به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی برق

پروژه درس طراحی سیستم‌های مبتنی بر Asic Fpga

امتیازی فاز ۴ (Viterbi)

پیاده‌سازی ماژول Viterbi

استاد درس: دکتر مهدی شعبانی

دانشجویان:

علی خدنگی ماهرود – 98101493

سیدسعید جزائری شوشتری – 98104885

دانشگاه صنعتی شریف

دانشکده مهندسی برق

زمستان 1401

مقدمه :

در این فاز یک ماژول ViterbiDecoder پیاده‌سازی کرده‌ایم. و از ماژول transmitter و ماژول receiver که در فاز سوم پیاده‌سازی کرده بودیم استفاده کردیم تا آن را تست کنیم. همچنین با اندکی تغییر از تست بنچ نوشته شده در فاز قبل receiver_tb نیز در این جا برای تست ماژول Viterbi استفاده کردیم. توضیحات مربوط به ماژول‌های receiver و transmitter که در فاز قبلی بود را نیز در اینجا آورده‌ایم.

ماژول ViterbiDecoder :

این ماژول همانند ماژول دیکودر CRC که در فاز سوم پیاده‌سازی کردیم، یک سیگنال start و یک دیتای ۸ بیتی ورودی می‌گیرد. همچنین بر خلاف CRC که فقط کار تشخیص خطا را انجام می‌داد، Viterbi می‌تواند خطا را تصحیح کند. به همین دلیل در این‌جا سیگنال خروجی valid دیگر وجود ندارد و فقط خروجی outputData و ready را برای ماژول قرار داده‌ایم. کلیت کار ماژول به این شکل است که هر state را (۴ state داریم) به صورت یک راس در هر مرحله در نظر می‌گیریم. و با توجه به کودینگ کانولوشنی که در متلب روی دیتا اعمال کرده‌ایم، یال‌های گراف را می‌سازیم و توسط الگوریتم Viterbi سعی می‌کنیم که دیتا را استخراج کنیم. در هر مرحله cost هر راس را حساب کرده و state قبلی که این راس در این مرحله از روی آن آپدیت شده است را در رجیستر lastState قرار می‌دهیم. به این شکل در آخر پس از ۴ کلاک سایکل دیتای خروجی را از روی این مقادیر می‌سازیم.

```
module ViterbiDecoder(
    input clk,
    input start,
    input [7:0] inputData,
    output reg [3:0] outputData,
    output reg ready
    );
    reg [1:0] error [1:0][1:0];
    reg [3:0] cost [3:0];
    reg [1:0] lastState [3:0][3:0];
```

```
    reg[7:0] inputTemp;
    reg[2:0] counter;

    integer i;
    integer j;

    always @(posedge clk) begin
        if(!start) begin
            outputData <= 0;
            inputTemp <= 0;
            counter <= 0;
            ready <= 0;
            for(i = 0; i < 4; i = i + 1)
                for(j = 0; j < 4; j = j + 1)
                    lastState[i][j] <= 0;
            cost[0] <= 0;
            for(i = 1; i < 4; i = i + 1)
                cost[i] <= 4'b111;
        end else begin
            if(counter == 0) begin
                counter <= counter + 1;
                inputTemp <= inputData;
            end else if(counter < 5) begin
                counter <= counter + 1;
                inputTemp <= inputTemp >> 2;
                if((cost[0] + error[0][0]) < (cost[2] + error[1][1])) begin
                    cost[0] <= cost[0] + error[0][0];
                    lastState[0][counter-1] <= 0;
                end else begin
                    cost[0] <= cost[2] + error[1][1];
                    lastState[0][counter-1] <= 2;
                end

                if((cost[0] + error[1][1]) < (cost[2] + error[0][0])) begin
                    cost[1] <= cost[0] + error[1][1];
                    lastState[1][counter-1] <= 0;
                end else begin
                    cost[1] <= cost[2] + error[0][0];
                    lastState[1][counter-1] <= 2;
                end

                if((cost[1] + error[0][1]) < (cost[3] + error[1][0])) begin
                    cost[2] <= cost[1] + error[0][1];
                    lastState[2][counter-1] <= 1;
                end else begin
```

```
                cost[2] <= cost[3] + error[1][0];
                lastState[2][counter-1] <= 3;
            end

            if((cost[1] + error[1][0]) < (cost[3] + error[0][1])) begin
                cost[3] <= cost[1] + error[1][0];
                lastState[3][counter-1] <= 1;
            end else begin
                cost[3] <= cost[3] + error[0][1];
                lastState[3][counter-1] <= 3;
            end

        end else begin
            ready <= 1;
            if(cost[0] < cost[1]) begin
                if(cost[0] < cost[2]) begin
                    if(cost[0] < cost[3]) begin
                        outputData <= {
                            lastState[lastState[lastState[0][3]][2]][1][0],
                            lastState[lastState[0][3]][2][0],
                            lastState[0][3][0],
                            1'b0};
                    end else begin
                        outputData <= {
                            lastState[lastState[lastState[3][3]][2]][1][0],
                            lastState[lastState[3][3]][2][0],
                            lastState[3][3][0],
                            1'b1};
                    end
                end else begin
                    if(cost[2] < cost[3]) begin
                        outputData <= {
                            lastState[lastState[lastState[2][3]][2]][1][0],
                            lastState[lastState[2][3]][2][0],
                            lastState[2][3][0],
                            1'b0};
                    end else begin
                        outputData <= {
                            lastState[lastState[lastState[3][3]][2]][1][0],
                            lastState[lastState[3][3]][2][0],
                            lastState[3][3][0],
                            1'b1};
                    end
                end
            end else begin
```

```
                    if(cost[1] < cost[2]) begin
                        if(cost[1] < cost[3]) begin
                            outputData <= {
                                lastState[lastState[lastState[1][3]][2]][1][0],
                                lastState[lastState[1][3]][2][0],
                                lastState[1][3][0],
                                1'b1};
                        end else begin
                            outputData <= {
                                lastState[lastState[lastState[3][3]][2]][1][0],
                                lastState[lastState[3][3]][2][0],
                                lastState[3][3][0],
                                1'b1};
                        end
                    end else begin
                        if(cost[2] < cost[3]) begin
                            outputData <= {
                                lastState[lastState[lastState[2][3]][2]][1][0],
                                lastState[lastState[2][3]][2][0],
                                lastState[2][3][0],
                                1'b0};
                        end else begin
                            outputData <= {
                                lastState[lastState[lastState[3][3]][2]][1][0],
                                lastState[lastState[3][3]][2][0],
                                lastState[3][3][0],
                                1'b1};
                        end
                    end
                end
            end
        end
    end

    always @(*) begin
        error[0][0] = {0, inputTemp[1]} + {0, inputTemp[0]};
        error[1][1] = {0, ~inputTemp[1]} + {0, ~inputTemp[0]};
        error[0][1] = {0, inputTemp[1]} + {0, ~inputTemp[0]};
        error[1][0] = {0, ~inputTemp[1]} + {0, inputTemp[0]};
    end

endmodule
```

ماژول transmitter :

این ماژول را برای تست این فاز پیاده‌سازی کرده‌ایم. متغیر END_OF_COUNTER تعداد کلاک لازم برای دریافت دیتا را مشخص می‌کند برای مثال برای این که با baud rate 115200 دیتا را دریافت کنیم و کلاک 50M داشته باشیم باید این پارامتر را 434 قرار دهیم. در این فاز برای تست این پارامتر را روی ۱۰ قرار می‌دهیم. دیتای ورودی روی سیگنال data که ۸ بیتی است قرار می‌گیرد و سیگنال start نیز برای شروع کار باید یک شده و یک بماند. پس از اتمام کار سیگنال done یک می‌شود. همچنین دیتا بر روی سیگنال TX فرستاده می‌شود.

```
module transmitter(
    input clk,
    input start,
    input [7:0] data,
    output reg TX,
    output reg done
    );
    parameter END_OF_COUNTER = 10;

    reg [3:0] i;
    wire [9:0] temp = {1'b1 , data , 1'b0};
    reg [7:0] counter; //determines Transmitting Rate ( in this code : 115200)
    reg running;

    always @(posedge clk)begin
        if(!start) begin
            counter <= 0;
            TX <= 1;
            i <= 0;
            done <= 0;
            running <= 0;
        end else if(!done) begin
            running <= 1;
        end

        if(running) begin
            counter <= counter + 1;
            if(counter == END_OF_COUNTER) begin
                i <= i + 1 ;
                TX <= temp [i];
                counter <= 0;
```

```
                if(i == 9)begin
                    i <= 0;
                    done <= 1;
                    running <= 0;
                end
            end
        end
    end

endmodule
```

ماژول receiver :

این ماژول اصلی این فاز است. این ماژول یک سیگنال start دارد برای شروع به کار این ماژول باید این سیگنال را یک کنیم. همچنین ورودی این ماژول از سیگنال rx می‌آید. همان پارامتر END_OF_COUTNER در این ماژول نیز وجود دارد و همان موارد ماژول transmitter در اینجا هم صدق می‌کند. یک instance از ماژول CRC نیز در این ماژول گرفته شده است و سیگنال start این ماژول را با crcStart مشخص کرده‌ایم. پس از دریافت کامل پکت، این سیگنال را یک می‌کنیم. سپس چند کلاک طول می‌کشد که ماژول CRC دیتای خروجی را حساب کند. سیگنال‌های ready، valid و outputData از ماژول CRC در خروجی ماژول receiver قرار گرفته اند.

```
module receiver(
    input clk,
    input start,
    input rx,
    output ready,
    output valid,
    output[3:0] outputData
    );
    parameter END_OF_COUNTER = 10;

    reg state;
    reg [8:0] data;
    reg [7:0] counter;
    reg [3:0] i;

    reg crcStart;
```

```
    always@(posedge clk) begin
        if(!start) begin
            data <= 0;
            counter <= 0;
            i <= 0;
            state <= 0;
            crcStart <= 0;
        end else begin
            if(!state) begin
                if(rx == 0)
                    state <= 1;
            end if(state) begin
                counter <= counter + 1;
                if(counter == END_OF_COUNTER) begin
                    i <= i + 1;
                    data[i] <= rx;
                    counter <= 0;
                    if(i == 8) begin
                        state <= 0;
                        i <= 0;
                        crcStart <= 1;
                    end else
                        crcStart <= 0;
                end
            end
        end
    end

    CRC crc(
        .clk(clk),
        .start(crcStart),
        .InputData(data[7:0]),
        .Ready(ready),
        .valid(valid),
        .OutputData(outputData)
    );

endmodule
```

شبیه‌سازی:

برای شبیه‌سازی ماژول receiver_tb که در فاز قبل پیاده‌سازی کرده‌بودیم را اندکی تغییر دادیم تا به صورت خودکار خطای پیاده‌سازی را از روی مقاسیه raw_data و دیتای خروجی بدست آمده، پیدا کند.

```
`define N 100

module receiver_tb;

    reg clk = 0;
    reg startT;
    reg startR;
    reg[7:0] inputData;

    wire doneT;


    wire data_wire;
    wire readyR;
    wire[3:0] outputData;

    always@(clk)
        clk <= #5 ~clk;

    integer numberOfErros = 0;
    integer i;
    integer file;
    reg[`N-1:0] raw_data;
    reg[2*`N-1:0] encoded_data;

    initial begin
        file = $fopen("raw_data.mem", "r");
        $fscanf(file, "%b\n", raw_data);
        $fclose(file);

        file = $fopen("encoded_data.mem", "r");
        $fscanf(file, "%b\n", encoded_data);
        $fclose(file);
    end

    initial begin
        startT = 0;
        startR = 0;
```

```
        inputData = 0;
        @(posedge clk);


        for(i = 0; i < 2 * `N - 8; i = i + 8) begin
            startT = #2 1;
            startR = 1;
            inputData[7] = encoded_data[i+7];
            inputData[6] = encoded_data[i+6];
            inputData[5] = encoded_data[i+5];
            inputData[4] = encoded_data[i+4];
            inputData[3] = encoded_data[i+3];
            inputData[2] = encoded_data[i+2];
            inputData[1] = encoded_data[i+1];
            inputData[0] = encoded_data[i];
            @(posedge doneT);

            @(posedge readyR);
            startT = #2 0;
            $write("output data = %b\n", outputData);
            if(raw_data[3:0] !== outputData) begin
                numberOfErros = numberOfErros + 1;
                $write("ERROR: expected %b but the output is %b\n", raw_data[3:0],
outputData);
            end
            raw_data = raw_data >> 4;

            @(posedge clk);
        end

        if(numberOfErros > 0)
            $write("Number of erros = %d\n", numberOfErros);
        else
            $write("All tests passed\n");

        #20;
        $finish;
    end

    receiver receiverModule(
        .clk(clk),
        .start(startR),
        .rx(data_wire),
        .ready(readyR),
        .outputData(outputData)
```

```
    );

    transmitter transmitterModule(
        .clk(clk),
        .start(startT),
        .data(inputData),
        .TX(data_wire),
        .done(doneT)
    );

endmodule
```

متلب :

در متلب یک دیتای ۱۰۰ بیتی را به صورت تصادفی ایجاد کرده‌ایم و Viterbi را برای هر دیتای ۴ تایی حساب می‌کنیم و به هم می‌چسبانیم. سپس این دو دیتا را در فایل می‌نویسیم.

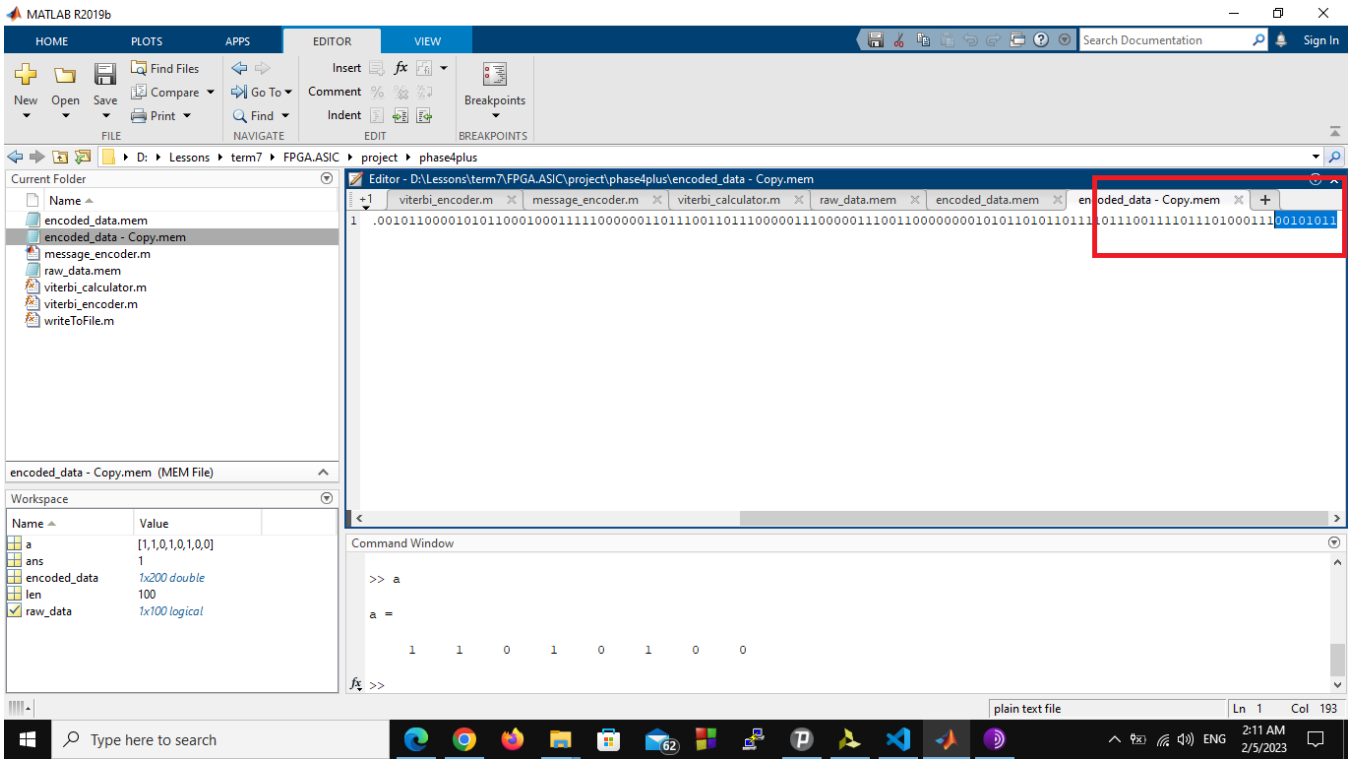
```
%% FPGA project phase 4+ viterbi
clear;
clc;

len = 100;

raw_data = rand(1, len) > 0.5;
writeToFile(raw_data, 'raw_data.mem');

encoded_data = viterbi_encoder(raw_data);
writeToFile(encoded_data, 'encoded_data.mem');
```

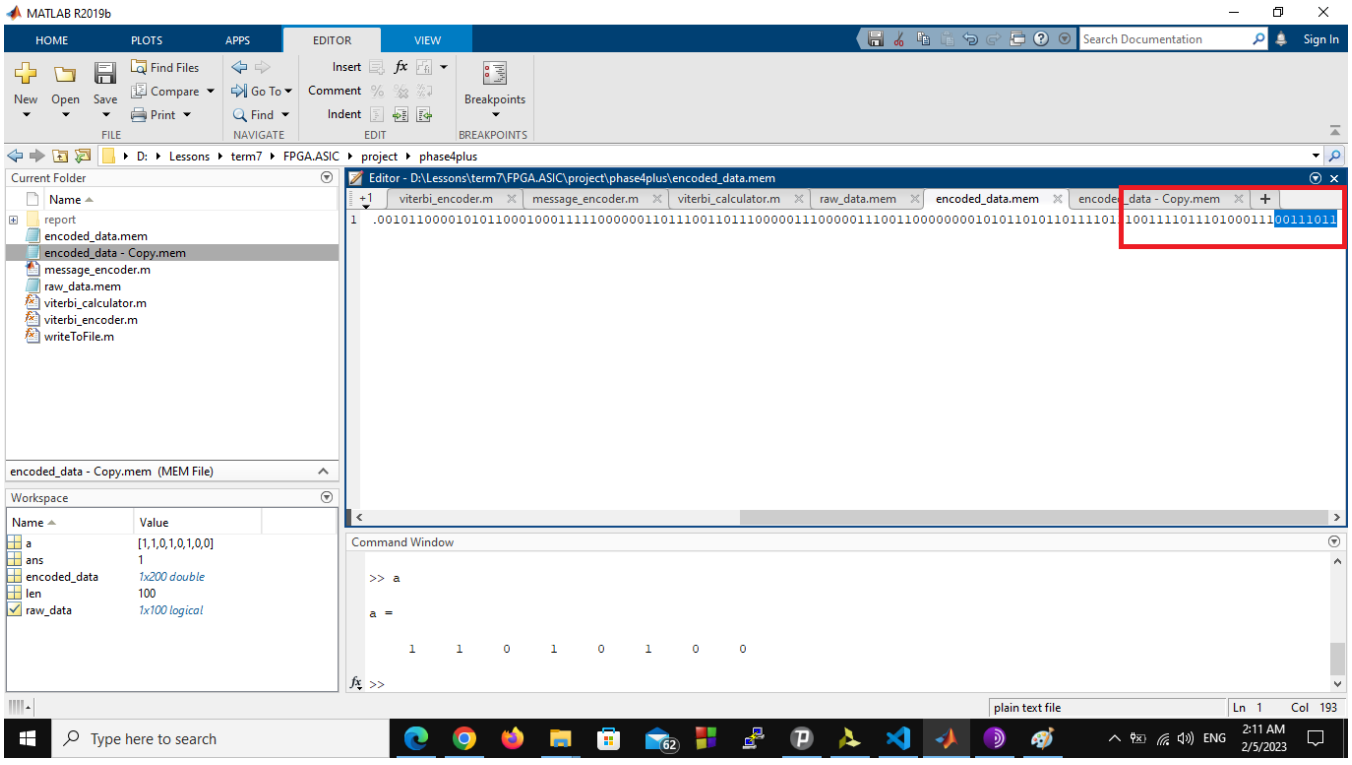
```
function viterbi = viterbi_calculator(packet)
    viterbi = zeros(1, 8);
    s0 = 0;
    s1 = 0;
    for i = 1:4
        bit = xor(packet(i), s1);
        viterbi(2*i-1:2*i) = [xor(bit, s0), bit];
        s1 = s0;
        s0 = packet(i);
    end
end
```

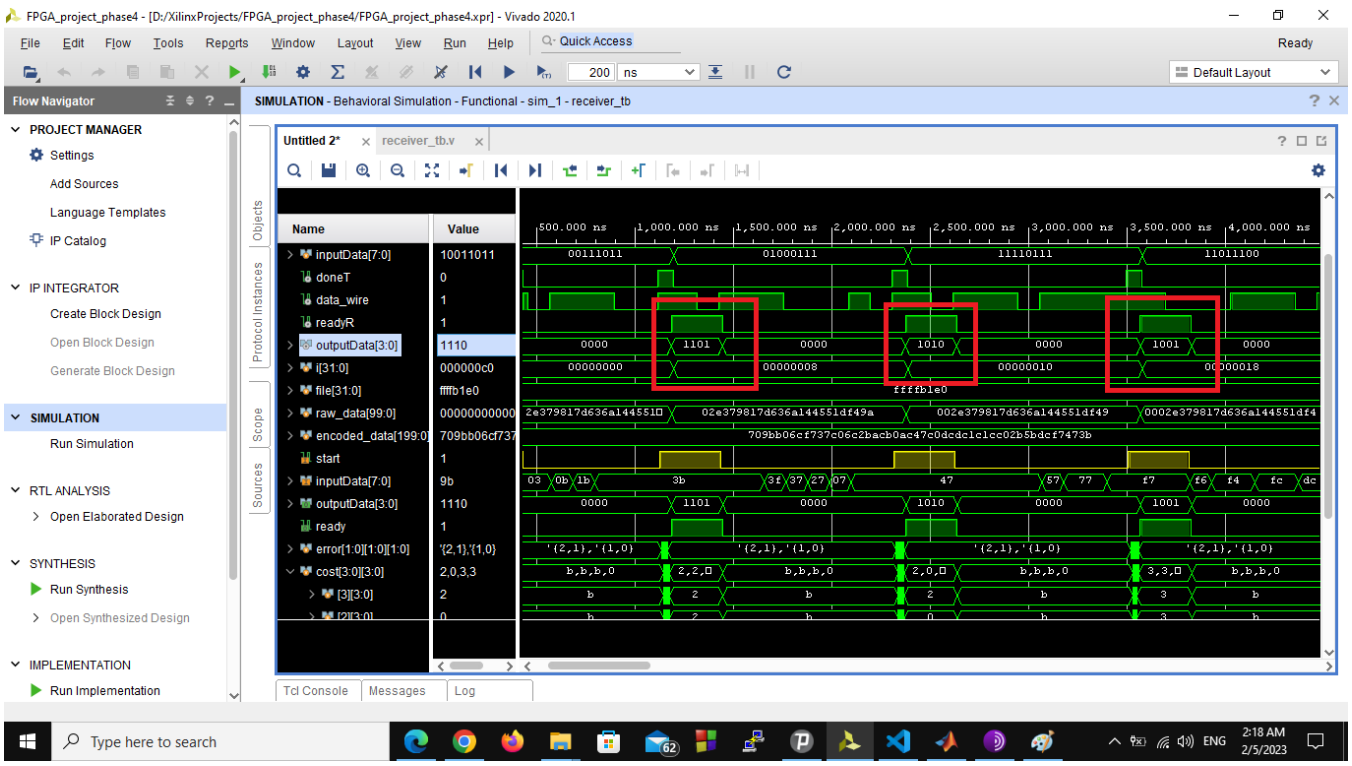
```
function writeToFile(raw_data, file_name)
    x_bin = char(double(raw_data) + 48);
    %x_bin = upsample(x_bin, 2);
    %x_bin(x_bin == 0) = newline;
    bin_data = x_bin;
    file = fopen(file_name, 'wt');
    fwrite(file, bin_data);
    fclose(file);
end
```

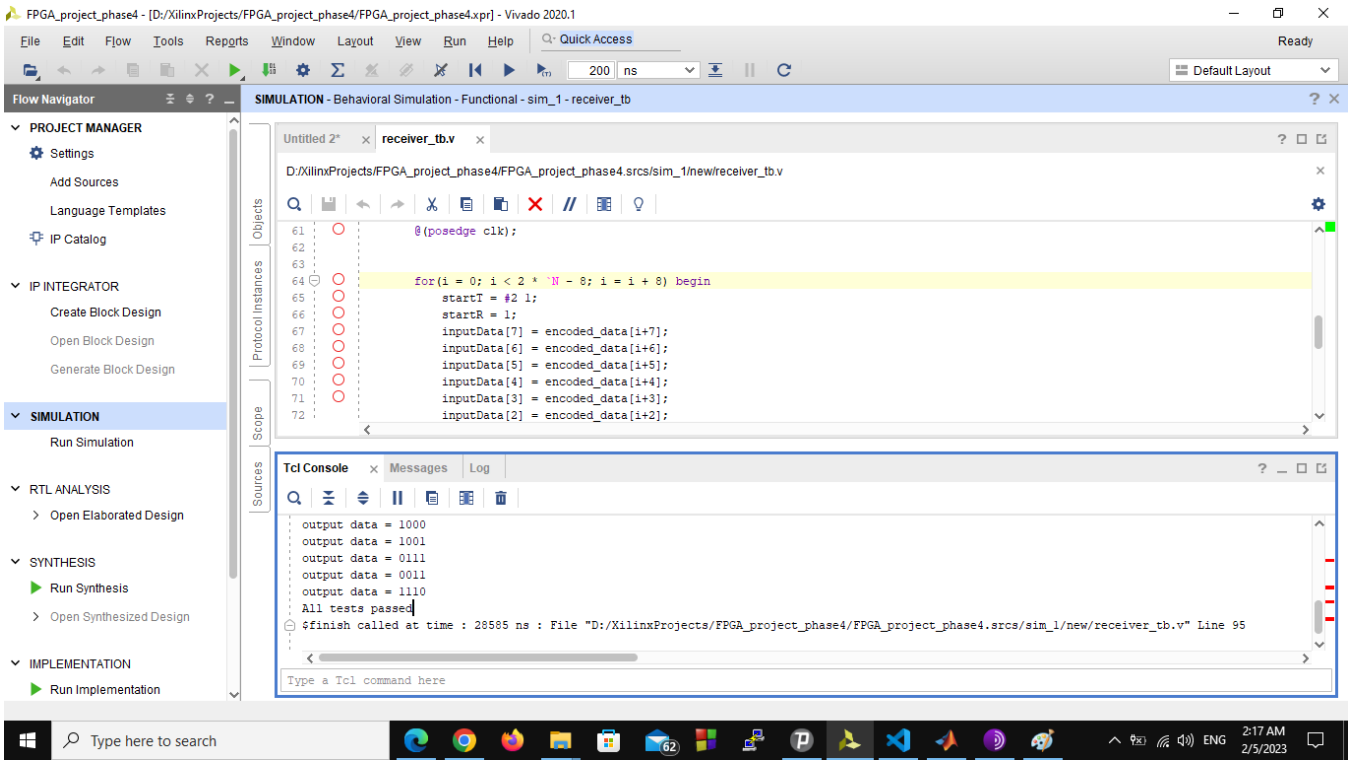
```
function encoded_data = viterbi_encoder(raw_data)
    raw_data = double(raw_data);
    len = length(raw_data);

    encoded_data = zeros(1, 2*len);
    for i = 1:len/4
       packet = raw_data(4*i-3:4*i);
       encoded_data(8*i-7:8*i) = fliplr(viterbi_calculator(packet));
    end
end
```

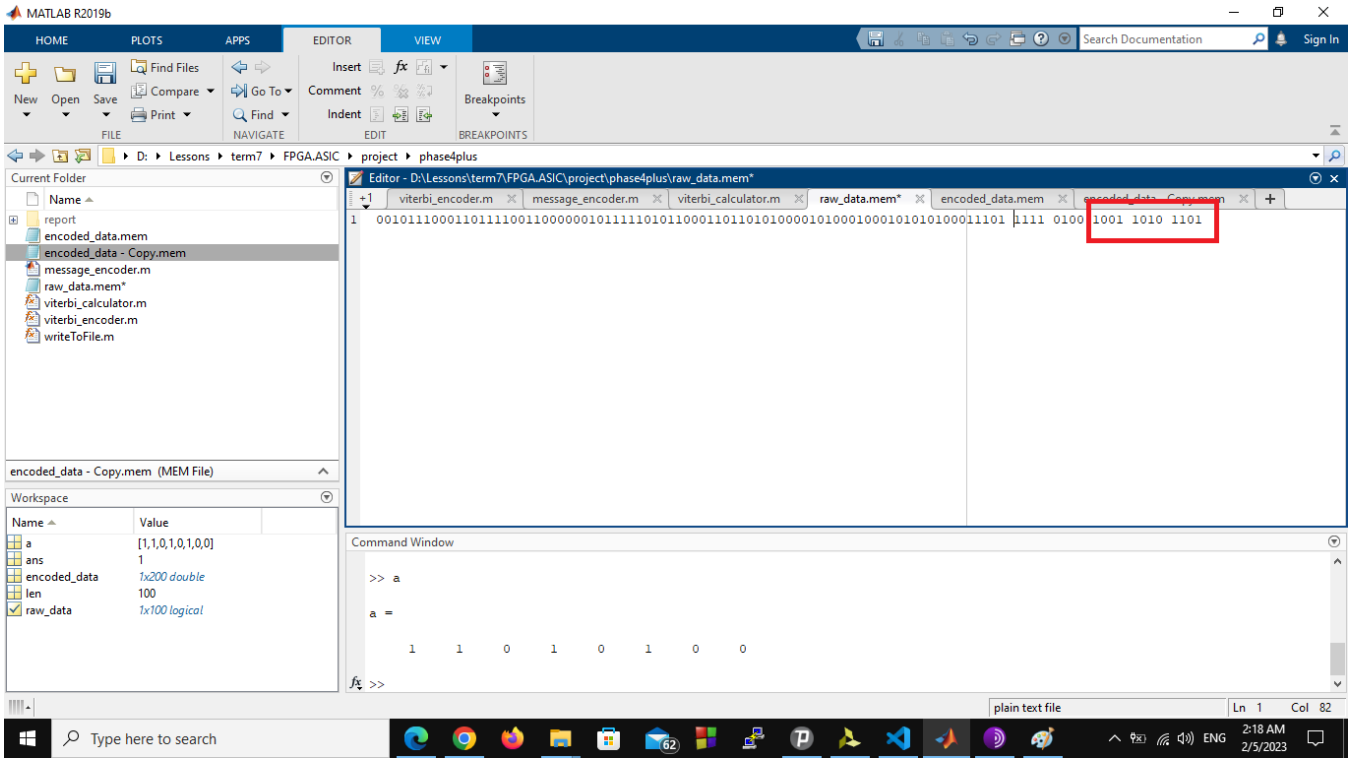
تصویر شبیه‌سازی را در شکل زیر مشاهده می‌کنیم. همان‌طور که می‌بینید در فایل encoded_data.mem مقدار پکت اول را تغییر داده‌ام تا ببینیم که آیا ماژول می‌تواند خطا را تصحیح کند یا خیر. سپس همانطور که در تصویر سوم می‌بینیم، خروجی درست بوده است و پکت اول درست شناسایی شده است و خطای آن تشخیص داده شده و تصحیح شده است. تصویر چهارم خروجی چاپ شده را نشان می‌دهد که همه تست‌ها پاس شده‌اند. تصویر پنجم دیتا پیش از code شدن توسط Viterbi encoder را نشان می‌دهد. (raw_data)


MATLAB R2019b
HOME
PLOTS
APPS
EDITOR
VIEW
Search Documentation
Sign In
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
FILE
NAVIGATE
EDIT
BREAKPOINTS
D: ▸ Lessons ▸ term7 ▸ FPGA.ASIC ▸ project ▸ phase4plus
Current Folder
Name
encoded_data.mem
encoded_data - Copy.mem
message_encoder.m
raw_data.mem
viterbi_calculator.m
viterbi_encoder.m
writeToFile.m
Editor - D:\Lessons\term7\FPGA.ASIC\project\phase4plus\encoded_data - Copy.mem
viterbi_encoder.m
message_encoder.m
viterbi_calculator.m
raw_data.mem
encoded_data.mem
encoded_data - Copy.mem
1 .00101100001010110001000111110000001101110011011100000111000001110011000000001010110101101111011100111101110100011100101011
encoded_data - Copy.mem (MEM File)
Workspace
Name
Value
a
[1,1,0,1,0,1,0,0]
ans
1
encoded_data
1x200 double
len
100
raw_data
1x100 logical
Command Window
>> a
a =
1 1 0 1 0 1 0 0
>>
plain text file
Ln 1
Col 193
Type here to search
ENG
2:11 AM
2/5/2023


MATLAB R2019b
HOME
PLOTS
APPS
EDITOR
VIEW
Search Documentation
Sign In
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
FILE
NAVIGATE
EDIT
BREAKPOINTS
D: ▸ Lessons ▸ term7 ▸ FPGA.ASIC ▸ project ▸ phase4plus
Current Folder
Name
report
encoded_data.mem
encoded_data - Copy.mem
message_encoder.m
raw_data.mem
viterbi_calculator.m
viterbi_encoder.m
writeToFile.m
Editor - D:\Lessons\term7\FPGA.ASIC\project\phase4plus\encoded_data.mem
viterbi_encoder.m
message_encoder.m
viterbi_calculator.m
raw_data.mem
encoded_data.mem
encoded_data - Copy.mem
1 .00101100001010110001000111110000001101110011011100000111000001110011000000001010110101101111011100111101110100011100111011
encoded_data - Copy.mem (MEM File)
Workspace
Name
Value
a
[1,1,0,1,0,1,0,0]
ans
1
encoded_data
1x200 double
len
100
raw_data
1x100 logical
Command Window
>> a
a =
1 1 0 1 0 1 0 0
>>
plain text file
Ln 1
Col 193
Type here to search
ENG
2:11 AM
2/5/2023


FPGA_project_phase4 - [D:/XilinxProjects/FPGA_project_phase4/FPGA_project_phase4.xpr] - Vivado 2020.1
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
SIMULATION - Behavioral Simulation - Functional - sim_1 - receiver_tb
Untitled 2*
receiver_tb.v
Name
Value
inputData[7:0]
doneT
data_wire
readyR
outputData[3:0]
i[31:0]
file[31:0]
raw_data[99:0]
encoded_data[199:0]
start
inputData[7:0]
outputData[3:0]
ready
error[1:0][1:0][1:0]
cost[3:0][3:0]
Tcl Console
Messages
Log
Type here to search
ENG
2:18 AM
2/5/2023


FPGA_project_phase4 - [D:/XilinxProjects/FPGA_project_phase4/FPGA_project_phase4.xpr] - Vivado 2020.1
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
SIMULATION - Behavioral Simulation - Functional - sim_1 - receiver_tb
Untitled 2*
receiver_tb.v
D:/XilinxProjects/FPGA_project_phase4/FPGA_project_phase4.srcs/sim_1/new/receiver_tb.v
@(posedge clk);
for(i = 0; i < 2 * `N - 8; i = i + 8) begin
startT = #2 1;
startR = 1;
inputData[7] = encoded_data[i+7];
inputData[6] = encoded_data[i+6];
inputData[5] = encoded_data[i+5];
inputData[4] = encoded_data[i+4];
inputData[3] = encoded_data[i+3];
inputData[2] = encoded_data[i+2];
Tcl Console
Messages
Log
output data = 1000
output data = 1001
output data = 0111
output data = 0011
output data = 1110
All tests passed
$finish called at time : 28585 ns : File "D:/XilinxProjects/FPGA_project_phase4/FPGA_project_phase4.srcs/sim_1/new/receiver_tb.v" Line 95
Type a Tcl command here
Type here to search
ENG
2:17 AM
2/5/2023


MATLAB R2019b
HOME
PLOTS
APPS
EDITOR
VIEW
Search Documentation
Sign In
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
FILE
NAVIGATE
EDIT
BREAKPOINTS
D: ▸ Lessons ▸ term7 ▸ FPGA.ASIC ▸ project ▸ phase4plus
Current Folder
Name
report
encoded_data.mem
encoded_data - Copy.mem
message_encoder.m
raw_data.mem*
viterbi_calculator.m
viterbi_encoder.m
writeToFile.m
encoded_data - Copy.mem (MEM File)
Workspace
Name
Value
a
[1,1,0,1,0,1,0,0]
ans
1
encoded_data
1x200 double
len
100
raw_data
1x100 logical
Editor - D:\Lessons\term7\FPGA.ASIC\project\phase4plus\raw_data.mem*
viterbi_encoder.m
message_encoder.m
viterbi_calculator.m
raw_data.mem*
encoded_data.mem
1 0010111000110111100110000001011111010110001101101010000101000100010101010001 1101 1111 0100 1001 1010 1101
Command Window
>> a
a =
1 1 0 1 0 1 0 0
>>
plain text file
Ln 1 Col 82
Type here to search
ENG
2:18 AM
2/5/2023