**BÁO CÁO THỰC HÀNH THIẾT KẾ VI MẠCH**

***CHỦ ĐỀ: UART***

Module baud\_rate\_gen

“module baud\_rate\_gen(input wire clk\_50m,

output wire rxclk\_en,

output wire txclk\_en);”

- Mục đích: Tạo tín hiệu bật tần số xung cho RX và TX ở tốc độ baud cụ thể.

- Input: `clk\_50m` là xung nhịp 50 MHz.

- Output: `rxclk\_en` và `txclk\_en` là các tín hiệu bật tần số cho nhận và truyền.

“parameter RX\_ACC\_MAX = 50000000 / (115200 \* 16);

parameter TX\_ACC\_MAX = 50000000 / 115200;”

- RX\_ACC\_MAX: Số chu kỳ xung nhịp cần để bật RX ở tốc độ baud 115200, lấy mẫu 16 lần.

- TX\_ACC\_MAX: Số chu kỳ xung nhịp cần để bật TX ở tốc độ baud 115200.





*“parameter RX\_ACC\_WIDTH = $clog2(RX\_ACC\_MAX);*

*parameter TX\_ACC\_WIDTH = $clog2(TX\_ACC\_MAX);”*

- Tính toán độ rộng bit cần thiết cho các bộ đếm dựa trên giá trị tối đa.



“reg [RX\_ACC\_WIDTH - 1:0] rx\_acc = 0;

reg [TX\_ACC\_WIDTH - 1:0] tx\_acc = 0;”

- rx\_acc và tx\_acc: Bộ đếm cho bật tần số RX và TX.

*“assign rxclk\_en = (rx\_acc == 5'd0);*

assign txclk\_en = (tx\_acc == 9'd0);”

- Bật `rxclk\_en` và `txclk\_en` khi bộ đếm tương ứng reset.

“always @(posedge clk\_50m) begin

if (rx\_acc == RX\_ACC\_MAX[RX\_ACC\_WIDTH - 1:0])

rx\_acc <= 0;

else

rx\_acc <= rx\_acc + 5'b1;

end”

- Tăng `rx\_acc` mỗi chu kỳ xung, reset khi đạt `RX\_ACC\_MAX`.

“always @(posedge clk\_50m) begin

if (tx\_acc == TX\_ACC\_MAX[TX\_ACC\_WIDTH - 1:0])

tx\_acc <= 0;

else

tx\_acc <= tx\_acc + 9'b1;

end”

- Tăng `tx\_acc` mỗi chu kỳ xung, reset khi đạt `TX\_ACC\_MAX`.



Đảm bảo cho việc nhận data bên phía receiver.

Module Receiver

module receiver(input wire rx,

output reg rdy,

input wire rdy\_clr,

input wire clk\_50m,

input wire clken,

output reg [7:0] data);

- Mục đích: Nhận dữ liệu nối tiếp và xuất dữ liệu song song.

- Đầu vào/Đầu ra: `rx` là đầu vào nối tiếp. `rdy` báo hiệu dữ liệu đã sẵn sàng. `data` là đầu ra 8-bit.

initial begin

rdy = 0;

data = 8'b0;

end

- Khởi tạo `rdy` và `data`.

parameter RX\_STATE\_START = 2'b00;

parameter RX\_STATE\_DATA = 2'b01;

parameter RX\_STATE\_STOP = 2'b10;

- Định nghĩa các trạng thái cho máy trạng thái nhận.

reg [1:0] state = RX\_STATE\_START;

reg [3:0] sample = 0;

reg [3:0] bitpos = 0;

reg [7:0] scratch = 8'b0;

- state: Theo dõi trạng thái hiện tại của máy trạng thái.

- sample: Đếm mẫu cho mỗi bit.

- bitpos: Theo dõi vị trí bit trong byte.

- scratch: Lưu trữ tạm thời dữ liệu nhận được.

always @(posedge clk\_50m) begin

if (rdy\_clr)

rdy <= 0;

- Xóa `rdy` khi `rdy\_clr` được kích hoạt.

if (clken) begin

case (state)

RX\_STATE\_START: begin

if (!rx || sample != 0)

sample <= sample + 4'b1;

if (sample == 15) begin

state <= RX\_STATE\_DATA;

bitpos <= 0;

sample <= 0;

scratch <= 0;

end

end

- RX\_STATE\_START: Phát hiện bit bắt đầu. Chuyển sang `RX\_STATE\_DATA` sau 16 mẫu (thời gian một bit).

RX\_STATE\_DATA: begin

sample <= sample + 4'b1;

if (sample == 4'h8) begin

scratch[bitpos[2:0]] <= rx;

bitpos <= bitpos + 4'b1;

end

if (bitpos == 8 && sample == 15)

state <= RX\_STATE\_STOP;

end

- RX\_STATE\_DATA: Lấy mẫu các bit dữ liệu ở giữa mỗi chu kỳ bit. Chuyển sang `RX\_STATE\_STOP` sau 8 bit.

RX\_STATE\_STOP: begin

if (sample == 15 || (sample >= 8 && !rx)) begin

state <= RX\_STATE\_START;

data <= scratch;

rdy <= 1'b1;

sample <= 0;

end else begin

sample <= sample + 4'b1;

end

end

- RX\_STATE\_STOP: Kiểm tra bit dừng. Nếu hợp lệ, đặt `data` và `rdy`, sau đó quay lại `RX\_STATE\_START`.

default: begin

state <= RX\_STATE\_START;

end

endcase

end

end

- Trạng thái mặc định đảm bảo máy trạng thái reset về `RX\_STATE\_START` nếu cần thiết.

Module Transmitter

module transmitter(input wire [7:0] din,

input wire wr\_en,

input wire clk\_50m,

input wire clken,

output reg tx,

output wire tx\_busy);

- Mục đích: Truyền dữ liệu song song dưới dạng nối tiếp.

- Đầu vào/Đầu ra: `din` là dữ liệu đầu vào 8-bit. `tx` là đầu ra nối tiếp. `tx\_busy` chỉ báo hoạt động truyền.

initial begin

tx = 1'b1;

end

- Khởi tạo `tx` ở trạng thái nhàn (cao).

parameter STATE\_IDLE = 2'b00;

parameter STATE\_START = 2'b01;

parameter STATE\_DATA = 2'b10;

parameter STATE\_STOP = 2'b11;

- Định nghĩa các trạng thái cho máy trạng thái truyền.

reg [7:0] data = 8'h00;

reg [2:0] bitpos = 3'h0;

reg [1:0] state = STATE\_IDLE;

- data: Lưu trữ dữ liệu cần truyền.

- bitpos: Theo dõi vị trí bit hiện tại.

- state: Theo dõi trạng thái hiện tại của máy trạng thái.

always @(posedge clk\_50m) begin

case (state)

STATE\_IDLE: begin

if (wr\_en) begin

state <= STATE\_START;

data <= din;

bitpos <= 3'h0;

end

end

- STATE\_IDLE: Đợi `wr\_en`. Tải `din` vào `data` và chuyển sang `STATE\_START`, .

STATE\_START: begin

if (clken) begin

tx <= 1'b0;

state <= STATE\_DATA;

end

end

- STATE\_START: Gửi bit bắt đầu (thấp) và chuyển sang `STATE\_DATA`.

STATE\_DATA: begin

if (clken) begin

if (bitpos == 3'h7)

state <= STATE\_STOP;

else

bitpos <= bitpos + 3'h1;

tx <= data[bitpos];

end

end

- STATE\_DATA: Gửi các bit dữ liệu, chuyển sang `STATE\_STOP` sau bit cuối cùng.

STATE\_STOP: begin

if (clken) begin

tx <= 1'b1;

state <= STATE\_IDLE;

end

end

- STATE\_STOP: Gửi bit dừng (cao) và quay lại `STATE\_IDLE`.

default: begin

tx <= 1'b1;

state <= STATE\_IDLE;

end

endcase

end

- Trạng thái mặc định đảm bảo máy trạng thái reset về `STATE\_IDLE` nếu cần thiết.

assign tx\_busy = (state != STATE\_IDLE);

- Đặt `tx\_busy` khi máy trạng thái không ở `STATE\_IDLE`.

Lab 6: Logic-level synthesis tool (Design Compiler)

Transmitter:

always @(posedge clk\_50m) begin

case (state)

STATE\_IDLE: begin

if (wr\_en) begin

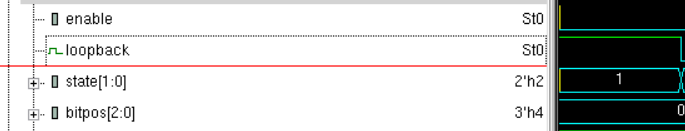
state <= STATE\_START;

data <= din;

bitpos <= 3'h0;

end

end



STATE\_START: begin

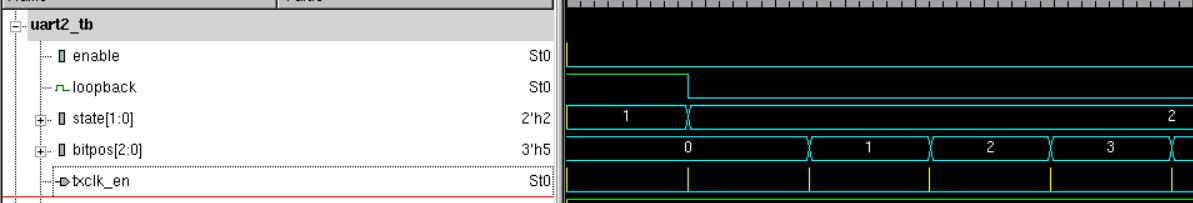
if (clken) begin

tx <= 1'b0;

state <= STATE\_DATA;

end

end



Receiver:

if (clken) begin

case (state)

RX\_STATE\_START: begin

if (!rx || sample != 0)

sample <= sample + 4'b1;

if (sample == 15) begin

state <= RX\_STATE\_DATA;

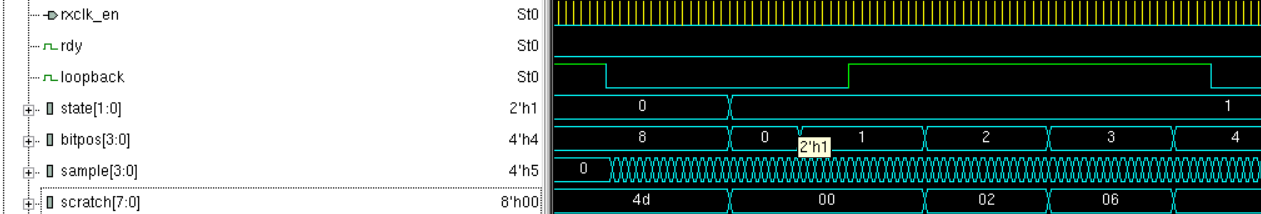
bitpos <= 0;

sample <= 0;

scratch <= 0;

end

end



Transmitter:

STATE\_DATA: begin

if (clken) begin

if (bitpos == 3'h7)

state <= STATE\_STOP;

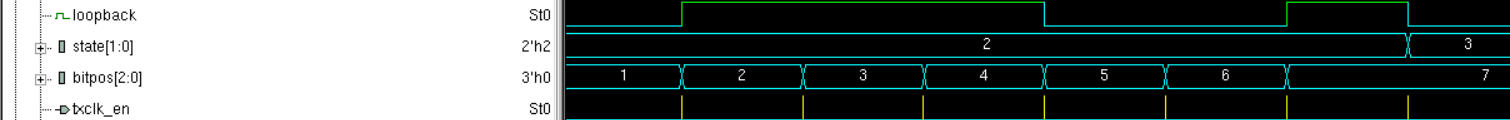
else

bitpos <= bitpos + 3'h1;

tx <= data[bitpos];

end

end



Receiver:

RX\_STATE\_DATA: begin

sample <= sample + 4'b1;

if (sample == 4'h8) begin

scratch[bitpos[2:0]] <= rx;

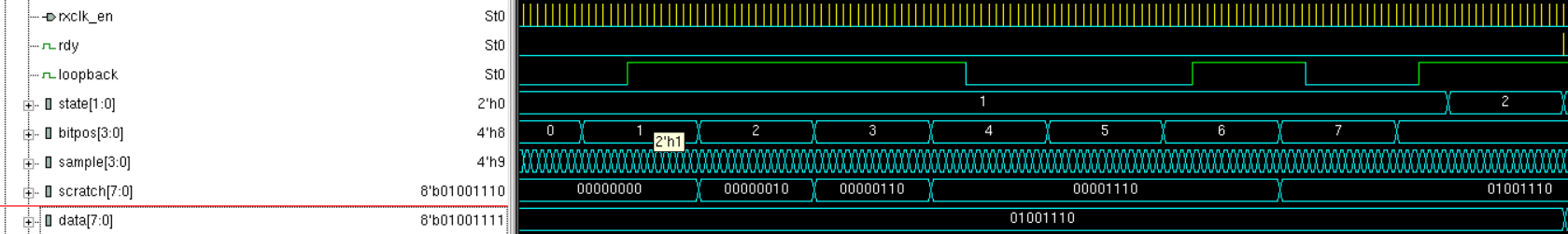
bitpos <= bitpos + 4'b1;

end

if (bitpos == 8 && sample == 15)

state <= RX\_STATE\_STOP;

end



Transmitter:

STATE\_STOP: begin

if (clken) begin

tx <= 1'b1;

state <= STATE\_IDLE;

end

end



Receiver:

RX\_STATE\_STOP: begin

if (sample == 15 || (sample >= 8 && !rx)) begin

state <= RX\_STATE\_START;

data <= scratch;

rdy <= 1'b1;

sample <= 0;

end else begin

sample <= sample + 4'b1;

end

end

