به نام خدا

آزمایشگاه طراحی سیستم‌های دیجیتال

آزمایش هفتم

طراحی UART

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

بهار ۱۴۰۱

استاد:

علیرضا اجلالی

دستیار آموزشی:

سحر رضاقلی

گروه:

هیربد بهنام

۹۹۱۷۱۳۳۳

علی نظری

۹۹۱۰۲۴۰۱

عرفان مجیبی

۹۹۱۰۵۷۰۷

فهرست

مقدمه

در این آزمایش قرار است یک UART یا Universal Asynchronous Receiver Transmitter طراحی کنیم که وظیفه ارسال یا دریافت بیت به بیت داده را دارند. سپس با اتصال دو تا از آنها به یکدیگر عملیات ارسال و دریافت داده را بررسی کنیم.

گزارش آزمایش

طراحی مدار اصلی

کد وریلاگ طراحی ماژول اصلی به صورت زیر است:

```
`timescale 1ns/1ns
module UART
  (
    input reset,                  // reset module
    input sysclk,                 // system clock
    input [15:0] divisor,         // to set baud rate

    input send,                   // start transmission
    input [7:0] tx_data_in,       // parallel transmit data <- proccessor
    output tx_data_out,           // serial transmit output

    input rx_data_in,             // serial receiver input
    output reg [7:0] rx_data_out,// parallel receiver data -> proccessor
    output reg ready,             // received byte is ready
    output error                  // check even parity
  );

  // generate tick
  wire tick;
  baude_rate_generator brg(
    .reset(reset),
    .clk(sysclk),
    .divisor(divisor),
    .tick(tick)
  );

  //RECEIVER
  reg rx_cs;
  reg rx_ns;
  reg [2:0] rx_counter;
  localparam IDLE = 1'b0;
  localparam DATA = 1'b1;

  // detect rx next state
  always @(*)
  begin
    if (rx_cs == IDLE)
```

```
      if (rx_data_in == 0)  // start bit
        rx_ns = DATA;
      else
        rx_ns = IDLE;

    else
      if (rx_counter == 7)  // last data bit
        rx_ns = IDLE;
      else
        rx_ns = DATA;
  end

  // at tick, go to next state
  always @(posedge tick)
  begin
    if (reset)
    begin
      rx_cs <= IDLE;
      rx_counter <= 0;
    end
    else
      rx_cs <= rx_ns;


    if (rx_cs == DATA)
    begin
      rx_data_out[rx_counter] <= rx_data_in;
      rx_counter <= rx_counter + 1'b1;
    end


    if (rx_cs != rx_ns)
      ready <= (rx_ns == IDLE);
  end

  // even parity
  assign error = ^ rx_data_out;

  //TRANSMITTER

  reg [1:0] tx_cs;
  reg [1:0] tx_ns;
  reg [2:0] tx_counter;
  localparam START = 2'b10;
```

```
82
83    //detect tx next state
84    always @(*)
85    begin
86      if (tx_cs == IDLE)
87        if (send == 1)
88          tx_ns = START;
89        else
90          tx_ns = IDLE;
91
92      else if (tx_cs == START)
93        tx_ns = DATA;
94
95      else
96      begin
97        if (tx_counter == 7)
98          tx_ns = IDLE;
99        else
100         tx_ns = DATA;
101     end
102
103   end
104
105   // at tick, go to tx next state
106   always @(posedge tick)
107   begin
108     if (reset)
109     begin
110       tx_cs <= IDLE;
111       tx_counter <= 0;
112     end
113     else
114       tx_cs <= tx_ns;
115
116
117     if (tx_cs == DATA)
118       tx_counter <= tx_counter + 1'b1;
119   end
120
121   assign tx_data_out =
122       (tx_cs == IDLE) ? 1'b1 :
123       (tx_cs == START) ? 1'b0 : tx_data_in[tx_counter];
124
125 endmodule
```

ورودی‌ها و خروجی‌های مدار شامل موارد زیر هستند:

- ورودی ریست.
- ورودی کلاک (مختص همین واحد است و مشترک نیست).
- ورودی divisor که مشخص می‌کند هر چند کلاک یک بار داده ارسال یا دریافت می‌شود. این مقدار را بر اساس توافق دوتا واحد UART که به هم متصل می‌کنیم تعیین می‌نماییم.
- ورودی send که شروع ارسال داده توسط transmitter را مشخص می‌کند.
- ورودی tx_data_in که دیتای ۸ بیتی است که می‌خواهیم ارسال شود.
- خروجی tx_data_out تک بیتی است که در هر مرحله ارسال می‌شود(می‌دانیم داده بیت به بیت ارسال می‌شود).
- ورودی rx_data_in که تک بیت دریافت شده است.
- خروجی rx_data_out که ۸ بیت دارد و در زمان پایان انتقال کل داده دریافت شده را به ما نمایش می‌دهد.
- خروجی ready که در زمان پایان انتقال یک می‌شود تا بدانیم دیتای داخل rx_data_out حاضر است.
- خروجی error که در صورت وجود خطا(بر اساس parity) یک می‌شود.

در ادامه به توضیح نحوه پیاده‌سازی و کارکرد مدار می‌پردازیم. ابتدا دقت کنید یک وایر به نام tick تعریف شده است که بعد از divisor بار کلاک برابر یک می‌شود و به ما نشان می‌دهد زمان دریافت یا ارسال بیت بعدی فرا رسیده است. این مقدار توسط baud_rate_generator تنظیم می‌شود که پس از این بخش به طور مجزا به توضیح آن خواهیم پرداخت.
برای طراحی UART باید قسمت‌های دریافت کننده و ارسال کننده داشته باشیم. مطابق آنچه در کد می‌بینید به توصیف بخش دریافت کننده(RECEIVER) و سپس ارسال کنندهTRANSMITTER می‌پردازیم.
در بخش دریافت کننده دو رجیستر rx_cs و rx_ns برای ذخیره کردن و تعیین حالت فعلی و حالت بعدی داریم. یک شمارنده rx_counter هم داریم که برای شمارش بیت‌های دریافتی و ورودی کاربرد دارد. در بخش دریافت کننده مدار در کل دو حالت بیکار(IDLE) و فعال(دریافت داده_ DATA) دارد. سپس دو بلاک always داریم.
بلاک اول برای تعیین حالت بعدی مدار است. اگر در حال حاضر مدار بیکار(IDLE) باشد، حالت بعدی به بیت ورودی بستگی دارد. اگر صفر باشد به حالت DATA خواهیم رفت که شروع فعالیت مدار است و در غیر این صورت در همان حالت بیکار خواهیم ماند. اما اگر حالت فعلی مدار IDLE نباشد؛ اگر شمارنده به ۷ رسیده باشد یعنی همه بیت‌ها دریافت شده باشند، کار مدار تمام است و به حالت IDLE می‌رود. اما اگر چنین نباشد، در همان حالت DATA خواهد ماند.
در بلاک دوم که بر اساس لبه بالارونده tick فعالیت می‌کند، تغییر حالت اتفاق میفتد. اگر ریست باشد که شمارنده و حالت به صفر تغییر می‌کنند. اما در غیر این صورت، حالت بعدی جانشین حالت کنونی می‌شود و همچنین در صورتی که در حالت DATA باشیم، دیتای ورودی به رجیستر rx_data_out هم منتقل شده و شمارنده یکی زیاد می‌شود. همچنین سیگنال ready در انتهای این بلاک زمانی یک میشود که از حالت DATA به حالت IDLE برویم یعنی کار مدار تمام شود.
در انتهای بخش دریافت کننده، سیگنال error بر اساس xor همه بیت‌های خروجی مشخص می‌شود.
در بخش ارسال کننده دو رجیستر tx_cs و tx_ns برای ذخیره کردن و تعیین حالت فعلی و حالت بعدی داریم. یک شمارنده tx_counter هم داریم که برای شمارش بیت‌های خروجی و ارسالی کاربرد دارد. در بخش ارسال کننده مدار علاوه دو حالت بیکار(IDLE) و فعال(دریافت داده_ DATA)، حالت START هم داریم. سپس دو بلاک always داریم.
بلاک اول برای تعیین حالت بعدی مدار است. اگر در حال حاضر مدار بیکار(IDLE) باشد، حالت بعدی به بیت send بستگی دارد. اگر صفر باشد مدار بیکار خواهد ماند اما اگر یک باشد، وارد حالت START می‌شود. اما اگر حالت فعلی مدار IDLE نباشد؛ اگر در حالت START باشد، باید به حالت بعدی که DATA است برود و در غیر این حالات، اگر شمارنده به ۷ رسیده باشد یعنی

همه بیت‌ها ارسال شده باشند، کار مدار تمام است و به حالت IDLE می‌رود. اما اگر چنین نباشد، در همان حالت DATA خواهد ماند.
در بلاک دوم که بر اساس لبه بالارونده tick فعالیت می‌کند، تغییر حالت اتفاق میفتد. اگر ریست باشد که شمارنده و حالت به صفر تغییر می‌کنند. اما در غیر این صورت، حالت بعدی جانشین حالت کنونی می‌شود و همچنین در صورتی که در حالت DATA باشیم، شمارنده یکی زیاد می‌شود.
مقدار tx_data_out که بیت ارسالی می‌باشد بر اساس حالت فعلی تعیین می‌شود. اگر در حالت IDLE باشیم، یک، اگر در حالت START باشیم صفر و اگر در حالت DATA باشیم برابر بیتی از ورودی است که شمارنده به آن اشاره دارد.

طراحی Baude Rate Generator

این واحد مسئولیت تعیین سینگال tick را دارد که به نوعی برای توافق بین دو واحد UART و بر اساس divisor تعیین می‌شود. طراحی آن به صورت زیر است:

```
`timescale 1ns/1ns
module baude_rate_generator
  (
    input reset,
    input clk,
    input [15:0] divisor,
    output tick
  );

  reg [15:0] counter;

  always @(posedge clk)
    if (reset)
      counter = 1;
    else
      if (counter == divisor)
        counter = 1;
      else
        counter = counter + 1'b1;

  assign tick = (counter == 1);

endmodule
```

همانطور که در کد مشخص است یک شمارنده با کلاک UART داریم که از یک تا divisor میشمارد و بعد از آن مجددا یک می‌شود. هر بار که شمارنده برابر یک شود، سیگنال tick یک می‌شود.

تست

در بخش تست همانطور که در دستور کار خواسته شده است، دو واحد را به همدیگر متصل کرده‌ایم و یک رشته کاراکتر اسکی(Hello, World!) را از یک واحد به دیگری ارسال کرده‌ایم:

```
`timescale 1ns/1ns
module TB;

  reg reset;
  reg clk1 = 1'b1;
  always @(clk1) clk1 <= #5 ~clk1;
  reg clk2 = 1'b1;
  always @(clk2) clk2 <= #7 ~clk2;


  reg [15:0] divisor1 = 7;
  reg [15:0] divisor2 = 5;

  // simplex comunication : uart1.tx_data_out -> uart2.rx_data_in
  wire simplex;


  reg send;
  reg [7:0] tx_data_in;

  UART uart1(

    .reset (reset),
    .sysclk (clk1),
    .divisor (divisor1),

    .send (send),
    .tx_data_in (tx_data_in),
    .tx_data_out (simplex)
  );

  wire [7:0] rx_data_out;
  wire ready;
  wire error;

  UART uart2 (
    .reset (reset),
    .sysclk (clk2),
    .divisor (divisor2),
```

```
40
41      .rx_data_in (simplex),
42      .rx_data_out (rx_data_out),
43      .ready (ready),
44      .error (error)
45    );
46
47    reg[13*8-1:0] text = "Hello, World!";
48    integer i = 0;
49
50    initial
51    begin
52
53      // set even parity bits
54      for (i = 12; i >= 0; i = i-1)
55        text[(i+1)*8-1] = ^ text[(i*8) +: 7];
56
57      reset = 1;
58      #70; // wait for 1 tick
59
60      reset = 0;
61      #70; // wait for 1 tick
62
63      send = 1;
64      for (i = 12; i >= 0; i = i-1)
65      begin
66        tx_data_in = text[(i*8) +: 8];
67        #700; // wait for 10 ticks
68      end
69      send = 0;
70
71      $stop;
72    end
73
74    always @(posedge ready)
75      $display("time = %6d ns",  $time,
76            " -> rx_data_out = %b (%c)", rx_data_out[6:0], rx_data_out
      [6:0],
77            " , parity error = ", error);
78
79  endmodule
```

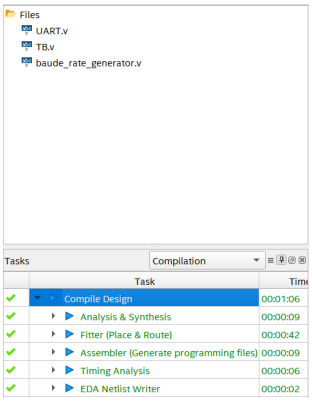
نتیجه اجرای تست به صورت زیر است:

```
# time =    826 ns -> rx_data_out = 1001000 (H) , parity error = 0
# time =   1526 ns -> rx_data_out = 1100101 (e) , parity error = 0
# time =   2226 ns -> rx_data_out = 1101100 (l) , parity error = 0
# time =   2926 ns -> rx_data_out = 1101100 (l) , parity error = 0
# time =   3626 ns -> rx_data_out = 1101111 (o) , parity error = 0
# time =   4326 ns -> rx_data_out = 0101100 (,) , parity error = 0
# time =   5026 ns -> rx_data_out = 0100000 ( ) , parity error = 0
# time =   5726 ns -> rx_data_out = 1010111 (W) , parity error = 0
# time =   6426 ns -> rx_data_out = 1101111 (o) , parity error = 0
# time =   7126 ns -> rx_data_out = 1110010 (r) , parity error = 0
# time =   7826 ns -> rx_data_out = 1101100 (l) , parity error = 0
# time =   8526 ns -> rx_data_out = 1100100 (d) , parity error = 0
# time =   9226 ns -> rx_data_out = 0100001 (!) , parity error = 0
```

شکل ۱: نتیجه اجرای تست

سنتز کد

این کار را در کوارتوس انجام می‌دهیم:



شکل ۲: سنتز کد

نتیجه‌گیری

در این آزمایش توانستیم با استفاده از وریلاگ، واحد UART را طراحی کنیم و سپس با اتصال دوتا از آنها انتقال یک رشته را انجام دادیم.