به نام خدا

آزمایش هشتم

آزمایشگاه طراحی سیستم‌های دیجیتال

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نویسندگان:

رادین چراغی ۴۰۱۱۰۵۸۱۵

امیرمحمد محفوظی ۴۰۱۱۰۶۴۶۹

سیدعلی جعفری ۴۰۰۱۰۴۸۸۹

تاریخ ارائه تکلیف:

۱۴۰۳/۰۵/۰۸

مقدمه

در این آزمایش باید یک Universal Asynchronous Receiver Transmitter یا همان UART طراحی کنیم. در قسمت Transmitter این دستگاه هر بار یک کد ۷ بیتی ASCII به صورت سریال به گیرنده ارسال می‌شود. در ابتدا یک بیت شروع (Start)، سپس یک بیت Parity و پس از آن ۷ بیت کد ASCII فرستاده می‌شود. در نهایت فرستنده یک بیت خاتمه (Stop) را ارسال می‌کند. پس در مجموع ۱۰ بیت از فرستنده به گیرنده ارسال می‌شود.

در قسمت گیرنده نیز پس از دریافت بیت شروع، ۸ بیت داده (parity و بیت‌های کد) در یک رجیستر ۸ بیتی ذخیره می‌شوند. در انتها نیز بیت stop توسط گیرنده دریافت می‌شود.

شرح آزمایش

در هنگام طراحی باید سه ماژول Sender، Receiver و UART را طراحی کنیم. حال به بررسی هر یک از این ماژول‌ها می‌پردازیم.

ماژول Sender

ابتدا به ورودی‌ها و خروجی‌های این ماژول می‌پردازیم.

ورودی‌ها

- rstN: همان ریست مدار بوده که فعال پایین می‌باشد.
- clk: همان کلاک مدار است.
- dataIn: ورودی ۷ بیتی که داده‌ی ورودی به بخش sender می‌باشد.

خروجی‌ها

- signalOut: سیگنال خروجی مدار بوده که به صورت سریال از فرستنده به گیرنده ارسال می‌شود.
- sent: هرگاه فرایند ارسال از فرستنده به گیرنده خاتمه یابد فعال می‌شود.

ابتدا رجیسترهای لازم را تعریف می‌کنیم. دو رجیستر data و index_of_data به ترتیب برای نگهداری داده ورودی و تعداد بیت‌های ارسال شده از فرستنده تا به حال استفاده می‌شوند. رجیستر current_state استیت فعلی را نگه می‌دارد که در ادامه درباره استیت‌ها توضیح می‌دهیم. در نهایت رجیستر prev_start نیز مشخص کننده این می‌باشد که آیا استیت قبلی start بوده یا خیر. همچنین وایر parity_bit را تعریف می‌کنیم که همان xor بیت‌های data می‌باشد.

```
module Sender # (parameter START_STOPN = 0) (
    input rstN, clk, start,
    input [6:0] dataIn,
    output reg signalOut, sent
);

// HELP REGISTERS
reg prev_start = 0;
reg [2:0] current_state;
reg [6:0] data;
reg [2:0] index_of_data; // number of bits have been transmitted minus one

// STATE SPECIFIER PARAMETERS
localparam REST = 0;
localparam START = 1;
localparam PARITY = 2;
localparam TRANSMIT = 3;
localparam STOP = 4;

// PARITY BIT : XOR OF DATA BITS
wire partiy_bit;
assign partiy_bit = ^data;
```

از توضیحات مقدمه می‌توان متوجه شد این ماژول پنج استیت دارد. این استیت‌ها عبارتند از :

- REST: این حالت، همان استیت ابتدایی می‌باشد که همواره به صورت پیش‌فرض در این استیت قرار داریم. در این حالت در صورتی که بیت ورودی start فعال باشد به استیت START خواهیم رفت و خروجی sent و index_of_data صفر خواهند شد و داده ورودی داخل رجیستر data قرار می‌گیرد. در غیر این‌صورت ماژول در این استیت باقی می‌ماند و اتفاقی رخ نخواهد داد.
- START: این استیت نشان‌دهنده آغاز عملیات ارسال می‌باشد و در این حالت، ماژول سیگنال خروجی را برابر با سیگنال start قرار می‌دهد. همچنین current_state از START به PARITY تغییر خواهد کرد.
- PARITY: در این استیت بیت parity_bit به عنوان خروجی سریال مدار قرار داده خواهد شد و مدار به استیت TRANSMIT می‌رود.
- TRANSMIT: این حالت مربوط به زمان ارسال کد ۷ بیتی می‌باشد. تا زمانی که در این استیت باشیم بیتی که در اندیس index_of_data در رجیستر data قرار دارد به عنوان خروجی سریال به گیرنده ارسال می‌شود. در صورتی index_of_data از شش کمتر باشد همچنان در این استیت باقی خواهیم ماند. در غیر این‌صورت فرایند ارسال به اتمام رسیده و به استیت STOP می‌رویم.

- STOP: در این استیت بیت STOP که همان صفر است را در خروجی سریال قرار می‌دهیم و خروجی sent را فعال می‌کنیم و به استیت REST می‌رویم.

تصویر زیر عملیات بالا را نشان می‌دهد.

```
always @(posedge clk or negedge rstN) begin
    if (~rstN) begin // reset stage
        signalOut <= 0;sent <= 0;
        current_state <= REST;
        index_of_data <= 0;
        prev_start = 0;
    end
    else begin
        prev_start <= start;
        case (current_state)
            REST: begin
                if (start && prev_start == 0) begin
                    index_of_data <= 0; sent <= 0;
                    data <= dataIn;
                    current_state <= START;
                end
            end
            START: begin
                signalOut <= START_STOPN;
                current_state <= PARITY;
            end
            PARITY: begin
                signalOut <= partiy_bit;
                current_state <= TRANSMIT;
            end
            TRANSMIT: begin
                signalOut <= data[index_of_data];
                index_of_data <= index_of_data + 1;
                if (index_of_data >= 6)
                    current_state <= STOP;
            end
            STOP: begin
                signalOut <= ~START_STOPN;
                current_state <= REST;
                sent <= 1;
            end
            default: current_state <= REST;
```

ماژول Receiver

ورودی‌ها

- rstN: همان ریست مدار بوده که فعال پایین می‌باشد.
- clk: همان کلاک مدار است.
- serial_in: بیت ورودی سریال می‌باشد که از فرستنده به آن ارسال می‌شود.

خروجی‌ها

- data: همان کد ۷ بیتی دریافت شده از فرستنده می‌باشد.
- parity_correctness: در این ماژول parity برای داده‌های دریافت شده محاسبه می‌شود و در صورتی که با parity دریافت شده برابر باشد این سیگنال فعال می‌شود.
- received: این بیت در پایان فرایند دریافت فعال می‌شود.

ابتدا رجیسترهای لازم را تعریف می‌کنیم. رجیستر index_of_data برای نگهداری تعداد بیت‌های ارسال شده از فرستنده تا به حال استفاده می‌شوند. رجیستر current_state استیت فعلی را نگه می‌دارد که در ادامه درباره استیت‌ها توضیح می‌دهیم. رجیستر received_parity بیت توازن دریافت شده از فرستنده را نگه می‌دارد. همچنین وایر correct_parity را تعریف می‌کنیم که همان xor بیت‌های data می‌باشد.

```
module Receiver # (parameter START_STOPN = 0) (
    input rstN, clk, serial_in,
    output reg received,
    output parity_correctness,
    output reg [6:0] data
);

wire        correct_parity;
reg [1:0]   current_state;
reg [2:0]   index_of_data;
reg         received_parity;

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

assign correct_parity = ^data;
assign parity_correctness = received_parity == correct_parity;
```

این ماژول چهار استیت دارد. این استیت‌ها عبارتند از :

- REST: این حالت، همان استیت ابتدایی می‌باشد که همواره به صورت پیش‌فرض در این استیت قرار داریم. در این حالت در صورتی که بیت دریافتی با یک برابر باشد به استیت PARITY خواهیم رفت و خروجی received و index_of_data صفر خواهند شد و مقدار صفر داخل رجیستر data قرار می‌گیرد. در غیر این‌صورت ماژول در این استیت باقی می‌ماند و اتفاقی رخ نخواهد داد.
- PARITY: در این استیت بیت parity_bit در رجیستر received_parity قرار گرفته و مدار به استیت RECEIVE می‌رود.
- RECEIVE: این حالت مربوط به زمان دریافت کد ۷ بیتی می‌باشد. تا زمانی که در این استیت باشیم بیت سریال ورودی در بیت با اندیس index_of_data در رجیستر data قرار می‌گیرد. در صورتی index_of_data از شش کمتر باشد همچنان در این استیت باقی خواهیم ماند. در غیر این‌صورت فرایند دریافت به اتمام رسیده و به استیت STOP می‌رویم.
- STOP: خروجی received را فعال می‌کنیم و به استیت REST می‌رویم.

تصویر زیر عملیات بالا را نشان می‌دهد.

```
always @(posedge clk or negedge rstN) begin
    if (~rstN) begin
        index_of_data <= 0; received <= 0;
        data <= 0;
        current_state <= REST;
    end
    else begin
        case (current_state)
            REST: begin
                if (serial_in == START_STOPN) begin
                    index_of_data <= 0; data <= 0;
                    current_state <= PARITY;
                    received <= 0;
                end
            end
            PARITY: begin
                received_parity <= serial_in;
                current_state <= RECEIVE;
            end
            RECEIVE: begin
                data[index_of_data] <= serial_in;
                index_of_data <= index_of_data + 1;
                if (index_of_data >= 6) begin
                    current_state <= STOP;
                end
            end
            STOP: begin
                current_state <= REST;
                received <= 1;
            end
            default: current_state <= REST;
        endcase
    end
end
```

در نهایت ماژول UART را طراحی می‌کنیم. این ماژول کلی آزمایش است و در آن برای از هر کدام از ماژول‌های گیرنده و فرستنده یک نمونه گرفته شده است.

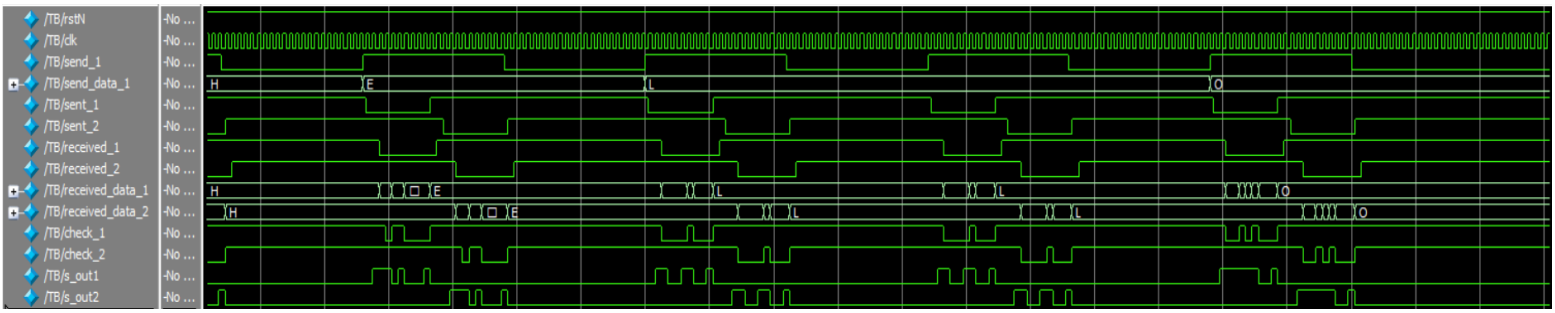
```
module UART #(
    parameter START_STOPN = 1
) (
    input rstN, clk, send,
    input [6:0]   sending_data,
    output s_out, sent, received,
    output [6:0] received_data,
    output parity_correctness
);
Sender #(START_STOPN) sender (rstN, clk, send, sending_data, s_out, sent);
Receiver #(START_STOPN) receiver (rstN, clk, s_out, received, parity_correctness, received_data);

endmodule
```

حال برای مدار یک تست‌بنچ طراحی می‌کنیم. در این ماژول پس از تعریف کردن سیگنال‌ها و رجیسترهای لازم برای دو نمونه گیری از ماژول UART، آزمون مدار را آغاز می‌کنیم. برای آزمون بهتر این ماژول نمونه‌های U1 و U2 از ماژول UART را به هم متصل می‌کنیم. به این‌صورت که خروجی received از U1 را به عنوان ورودی send به U2 می‌دهیم. همچنین خروجی received_data از U1 را به عنوان ورودی sending_data به U2 می‌دهیم. حال پس از شبیه‌سازی کلاک با استفاده از بلاک‌های always و initial، در یک بلاک initial تست مدار را قرار می‌دهیم. ورودی مدار در حقیقت همان "HELLO" می‌باشد که در واحدهای زمانی متوالی حروف آن را به عنوان ورودی به U1 می‌دهیم تا بتوانیم خروجی U2 را مشاهده کنیم.

```
module TB;
    localparam START_STOPN = 1;

    reg rstN, clk;
    reg send_1;
    reg [6:0] send_data_1;

    wire sent_1, sent_2;
    wire received_1, received_2;
    wire [6:0] received_data_1, received_data_2;
    wire check_1, check_2;
    wire s_out1, s_out2;

    UART #(START_STOPN) U1 (rstN, clk, send_1, send_data_1,s_out1, sent_1, received_1, received_data_1, check_1);
    UART #(START_STOPN) U2 (rstN, clk, received_1, received_data_1, s_out2, sent_2, received_2, received_data_2, check_2);

    initial clk = 0;
    always #5 clk = ~clk;
```

```
    initial begin
        rstN = 0;
        #220
        rstN = 1;
        send_data_1 = "H";
        send_1 = 1;
        #220
        send_1 = 0;
        #220
        send_data_1 = "E";
        send_1 = 1;
        #220
        send_1 = 0;
        #220
        send_data_1 = "L";
        send_1 = 1;
        #220
        send_1 = 0;
        #220
        send_data_1 = "L";
        send_1 = 1;
        #220
        send_1 = 0;
        #220
        send_data_1 = "O";
        send_1 = 1;
        #220
        send_1 = 0;
        #320
        $stop;
    end
endmodule
```

تصویر زیر نمایش waveform را نشان می‌دهد. همانطور که در تصویر مشخص است ورودی "HELLO" به تدریج از ورودی U1 به خروجی U2 انتقال پیدا کرده است و هر بار با انتقال هر حرف، خروجی sent_2 فعال شده است.

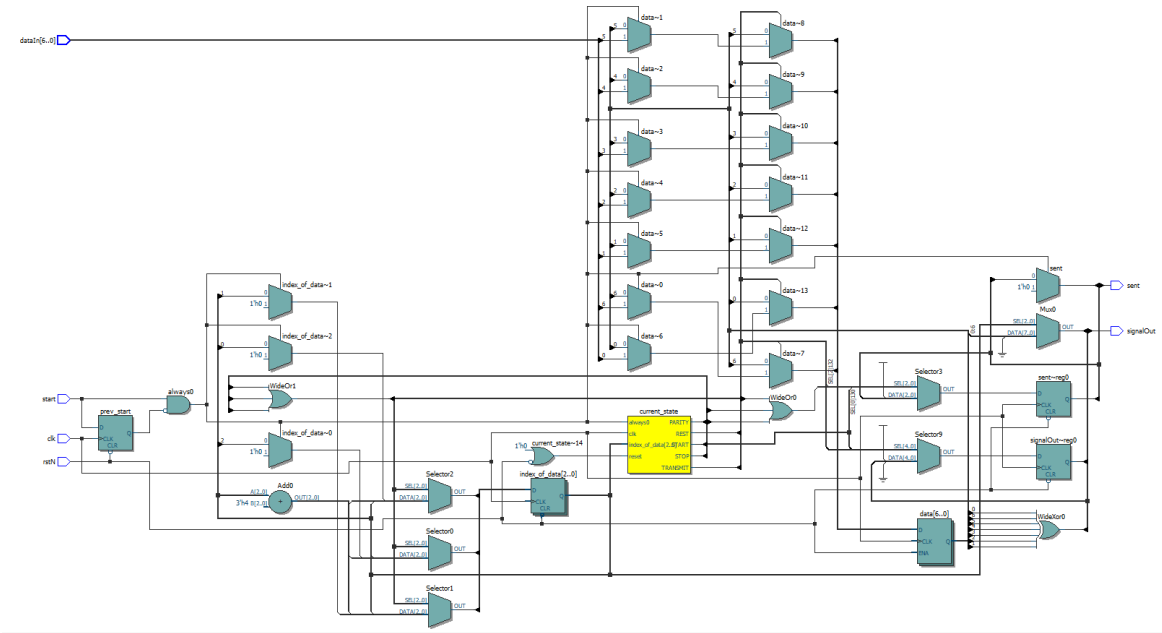


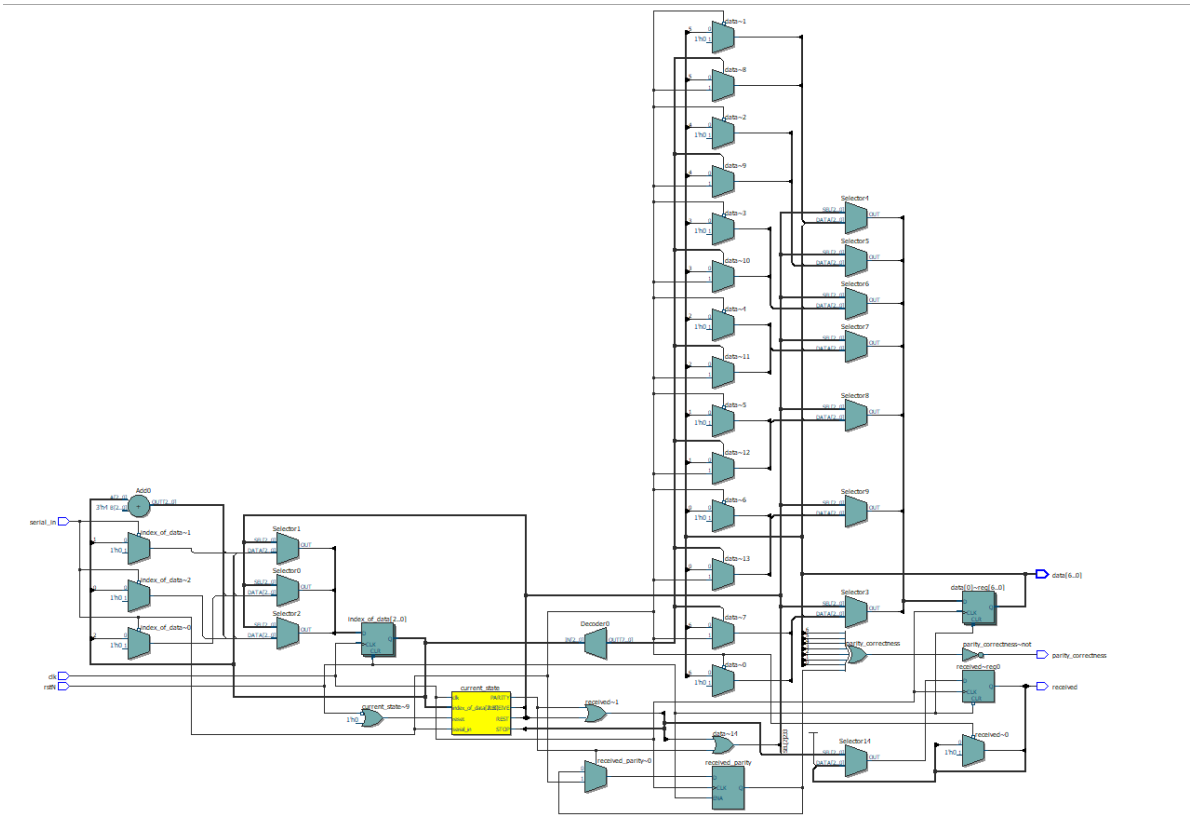
خروجی **flow summary**

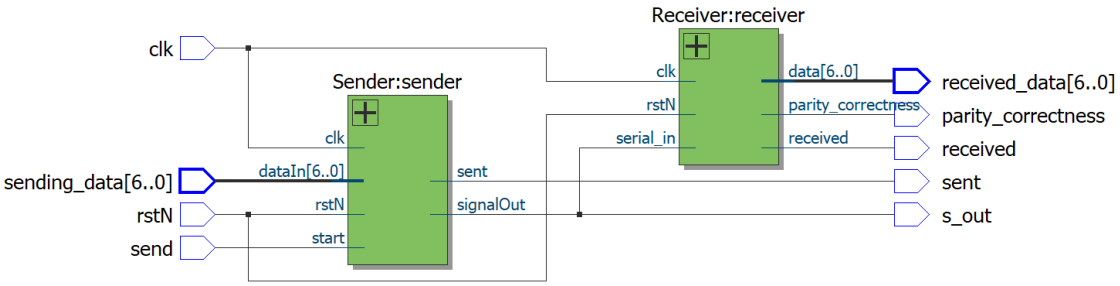
Flow Summary	
Flow Status	Successful - Mon Jul 29 19:19:01 2024
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	UART_7
Top-level Entity Name	UART
Family	Cyclone IV GX
Total logic elements	53 / 14,400 (< 1 %)
Total combinational functions	51 / 14,400 (< 1 %)
Dedicated logic registers	34 / 14,400 (< 1 %)
Total registers	34
Total pins	21 / 81 (26 %)
Total virtual pins	0
Total memory bits	0 / 552,960 (0 %)
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0 / 2 (0 %)
Total GXB Receiver Channel PMA	0 / 2 (0 %)
Total GXB Transmitter Channel PCS	0 / 2 (0 %)
Total GXB Transmitter Channel PMA	0 / 2 (0 %)
Total PLLs	0 / 3 (0 %)
Device	EP4CGX15BF14C6
Timing Models	Final

خروجی **RTL Viewer**

تصاویر زیر به ترتیب خروجی RTL viewer مربوط به ماژول Sender، Receiver و UART را نشان می‌دهند.






Receiver:receiver
clk
Sender:sender
clk
data[6..0]
received_data[6..0]
rstN
parity_correctness
parity_correctness
serial_in
received
received
clk
sent
sent
sending_data[6..0]
dataIn[6..0]
rstN
signalOut
s_out
rstN
start
send