**TITLE:** Design and Implement the UART FIFO protocol.

UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol used for serial communication between devices. It allows the transmission and reception of data one bit at a time over a single wire. UART is asynchronous, meaning it does not require a clock line to synchronize the transmitter and receiver.

UART communication requires two devices:

* Transmitter (TX): Sends data
* Receiver (RX): Receives data

Steps:

1. The transmitter converts parallel data (usually 8 bits) from the CPU into a serial stream.
2. The receiver converts the serial data back into parallel form.
3. Both sides must agree on:
   * Baud rate: Speed of communication (e.g., 9600, 115200 bps)
   * Data bits: Usually 8 bits
   * Stop bits: 1 or 2 bits to indicate end of data
   * Parity bit (optional): Error detection

**How UART used in Industries:**

UART might seem simple, but it’s one of the most widely used communication protocols in industrial systems because of its simplicity, reliability, and versatility.

UART provides point to point serial communication between two devices like sensors, controllers and monitoring system

In industry, it’s often used for:

* Machine-to-controller communication
* Debugging and diagnostics
* Firmware updates
* Data logging
* Sensor/actuator interfacing

**My idea:**

The general UART has single bit Transmitter and receiver now I was developing the transmitter FIFO and receiver FIFO.

* Data comes faster than the CPU or logic can handle.
* You need to stream multiple bytes continuously.
* A **FIFO (First-In, First-Out)** buffer solves this by storing several bytes temporarily — just like a queue.
* **TX FIFO:** holds outgoing bytes until TX engine transmits them.
* **RX FIFO:** holds received bytes until CPU reads them.

**Benefits:**

* Prevents data loss during bursts.
* Smooths timing mismatches between TX/RX and CPU.
* Mimics how real UART IPs (e.g., 16550 UART) work — with 16-byte FIFOs.

**Testbench:**

timescale 1us/1ns

module uart\_fifo\_tb;

reg clk, reset;

reg [7:0] tx\_data;

reg tx\_wr\_en, rx\_rd\_en;

wire tx\_full, tx, rx\_empty;

wire [7:0] rx\_data;

// Loopback connection (TX → RX)

assign rx = tx;

uart\_fifo\_top uut (

.clk(clk),

.reset(reset),

.tx\_data\_in(tx\_data),

.tx\_wr\_en(tx\_wr\_en),

.tx\_full(tx\_full),

.tx(tx),

.rx(rx),

.rx\_rd\_en(rx\_rd\_en),

.rx\_data\_out(rx\_data),

.rx\_empty(rx\_empty)

);

// Generate 1 MHz clock (1 µs period)

always #0.5 clk = ~clk;

initial begin

clk = 0;

reset = 1;

tx\_data = 0;

tx\_wr\_en = 0;

rx\_rd\_en = 0;

#10 reset = 0;

// Send multiple bytes

repeat (5) begin

@(negedge clk);

tx\_data = $random;

tx\_wr\_en = 1;

@(negedge clk);

tx\_wr\_en = 0;

#1000; // Wait a bit between sends

end

// Read all received data

#100000; // Wait for all data to be received

while (!rx\_empty) begin

rx\_rd\_en = 1;

@(negedge clk);

rx\_rd\_en = 0;

@(posedge clk);

$display("Received Byte: %h", rx\_data);

end

#1000;

$display("✅ UART FIFO Test Completed Successfully!");

$finish;

end

endmodule

**RTL Code:**

`timescale 1ns/1ps

// =======================================================

// UART with FIFO Buffers

// =======================================================

module uart\_fifo\_top #(

parameter CLOCK\_FREQ = 50000000,

parameter BAUD\_RATE = 9600,

parameter FIFO\_DEPTH = 16

)(

input wire clk,

input wire reset,

// TX interface

input wire [7:0] tx\_data\_in,

input wire tx\_wr\_en,

output wire tx\_full,

output wire tx,

// RX interface

input wire rx,

input wire rx\_rd\_en,

output wire [7:0] rx\_data\_out,

output wire rx\_empty

);

wire tick;

wire [7:0] tx\_fifo\_data;

wire tx\_fifo\_empty;

wire tx\_fifo\_rd;

wire tx\_busy;

wire [7:0] rx\_fifo\_data;

wire rx\_fifo\_full;

wire rx\_fifo\_wr;

// ------------------------------

// Baud Rate Generator

// ------------------------------

localparam integer DIV\_COUNT = CLOCK\_FREQ / (BAUD\_RATE \* 16);

reg [15:0] count;

reg tick\_reg;

assign tick = tick\_reg;

always @(posedge clk or posedge reset) begin

if (reset) begin

count <= 0;

tick\_reg <= 0;

end else if (count == DIV\_COUNT) begin

count <= 0;

tick\_reg <= 1;

end else begin

count <= count + 1;

tick\_reg <= 0;

end

end

// ------------------------------

// TX FIFO

// ------------------------------

fifo #(.DATA\_WIDTH(8), .DEPTH(FIFO\_DEPTH)) tx\_fifo (

.clk(clk), .reset(reset),

.wr\_en(tx\_wr\_en), .rd\_en(tx\_fifo\_rd),

.din(tx\_data\_in), .dout(tx\_fifo\_data),

.full(tx\_full), .empty(tx\_fifo\_empty)

);

// ------------------------------

// RX FIFO

// ------------------------------

fifo #(.DATA\_WIDTH(8), .DEPTH(FIFO\_DEPTH)) rx\_fifo (

.clk(clk), .reset(reset),

.wr\_en(rx\_fifo\_wr), .rd\_en(rx\_rd\_en),

.din(rx\_fifo\_data), .dout(rx\_data\_out),

.full(rx\_fifo\_full), .empty(rx\_empty)

);

// ------------------------------

// UART Transmitter

// ------------------------------

uart\_tx tx\_unit (

.clk(clk), .reset(reset), .tick(tick),

.fifo\_data(tx\_fifo\_data), .fifo\_empty(tx\_fifo\_empty),

.fifo\_rd\_en(tx\_fifo\_rd),

.tx(tx), .busy(tx\_busy)

);

// ------------------------------

// UART Receiver

// ------------------------------

uart\_rx rx\_unit (

.clk(clk), .reset(reset), .tick(tick),

.rx(rx),

.fifo\_full(rx\_fifo\_full),

.fifo\_data(rx\_fifo\_data),

.fifo\_wr\_en(rx\_fifo\_wr)

);

endmodule

// =======================================================

// FIFO Module

// =======================================================

module fifo #(parameter DATA\_WIDTH=8, parameter DEPTH=16)(

input clk, reset,

input wr\_en, rd\_en,

input [DATA\_WIDTH-1:0] din,

output reg [DATA\_WIDTH-1:0] dout,

output reg full, empty

);

reg [DATA\_WIDTH-1:0] mem [0:DEPTH-1];

reg [$clog2(DEPTH):0] wr\_ptr, rd\_ptr, count;

always @(posedge clk or posedge reset) begin

if (reset) begin

wr\_ptr <= 0; rd\_ptr <= 0; count <= 0;

full <= 0; empty <= 1;

end else begin

if (wr\_en && !full) begin

mem[wr\_ptr] <= din;

wr\_ptr <= wr\_ptr + 1;

count <= count + 1;

end

if (rd\_en && !empty) begin

dout <= mem[rd\_ptr];

rd\_ptr <= rd\_ptr + 1;

count <= count - 1;

end

full <= (count == DEPTH);

empty <= (count == 0);

end

end

endmodule

// =======================================================

// UART Transmitter with FIFO interface

// =======================================================

module uart\_tx(

input clk, reset, tick,

input [7:0] fifo\_data,

input fifo\_empty,

output reg fifo\_rd\_en,

output reg tx, busy

);

reg [3:0] bit\_count;

reg [9:0] shift\_reg;

always @(posedge clk or posedge reset) begin

if (reset) begin

tx <= 1'b1;

busy <= 0;

fifo\_rd\_en <= 0;

bit\_count <= 0;

shift\_reg <= 0;

end else begin

fifo\_rd\_en <= 0;

if (!busy && !fifo\_empty) begin

fifo\_rd\_en <= 1;

shift\_reg <= {1'b1, fifo\_data, 1'b0};

busy <= 1;

bit\_count <= 0;

end else if (busy && tick) begin

tx <= shift\_reg[0];

shift\_reg <= shift\_reg >> 1;

bit\_count <= bit\_count + 1;

if (bit\_count == 9) busy <= 0;

end

end

end

endmodule

// =======================================================

// UART Receiver with FIFO interface

// =======================================================

module uart\_rx(

input clk, reset, tick,

input rx,

input fifo\_full,

output reg [7:0] fifo\_data,

output reg fifo\_wr\_en

);

reg [3:0] bit\_count;

reg [7:0] data\_buf;

reg [1:0] state;

localparam IDLE = 0, START = 1, DATA = 2, STOP = 3;

always @(posedge clk or posedge reset) begin

if (reset) begin

state <= IDLE;

bit\_count <= 0;

fifo\_wr\_en <= 0;

end else begin

fifo\_wr\_en <= 0;

case (state)

IDLE: if (~rx) state <= START;

START: if (tick) begin

bit\_count <= 0;

state <= DATA;

end

DATA: if (tick) begin

data\_buf <= {rx, data\_buf[7:1]};

bit\_count <= bit\_count + 1;

if (bit\_count == 7)

state <= STOP;

end

STOP: if (tick) begin

if (rx && !fifo\_full) begin

fifo\_data <= data\_buf;

fifo\_wr\_en <= 1;

end

state <= IDLE;

end

endcase

end

end

endmodule