UART Communication using SystemVerilog

Universal Asynchronous Receiver-Transmitter Implementation

SystemVerilog Implementation • Project Report • GitHub Ready

Project Overview

This project implements a UART (Universal Asynchronous Receiver-Transmitter) communication module using SystemVerilog. The design features a finite state machine-based approach for reliable serial data transmission with configurable baud rate.

Key Features

- 4-State FSM Design
- Configurable Baud Rate
- 8-bit Data Transmission
- Start/Stop Bit Protocol

Technical Specifications

- SystemVerilog HDL
- Synchronous Reset
- Clock Domain Crossing
- State-based Control

Module Interface

UART Module Ports

Input Ports

`clk`	System clock input
`rst`	Reset signal (active high)
`data[0:7]`	8-bit parallel data input

Output Ports

`tx`	Transmission enable signal
`tx_out`	Serial data output line

State Machine Design

State Transitions

- `IDLE → START`
- `START → DATA`
- `DATA → STOP` (after 8 bits)
- `STOP → IDLE`

Code Analysis

UART Implementation

```
module uart(input clk,rst,
  input [0:7]data,
  output logic tx,
  output logic tx_out
  );
  parameter baud_rate=1;
  typedef enum logic [1:0] {
  IDLE,
  START,
  DATA,
  STOP
  } states;

  states state,next_state;
  logic [0:7] tx_buff;
  logic [5:0] count;
  logic  [2:0]bit_in;
  logic g_clk;
```

```
always_ff @(posedge clk or posedge rst) begin
  if(rst) begin
    tx<=0; tx_out<=0; tx_buff<=0; count<=0;
    bit_in <= 0; state<=IDLE; next_state <= IDLE; g_clk = 0;
  end
  else
    count = count + 1;
  if(count == baud_rate) begin
    g_clk <= !g_clk; count <= 0; state <= next_state;
  end
end
```

```
always_ff @(posedge g_clk) begin
  case(state)
    IDLE: begin
      tx<=0; tx_buff <= 0; tx_out <= 0; bit_in <= 0; next_state <= START;
    end
    START: begin
      tx<=0; tx_buff <= data; next_state <= DATA;
    end
    DATA: begin
      tx <= 1; tx_out <= tx_buff[bit_in]; bit_in <= bit_in+1;
      if(bit_in == 7) next_state <= STOP;
    end
    STOP: begin
      tx <= 0; tx_out <= 0; tx_buff<=0; next_state <= IDLE;
    end
  endcase
end
```

Code Review & Improvements

Identified Issues

- **Blocking Assignment in Clocked Block**
  Using = instead of <= for count
- **Low Baud Rate Parameter**
  baud_rate=1 is too low for practical use
- **Missing Start Bit**
  No proper start bit transmission (should be logic 0)
- **Missing Stop Bit**
  No proper stop bit transmission (should be logic 1)

Suggested Improvements

- **Use Non-blocking Assignments**
  Change `count = count + 1` to `count <= count + 1`
- **Add Data Valid Signal**
  Include input signal to trigger transmission
- **Implement Proper UART Protocol**
  Add start bit (0) and stop bit (1) transmission
- **Add Transmission Complete Flag**
  Signal when transmission is finished

UART Timing Protocol

Standard UART transmission format for 8-bit data with 1 start bit and 1 stop bit:

```
UART Frame Structure:

| IDLE | START | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | STOP | IDLE | | 1 | 0 | b0 | b1 | b2 | b3 | b4
| b5 | b6 | b7 | 1 | 1 |
```

* D0-D7: Data bits (LSB first)
* Each bit duration = 1/baud_rate

Usage Instructions

Simulation Setup

1. Instantiate the UART module in your testbench
2. Configure the baud_rate parameter (e.g., 9600, 115200)
3. Apply clock and reset signals
4. Provide 8-bit data input and monitor tx_out

Synthesis Guidelines

- Suitable for FPGA implementation
- Requires external clock source
- Minimal resource utilization
- Adjust baud_rate for target frequency

Project Structure

```
uart_project/
  uart.sv // Main UART module
  uart_tb.sv // Testbench file
  README.md // Project documentation
  constraints.xdc // FPGA constraints (optional)
```

Conclusion

This UART implementation provides a solid foundation for serial communication in SystemVerilog. While the current version demonstrates the basic concepts of state machine-based UART design, several improvements are recommended for production use.

Future Enhancements:

- Add UART receiver functionality
- Add parity bit support
- Error detection and handling
- Implement FIFO buffers
- Flow control (RTS/CTS)
- Configurable data width