گزارش کار آزمایشگاه DSD

آزمایش شماره 7

11 اردیبهشت 1400
عرشیا اخوان
محمدحسین عبدی
علیرضا ایلامی

شماره آزمایش: **7**	موضوع: UART	تاریخ آزمایش: 11 اردیبهشت 1400
عرشیا اخوان **97110422**	محمدحسین عبدی **97110285**	علیرضا ایلامی **97101286**

مقدمه:

هدف از انجام این آزمایش طراحی یک Universal Asynchronous Receiver Transmitter - UART است.

شرح آزمایش:

یو آرت (UART) چیست؟

UART یکی از پر استفاده ترین پروتکل های ارتباط device-to-device است.

UART از دو بخش تشکیل شده است. یک بخش فرستنده و یک بخش گیرنده.

هدف این است که اگر یک داده ی موازی داریم، بتوانیم این را به صورت سری روی یک تک سیم منتقل کنیم. یعنی کلا هدفش تبدیل داده موازی به داده سریال است. و اینکه این داده سریال Decode شود و آن را به داده موازی برگرداند.

در واقع هدف اصلی UART انتقال داده در مسیرهای طولانی و اطمینان از سلامت داده دریافتی (به کمک بیت Parity که بعدا به آن می پردازیم) است.

بخش فرستنده:

یک داده موازی (در این آزمایش به پهنای 7 بیت) داریم.

این ها را باید به بیت های سری تبدیل کنیم.

ابتدا بیت start داده میشود که دریافت کننده متوجه شود از اینجا کار را آغاز کند.

سپس یک بیت به نام Parity داده میشود که برای بررسی این است که آیا بیتی در این انتقال تغییر یافته است یا نه. مثلا اگر noise داشته باشیم این اتفاق می افتد و یک بیت مقدارش عوض می شود.

پس از آن، بیت های داده ورود را به صورت سریال به خروجی می دهد. (در اینجا 7 بیت) و در پایان بیت end داده می شود و کار به پایان می رسد.

بخش گیرنده:

ورودی این بخش یک سیم است که همان بیت ها (و سه سیگنال start, parity, end) به صورت سریال، ورودی داده میشود.

کار این بخش این است که آن 7 بیت داده را جدا کند و به صورت موازی خروجی بدهد. و البته اگر parity داده شده با parity محاسبه شده تفاوت داشت، یعنی داده غلط به مقصد رسیده است و میگوید که خطا داریم.

این دو ماژول در هر UART مستقل از یکدیگر هستند.

برای ایجاد یک کانال یک طرفه، دو تا UART را باید به یکدیگر متصل کنیم. نحوه اتصال هم بدین صورت است که فرستنده اولی را به گیرنده دومی وصل می کنیم.

برای ایجاد کانال دو طرفه، علاوه بر ارتباط بالا، باید فرستنده دومی را نیز به گیرنده اولی وصل کنیم.

کد:

UART.v

```
module UART(clk,
        rstn,
        tx_start,
        tx_data_in,
        tx_channel_out,
```

```
        rx_channel_in,
        rx_data_out,
        rx_out_vaild);

    parameter BIT_LEN = 7;

    input clk, rstn, tx_start, rx_channel_in;
    input [BIT_LEN-1:0] tx_data_in;
    output tx_channel_out, rx_out_vaild;
    output [BIT_LEN-1:0] rx_data_out;

    TX #(.BIT_LEN(BIT_LEN)) tx (
    .clk(clk),
    .rstn(rstn),
    .start(tx_start),
    .data_in(tx_data_in),
    .channel_out(tx_channel_out)
    );

    RX #(.BIT_LEN(BIT_LEN)) rx (
    .clk(clk),
    .rstn(rstn),
    .channel_in(rx_channel_in),
    .data_out(rx_data_out),
    .is_valid(rx_out_vaild)
    );


endmodule
```

این ماژول اصلی است که در آن از دو ماژول RX و TX دو تا instance ساخته ایم.

و پارامتر bit_len را برابر با 7 قرار داده ایم.

TX.v

```
`define RST 1'b0
`define SEND 1'b1

module TX (clk,
        rstn,
        start,
        data_in,
```

```
      channel_out);

parameter BIT_LEN = 7;

input clk, rstn, start;
input [BIT_LEN - 1:0] data_in;
output reg channel_out;

reg state;
reg [BIT_LEN - 1:0] buffer;
reg [$clog2(BIT_LEN + 1 + 1 + 1):0] send_idx;

always @(posedge clk or negedge rstn) begin
   if (!rstn) begin
      state <= `RST;
   end
   else begin
      case (state)
         `SEND:
         begin
            if (send_idx > BIT_LEN + 2) begin
               state <= `RST;
            end
            else begin
               send_idx <= send_idx + 1;
            end
         end
         `RST:
         begin
            if (start) begin
               state <= `SEND;
            end
         end
      endcase
   end
end

always @(state or send_idx) begin
   case (state)
      `SEND:
      begin
         if (send_idx == 0) begin
            buffer      <= data_in;
            channel_out <= 1;
         end
         else if (send_idx == 1) begin
            channel_out <= ^buffer;
         end
         else if (1 < send_idx && send_idx <= BIT_LEN + 1) begin
            channel_out <= buffer[send_idx - 2];
```

```
            end
            else if (send_idx > BIT_LEN + 1) begin
               channel_out <= 1;
            end
         end
         `RST:
         begin
            buffer      <= 0;
            send_idx    <= 0;
            channel_out <= 0;
         end
      endcase
   end

endmodule
```

در ماژول TX که همان Transfer مان است، دو تا state داریم. یکی state = reset که هیچ کاری نمی کند و یک استیت state = send که داده را منتقل می کند.

این ماژول با بیت start شروع به کار میکند. و بعدی ها رو transfer میکند. و 7 بیت ورودی موازی را سریال میکنیم. (data_in) و channel_out همان یک سیگنال خروجی سریال است.

در always block اول (خطوط 20 تا 43) ابتدا مشخص میکنیم که در هر کلاک، متناسب با state فعلی، state بعدی را مشخص میکنیم. (اگر در حال انتقال بودیم و send_idx به بیشتر از تعداد بیتهای مورد انتقال شد، به استیت reset برمیگردیم.)

در always block دوم (خطوط 45 تا 70) باید کاری را که در هر state انجام دهیم، مشخص کنیم.

اگر send_idx = 0 باشد بیت start فرستاده شود.

اگر send_idx = 1 باشد بیت parity فرستاده شود.

اگر بین 2 تا 9 باشد باید آن بیت را بفرستد و اگر 10 شد بیت end را میفرستد.

RX.v

```
`define RST 1'b0
```

```
`define RECV 1'b1

module RX (clk,
           rstn,
           channel_in,
           data_out,
           is_valid);

    parameter BIT_LEN = 7;

    input clk, rstn, channel_in;
    output reg is_valid;
    output reg [BIT_LEN - 1:0] data_out;

    reg state, parity;
    reg [BIT_LEN - 1:0] buffer;
    reg [$clog2(BIT_LEN + 1 + 1 + 1):0] fetch_idx;

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            state <= `RST;
        end
        else begin
            case (state)
                `RECV:
                begin
                    if (channel_in && fetch_idx > BIT_LEN + 1) begin
                        state <= `RST;
                    end
                    else if (fetch_idx < BIT_LEN + 2) begin
                        fetch_idx <= fetch_idx + 1;
                    end
                end
                `RST:
                begin
                    if (channel_in) begin
                        state <= `RECV;
                    end
                end
            endcase
        end
    end

    always @(state or fetch_idx) begin
        case (state)
            `RECV:
            begin
                if (fetch_idx == 1)  begin
                    parity <= channel_in;
                end
```

```
            else if (1 < fetch_idx && fetch_idx <= BIT_LEN + 1) begin
               buffer[fetch_idx - 2] <= channel_in;
            end
            else if (channel_in && fetch_idx == BIT_LEN + 2) begin
               data_out <= buffer;
               is_valid <= (^buffer == parity);
            end
         end
         `RST:
         begin
            buffer    <= 0;
            fetch_idx <= 0;
         end
      endcase
   end

endmodule
```

در ماژول Receiver هم مشابه ماژول Transfer، دو تا always block داریم که اولی state بعدی را مشخص میکند و دومی هر کاری که در استیت فعلی باید انجام دهیم را انجام میدهد.

در بلاک اول (خط 20 تا 43) که state بعدی را ست میکنیم. اما یک نکته وجود دارد:

چون همیشه منتظر اولین سیگنال (start) است، همیشه Receiver یک کلاک از Transfer عقب تر است.

پس زمانی fetch_idx تغییر می کند که آن طرف send_idx تغییر کند و یک کلاک نیز رد شده باشد.

در بلاک دوم:

اگر در state = RECV باشیم، متناسب با fetch idx کارهای مختلفی میکند:

اگر fetch idx = 1 بود، parity را دریافت میکند.

اگر بین 1 و 8 بود، داده های سریال را در یک بافر نگه می دارد.

و در نهایت هم اگر fetch idx > 8 بود، سیگنال is valid را باید ست کند.

توضیح تکمیلی درباره parity:

سیگنال parity در واقع xor تمام بیت های بافر است. سیگنال is valid بررسی میکند که آیا سیگنال parity با xor داده های دریافت شده یکسان است یا خیر.

در صورتی که یکی بود، سیگنال is valid = 1 می شود.

توضیحات بافر:

بافر حالت backup دارد. که اگر احیانا ورودی تغییر کرد ما ورودی قبلی را داشته باشیم و آن را از دست ندهیم.

به محض اینکه بیت start آمد، ما ورودی را در بافر مینویسیم و شروع میکنیم.

Noise_gen.v

```
module NOISE_GENERATOR(clk,
                       sig_in,
                       en,
                       sig_out);

   input clk, sig_in, en;
   output sig_out;

   integer counter;

   assign sig_out = (en) ? ((counter == 5) ? !sig_in : sig_in) : sig_in;

   always @(posedge clk) begin
      if (counter < 15) begin
         counter = counter + 1;
      end
      else begin
         counter = 0;
      end
   end
endmodule
```

testbench.v

```
`define NULL 0
module testbench();

   parameter BIT_LEN = 7;
```

```
parameter clk_c   = 10;

reg clk, u0_rstn, u1_rstn, u0_start, u1_start, ng0_en, ng1_en;
reg [BIT_LEN - 1:0] u0_in, u1_in;
wire [BIT_LEN - 1:0] u0_out, u1_out;
wire u0_rx_channel, u0_tx_channel, u1_rx_channel, u1_tx_channel, u0_valid, u1_vaild;

UART #(.BIT_LEN(BIT_LEN)) uart0(
.clk(clk),
.rstn(u0_rstn),
.tx_start(u0_start),
.tx_data_in(u0_in),
.tx_channel_out(u0_tx_channel),
.rx_channel_in(u0_rx_channel),
.rx_data_out(u0_out),
.rx_out_vaild(u0_valid)
);

UART #(.BIT_LEN(BIT_LEN)) uart1(
.clk(clk),
.rstn(u1_rstn),
.tx_start(u1_start),
.tx_data_in(u1_in),
.tx_channel_out(u1_tx_channel),
.rx_channel_in(u1_rx_channel),
.rx_data_out(u1_out),
.rx_out_vaild(u1_valid)
);

NOISE_GENERATOR ng0 (
.clk(clk),
.sig_in(u0_tx_channel),
.en(ng0_en),
.sig_out(u1_rx_channel)
);

NOISE_GENERATOR ng1 (
.clk(clk),
.sig_in(u1_tx_channel),
.en(ng1_en),
.sig_out(u0_rx_channel)
);

initial begin
   $dumpfile("report/waveform.vcd");
   $dumpvars(0,uart0, uart1);
end

initial begin
   clk        = 0;
```

```
    forever clk = #(clk_c/2) ~clk;
end

integer data_file;
integer scan_file;
integer seed;
initial begin
    data_file = $fopen("seed.dat", "r");
    if (data_file == `NULL) begin
        $display("data_file handle was NULL");
        $finish;
    end
    scan_file = $fscanf(data_file, "%d", seed);
    if (scan_file == `NULL) begin
        $display("integer read error");
        $finish;
    end
end

integer i;
integer n;
initial begin
    u0_rstn = 0;
    u1_rstn = 0;
    ng1_en  = 0;
    ng0_en  = 0;
    #clk_c;
    u0_rstn = 1;
    u1_rstn = 1;
    n       = 50;

    $display("test connection uart0 -> uart1");
    for (i = 0; i < n; i++) begin
        u0_start = 1;
        u0_in    = {BIT_LEN{$random(seed)}};
        #clk_c
        u0_start = 0;
        while(uart1.rx.state || uart0.tx.state) begin
            #clk_c;
        end
        if (u0_in == u1_out && u1_valid) begin
            $display("#%d (u0:%b -> u1-v:%b(%b)) test passed", i, u0_in, u1_out, u1_valid);
        end
        else begin
            $display("#%d (u0:%b -> u1-v:%b(%b)) test failed", i, u0_in, u1_out, u1_valid);
        end
    end

    $display("test connection uart1 -> uart0");
    for (i = 0; i < n; i++) begin
```

```
            u1_start = 1;
            u1_in    = {BIT_LEN{$random(seed)}};
            #clk_c
            u1_start = 0;
            while(uart0.rx.state || uart1.tx.state) begin
               #clk_c;
            end
            if (u1_in == u0_out && u0_valid) begin
               $display("#%d (u1:%b -> u0-v:%b(%b)) test passed", i, u1_in, u0_out, u0_valid);
            end
            else begin
               $display("#%d (u1:%b -> u0-v:%b(%b)) test failed", i, u1_in, u0_out, u0_valid);
            end
         end

         n = 10;

         ng0_en = 1;
         #clk_c
         $display("test connection with noise uart0 -> uart1");
         for (i = 0; i < n; i++) begin
            u0_start = 1;
            u0_in    = {BIT_LEN{$random(seed)}};
            #clk_c
            u0_start = 0;
            while(uart1.rx.state || uart0.tx.state) begin
               #clk_c;
            end
            if (u0_in == u1_out && u1_valid) begin
               $display("#%d (u0:%b -> u1-v:%b(%b)) test passed", i, u0_in, u1_out, u1_valid);
            end
            else begin
               $display("#%d (u0:%b -> u1-v:%b(%b)) test failed (noise detected)", i, u0_in,
u1_out, u1_valid);
            end
         end

         ng1_en = 1;
         #clk_c;
         $display("test connection with noise uart0 -> uart1");
         for (i = 0; i < n; i++) begin
            u1_start = 1;
            u1_in    = {BIT_LEN{$random(seed)}};
            #clk_c
            u1_start = 0;
            while(uart0.rx.state || uart1.tx.state) begin
               #clk_c;
            end
            if (u1_in == u0_out && u0_valid) begin
               $display("#%d (u1:%b -> u0-v:%b(%b)) test passed", i, u1_in, u0_out, u0_valid);
```

```
            end
            else begin
                $display("#%d (u1:%b -> u0-v:%b(%b)) test failed (noise detected)", i, u1_in,
u0_out, u0_valid);
            end
        end
        $finish;
    end

endmodule
```

توضیحات تست بنچ:

یک ماژول noise generator قرار داده ایم که در یک سری کلاک خاص می آید یک بیت را not میکند تا تغییر کند.

در تست بنچ ابتدا دو ماژول UART و دو ماژول 1ng0, ng را instantiate میکنیم.

ابتدا انتقال از 0uart به 1uart را چک میکنیم.

سپس انتقال از 1uart به 0uart را چک میکنیم.

در ادامه ورودی را از noise generator رد میکنیم که در آن noise بیاندازد و درست بودن سیگنال parity را نیز تست کنیم.

یک بار از 0uart به 1uart

و بار دیگر برعکس.

برای بعضی از ورودی ها نویز ایجاد می شود که سیگنال parity این را نشان میدهد.

هرکدام از این انتقال ها را برای 50 تا ورودی بررسی میکنیم.

برای ساختن عدد رندوم هم مشابه آزمایش های قبل یک فایل .sh داریم که عدد رندوم را برایمان به کمک seed.dat تولید کند و موقع اجرا به تست بنچ داده می شود.

نتایج تست بنچ در پوشه report و فایل results.txt موجود است

Results.txt

```
VCD info: dumpfile report/waveform.vcd opened for output.
test connection uart0 -> uart1
#         0 (u0:1010001 -> u1-v:1010001(1)) test passed
#         1 (u0:1000110 -> u1-v:1000110(1)) test passed
#         2 (u0:0100001 -> u1-v:0100001(1)) test passed
#         3 (u0:1011010 -> u1-v:1011010(1)) test passed
#         4 (u0:0001001 -> u1-v:0001001(1)) test passed
#         5 (u0:1110111 -> u1-v:1110111(1)) test passed
#         6 (u0:0111111 -> u1-v:0111111(1)) test passed
#         7 (u0:1000001 -> u1-v:1000001(1)) test passed
#         8 (u0:1001010 -> u1-v:1001010(1)) test passed
#         9 (u0:1001101 -> u1-v:1001101(1)) test passed
#        10 (u0:0101100 -> u1-v:0101100(1)) test passed
#        11 (u0:0011110 -> u1-v:0011110(1)) test passed
#        12 (u0:1011110 -> u1-v:1011110(1)) test passed
#        13 (u0:0110110 -> u1-v:0110110(1)) test passed
#        14 (u0:1100000 -> u1-v:1100000(1)) test passed
#        15 (u0:0000100 -> u1-v:0000100(1)) test passed
#        16 (u0:1110100 -> u1-v:1110100(1)) test passed
#        17 (u0:0100011 -> u1-v:0100011(1)) test passed
#        18 (u0:1100101 -> u1-v:1100101(1)) test passed
#        19 (u0:0010110 -> u1-v:0010110(1)) test passed
#        20 (u0:0111001 -> u1-v:0111001(1)) test passed
#        21 (u0:1001101 -> u1-v:1001101(1)) test passed
#        22 (u0:1111011 -> u1-v:1111011(1)) test passed
#        23 (u0:1101101 -> u1-v:1101101(1)) test passed
#        24 (u0:1011101 -> u1-v:1011101(1)) test passed
#        25 (u0:0011010 -> u1-v:0011010(1)) test passed
#        26 (u0:1111001 -> u1-v:1111001(1)) test passed
#        27 (u0:1011000 -> u1-v:1011000(1)) test passed
#        28 (u0:1110111 -> u1-v:1110111(1)) test passed
#        29 (u0:0100100 -> u1-v:0100100(1)) test passed
#        30 (u0:0011010 -> u1-v:0011010(1)) test passed
#        31 (u0:1110110 -> u1-v:1110110(1)) test passed
#        32 (u0:1001000 -> u1-v:1001000(1)) test passed
#        33 (u0:0010001 -> u1-v:0010001(1)) test passed
#        34 (u0:1110000 -> u1-v:1110000(1)) test passed
#        35 (u0:1101011 -> u1-v:1101011(1)) test passed
#        36 (u0:1010011 -> u1-v:1010011(1)) test passed
#        37 (u0:0111100 -> u1-v:0111100(1)) test passed
#        38 (u0:0111101 -> u1-v:0111101(1)) test passed
#        39 (u0:0001101 -> u1-v:0001101(1)) test passed
#        40 (u0:0111111 -> u1-v:0111111(1)) test passed
#        41 (u0:1100001 -> u1-v:1100001(1)) test passed
#        42 (u0:1011011 -> u1-v:1011011(1)) test passed
#        43 (u0:1101000 -> u1-v:1101000(1)) test passed
```

```
#        44 (u0:1110110 -> u1-v:1110110(1)) test passed
#        45 (u0:1001010 -> u1-v:1001010(1)) test passed
#        46 (u0:1111010 -> u1-v:1111010(1)) test passed
#        47 (u0:1101000 -> u1-v:1101000(1)) test passed
#        48 (u0:0001101 -> u1-v:0001101(1)) test passed
#        49 (u0:0101010 -> u1-v:0101010(1)) test passed
test connection uart1 -> uart0
#         0 (u1:1100100 -> u0-v:1100100(1)) test passed
#         1 (u1:0000001 -> u0-v:0000001(1)) test passed
#         2 (u1:1010011 -> u0-v:1010011(1)) test passed
#         3 (u1:0110001 -> u0-v:0110001(1)) test passed
#         4 (u1:1111101 -> u0-v:1111101(1)) test passed
#         5 (u1:0111011 -> u0-v:0111011(1)) test passed
#         6 (u1:0010100 -> u0-v:0010100(1)) test passed
#         7 (u1:1111000 -> u0-v:1111000(1)) test passed
#         8 (u1:0111110 -> u0-v:0111110(1)) test passed
#         9 (u1:1100010 -> u0-v:1100010(1)) test passed
#        10 (u1:1111000 -> u0-v:1111000(1)) test passed
#        11 (u1:1011100 -> u0-v:1011100(1)) test passed
#        12 (u1:1100010 -> u0-v:1100010(1)) test passed
#        13 (u1:1001001 -> u0-v:1001001(1)) test passed
#        14 (u1:0000101 -> u0-v:0000101(1)) test passed
#        15 (u1:1010111 -> u0-v:1010111(1)) test passed
#        16 (u1:1110010 -> u0-v:1110010(1)) test passed
#        17 (u1:1110000 -> u0-v:1110000(1)) test passed
#        18 (u1:0101111 -> u0-v:0101111(1)) test passed
#        19 (u1:1111101 -> u0-v:1111101(1)) test passed
#        20 (u1:1111011 -> u0-v:1111011(1)) test passed
#        21 (u1:1010111 -> u0-v:1010111(1)) test passed
#        22 (u1:1110110 -> u0-v:1110110(1)) test passed
#        23 (u1:0110011 -> u0-v:0110011(1)) test passed
#        24 (u1:0010000 -> u0-v:0010000(1)) test passed
#        25 (u1:1110010 -> u0-v:1110010(1)) test passed
#        26 (u1:0000111 -> u0-v:0000111(1)) test passed
#        27 (u1:0100100 -> u0-v:0100100(1)) test passed
#        28 (u1:1101000 -> u0-v:1101000(1)) test passed
#        29 (u1:1111110 -> u0-v:1111110(1)) test passed
#        30 (u1:1011001 -> u0-v:1011001(1)) test passed
#        31 (u1:0101101 -> u0-v:0101101(1)) test passed
#        32 (u1:0111011 -> u0-v:0111011(1)) test passed
#        33 (u1:0010011 -> u0-v:0010011(1)) test passed
#        34 (u1:0111010 -> u0-v:0111010(1)) test passed
#        35 (u1:0111010 -> u0-v:0111010(1)) test passed
#        36 (u1:0011110 -> u0-v:0011110(1)) test passed
#        37 (u1:1011111 -> u0-v:1011111(1)) test passed
#        38 (u1:1010010 -> u0-v:1010010(1)) test passed
#        39 (u1:1000001 -> u0-v:1000001(1)) test passed
#        40 (u1:0011100 -> u0-v:0011100(1)) test passed
#        41 (u1:1000010 -> u0-v:1000010(1)) test passed
#        42 (u1:1110110 -> u0-v:1110110(1)) test passed
```

```
#         43 (u1:1100000 -> u0-v:1100000(1)) test passed
#         44 (u1:0111101 -> u0-v:0111101(1)) test passed
#         45 (u1:0111001 -> u0-v:0111001(1)) test passed
#         46 (u1:1110000 -> u0-v:1110000(1)) test passed
#         47 (u1:1010000 -> u0-v:1010000(1)) test passed
#         48 (u1:0101111 -> u0-v:0101111(1)) test passed
#         49 (u1:1110100 -> u0-v:1110100(1)) test passed
test connection with noise uart0 -> uart1
#          0 (u0:0000010 -> u1-v:0000011(0)) test failed (noise detected)
#          1 (u0:0100101 -> u1-v:0110101(0)) test failed (noise detected)
#          2 (u0:1010110 -> u1-v:1010110(1)) test passed
#          3 (u0:1010110 -> u1-v:0101100(1)) test failed (noise detected)
#          4 (u0:1000110 -> u1-v:1000110(1)) test passed
#          5 (u0:0010000 -> u1-v:0010000(0)) test failed (noise detected)
#          6 (u0:1110000 -> u1-v:1111000(0)) test failed (noise detected)
#          7 (u0:1001111 -> u1-v:1111000(0)) test failed (noise detected)
#          8 (u0:1000110 -> u1-v:1000110(1)) test passed
#          9 (u0:0100101 -> u1-v:0100101(0)) test failed (noise detected)
test connection with noise uart1 -> uart0
#          0 (u1:0000111 -> u0-v:0000011(0)) test failed (noise detected)
#          1 (u1:1100010 -> u0-v:0100010(0)) test failed (noise detected)
#          2 (u1:1011010 -> u0-v:1011010(1)) test passed
#          3 (u1:1010001 -> u0-v:1010001(1)) test passed
#          4 (u1:1101101 -> u0-v:1101001(0)) test failed (noise detected)
#          5 (u1:0100100 -> u0-v:1100100(0)) test failed (noise detected)
#          6 (u1:0110001 -> u0-v:0110001(1)) test passed
#          7 (u1:0000001 -> u0-v:0000001(1)) test passed
#          8 (u1:1111001 -> u0-v:1111101(0)) test failed (noise detected)
#          9 (u1:0011101 -> u0-v:1011101(0)) test failed (noise detected)
```

پایان