Systemy Dedykowane w Układach Programowalnych

Projekt – Hamming

Aleksandra Szydłowska
Joanna Koczwara

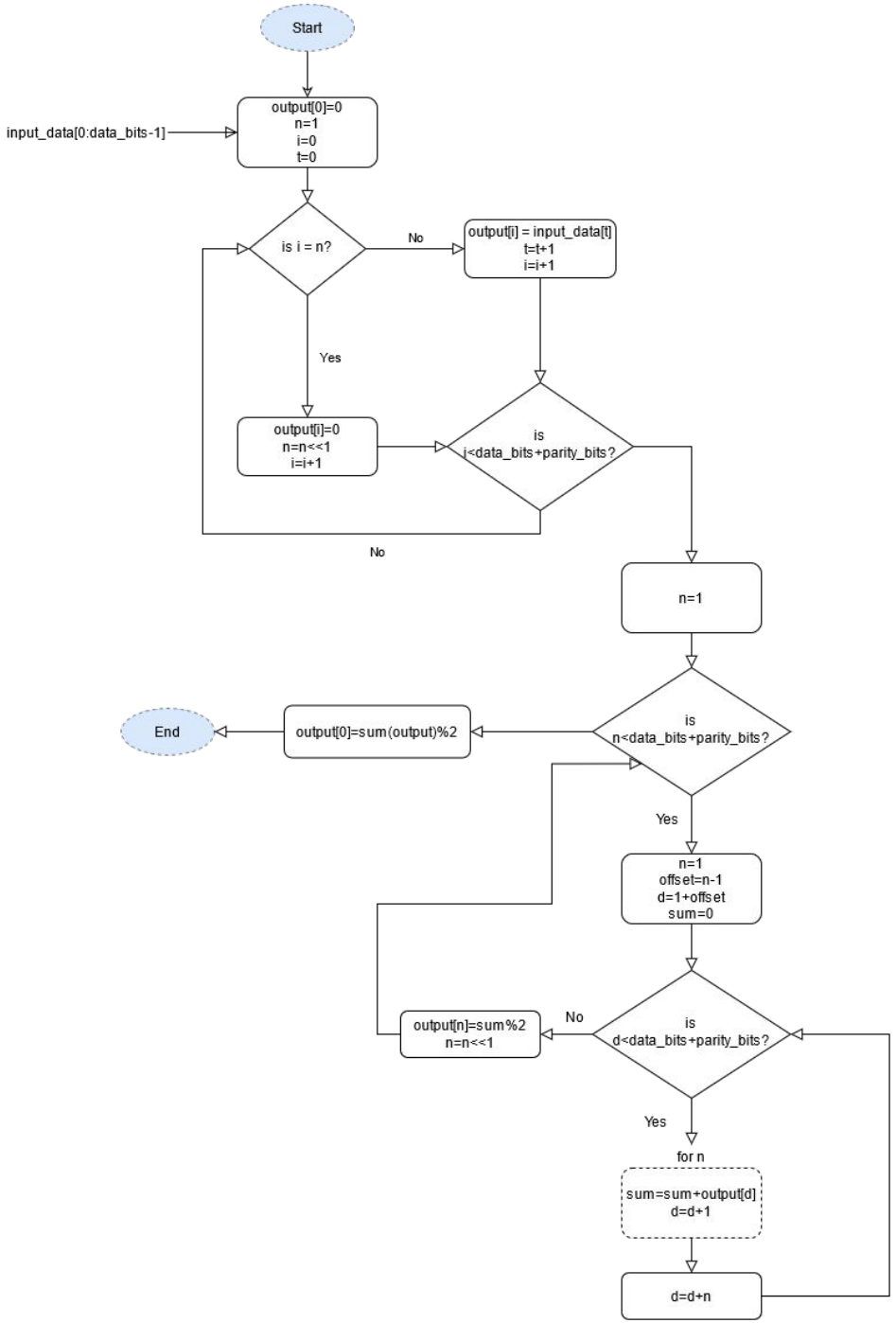
Hamming (32,26)

SECDED - single error correction, double error detection

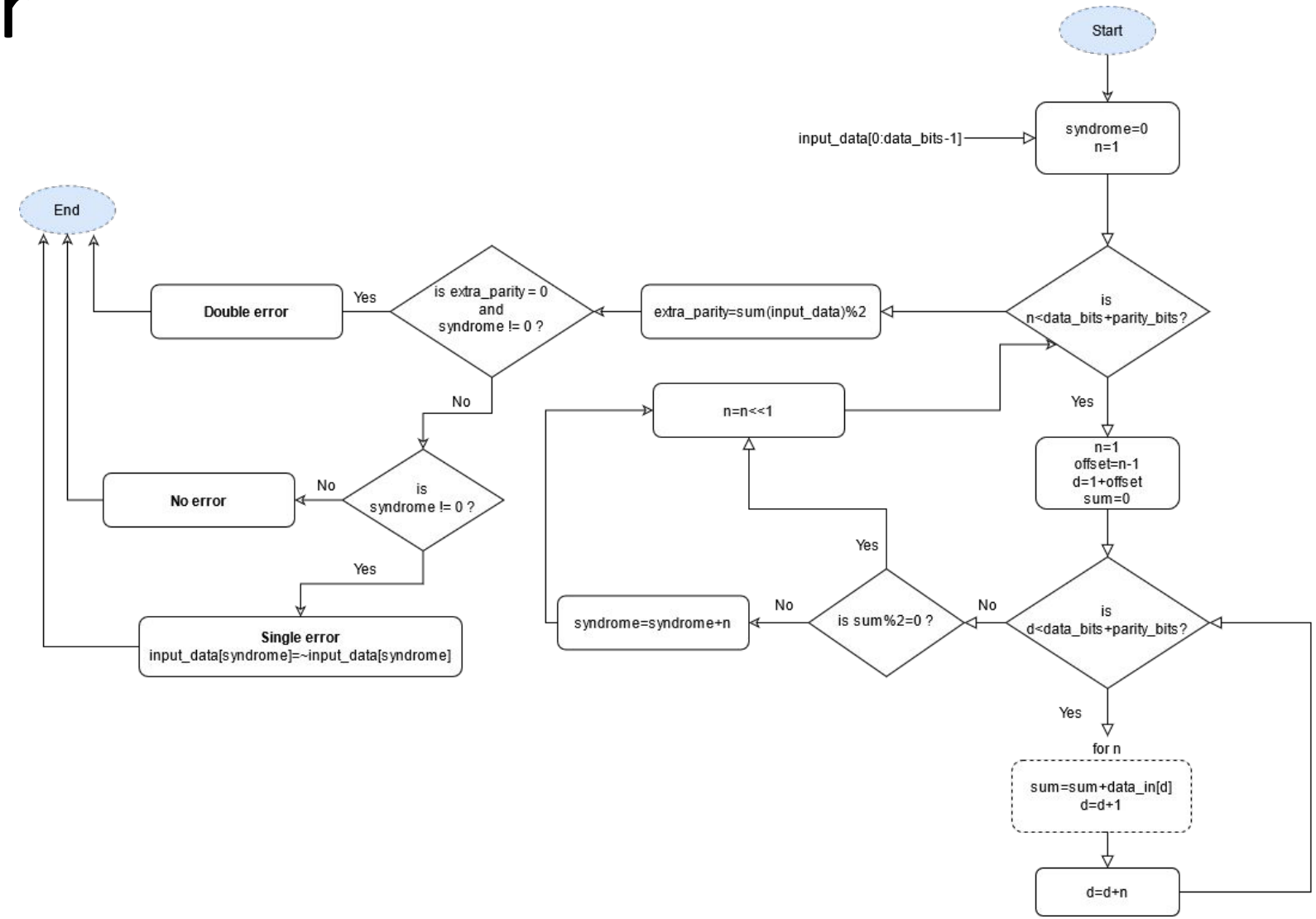
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	P0	P1	P2	D0	P3	D1	D2	D3	P4	D4	D5	D6	D7	D8	D9	D 10	P5	D 11	D 12	D 13	D 14	D 15	D 16	D 17	D 18	D 19	D 20	D 21	D 22	D 23	D 24	D 25
P0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
P1		X		X		X		X		X		X		X		X		X		X		X		X		X		X		X	
P2			X	X			X	X			X	X			X	X			X	X			X	X			X	X			X	X
P3					X	X	X	X					X	X	X	X					X	X	X	X					X	X	X	X
P4									X	X	X	X	X	X	X	X									X	X	X	X	X	X	X	X
P5																	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

P0 – dodatkowy bit parzystości, obliczany na podstawie wszystkich pozostałych bitów zakodowanego ciągu, zwiększa odległość Hamminga do 4 i pozwala na detekcje dwóch błędów z korekcją błędów pojedynczych lub na detekcje trzech błędów.

Algorytm Koder



Algorytm Dekoder



Koder – model behawioralny

```
for (i=1; i < all_bits; i=i+1) begin  //add parity bits
    if ( i == parity_position )begin
        encoder_output[i] = 0;
        parity_position = parity_position<<1;
    end
    else begin
        encoder_output[i] = data[data_counter];
        data_counter=data_counter+1;
    end
end

parity_position=1;
```

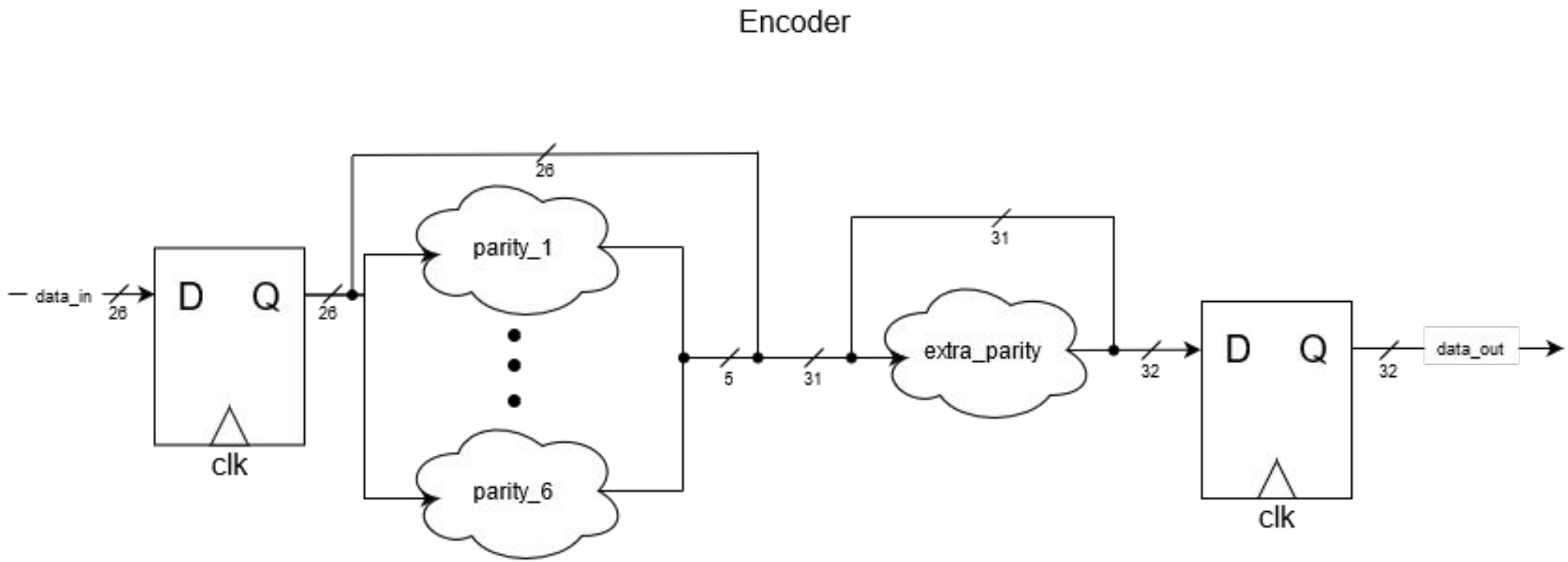
```
while ( parity_position < all_bits) begin //parity value
    offset = parity_position-1;
    output_data_counter = 1 + offset;
    sum=0;
    while (output_data_counter < all_bits) begin
        for (i=0; i < parity_position; i=i+1) begin
            sum=sum+encoder_output[output_data_counter];
            output_data_counter=output_data_counter+1;
        end
        output_data_counter=output_data_counter+parity_position;
    end
    encoder_output[parity_position] = sum%2;
    parity_position = parity_position<<1;
end
sum=0;
for(i=0; i< all_bits; i=i+1) begin //extra parity bit
    sum = sum + encoder_output[i];
end
encoder_output[0] = sum%2;
```

Dekoder – model behawioralny

```
extra_parity=0;
parity_position = 1;
sum=0;
syndrome=0;
while (parity_position<all_bits) begin
    offset=parity_position-1;
    input_data_counter=1+offset;
    sum=0;
    while(input_data_counter<all_bits) begin
        for (i=0; i<parity_position; i=i+1) begin
            if(input_data_counter < all_bits) begin
                sum=sum+hamming_error[input_data_counter]; //suma odpowiednich bitów danych
                input_data_counter=input_data_counter+1;
            end
        end
        input_data_counter=input_data_counter+parity_position; //bity danych nie branych do oblicznia parity
    end
    if(sum%2!=0) begin
        syndrome=syndrome+parity_position;
    end
    parity_position=parity_position<<1;
end
sum=0;
```

```
sum=0;
for( i=0; i< all_bits; i=i+1) begin
    sum = sum + hamming_error[i];
end

extra_parity=sum%2;

if(extra_parity==0 && syndrome!=0) begin
    status = 2'b10;  //00 -correct; 01 -single error, corrected; 10 -double error, detected
    //$display("Double error detected in %b", hamming_error);
end
else begin
    if(syndrome!=0) begin
        status = 2'b01;
        //$display("Single error detected on position: %0d, in %b", syndrome, hamming_error);
        hamming_error[syndrome] = ~hamming_error[syndrome];
        //$display("Corrected: %b", hamming_error);
        //$display("Data out: %b", data);
    end
    else begin
        status = 2'b00;
        //$display("No error detected in %b", hamming_error);
        //$display("Data out: %b", data);
    end
```

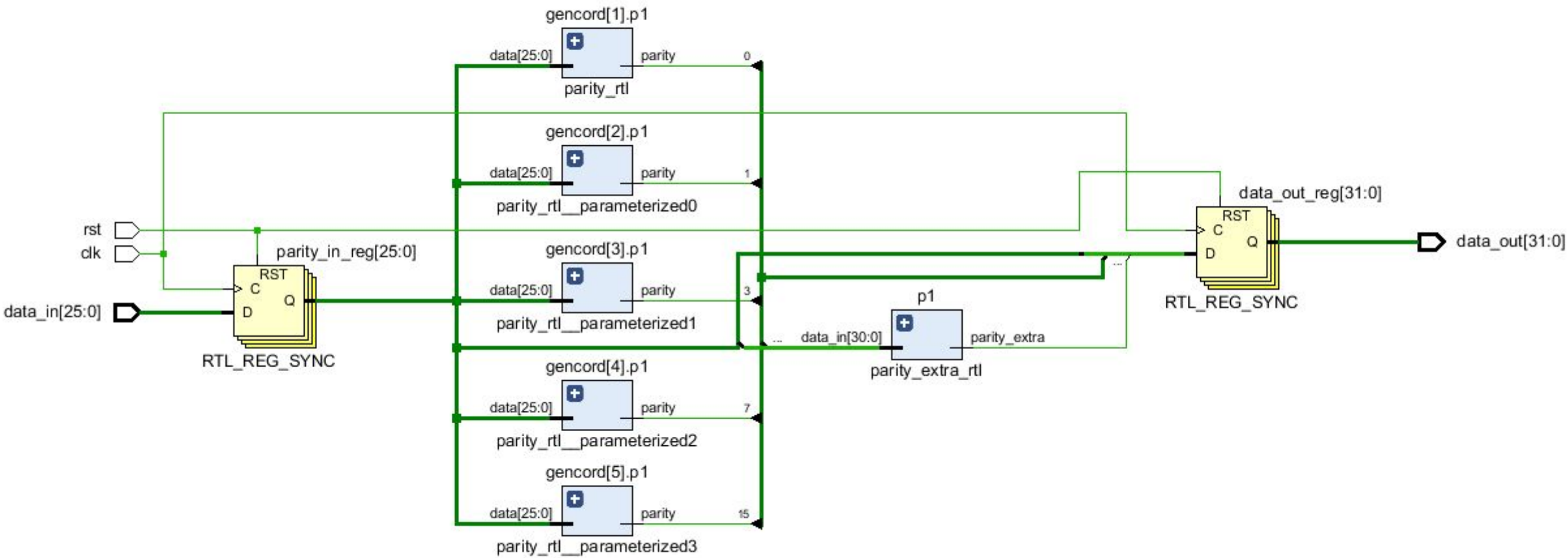
Koder schemat



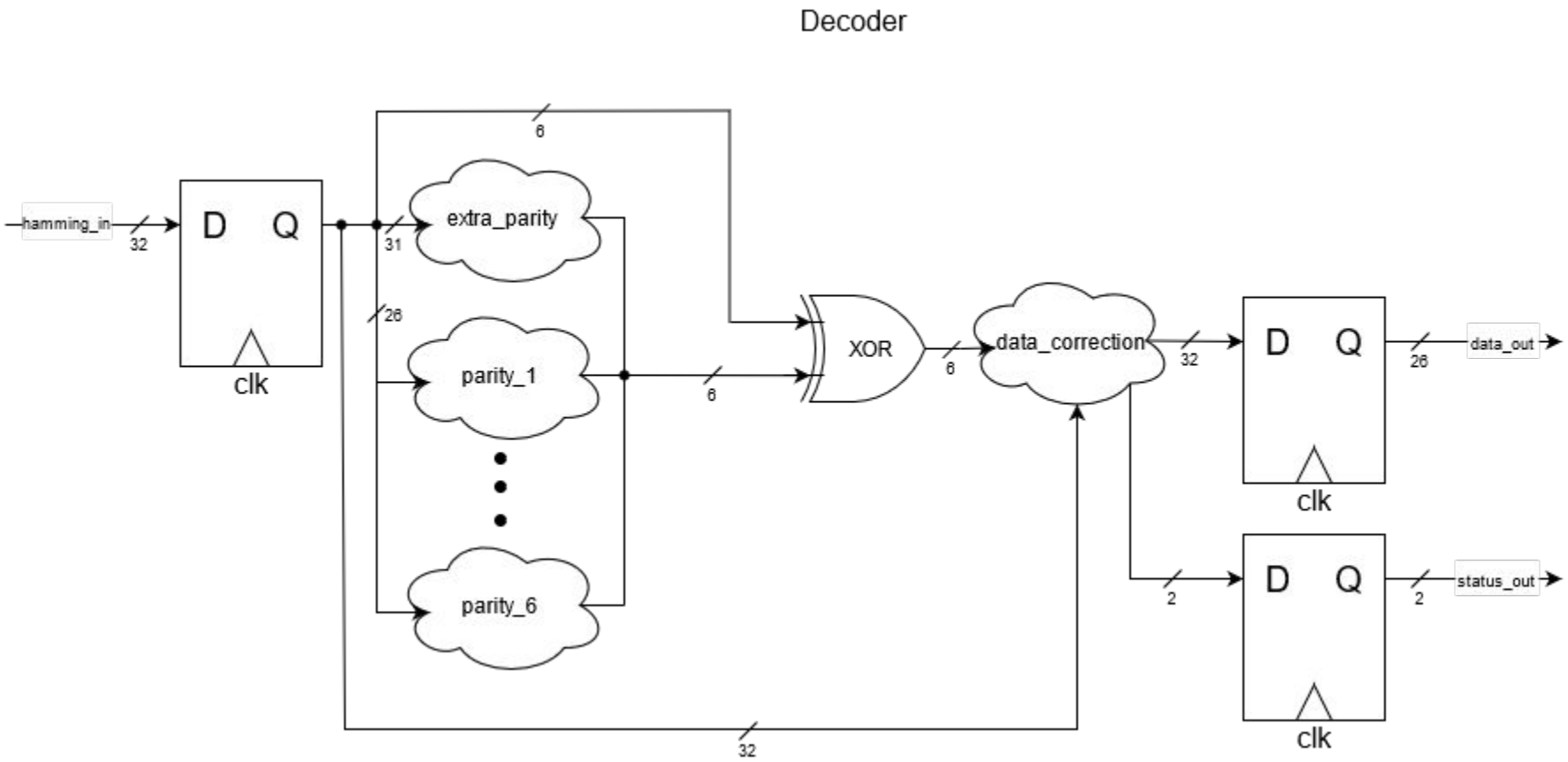
Koder RTL

```
module parity_rtl(
    input [25:0] data,
    output reg parity
    );
    parameter parity_bit=1;

    always@*
        case(parity_bit)
            1: parity=^(data&26'b10101010101010110101011011);
            2: parity=^(data&26'b11001100110011011001101101);
            3: parity=^(data&26'b11110000111100011110001110);
            4: parity=^(data&26'b11111111000000011111110000);
            5: parity=^(data&26'b11111111111111100000000000);
        endcase


endmodule
```

```
genvar j;
generate
    for(j=1; j<6; j=j+1) begin: gencord
        parity_rtl #(.parity_bit(j)) p1 (.data (parity_in), .parity (parity[j-1]));
    end
endgenerate

assign extra_parity_in[1:0]=parity[1:0];
assign extra_parity_in[2]=parity_in[0];
assign extra_parity_in[3]=parity[2];
assign extra_parity_in[6:4]=parity_in[3:1];
assign extra_parity_in[7]=parity[3];
assign extra_parity_in[14:8]=parity_in[10:4];
assign extra_parity_in[15]=parity[4];
assign extra_parity_in[30:16]=parity_in[25:11];
```

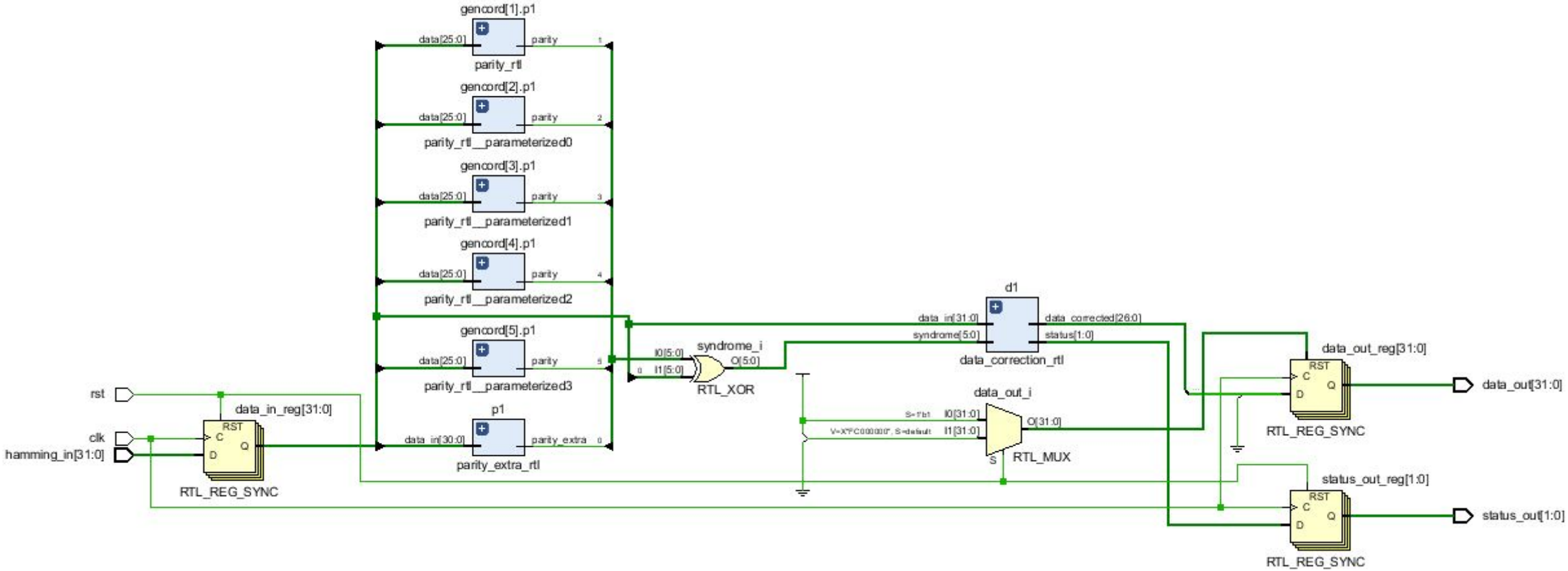

gencord[1].p1
data[25:0]
parity
0
parity_rtl
gencord[2].p1
data[25:0]
parity
1
parity_rtl__parameterized0
gencord[3].p1
data[25:0]
parity
3
parity_rtl__parameterized1
gencord[4].p1
data[25:0]
parity
7
parity_rtl__parameterized2
gencord[5].p1
data[25:0]
parity
15
parity_rtl__parameterized3
rst
clk
data_in[25:0]
parity_in_reg[25:0]
RST
C
D
Q
RTL_REG_SYNC
p1
data_in[30:0]
parity_extra
parity_extra_rtl
data_out_reg[31:0]
RST
C
D
Q
RTL_REG_SYNC
data_out[31:0]

Dekoder schemat



Dekoder RTL

```
module data_correction_rtl(
    input [31:0] data_in,
    input [5:0] syndrome,
    output reg [26:0] data_corrected,
    output reg [1:0] status //00 -correct; 01 -single error, corrected; 10 -double error, detected
    );
  always@* begin
      if(syndrome[5:1]!=5'b00000)
          if(syndrome[0]==0)begin
              status=2'b10;
              temp=data_in;
          end
          else begin
          error_pos = (syndrome[1])+(syndrome[2]<<1)+(syndrome[3]<<2) + (syndrome[4]<<3) + (syndrome[5]<<4);
          temp=data_in;
          temp[error_pos]= ~temp[error_pos];
          status = 2'b01;
          end
      else begin
          status=2'b00;
          temp=data_in;
      end
      data_corrected={temp[31:17],temp[15:9],temp[7:5], temp[3]};
  end
```


gencord[1].p1
data[25:0]
parity
parity_rtl
gencord[2].p1
parity_rtl__parameterized0
gencord[3].p1
parity_rtl__parameterized1
gencord[4].p1
parity_rtl__parameterized2
gencord[5].p1
parity_rtl__parameterized3
p1
data_in[30:0]
parity_extra
parity_extra_rtl
syndrome_i
I0[5:0]
I1[5:0]
O[5:0]
RTL_XOR
d1
data_in[31:0]
syndrome[5:0]
data_corrected[26:0]
status[1:0]
data_correction_rtl
data_out_i
I0[31:0]
I1[31:0]
O[31:0]
S
RTL_MUX
rst
clk
hamming_in[31:0]
data_in_reg[31:0]
RST
C
D
Q
RTL_REG_SYNC
data_out_reg[31:0]
data_out[31:0]
status_out_reg[1:0]
status_out[1:0]

Synteza

Dekoder

Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	50	53200	0.09
FF	60	106400	0.06
IO	68	200	34.00
BUFG	1	32	3.13

Koder

Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	17	53200	0.03
FF	58	106400	0.05
IO	60	200	30.00
BUFG	1	32	3.13

Testbench

```
encoder ebh (clk, to_encode, rst, encoded);
decoder dbh (clk, to_decode, rst, data_out, status);
//encoder_rtl ert1 (.clk(clk), .data_in(to_encode), .rst(rst), .data_out(encoded));
//decoder_rtl drt1 (.clk(clk), .hamming_in(to_decode), .rst(rst), .data_out(data_out), .status_out(status));
clk cl (.clk(clk));

initial begin
    rst=1; #10
    rst=0;
    file_in = $fopen("D:/ham/data.TXT", "r");
    file_out = $fopen("D:/ham/result.TXT", "w");

    while(!$feof(file_in))
        begin
            to_decode=0;
            data = $fscanf(file_in, "%b %b %b\n", data_from_file, expected_data);
            to_encode = data_from_file;
            #20
            if(encoded==expected_data) begin
                $fwrite(file_out, "Test %0d for encoder passed\n", i);
                to_decode=encoded;
                #20
                if(data_out==to_encode && status==2'b00)
                    $fwrite(file_out, "Test %0d for decoder passed, status %b\n", i, status);
                else
                    $fwrite(file_out, "Test %0d for decoder failed, status %b\n", i, status);
                to_decode={encoded[31:11],~encoded[10],encoded[9:0]};
                #20
                if(data_out==to_encode && status==2'b01)
                    $fwrite(file_out, "Test %0d for decoder one error passed, status %b\n", i, status);
                else
                    $fwrite(file_out, "Test %0d for decoder one error failed, status %b\n", i, status);
                to_decode={encoded[31:13],~encoded[12],encoded[11],~encoded[10],encoded[9:0]};
                #20
                $display("%h",data_out);
                if(status==2'b10)
                    $fwrite(file_out, "Test %0d for decoder double error passed, status %b\n", i, status);
                else
                    $fwrite(file_out, "Test %0d for decoder double error failed, status %b\n", i, status);
                end
            else
                $fwrite(file_out, "Test %0d for encoder failed %b, expected %b\n", i, encoded, expected_data);
        i=i+1;
        end

    $fclose(file_in);
    $fclose(file_out);
```

Dane i wyniki dla modułu rtl

data — Notatnik

Plik Edycja Format Widok Pomoc

```
10101011011000101101101001 10101011011000100110110110011000
10101011011110101101101001 10101011011110100110110110001111
10101011011000101100101001 10101011011000100110010010011110
```

result — Notatnik

Plik Edycja Format Widok Pomoc

```
Test 0 for encoder passed
Test 0 for decoder passed, status 00
Test 0 for decoder one error passed, status 01
Test 0 for decoder double error passed, status 10
Test 1 for encoder passed
Test 1 for decoder passed, status 00
Test 1 for decoder one error passed, status 01
Test 1 for decoder double error passed, status 10
Test 2 for encoder passed
Test 2 for decoder passed, status 00
Test 2 for decoder one error passed, status 01
Test 2 for decoder double error passed, status 10
```