SPART

A Special Purpose Asynchronous Receiver/Transmitter

Introduction

In this miniproject you are to implement a Special Purpose Asynchronous Receiver/Transmitter (SPART). The SPART can be integrated into the processor of your final project to serve as the serial I/O interface between the processor and serial I/O port on the lab workstations. Using the Hypertermial Accessory program, this will permit you to input characters from the keyboard and to output characters to the screen on the lab workstations.

The objectives of this *miniproject* are to:

- Familiarize you with design in the ECE 554 Virtex-5 Board environment
- Practice the use of an HDL in design
- Generate a useful design for your final project
- Acquire an initial experience in efficiently and effectively performing a design as a team

SPART Design

SPART Functional Description

This section specifies the subsystem to be designed. In order to classify the description, some terminology is necessary. The term output or write are used when the processor is sending information to the SPART. The term *transmit* is used when the SPART is transmitting data to the serial I/O port on the workstation. Conversely, the terms *input* or *read* are used when the processor is retrieving information from the SPART. Finally, the term receive is used when the SPART is receiving data from the serial I/O port on the workstation.

IOADDR	SPART Register
00	Transmit Buffer (IOR/W = 0); Receive Buffer (IOR/W = 1)
01	Status Register (IOR/W = 1)
10	DB(Low) Division Buffer
11	DB(High) Division Buffer

Table 1: Address Mappings

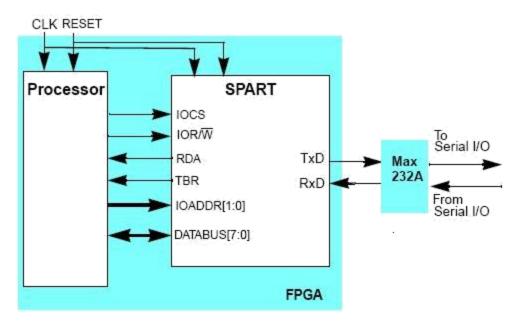


Figure 1: SPART Environment

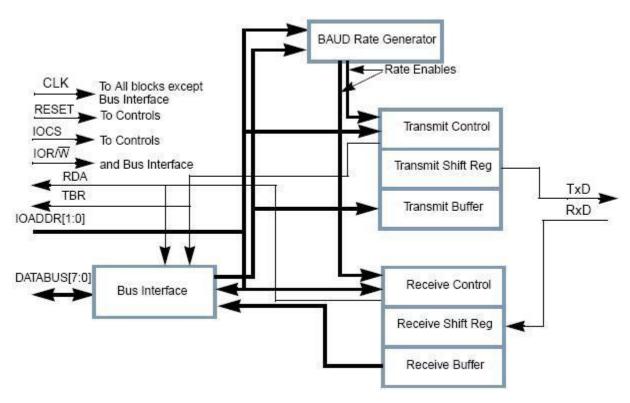
A top level diagram of the SPART and its environment is shown in Figure 1. The FPGA interfaces with a chip on the board which generates appropriate voltage levels for the RS232 interface. The TxD pin transmits serial data from the FPGA and RxD receives serial data.

The SPART and Processor driver share many interconnections in order to control the reception and transmission of data. On the left, the SPART interfaces to an 8-bit, 3-state bidirectional bus, DATABUS[7:0]. This bus is used to transfer data and control information between the Processor and the SPART. In addition, there is a 2-bit address bus, IOADDR[1:0] which is used to select the particular register that interacts with the DATABUS during an I/O operation. The IOR/W signal determines the direction of data transfer between the Processor and SPART. For a Read (IOR/W=1), data is transferred from the SPART to the Processor and for a Write (IOR/W=0), data is transferred from the processor to the SPART. IOCS and IOR/W are crucial signals in properly controlling the three-state buffer on DATABUS within the SPART. Receive Data Available (RDA), is a status signal which indicates that a byte of data has been received and is ready to be read from the SPART to the Processor. When the read operation is performed, RDA is reset. Transmit Buffer Ready (TBR) is a status signal which indicates that the transmit buffer in the SPART is ready to accept a byte for transmission. When a write operation is performed and the SPART is not ready for more transmission data, TBR is reset. The SPART is fully synchronous with the clock signal CLK; this implies that transfers between the Processor and SPART can be controlled by applying IOCS, IOR/W, IOADDR, and DATABUS (in the case of a write operation) for a single clock cycle and capturing the transferred data on the next positive clock edge. The received data on RxD, however,

is asynchronous with respect to CLK. Also, the serial I/O port on the workstation which receives the transmitted data from TxD has no access to CLK. This interface thus constitutes the "A" for "Asynchronous" in SPART and requires an understanding of RS-232 signal timing and (re)synchronization.

SPART Structure

A block diagram of the SPART is given in Figure 2. Each subsystem is briefly described in this section.





Bus Interface

The Bus Interface contains the 3-state drives which attach the SPART to the DATABUS. In addition, it contains the multiplexer which selects the Receive Buffer or the Status Register. The Status Register consists of RDA and TBR in positions 0 and 1, respectively. The Status Register is not actually a register, but just connections from RDA and TBR which are stored at their respective sources. The remaining six bits connected to the multiplexer for the Status Register are zeros. Note that RDA and TBR are provided both as direct signals to the Processor and as part of the Status Register content accessible by the Processor via the DATABUS. If interrupt-based I/O is used for the SPART, then the

direct signals can be used as inputs to the interrupt system. If program-based I/O is used, then the Status Register content (RDA, TBR) can be accessed by the program using an I/O read operation on the Status Register to determine if an I/O data operation is needed. In either case, RDA and TBR can be used as part of a "handshake" between the processor and the SPART during I/O transactions.

In addition to the above datapath constructs, the Bus Interface also contains combinational control logic for the above. In particular, it uses IOCS and IOR/W to make sure that the 3-state drivers are never turned on in conflict with other drivers on DATABUS.

Baud Rate Generator

The BAUD Rate Generator (BRG) produces an enabling signal or signals for controlling the transmitter and the receiver. In traditional UART designs, transmitter and receiver clocks, which typically are the same frequency, are used to perform the necessary timing for controlling the BAUD rate of the transmitted serial information and for controlling the sampling of received information. Since we have no separate clock source, we cannot use this approach, but must instead depend upon the BRG to produce enable signals for these purposes instead of separate clocks. The reason for producing an enable signal instead of a clock is to avoid the problem of having multiple clock domains. The enable signal is produced by a down counter and decoder circuit to perform divisions of the frequency of CLK. Note that in Verilog, an enable is not used as a clock, but as a condition for performing or not performing actions:

```
always@(posedge clk)
if (enable)
...
else
```

```
....
```

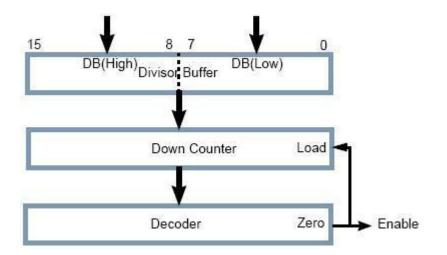


Figure 3: Baud Rate Generator

The frequency of the occurrence of the most frequent enable, which has duration of one CLK period, is typically $2^n \times$ baud rate, where n ranges from 2 to 4. We will assume that 4 is used. In the design of the BRG, we have a special problem in that we will vary the frequency of CLK driving the BRG. Thus, the BRG must be programmable to maintain a fixed baud rate in the face of a changing clock frequency. Programming is achieve by the processor loading two bytes, DB(High) and DB(Low) into the Divisor Buffer in Fig. 3. This buffer drives the data inputs to the down counter as shown in Figure 3. The divisor is the nearest integer to (clock frequency/ $(2^n \times \text{baud rate}) - 1$). When the counter contains 0, it is loaded with the divisor. So the count goes from the divisor to zero at the CLK rate. If the decoder consists of a zero detect on the entire counter, then an enable is produced for a single clock period at a rate of one every (CLK/divisor + 1) = $CLK/(2^n \times baud rate)$ With the divisor ranging from 1 to 65,535, division by 2 through 65,536 can be accomplished. By using appropriate divisor and designing appropriate decoder, pulses can be produced at 2^n times the baud rate. If an additional enable is required, for example, at the baud rate itself, it can be generated by a 4-bit down counter with a zero decoder and with the $(2^n \times \text{baud rate})$ enable as an input. The following example illustrates the Basic Baud Rate concept.

Example: Suppose that CLK has frequency 25 MHz and that the desired enable frequency is $(2^n \times \text{baud rate})$ where n = 4 and the baud rate is 9600 (bits/second). The divisor required is 25,000,000/(16 \times 9,600) – 1 = 161.76 which is rounded to 162. This becomes A2 in hexadecimal. At count time 0, the counter will be loaded with 162 and will be decremented every 40 ns. The counter will be 0 at count time 163, so the interval of time for the counter to be loaded and count down is 163 \times 40 ns = 6.52 µs. Inverting, the frequency is 153,374.23 Hz. Dividing by 16, this corresponds to a baud rate of 9586 bits/second. Based on calculations, we have estimated that an error of + or - 3 percent is tolerable, so this design is well within tolerance.

In your final project, to insure communication with the "console" (Hyperterminal) immediately after a reset, a divisor should be loaded into the Divisor Buffer upon processor reset. This divisor should be in dedicated locations in memory. The memory should also contain a "boot program" that executes automatically on reset to load the divisor in memory into the Divisor Buffer. The clock frequency has been set before the reset is applied. Further, in order to provide means of setting a division before your system can execute code, the reset should initialize the Divisor Buffer to the divisor corresponding to a clock of 100 MHz and 9600 bits/second. You can modify this value to something else if necessary for your design.

More Information

For information on the remaining parts of the SPART, please consult the reference and the files in the folder Miscellaneous HDL Code for UART-Like Hardware in the FAQ folder on the course Website. Note that this design does not need to contain many of the features in these examples. For example, there is no synchronous operation as in the 8251A and there is no initialization except for Reset and the loading of the BAUD Rate Generator. Further, there is no error checking, no parity bit and a fixed number of stop bits.

Hardware Testbench

In order to test your SPART in the lab, you will need circuitry to mimic the behavior of the Processor. We will refer to this as a hardware testbench. This hardware testbench should be able to:

- Demonstrate the ability to transmit and receive characters by, for example, entering characters on a dumb terminal keyboard and echoing them back to the dumb terminal display
- Loading the Baud Rate Generator with an arbitrary value.

The testbench needs to provide four hardwired divisor values. The testbench must load one of these values into the Divisor Buffer after reset has been applied and removed. The value loaded will be determined by the values provided by two DIP switches that are set before reset is applied.

DIP Setting	Baud Rate
00	4800
01	9600
10	19200
11	38400

Hardware Harness

Due to the complexity of the XUP board and the possibility of causing damage by improper design or pin assignment, you are required to use a harness that surrounds your design. Later designs done by you will use your own pin assignments and configurations, but for now a harness will be provided. There are 4 provided files for the miniproject.

• Top_level.v - A the top level of the project which connects to external I/O pins as well as instantiating the SPART and your "Processor"

- Top_level.ucf the Universal Constraints File which specifies I/O pins and clocking parameters
- Spart.v An empty module for you to create your spart within.
- Driver.v An empty module for you to create your Processor driver within.

Implementation Information

When doing implementation, note that the family is Virtex-5 and the device model is XC5VLX110T.

Lab Work

In lab you are to demonstrate the operation of your SPART as follows:

- Show that characters can be transferred between the dumb terminal and your hardware testbench via the SPART.
- By transmitting at multiple baud rates

Report

Your report should consist of the following:

- Verilog code for your entire design with clear, useful commenting
- An accompanying narrative description of the function for the overall SPART and each of the blocks including the testbench
- A record of the experiment conducted including the characters transmitted for a basic test
- A discussion of problems encountered in the design and solutions employed.

Updated 9/1/2014

ECE 554 Miniproject 1 Design Document Fall 2015 Kai Zhao John Roy

Function of SPART

The SPART is divided into several modules that each perform specific parts of the functionality of the device. These modules are the bus interface, the baud rate generator, the transmit unit, and the receive unit.

The bus interface acts as the connection between the processor and the SPART. Its external connections are the databus and the I/O address, ioaddr. The databus acts as a bi-directional 8-bit bus between the processor and the SPART. It carries data that is sent and received as well as baud rate information. The ioaddr line specifies which data will be sent and where. It can specify that data through the databus is to be received, sent, used as the baud rate, or used by the processor as a status register.

The baud rate generator gives enable signals to the transmit and receive control units. It is loaded with a count-down corresponding to a baud rate specified by the processor (either 4800, 9600, 19200, or 38400) and then it counts down from that number on each clock cycle. When the counter reaches zero, an enable signal is sent to the transmit or the receive control unit to send or capture data in the serial line. The serial line operates at a frequency much slower than the internal clock signal of the SPART, so each serial bit is sent at a specific multiple of the internal clock frequency. Similarly, the line is sampled at a multiple of the internal clock frequency.

The transmit control unit takes data from the bus interface and sends it down the serial line. It shifts the data out to convert it from parallel to serial. The shifting happens the baud rate generator gives an enable signal. This ensures that the data is sent out at the rate specified for the serial line.

The receive control unit takes data from the serial port and shifts it into a register to send to the bus interface. It only shifts data in when the baud rate generator sends an enable signal when the divisor is counted down, so the unit only samples data once per cycle of the baud rate.

Together, these units send out some control signals to the processor to ensure proper operation. The receive control unit sends a receive data available signal (rda) to ensure the processor takes in data when it is full and available. The transmit control unit sends a transmit buffer ready (tbr) signal to the processor so the processor won't send data before the buffer is sending out previous data. The processor also sends signals to the SPART including chip selection, clock, reset, and read/write. All of these ensure proper functionality of the SPART.

Record of Experiments

Unit testing was used to ensure proper functionality of each module. For the bus interface, the tests included writing to each baud rate generator register, reading the status register, writing to the transmit controller, and reading from the receive controller. For the baud rate generator, the tests included writing to each baud rate register, timing the enables after each possible baud rate configuration, and ensuring that no enable signals were sent after chip select became low. For the transmit unit, the tests included writing out a specific value to the serial line and checking that it had the proper start and stop bits. For the receive unit, the tests included receiving a specific value from the serial line with the proper start and stop bits and ensuring that the received value was correct.

All of these tests included checks to ensure the values being passed were correct and informational error messages were included to inform the user of the incorrect value being passed.

Discussion of Problems

The main problem we had was figuring exactly what signals each finite state machine should wait for and exactly when to change states. For example, when receiving, we were not sure whether it is done when we receive all 10 bits (1 start bit, 8 bits of data, and 1 stop bit), or whether it was done when we receive only 9 bits and 1 stop bit. We were also not sure what signal we had to wait for before going back to IDLE state and being able to receive again.

One major problem we had was with the hardware and software issues. Xilinx did not work on some computers, ModelSim did not work in some computer, and the internet browser did not work on other computers, some Vertix 5 boards did not work, and some serial cables did not work. Xilinx ISE software, which can be used for development and testing, had frequently crashed or throw file not found errors. To work around this, we did most of our development and testing in ModelSim.

Another problem we faced was with our finite state machines, especially in the hardware testbench. Once we ensured that our state transitions were logically separate from our inputs and outputs the device worked as intended.

A final problem we faced was with viewing internal signals in ModelSim. Internal signals are essential to proper test-benching, and without them we were left with a black box to test. In order to view internal signals, we figured out that we had to run the command "log -r /*" to put all the signals, external and internal, in the objects window, then add them to the waveform from there.

Verilog Code: top_level

```
`timescale 1ns / 1ps
   // Company:
                       University of Wisconsin-Madison
     // Engineer:
                       Kai Zhao, John Roy
 4
5
     11
     // Create Date:
 6
                       2015 Sept 15
     // Design Name:
                       Miniproject1
8
     // Module Name:
                       top_level
     // Project Name:
9
     // Target Devices: Vertix 5
     // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
     // Description:
                       top level for demonstration
     11
    // Dependencies: spart, driver
14
15
     11
16
     // Revision:
     // Revision 0.01 - File Created
     // Additional Comments:
18
19
     11
     // IOCS = I/O chip select. Set to one to activate the SPART
20
     // IOR/Wbar - When 1, the reading from SPART to the PROCESSOR, when 0 reading from the PROCESSOR to the SPART
22
     // RDA - Receive data available => data can be read by the processor from the SPART.
     // TBR - Transmit buffer ready => data can be sent from the processor to the SPART
24
     // IOADDR - I/O address of register to read or write
     // DATABUS - Data to be sent or received
25
     // SPART is fully synchronous with the clock - all transfers occur on a positive clock edge.
26
27
     // The received data on RxD is asynchronous. The transmit via TxD is also asynchronous.
    28
29
                          // inputs and outputs
30 [module top_level(
31
        input clk,
                           // 100mhz clock
        input rst,
                           // Asynchronous reset, tied to dip switch 0
32
         output txd,
                           // RS232 Transmit Data
                           // RS232 Recieve Data
34
        input rxd.
35
         input [1:0] br_cfg // Baud Rate Configuration, Tied to dip switches 2 and 3
   L);
36
37
38
        wire iocs;
                          // wires the connect the spart to driver
39
        wire iorw;
        wire rda;
40
        wire tbr;
41
        wire [1:0] ioaddr;
42
43
        wire [7:0] databus;
44
        // Instantiate your SPART here
45
46 🖂
        spart spart0(
           .clk(clk),
47
48
            .rst(rst),
49
            .iocs (iocs) ,
           .iorw(iorw),
            .rda(rda),
            .tbr(tbr).
53
            .ioaddr(ioaddr),
54
            .databus (databus),
55
            .txd(txd),
56
            .rxd(rxd)
        );
58
59
         // Instantiate your driver here
60 🖂
        driver driver0(
61
            .clk(clk),
62
            .rst(rst),
63
            .br_cfg(br_cfg),
64
            .iocs (iocs) ,
65
            .iorw(iorw).
66
            .rda (rda),
67
            .tbr(tbr),
68
            .ioaddr (ioaddr) ,
            .databus (databus)
69
        );
72 endmodule
```

Spart

// Company:	University of Wisconsin-Madison
// Company: // Engineer:	University of Wisconsin-Madison Kai Zhao, John Roy
// Engineer:	
// Create Date:	2015 Sept 15
// Design Name:	Miniproject1
// Module Name:	spart
// Project Name:	
<pre>// Target Devices:</pre>	
<pre>// Tool versions: // Description:</pre>	ModelSim SE 10.3c; Xilinx 14.7
// Description: //	spart for handling transmission to/from controller
// Dependencies:	bus_interface, baud_rate_generator, transmit_unit, receive_
// // Revision:	
// Revision 0.01 -	File Created
// Additional Comme	ents:
//	
///////////////////////////////////////	
module spart(<pre>// inputs and outputs</pre>
input clk,	
input rst,	
input iocs,	
input iorw,	
output rda,	
output tbr, input [1:0] ioa	ddr
inout [7:0] dat	
output txd,	······································
input rxd	
);	
wire [7:0] rate	
wire [7:0] data	received;
bus interface	<pre>bus_interface0(// instantiate the DUT</pre>
	3).
.iocs(ioc .iosrw(io	
.iosrw(io	srw),
.iosrw(io .rda(rda)	sim),
.iosrw(io	эгм), ,
.iosrw(io .rda(rda) .tbr(tbr)	srw), , , paddr),
.iosrw(io .rda (rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_	srw), , , , , , , , , , , , , , , , , , ,
.iosrw(io .rda (rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_	srw), , , Daddr), databus),
.iosrw(io .rda (rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_	srw), , , , , , , , , , , , , , , , , , ,
.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec	srw), , , databus), data(rate_tx_data), sived(data_received)
.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .); baud_rate_gener	srw), , , , , , , , , , , , , , , , , , ,
.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_gene: .clk(clk),	srw), , , databus), data(rate_tx_data), sived(data_received)
.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_genes .clk(clk),	srw), , , , , , , , , , , , , , , , , , ,
.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_gener .clk(clk), .rst(rst),	srw), , , , , databus), data(rate_tx_data), sived(data_received)
.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc	srw), , , , , databus), data(rate_tx_data), sived(data_received)
.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc	<pre>srw), , , , , , data(rate_tx_data), data(rate_tx_data), sived(data_received) </pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i) .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate_tx_dd)</pre>	<pre>srw), , , , , , data(rate_tx_data), data(rate_tx_data), sived(data_received) </pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_genes .clk(clk), .rst(rst), .ioaddr(ior .rate_tx_d</pre>	<pre>srw), , , data(rate_tx_data), data(rate_tx_data), eived(data_received) cator baud_rate_generator0(// instantiate t) // confirm? not shown in diagram, but we th: , ddr), tta(rate_tx_data), bble)</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_gene: .clk(clk), .rst(rst), .iocs(iccs) .ioaddr(io: .rate_tx_di .enable(end); transmit_unit</pre>	<pre>srw), , , , , , data(rate_tx_data), data(rate_tx_data), sived(data_received) </pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioa .rate_tx_d .enable(ena); transmit_unit f .clk(clk),</pre>	<pre>srw), , , data(rate_tx_data), data(rate_tx_data), eived(data_received) cator baud_rate_generator0(// instantiate t) // confirm? not shown in diagram, but we th: , ddr), tta(rate_tx_data), bble)</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_gene: .clk(clk), .rst(rst), .iocs(iccs) .ioaddr(io: .rate_tx_di .enable(end); transmit_unit</pre>	<pre>srw), , , , , , data(rate_tx_data), data(rate_tx_data), intor baud_rate_generator0(// instantiate th // confirm? not shown in diagram, but we th: , , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .); baud_rate_genen .clk(clk), .rst(rst), .ioaddr(iod .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst),</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i) .databus(.rate_tx_ .data_rec); baud_rate_genes .clk(clk), .rst(rst), .ioaddr(iou .rate_tx_du .enable(enu); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs)</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .icada_rec .icada_rec .clk(clk), .rst(rst), .ioaddr(ioc .rate_tx_d .enable(end); transmit_unit (.clk(clk), .rst(rst), .iocs(iocs) .iors(iocs) .iors(iocs) .iors(iors) .iors(iors) .iors(iors) .iors(iors) .iors(iors) .iors(iors) .iors(iors) .iors(iors) .iors(iors) .iors(iors)</pre>	<pre>srw), , , , , , databus), data(rate_tx_data), sived(data_received) </pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i) .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(ioc .txd(txd),</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(iod .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(iod .rate_tx_dd .rate_tx_dd</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .itk(clk), .rst(rst), .iocs(iocs) .ioaddr(iod .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(iod .rst(rst), .iocs(iocs) .ioaddr(iod .rst(rst), .iocs(iocs) .ioaddr(iod .tbr(tbr), .ioaddr(iod .tbr(tbr), .rate_tx_d .enable(end .tbr(tch), .rate_tx_d .enable(end)</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(iod .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(iod .rate_tx_dd .rate_tx_dd</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i) .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(iot .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(iot .rate_tx_dd .enable(end);</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_gener .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate_tx_dd .enable(end); transmit_unit (.clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate_tx_dd .enable(end .tbr(tbr), .ioadd(ioc .txd(txd), .rate_tx_dd .enable(end .txd(txd), .rate_tx_dd .enable(end .txd(txd), .rate_tx_dd .enable(end .);</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .icad(rla), .rat(rat), .iocs(iocs) .ioaddr(ioc .rate_tx_dd .enable(end); transmit_unit (.clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate(txd), .rst(rst), .iocs(iocs) .ioaddr(ioc .txd(txd), .rate_tx_dd .enable(end);</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_genes .clk(clk), .rst(rst), .ioaddr(iou .rate_tx_du .enable(enu); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(iou .txd(txd), .rate_tx_du .enable(enu); trate_tx_du .enable(enu); trate_tx_du .enable(enu);</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .icad(rla), .rat(rat), .iocs(iocs) .ioaddr(ioc .rate_tx_dd .enable(end); transmit_unit (.clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate(txd), .rst(rst), .iocs(iocs) .ioaddr(ioc .txd(txd), .rate_tx_dd .enable(end);</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(ii .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(ioc .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(iod .rate_tx_dd .enable(end); receive_unit rr .clk(clk), .rst(rst, .iocs(iocs) .iorw(iorw) .td(tdd), .rate_tx_dd .enable(end);</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .itk(clk), .rst(rst), .iocs(iocs) .ioaddr(iod .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .ioaddr(iod .rate(tk), .rst(rst), .iocs(iocs) .ioaddr(iod .tbr(tbr), .ioaddr(iod .tbr(tbr), .ioaddr(iod .td(tdd), .rate_tx_dd .enable(end); trate_tx_dd .enable(end);</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .itk(clk), .rst(rst), .iocs(iocs) .ioaddr(iod .rate_tx_dd .enable(end); transmit_unit (.clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .iorw(iorw) .tbr(tbr), .ioaddr(iod .txd(txd), .rate_tx_dd .enable(end); receive_unit rec .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .trat(rst), .iocs(iocs) .iorw(iorw) .trat(rst), .iocs(iocs) .iorw(iorw) .trat(rat), .iocs(iocs) .iorw(iorw) .iorw(iorw) .ioaddr(iod .rxd(rxd),</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .ioaddr(i .data_rec .ioaddr(idd .rate_tx_d .clk(clk), .iocs(iocs) .ioaddr(idd .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(idd .rate_tx_dd .enable(end); trate_tx_dd .clk(clk), .rst(rst), .ioaddr(idd .rate_tx_dd .enable(end); receive_unit rr .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .iorw(iorw) .rda(rda), .iorw(iorw) .rdata_receita .rate_receita .iorw(iorw)</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec); baud_rate_genen .clk(clk), .rat(rst), .ioaddr(ioa .rate_tx_d .enable(ena); transmit_unit 4 .clk(clk), .rat(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(ioa .txd(txd), .rat(rst), .ioaddr(ioa .txd(txd), .rat(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(ioa .txd(txd), .rat(rst), .iors(rat), .iorw(iorw) .iorw(iorw) .torw(iorw) .tata_rece .enable(ena</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>
<pre>.iosrw(io .rda(rda) .tbr(tbr) .ioaddr(i .databus(.rate_tx_ .data_rec .ioaddr(i .data_rec .ioaddr(idd .rate_tx_d .clk(clk), .iocs(iocs) .ioaddr(idd .rate_tx_dd .enable(end); transmit_unit f .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .tbr(tbr), .ioaddr(idd .rate_tx_dd .enable(end); trate_tx_dd .clk(clk), .rst(rst), .ioaddr(idd .rate_tx_dd .enable(end); receive_unit rr .clk(clk), .rst(rst), .iocs(iocs) .iorw(iorw) .iorw(iorw) .rda(rda), .iorw(iorw) .rdata_receita .rate_receita .iorw(iorw)</pre>	<pre>srw), , , , , , , , , , , , , , , , , , ,</pre>

Bus Interface

```
`timescale 1ns / 1ps
 1
 2
    3
      // Company:
                          University of Wisconsin-Madison
                           Kai Zhao, John Roy
      // Engineer:
 4
 5
      11
      // Create Date:
                          2015 Sept 15
 6
                        Miniproject1
 7
      // Design Name:
      // Module Name:
                          bus_interface
 8
 9
      // Project Name:
10
      // Target Devices: Vertix 5
11
      // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
12
      // Description: bus interface to control which data to send and where to send it
13
      11
14
      // Dependencies:
                           none
      11
15
      // Revision:
16
      // Revision 0.01 - File Created
17
      // Additional Comments:
18
19
      11
     20
22 [module bus_interface(
      input iocs,
24
       input iosrw,
25
      input rda,
26
27
      input tbr,
      input [1:0] ioaddr.
      inout [7:0] databus,
28
29
      output reg [7:0] rate_tx_data,
30
31 );
      input [7:0] data_received
32
33 E
34 E
      always @(*) begin
         //if (ioaddr[1]) begin // if (ioaddr[1] == 1)
                                                            // processor writing to baud rate generator
         // rate_tx_data <= databus;</pre>
36
         //end
37
         rate_tx_data <= databus;
38
       end
39
40
      wire [7:0] databus_out;
       assign databus_out = (ioaddr[0]) ? {6'b000000, tbr, rda} : data_received; // processor reading/SPART writing
41
      assign databus = (!iosrw || ioaddr[1]) ? 8'hzz : databus_out;
42
                                                             // high impedence is required for processor writing/SPART reading
43
      //initial begin
44 📮
                                         // pseudocode for the assign statement
45
       // if (iosrw || ioaddr[1]) begin
46
      11
            databus = 8'bzzzzzzz;
      // end else if (ioaddr[0]) begin
47
      11
           databus = {6'b000000, tbr, rda};
48
      // end else begin
49
50
      11
           databus = data received;
51
       // end
52
       //end
53
54
55 endmodule
```

Bus Interface Test bench

1	timescale lns / lps
2	
3	// Company: University of Wisconsin-Madison
4	
	// Engineer: Kai Zhao, John Roy
5	
6	// Create Date: 2015 Sept 15
7	// Design Name: Miniproject1
8	// Module Name: bus_interface_tb
9	// Project Name:
10	// Target Devices: Vertix 5
11	// Tool versions: ModelSim SE 10.3c; Xilinx 14.7
12	// Description: test bench for testing bus_interface
13	11
14	// Dependencies: bus interface
15	
16	// Revision:
17	// Revision 0.01 - File Created
18	// Additional Comments:
19	<pre>// test case 1: test processor writing to baud rate generator low");</pre>
20	<pre>// test case 2: test processor writing a different value to baud rate generator high");</pre>
21	// test case 3: test processor reading the status register demo";
22	// test case 3. test processor realing one scatus register demoty, // test case 4. test processor writing a value to transmit controller");
23	
24	
25	<pre>module bus_interface_tb();</pre>
26	
27	reg iocs; // signals that are connected to the DUT
28	reg iosrw;
29	reg rda;
	reg tbr;
30	
30 31	reg [1:0] ioaddr;
	<pre>reg [1:0] ioaddr; wire [7:0] databus;</pre>
31	wire [7:0] databus;
31 32 33 34 35	
31 32 33 34 35 36 37	<pre>wire [7:0] databus; wire [7:0] rate_tx_data;</pre>
31 32 33 34 35 36 37 38	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg;</pre>
31 32 33 34 35 36 37 38 39	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT</pre>
31 32 33 34 35 36 37 38 39 40	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT icos(icos),</pre>
31 32 33 34 35 36 37 38 39 40 41	<pre>wire [7:0] databus; wire [7:0] databus; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iocs(iocsw),</pre>
31 32 33 34 35 36 37 38 39 40 41 42	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iosrw(iosrw), .rda(rda),</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .ioss(icos), .iosr(icos), .iosr(icos), .iosr(icos), .iotr(tbr),</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43 44	<pre>wire [7:0] databus; wire [7:0] data_recived; reg [7:0] data_received; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iocs(iocs(iocs), .iocs(iocs), .iocs(iocs(iocs), .iocs(iocs(iocs), .iocs(iocs(iocs), .iocs(iocs(iocs(iocs), .iocs(iocs(iocs(iocs(iocs(iocs(iocs(iocs(</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iosrw(iosrw), .rda(rda), .tbr(tbr), .ioadr(ioaddr), .databus(databus),</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .icos(icos), .icos(ic</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iosrw(iosrw), .rda(rda), .tbr(tbr), .ioadr(ioaddr), .databus(databus),</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .icos(icos), .icos(ic</pre>
31 32 33 34 35 36 37 38 37 38 39 40 41 42 43 44 45 46 47 48 9	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] data_received; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iocs(iocs), .iocs(iocs), .iocd(iocd), .ioaddr(ioaddr), .ioatabus(databus), .rate_tx_data(rate_tx_data), .data_received(data_received));</pre>
31 32 33 34 35 36 37 38 37 38 39 40 41 42 43 44 45 46 47 48 49	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] data_received; bus_interface DUT (// register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iocs(iocs), .iosrw(iorrw), .rda(tda), .tbr(tbr), .ioaddr(ioaddr), .databus(databus), .rate_tx_data(rate_tx_data), .data_received(data_received)</pre>
31 32 33 34 35 36 37 38 37 38 39 40 41 42 43 44 45 46 47 48 9	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] data_received; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iocs(iocs), .iocs(iocs), .iocd(iocd), .ioaddr(ioaddr), .ioatabus(databus), .rate_tx_data(rate_tx_data), .data_received(data_received));</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	<pre>wire [7:0] databus; wire [7:0] databus; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iocs(iocs), .iocs(iocs), .iocs(iocadh), .ioddr(iocadh), .ioaddr(iocadh), .databus(databus), .data_received(data_received)); initial begin</pre>
31 32 33 34 35 36 37 38 37 38 37 38 39 40 41 42 43 44 45 46 47 48 49 50	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] data_received; assign databus = databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .icos(icos),</pre>
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52	<pre>wire [7:0] databus; wire [7:0] databus; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iocs(iocs), .iocs(iocs), .iocs(iocs), .iocs(iocit), .iocidr(ioaddr), .databus(databus), .rate_tx_dat(rate_tx_data), .data_received(data_received)); initial begin Sdisplay("testing bus_interface"); iocs = 1;</pre>
31 32 33 34 35 36 37 38 40 41 42 44 45 44 45 55 52 53 54	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] data_received; bus_interface DUT (// instantiate the DUT iocs(icos), iocs(icos), iocs(icos), ioad(ricaddr), ioatd(icaddr), ioatabus(databus), rate_tx_data(rate_tx_data), data_received(data_received)); binitial begin Sdisplay("testing bus_interface"); icos = 1; icos = 1; icos = 1; icod = 2*500;</pre>
31 32 33 34 35 36 37 38 39 41 42 43 44 45 50 51 52 53 54 55	<pre>wire [7:0] databus; wire [7:0] databus; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .ioos(icos), .ioos(icos), .ioos(icos), .iosrw(icorw), .idada(a), .databus(databus), .databus(databus), .data_received(data_received)); initial begin \$display("testing bus_interface"); icos = 1; icorw = 1;</pre>
31 32 33 34 35 36 37 38 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 55 56	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] data_received; assign databus = databus_reg; bus_interface DUT (// instantiate the DUT</pre>
31 32 33 34 35 36 37 38 40 41 42 33 40 41 42 43 44 45 55 55 55 55 55 57	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] data_received; bus_interface DUT (// instantiate the DUT</pre>
31 32 33 34 35 36 37 38 40 41 42 43 44 45 51 52 53 54 55 55 57 58	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .icor(icor), .icorw(icorw), .rida(tda), .tbr(tbr), .icaddr(icadat), .tatabus(databus), .rate_tx_data(rate_tx_data), .data_received(dat_received)); initial begin \$display("testing bus_interface"); icors = 1; icors = 1;</pre>
31 32 33 34 35 36 37 38 40 41 42 44 44 44 44 44 55 55 55 55 55 55 55 55	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .icos(icos), .icorw(icorw), .rate(tabus), .rate_tx_data), .rate_tx_data(unte_tx_data), .data_received(data_received)); initial begin Sdisplay("testing bus_interface"); icorw = 1; icorw = 1; icorw = 2'b00; data_received = 8'h55; #5 Sdisplay("test processor writing to baud rate generator low"); icodd = 2'b10;</pre>
31 32 33 34 35 36 37 38 37 38 39 40 42 43 44 42 43 44 45 55 55 55 55 55 55 56 57 8 59 60	<pre>wire [7:0] databus; wire [7:0] rate_tx_data; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .iosr(iosr), .iosrw(iosrw), .ioard(ioadd), .idatabus(databus), .idatabus(databus), .idatabus(databus), .idata_received(data_received)); initial begin Sdisplay("testing bus_interface"); ioard = 1; ioarw = 1; ioarw = 1; ioarw = 2;b00; data_received = 8*h55; #5 Sdisplay("test processor writing to baud rate generator low"); ioaddr = 2*b1; databus_reg = 8*has;</pre>
31 32 33 34 35 36 37 39 41 42 43 44 45 51 55 55 55 55 55 55 55 55 55 55 55 55	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus_databus_reg; bus_interface DUT (// instantiate the DUT iocs(icos), iocarw(icorw), ida(rda), ida(rda), idatabus(databus), irate_tx_data), idatabus(databus), irate_tx_databus, irate_tx_databus, idatabus(databus), ioser=1; icos=1; </pre>
31 32 33 34 35 36 37 38 41 42 43 44 45 55 55 55 55 55 55 55 55 55 55 55	<pre>wire [7:0] databus; wire [7:0] databus; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT .icorw(icorw), .icorw(icorw</pre>
31 32 33 34 35 36 37 39 40 41 42 43 44 45 51 55 55 55 55 55 55 55 55 55 55 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 57 57 57 57 57 57 57 57 57 57 57 57 57	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus_edatabus_reg; bus_interface DUT (// instantiate the DUT icos(icos), ic</pre>
31 32 33 34 35 36 37 39 40 41 42 43 44 44 44 44 55 55 55 55 55 55 55 55 55	<pre>wire [1:0] databus; wire [1:0] data_received; reg [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus = databus_reg; bus_interface DUT (// instantiate the DUT</pre>
31 32 33 34 35 36 37 39 40 41 42 43 44 45 51 55 55 55 55 55 55 55 55 55 55 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 55 56 57 57 57 57 57 57 57 57 57 57 57 57 57	<pre>wire [7:0] databus; wire [7:0] data_received; reg [7:0] databus_reg; // register for handling inouts assign databus_edatabus_reg; bus_interface DUT (// instantiate the DUT icos(icos), ic</pre>

68	10 O O	#5
69		\$display("test processor writing a different value to baud rate generator high");
70		ioaddr = 2'b11;
71		databus reg = 8'hbc;
72		#5
73	卓	if (rate_tx_data == databus_reg) begin
74		\$display("\ttest passed");
75		end else begin
76		<pre>\$display("\ttest failed, expected: rate_tx_data == databus_reg, rate_tx_data = 8'h%h, databus_reg = 8'h%h", rate_tx_data, databus_reg);</pre>
77	-	end
78		
79		#5
80		\$display("test processor reading the status register demo");
81		databus_reg = 8'hzz;
82		tbr = 1;
83		rda = 0;
84		ioaddr = 2'b01;
85		iosrw = 1;
86		#5
87	F	if (databus == {6'b00000, tbr, rda}) begin
88		<pre>\$display("\ttest passed");</pre>
89		end else begin
90		<pre>\$display("\ttest failed, expected: databus == {6'b000000, tbr, rda}, databus = 8'h%h, {6'b000000, tbr, rda} = 8'h%h", databus, { 6'b000000, tbr, rda});</pre>
91	- 1	end
92		
93		#5
94		\$display("test processor writing a value to transmit controller");
95		iosrw = 0;
96		databus_reg = 8'hde;
97		ioaddr = 2'b00;
98		#5
99	¢ i	if (rate_tx_data == databus_reg) begin
100		<pre>\$display("\ttest passed");</pre>
101		end else begin
102		<pre>\$display("\ttest failed, expected: rate_tx_data == databus_reg, rate_tx_data = 8'h%h, databus_reg = 8'h%h", rate_tx_data, databus_reg);</pre>
103	F	end -
105		45
106		\$display("test processor reading a value from receive controller");
107		ioaddr = 2'b00;
108		iosrw = 1;
109		data received = 8'hf1;
110		databus reg = 8'hzz;
111		#5
112	白	if (databus == data_received) begin
113		<pre>\$display("\ttest passed");</pre>
114		end else begin
115		\$display("\ttest failed, expected: databus == data_received, rate_tx_data = 8'h%h, databus_reg = 8'h%h", databus, data_received);
116	F :	end
117		
118		\$stop;
119	L end	
120		
121	endmodu	le

Baud rate generator

```
`timescale 1ns / 1ps
    University of Wisconsin-Madison
      // Company:
  4
     // Engineer:
                       Kai Zhao, John Roy
  5
     // Create Date:
                       2015 Sept 15
  6
     // Design Name:
                       Miniproject1
  8
     // Module Name:
                        baud_rate_generator
     // Project Name:
  9
      // Target Devices: Vertix 5
     // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
     // Description: baud_rate_generator to take baud_rate input and control enable signal
     // Dependencies: none
 14
     11
     // Revision:
 16
      // Revision 0.01 - File Created
      // Additional Comments:
 18
 19
     // need to support baud rate 4800, 9600, 19200, 38400
            divsion for 4800 = 100000000/(16*4800) - 1 = 1300
     11
             divsion for 9600 = 100000000/(16*9600) - 1 = 650
 21
            divsion for 19200 = 100000000/(16*19200) - 1 = 325
             divsion for 38400 = 100000000/(16*38400) - 1 = 162
      11
 24
     25
 26
    pmodule baud_rate_generator(
                                                                 // inputs and outputs
 27
         input clk,
 28
         input rst,
 29
         input iocs,
         input [1:0] ioaddr,
          input [7:0] rate_tx_data,
 32
          output reg enable
 33
     L);
 34
 35
          reg [15:0] division_buffer;
 36
         reg [15:0] baud rate counter;
 38 🖂
          always @(posedge clk) begin
 39 E
             if (iocs) begin
                 if (rst) begin
                                                                   // if reset, then enable
 41
                     division buffer <= 1;
 42
                     baud_rate_counter <= 1;</pre>
 43
                  end if (ioaddr == 2'b11 && division_buffer[15:8] != rate_tx_data) begin
                    division_buffer[15:8] <= rate_tx_data;
 44
                                                                  // if high and high is not already set
                     baud_rate_counter <= {rate_tx_data, division_buffer[7:0]} - 1;</pre>
 45
                 end else if (ioaddr == 2'b10 && division_buffer[7:0] != rate_tx_data) begin
 46
 47
                     division_buffer[7:0] <= rate_tx_data;</pre>
                                                                  // if low and low is not already set
                     baud rate counter <= {division buffer[15:8], rate tx data} - 1;</pre>
 48
 49
                  end else if (baud_rate_counter == 0) begin
                                                                  // if 0. then reset
                     baud_rate_counter <= division_buffer;</pre>
                  end else begin
                                                                   // normally decrement
 52
                     baud rate counter <= baud rate counter - 1;</pre>
                 end
 54
              end
 55
          end
 56
 57 Ę
          always @ (posedge clk) begin
                                                                   // never enable if iocs is low
 58
              if (iocs) begin
 59
                 if (baud_rate_counter == 0) begin
                                                                   // enable when counter reaches 0
                     enable <= 1;
 60
 61
                  end else begin
 62
                    enable <= 0;
 63
                  end
             end
 64
 65
          end
 66
 67
68 endmodule
```

Baud rate generator test bench

	escale ins / ips
	//////////////////////////////////////
4 // Er	ngineer: Rai Zhao, John Roy
5 // 6 // Cr	reate Date: 2015 Sept 15
	ssign Name: Miniproject1
	odule Name: baud_rate_generator_tb
	roject Name: Arget Devices: Vertix 5
1 // To	bol versions: ModelSim SE 10.3c; Xilinx 14.7
2 // De 3 //	escription: test bench for testing baud_rate_generator
	apendencies: baud_rate_generator
5 //	
	evision: evision 0.01 - File Created
8 // Ac	dditional Comments:
9 //	test case 1: test first cycle of baud rate by setting only the low byte test case 2: test second cycle of baud rate by setting only the low byte
1 //	test case 3: test first cycle of baud rate by setting only the high byte
2 // 3 //	test case 4: test second cycle of baud rate by setting only the high byte test case 5: test first cycle of baud rate by setting baud rate = 16'ha531
4 11	test case 6: test second cycle of baud rate by setting baud rate = 16 has31
5 //	test case 7: test baud rate still works after changing the address
6 //	test case 8: test that enable is never high when iocs is low
8	
9 modul	<pre>le baud_rate_generator_tb();</pre>
1 1	reg clk; // signals that are connected to the DUT
	reg rst;
	reg iocs; reg [1:0] ioaddr;
5 1	reg [7:0] rate_tx_data;
6	rire enable;
8 E k	<pre>aud_rate_generator DUT (// instantiate the DUT .clk(clk),</pre>
0	.rst (rst),
1	.iocs(iocs),
2	.ioaddr(ioaddr), .rate tx data(rate tx data),
4	.enable(enable)
5 L) 6	
	initial begin // initialize all variables
7 E i	<pre>\$display("testing baud_rate_generator");</pre>
7 🗗 i 8 9	<pre>\$display("testing baud_rate_generator"); clk = 0;</pre>
7 E i	<pre>\$display("testing baud_rate_generator");</pre>
7 日 i 8 9 0 1 2	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; ioaddr = 2'b00;</pre>
7 E i 8 9 0 1	<pre>\$display("testing baud_rate_generator"); clk = 0; rat = 1; iccs = 1;</pre>
7 🗗 i 8 9 0 1 2 3 4 5	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; iocaddr = 2'b00; rste_tx_data = 0; @(posedge clk);</pre>
7 j i 8 9 0 1 2 3 4 5 6	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; icaddr = 2'b00; rste_tx_data = 0; @(posedge clk); rst = 0;</pre>
7 🗗 i 8 9 0 1 2 3 4 5	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; iocaddr = 2'b00; rste_tx_data = 0; @(posedge clk);</pre>
7 日 3 8 9 9 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 9	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; iccs = 1; icaddr = 2'b00; rate_tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte</pre>
7 E i 8 9 9 0 1 1 2 3 4 4 5 5 6 7 7 9 0	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; ioaddr = 2"b00; rate_tx_data = 0; @(posedge clk); rate_tx_data = 0; %(posedge clk); %(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low");</pre>
7 日 3 8 9 9 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 9	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; icaddr = 2'b00; rate_tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte</pre>
7 日 i 8 9 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 9 9 0 0 1 1 2 2 3 3 4 3 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; ioaddr = 2"b00; rate_tx_data = 0; @(posedge clk); rate_tx_data = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2"b11; rate_tx_data = 0; @(posedge clk);</pre>
7 D i i i i i i i i i i i i i i i i i i	<pre>&display("testing baud_rate_generator"); clk = 0; rst = 1; locs = 1; locdst = 2'b00; rate_tx_data = 0; @ (posedge clk); rst = 0; @ (posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low"); loaddr = 2'b11; rate_tx_data = 0; @ (posedge clk); ieaddr = 2'b10;</pre>
7 D i i i i i i i i i i i i i i i i i i	<pre>&display("testing baud_rate_generator"); clk = 0; rst = 1; icods = 1; icods = 2'b00; rate_tx_data = 0; @ (posedge clk); rst = 0; @ (posedge clk); /// test first cycle of baud rate by setting only the low byte &display("test first cycle of processor only writing to baud rate generator low"); icoddr = 2'b11; rate_tx_data = 0; @ (posedge clk); icoddr = 2'b11; rate_tx_data = 0; @ (posedge clk); icoddr = 2'b11; rate_tx_data = 3; @ (posedge clk); // wait 1 cycle for baud rate to load</pre>
7 D i i i i i i i i i i i i i i i i i i	<pre>%display("testing baud_rate_generator"); clk = 0; rst = 1; locs = 1; locd = 2'b00; rate_tx_data = 0; % (posedge clk); rst = 0; % (posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte %display("test first cycle of processor only writing to baud rate generator low"); loadtr = 2'b11; rate_tx_data = 0; % (posedge clk); licadtr = 2'b10; rate_tx_data = 3; % (posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded</pre>
7 P 3 8 9 9 00 1 1 2 2 3 3 4 4 5 5 6 6 7 7 9 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 9 8 8	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; iccs = 1; ioaddr = 2'b00; rst=tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b11; rst=tx_data = 0; @(posedge clk); ioaddr = 2'b10; rst=tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk);</pre>
7 0 5 8 9 0 0 1 2 2 3 3 4 5 5 6 6 7 0 9 0 1 1 2 3 3 4 4 5 5 6 6 7 0 9 0 1 1 2 8 3 4 4 5 5 7 6 9 9 0 0 1 1 8 9 9 9 0 0 1 1 8 9 9 9 0 1 8 9 9 9 0 1 8 9 9 9 0 1 8 9 9 9 0 9 0 1 8 9 9 9 0 9 0 9 0 9 0 9 0 1 8 9 9 9 0 9 0 9 0 9 0 9 0 9 0 9 0	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; iocd = 2'b00; rate_tx_data = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b11; rate_tx_data = 0; @(posedge clk); ioaddr = 2'b10; rate_tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin \$display("\test failed, enable should have been kept low for 3 cycles");</pre>
7 C S S S S S S S S S S S S S S S S S S	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; iccs = 1; ioaddr = 2'b00; rst=tx_data = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b11; rst=tx_data = 0; @(posedge clk); ioaddr = 2'b10; rst=tx_data = 0; @(posedge clk); ioaddr = 2'b10; rst=tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin \$display("\ttest failed, enable should have been kept low for 3 cycles"); end</pre>
7 0 5 8 9 0 0 1 2 2 3 3 4 5 5 6 6 7 0 9 0 1 1 2 3 3 4 4 5 5 6 6 7 0 9 0 1 1 2 8 3 4 4 5 5 7 6 9 9 0 0 1 1 8 9 9 9 0 0 1 1 8 9 9 9 0 1 8 9 9 9 0 1 8 9 9 9 0 1 8 9 9 9 0 9 0 1 8 9 9 9 0 9 0 9 0 9 0 9 0 1 8 9 9 9 0 9 0 9 0 9 0 9 0 9 0 9 0	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; iccds = 2'b00; rste_tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low"); icaddr = 2'b11; rste_tx_data = 0; @(posedge clk); iiaddr = 2'b10; rste_tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (cosedge clk); if (cosedge clk); if (cosedge clk); d(posedge clk); if (cosedge clk); if (</pre>
7 0 3 4 8 9 0 0 1 2 3 4 5 5 6 6 7 9 10 1 2 3 4 5 5 6 6 7 1 9 10 1 2 3 4 5 5 6 7 1 9 0 1 2 3 4 5 5 6 7 9 0 1 2 3 1 4 5 6 7 7 8 9 0 1 2 3 1 4 5 6 7 8 9 0 1 2 3 1 4 5 6 7 7 8 9 0 1 2 3 1 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 7 8 9 0 1 2 3 4 5 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8	<pre>Sdisplay("testing baud_rate_generator"); clk = 0; rst = 1; icads = 2'b00; rate_tx_data = 0; @ (posedge clk); rst = 0; @ (posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); icaddr = 2'b11; rate_tx_data = 0; @ (posedge clk); icaddr = 2'b10; rate_tx_data = 0; @ (posedge clk); icaddr = 2'b10; rate_tx_data = 3; @ (posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @ (posedge clk); if (enable) begin Sdisplay("'ttest failed, enable should have been kept low for 3 cycles"); end end @ (posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin</pre>
7 0 3 4 8 9 0 0 1 2 2 3 3 4 5 5 6 7 7 0 9 0 0 1 2 2 3 4 4 5 5 6 7 9 0 0 1 2 2 3 4 4 5 5 6 7 9 0 0 1 2 2 3 4 4 5 5 6 9 0 0 1 2 2 3 4 4 5 5 6 9 0 0 1 2 2 3 4 4 5 5 6 9 0 0 1 1 2 2 3 4 4 5 5 6 9 0 0 1 1 2 2 3 4 4 5 5 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	<pre>%display("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; iocdit = 2'b00; rste_tx_data = 0; % (posedge clk); rst = 0; % (posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte %display("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b11; rate_tx_data = 0; % (posedge clk); iiaddr = 2'b10; rste_tx_data = 3; % (posedge clk); iiaddr = 2'b10; rste_tx_data = 3; % (posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 2 cycles since 3 is loaded % (posedge clk); if (enable) begin</pre>
7 0 3 4 8 9 0 0 1 2 3 4 5 5 6 6 7 9 10 1 2 3 4 5 5 6 6 7 1 9 10 1 2 3 4 5 5 6 7 1 9 0 1 2 3 4 5 5 6 7 9 0 1 2 3 1 4 5 6 7 7 8 9 0 1 2 3 1 4 5 6 7 8 9 0 1 2 3 1 4 5 6 7 7 8 9 0 1 2 3 1 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 7 8 9 0 1 2 3 4 5 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8	<pre>Sdisplay("testing baud_rate_generator"); clk = 0; rst = 1; icads = 2'b00; rate_tx_data = 0; @ (posedge clk); rst = 0; @ (posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); icaddr = 2'b11; rate_tx_data = 0; @ (posedge clk); icaddr = 2'b10; rate_tx_data = 0; @ (posedge clk); icaddr = 2'b10; rate_tx_data = 3; @ (posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @ (posedge clk); if (enable) begin Sdisplay("'ttest failed, enable should have been kept low for 3 cycles"); end end @ (posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin</pre>
7 0 1 1 8 9 0 1 1 2 3 4 5 6 6 7 9 0 1 1 2 3 4 5 5 6 7 8 9 0 1 2 3 4 5 7 8 9 0 1 2 3 4 5 7 8 9 0 1 2 3 4 5 7 7 8 9 0 1 2 3 4 5 7 7 8 9 0 1 2 3 4 5 7 7 8 9 0 1 7 8 9 0 1	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; iccs = 1; icaddr = 2'b00; rste_tx_data = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low"); icaddr = 2'b11; rste_tx_data = 0; @(posedge clk); icaddr = 2'b10; rste_tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin { \$display("\test failed, enable should have been kept low for 3 cycles"); end end @(posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin { \$display("\test passed"); end else begin { \$dis</pre>
7 0 3 4 8 9 0 0 1 2 3 4 5 5 6 7 9 0 1 2 3 4 5 5 6 7 8 9 9 0 1 2 2 3 4 5 5 6 7 8 9 0 1 2 2 3 4 5 5 7 8 9 9 0 1 2 2 3 4 5 5 7 8 9 9 0 1 2 2 3 4 5 5 7 8 9 9 0 1 2 2 3 4 5 6 7 8 9 9 0 1 2 2 3 4 5 6 7 8 9 9 0 1 2 3 4 5 6 6 7 8 9 9 0 1 2 3 4 5 6 6 7 8 9 9 0 1 2 3 4 5 7 8 9 9 0 1 2 3 4 5 7 8 9 9 0 1 2 3 4 5 7 8 9 9 0 1 2 7 8 9 9 9 1 7 7 8 9 9 7 7 8 9 9 7 7 8 9 9 7 7 8 9 9 7 7 8 9 9 7 7 7 8 9 9 7 7 7 8 9 9 7 7 7 8 9 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	<pre>\$display("testing baud_rate_generator"); clk = 0; rst = 1; iccs = 1; iccs = 1; ioaddr = 2'b00; rste_tx_data = 0; f(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte \$display("test first cycle of processor only writing to baud rate generator low"); icaddr = 2'b11; rste_tx_data = 0; f(posedge clk); icaddr = 2'b10; rste_tx_data = 0; f(posedge clk); icaddr = 2'b10; rste_tx_data = 3; f(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded f(posedge clk); if (enable) begin { Sdisplay("\test failed, enable should have been kept low for 3 cycles"); end end f(enable) begin { Sdisplay("\test passed"); end else begin { Sdisplay("\test failed, enable should have been pulsed high on the 4th cycle"); end } </pre>
7 0 1 1 8 9 0 1 1 2 3 4 5 6 6 7 9 0 1 1 2 3 4 5 5 6 7 8 9 0 1 2 3 4 5 7 8 9 0 1 2 3 4 5 7 8 9 0 1 2 3 4 5 7 7 8 9 0 1 2 3 4 5 7 7 8 9 0 1 2 3 4 5 7 7 8 9 0 1 7 8 9 0 1	<pre>%display("testing baud_rate_generator"); clk = 0; rst = 1; icadsr = 2'b00; rste_tx_data = 0; % (posedge clk); rst = 0; % (posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte % display("test first cycle of processor only writing to baud rate generator low"); icaddr = 2'b11; rste_tx_data = 0; % (posedge clk); icaddr = 2'b10; rste_tx_data = 0; % (posedge clk); icaddr = 2'b10; rste_tx_data = 3; % (posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded % (posedge clk); if (enable) begin % display("\ttest failed, enable should have been kept low for 3 cycles"); end end % (posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin % display("\ttest passed"); end else begin % display("\ttest failed, enable should have been pulsed high on the 4th cycle");</pre>
7 0 0 1 8 9 00 1 2 3 4 5 6 7 7 9 00 1 2 3 4 5 6 7 8 9 00 1 2 3 4 5 6 7 7 8 9 00 1 2 3 4 5 6 7 8 9 00 1 2 3 4 5 5 6 7 8 9 00 1 2 3 4 5 7 8 9 0 1 2 3 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7	<pre>Sdisplay("testing baud_rate_generator"); clk = 0; rst = 1; ioss = 1; iaddr = 2'b00; rate_tx_data = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b10; rate_tx_data = 0; @(posedge clk); iteaddr = 2'b10; rate_tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end end @(posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("test second cycle of processor only writing to baud rate generator low"); repeat (3) begin</pre>
7 0 3 4 8 9 0 0 1 1 2 3 4 5 6 6 7 9 0 1 2 3 4 5 6 6 7 9 0 1 2 3 4 4 5 6 6 7 9 0 1 2 3 4 4 5 6 6 7 9 0 1 2 3 4 4 5 6 6 7 1 2 3 4 4 5 6 6 7 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2	<pre>Sdisplay("testing baud_rate_generator"); clk = 0; rst = 1; iocs = 1; iocdst = 2'b00; rate_tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); ioaddt = 2'b10; rate_tx_data = 0; @(posedge clk); isdatd = 2'b10; rate_tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin // wait 3 cycles since 3 is loaded @(posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end end @(posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("test second cycle of processor only writing to baud rate generator low"); repeat (3) begin @(posedge clk);</pre>
7 0 0 1 8 9 00 1 2 3 4 5 6 7 7 9 00 1 2 3 4 5 6 7 8 9 00 1 2 3 4 5 6 7 7 8 9 00 1 2 3 4 5 6 7 8 9 00 1 2 3 4 5 5 6 7 8 9 00 1 2 3 4 5 7 8 9 0 1 2 3 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7	<pre>Sdisplay("testing baud_rate_generator"); clk = 0; rst = 1; ioss = 1; iaddr = 2'b00; rate_tx_data = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b10; rate_tx_data = 0; @(posedge clk); iteaddr = 2'b10; rate_tx_data = 3; @(posedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end end @(posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (enable) begin Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("test second cycle of processor only writing to baud rate generator low"); repeat (3) begin</pre>
7 0 0 1 8 9 0 1 2 3 4 5 6 7 9 0 1 2 3 4 5 6 7 8 9 0 1 3 4 5 6 7 8 9 0 1 3 5 7 8 9 1 8 9 0 1 3 7 8 9 0 1 3 8 9 0	<pre>Salisplay("testing baud_rate_generator"); clk = 0; rst = 1; iosds = 2'b00; rate_tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b10; rate_tx_data = 0; @(posedge clk); iioaddr = 2'b10; rate_tx_data = 0; @(posedge clk); iii(casle) begin // wait 1 cycle for baud rate to load repest (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end sdisplay("ttest passed"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("test passed"); end</pre>
7 E 3 8 9 00 1 2 3 4 5 6 7 9 00 1 2 3 4 5 6 7 9 00 1 2 3 4 5 6 7 8 9 00 1 2 3 4 5 7 8 9 00 1 2 3 7 8 9 00 1 2 3 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8	<pre>Sdisplay("testing baud_rate_generator"); clk = 0; rst = 1; icods = 2:b00; rste_tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); rst = 0; @(posedge clk); rate tx_data = 0; @(posedge clk); icoddr = 2:b10; rate tx_data = 0; @(posedge clk); icoddr = 2:b10; rate tx_data = 3; @(posedge clk); if (cnable) begin & display("\test failed, enable should have been kept low for 3 cycles"); end @(posedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (cnable) begin & display("\test failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud_rate by setting only the low byte Sdisplay("\test failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud_rate by setting only the low byte Sdisplay("\test failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud_rate by setting only the low byte Sdisplay("\test failed, enable should have been pulsed high on the 4th cycle"); end @(posedge clk); if (enable) begin % display("\test failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of processor only writing to baud_rate generator low"); repeat (6) begin % display("\test failed, enable should have been kept low for 3 cycles"); end end</pre>
7 0 0 1 8 9 0 1 2 3 4 5 6 7 9 0 1 2 3 4 5 6 7 8 9 0 1 3 4 5 6 7 8 9 0 1 3 5 7 8 9 1 8 9 0 1 3 7 8 9 0 1 3 8 9 0	<pre>Salisplay("testing baud_rate_generator"); clk = 0; rst = 1; iosds = 2'b00; rate_tx_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'b10; rate_tx_data = 0; @(posedge clk); iioaddr = 2'b10; rate_tx_data = 0; @(posedge clk); iii(casle) begin // wait 1 cycle for baud rate to load repest (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (enable) begin Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end sdisplay("ttest passed"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("ttest failed, enable should have been kept low for 3 cycles"); end // test case 2: test second cycle of baud rate by setting only the low byte Sdisplay("test passed"); end</pre>
7 8 9 00 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 9 0 1 1 2 3 4 5 6 7 8 8 9 0 1 1 2 3 4 5 6 7 8 8 9 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	<pre>Sdisplay("testing baud_rate_generator"); clk = 0; rst = 1; icodsr = 2*b00; rate_ts_data = 0; @(posedge clk); rst = 0; @(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte Sdisplay("test first cycle of processor only writing to baud rate generator low"); icoddr = 2*b10; rate_ts_data = 0; @(posedge clk); if (cosedge clk); // wait 1 cycle for baud rate to load repeat (3) begin // wait 3 cycles since 3 is loaded @(posedge clk); if (cosedge clk); if (cosedge clk); if (cosedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (cosedge clk); // wait another for baud_rate_generator to react to baud_rate_counter reaching 0 if (staplay("\ttest passed"); end end (// test case 2: test second cycle of baud rate by setting only the low byte fdisplay("\ttest failed, enable should have been pulsed high on the 4th cycle"); end // test case 2: test second cycle of baud rate by setting only the low byte fdisplay("test failed, enable should have been kept low for 3 cycles"); end [@ (posedge clk); if (conble) begin [@ (posedge clk); if (conble) beg</pre>
7 8 9 00 1 2 3 4 5 6 7 8 9 00 1 1 2 3 4 5 6 7 8 9 00 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	<pre>display("test second cycle of baud rate by setting only the low byte fdisplay("test failed, enable should have been pulsed high on the fit cycle"); end d(posedge clk); // test case 1: test first cycle of baud rate by setting only the low byte fdisplay("test first cycle of processor only writing to baud rate generator low"); ioaddr = 2'bli; rate_tx_data = 0; d(posedge clk); it case = 2'bli; rate_tx_data = 3; d(posedge clk); if (enable) begin</pre>

96	<pre>// test case 3: test first cycle of baud rate by setting only the high byte</pre>			
97	<pre>(posedge clk);</pre>			
98	\$display("test first cycle of processor only writing to baud rate generator high");			
99	ioaddr = 2'b11;			
LOO	<pre>rate_tx_data = 8'h20;</pre>			
101	<pre>@(posedge clk);</pre>			
102	ioaddr = 2'b10;			
103	rate_tx_data = 0;			
104 105 E	<pre>@(posedge clk); repeat (8192) begin // equivalent to 16'h2000</pre>			
105 🖻 106				
107 F	0(posedge clk); if (enable) begin			
108	\$display("\ttest failed, enable should have been kept low for 8192 cycles");			
109 -	end			
110 -	end			
111	0 (posedge clk);			
112 🛱	if (enable) begin			
113	<pre>\$display("\ttest passed");</pre>			
114	end else begin			
115	<pre>\$display("\ttest failed, enable should have been pulsed high on the 8193rd cycle")</pre>			
116 -	end			
117 118				
118	// test case 4: test second cycle of baud rate by setting only the high byte			
120	<pre>\$display("test second cycle of processor only writing to baud rate generator high");</pre>			
121 日	repeat (8192) begin			
122	<pre>@(posedge clk);</pre>			
123 🛱	if (enable) begin			
124	<pre>\$display("\ttest failed, enable should have been kept low for 8192 cycles");</pre>			
125 -	end			
126 -	end			
127	0(posedge clk);			
128 🛱	if (enable) begin			
129	<pre>\$display("\ttest passed");</pre>			
130 131	end else begin			
132 -	<pre>\$display("\ttest failed, enable should have been pulsed high"); end</pre>			
135	<pre>// test case 5: test first cycle of baud rate by setting baud rate = 16'ha531</pre>			
136	0 (posedge clk);			
137	<pre>\$display("test first cycle of processor writing 8'ha531 to baud rate generator");</pre>			
138	<pre>ioaddr = 2'b11;</pre>			
139	<pre>rate_tx_data = 8'ha5;</pre>			
140 141	<pre>@(posedge clk); ioaddr = 2'b10;</pre>			
141				
192				
142	rate_tx_data = 8'h31;			
	0 (posedge clk);			
144	<pre>@ (posedge clk); repeat (42289) begin // equivalent to 16'ha531</pre>			
144 🖻 145	0 (posedge clk);			
144 0 145 1 146 0	<pre>@ (posedge clk); repeat (42289) begin // equivalent to 16'ha531 @ (posedge clk);</pre>			
144 E 145 146 E 147	<pre>@(posedge clk); repeat (4228) begin // equivalent to 16'ha531 @(posedge clk); if (enable) begin</pre>			
144 E 145 146 E 147 148 - 149 -	<pre>@(posedge clk); repeat (42289) begin // equivalent to 16'ha531 @(posedge clk); if (enable) begin \$display("\ttest failed, enable should have been kept low for 42289 cycles"); end end</pre>			
144 🕀 145 – 146 🕀 147 – 148 – 149 – 150	<pre>% (posedge clk); repeat (42289) begin // equivalent to 16'ha531</pre>			
144 🖸 145 – 146 🛱 147 – 148 – 149 – 150 –	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin</pre>			
144 E 145 E 146 E 147 - 148 - 149 - 150 E 151 E 152 E	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed");</pre>			
144 E 145 - 146 E 147 - 148 - 149 - 150 - 151 E 152 - 153 -	<pre>@ (posedge clk); repeat (42289) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin sdisplay("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end else begin</pre>			
144 🖯 145 - 146 🖯 147 - 148 - 150 - 151 🖯 152 - 153 - 154 -	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end else begin \$ display("\ttest failed, enable should have been pulsed high");</pre>			
144 ⊟ 145 □ 146 □ 147 □ 149 □ 150 □ 151 ⊟ 152 □ 153 □ 154 □ 155 □	<pre>@ (posedge clk); repeat (42289) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin sdisplay("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end else begin</pre>			
144 ⊟ 145 146 ⊟ 147 148 - 149 - 150 151 ⊟ 152 153 154 155 - 156	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end else begin \$ display("\ttest failed, enable should have been pulsed high");</pre>			
144 日 145 日 146 日 147 - 149 - 150 日 151 日 152 153 - 155 - 156 157	<pre>@ (posedge clk); repeat (#228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end else begin \$ display("\ttest failed, enable should have been pulsed high"); end</pre>			
144 E 145 E 147 1 148 - 149 - 150 1 151 E 152 1 153 1 154 1 155 - 156 1 157 1 158	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ \$display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ \$display("\ttest passed"); end else begin \$ \$display("\ttest failed, enable should have been pulsed high"); end</pre>			
144 E 145 E 146 E 147 - 148 - 149 - 150 - 152 - 153 E 154 - 155 - 155 - 156 - 158 - 159 -	<pre>@ (posedge clk); repeat (#228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end else begin \$ display("\ttest failed, enable should have been pulsed high"); end</pre>			
144	<pre>@ (posedge clk); repeat (#228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\test failed, enable should have been kept low for 42289 cycles"); end end @(posedge clk); if (enable) begin \$ display("\test passed"); end else begin \$ display("\test failed, enable should have been pulsed high"); end // test case 6: test second cycle of baud rate by setting baud rate = 16'ha531 \$ display("test second cycle of processor writing 8'ha531 to baud rate generator");</pre>			
144 E 145 E 147 - 149 - 150 E 151 E 152 - 153 - 154 - 155 - 156 E 157 - 158 E 159 E 160 E 161	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end else begin \$ display("\ttest failed, enable should have been pulsed high"); end // test case 6: test second cycle of baud rate by setting baud rate = 16'ha531 \$ display("test second cycle of processor writing 8'ha531 to baud rate generator"); repeat (42289) begin</pre>			
144 E 146 E 147 - 147 - 149 - 150 - 152 - 153 - 154 - 155 - 156 - 157 - 158 - 161 - 161 - 162 - 161 - 162 - 163 - 164 - 16	<pre>@ (posedge clk); repeat (42289) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin</pre>			
144	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin</pre>			
144 145 146 147 147 148 147 148 147 148 147 150 151 151 155 155 155 155 156	<pre>@ (posedge clk); repeat (#228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end % (posedge clk); if (enable) begin \$ display("\ttest passed"); end end % display("\ttest failed, enable should have been pulsed high"); end // test case 6: test second cycle of baud rate by setting baud rate = 16'ha531 \$ display("test second cycle of baud rate by setting baud rate = 16'ha531 \$ display("test second cycle of processor writing 8'ha531 to baud rate generator"); repeat (42289) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end</pre>			
144	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @(posedge clk); if (enable) begin \$display("\ttest passed"); end // test case 6: test second cycle of baud rate by setting baud rate = 16'ha531 \$display("test second cycle of processor writing 8'ha531 to baud rate generator"); repeat (4228) begin @(posedge clk); if (enable) begin @(posedge clk); if (enable) begin %display("\ttest failed, enable should have been kept low for 42289 cycles"); end end end @(posedge clk);</pre>			
144	<pre>% (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin</pre>			
144	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\test failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\test passed"); end // test case 6: test second cycle of baud rate by setting baud rate = 16'ha531 \$ display("\test second cycle of processor writing 8'ha531 to baud rate generator"); repeat (42289) begin @ (posedge clk); if (enable) begin \$ display("\test failed, enable should have been kept low for 42289 cycles"); end @ (posedge clk); if (enable) begin \$ display("\test failed, enable should have been kept low for 42289 cycles"); end @ (posedge clk); if (enable) begin \$ display("\test passed");</pre>			
143 144 145 146 147 148 149 150 151 153 154 155 153 154 155 157 158 159 160 161 161 162 163 164 165 165 165 165 165 165 165 165 165 165	<pre>@ (posedge clk); repeat (4228) begin // equivalent to 16'ha531 @ (posedge clk); if (enable) begin \$ display("\ttest failed, enable should have been kept low for 42289 cycles"); end end @ (posedge clk); if (enable) begin \$ display("\ttest passed"); end \$ display("\ttest failed, enable should have been pulsed high"); end // test case 6: test second cycle of baud rate by setting baud rate = 16'ha531 \$ display("test second cycle of processor writing 8'ha531 to baud rate generator"); repeat (42289) begin @ display("ttest failed, enable should have been kept low for 42289 cycles"); end end @ diposedge clk); if (enable) begin @ diposedge clk); if (enable) begin</pre>			

74	<pre>// test case 7: test baud rate</pre>	e still works after changing the address
75	\$display("test baud rate generator	still workes even after changing address");
76	ioaddr = 2'b00;	
77	rate tx data = 8'h76;	
78	repeat (42289) begin	
79	@(posedge clk);	
80 🗄	if (enable) begin	
81	\$display("\ttest failed, en	nable should have been kept low for 42289 cycles");
82 -	end	
83 -	end	
84	<pre>@(posedge clk);</pre>	
85 🛱	if (enable) begin	
86	<pre>\$display("\ttest passed");</pre>	
87	end else begin	
88	\$display("\ttest failed, enable	e should have been pulsed high");
89 -	end	
90		
91		
92	<pre>// test case 8: test that enable</pre>	ble is never high when iocs is low
93	\$display("test that enable is never	r high when iocs is low");
94	iocs = 0;	
95 🛱	repeat (65536) begin // equivale	ent to 16'hffff + 1
96	0 (posedge clk) ;	
97 白	if (enable) begin	
98	\$display("\ttest failed, en	nable should should not go high if iocs is low");
99	\$stop;	
00 -	end	
01 -	end	
02	<pre>\$display("\ttest passed");</pre>	
03		
04	\$stop;	
05 L	end	
06		
07	always	
08	#2 clk = !clk;	
09		
10 €	endmodule	

Transmit unit

1	`ti	mescale 1ns / 1p	38
2	戶///	111111111111111	(1)))))))))))))))))))))))))))))))))))))
3	11	Company:	University of Wisconsin-Madison
4	11	Engineer:	Kai Zhao, John Roy
5	11		
6	11	Create Date:	2015 Sept 15
7		Design Name:	Miniproject1
8		Module Name:	transmit_unit
9		Project Name:	
10		Target Devices:	
11			ModelSim SE 10.3c; Xilinx 14.7
12	10110	Description:	transmit_unit to transmit data from SPART to RS232
13	11		
14		Dependencies:	none
15 16	11	Revision:	
17		Revision: Revision 0.01 -	File granted
18		Additional Comme	
19	11	Additional comme	ints:
20		,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	
21	,,,,	1111111111111111	
22	Emod	ule transmit uni	it (// inputs and outputs
23	1	input clk,	
24		input rst,	
25		input iocs,	
26		input iorw,	
27		output reg tbr,	
28		input [1:0] ioa	
29		output txd,	
30		input [7:0] rat	:e_tx_data,
31		input enable	
32	L);		
33			
34		reg [9:0] shift	;_reg;
35		reg [3:0] bit_c	
36		reg load, shift	
37		localparam numb	perOfBitsPerPacket = 9; // 1 start bit, 1 stop bit, and 8 bits of
34		reg [9:0] shif	t_reg;
35		reg [3:0] bit_	ent;
36		reg load, shif	
37		localparam num	berOfBitsPerPacket = 9; // 1 start bit, 1 stop bit, and 8 bits of data, -1 because checked after shifting
38			
39		<pre>// send 1 bit</pre>	of txd at a time
40		assign txd = s	hift_reg[0];
41			
42	F	<pre>// handle tx_b</pre>	wiffer, shift_reg, and bit_cnt based on the FSM output
43			ready to tx , send 1 low start bit, then 8 data bits, then 1 high stop bit.
44	L	// Otherwise,	hold the output high
45	F	always @ (posed	ge clk) begin
46	白	if (load)	begin
47			rt at 1, because this takes a clock cycle as well
48			<pre>reg <= {1'b1, rate_tx_data, 1'b0};</pre>
49			<pre>t <= numberOfBitsPerPacket;</pre>
50			f (set_tbr) begin
51			tx_data_out[0] to not start the receiver
52			reg[0] <= 1'b1;
53		end else i	f (shift) begin
54		bit_cn	<pre>st <= bit_cnt - 1;</pre>
55		shift_	<pre>reg <= {shift_reg[0], shift_reg[9:1]};</pre>
	-	end	
56 57			

```
59
          localparam IDLE = 1'b0, TRANS = 1'b1;
60
          reg state, nxt_state;
61
62
          // handle start transitions of the FSM
63 E
          always @(posedge clk) begin
64 🗄
              if(rst) begin
65
                  state <= IDLE;</pre>
66
                  tbr <= 0;
67
              end else begin
68
                  state <= nxt_state;</pre>
69
                  tbr <= set_tbr;</pre>
              end
71
          end
          always @(*) begin
73 E
74
              // set defaults
 75
              nxt state = IDLE;
76
              load = 0;
77
              shift = 0;
              set_tbr = 0;
78
79 🗄
              case (state)
80 🖻
                  IDLE: begin
                      // if begin, then wait for trmt signal
81
82 🛱
                      if (ioaddr == 2'b00 && !iorw) begin // if write signal, then load variables
83
                          load = 1;
84
                         nxt_state = TRANS;
85
                      end else begin
86
                         set_tbr = 1;
87
                      end
88
                  end
89 🗄
                  TRANS: begin
90
                      // wait for baud count to increase until it is time to shift
91
                      nxt state = TRANS;
92
                      // if enable, then shift
93 🛱
                      if (enable) begin
                          shift = 1;
94
95
                          // if bit_cnt == 0, then done and return to IDLE
                          if (bit_cnt == 0) begin
96 🗄
                              nxt_state = IDLE;
97
98
                          end
99
                      end
100
                  end
101 🛱
                  default: begin
102 🛱
                      // default state for safety
103
                      // return to IDLE
104
                  end
105
              endcase
106
     L
          end
107
108
     endmodule
```

Transmit unit test bench

```
`timescale 1ns / 1ps
    2
                      University of Wisconsin-Madison
 3
     // Company:
 4
     // Engineer:
                      Kai Zhao, John Roy
 5
     11
     // Create Date:
                      2015 Sept 15
 6
     // Design Name:
                      Miniproject1
     // Module Name:
 8
                      transmit_unit_tb
 9
     // Project Name:
     // Target Devices: Vertix 5
     // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
 12
     // Description: test bench for testing transmit unit
 13
     // Dependencies: transmit_unit
 14
     11
 16
     // Revision:
     // Revision 0.01 - File Created
 18
     // Additional Comments:
           test case 1: test transmitting 8'haa
 19
     11
            test case 2: test transmitting 8'b0011_1001
 20
     11
     21
 22
23
     module transmit unit tb();
 24
                             // signals that are connected to the DUT
25
        reg clk;
 26
        reg rst;
 27
         reg iocs;
 28
        reg iorw;
 29
         wire tbr;
        reg [1:0] ioaddr;
 30
 31
         wire txd:
         reg [7:0] rate_tx_data;
33
         reg enable;
 35 🛱
         transmit unit DUT(
                                  // instantiate the DUT
            .clk(clk),
 36
 37
             .rst(rst),
 38
            .iocs (iocs) ,
            .iorw(iorw),
 39
            .tbr(tbr),
 40
 41
            .ioaddr(ioaddr),
 42
             .txd(txd),
            .rate_tx_data(rate_tx_data),
 43
             .enable(enable)
 44
 45
         );
 46
 47 🖻
                                  // initialize all variables
         initial begin
 48
            $display("testing transmit_unit");
 49
             clk = 0;
            rst = 1;
 50
            iocs = 1;
 51
            iorw = 1;
 52
 53
             ioaddr = 0;
            rate tx data = 8'haa; // 8'b1010 1010 = 10'b11 0101 0100 = 10'h354
 54
             enable = 0:
 56
             @ (posedge clk) ;
 57
            rst = 0;
            ioaddr = 2'b00;
 58
 59
 60
            $display("testing transmitting first data byte of oscillating 0s and 1s (8'haa)");
 61
             iorw = 0;
             @ (posedge clk) ;
 62
             iorw = 1;
 63
 64
             while(tbr != 1) begin // wait until done
65
              pulse_enable();
             end
66
```

```
68
             $display("testing transmitting second data byte of 8'b0011_1001");
 69
             rate_tx_data = 8'b0011_1001; // 10_0111_0010 = 272
             iorw = 0;
 71
             @(posedge clk);
             iorw = 1;
 72
 73
             @(posedge clk);
 74
             while(tbr != 1) begin // wait until done
 75
             pulse_enable();
 76
             end
 77
             $stop;
 78 L
         end
 79
 80
 81 📮
                                   // task to pulse enable
          task pulse_enable;
 82
                                   // so that receiver and transmitter can continue
            begin
                repeat (50)
                                   // wait some time
 83
 84
                    @(posedge clk);
                                   // set enable high
 85
                 enable = 1;
 86
                 @(posedge clk);
                                 // for 1 clock cycle
                                  // then reset enable
                 enable = 0;
 87
 88
             end
 89 L
          endtask
 90
 91
 92
          always
93
          #2 clk = !clk;
94 endmodule
```

Receive unit

```
`timescale 1ns / 1ps
   2
     // Company:
                       University of Wisconsin-Madison
     // Engineer:
                       Kai Zhao, John Roy
 4
 5
     11
     // Create Date:
                       2015 Sept 15
 6
     // Design Name:
                       Miniproject1
 8
     // Module Name:
                       receive_unit
     // Project Name:
 9
     // Target Devices: Vertix 5
     // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
12
     // Description: receive unit to receive data from RS232 into SPART
13
     11
14
     // Dependencies: none
15
     11
     // Revision:
16
17
     // Revision 0.01 - File Created
     // Additional Comments:
18
19
     11
    20
21
                              // inputs and outputs
   Emodule receive_unit(
23
        input clk,
         input rst,
24
        input iocs,
26
        input iorw,
         output reg rda,
27
28
         input [1:0] ioaddr,
29
        input rxd,
        output reg [7:0] data received,
         input enable
32 L);
34
        reg [9:0] shift_reg;
                                   // used to hold data
                                   // used to count number of bits to determine whether it is finished
       reg [3:0] bit_cnt;
                                   // FSM signals
        reg load, shift, set_rda;
36
       localparam numberOfBitsPerPacket = 9; // 1 start bit, 1 stop bit, and 8 bits of data, -1 because checked after shifting
37
38
39 🖯
        always @(posedge clk) begin
                                   // if load, then set the number of bits
40 白
          if (load) begin
41
              bit_cnt <= numberOfBitsPerPacket;</pre>
42
            end else if (shift) begin // if shift, then rotate shift register and decrement bit counter
              bit_cnt <= bit_cnt - 1;</pre>
43
              shift_reg <= {rxd, shift_reg[9:1]};</pre>
44
45
           end else if (set rda) begin // if done, then move byte to output
              data_received <= shift_reg[8:1];</pre>
46
           end
47
        end
48
49
        localparam IDLE = 2'b00, RECV = 2'b01, DONE = 2'b10; // states
51
        reg [1:0] state, nxt_state;
        // handle start transitions of the FSM
54 E
        always @ (posedge clk) begin // always go to next state and set rda <= set_rda
55 🛱
          if (rst) begin
              state <= IDLE;
56
              rda <= 0;
57
           end else begin
58
59
             state <= nxt state;</pre>
60
              rda <= set_rda;
61
           end
62 [
        end
```

```
64 📮
         always @(*) begin
65
            // set defaults
             nxt_state = IDLE;
66
            load = 0;
67
68
            shift = 0;
69
            set_rda = 0;
70 日
71 日
             case (state)
                IDLE: begin
                               // initial IDLE state, go to RECV receiving and if saw a start bit
72
                    if (ioaddr == 2'b00 && iorw && !rxd) begin
73
                        load = 1;
74
                        nxt_state = RECV;
75
                    end
76
                 end
77 🛱
                 RECV: begin
                              // RECV state to receive data, shift everytime a bit comes in
                    nxt_state = RECV;
78
79
                    // if enable, then shift
80 白
                    if (enable) begin
81
                        shift = 1;
82
                        // if bit_cnt == 0, then done, hold data until it is read
83 白
                        if (bit_cnt == 0) begin
84
                           nxt_state = DONE;
85
                        end
86
                     end
87
                 end
88
                 default: begin // same as DONE state to hold data
                    // wait for transmit signal to be able to receive next byte of data
89
90
                    nxt_state = DONE;
91
                    set_rda = 1;
                    if (ioaddr == 2'b00 && !iorw) begin
92
93
                        nxt_state = IDLE;
94
                     end
95
                 end
96
             endcase
97
         end
98
99
     endmodule
```

Receive unit test bench

	mescale 1ns / 1p	
11		
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Company:	University of Wisconsin-Madison
11	Engineer:	Kai Zhao, John Roy
11		
	Create Date:	2015 Sept 15
	Design Name:	Miniproject1
	Module Name:	receive_unit_tb
	Project Name:	
	Target Devices:	Vertix 5
	Tool versions:	ModelSim SE 10.3c; Xilinx 14.7
	Description:	test bench for testing receive unit
11	Decertification	core countries contribution
	Dependencies:	receive_unit
11		
	Revision:	
	Revision 0.01 -	File Created
	Additional Comme	
11		: test receiving first data byte of oscillating 0s and 1s
11		: test receiving second data byte of 8'b0011_1001
	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	
	ule receive_unit	();
j -	reg clk;	// signals that are connected to the DUT
5	reg rst;	
	reg iocs;	
	reg iorw;	
	wire rda;	
)	reg [1:0] ioado	ir;
	reg rxd;	
	wire [7:0] data	_recelevea;
l.	reg enable;	
_	Sector Contractor Contractor	
Ē	receive_unit DU	T(// instantiate the DUT
	.clk(clk),	
r E	.rst(rst),	
	.iocs (iocs)	
	.iorw(iorw)	,
	.rda(rda),	
	.ioaddr(ioa	ddr),
	.rxd(rxd),	
		eved(data_receieved),
	.enable(ena	
i L);	
	initial bosts	
r F	initial begin	<pre>// initialize all variables esting receive unit");</pre>
	\$display("t	<pre>// initialize all variables esting receive_unit");</pre>
	<pre>\$display("t clk = 0;</pre>	
	<pre>\$display("t clk = 0; rst = 1;</pre>	
	<pre>\$display("t clk = 0; rst = 1; iocs = 1;</pre>	
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0;</pre>	esting receive_unit");
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0;</pre>	esting receive_unit");
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rxd = 0;</pre>	esting receive_unit");
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rxd = 0; enable = 0;</pre>	esting receive_unit");
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rxd = 0; enable = 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0</pre>	esting receive_unit");
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rxd = 0; enable = 0; 0; foosedge c rst = 0;</pre>	<pre>esting receive_unit"); lk);</pre>
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rxd = 0; enable = 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0</pre>	<pre>esting receive_unit"); 1k);</pre>
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rxd = 0; enable = 0; @(posedge c rst = 0; ioaddr = 2"</pre>	<pre>esting receive_unit"); 1k); b00;</pre>
	<pre>\$display("t clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rst = 0; enable = 0; f(posedge classes) \$d(posedge classes) \$d(splay("test)) \$d(splay("test)</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s");</pre>
	<pre>\$display("t clk = 0; clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rst = 0; enable = 0; ioaddr = 2' \$display("tes iorw = 0;</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive</pre>
	<pre>\$display("to clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rsd = 0; enable = 0; d(posedge co rst = 0; ioaddr = 2' \$display("tos iorw = 0; d(posedge clk</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive);</pre>
	<pre>\$display("t clk = 0; clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; e(posedge cl rst = 0; ioaddr = 2* \$display("tes iorw = 0; e(posedge clk iorw = 1;</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive</pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rsd = 0; enable = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 1; while(rda !=</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate EX to see 01010101 in cmd</pre>
	<pre>\$display("to clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; d(posedge c0; rst = 0; ioaddr = 2' \$display("tos iorw = 1; while(rda != set_data()</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1);</pre>
	<pre>\$display("t clk = 0; clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge cl rst = 0; ioaddr = 2' \$display("tes iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1);</pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rsd = 0; enable = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data())</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1); 0);</pre>
	<pre>\$display("to clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge c0; rst = 0; ioaddr = 2' \$display("tes iorw = 1; while(rda != set_data(end if(data_rece</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1; 0); ieved == 8"h55) begin // check if equals 8"h55 since i sent it oscillating 0s and 1st</pre>
	<pre>\$display("to clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge c0; rst = 0; ioaddr = 2' \$display("tes iorw = 1; while(rda != set_data(end if(data_rece</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate EX to see 01010101 in cmd 1); 0); ieved == 8"h55) begin // check if equals 8"h55 since i sent it oscillating 0s and 1st "cest passed");</pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rsd = 0; enable = 0; @(posedge cl ioaddr = 2' \$display("tes iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(set_data); end else begi</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive) begin // oscillate RX to see 01010101 in cmd 1; 0); ieved -= 8"h55) begin // check if equals 8"h55 since i sent it oscillating 0s and 1st "test passed"); n</pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rsd = 0; enable = 0; @(posedge cl ioaddr = 2' \$display("tes iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(set_data); end else begi</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1; 0); ieved -= 8"h55) begin // check if equals 8"h55 since i sent it oscillating 0s and 1st "test passed"); n</pre>
	<pre>\$display("to clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rsd = 0; enable = 0; @(posedge cl iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(set_data(set_data)) %display("tes source source s</pre>	<pre>esting receive_unit"); k); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive) begin // oscillate RX to see 01010101 in cmd 1; 0; ieved -= 8*h55) begin // check if equals 8*h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_receieved = 8*h55, actual data_receieved = 8*h5h", data_receieved; </pre>
	<pre>\$display("te clk = 0; rat = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge c0; rat = 0; ioaddr = 2' \$display("tes iorw = 0; @(posedge clk iorw = 1; while (rda != set_data(end if (data_rece Sdisplay("tes Sdisplay("tes</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate EX to see 01010101 in cmd 1; 0); ieved == 8"h55) begin // check if equals 8"h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8"h55, actual data_received = 8"h\$h", data_received ting receiving second data byte of 8"b0011_1001");</pre>
	<pre>\$display("te clk = 0; rst = 1; iccs = 1; icrw = 0; icaddr = 0; rad = 0; enable = 0; %d(posedge clk icrw = 0; %d(posedge clk icrw = 1; while(rda != set_data(set_data(end if (data_rece \$display("tes staiplay("tes rxd = 1;</pre>	<pre>esting receive_unit"); http:// set iorw = 0 to make receiver avialable to receive // set iorw = 0 to make receiver avialable to receive // set iorw = 0 to receive // cocillate RX to see 01010101 in cmd // set iorw = 8"h55) bagin // check if equals 8"h55 since i sent it oscillating 0s and 1st "test passed"); " "test failed, expected data_received = 8"h55, actual data_received = 8"h%h", data_received ting receiving second data byte of 8"b0011_1001"); // need to send 1 for 1 clock cycle for stop bit</pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; rsd = 0; enable = 0; @(posedge cl iorw = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(set_data(set_data)) %display("tes Sdisplay("tes Sdisplay("tes rxd = 1; iorw = 0;</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive) begin // oscillate RX to see 01010101 in cmd 1); 0); ieved -= %'h55) begin // check if equals %'h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = %'h55, actual data_received = %'h%h", data_received ting receiving second data byte of %'b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again</pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge c0; rst = 0; ioaddr = 2' \$display("tes iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(end if (data_rece \$display("tes rsd = 1; iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(end \$display("tes rsd = 1; iorw = 0; @(posedge clk)</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive) begin // oscillate RX to see 01010101 in cmd 1); 0); ieved -= %'h55) begin // check if equals %'h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = %'h55, actual data_received = %'h%h", data_received ting receiving second data byte of %'b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again</pre>
	<pre>\$display("te clk = 0; rst = 1; icos = 1; icorw = 0; icoaddr = 0; rad = 0; enable = 0; @(posedge = 0; icoaddr = 2' \$display("tes icorw = 0; @(posedge clk icorw = 0; @(posedge clk icorw = 1; while(rda != set_data(set_data, rece \$display("tes rxd = 1; icorw = 0; @(posedge clk icorw = 1;)</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1); 0); ieved == & h55) begin // check if equals & h55 since i sent it oscillating 0s and 1st "test passed"); m "test failed, expected data_receieved = & h55, actual data_receieved = & hthm", data_receieved ting receiving second data byte of & b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); </pre>
	<pre>\$display("tc clk = 0; rst = 1; iccs = 1; icrw = 0; icaddr = 0; rsd = 0; enable = 0; @(posedge clk icrw = 0; %display("tcs rst = 0; icaddr = 2" \$display("tcs rst_dta(set_dta(set_dta(sdisplay("tcs rsd = 1; icrw = 0; %display("tcs rsd = 1; icrw = 1; %display("tcs rsd = 1; icrw = 1; %display("tcs rsd = 1; %display("t</pre>	<pre>esting receive_unit"); kb00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive); // set iorw = 0 to receive); // set iorw = 0 to receive); ived =- 8*h55) begin // check if equals 8*h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8*h55, actual data_received = 8*h5h", data_received ting receiving second data byte of 8*b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start </pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; enable = 0; ioaddr = 2* \$display("tes iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(end if (data_rece \$display("tes rxd = 1; iorw = 0; @(posedge clk iorw = 1; sdisplay("tes rxd = 1; iorw = 0; @(posedge clk iorw = 1; set_data(0); set_data(1);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // cocillate EX to see 01010101 in cmd 1); 0); ieved -= 8*h55) begin // check if equals 8*h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8*h55, actual data_received = 8*h%h", data_received ting receiving second data byte of 8*b0011_1001*); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // lst bit, need to be 0 to start // 2nd bit (least significant data bit)</pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(end \$display("tes \$display("tes rxd = 1; iorw = 0; @(posedge clk iorw = 1; set_data(0); set_data(1); set_data(1);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1); 0); ieved -= 8*h55) begin // check if equals 8*h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8*h55, actual data_received = 8*h5**, data_received ting receiving second data byte of 8*b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start // 2nd bit (least significant data bit) // 3rd bit</pre>
	<pre>\$display("tc clk = 0; rst = 1; iccs = 1; icrw = 0; icaddr = 0; enable = 0; @(posedge clk icrw = 0; icaddr = 2' \$display("tcs icrw = 0; @(posedge clk icrw = 0; @(posedge clk icrw = 0; @(posedge clk icrw = 0; @(posedge clk icrw = 1; while(rda != set_data(\$display("tcs rxd = 1; icrw = 0; @(posedge clk icrw = 1; icrw = 0; @(posedge clk icrw = 0; for a classification); set_data(0); set_data(0);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1); 0); isved == 8%55) begin // check if equals 8%55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8%55, actual data_received = 8%%%, data_received ting receiving second data byte of 8%0011_1001"); // newl to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start // 3rd bit // 3rd bit // 3rd bit </pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; d(posedge cl ioaddr = 2* \$display("tes iorw = 0; d(posedge clk iorw = 1; while(rda != set_data(end if (data_rece \$display("tes rxd = 1; iorw = 0; d(posedge clk iorw = 1; st_data(0); set_data(0); set_data(0); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1); 0); ieved == 8*h55) begin // check if equals 8*h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_receieved = 8*h55, actual data_receieved = 8*h%h", data_receieved ting receiving second data byte of 8*b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start // 2nd bit (least significant data bit) // 3rd bit // 4rd bit // 5th bit </pre>
	<pre>\$display("te clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge cl ioaddr = 2" \$display("tes iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(end if (data_rece \$display("tes st_data(0); set_data(0); set_data(1); set_data(1); set_data(1);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate EX to see 01010101 in cmd 1); 0); ieved -= 8*h55) begin // check if equals 8*h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8*h55, actual data_received = 8*h5**, data_received ting receiving second data byte of 8*b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start // 2nd bit (least significant data bit) // 3rd bit // 6th bit // 6th bit </pre>
	<pre>\$display("tc clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge cl iorw = 0; @(posedge cl iorw = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1); 0); isved == 8*h55) begin // check if equals 8*h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8*h55, actual data_received = 8*h5h", data_received; ting receiving second data byte of 8*b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start // 3rd bit // 4th bit // 5th bit // 7th bit </pre>
	<pre>\$display("te clk = 0; rst = 1; iccs = 1; icrw = 0; icaddr = 0; enable = 0; enable = 0; d(posedge cl forw = 0; enable = 0; d(posedge clk icrw = 0; end = 2* %display("tes set_data(end if (data_rece \$display("tes rxd = 1; icrw = 0; d(posedge clk icrw = 1; while(rda != set_data(); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // cocillate BX to see 01010101 in cmd 1); 0); ieved == 8"h55) begin // check if equals 8"h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 8"h55, actual data_received = 8"h\$h", data_received ting receiving second data byte of 8"b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start // 2nd bit (least significant data bit) // 3rd bit // fth bit // 6th bit // 7th bit // 7th bit </pre>
	<pre>\$display("tc clk = 0; rst = 1; iocs = 1; iorw = 0; ioaddr = 0; enable = 0; @(posedge cl iorw = 0; @(posedge cl iorw = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 0; @(posedge clk iorw = 1; while(rda != set_data(set_data(); set_data(1); set_data(1); set_data(1); set_data(1); set_data(1);</pre>	<pre>esting receive_unit"); lk); b00; ting receiving first data byte of oscillating 0s and 1s"); // set iorw = 0 to make receiver avialable to receive); // set iorw = 0 to receive 1) begin // oscillate RX to see 01010101 in cmd 1); 0); isved == 81h55) begin // check if equals 81h55 since i sent it oscillating 0s and 1st "test passed"); n "test failed, expected data_received = 81h55, actual data_received = 81h5h", data_received) ting receiving second data byte of 81b0011_1001"); // need to send 1 for 1 clock cycle for stop bit // make receiver avialable again); // 1st bit, need to be 0 to start // 3rd bit // 3rd bit // 4th bit // 5th bit // 7th bit </pre>

89	Ė	while (rda != 1) begin // wait until done
90		#2;
91	-	end
92	þ	if (data received == 8'b0011 1001) begin // check if it makes the data i sent
93		\$display("test passed");
94		end else begin
95		\$display("test failed, expected data_received = 8'b0011_1001, actual data_received = 8'b%b", data_received);
96	-	end
97		\$stop;
98	L	end
99		
100		
101	Ę	task set_data; // task for sending data
102		input data;
103	白	begin
104		rxd = data;
105		#200;
106		@(posedge clk);
107		<pre>enable = 1;</pre>
108		@(posedge clk);
109		enable = 0;
110	-	end
111	L	endtask
112		
113		
114		always // clock
115		#2 clk = !clk;
116		
117	end	dmodule

Transmit receive unit test bench

```
`timescale 1ns / 1ps
3
    // Company:
                    University of Wisconsin-Madison
4
    // Engineer:
                    Kai Zhao, John Roy
5
    11
    // Create Date:
                   2015 Sept 15
6
7
    // Design Name: Miniproject1
    // Module Name:
                    transmit_receive_unit_tb
8
9
    // Project Name:
    // Target Devices: Vertix 5
    // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
11
    // Description: test bench for testing transmit_unit and receive_unit
12
13
    11
    // Dependencies: transmit_unit, receive_unit
14
15
    11
    // Revision:
16
    // Revision 0.01 - File Created
17
18
    // Additional Comments:
19
    11
        test case 1: test transmitting and receiving oscillating 0s and 1s (8'haa)
20
           test case 2: test transmitting and receiving 8'b0011 1001
   11
    21
22
23
   module transmit_receive_unit_tb();
24
                            // signals that are connected to the DUT
25
       reg clk;
26
       reg rst;
27
       reg iocs;
28
       req iorw;
29
       wire tbr;
       wire rda;
31
       reg [1:0] ioaddr;
32
       wire trxd:
33
       reg [7:0] rate tx data;
34
        wire [7:0] data_received;
35
       reg enable;
37 E
                              // instantiate the first DUT
        transmit unit tDUT(
          .clk(clk),
38
39
           .rst(rst),
           .iocs (iocs) ,
40
41
           .iorw(iorw),
42
           .tbr(tbr),
43
           .ioaddr(ioaddr),
44
           .txd(trxd),
45
           .rate_tx_data(rate_tx_data),
46
           .enable(enable)
47
       );
48
49 🖃
        receive_unit rDUT(
                           // instantiate the second DUT
50
           .clk(clk),
51
           .rst (rst) ,
52
           .iocs (iocs) ,
53
           .iorw(iorw),
54
           .rda(rda),
55
           .ioaddr(ioaddr),
56
           .rxd(trxd),
57
           .data_receieved(data_receieved),
58
59
           .enable(enable)
        );
```

<pre>100 end \$stop; end 101 102 end 103 104 105 task pulse_enable; // task to pulse enable signal 106 begin 107 repeat (50) // wait multiple cycl cycles 108 0 (posedge clk); 109 enable = 1; // set enable high 0 0 0 0 0 0; // for 1 clock cycle 110 enable = 0; // reset enable 111 endtask 113 114 115 116 always</pre>		
<pre>dist = 0;</pre>	T	
<pre>44 45 45 45 45 45 45 45 45 45 45 45 45 4</pre>		
<pre>66 icor = 1; icor = 0; fd(peedge cl3); icor = 0; fd(peedge cl3); icor = 1; while(tr != 1) hegin // wait until done pulse_snable(); end if (data_received = 0*hea) begin if (data_received = 0*heal) begin if (data_received = 0*heal) begin if (data_received = 0*heal) if (data_received = 0*001_1001, actual data_received = 0*hear, data_received); end if (data_received = 0*heal); if (data_received = 0*heal) if (data_received = 0*hear); data_received); end if (data_received = 0*hear); // task to pulse enable signal if (data_received = 0*hear); // task to pulse enable signal if (data_received = 0*hear); // task to pulse enable signal if (data_received = 0*hear); // task to pulse enable signal if (data_received = 0*hear); // task to pulse enable signal if (data_received = 0; // wait multiple cycl cycles if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse enable signal if (data_received = 0; // task to pulse</pre>		
<pre>66 67 68 68 69 69 69 69 69 69 69 60 70 70 70 6 6 6 6 6 6 6 6 6 6 6 6 6 6</pre>		
<pre>97 97 98 98 99 99 99 99 99 99 90 90 90 90 90 90 90</pre>		iocs = 1;
<pre>set rate rate - 0 * hes; // 0 * biolo_1010 = 10 * bil_0010 = 10 * bist emble = 0; isoddr = 2 * boo; fd (pready cik); rst = 0; isoddr = 2 * boo; fd (pready cik); isor = 1; fd (pready cik); isor = 1; fd (pready cik); isor = 1; fd (pready cik); isor = 1; fd (pready cik); fd (pready cik); fd (pready cik); fd (pready cik); fd (pready cik); for = 0; fd (pready cik); for = 0; fd (pready cik); for = 0; fd (pready cik); for = 0; fd (pready cik); for = 1; fd (pready cik); for = 1; fd (pready cik); for = 0; fd (pready cik); fd (pready cik); fd (pready cik); for = 0; fd (pready cik); for = 0; fd (pready cik); for = 0; fd (pready cik); fd (pready cik); makis = 1; fd (pready cik); fd (pready cik); fd (pready cik); makis = 0; fd (pready cik); fd (pread</pre>	66	iorw = 1;
<pre>emable = 0; (procedy cik); rst = 0; isaddr = 2'b00; display("testing transmitting first data byte of oscillating 0s and 1s (5'has)"); icrw = 0; icrw = 0; icrw = 0; icrw = 0; display("testing transmitting first data byte of oscillating 0s and 1s (5'has)"); icrw = 0; display("testing transmitting first data byte of oscillating 0s and 1s (5'has)"); icrw = 0; display("testing transmitting first data byte of oscillating 0s and 1s (5'has)"); icrw = 0; display("testing transmitting first data byte of oscillating 0s and 1s (5'has)"); icrw = 0; display("testing transmitting second data preceived = 8'has, actual data_received = 8'hth", data_received); end display("testing transmitting second data byte of 8'b001_1001"); rect_ts_data = 4'b001_1001; // 10_011_0010 = 272 icrw = 0; display("testing transmitting second data byte of 8'b001_1001"); rect_ts_data = 4'b001_1001; // 10_011_0010 = 272 icrw = 0; display("testing transmitting second data byte of 8'b001_1001"); rect_ts_data = 4'b001_1001; // 10_011_0010 = 272 icrw = 0; display("testing transmitting begin display("testing transmitting second data preceived = 8'bbb", data_received); end display("test failed, expected data_received = 8'0'b001_1001, actual data_received = 8'bbb", data_received); end display("test failed, expected data_received = 8'b'b001_1001, actual data_received = 8'bbb", data_received; end display("test failed, expected data_received = 8'b'b001_1001, actual data_received = 8'bbb", data_received; end display("test failed, expected cycle mable = 0; // rest enable signal display("test failed, expected cycle mable = 0; // rest enable end display("test failed, expected cycle mable = 0; // rest enable end display("test enable signal display("test enable</pre>	67	<pre>ioaddr = 0;</pre>
<pre>70 70 71 72 73 74 75 75 75 75 75 75 75 75 75 75 75 75 75</pre>	68	rate_tx_data = 8'haa; // 8'b1010_1010 = 10'b11_0101_0100 = 10'h354
<pre>71 72 73 74 75 75 75 75 75 75 75 75 75 75 75 75 75</pre>	69	enable = 0;
<pre>icadir = 2'b00; fsieplay("testing transmitting first data byte of oscillating 0s and 1s (8'haa)"); icaw = 0; f(gosedge clk); icaw = 0; f(gosedge clk); icaw = 1; while(thr != 1) bgin // wait until dome pulse_smable(); end if (data_received == 8'haa) begin fsdisplay("titest passed"); end isdisplay("titest failed, expected data_received = 8'haa, actual data_received = 8'hbh", data_received); end fsdisplay("titest failed, expected data_received = 8'haa, actual data_received = 8'hbh", data_received); end fsdisplay("titest failed, expected data_received = 8'hbab", data_received); end fsdisplay("titest failed, expected data_received = 8'b0011_1001"); rate_rtx_data = 8'b0011_0010 = 272 icar = 0; f(gosedge clk); if (data_received == 8'b0011_1001) begin fsdisplay("titest failed, expected data_received = 8'8'b0011_1001, actual data_received = 8'bbb", data_received); end fstop; end task pulse_enable; // task to pulse enable signal begin repeat (50) // wait multiple cycl cycles end endtask inter = 0; // test enable signal begin repeat (50) // wait multiple cycl cycles endtesk inter = 0; // test enable inter = 0; // test enable signal endtask inter = always</pre>	70	@(posedge clk);
<pre>73 74 75 75 76 76 76 76 76 76 76 76 76 76 76 76 76</pre>	71	rst = 0;
<pre>4 \$display("tasting transmitting first data byte of oscillating 0s and 1s (0*haa)"); iorw = 0; 0 {posedge cll}; iorw = 1; while(thr [= 1) begin // wait until done</pre>	72	<pre>ioaddr = 2'b00;</pre>
<pre>75 76 77 76 77 77 77 78 8 9 9 9 9 9 9 9 9 9 9 9 9 9</pre>	73	
<pre>76 77 78 8 9 (posedge clk); iorw = 1; vhile(thr != 1) begin // wait until done 79 81 81 81 84 84 85 85 8 8 8 8 8 8 8 8 8 8 8 8 8 8</pre>	74	\$display("testing transmitting first data byte of oscillating 0s and 1s (8'haa)");
<pre>177 178 179 1 10 10 10 10 10 10 10 10 10 10 10 10 10</pre>	75	iorw = 0;
<pre>rs pulse_enable(); end end if (dats_received == \$'hen) begin stipplay("ttest failed, expected dats_received = 8'has, actual dats_received = 8'hth", dats_received); end sdisplay("ttest failed, expected dats_received = 8'has, actual dats_received = 8'hth", dats_received); end sdisplay("ttest failed, expected dats_received = 8'has, actual dats_received = 8'hth", dats_received); end sdisplay("ttest failed, expected dats_received = 8'holl_loll"); rate_ts_dats = 8'holl_loll // 10_011_0010 = 272 iow = 0; g(posedge cll); iow = 1; f(dats_received == 8'holl_loll begin sdisplay("ttest passed"); end end else begin stop; end task pulse_enable; // task to pulse enable signal begin requet (\$0) // wait multiple cycl cycles f(posedge cll); // set enable high f(posedge cll); // for 1 clock cycle end endtask inter = 0; // reset enable inter = 0; // reset enable inter = 0; // reset enable</pre>	76	(posedge clk);
<pre>79 80 80 80 91 80 80 80 80 80 81 82 84 84 84 85 84 84 85 84 85 84 85 84 85 84 85 84 85 84 85 85 85 85 85 85 85 85 85 85 85 85 85</pre>	77	iorw = 1;
<pre>60</pre>	78 🛱	<pre>while(tbr != 1) begin // wait until done</pre>
<pre>60</pre>	79	
<pre>82 84 85 84 85 84 85 84 85 85 86 84 85 85 85 85 85 85 85 85 85 85 85 85 85</pre>	80 -	
<pre>22 23 24 25 25 25 25 26 27 27 28 28 29 29 29 29 29 29 29 20 20 20 20 20 20 20 20 20 20 20 20 20</pre>	81 🗄	if (data receieved == 8'haa) begin
<pre>83 84 84 84 84 84 84 84 85 84 84 84 85 84 84 85 84 85 84 85 84 85 84 85 84 85 84 85 84 85 85 85 85 85 85 85 85 85 85 85 85 85</pre>	82	
<pre>84 signlay("\test failed, expected data_receieved = 8'haa, actual data_receieved = 8'hh", data_receieved); 85 87 88 88 87 89 90 90 90 90 90 90 90 90 90 90 90 90 90</pre>	83	
<pre>85 end '''''''''''''''''''''''''''''''''''</pre>	84	
<pre>88 rate_tx_data = 8'b0011_1001; // 10_0111_0010 = 272 10 rate_tx_data = 8'b0011_1001; // 10_0111_0010 = 272 10 rate_tx_data = 8'b0011_1001; // 10_0111_0010 = 272 10 rate_tx_data = 8'b0011; // 10_0111_0010 = 272 10 rate_tx_data = 8'b0011; // 10_011 = 272 10 rate_tx_data = 8'b0011; // 10_011; // 10_01 10 rate_tx_data = 8'b0011; // 10_0; // 10_0; // 10_0; // 10_0; // 10_0; // 10_0; // 10_0; // 10</pre>	85 -	
<pre>103 104 105 E task pulse_enable; // task to pulse enable signal 106 begin 107 108 109 @ (posedge clk); 109 @ @ (posedge clk); 109 @ @ (posedge clk); // set enable high 100 @ @ (posedge clk); // for 1 clock cycle 111 enable = 0; // reset enable 112 enable = 0; // reset enable 113 114 114 115 116 always</pre>	90 91 92 93 94 95 96 97 98 99 99 100 101	<pre>iorw = 0; @(posedge clk); iorw = 1; @(posedge clk); while(tbr != 1) begin // wait until done pulse_enable(); end if (data_received == 8'b0011_1001) begin \$display("\ttest passed"); end else begin \$display("\ttest failed, expected data_received = 8'8'b0011_1001, actual data_received = 8'b%b", data_received); end \$stop;</pre>
<pre>104 105 task pulse_enable; // task to pulse enable signal 106 begin 107 repeat (50) // wait multiple cycl cycles 108 @ (posedge clk); 109 @ (posedge clk); 110 @ (posedge clk); // for 1 clock cycle 111 enable = 0; // reset enable 112 end 113 endtask 114 115 116 always</pre>		ena
<pre>105 F task pulse_enable; // task to pulse enable signal 106 begin 107 107 108 108 109 109 109 10 10 10 10 10 10 10 10 10 10 10 10 10</pre>	103	
<pre>106 begin</pre>	105 🖂	task pulse enable: // task to pulse enable signal
<pre>107 107 108 109 109 109 109 109 109 109 109 109 109</pre>	106	
108 @ (posedge clk); 109 enable = 1; // set enable high 110 @ (posedge clk); // for 1 clock cycle 111 enable = 0; // reset enable 112 end 113 endtask 114 115 115 always	107	
<pre>109 enable = 1; // set enable high 110 @ (posedge clk); // for 1 clock cycle 111 enable = 0; // reset enable 112 end 113 end 114 115 116 always</pre>	108	
<pre>110 @ (posedge clk); // for 1 clock cycle enable = 0; // reset enable 111 end endtask 114 115 116 always</pre>	109	
112 end 113 endtask 114	110	
113 endtask 114 115 116 always	111	enable = 0; // reset enable
114 115 116 always	112 -	end
115 116 always	113 L	endtask
116 always	114	
	115	
$\frac{117}{12}$ #2 clk = !clk:	116	
	117	#2 clk = !clk;
118 endmodule	118 en	dmodule

Driver

timescale 1ns / 1p // Company: University of Wisconsin-Madison // Engineer: Kai Zhao, John Roy 2015 Sept 15 // Create Date: 6 // Design Name: Miniproject1 8 // Module Name: driver // Project Name: 9 // Target Devices: Vertix 5 // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
// Description: driver for sending data to SPART // // Dependencies: none 14 // Revision:
// Revision 0.01 - File Created 16 18 // Additional Comments: 19 22 Emodule driver(// inputs and outputs input clk, 24 25 input rst, input [1:0] br cfg, output iocs, 26 27 28 output iorw, input rda, 29 input thr, output reg [1:0] ioaddr, 31 inout [7:0] databus 32 L); 34 assign iocs = 1; // can't stop, won't stop reg iorw_reg; assign iorw = iorw_reg; // set to reg so that it can be set and keep its prototype reg [7:0] inout data; 38 reg [7:0] held_data; 39 reg update_baud, start_sending_baud_hi, start_sending_baud_lo, start_receiving, start_transmitting, receiving, transmitting; 40 41 reg [2:0] old_br_cfg; reg [2:0] Ol_Db_ctg; localparam BAUD_4800 = 16'h0514; // divsion for 4800 = 10000000/(16*4800) - 1 = d1300 = h0514 localparam BAUD_9600 = 16'h028a; // divsion for 9600 = 100000000/(16*9600) - 1 = d550 = h028a localparam BAUD_19200 = 16'h0145; // divsion for 19200 = 100000000/(16*19200) - 1 = d325 = h0145 localparam BAUD_38400 = 16'h00a2; // divsion for 38400 = 100000000/(16*38400) - 1 = d162 = h00a2 42 43 45 46 reg [15:0] new_baud; 47 48 localparam IDLE = 3'b000, BAUDHI = 3'b001, BAUDLO = 3'b010, RECV = 3'b011, TRANS = 3'b100; 49 reg [2:0] state, nxt_state; 51 // handle start transitions of the FSM always @ (posedge clk) begin if (rst) begin 53 54 state <= IDLE;</pre> old br cfg <= 3'b100; 56 end else begin if (update_baud) begin // load baud into new baud register old_br_cfg <= {1'b0, br_cfg};</pre> // case statement to load bad rate 59 new_baud <= br_cfg == 0 ? BAUD_4800 : 60 // based on br_cfg br_cfg == 2 ? BAUD_4800 : br_cfg == 2 ? BAUD_19200 : br_cfg == 3 ? BAUD_38400 : 62 63 BAUD_9600; // default case end else if (start_sending_baud_hi) begin // load baud rate high 64 65 inout_data <= new_baud[15:8];</pre> 66 ioaddr <= 2'b11; end else if (start_sending_baud_lo) begin // load baud rate low 67 68 inout_data <= new_baud[7:0];</pre> 69 ioaddr <= 2'b10; end else if (start_receiving) begin // load available data inout_data <= databus; end else if (start_transmitting) begin // put available data back on inout bus 73 74 // do nothing, since value is already loaded in inout_data, which is pushed by databus if in receive state
inout_data <= held_data;</pre> end else if (receiving) begin // set it to receive 76 77 ioaddr <= 2'b00;</pre> iorw_reg <= 1; end else if (transmitting) begin 78 // set it to transmite ioaddr <= 2'b00; iorw_reg <= 0;</pre> 79 80 81 end else begin 82 ioaddr <= 2'b01; // set to be able to read rda and tbr end 84 state <= nxt_state;</pre> 85 end 86 end

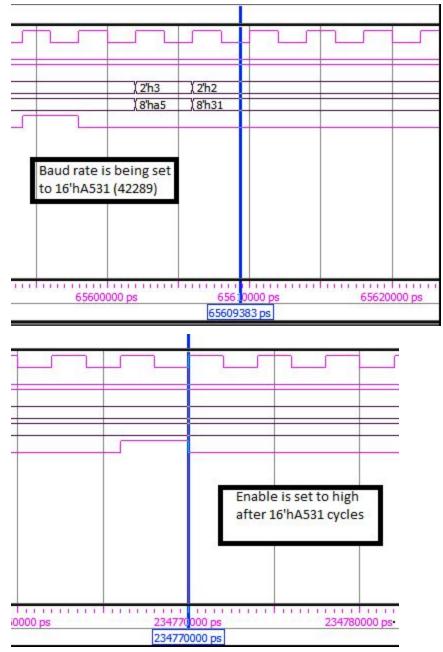
```
88
          assign databus = (state == IDLE || state == RECV) ? 8'bzz : inout_data;
 89
 90 🖃
          always @(*) begin
              // set defaults
 91
 92
              nxt state = IDLE;
              update_baud = 0;
 93
 94
              start_sending_baud_hi = 0;
 95
              start sending baud lo = 0;
              start_receiving = 0;
 96
 97
              receiving = 0;
 98
              start_transmitting = 0;
 99
              transmitting = 0;
100 🛱
              case (state)
                                            // send baud rate if unitialized
                  IDLE: begin
102
                      if (old_br_cfg[2] || br_cfg != old_br_cfg[1:0]) begin
103
                          update_baud = 1;
104
                         nxt state = BAUDHI;
105
                      end else if (rda) begin // receive data if data is ready to be received
106
                         start_receiving = 1;
107
                          nxt state = RECV;
108
                      end
109
                  end
                  BAUDHI: begin
                                             // send baud rate high
                     start_sending_baud_hi = 1;
112
                     nxt state = BAUDLO;
113
                  end
114 🗄
                                             // send baud rate low
                  BAUDLO: begin
115
                      start sending baud lo = 1;
116
                      nxt_state = IDLE;
117
                  end
118
                  RECV: begin
                                             // stay until you can transmit the data back
119
                     nxt_state = RECV;
120
                      receiving = 1;
                      if (tbr) begin
122
                         held_date = databus;
123
                          start transmitting = 1;
124
                          nxt_state = TRANS;
125
                      end
                  end
126
127
                                              // stay until done transmitting
                  TRANS: begin
                     nxt_state = TRANS;
128
129
                      transmitting = 1;
130 🛱
                      if (tbr) begin
131
                        nxt state = IDLE;
132
                      end
133
                  end
134 🛱
                  default: begin
135
                     // nothing, so return to IDLE
136
                  end
137
               endcase
138
           end
139
140 endmodule
```

Driver test bench

```
1 `timescale 1ns / 1ps
 University of Wisconsin-Madison
 3
    // Company:
    // Engineer:
 4
                      Kai Zhao, John Roy
 5
 6
    // Create Date:
                      2015 Sept 15
    // Design Name:
                      Miniproject1
8
    // Module Name:
                      driver tb
    // Project Name:
9
    // Target Devices: Vertix 5
    // Tool versions: ModelSim SE 10.3c; Xilinx 14.7
// Description: test bench for testing driver
12
    // Dependencies: driver
14
15
    // Revision:
16
     // Revision 0.01 - File Created
18
    // Additional Comments:
           test case 1: test baud rate 4800
19
    11
            test case 2: test baud rate 9600
    11
           test case 3: test baud rate 19200
21
    11
     11
           test case 4: test baud rate 38400
    23
24
    module driver_tb();
25
26
                         // signals that are connected to the DUT
27
        reg clk:
28
        reg rst;
        reg [1:0] br_cfg;
29
30
        wire iocs;
        wire iorw:
32
        reg rda;
        reg tbr;
        wire [1:0] ioaddr;
34
        wire [7:0] databus;
36
                                  // register for handling inouts
37
        reg[7:0] databus_reg;
38
        assign databus = databus_reg;
39
                         // instantiate the DUT
40 🚍
        driver DUT (
41
            .clk(clk),
42
            .rst (rst) ,
43
            .br_cfg(br_cfg),
            .iocs (iocs) ,
44
45
            .iorw(iorw),
46
            .rda(rda),
            .tbr(tbr),
47
48
            .ioaddr(ioaddr),
49
            .databus (databus)
50
        );
52 E
                              // initialize variables
        initial begin
            $display("testing driver");
            clk = 0:
54
            rst = 1;
            br_cfg = 2'b01;
56
57
            @ (posedge clk) ;
58
            rst = 0;
59
            rda = 0;
60
            tbr = 0;
61
            databus_reg = 8'hzz;
62
            @ (posedge clk) ;
63
64
            $display("Testing baud rate 4800");
            br_cfg = 2'b00;
repeat (5) // wait for baud_rate to load
65
66
67
              @ (posedge clk);
            if (DUT.new_baud == DUT.BAUD_4800) begin
  $display("\ttest passed");
68 🛱
69
            end else begin
              $display("\ttest failed, DUT.new_baud = 16'h%h, DUT.BAUD_4800 = 16'h%h", DUT.new_baud, DUT.BAUD_4800);
            end
72
74
            $display("Testing baud rate 9600");
```

75	br_cfg = 2'b01;
76	repeat (5)
77	0 (posedge clk);
78 白	if (DUT.new_baud == DUT.BAUD_9600) begin
79	\$display("\ttest passed");
80	end else begin
81	\$display("\ttest failed, DUT.new_baud = 16'h%h, DUT.BAUD_4800 = 16'h%h", DUT.new_baud, DUT.BAUD_9600);
82 -	end
83	
84	<pre>\$display("Testing baud rate 19200");</pre>
85	br_cfg = 2'bl0;
86	repeat (5)
87	0 (posedge clk);
88	if (DUT.new_baud == DUT.BAUD_19200) begin
89	\$display("\ttest passed");
90	end else begin
91	\$display("\ttest failed, DUT.new baud = 16'h%h, DUT.BAUD 4800 = 16'h%h", DUT.new baud, DUT.BAUD 19200
92 -	end
93	
94	<pre>\$display("Testing baud rate 38400");</pre>
95	<pre>br_cfg = 2'bl1;</pre>
96	repeat (5)
97	<pre>@(posedge clk);</pre>
98 白	if (DUT.new_baud == DUT.BAUD_38400) begin
99	<pre>\$display("\ttest passed");</pre>
100	end else begin
101	\$display("\ttest failed, DUT.new_baud = 16'h%h, DUT.BAUD_4800 = 16'h%h", DUT.new_baud, DUT.BAUD_38400)
102 -	end
103	
104	#100;
105	\$stop;
106 L	end
107	
108	always
109	#2 clk = !clk;
110	
111 en	dmodule

Baud Rate Generator Testbench



```
VSIM 5> run -all
# testing baud_rate_generator
# test first cycle of processor only writing to baud rate generator low
       test passed
# test second cycle of processor only writing to baud rate generator low
±
       test passed
# test first cycle of processor only writing to baud rate generator high
       test passed
# test second cycle of processor only writing to baud rate generator high
       test passed
# test first cycle of processor writing 8'ha531 to baud rate generator
       test passed
# test second cycle of processor writing 8'ha531 to baud rate generator
       test passed
# test baud rate generator still workes even after changing address
       test passed
# test that enable is never high when iocs is low
       test passed
# ** Note: $stop : I:/Oschool/ece554/miniproject1/baud_rate_generator_tb.v(188)
# Time: 835238 ns Iteration: 1 Instance: /baud_rate_generator_tb
# Break in Module baud_rate_generator_tb at I:/0school/ece554/miniproject1/baud_rate_generator_tb.v line 188
      1
```

Bus Interface Testbench

A B S Now	,	a	/bus_interface_tb/rate_tx_data	/bus_interface_tb/databus	D-*/bus_interface_tb/ioaddr		 /bus_interface_tb/iosrw /bus interface_tb/irda 		
50000 ps	Shtz:	8'hf1	8'hf1	8'hf1	2'h0	1'h1	1'h1 1'h0	1'h1	Msgs
sq 0000 sc 5000 sc	Write to baud rate generator low	8h55		8	(2'h0 2'h2				
s 10000 ps	te aa		8'haa	8'haa	12		<u>(</u>)		
111111111111 15000 ps	Write t genera		8'hbc	, 8'hbc	2'h3		2 ³ 2		
20000 ps 2	Write to baud rate generator high	6-000					- <u>-</u>	<u>at - 1</u> 2	
25000 ps 30000 ps	Read the status		, 8'h02	, 8'h02	2'h1				
0 ps 35000 ps	tus R & Rde		, 8'hde	, 8'hde	2'h0				
40000 ps	de Write a value to TX controller		de	de	0				
se 5000 ps 10000 ps 15000 ps 20000 ps 25000 ps 35000 ps 35000 ps 46000 ps 45000 ps 500	Read a value from RX controller	,8'hf1	8'hf1	8'hf1					
50000 ps	. 🏹								

```
VSIM 13> run -all
# testing bus interface
# test processor writing to baud rate generator low
#
      test passed
# test processor writing a different value to baud rate generator high
      test passed
#
# test processor reading the status register demo
      test passed
#
# test processor writing a value to transmit controller
ŧ
       test passed
# test processor reading a value from receive controller
       test passed
# ** Note: $stop : I:/Oschool/ece554/miniproject1/bus_interface_tb.v(114)
   Time: 50 ns Iteration: 0 Instance: /bus_interface_tb
#
# Break in Module bus_interface_tb at I:/0school/ece554/miniproject1/bus_interface_tb.v line 114
```

Transmit Unit Testbench

VSIM 2> run -all
testing transmitting first data byte of oscillating 0s and 1s (8'haa)
testing transmitting second data byte of 8'b0011_1001
** Note: \$stop : I:/0school/ece554/miniproject1/transmit_unit_tb.v(74)
Time: 2258 ns Iteration: 1 Instance: /transmit_unit_tb
Break at I:/0school/ece554/miniproject1/transmit_unit_tb.v line 74

Now		/receive_unit_tb/enable	Interviewunit_tb/data_received	/receive_unit_tb/rxd	Image: Arrow of the second	🔶 /receive_unit_tb/rda	<pre>/receive_unit_tb/jorw</pre>	/receive_unit_m/rst	<pre>/receive_unit_tb/dk</pre>	8 .
5004000 ps 5008770 ps		-No Data-	-No Data-	-No Data-	-No Data-	-No Data-	-No Data-	-No Data-	-No Data-	Msgs
		+		P	2'h0		_			
-		╞		5				+		
1000000 ps		-	+				_	+	-	
			+	5			_	+	-	
-		_					_			
-		-	+	L			_		-	
-										
1 2000000 ps										
10000000 ps	Sending TX line									
-	TX line		8h55	ſ						
-	55 ove					Ľ				
-	r the	-								
30000	5	-		Ľ						
3000000 ps		F								
-		-		5						
-		-								
-										
- 		F		٢						
4000000 ps		-								
-		-		5						
	the TX line	-	(8'h39			5		\parallel		
	Sending 8'h39 over the TX line		139					\parallel		
) over								I	

Receive Unit Testbench

ModelSim> run -all
testing receiving first data byte of oscillating 0s and 1s
test passed
testing receiving second data byte of 8'b0011_1001
test passed
** Note: \$stop : I:/0school/ece554/miniproject1/receive_unit_tb.v(94)
Time: 4504 ns Iteration: 0 Instance: /receive_unit_tb
Break in Module receive_unit_tb at I:/0school/ece554/miniproject1/receive_unit_tb.v line 94

			÷ 🔶 /		± 🔶 /				*	•	8
		/receive_unit_tb/en	/receive_unit_tb/da	/receive_unit_tb/rxd	/receive_unit_tb/io	/receive_unit_tb/rda	/receive_unit_tb/iorw 1'h1	/receive_unit_tb/locs	/receive_unit_tb/rst	/receive_unit_tb/dk	
		_unit_t	_unit_t	_unit_t	_unit_t	unit_t	_unit_t	_unit_t	_unit_t	unit t	
		tb/en	tb/da	tb/rxd	tb/io	tb/rda	tb/iorw	tb/iocs	tb/rst	tb/clk	
		1'ho	. 8'h39	1'h1	2'h0	1'h1	1'h1	1'h1	1 ^{th0}	1'h1	
5			9								
5004000											efici.i
				7	ZhO						ľ
					0						
2		22									
-		—F	-	7		-	_		-		
		-		٢							
Ξ				4							
- -											
				Γ							
3				5							
2		_									
2				T	T	T				T	
3				٦							
				۲							
		_		L		-	_		-		
2	Receiving 8'h55 over the RX line										
2	ving X line		8h55	Г		٦	8.1				
-	e 8'h5		5	-		┙	_				
-	ove			L							
3	-										
-				Г							
-		-			Π	Γ				I	
				L							
-		-		-	$\left \right $	+					
8											
								-	-		
		-									
				4							
	Rece the R	2.5									
3	Receiving 8'h39 over the RX line		8'h39								
	8 h3										
E	9 ove										
				-			-	-			

Driver Testbench

VSIM 17> run -all
Testing baud rate 4800
test passed
Testing baud rate 9600
test passed
Testing baud rate 19200
test passed
Testing baud rate 38400
test passed
test passed
** Note: \$stop : I:/0school/ece554/miniproject1/driver_tb.v(108)
Time: 186 ns Iteration: 0 Instance: /driver_tb
Break in Module driver_tb at I:/0school/ece554/miniproject1/driver_tb.v line 108