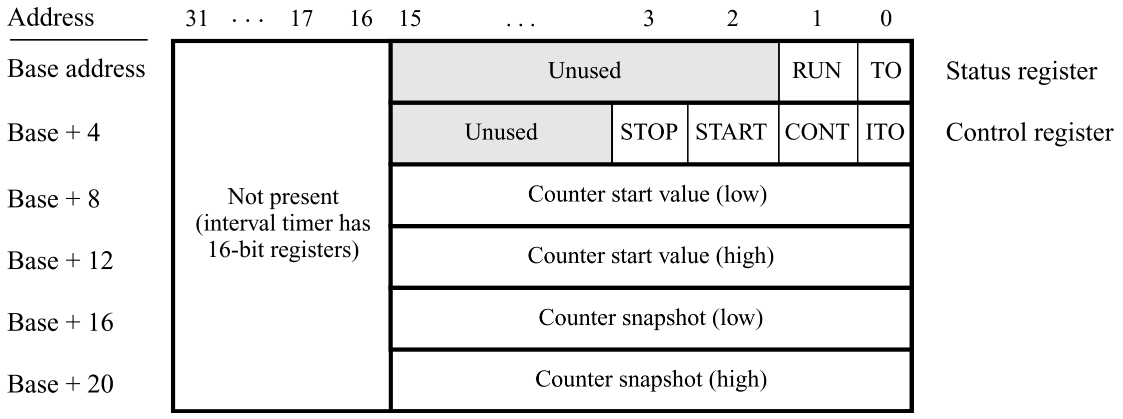


Figure 6. Registers in the Interval Timer.

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 initial 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.

More information on the Interval Timer may be found in Chapter 12 of the *Altera Embedded Peripherals Handbook*.

To enable both interrupts, from the Interval Timer and the JTAG UART for reading characters (from Part III), the bits b_8 and b_0 of the control register *ctl3* must both be set to 1. The control register *ctl4*, also referred to as *ipending*, can be used to determine which interrupt has occurred. If an interrupt is disabled using the control register *ctl3*, it will not cause the interrupt-service routine to execute, nor will it show as being triggered in the control register *ctl4*, even if the device is driving its interrupt-request line to 1.

Perform the following steps:

1. Modify the program from Part III, so that the main program sets the desired count period in the Interval Timer, enables interrupts, and then waits in an infinite loop.
2. Modify the interrupt-service routine to handle both the Interval Timer and the JTAG UART's read interrupts.
3. To enable interrupts, appropriate values must be written to the *Control* register of the JTAG UART, the *Control* register of the Interval Timer and the Nios II control registers *ctl0* and *ctl3*.
4. Compile, load and run your program.

Preparation

As your preparation do the following:

1. Design the system in Figure 3.
2. Write the assembly-language programs for parts II to V.