گزارش آزمایش هفتم طراحی سیستم‌های دیجیتال

گروه شش

اعضا:

احمد سلیمی

همیلا میلی

درنا دهقانی

شرح آزمایش

در این آزمایش می‌خواهیم یک UART یا Universal Asynchronous Receiver Transmitter طراحی کنیم. در بخش فرستنده این UART هر بار ۱۰ بیت ارسال می‌شود که بیت اول برای شروع، بیت دوم برای توازن، ۷ بیت بعدی برای داده و نهایتاً بیت آخر برای خاتمه ارسال می‌شوند. در بخش گیرنده این UART پس از دریافت بیت شروع، ۸ بیت بعدی یعنی توازن و داده به صورت سریالی دریافت شده و هر یک در ثباتی مجزا ذخیره می‌شوند.

ماژول‌ها

Sender

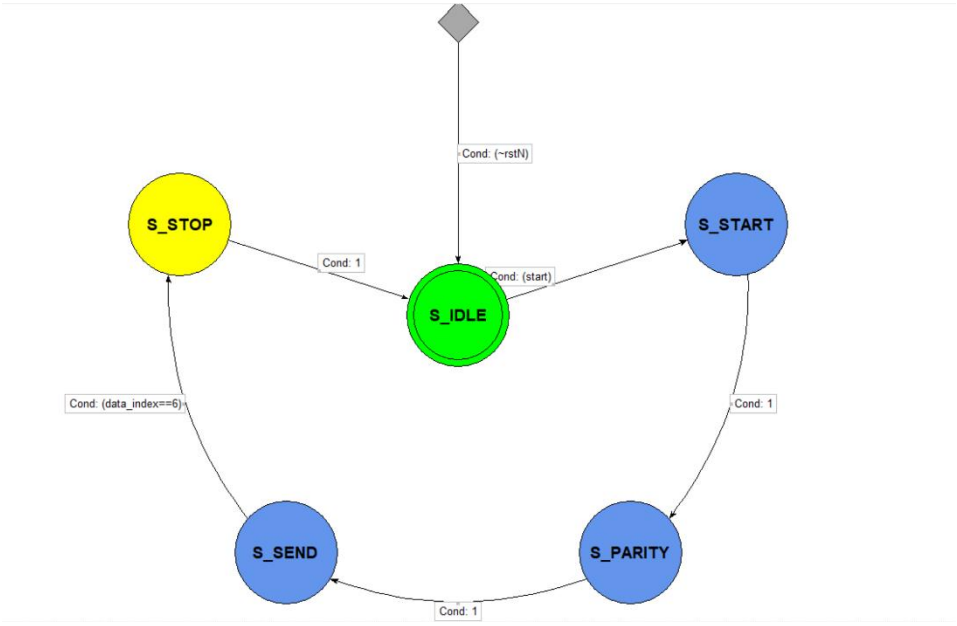
ورودی‌ها:

- rstN
- clk
- start: برای شروع ارسال برابر با ۱ می‌شود.
- data_in: داده‌ای که قرار است ارسال شود.

خروجی‌ها:

- s_out: خروجی سریالی فرستنده.
- Sent: با اتمام ارسال، برابر با ۱ می‌شود.

FSM این فرستنده به شکل زیر است که توسط Modelsim تولید شده است:



این ماژول، فرستنده‌ی UART است.

برای این ماژول، ۵ حالت مختلف داریم: idle, start, parity, send, stop

چون مدار ترتیبی است، از always block استفاده می‌کنیم که به لبه بالارونده کلاک و لبه پایین‌رونده ریست حساس است. در صورت ۰ شدن سیگنال rstN، فرستنده در حالت idle قرار می‌گیرد، اندیس داده برابر با ۰ می‌شود و خروجی‌ها ۰ می‌شوند.

در غیر اینصورت با لبه بالارونده کلاک، بر حسب حالتی که در آن قرار داریم عمل می‌کنیم:

- Idle: اگر ورودی start برابر با ۱ شده باشد، وارد حالت بعدی یعنی start می‌شویم، ورودی data_in در رجیستر data قرار می‌گیرد و اندیس داده و خروجی sent برابر با صفر می‌شوند. در غیر اینصورت اتفاقی نمی‌افتد.
- Start: بیت شروع که برابر با 1 است در خروجی سریالی قرار می‌گیرد و به حالت parity می‌رویم.
- Parity: مقدار parity که برابر با حاصل xor بیت‌های data است در خروجی سریالی قرار می‌گیرد و به حالت send می‌رویم.
- Send: بیتی که در اندیس data_index از رجیستر data قرار دارد، وارد خروجی سریالی می‌شود، اندیس داده یکی بیشتر می‌شود و اگر برابر با آخرین اندیس داده که ۶ است بشود، از حالت send خارج شده و به stop می‌رویم. در غیر اینصورت در همین حالت میمانیم.
- Stop: بیت پایان که برابر با ۰ است به خروجی سریالی می‌رود، به حالت idle بازمی‌گردیم و خروجی sent برابر با ۱ می‌شود، چون ارسال به پایان رسیده است.
- اگر غیر از این‌ها بود وارد حالت idle می‌شویم.

کد وریلاگ این بخش به شکل زیر است:

```
module sender # (
    parameter START_SIG = 1
) (
    input           rstN,
    input           clk,
    input           start,
    input   [6:0]   data_in,
    output  reg     s_out,
    output  reg     sent
);

localparam S_IDLE       = 0;
localparam S_START      = 1;
localparam S_PARITY     = 2;
localparam S_SEND       = 3;
localparam S_STOP       = 4;

reg [2:0]   state;
reg [6:0]   data;
reg [2:0]   data_index;

wire parity_sig;
assign parity_sig = ^data;

always @(posedge clk or negedge rstN) begin
    if (~rstN) begin
        state <= S_IDLE;
        data_index <= 0;
        s_out <= 0;
        sent <= 0;
    end
    else begin
        case (state)
            S_IDLE: begin
                if (start) begin
                    data_index <= 0;
                    data <= data_in;
                    state <= S_START;
                    sent <= 0;
                end
            end
```

```
            S_START: begin
                s_out <= START_SIG;
                state <= S_PARITY;
            end
            S_PARITY: begin
                s_out <= parity_sig;
                state <= S_SEND;
            end
            S_SEND: begin
                s_out <= data[data_index];
                data_index <= data_index + 1;
                if (data_index == 6)
                    state <= S_STOP;
            end
            S_STOP: begin
                s_out <= ~START_SIG;
                state <= S_IDLE;
                sent <= 1;
            end
            default: state <= S_IDLE;
        endcase
    end
end

endmodule
```

Receiver

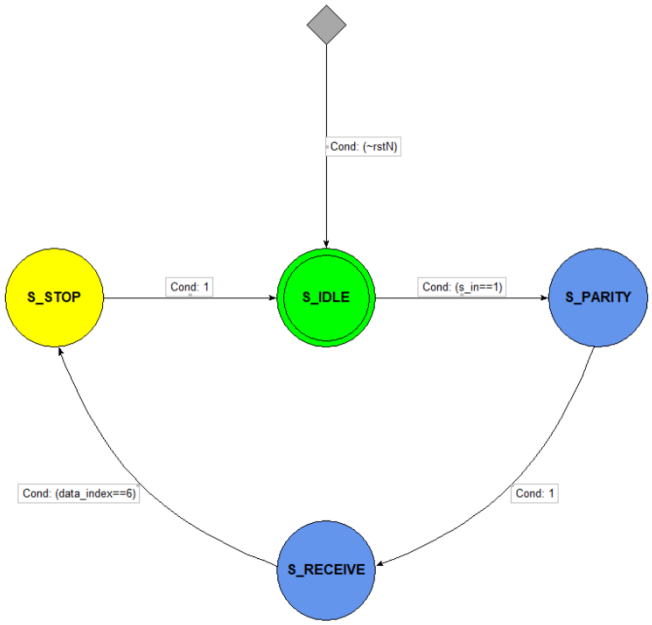
ورودی‌ها:

- rstN
- clk
- s_in: ورودی سریالی.

خروجی‌ها:

- Received: در صورت پایان گرفتن ورودی‌ها، ۱ می‌شود.
- chack_parity: در صورت برابر parity ارسالی با parity داده‌ی گرفته برابر بود، ۱ می‌شود.
- Data: داده‌ی گرفته‌شده.

- FSM این گیرنده به شکل زیر است که توسط Modelsim تولید شده است:



این ماژول، گیرنده‌ی UART است.

برای این ماژول ۴ حالت مختلف داریم: idle, parity, receive, stop

چون مدار ترتیبی است، از always block استفاده می‌کنیم که به لبه بالارونده کلاک و لبه پایین‌رونده ریست حساس است. در صورت ۰ شدن سیگنال rstN، فرستنده در حالت idle قرار می‌گیرد، اندیس داده برابر با ۰ می‌شود و خروجی‌ها ۰ می‌شوند.

در غیر اینصورت با لبه بالارونده کلاک، بر حسب حالتی که در آن قرار داریم عمل می‌کنیم:

- idle: اگر ورودی سریالی با بیت شروع یعنی ۱ برابر بود، وارد حالت parity می‌شویم و اندیس داده و خروجی‌ها برابر با ۰ می‌شوند. در غیر اینصورت اتفاقی نمی‌افتد.
- Parity: مقدار ورودی سریالی که parity ارسال شده است، در expected_parity قرار گرفته و به حالت receive می‌رویم.
- Receive: مقدار ورودی سریالی در اندیس data_index از رجیستر data قرار می‌گیرد، اندیس داده یکی بیشتر می‌شود و اگر برابر با آخرین اندیس داده که ۶ است بشود، از حالت send خارج شده و به stop می‌رویم. در غیر اینصورت در همین حالت میمانیم.
- Stop: به حالت idle بازمی‌گردیم و خروجی received برابر با ۱ می‌شود، چون گرفتن داده به پایان رسیده است.
- اگر غیر از این‌ها بود وارد حالت idle می‌شویم.

کد وریلاگ این بخش به شکل زیر است:

```
module receiver # (
    parameter START_SIG = 1
) (
    input               rstN,
    input               clk,
    input               s_in,
    output  reg         received,
    output              check_parity,
    output  reg [6:0]   data
);

localparam S_IDLE       = 0;
localparam S_PARITY     = 1;
localparam S_RECEIVE    = 2;
localparam S_STOP       = 3;

reg [1:0]   state;
reg [2:0]   data_index;
reg         expected_parity;
wire        actual_parity;

assign actual_parity = ^data;
assign check_parity = actual_parity == expected_parity;

always @(posedge clk or negedge rstN) begin
    if (~rstN) begin
        state <= S_IDLE;
        data_index <= 0;
        received <= 0;
        data <= 0;
    end
    else begin
        case (state)
            S_IDLE: begin
                if (s_in == START_SIG) begin
                    data_index <= 0;
                    data <= 0;
                    state <= S_PARITY;
                    received <= 0;
                end
            end
```

```
            S_PARITY: begin
                expected_parity <= s_in;
                state <= S_RECEIVE;
            end
            S_RECEIVE: begin
                data[data_index] <= s_in;
                data_index <= data_index + 1;
                if (data_index == 6) begin
                    state <= S_STOP;
                end
            end
            S_STOP: begin
                state <= S_IDLE;
                received <= 1;
            end
            default: state <= S_IDLE;
        endcase
    end
end

endmodule
```

Uart

ورودی‌ها:

- rstN
- clk
- s_in
- send
- send_data

خروجی‌ها:

- s_out
- sent
- received
- received_data
- check_receive_parity

این ماژول کلی آزمایش است و در آن برای هر یک گیرنده و فرستنده، یک شیء ساخته شده است. کد وریلاگ این بخش به شکل زیر است:

```
module uart #(
    parameter START_SIG = 1
) (
    input           rstN,
    input           clk,
    input           s_in,
    input           send,
    input   [6:0]   send_data,
    output          s_out,
    output          sent,
    output          received,
    output  [6:0]   received_data,
    output          check_receive_parity
);

sender #(START_SIG) SENDER (rstN, clk, send, send_data, s_out, sent);
receiver #(START_SIG) RECEIVER (rstN, clk, s_in, received, check_receive_parity, received_data);

endmodule
```

Top

در این ماژول به بررسی کارکرد این دستگاه می‌پردازیم. ابتدا از ماژول Uart دو شیء می‌سازیم.

برای تست اول، از فرستنده‌ی اول برای گیرنده‌ی دوم پیغام Hello را می‌فرستیم. برای تست دوم از فرستنده‌ی دوم برای گیرنده‌ی اول پیغام Bye را ارسال می‌کنیم و نتایج را در transcript و waveform مشاهده خواهیم کرد.

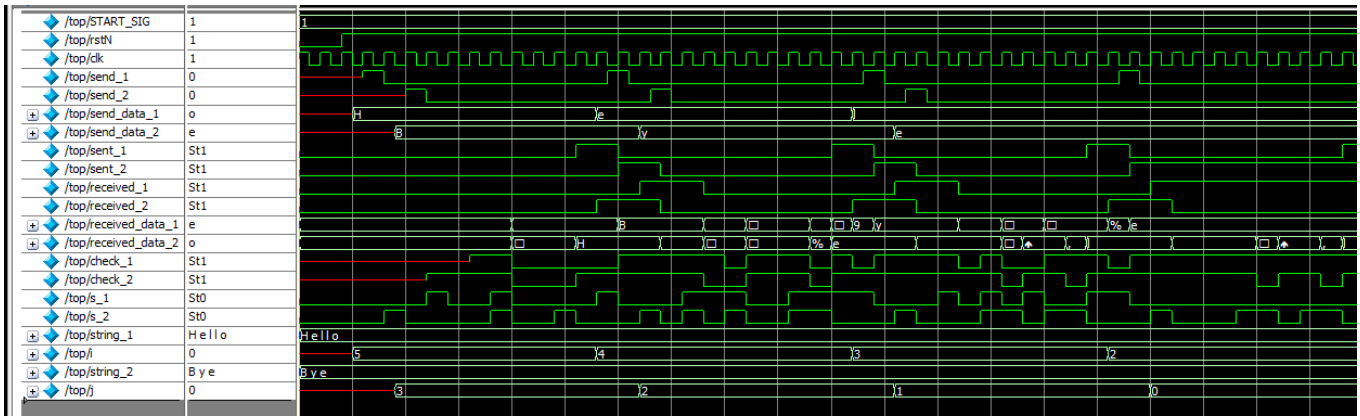
کد وریلاگ این بخش از مدار به شکل زیر است:

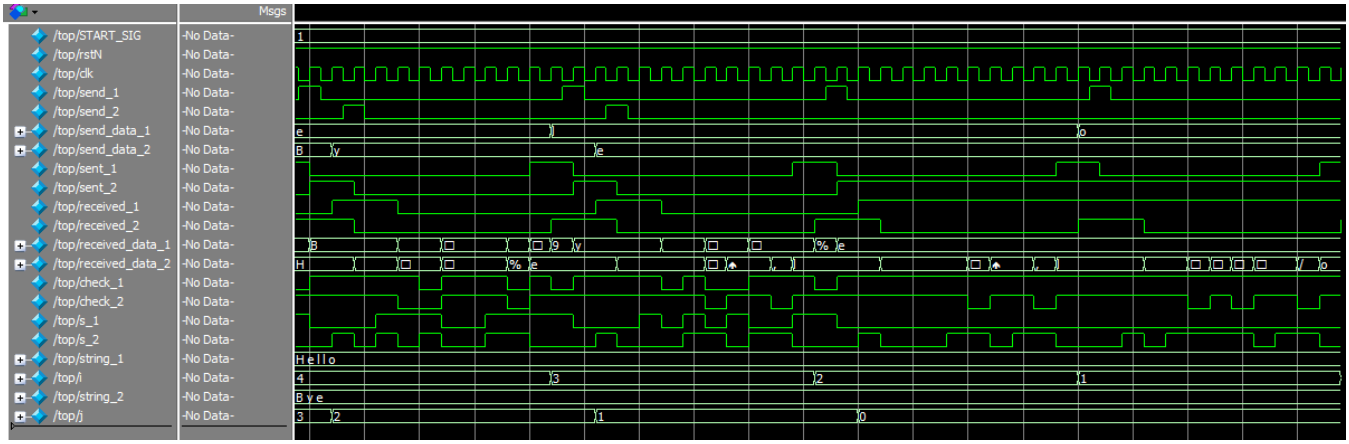
```
module top ();
localparam START_SIG = 1;

reg     rstN = 0, clk = 1;
reg     send_1, send_2;
reg     [6:0]   send_data_1, send_data_2;

wire    sent_1, sent_2;
wire    received_1, received_2;
wire    [6:0]   received_data_1, received_data_2;
wire    check_1, check_2;

uart #(START_SIG) U1 (rstN, clk, s_1, send_1, send_data_1, s_2, sent_1, received_1, received_data_1, check_1);
uart #(START_SIG) U2 (rstN, clk, s_2, send_2, send_data_2, s_1, sent_2, received_2, received_data_2, check_2);

always #10 clk = ~clk;

initial begin
    #40 rstN = 1;
end

wire [6:0]  string_1 [4:0];
assign string_1[4] = "H";
assign string_1[3] = "e";
assign string_1[2] = "l";
assign string_1[1] = "l";
assign string_1[0] = "o";
integer i;

initial begin
    #50;
    for (i = 5; i > 0; i = i - 1) begin
        send_data_1 = string_1[i-1];
        #10 send_1 = 1;
        #20 send_1 = 0;

        wait (sent_1);
        $display("%c sent from U1", send_data_1);
        wait (received_2);
        $display("%c received by U2. check: %d", received_data_2, check_2);
    end
    $stop();
end
```

```
wire [6:0]  string_2 [2:0];
assign string_2[2] = "B";
assign string_2[1] = "y";
assign string_2[0] = "e";
integer j;

initial begin
    #90;
    for (j = 3; j > 0; j = j - 1) begin
        send_data_2 = string_2[j-1];
        #10 send_2 = 1;
        #20 send_2 = 0;

        wait (sent_2);
        $display("%c sent from U2", send_data_2);
        wait (received_1);
        $display("%c received by U1. check: %d", received_data_1, check_1);
    end
end

endmodule
```

Waveform

حاصل شبیه‌سازی ماژول top به شکل زیر است:






```
add wave -position end  sim:/top/j
VSIM 23> run 1000
# H sent from U1
# H received by U2. check: 1
# B sent from U2
# B received by U1. check: 1
# e sent from U1
# e received by U2. check: 1
# y sent from U2
# y received by U1. check: 1
# l sent from U1
# l received by U2. check: 1
# e sent from U2
# e received by U1. check: 1
# l sent from U1
# l received by U2. check: 1
VSIM 24> run 1000
# o sent from U1
# o received by U2. check: 1
# Break in Module top at E:/Documents/DSD-az/7/top.v line 42
```