Laboratory Exercise 4

Polling and Interrupts

The purpose of this exercise is to learn how to send and receive data to/from I/O devices. There are two methods used to indicate whether or not data can be sent or is ready to be received to/from I/O devices. The first method, *polling*, is where the processor queries devices to see if they can receive data or have data available. The second method, *interrupts*, is when devices indicate to the processor that they can receive data or that they have data available, without the processor explicitly requesting.

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 fashion, 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 SOPC Builder can implement an interface of the UART type for use in Nios II systems, which is called the JTAG UART. This circuit can be used to provide a connection between a Nios II processor and the host computer connected to Altera's DE2 board. Figure 1 shows a block diagram of the JTAG UART circuit. On one side the JTAG UART connects to the Avalon switch fabric, while on the other side it connects 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.

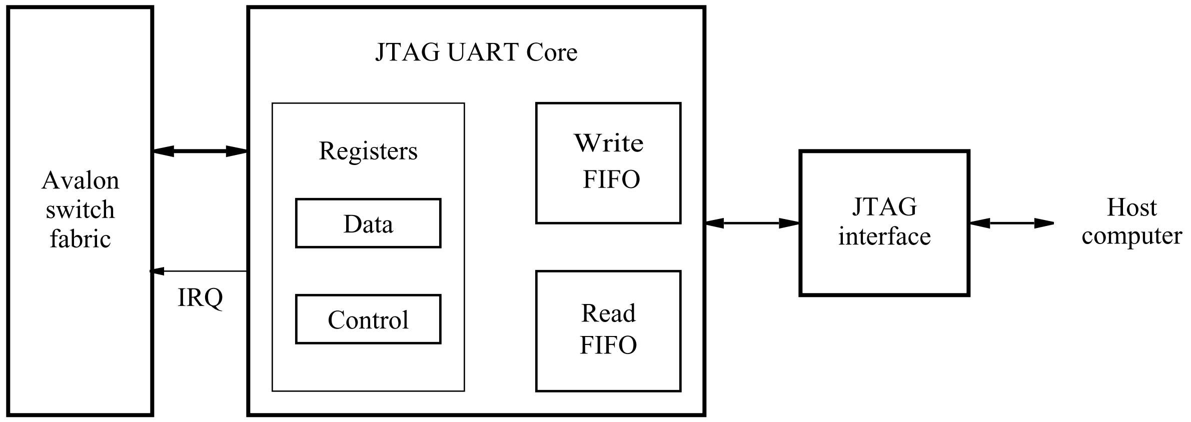
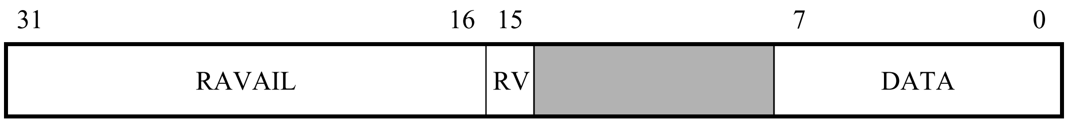
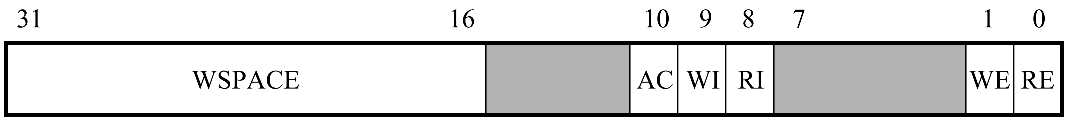


Figure 1. A block diagram of the JTAG UART circuit



(a) Data register



(b) Control register

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 DE2 board and the host computer. We will also make use of an "interval timer" circuit to provide fixed delays.

Part I

Use the SOPC Builder to create the system in Figure 3, which consists of a Nios II/e processor, a JTAG UART, a memory block and an Interval Timer.

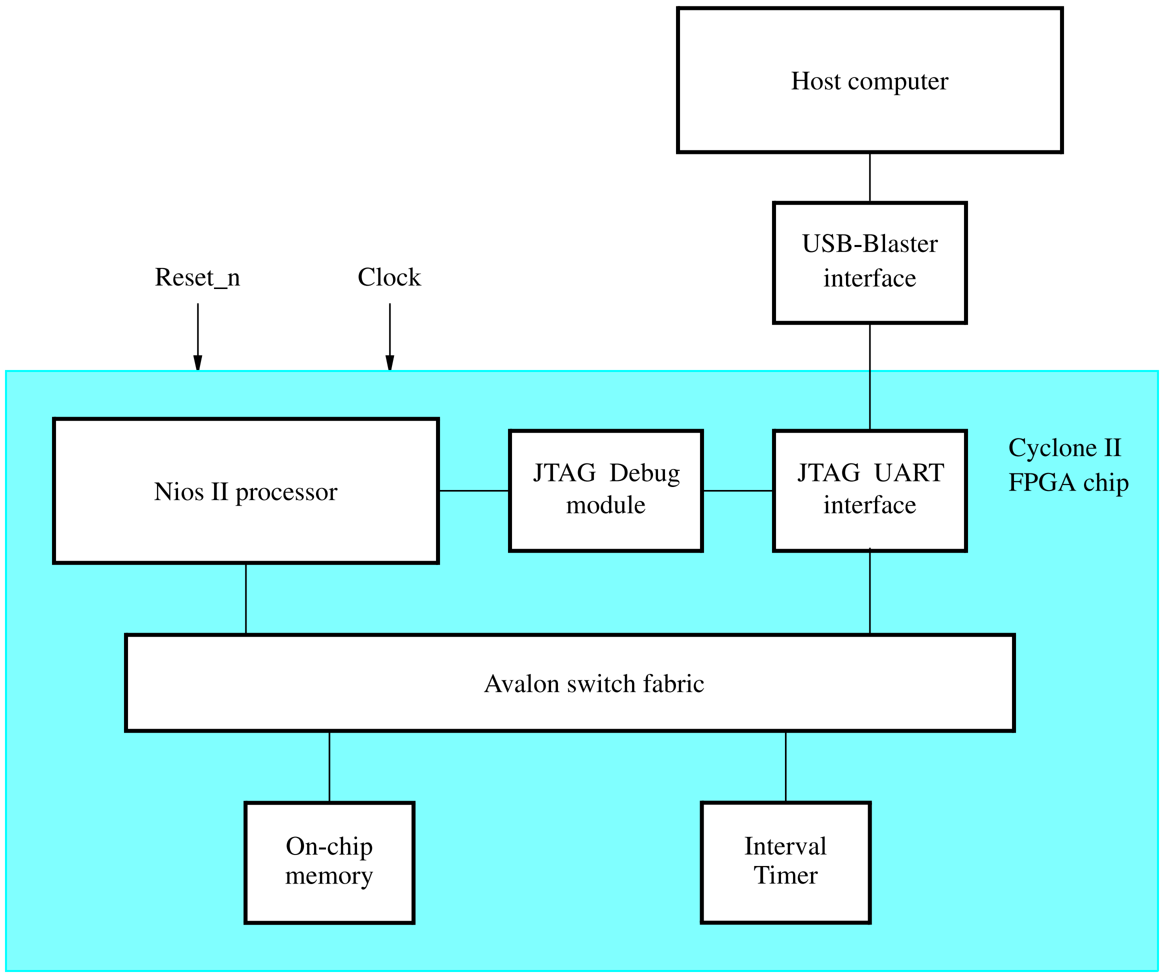
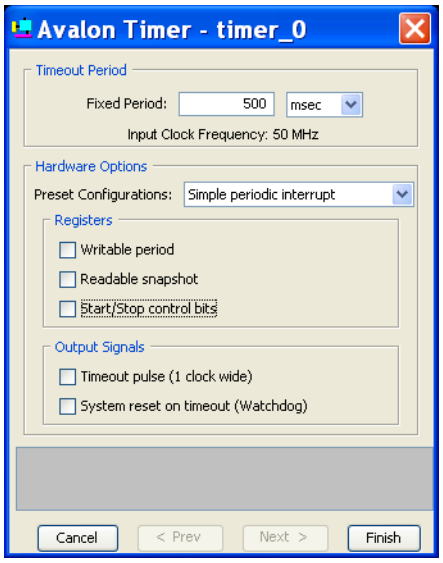


Figure 3. The desired Nios II system

Implement the system as follows:

1. Create a new Quartus II project. Select Cyclone II EP2C35F672C6 as the target chip, which is the FPGA chip on the Altera DE2 board.
2. Use the SOPC Builder to create a system named *nios_system*, which includes the following components:
   - Nios II/s processor with JTAG Debug Module Level 1
   - On-chip memory - RAM mode and 32 Kbytes in size
   - JTAG UART - use the default settings
   - Interval Timer - Located in the component section named Other
     - For the Hardware Options - Preset Configurations choose Simple periodic interrupt
     - For the Timeout Period choose a Fixed Period of 500 msec

   as shown in Figure 4.



Timeout Period

Fixed Period: 500 msec

Input Clock Frequency: 50 MHz

Hardware Options

Preset Configurations: Simple periodic interrupt

Registers

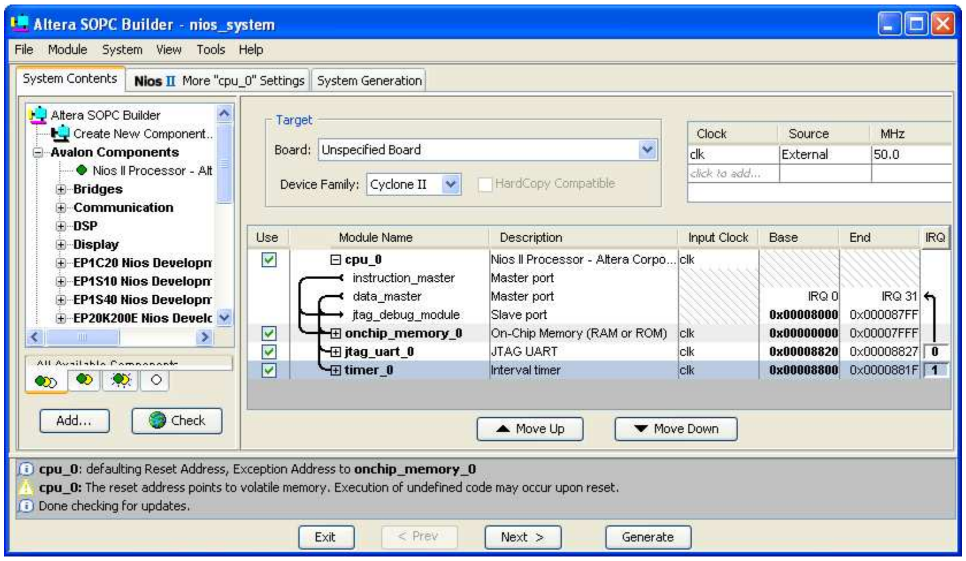
- [ ] Writable period
- [ ] Readable snapshot
- [ ] Start/Stop control bits

Output Signals

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

Figure 4. Specification for the Interval Timer

3. From the System menu, select Auto-Assign Base Addresses. You should now have the system shown in Figure 5.



Target

Board: Unspecified Board

Device Family: Cyclone II

HardCopy Compatible

Clock	Source	MHz
clk	External	50.0
click to add...		

Use	Module Name	Description	Input Clock	Base	End	IRQ
✓	cpu_0	Nios II Processor - Altera Corpo...	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00008000	0x000087FF	
✓	onchip_memory_0	On-Chip Memory (RAM or ROM)	clk	0x00000000	0x00007FFF	
✓	jtag_uart_0	JTAG UART	clk	0x00008820	0x00008827	0
✓	timer_0	Interval timer	clk	0x00008800	0x0000881F	1

```
cpu_0: defaulting Reset Address, Exception Address to onchip_memory_0
cpu_0: The reset address points to volatile memory. Execution of undefined code may occur upon reset.
Done checking for updates.
```

Figure 5. The Nios II system implemented by the SOPC Builder.

4. Generate the system, exit the SOPC Builder and return to the Quartus II software.
5. Instantiate the generated Nios II system within a Verilog/VHDL module.
6. Assign the pin connections:

- clk - PIN_N2 (This is a 50 MHz clock)
- reset_n - PIN_G26 (This is the pushbutton switch KEY_0)

7. Compile the Quartus II project.
8. Program and configure the Cyclone II FPGA on the DE2 Board to implement the generated system.

Part II

The JTAG UART can send ASCII characters to the Altera Debug Client, 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 Debug Client. To write a character to the Debug Client, 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 500 ms in the terminal window of the Altera Debug Client. 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.
2. Write the letter Z to the *Data* register.
3. Using the Altera Debug Client, compile and load the 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 Debug Client can handle.
5. In the assembly-language code, create a delay loop so that characters are only printed approximately every half second.
6. Recompile, load and run the program.

Part III

The JTAG UART can receive ASCII characters from the terminal window, as well as write them. The RVALID bit, b_{15}, in the *Data* register indicates whether or not 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 that is received by the JTAG UART from the host computer and then display this character in the terminal window of the Debug Client. 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 Debug Client to write characters to the JTAG UART's receive port.

Part IV

Polling the JTAG UART is inefficient, due to the overhead of reading its registers to determine the UART's state. 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, because this is the default location for the *exception* *handler* as chosen by the SOPC Builder. The exception return address in the *ea* register must be decremented by 4 for external interrupts. Figure 5 gives a skeleton of the interrupt-service routine written in the Nios II assembly language.

```
.include "nios_macros.s"

.text

.org   0x20                          /* Place the interrupt service routine */
                                     /*   at the appropriate address */

ISR:
    rdctl      et, ctl4              /* Check if an external */
    beq        et, r0, SKIP_EA_DEC   /*   interrupt has occurred */

    subi       ea, ea, 4             /* If yes, decrement ea to execute */
                                     /*   interrupted instruction */

SKIP_EA_DEC:

    ... the interrupt-service routine

END_ISR:
    eret                             /* Return from exception */


.global _start
_start:                              /* Program start location */

    ... enable interrupts code

    ... the main program code

LOOP:
    br         LOOP                  /* Endless loop */

.end
```

Figure 6. Assembly language code skeleton for the interrupt-service routine.

Nios II's control register *ctl3*, also referred to as *ienable*, enables interrupts on an individual basis. Note that when the system was created in Part I, the JTAG UART was placed at interrupt level 0. This means that bit *0* of the control register *ctl3* must be set to 1 to enable the JTAG UART's interrupts.

Perform the following:

1. Create 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 Altera Debug Client.
3. Compile, load and run your program.

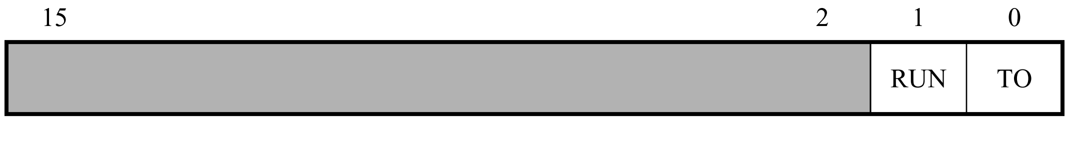
If your program does not work at a first try, you will have to debug it and fix the errors. One aid in debugging is the single-step feature of the Altera Debug Client, which allows the user to observe the flow of execution and the contents of Nios II registers as each instruction is being executed. However, 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 required nested interrupts, the interrupts would have to be re-enabled within the interrupt-service routine.

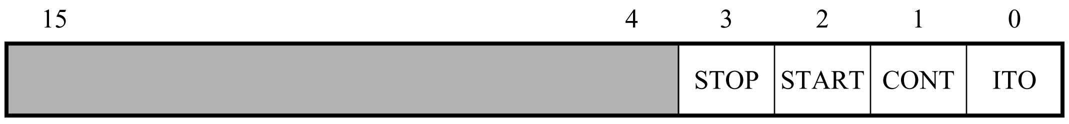
Part V

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 Debug Client'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. Two of these registers, *Status* and *Control*, are shown in Figure 7. The address of the *Status* register is the base address assigned to the Interval Timer, while the address of the *Control* register is 4 bytes higher.



(a) Status register



(b) Control register

Figure 7. 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.
- 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 IV), the bits b_1 and b_0 of the control register *ctl3* must both be set to 1. As seen in Figure 5, the JTAG UART is on interrupt line 0 (or b_0) and the Interval Timer is on interrupt line 1 (or b_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 IV, so that the main program 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.