BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM
KHOA ĐIỆN – ĐIỆN TỬ

MÔN HỌC: THIẾT KẾ HỆ THỐNG VÀ VI MẠCH TÍCH HỢP

TIỂU LUẬN

UART RECEIVING SUBSYSTEM

GVHD: TS. PHẠM VĂN KHOA

SVTH:	MSSV
1. Võ Hoàng Quốc	19161278
2. Trần Thanh Ngọc	19119204
3. Phan Công Danh	19119160
4. Phạm Hải Nguyên	19119205
5. Võ Minh Hậu	19119174
6. Nguyễn Hà Nhật Linh	19119190

Lớp thứ 4 - Tiết 7-9

Mã môn học: 202ICSD336764

Mã lớp 11

Tp. Thủ Đức, tháng 6 năm 2021

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN

1.1. Đặt vấn đề:

Trong thời đại 4.0, dữ liệu được xem là vàng. Vậy nên việc truyền dữ liệu rất được chú trọng và phát triển. Sự phát triển của ngành công nghiệp bán dẫn, các IC ra đời phục vụ cho truyền dữ liệu ngày càng nhiều và hiệu quả ngày càng cao. Các IC ngày càng trở nên nhỏ hơn và nhanh hơn. Kết quả là giao tiếp song song trở thành nút thắt cổ chai trong quá trình giao tiếp giữa các thiết bị với việc sử dụng quá nhiều chân để truyền, nhận dữ liệu.

Dù truyền dữ liệu song song được sử dụng rộng rãi, nhưng truyền dữ liệu nối tiếp luôn có những ưu điểm khiến nó không thể bị thay thế. Các chuẩn giao tiếp nối tiếp thường được sử dụng là Ethernet, SPI, I2C… Nhưng bài báo cáo này sẽ tìm hiểu về chuẩn giao tiếp UART.

1.2. Mục tiêu:

-Tìm hiểu được code của UART Receiver

-Thông qua code mẫu hiểu được nguyên lý hoạt động của UART Receiver

-Thực hiện mô phỏng thành công trên phần mềm Xilinx ISE

1.3. Nội dung nghiên cứu:

-Nguyên lý hoạt động

-Cách thức nhận dữ liệu

CHƯƠNG 2: SƠ LƯỢC VỀ UART

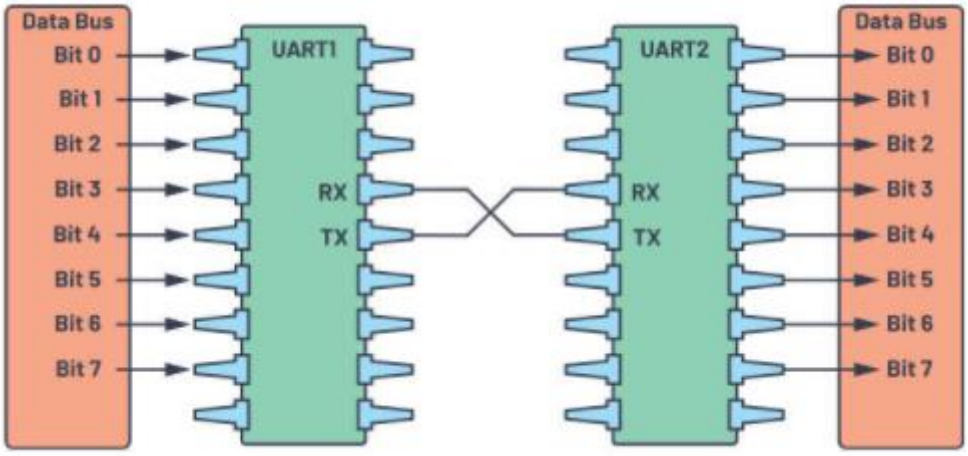
2.1. UART là gì?

UART (Universal Asynchronous Receiver-Transmitter) máy thu-phát không đồng bộ: là một thiết bị phần cứng máy tính dùng trong giao tiếp nối tiếp không đồng bộ, trong đó định dạng dữ liệu và tốc độ truyền có thể được định cấu hình. Nó gửi từng bit dữ liệu một, từ LSB đến MSB được đóng khung bởi *start bit* và *stop bit* để thời gian chính xác được xử lý bởi các kênh truyền thông (communication channel). Các mức tín hiệu điện được xử lý bởi một mạch điều khiển bên ngoài UART. Hai chuẩn tín hiệu phổ biến là RS-232 (12-volt system) và RS-485 (5-volt system).

Nó là một trong những thiết bị giao tiếp máy tính sớm nhất, được sử dụng để gắn máy điện báo (teletypewriters) cho bảng điều khiển (operator console).

UART thường là một (hoặc một phần) mạch tích hợp (IC) được sử dụng cho giao tiếp qua cổng nối tiếp của máy tính hoặc các thiết bị ngoại vi. Một hay nhiều thiết bị ngoại vi UART thường được tích hợp trong chip vi điều khiển (microcontroller chips). Các UART chuyên dụng được sử dụng cho ô tô, thẻ thông minh, SIM, …

Bên cạnh đó, bộ thu-phát đồng bộ và không đồng bộ (USART) cũng hỗ trợ cho UART.



Hình 1: Bộ thu-phát đồng bộ và không đồng bộ

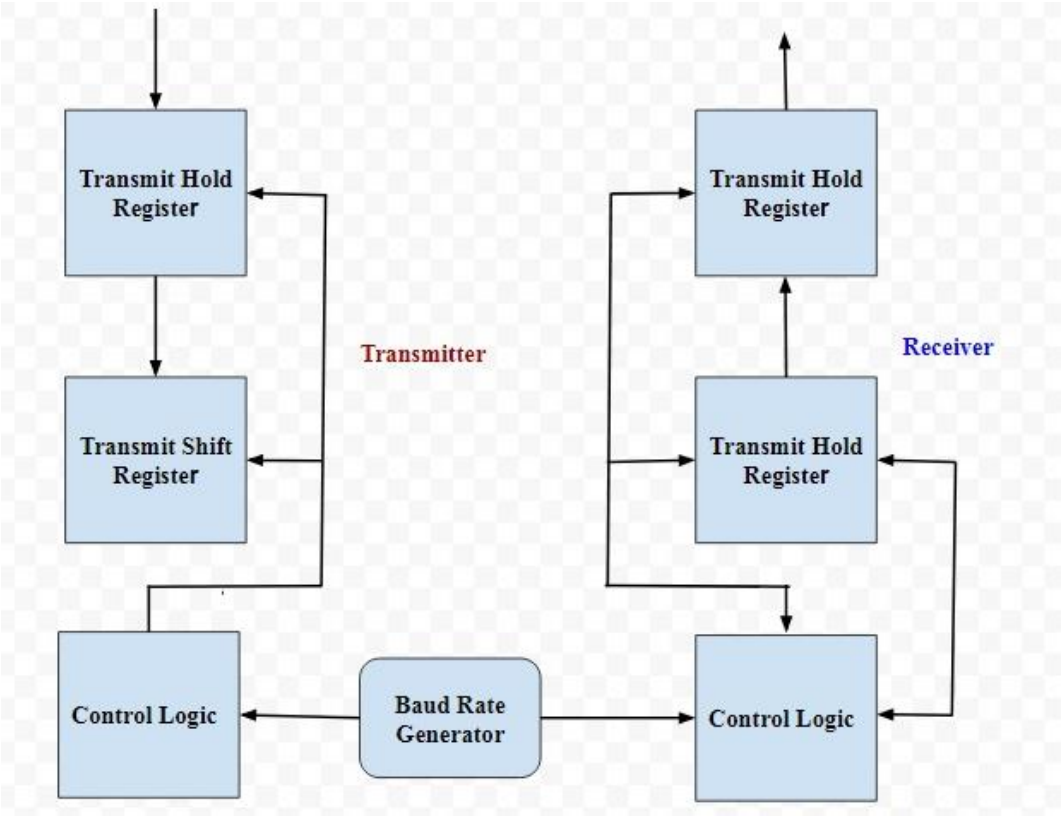
2.2 Sơ đồ khối của một UART

Sơ đồ khối UART bao gồm hai thành phần là máy phát và máy thu được hiển thị bên dưới. Phần máy phát bao gồm ba khối là thanh ghi giữ truyền, thanh ghi dịch chuyển và logic điều khiển. Về phần máy thu thì bao gồm hai thanh ghi giữ và logic điều khiển. Hai phần này thường được

cung cấp bởi một bộ tạo tốc độ baud. Trình tạo này được sử dụng để tạo tốc độ khi phần máy phát và phần máy thu phải truyền hoặc nhận dữ liệu.

Một UART bao gồm một hệ thống máy phát (Transmitter Subsystem) và một hệ thống máy thu (Receiver Subsystem). Máy phát về cơ bản là một thanh ghi dịch đặc biệt, tải dữ liệu song song và sau đó dịch nó ra từng bit với một tốc độ cụ thể. Máy thu dịch bit dữ liệu từng bit một và sau đó tập hợp lại dữ liệu. Dòng nối tiếp (serial line) là 1 khi nó ngưng hoạt động.

⇨ Ở đây nhóm sẽ trình bày về phần UART Receiving Subsystem.



Hình 2. Sơ đồ khối UART

CHƯƠNG 3: CƠ SỞ LÝ THUYẾT

Khi không có thông tin xung clock từ tín hiệu đã truyền, phía máy nhận có thể khôi phục các bit dữ liệu bằng cách sử dụng các tham số định trước. Chúng ta sử dụng một lược đồ quá mẫu (oversampling scheme) để ước tính các điểm giữa các bit truyền và sau đó truy xuất chúng tại các điểm phù hợp.

3.1. Phương thức quá mẫu (Oversampling procedure):

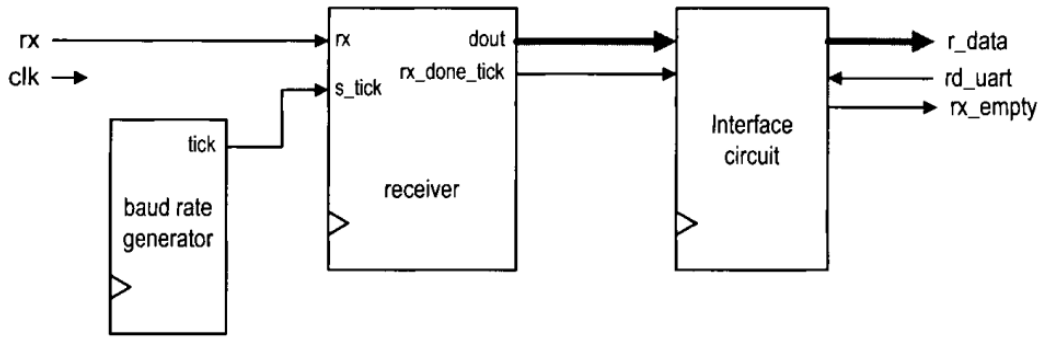
Tốc độ lấy mẫu được sử dụng phổ biến nhất là bằng 16 lần tốc độ baud, có nghĩa là mỗi bit nối tiếp được lấy mẫu 16 lần. Ví dụ giao tiếp sử dụng N bit dữ liệu và M stop bits. Sơ đồ lấy mẫu quá mức hoạt động như sau:

1. Chờ khi có tín hiệu đến trở thành 0, điểm bắt đầu của start bit, sau đó bắt đầu bộ đếm lấy dấu mẫu (sampling tick counter).
2. Khi counter đếm đến 7, tín hiệu đến đạt điểm giữa của start bit. Xóa counter về 0 và khởi động lại.
3. Khi counter đếm đến 15, tín hiệu đến trở thành 1 bit và đạt đến giữa bit dữ liệu đầu tiên. Truy suất giá trị của nó, dịch nó vào thanh ghi và khởi động lại counter.
4. Thực hiện bước 3 *N-1* lần nữa để lấy các bit dữ liệu còn lại.
5. Nếu sử dụng bit chẵn lẻ (parity bit), lặp lại bước 3 thêm một lần nữa để lấy parity bit.
6. Thực hiện bước 3 *M* lần nữa để đạt được stop bits.

Oversampling scheme về cơ bản thực hiện chức năng của một tín hiệu xung clock. Thay vì sử dụng tích cực cạnh lên để chỉ ra thời điểm tín hiệu đầu vào hợp lệ, nó sử dụng phép lấy dấu mẫu để ước tính điểm giữa của mỗi bit. Khi phía máy nhận không có thông tin về thời gian bắt đầu của start bit, ước tính có thể bị sai lệch nhiều nhất là 1/16. Các lần truy xuất bit dữ liệu tiếp theo cũng bị lệch nhiều nhất 1/16 từ điểm giữa. Dựa vào quá mẫu, tốc độ baud có thể chỉ là một phần nhỏ của tốc độ hệ thống xung clock, và do đó, sơ đồ này không phù hợp với tốc độ dữ liệu cao.

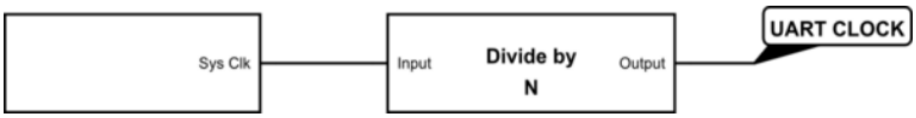
Sơ đồ *hình 3.1* bao gồm 3 thành phần chính:

- UART receiver: mạch để thu dữ liệu thông qua quá trình quá mẫu.
- Baud rate generator: mạch để tạo ra các dấu tick lấy mẫu.
- Interface circuit: mạch cung cấp bộ đệm (buffer) và trạng thái giữa UART receiver với hệ thống sử dụng UART.



Hình 3.1. Sơ đồ khối hệ thống con UART receiver

3.2. Baud rate generator



Hình 3.2.1. Cấu tạo Baud rate generator

Để tránh tạo miền thời gian mới và vi phạm nguyên tắc thiết kế, tín hiệu lấy mẫu phải hoạt động như enable ticks thay vì tín hiệu xung clock tới UART receiver.

$$Baud\ rate\ divisor = \frac{UART\ Sys\ Clk}{ClkDiv\ \times Baud\ Rate}$$

Đó là một bộ tạo xung clk được sử dụng để xác định truyền dữ liệu tốc độ bằng UART.

Nó hoạt động qua việc nhận tần số tương đối cao, việc sử dụng tần số clk cao nó sẽ nhỏ gọn và rẻ hơn rất nhiều so với việc sử dụng tần số UART thông thường.

Sys Clk thường được tính bằng MHz, quá nhanh so với tốc độ UART thông thường. Do đó, đồng hồ hệ thống cần được chia nhỏ, Giá trị *ClkDiv* thường bị hạn chế bởi phần cứng, các div xung nhịp thường rơi vào 8 hoặc 16.

- **Ví dụ rõ hơn về công thức.**

Cho phép nói rằng UART sys CLK là 80MHz, Và chọn clkdiv = 16 và muốn tốc độ truyền cho ra 57600, ta có:

$$Baud\ rate\ divisor = \frac{80000000}{16\times57600}.$$

Vì thế Baud rate divisor ra được 86.8056. Thanh ghi thường là số nguyên nên nó sẽ cho ra giá trị 87. Nếu chúng ta có tốc độ truyền nhỏ hơn, ta sẽ có ước số lớn hơn. Chúng ta cần phải cẩn thận để không có một số chia quá lớn vì nó sẽ làm tràn thanh ghi số chia. Ví dụ thanh ghi 8 bit thì không thể có bộ divisor lớn hơn 300.

Bằng cách đặt baud rate divisor có giá trị chính xác, về cơ bản nó sẽ tạo ra một bộ đếm thời gian ở tốc độ baud chính xác để truyền.

- **Nói thêm về biến baud rate divisor**

Chúng ta có thể thay đổi biến baud rate divisor trong bộ generator để có được tốc độ ở ngõ ra như mong muốn.

Ví dụ, chúng ta có Sys Clk = 2457600, và chúng ta muốn có tốc độ baud ở đầu ra là 9600 và 19200 baud/s. Thay vì xây dựng từng biến baud rate divisor riêng, như 128 với 19200 và 256 với 9600.

Thay vào đó, ta chỉ cần xây dựng đúng Clkdiv = 128, và chỉ cần thay đổi biến baud rate divisor qua bộ đếm mod. Kết quả như sau:

$$Baud\ rate\ dvrs = \frac{SysClk}{ClkDiv \times Baud\ rate} \leftrightarrow Baud\ rate = \frac{SysClk}{Clkdiv\ x\ Baud\ rate\ dvrs}$$

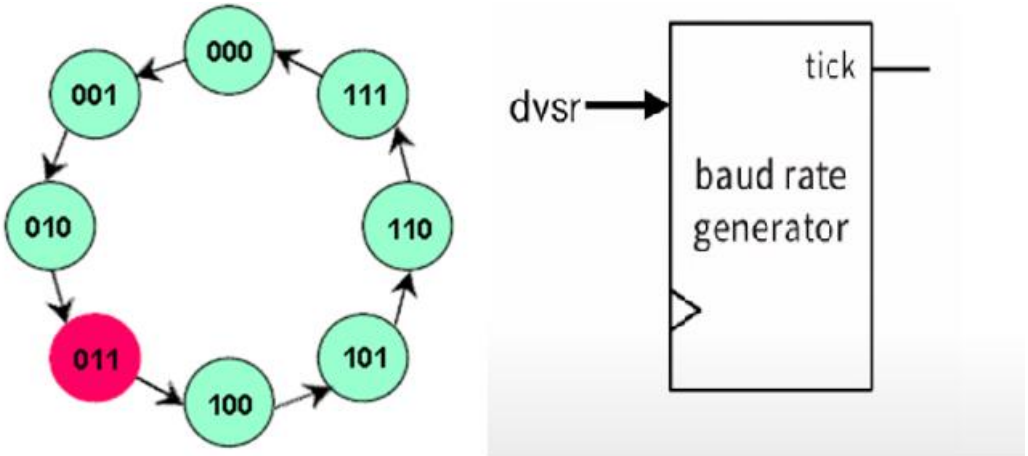
$$Baud\ rate\ divisor = 1 \leftrightarrow Baud\ rate = \frac{2457000}{128 \times 1} = 19195{,}3125 \sim 19200\ baud/s$$

$$Baud\ rate\ divisor = 2 \leftrightarrow Baud\ rate = \frac{2457000}{128 \times 2} = 9597{,}65625 \sim 9600\ baud/s$$

- **Vì sao xuất hiện sai số trên các phép tính trên?**

Baud rate generator có cấu tạo bên trong là thanh ghi số chia, thanh ghi thì không chấp nhận số thập phân, mà chỉ chấp nhận số nguyên, nên sẽ xảy ra sai số ($19195{,}3125 \sim 19200\ baud/s$, sai số khoảng 0.02%) không đáng kể, càng gần 0% càng tốt. Thông thường, chọn một Baud rate divisor càng nhỏ sẽ giảm sai số với Baud rate 1 cách nhất định.

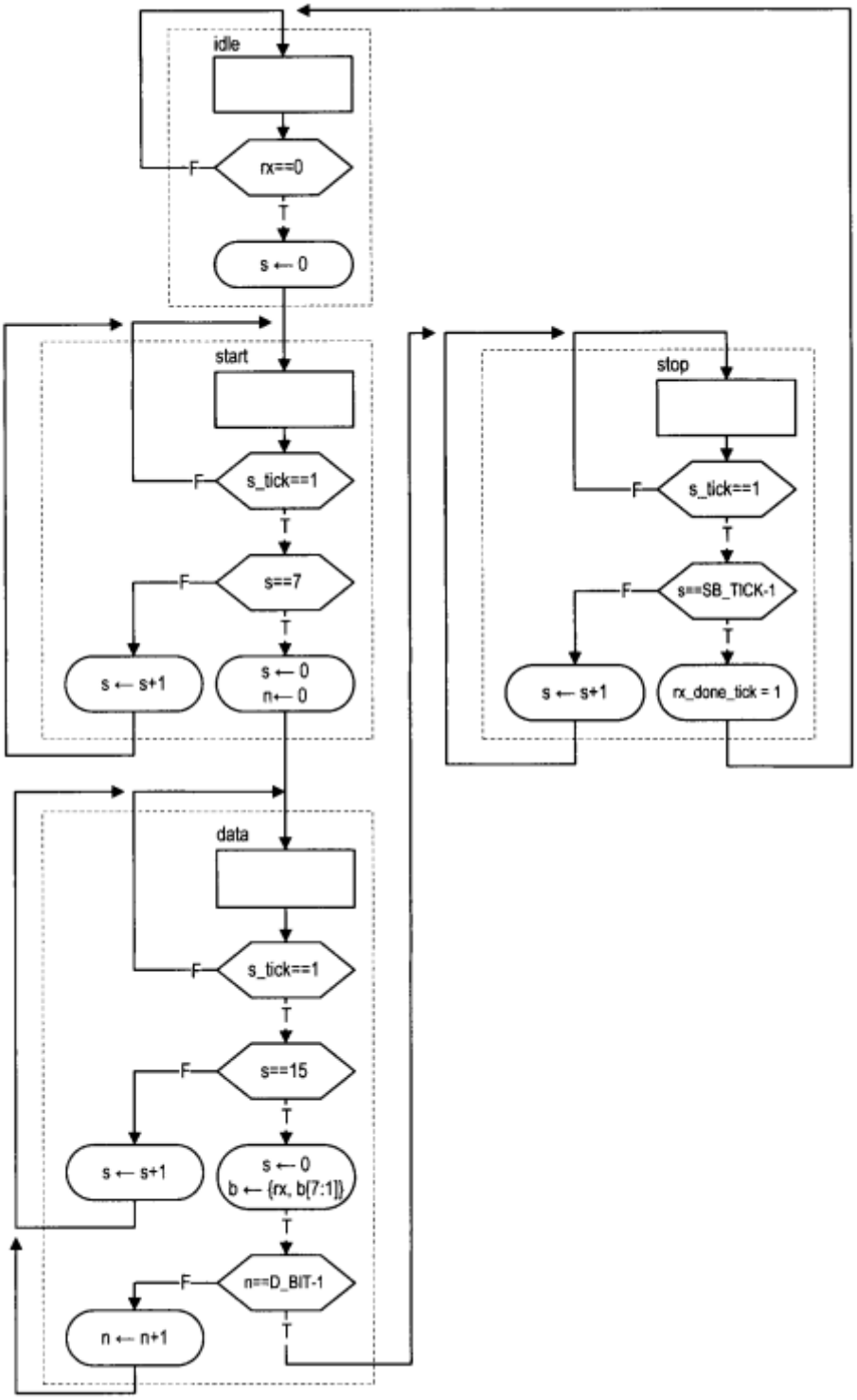
Sau khi bộ đếm mod đếm đủ ***n*** giá trị, ***s_tick*** sẽ nhảy lên 1, quá trình lấy mẫu bắt đầu với số lần bằng **ClkDiv.**



Hình 3.2.2. Bộ đếm mod n

3.3. UART receiver.

Dựa vào quy trình quá mẫu, chúng ta có thể làm nên đồ thị ASMD bên dưới. Để phù hợp với sửa đổi trong tương lai, hai hằng số được sử dụng trong phần mô tả. Hằng số *D_BIT* cho biết số bit dữ liệu và hằng số *SB_TICK* cho biết số lượng dấu tick cần thiết cho các stop bit, tương ứng là 16, 24 và 32 cho 1, 1.5 và 2 stop bit. *D_BIT* và *SB_TICK* được gán là 8 và 16 trong thiết kế này.



Hình 3.3. Biểu đồ ASMD của bộ thu UART

Đồ thị có 3 trạng thái chính, *start*, *data* và *stop*, đại diện cho việc xử lý bit start, bit dữ liệu và bit stop. Tín hiệu *s_tick* cho phép kích hoạt dấu tick từ bộ tạo tốc độ truyền (baud rate generator) và có 16 dấu tick trong một khoảng thời gian bit. Lưu ý rằng FSMD vẫn trong cùng trạng thái trừ khi tín hiệu *s_tick* được xác định. Có 2 bộ đếm counter, đại diện bởi các thanh ghi s và n. Thanh ghi s kiểm soát số lượng dấu lấy mẫu và đếm đến 7 trong trạng thái bắt đầu, đếm đến 15 ở trạng thái dữ liệu và đếm đến *SB_TICK* ở trạng thái dừng. Thanh ghi n kiểm soát số bit dữ liệu nhận được ở trạng thái dữ liệu. Các bit truy xuất được dịch vào và tập hợp trong thanh ghi *b*. Một tín hiệu trạng thái, *rx_done_tick*, nó được xác nhận trong một chu kỳ xung clock sau khi quá trình nhận hoàn tất. Code tương ứng của bộ thu UART:

```
module uart_rx
# (
        parameter   DBIT = 8, // # data bits
                    SB_TICK = 16 // # ticks for stop bits
)
(
        input wire clk, reset,
        input wire rx, s_tick,
        output reg rx_done_tick,
        output wire [7:0] dout
);

// symbolic state declaration
        localparam [1:0]
                idle = 2'b00,
                start = 2'b01,
                data = 2'b10,
                stop = 2'b11;

// signal declaration
        reg [1:0] state_reg , state_next ;
        reg [3:0] s_reg , s_next ;
        reg [2:0] n_reg , n_next ;
        reg [7:0] b_reg , b_next ;

// body
```

```
// FSMD state & data registers
always @( posedge clk , posedge reset)
        if (reset)
                begin
                        state_reg <= idle;
                        s_reg <= 0;
                        n_reg <= 0;
                        b_reg <= 0;
                end
        else
                begin
                        state_reg <= state_next ;
                        s_reg <= s_next;
                        n_reg <= n_next;
                        b_reg <= b_next;
                end

// FSMD next-state logic
always @*
begin
        state_next = state_reg ;
        rx_done_tick = 1'b0;
        s_next = s_reg;
        n_next = n_reg;
        b_next = b_reg;
        case (state_reg)
                idle :
                        if (~rx)
                                begin
                                        state_next = start;
                                        s_next = 0;
                                end
                start :
                        if (s_tick)
                                if (s_reg == 7)
                                        begin
                                                state_next = data;
                                                s_next = 0;
                                                n_next = 0;
                                        end
                                else
                                        s_next = s_reg + 1;
                data :
```

```
                if (s_tick)
                    if (s_reg == 15)
                            begin
                                    s_next = 0;
                                    b_next = {rx , b_reg [7 : 1]} ;
                                    if (n_reg == (DBIT-1))
                                            state_next = stop ;
                                    else
                                            n_next = n_reg + 1;
                            end
                    else
                            s_next = s_reg + 1;
            stop:
                if (s_tick)
                    if (s_reg == (SB_TICK-1))
                            begin
                                    state_next = idle;
                                    rx_done_tick = l'bl;
                            end
                    else
                            s_next = s_reg + 1;
        endcase
end

// output
assign dout = b_reg;
endmodule
```

3.4 Interface Circuit

Trong một hệ thống lớn, UART thường là một mạch ngoại vi để truyền dữ liệu nối tiếp. Hệ thống này sẽ liên tục kiểm tra trạng thái của nó theo định kỳ để lấy dữ liệu và xử lý từ đã nhận. Mạch Interface của máy thu có hai chức năng. Đầu tiên, nó cung cấp một cơ chế báo hiệu sự sẵn có của một dữ liệu mới và ngăn không cho dữ liệu đã nhận được lấy lại nhiều lần. Thứ hai, nó có thể cung cấp không gian đệm giữa máy thu (receiver) và hệ thống chính. Có ba phương án thường được sử dụng:

- Cờ FF
- Một cờ FF và một bộ đệm dữ liệu
- Bộ đệm FIFO

Lưu ý rằng bộ thu UART xác nhận rx-done-tick trên một chu kỳ xung đồng hồ sau khi nhận được từ dữ liệu. Lược đồ đầu tiên sử dụng cờ FF để theo dõi xem có từ dữ liệu mới hay không. FF có hai tín hiệu đầu

vào. Một là độ trễ set-flag, đặt cờ FF thành 1, và nút còn lại là độ trễ clr-flag, xóa cờ FF thành 0. Tín hiệu rx-done-tick được kết nối với tín hiệu trễ set-flag và sẽ được đặt cờ khi có một dữ liệu mới đến. Hệ thống chính kiểm tra đầu ra của cờ FF để xem liệu có dữ liệu mới hay không. Nó xác nhận tín hiệu trễ clr-flag trên một chu kỳ cung clock sau khi truy xuất dữ liệu. Sơ đồ khối cấp cao nhất được thể hiện trong Hình 8.4 (a). Để phù hợp với các sơ đồ khác, đầu ra của cờ FF được đảo ngược để tạo ra tín hiệu rx_empty cuối cùng, cho biết rằng không có từ mới nào. Trong lược đồ này, hệ thống chính truy xuất dữ liệu trực tiếp từ thanh ghi dịch chuyển của bộ thu UART và không cung cấp thêm bất kỳ không gian buffer nào. Nếu hệ thống từ xa bắt đầu truyền dữ liệu mới trước khi hệ thống chính xử lý dữ liệu cũ (tức là cờ FF vẫn được xác nhận), dữ liệu cũ sẽ bị ghi đè, một lỗi được gọi là chạy quá tải dữ liệu.

Để cung cấp một số đệm, một bộ đệm một dữ liệu có thể được thêm vào, như thể hiện trong Hình 8.4 (b). Khi tín hiệu đánh dấu rx-done-tick được xác nhận, dữ liệu nhận được sẽ được tải vào bộ đệm và cờ FF cũng được đặt. Người nhận có thể tiếp tục hoạt động mà không cần xóa nội dung của dữ liệu nhận được cuối cùng. Việc chạy tràn dữ liệu sẽ không xảy ra miễn là hệ thống chính truy xuất cũ khi có dữ liệu mới.

Thường thì các Interface Circuit trong phần này chính là các bộ đệm First In First Out (FIFO).

Bộ đệm này được thực hiện dựa trên một mảng. Kèm theo đó là 2 con trỏ Write (*WritePt*) và Read (*ReadPt*).

Có nhiều cách để tạo một bộ đệm FIFO, tuy nhiên cách hay được sử dụng nhất là bộ FIFO vòng (circular buffer, ring buffer, hay array-base bufer).

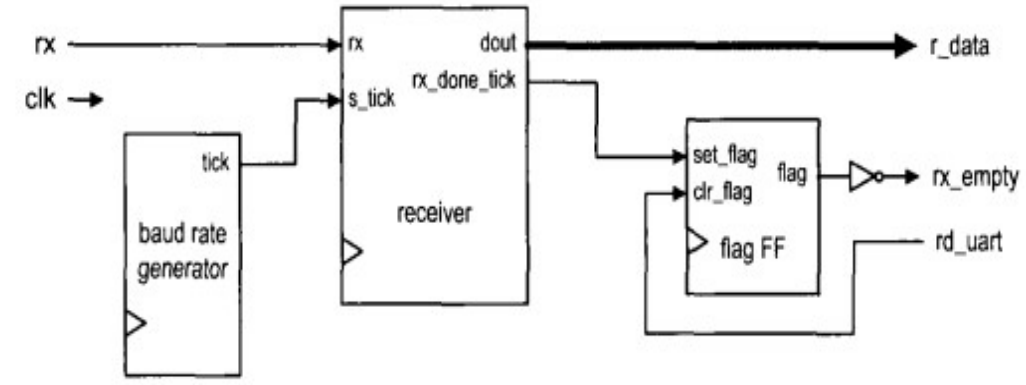
- Mỗi khi nhận lệnh ghi, con trỏ *Write* sẽ ghi data vào bộ đệm, sau đó sẽ tăng lên 1 đơn vị.
- Mỗi khi nhận lệnh đọc, cỏn trỏ *Read* sẽ tăng lên một đơn vị. Sau đó đọc giá trị từ bộ đệm ra.

Khi 1 con trỏ tới được cuối mảng, nó sẽ quay lại vị trí đầu tiên. Đó là lý do vì sao gọi đây là bộ đệm vòng.

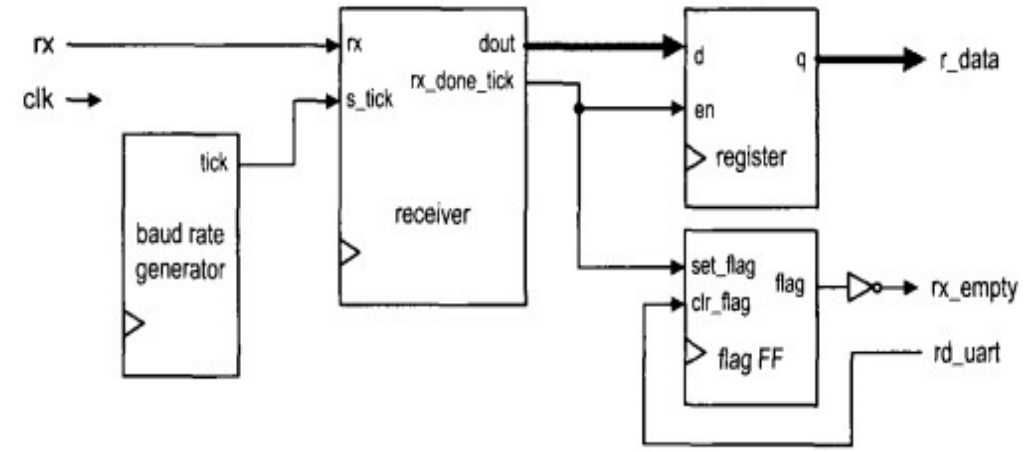
Trong một bộ đệm FIFO sẽ có 2 tín hiệu là *full* và *empty* để xác định rằng FIFO đang ở trạng thái đầy (không thể đọc thêm dữ liệu) hay trạng thái trống (không để được đọc nữa). Một trong hai trường hợp này sẽ xảy ra khi mà con trỏ *Write* và *Read* gặp nhau. Bộ điều khiển sẽ khó có thể nào phân biệt được một trong hai trường hợp trên.

Để giải quyết vấn đề này, ta có thể thêm một biến đếm vào, biến này sẽ tăng 1 đơn vị khi con trỏ Write ghi dữ liệu, giảm 1 đơn vị khi con

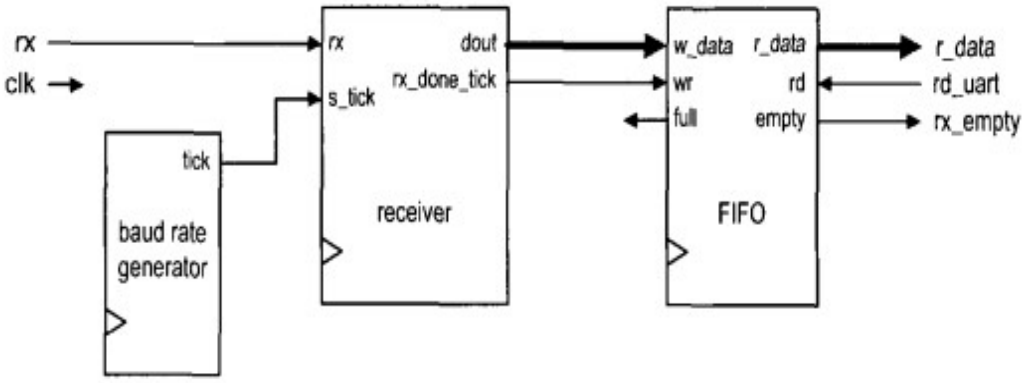
trỏ Read đọc dữ liệu. Khi 2 con trỏ gặp nhau, nếu biến đếm này bằng 1 thì FIFO đang ở trạng thái *full* và ngược lại.



(a) Flag FF



(b) Flag FF and one-word buffer



(c) FIFO buffer

Figure 8.4 Interface circuit of a UART receiving subsystem.

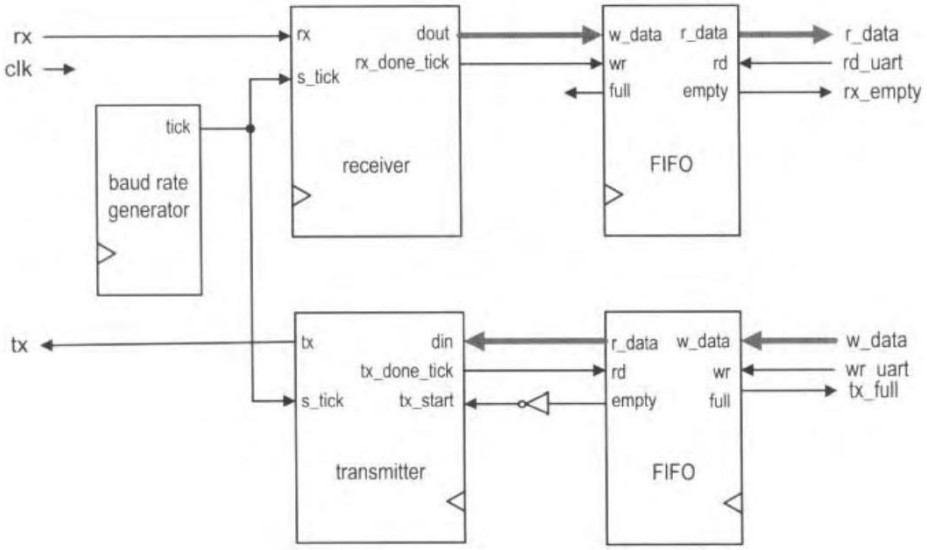
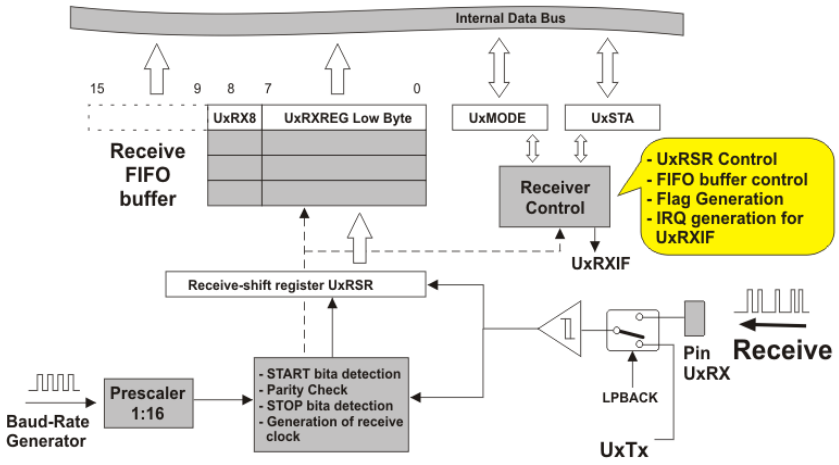


Figure 8.5 Block diagram of a complete UART.

CHƯƠNG 4: THIẾT KẾ (UART RECEIVER)

4.1. Sơ đồ khối:



4.2. Thiết kế giải thuật UART Receiver:

Tóm tắt các thông tin yêu cầu của thiết kế và các ý tưởng thiết kế sơ khai:

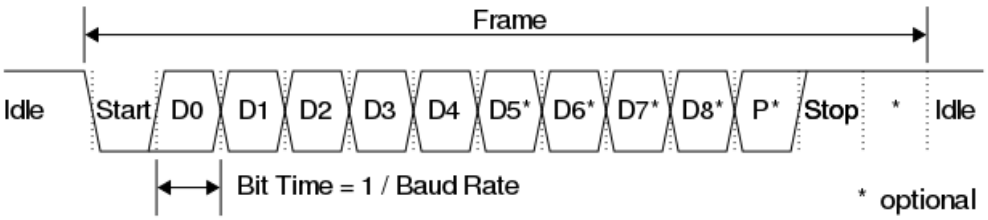
Để thiết kế được phần cứng nhận theo chuẩn UART, chúng ta cần 1 số lưu ý sau:

1.Baudrate: Là tốc độ bit được truyền, tính bằng số bit/giây. Giá trị này chỉ đúng khi xét ở một lần truyền, với 1 byte data tương ứng.

2.Tín hiệu clock: Để xử lý nhận dữ liệu thông qua chân *RX*, chúng ta phải sử dụng internal clock. Internal clock này có tần số phải đủ lớn để có thể lấy mẫu được *RX* đồng thời có thể phát hiện được sự thay đổi mức của *RX*. Thông thường, các board FPGA đều có nguồn clock trên chip từ 50Mhz - 100Mhz. Chúng ta có thể sử dụng trực tiếp nguồn clock này.

3.Số lượng bit trong 1 lần truyền: Thông thường, UART có 3 thông tin cần phân biệt: Baudrate, Stop bit, Parity bit. Tùy theo nhu cầu, chuẩn UART có dạng như sau:

- START BIT - 8BIT DATA - PARITY BIT - STOP BIT: Tổng cộng sẽ có 11bits trong 1 lần truyền.
- START BIT - 8BIT DATA - STOP BIT: Tổng cộng sẽ có 10bits trong 1 lần truyền
- START BIT - 8BIT DATA: Tổng cộng sẽ có 9bits trong 1 lần truyền.

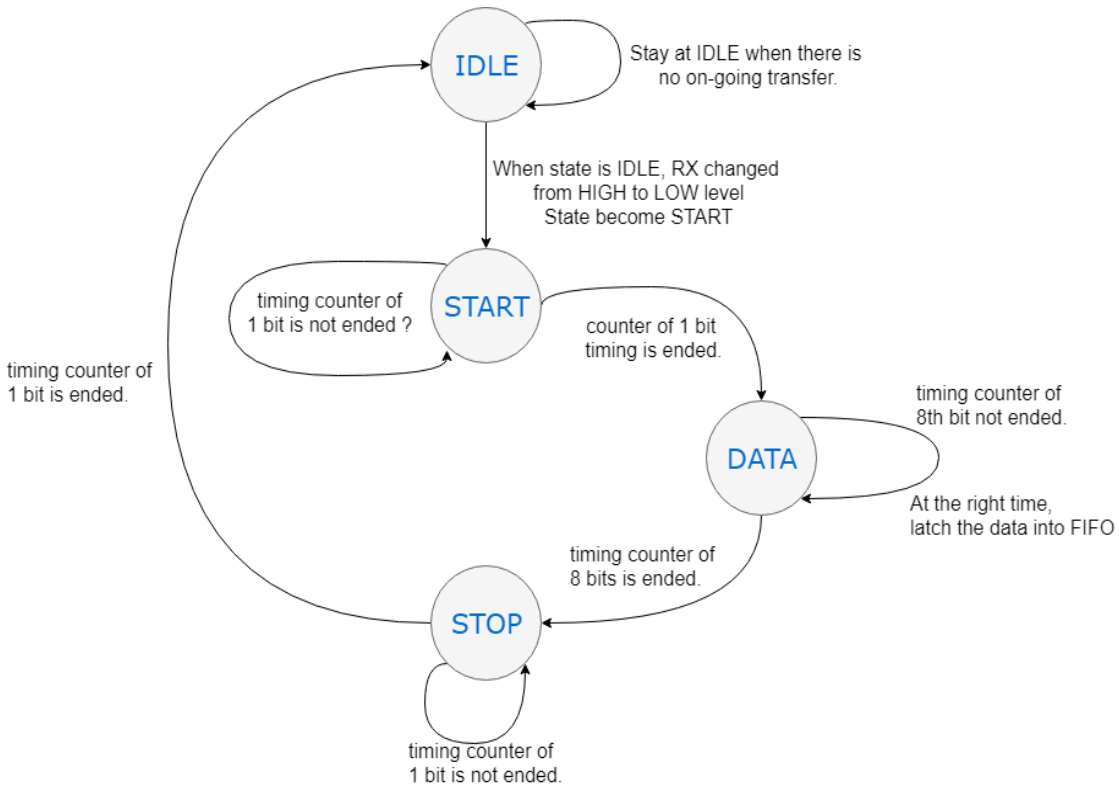


Hình 4.2.1. Dạng sóng của UART

4. FIFO lưu dữ liệu: Vì UART truyền đi 1 byte với các bit là nối tiếp nhau, do đó, để lưu lại và sử dụng data 8bits sau khi kết thúc, chúng ta cần có 1 bộ đệm để lưu các bits dữ liệu. Người ta gọi đó là Buffer, hoặc FIFO, hoặc đơn giản là Registers.
5. State Machine: Đối với những khối logic có chức năng và hoạt động mang tính chu trình, lặp đi lặp lại và đi qua các trạng thái cố định, chúng ta thường sử dụng State Machine. Logic của State Machine này sẽ được nói đến chi tiết ở phần sau.

Sau đây, chúng ta tìm hiểu thiết kế 1 module nhận - UART Receiver có đặc điểm:
- 1 bit Start, 1 bit Stop, 8 Bit data, 0 Parity
- Baudrate 9600
- Internal clock 100Mhz
- Bộ đệm nhận giá trị (Receiver FIFO) 8Bit.

Triển khai mô tả đầy đủ của phần cứng dạng Block Diagram
Sơ đồ State Machine:
Sử dụng state machine với 4 trạng thái: IDLE, START, DATA, STOP. Do đó, cần 1 thanh ghi 2 bits để có thể mã hóa được 4 trạng thái trên.



Timing counter: dùng để đếm số lượng xung internal clock để cho ra thời gian bit theo tốc độ baud. Ví dụ, đối với tốc độ baud là 9600 Bd, số lượng bit trong 1 giây là 9600 bits, tương ứng với tần số 9600Hz, hay 1 chu kỳ bit là 1/9600 xấp xỉ 0.1 (ms)

Cách vận hành của sơ đồ này như sau:

Trạng thái IDLE: vẫn tồn tại khi không nhận được yêu cầu từ tín hiệu RX. Nhưng khi tín hiệu RX chuyển từ mức cao xuống mức thấp thì trạng thái sẽ trở thành START.

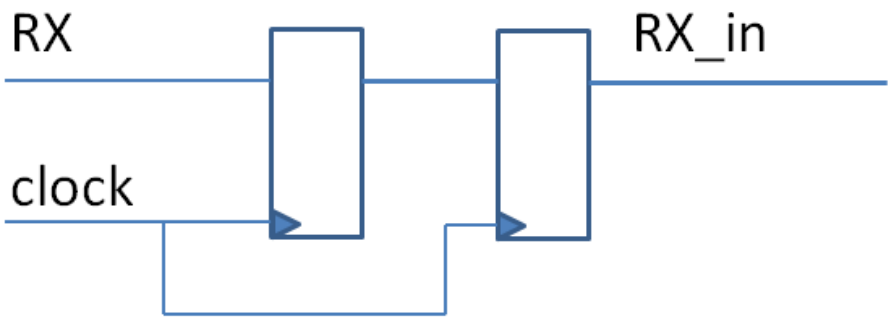
Trạng thái START: kéo dài trong thời gian đếm 1 bit theo tốc độ baud, lúc này từ trạng thái START chuyển sang trạng thái DATA, trường hợp trạng thái không thay đổi khi thời gian đếm 1 bit chưa kết thúc.

Trạng thái DATA: kéo dài trong thời gian đếm 8 bit theo tốc độ baud, lúc này từ trạng thái DATA chuyển sang trạng thái STOP, đúng thời điểm sẽ ghi dữ liệu vào FIFO, trường hợp trạng thái không thay đổi khi chưa đếm xong bit thứ 8.

Trạng thái STOP: tương tự trạng thái START, kéo dài trong thời gian đếm 1 bit theo tốc độ baud, lúc này từ trạng thái STOP chuyển sang lại trạng thái IDLE, trường hợp trạng thái không thay đổi khi thời gian đếm 1 bit chưa kết thúc.

Trước tiên, chúng ta phải đồng bộ tín hiệu *RX* về internal clock để tránh hiện tượng không ổn định của tín hiệu do sai lệch về pha của tín hiệu *RX* và internal clock

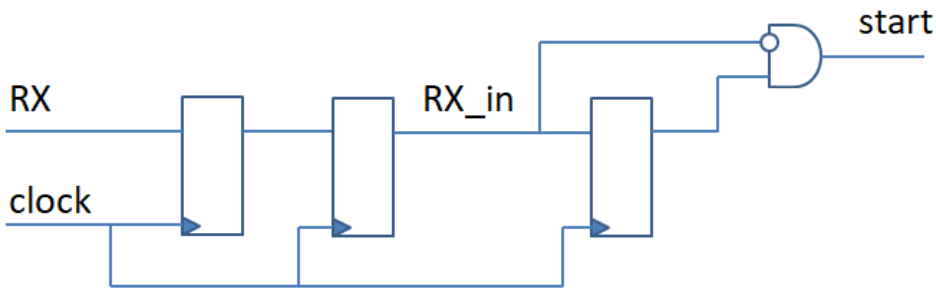
Mạch đồng bộ thường gồm 2 FlipFlop nối tiếp với nhau, được cấp xung clock. Sau khi đồng bộ sẽ cho ra tín hiệu RX_in.



Hình 4.2.2. Mạch đồng bộ tín hiệu RX

Để từ trạng thái IDLE chuyển sang trạng thái START, tín hiệu RX cần tích cực mức thấp. Vì vậy mạch cần có 1 cổng đảo để tín hiệu RX_in đưa vào sẽ chuyển từ mức cao sang mức thấp. Tín hiêu ở ngõ ra sẽ báo

hiệu thời điểm bắt đầu của START bit.



Hình 4.2.3. Mạch báo hiệu trạng thái START

Tiếp theo tiểu luận sẽ trình bày sơ đồ điều khiển trạng thái qua bộ state logic.

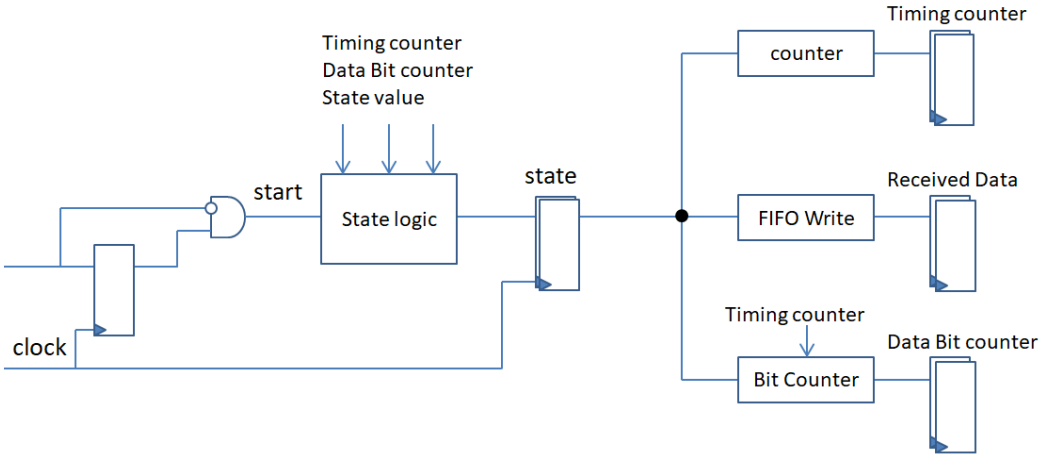
Tín hiệu start sẽ được nối vào state logic, để chuyển trạng thái đầy đủ thì chúng ta cần các tín hiệu khác nữa, bao gồm Timing Counter và Data Bit counter

Data Bit counter có 2 tác dụng:

- Để tạo điều kiện dừng cho State Machine trong trạng thái DATA.
- Tạo chỉ số để lưu giá trị nhận được vào FIFO.

Ngõ ra state sẽ nối với 3 ngõ vào như hình dưới:

- Ngõ vào counter nối tiếp với timing counter để đo thời gian đếm 1 bit, sau khi hết thời gian này, từ trạng thái START sẽ chuyển sang DATA và tương tự từ STOP sang IDLE
- Riêng trạng thái DATA cần đếm đủ 8 bit mới chuyển sang trạng thái STOP, ngoài timing counter còn phải kết hợp với ngõ vào bit counter nối với data bit counter để kiểm soát số bit đếm được.
- Sau khi đếm đủ, dữ liệu sẽ được ghi vào FIFO và truy xuất dữ liệu nhận được.



Hình 4.2.4. Phần điều khiển của state machine

CHƯƠNG 5: ĐÁNH GIÁ QUA TESTBENCH

Full Code Test Bench:

```
/*Testbench uses a 25 MHz clock
   Want to interface to 115200 baud UART
   25000000 / 115200 = 217 Clocks Per Bit.*/

`timescale 1ns/10ps
module UART_RX_TB();

  parameter c_CLOCK_PERIOD_NS = 40;
  parameter c_CLKS_PER_BIT    = 217;
  parameter c_BIT_PERIOD      = 8600;

  reg r_Clock = 0;
  reg r_RX_Serial = 1;
  wire [7:0] w_RX_Byte;


  // Takes in input byte and serializes it
  task UART_WRITE_BYTE;
    input [7:0] i_Data;
    integer     ii;
    begin

      // Send Start Bit
      r_RX_Serial <= 1'b0;
      #(c_BIT_PERIOD);
      #1000;

      // Send Data Byte
      for (ii=0; ii<8; ii=ii+1)
        begin
          r_RX_Serial <= i_Data[ii];
          #(c_BIT_PERIOD);
        end

      // Send Stop Bit
      r_RX_Serial <= 1'b1;
      #(c_BIT_PERIOD);
```

```
    end
  endtask // UART_WRITE_BYTE


  UART_RX #(.CLKS_PER_BIT(c_CLKS_PER_BIT))
UART_RX_INST//khoi tao may thu
    (.i_Clock(r_Clock),
     .i_RX_Serial(r_RX_Serial),
     .o_RX_DV(),
     .o_RX_Byte(w_RX_Byte)
     );

  always
    #(c_CLOCK_PERIOD_NS/2) r_Clock <= !r_Clock;//tạo xung clock


  // Main Testing:
  initial
    begin
      // Send a command to the UART (exercise Rx)
      @(posedge r_Clock);
      UART_WRITE_BYTE(8'h35);
      @(posedge r_Clock);

      // Check that the correct command was received
      if (w_RX_Byte == 8'h35)
        $display("Test Passed - Correct Byte Received");
      else
        $display("Test Failed - Incorrect Byte Received");
    $finish();
    end
  // Ở đây thử truyền dữ liệu 8'h35 để kiểm tra tín hiệu nhận
  initial
  begin
    // Required to dump signals to EPWave
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

endmodule
```

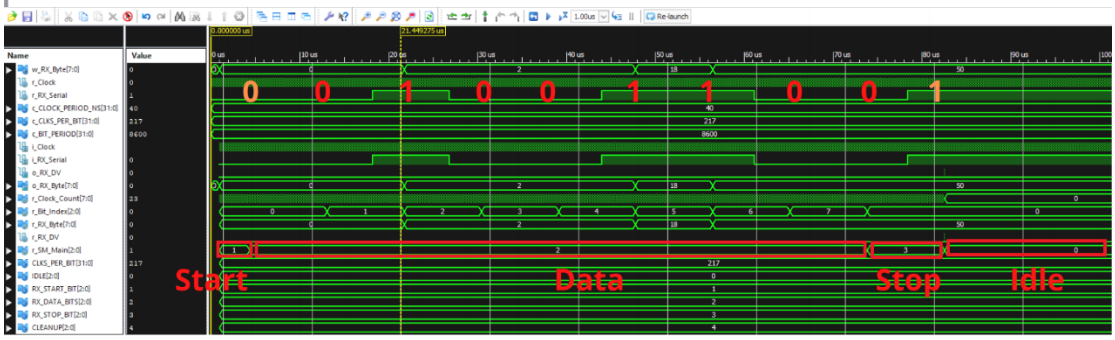
Mô phỏng testbench:

Case1: gửi 8’h32: kiểm tra là 8’h35

```
// Main Testing:
initial
  begin
    //gui lenh den UART
    @(posedge r_Clock);
    UART_WRITE_BYTE(8'h32);//32:0011 0010
    @(posedge r_Clock);

    //kiem tra xem da nhan dung du lieu chua
    if (w_RX_Byte == 8'h35)//35:0011 0101
      $display("Test Passed - Correct Byte Received");
    else
      $display("Test Failed - Incorrect Byte Received");
  $finish();
  end
```

```
Console
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run 100.00us
Test Failed - Incorrect Byte Received
Stopped at time : 87020 ns : File "C:/Users/ACER/OneDrive/Desktop/Xilinx_Project/ba/ba_tb.v" Line 66
ISim>
# run 100.00us
ISim>
```

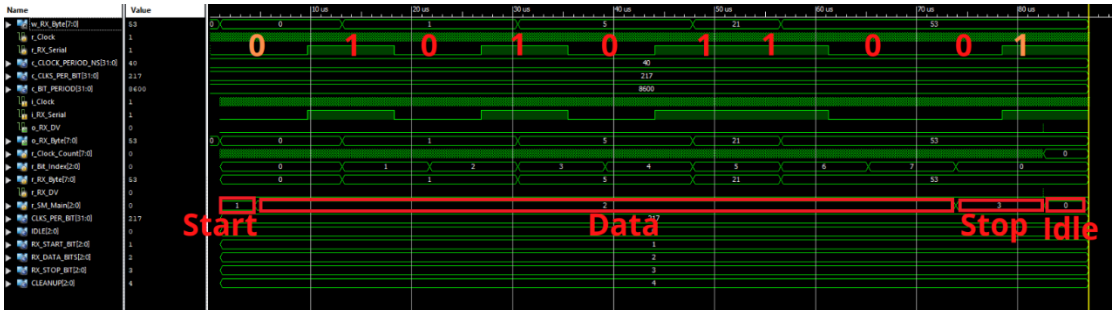


Case1: gửi 8’h35: kiểm tra là 8’h35

```
// Main Testing:
initial
  begin
    //gui lenh den UART
    @(posedge r_Clock);
    UART_WRITE_BYTE(8'h32);
    @(posedge r_Clock);

    //kiem tra xem da nhan dung du lieu chua
    if (w_RX_Byte == 8'h35)//35:0011 0101
      $display("Test Passed - Correct Byte Received");
    else
      $display("Test Failed - Incorrect Byte Received");
  $finish();
  end
```

```
Console
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run 100.00us
Test Passed - Correct Byte Received
Stopped at time : 87020 ns : File "C:/Users/ACER/OneDrive/Desktop/Xilinx_Project/ba/ba_tb.v" Line 66
ISim>
```



CHƯƠNG 6: KẾT LUẬN

Không có giao thức truyền thông nào là hoàn hảo, nhưng UART thực hiện khá tốt công việc của nó. Các hệ thống nhúng, vi điều khiển và máy tính hầu hết sử dụng UART như một dạng giao thức giao thức phần cứng giữa thiết bị và thiết bị.

Dưới đây là một số ưu và nhược điểm để giúp chúng ta quyết định xem nó có phù hợp với nhu cầu của bản thân hay không:

❖ ***Ưu điểm***

-Chỉ sử dụng hai dây
-Không cần tín hiệu clock
-Có một bit chẵn lẻ để cho phép kiểm tra lỗi
-Cấu trúc của gói dữ liệu có thể được thay đổi miễn là cả hai bên đều được thiết lập cho nó
-Phương pháp có nhiều tài liệu tham khảo và được sử dụng rộng rãi

❖ ***Nhược điểm***

-Kích thước của khung dữ liệu được giới hạn tối đa là 9 bit
-Không hỗ trợ nhiều hệ thống slave hoặc nhiều hệ thống master
-Tốc độ truyền của mỗi UART phải nằm trong khoảng 10% của nhau

TÀI LIỆU THAM KHẢO

1. FPGA Prototyping by verilog examples – Pong P. Chu
2. Universal asynchronous receiver-transmitter- Wikipedia.org