Nang Thiri Wutyi
20113
11/27/2024

Verilog Lab #8

UART Transmitter Design Module

```
`timescale 1ns/1ps

module UART(
    input clk, rst_i,
    input byteReady, TByte, bitCount_r,
    output reg loadXMTShiftreg_o,
    output reg start_o,
    output reg shift_o,
    output reg clear_o
);
    // State encoding
    parameter pIdle    = 3'b001;
    parameter pWaiting = 3'b010;
    parameter pSending = 3'b100;

    // State registers
    reg [2:0] curSt, nxtSt;

    // State transition logic
    always @(posedge clk or posedge rst_i) begin
        if (rst_i)
            curSt <= pIdle; // Initialize to Idle state on reset
        else
            curSt <= nxtSt; // Transition to next state
    end

    // Combinational logic for next state and outputs
    always @(*) begin
        // Default output values
        loadXMTShiftreg_o = 1'b0;
        start_o           = 1'b0;
        shift_o           = 1'b0;
        clear_o           = 1'b0;

        case (curSt)
```

```
            // Idle state
            pIdle: begin
               if (byteReady) begin
                  loadXMTShiftreg_o = 1'b1;
                  nxtSt = pWaiting;
               end else
                  nxtSt = pIdle;
            end

            // Waiting state
            pWaiting: begin
               if (TByte) begin
                  start_o = 1'b1;
                  nxtSt = pSending;
               end else
                  nxtSt = pWaiting;
            end

            // Sending state
            pSending: begin
               if (bitCount_r) begin
                  clear_o = 1'b1;
                  nxtSt = pIdle;
               end else begin
                  shift_o = 1'b1;
                  nxtSt = pSending;
               end
            end

            // Default case for safety
            default: nxtSt = pIdle;
         endcase
      end
endmodule
```

Testbench Module

```
`timescale 1ns/1ps

module UART_tb;
   // Testbench signals
```

```
reg clk, rst_i;
reg byteReady, TByte, bitCount_r;
wire loadXMTShiftreg_o, start_o, shift_o, clear_o;

// Instantiate the UART module
UART uut (
    .clk(clk),
    .rst_i(rst_i),
    .byteReady(byteReady),
    .TByte(TByte),
    .bitCount_r(bitCount_r),
    .loadXMTShiftreg_o(loadXMTShiftreg_o),
    .start_o(start_o),
    .shift_o(shift_o),
    .clear_o(clear_o
));

// Clock generation (50 MHz)
initial clk = 0;
always #10 clk = ~clk; // 20 ns clock period

// Test stimulus
initial begin
    // Initialize inputs
    rst_i = 1;
    byteReady = 0;
    TByte = 0;
    bitCount_r = 0;

    // Apply reset
    #40;
    rst_i = 0;

    // Test case 1: Transition from pIdle to pWaiting
    #20;
    byteReady = 1; // Data ready
    #40;
    byteReady = 0; // Ensure no repeated transition

    // Test case 2: Transition from pWaiting to pSending
```

```
        #40;
        TByte = 1; // Start transmission
        #40;
        TByte = 0; // Ensure no repeated transition

        // Test case 3: Simulate shifting bits in pSending
        #80;
        bitCount_r = 1; // All bits sent
        #40;
        bitCount_r = 0; // Reset bit counter for further tests

        // Finish simulation
        #100;
        $stop;
    end

    // Monitor signals
    initial begin
        $monitor(
            "Time=%0t | State=%b | byteReady=%b | TByte=%b | bitCount_r=%b |
loadXMTShiftreg_o=%b | start_o=%b | shift_o=%b | clear_o=%b",
            $time, uut.curSt, byteReady, TByte, bitCount_r, loadXMTShiftreg_o, start_o, shift_o,
clear_o
        );
    end
endmodule
```

Output

```
[2024-11-28 02:06:34 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out
Time=0 | State=001 | byteReady=0 | TByte=0 | bitCount_r=0 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=0 | clear_o=0
Time=60000 | State=001 | byteReady=1 | TByte=0 | bitCount_r=0 | loadXMTShiftreg_o=1 |
start_o=0 | shift_o=0 | clear_o=0
Time=70000 | State=010 | byteReady=1 | TByte=0 | bitCount_r=0 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=0 | clear_o=0
Time=100000 | State=010 | byteReady=0 | TByte=0 | bitCount_r=0 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=0 | clear_o=0
Time=140000 | State=010 | byteReady=0 | TByte=1 | bitCount_r=0 | loadXMTShiftreg_o=0 |
start_o=1 | shift_o=0 | clear_o=0
Time=150000 | State=100 | byteReady=0 | TByte=1 | bitCount_r=0 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=1 | clear_o=0
Time=180000 | State=100 | byteReady=0 | TByte=0 | bitCount_r=0 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=1 | clear_o=0
```

```
Time=260000 | State=100 | byteReady=0 | TByte=0 | bitCount_r=1 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=0 | clear_o=1
Time=270000 | State=001 | byteReady=0 | TByte=0 | bitCount_r=1 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=0 | clear_o=0
Time=300000 | State=001 | byteReady=0 | TByte=0 | bitCount_r=0 | loadXMTShiftreg_o=0 |
start_o=0 | shift_o=0 | clear_o=0
testbench.sv:58: $stop called at 400000 (1ps)
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 400000 ticks.
```

UART Receiver Module

```
`timescale 1ns/1ps

module UARTRx (
    input clk,            // System clock
    input rst_i,          // Reset signal
    input serialIn,       // Serial data input
    input sampleClk,      // Sample clock (typically 8x baud rate)
    output reg [7:0] dataOut,  // Parallel data output (8 bits)
    output reg dataReady,  // Data ready flag to indicate new data
    output reg err1,      // Error flag 1 (indicates if not ready)
    output reg err2       // Error flag 2 (indicates missing stop bit)
);
    // States
    parameter IDLE    = 3'b000;
    parameter RECEIVE = 3'b001;
    parameter SHIFT   = 3'b010;
    parameter DONE    = 3'b011;

    reg [2:0] curState, nextState;  // Current and next states
    reg [3:0] bitCount;           // 4-bit counter for data bits (up to 8)
    reg [7:0] shiftReg;           // Shift register to store received data

    // State machine for receiving data
    always @(posedge sampleClk or posedge rst_i) begin
        if (rst_i) begin
            curState <= IDLE;
            bitCount <= 0;
            dataOut <= 8'b0;
            dataReady <= 0;
            shiftReg <= 8'b0;
            err1 <= 0;
```

```
        err2 <= 0;
    end else begin
        curState <= nextState;
    end
end

// Next state and output logic
always @(*) begin
    nextState = curState;
    dataReady = 0;
    err1 = 0;
    err2 = 0;

    case (curState)
        IDLE: begin
            if (serialIn == 0) begin  // Start bit detected
                nextState = RECEIVE;
                bitCount = 0;
            end else begin
                err1 = 1;  // Error flag when not in IDLE state
            end
        end

        RECEIVE: begin
            shiftReg = {serialIn, shiftReg[7:1]};  // Shift in the data
            if (bitCount == 7) begin
                nextState = DONE;
            end else begin
                bitCount = bitCount + 1;
                nextState = RECEIVE;
            end
        end

        DONE: begin
            if (serialIn == 1) begin  // Stop bit check
                dataOut = shiftReg;
                dataReady = 1;  // New data ready
                nextState = IDLE;  // Transition to IDLE after receiving data
            end else begin
                err2 = 1;  // Stop bit error
```

```
                nextState = IDLE;  // Transition to IDLE after error
            end
        end

        default: nextState = IDLE;  // Safety net for default case
    endcase
  end
endmodule
```

Testbench Module

```
`timescale 1ns/1ps

module UARTRx_tb;
  // Testbench signals
  reg clk, rst_i;
  reg serialIn;          // Serial input data
  reg sampleClk;         // Sample clock
  wire [7:0] dataOut;    // Parallel data output
  wire dataReady;        // Data ready flag
  wire err1, err2;       // Error flags

  // Instantiate the UART Receiver module from design.sv
  UARTRx uut (
    .clk(clk),
    .rst_i(rst_i),
    .serialIn(serialIn),
    .sampleClk(sampleClk),
    .dataOut(dataOut),
    .dataReady(dataReady),
    .err1(err1),
    .err2(err2)
  );

  // Clock generation (50 MHz for system clock)
  initial clk = 0;
  always #10 clk = ~clk; // 20 ns period (50 MHz)

  // Clock generation for sample clock (8x baud rate, e.g., 400 MHz)
  initial sampleClk = 0;
  always #1 sampleClk = ~sampleClk; // 2 ns period (400 MHz)
```

```
// Test stimulus
initial begin
    // Initialize inputs
    rst_i = 1;
    serialIn = 1;  // Idle high
    #40 rst_i = 0;  // Release reset

    // Test case 1: Start bit followed by 8 data bits
    #20 serialIn = 0;  // Start bit
    #20 serialIn = 1;  // Data bit 1
    #20 serialIn = 0;  // Data bit 2
    #20 serialIn = 1;  // Data bit 3
    #20 serialIn = 0;  // Data bit 4
    #20 serialIn = 1;  // Data bit 5
    #20 serialIn = 0;  // Data bit 6
    #20 serialIn = 1;  // Data bit 7
    #20 serialIn = 1;  // Stop bit

    // Wait for dataReady signal
    #40;

    // Test case 2: Start bit with incorrect stop bit (Err2 should be set)
    #20 serialIn = 0;  // Start bit
    #20 serialIn = 1;  // Data bit 1
    #20 serialIn = 0;  // Data bit 2
    #20 serialIn = 1;  // Data bit 3
    #20 serialIn = 0;  // Data bit 4
    #20 serialIn = 1;  // Data bit 5
    #20 serialIn = 0;  // Data bit 6
    #20 serialIn = 1;  // Data bit 7
    #20 serialIn = 0;  // Incorrect stop bit (should be 1)

    // Test case 3: Start bit with no ready signal (Err1 should be set)
    #20 serialIn = 0;  // Start bit
    #20 serialIn = 1;  // Data bit 1
    #20 serialIn = 0;  // Data bit 2
    #20 serialIn = 1;  // Data bit 3
    #20 serialIn = 0;  // Data bit 4
    #20 serialIn = 1;  // Data bit 5
```

```
        #20 serialIn = 0;  // Data bit 6
        #20 serialIn = 1;  // Data bit 7
        #20 serialIn = 1;  // Stop bit

        // Finish simulation
        #100 $stop;
    end

    // Monitor signals
    initial begin
        $monitor(
            "Time=%0t | DataOut=%b | DataReady=%b | Err1=%b | Err2=%b",
            $time, dataOut, dataReady, err1, err2
        );
    end
endmodule
```

Output

```
[2024-11-28 02:38:13 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out
Time=0 | DataOut=00000000 | DataReady=0 | Err1=1 | Err2=0
Time=1000 | DataOut=00000000 | DataReady=0 | Err1=0 | Err2=0
Time=201000 | DataOut=10101010 | DataReady=1 | Err1=0 | Err2=0
Time=203000 | DataOut=10101010 | DataReady=0 | Err1=1 | Err2=0
Time=280000 | DataOut=10101010 | DataReady=0 | Err1=0 | Err2=0
Time=421000 | DataOut=10101010 | DataReady=1 | Err1=0 | Err2=0
Time=423000 | DataOut=10101010 | DataReady=0 | Err1=1 | Err2=0
Time=440000 | DataOut=10101010 | DataReady=0 | Err1=0 | Err2=0
Time=601000 | DataOut=10101010 | DataReady=1 | Err1=0 | Err2=0
Time=603000 | DataOut=10101010 | DataReady=0 | Err1=1 | Err2=0
testbench.sv:76: $stop called at 720000 (1ps)
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 720000 ticks.
Done
```