دانشکده مهندسی کامپیوتر

آزمایشگاه طراحی سیستم‌های دیجیتال

آزمایش هفتم - طراحی UART

دکتر اجلالی، مهندس اثنی عشری

امیرمهدی کوششی — ۹۸۱۷۱۰۵۳
ایمان محمدی — ۹۹۱۰۲۲۰۷
شایان صالحی — ۹۹۱۰۵۵۶۱

۶ شهریور ۱۴۰۲

فهرست مطالب

۱ مقدمه

هدف از انجام این آزمایش این است که یک قسمت ارسال کننده و یک قسمت دریافت کننده داشته باشیم و ۷ بیت داده بین آنها انتقال دهیم.

۲ شرح آزمایش

در اینجا برای انتقال ۷ بیت، یک بیت را برای شروع رشته و یک بیت را برای خاتمه رشته درنظر می‌گیریم. همچنین یک بیت Parity درکنار داده‌ها می‌فرستیم که از XOR تمامی بیت‌ها به دست می‌اید تا صحت داده دریافت شده را ارزیابی کنیم. در صورتی که بیت Parity با آنچه که می‌بایست باشد تطابق نداشت سینگال error را فعال خواهیم کرد.

۳ ساختار مدار

در اینجا به ساختار مدار این قسمت می‌پردازیم.

۱.۳ منطق فرستنده

در اینجا برای فرستنده پنج‌استیت خواهیم داشت. استیت اول idle خواهد بود که در اینجا هیچ اتفاقی صورت نمی‌گیرد و با فرستادن بیت ۱ در انتظار شروع هستیم. استیت دوم start بوده که بیت صفر را ارسال می‌کند. استیت سوم parity بوده که XOR تمامی بیت‌ها را محاسبه کرده و می‌فرستد. استیت چهارم data و استیت پنجم end بوده که بیت ۱ را به عنوان خاتمه رشته ارسال کرده و دوباره وارد استیت idle می‌شود.

در اینجا نیز کد قسمت فرستنده را مشاهده می‌کنید. در اینجا دو خروجی داریم که یکی نشان دهنده مشغول بودن فرستنده (busy) و دیگری خروجی خود بیت‌های فرستنده به صورت سریال است. چهار ورودی کلاک، ریست، داده و داده جدید هم داریم که ۷ بیت از این طریق به فرستنده وارد می‌شود و سیگنال ریست استیت فرستنده را در هر حالتی باشد به حالت idle بازمی‌گرداند.

```
module UARTSender (tx, busy, data, new_data, rstN, clk);

    output reg tx, busy;
    input [6:0] data;
    input new_data, rstN, clk;

    reg [2:0] state;
    reg [2:0] index_pointer;

    always @(posedge clk or negedge rstN) begin
        if (~rstN) begin
            state <= 0;
            index_pointer <= 0;
            tx <= 1;
            busy <= 0;
        end
        else begin
            if (state == 0 && new_data) begin
                state <= 1;
                index_pointer <= 0;
                busy <= 1;
            end
            else if (state == 1) begin
                state <= 2;
                tx <= 0;
            end
            else if (state == 2) begin
                state <= 3;
                tx <= ^data;
            end
            else if (state == 3) begin
                tx <= data[index_pointer];
                index_pointer <= index_pointer + 1;
                if (index_pointer == 6) begin
                    state <= 4;
                end
            end
            else if (state == 4) begin
                state <= 0;
                tx <= 1;
                busy <= 0;
            end
        end
    end
```

شکل ۱: مدل فرستنده

۲.۳ منطق گیرنده

همانند فرستنده برای گیرنده نیز ۵ استیت خواهیم داشت. این استیت‌ها همان idle ، start ، parity ، data و end خواهند بود. یک تفاوت در این است که در گیرنده نیز بیت parity را مجدد محاسبه می‌کنیم و اگر با بیت‌های دریافت شده مطابقت نداشت سیگنال خطا را فعال می‌کنیم. در اینجا نیز سه خروجی داریم، خروجی اول data بوده که داده‌های دریافتی از فرستنده هستند. خروجی دوم correct-data بوده که نشان دهنده صحت داده‌های دریافتی از طریق بیت parity هستند و خروجی سوم نیز new-data بوده که نشان دهنده ارتباط بین فرستنده و گیرنده است.
ورودی‌های هم به صورت کلاک، ریست و سریال دریافتی از فرستنده خواهند بود.

```
module UARTReciever (data, new_data, correct_data, rx, rstN, clk);
    input rx, rstN, clk;
    output reg new_data, correct_data;
    output reg [6:0] data;

    reg [1:0] state;
    reg [2:0] index_pointer;
    reg parity;

    always @(posedge clk or negedge rstN) begin
        if (~rstN) begin
            state <= 0;
            index_pointer <= 0;
            data <= 0;
            new_data <= 1;
        end
        else begin
            if (state == 0 && rx == 0) begin
                state <= 1;
                index_pointer <= 0;
                data <= 0;
                new_data <= 1;
            end
            else if (state == 1) begin
                state <= 2;
                parity <= rx;
            end
            else if (state == 2) begin
                data[index_pointer] <= rx;
                index_pointer <= index_pointer + 1;
                if (index_pointer == 6) begin
                    state <= 3;
                end
            end
            else if (state == 3) begin
                correct_data <= (^data == parity);
                state <= 0;
                new_data <= 0;
            end
```

شکل ۲: مدل گیرنده

۳.۳ منطق مدل اصلی

در مدل baud دو مدل فرستنده و گیرنده به هم متصل شده و براساس پارامتر baud rate کلاک فرستنده را تعین می‌کنیم. درواقع کلاک اصلی فقط برای مدل اصلی بوده و کلاک فرستنده از تقسیم نرخ کلاک بر rate baud به دست می‌اید. همین کار برای مدل گیرنده صورت می‌گیرد با این تفاوت که بررسی می‌کنیم که داده‌ها دریافت شده باشند. پیاده‌سازی این قسمت به این شکل خواهد بود:

```verilog
`include "sender.v"
`include "reciever.v"

module BaudSender (tx, busy, data, new_data, rstN, clk);
    parameter CLOCK_RATE = 50000000;
    parameter BAUD_RATE = 115200;

    output tx, busy;
    input [6:0] data;
    input new_data, rstN, clk;

    reg tick;
    reg [31:0] counter;

    UARTSender sender(tx, busy, data, new_data, rstN, tick);

    always @(posedge clk) begin
        if (~rstN) begin
            tick <= 0;
        end
        else begin
            if (counter < CLOCK_RATE / BAUD_RATE) begin
                counter <= counter + 1;
            end
            else begin
                tick <= ~tick;
                counter <= 0;
            end
        end
    end
endmodule
```

شکل ۳: قسمت اول مدل اصلی

```verilog
module BaudReciever (data, new_data, correct_data, rx, rstN, clk);
    parameter CLOCK_RATE = 50000000;
    parameter BAUD_RATE = 115200;

    input rx, rstN, clk;
    output new_data, correct_data;
    output [6:0] data;

    reg tick;
    reg [31:0] counter;

    UARTReciever reciever(data, new_data, correct_data, rx, rstN, tick);

    always @(posedge clk) begin
        if (~rstN) begin
            tick <= 0;
        end
        else if (new_data) begin
            if (counter < CLOCK_RATE / BAUD_RATE) begin
                counter <= counter + 1;
            end
            else begin
                tick <= ~tick;
                counter <= 0;
            end
        end
    end
endmodule
```

شکل ۴: قسمت دوم مدل اصلی

۴ شبیه‌سازی عملکرد مدل

در این قسمت به بررسی و توضیح مدل تست و نتایج آن می‌پردازیم.

۱.۴ مدل تست

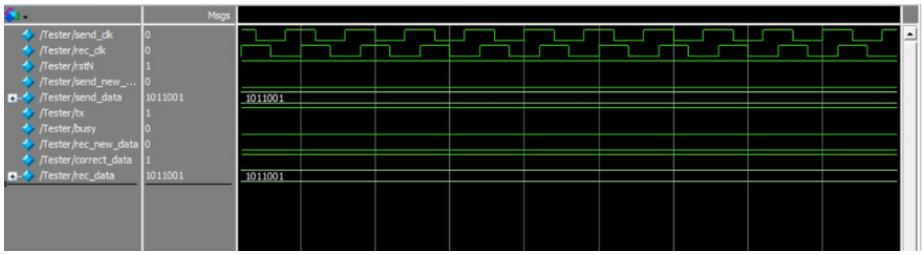
در مدل تست بنچ یک گیرنده و فرستنده تعریف می‌کنیم و این دو مدل کلاک‌های مختلفی دارند. سپس خروجی فرستنده را به ورودی گیرنده متصل کرده و پس از ریست کردن هر دو یک داده فرضی به آنها می‌دهیم. در آخر زمانی که از صحت داده‌های دریافتی مطمئن شدیم داده‌ها را در خروجی بررسی می‌کنیم.

```verilog
`include "baud.v"

`timescale 1ps/1ps
module Tester();

    reg send_clk, rec_clk, rstN, send_new_data;
    reg [6:0] send_data;

    wire tx, busy, rec_new_data, correct_data;
    wire [6:0] rec_data;

    BaudReciever reciever (rec_data, rec_new_data, correct_data, tx, rstN, rec_clk);

    BaudSender sender (tx, busy, send_data, send_new_data, rstN, send_clk);

    always begin
        #1
        send_clk = ~send_clk;
        #1
        rec_clk = ~rec_clk;
        if (correct_data == 1) begin
            send_new_data = 0;
        end
    end

    initial begin
        send_clk = 0;
        rec_clk = 0;
        send_data = 7'b1011001;
        rstN = 0;
        send_new_data = 1;
        #1000
        rstN = 1;
        $monitor("%d %d", tx, busy);
    end
endmodule
```

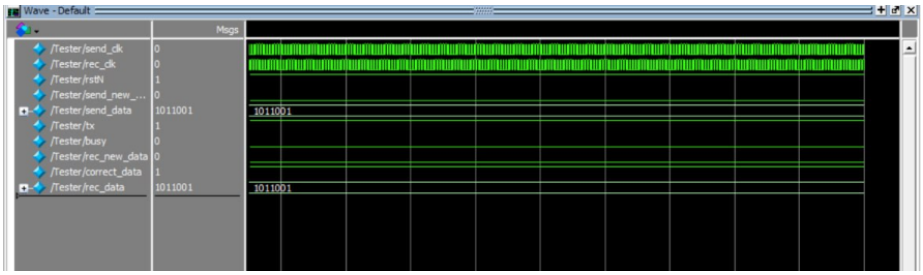
شکل ۵: تست بنچ برای گیرنده و فرستنده

۲.۴ نتایج تست

نتایج تست طراحی شده را به این صورت می‌توانید مشاهده کنید



شکل ۶: قسمت اول نتایج تست



شکل ۷: قسمت دوم نتایج تست