پیش گزارش آزمایش هفتم

روژین تقی زادگان ۴۰۱۱۰۵۷۷۵
رادین شاه دائی ۴۰۱۱۰۶۰۹۶
باربد شهرآبادی ۴۰۱۱۰۶۱۲۵

استاد محسن انصاری

توصیف آزمایش:

هدف از انجام این آزمایش طراحی یک Universal Asynchronous Receiver Transmitter (UART) می باشد. در قسمت ارسال کننده این دستگاه هربار یک کد ۷بیتی ASCII بصورت سریال ارسال می گردد. در ابتدا یک بیت شروع (Start)، سپس یک بیت توازن (Parity) و بعد ۷ بیت داده ارسال می شوند. در انتها نیز حداقل یک بیت خاتمه (Stop) ارسال می شود (در مجموع ۱۰ بیت).

در قسمت گیرنده نیز پس از دریافت بیت شروع (Start)، ۸ بیت مربوط به داده و توازن (Parity) بصورت سریال دریافت شده و در یک ثبات (Register) ۸ بیتی ذخیره می شود.

در این آزمایش، باید یک UART طراحی کنیم. یک UART شامل دو بخش sender و receiver است که باعث می‌شود دو UART بتوانند با هم ارتباط داشته باشند و به هم داده ارسال کنند. برای سادگی کار و تست، در این ماژول، اطلاعات را بین sender و receiver یک UART ارسال می‌کنیم تا از صحت کار دو ماژول sender و receiver مطمئن شویم.

ماژول sender:

کد این ماژول در زیر آمده‌است:

```
module sender # (parameter START_SIG = 0) (
    input           resetNot,
    input           CLK,
    input           start,
    input   [6:0]   dataIn,
    output  reg     signalOut,
    output  reg     sent
);

localparam IDLE     = 0;
localparam START    = 1;
localparam PARITY   = 2;
localparam SEND     = 3;
localparam STOP     = 4;

reg [2:0]           state;
reg [6:0]           data;
reg [2:0]           dataIndex;
reg                 prevStart = 0;

wire                paritySig;
assign              paritySig = ^data;
```

```
always @(posedge CLK or negedge resetNot) begin
    if (~resetNot) begin
        state       <= IDLE;
        dataIndex   <= 0;
        signalOut   <= 0;
        sent        <= 0;
        prevStart   = 0;
    end
    else begin
        prevStart <= start;
        case (state)
            IDLE: begin
                if (start && prevStart == 0) begin
                    dataIndex   <= 0;
                    data        <= dataIn;
                    state       <= START;
                    sent        <= 0;
                end
            end
            START: begin
                signalOut       <= START_SIG;
                state           <= PARITY;
            end
            PARITY: begin
                signalOut       <= paritySig;
                state           <= SEND;
            end
            SEND: begin
                signalOut       <= data[dataIndex];
                dataIndex       <= dataIndex + 1;
                if (dataIndex == 6)
                    state       <= STOP;
            end
            STOP: begin
                signalOut       <= ~START_SIG;
                state           <= IDLE;
                sent            <= 1;
            end
            default: state      <= IDLE;
        endcase
    end
end

endmodule
```

این ماژول شامل پارامتر START_SIG می‌باشد که نشان‌گر این است که سیگنال شروع ارسال داده چه مقداری باشد. سپس ورودی‌های reset، clock، start و data را می‌گیرد که ورودی start برای شروع ارسال داده و ورودی data نیز داده‌ای که باید ارسال شود است. خروجی‌های این ماژول شامل سیگنال signal out بوده که نشانگر سیگنالی‌است که قرار است به صورت سریال، داده را ارسال کند. همچنین خروجی sent را داریم که نشانگر اتمام ارسال داده است. این ماژول شامل ۵ حالت IDLE, START, PARITY, SEND, STOP است که این ماژول به صورت متوالی بین این حالات تغییر می‌کند.

ماژول receiver:

کد این ماژول در زیر آمده‌است:

```
module receiver # (parameter START_SIG = 0) (
    input               resetNot,
    input               CLK,
    input               signalIn,
    output  reg         received,
```

```
    output              parityChecked,
    output  reg [6:0]   data
);

localparam IDLE       = 0;
localparam PARITY     = 1;
localparam RECEIVE    = 2;
localparam STOP       = 3;

reg [1:0]   state;
reg [2:0]   dataIndex;
reg         parityReceived;
wire        actualParity;

assign actualParity = ^data;
assign parityChecked = actualParity == parityReceived;

always @(posedge CLK or negedge resetNot) begin
    if (~resetNot) begin
        state <= IDLE;
        dataIndex <= 0;
        received <= 0;
        data <= 0;
    end
    else begin
        case (state)
            IDLE: begin
                if (signalIn == START_SIG) begin
                    dataIndex <= 0;
                    data <= 0;
                    state <= PARITY;
                    received <= 0;
                end
            end
            PARITY: begin
                parityReceived <= signalIn;
                state <= RECEIVE;
            end
            RECEIVE: begin
                data[dataIndex] <= signalIn;
                dataIndex <= dataIndex + 1;
                if (dataIndex == 6) begin
                    state <= STOP;
                end
            end
            STOP: begin
                state <= IDLE;
                received <= 1;
            end
            default: state <= IDLE;
        endcase
    end
end

endmodule
```

این ماژول شامل پارامتر START_SIG می‌باشد که نشان‌گر این است که سیگنال شروع ارسال داده چه مقداری باشد. سپس ورودی‌های reset، clock، signalIn را می‌گیرد که ورودی signalIn همان سیگنالی‌ست که از sender می‌گیرد. خروجی این ماژول شامل data است که همان داده‌ای است که از ورودی signalIn به صورت سریال گرفته است، همچنین شامل parityCheck است که درستی parity ارسالی را چک می‌کند. همچنین شامل خروجی received است که نشانگر اتمام کار گیرنده است. این ماژول شامل ۵ حالت IDLE , PARITY, RECEIVE, STOP است که این ماژول به صورت متوالی بین این حالات تغییر می‌کند.

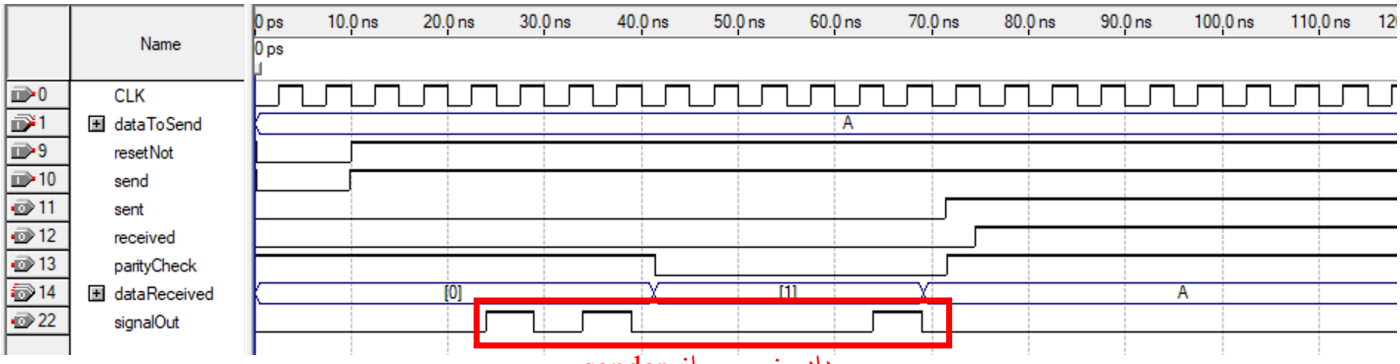
تست:

برای تست این دو ماژول، یک ماژول UART استفاده می‌کنیم و مقدار A را از sender به receiver ارسال می‌کنیم. این ماژول در زیر آمده‌است و همچنین خروجی waveform نیز در زیر می‌بینید.

```
module uart #(
    parameter START_SIG = 1
) (
    input           resetNot,
    input           CLK,
    input           send,
    input   [6:0]   dataToSend,
    output          signalOut,
    output          sent,
    output          received,
    output  [6:0]   dataReceived,
    output          parityCheck
);

sender #(START_SIG) SENDER (resetNot, CLK, send, dataToSend, signalOut, sent);
receiver #(START_SIG) RECEIVER (resetNot, CLK, signalOut, received, parityCheck, dataReceived);

endmodule
```



داده خروجی از sender