به نام خدا

دانشکده مهندسی برق

گزارش فاز 4 پروژه

درس سیستم های مبتنی بر FPGA/ASIC

محمدامین حاجی خداوردیان

97101518

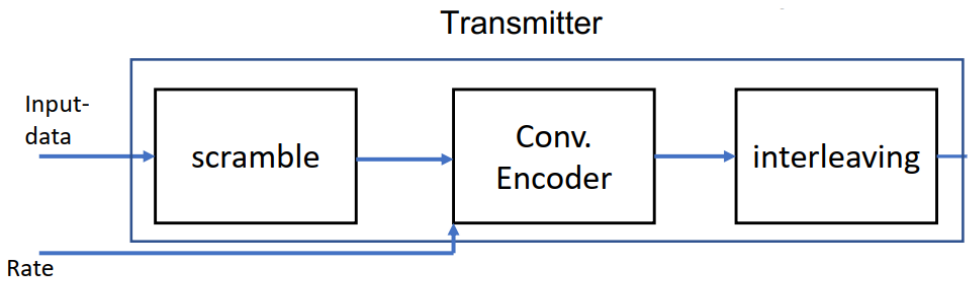
استاد:دکتر مهدی شعبانی

نیمسال دوم 99-00

در این فاز بخش هایی که در فاز های قبل پیاده سازی کرده ایم را درکنار هم قرار میدهیم تا به یک پیاده سازی از فرستنده و گیرنده استاندارد 802.11a برسیم.

مقدمه:

بخش فرستنده:



بخش فرستنده همانطور که در شکل بالا دیده میشود از سه بخش scramble ، Convolutional Encoder و بخش Interleave تشکیل شده است.

بخش های مختلف هر کدام در فاز های قبلی توضیحات کامل راجع به جزئیات آن ها گفته شده است که گزارش های مربوط به آن ها در فایل ارسال شده قرار دارد.(برای بخش Scramble گزارش با نام Phase1 ، برای بخش Encoder گزارش با نام Phase2 و برای بخش Interleave گزارش با نام Phase3 در فایل ارسالی موجود است که میتوان به آن ها مراجعه کرد).

در ابتدای داده ای که قرار است فرستاده شود با یک seed رندوم scramble میشود تا از خطاهای احتمالی تصادفی جلوگیری کند در ادامه توسط یک Encoder به صورت مشخصی کدگذاری میشوند. این بخش برای این اهمیت دارد که الگوریتمی که در گیرنده استفاده میشود(Viterbi) با استفاده از این کدگذاری میتواند بیت های خطا را پیدا و تصحیح کند و با احتمال خوبی خطا های تصادفی ایجاد شده را برطرف نماید.

در انتها برای قرار دادن بیت ها در Sub carrier های متفاوت داده ها Interleave میشوند.

حال به سراغ پیاده سازی فرستنده در متلب و وریلاگ میپردازیم.

پیاده سازی متلب:

```
clear;
clc;
%%%Data Frame & Scramble
f = fopen('In.txt' , 'w+');
o = fopen('Out_Transmitter.txt' , 'w+');
x = fopen('N_pad.txt' , 'w+');
rate = dec2bin(13 , 4);
rate_param = SET_RATE_PARAMS(6);%Calculate different parameters of Data rate
                                % of 6Mbit/s (NBPSC ,NCBPS ,NDBPS)
length_n = dec2bin(5 , 12); %Make a PSDU with the length of 8*5 bits
%Calculate N_pad
N_SYM = ceil((16 + 8*5+6)/rate_param.NDBPS);
N_DATA = N_SYM * rate_param.NDBPS;
N_PAD = N_DATA - (16 + 8 * 5 + 6);
n_pad = dec2bin(N_PAD , 6);

fprintf(x ,'%c' , n_pad(:));%Save N_pad for verilog simulation
scramble_state = [1 ,0 ,1 ,1 ,0 ,1 ,1 ];
psdu = floor(2.*rand(1 ,bin2dec([length_n,'000'])));
signal_preamble = ['111111111111'  , rate ,'0' ,length_n ,'0' ,'000000'];

for i=1:length(signal_preamble)
    fprintf(f ,'%c \n' , signal_preamble(i));
    fprintf(o ,'%c \n' , signal_preamble(i));
end
```

در ابتدا برای تست کردن پیاده سازی وریلاگ ورودی های تولید شده و خروجی های شبیه سازی را در دو فایل مختلف با نام های In.txt و Out_Transmitter.txt ذخیره میکنیم.

سپس با به صورت دلخواه شروع به ساخت یک فریم داده میکنیم.(پیاده سازی ها مبتنی بر مدولاسیون BPSK است)

تابع SET_RATE_PARAMS با توجه به Datarate داده شده به آن مقادیر متناسب مدنظر ما را تولید میکند که از جدول صفحه بعد به دست می آید.

سپس با استفاده از رابطه زیر تعداد Pad bit را حساب میکنیم.

$$N_{SYM} = \text{Ceiling}\left((16 + 8 \times \text{LENGTH} + 6)/N_{DBPS}\right)$$

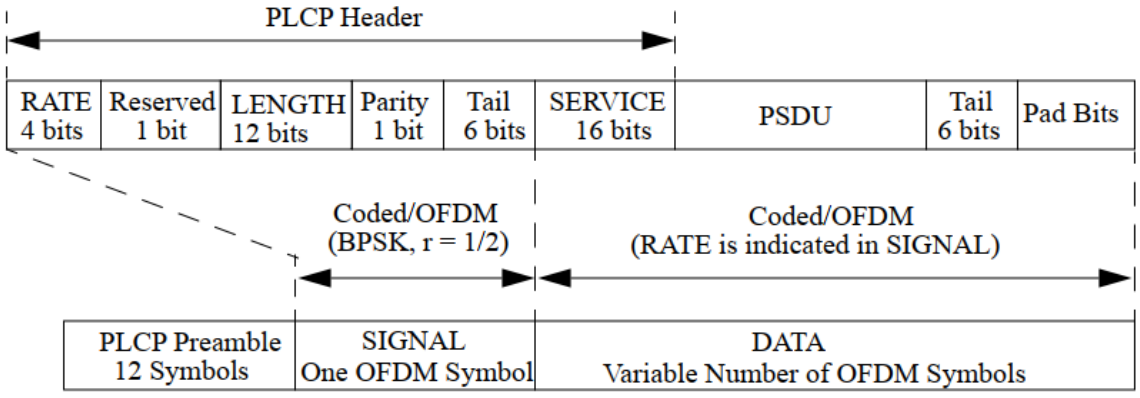
$$N_{DATA} = N_{SYM} \times N_{DBPS}$$

$$N_{PAD} = N_{DATA} - (16 + 8 \times \text{LENGTH} + 6)$$

Data rate (Mbits/s)	Modulation	Coding rate (R)	Coded bits per subcarrier (N_{BPSC})	Coded bits per OFDM symbol (N_{CBPS})	Data bits per OFDM symbol (N_{DBPS})
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

سپس با توجه به طول در نظر گرفته شده شروع به ساخت تصادفی بخش داده اصلی که قرار است فرستاده شود میکنیم.(با هر بار اجرای کد فرستنده یک داده جدید ساخته میشود بنابراین در صورت اجرای آن فایل های خروجی آن را که برای بخش گیرنده و تست کردن پیاده سازی وریلاگ استفاده میکنیم را باید با فایل های جدید جایگزین کنیم)

در ادامه نیز بخش SIGNAL فریم ارسالی را میسازیم که مطابق شکل زیر استاندارد است:



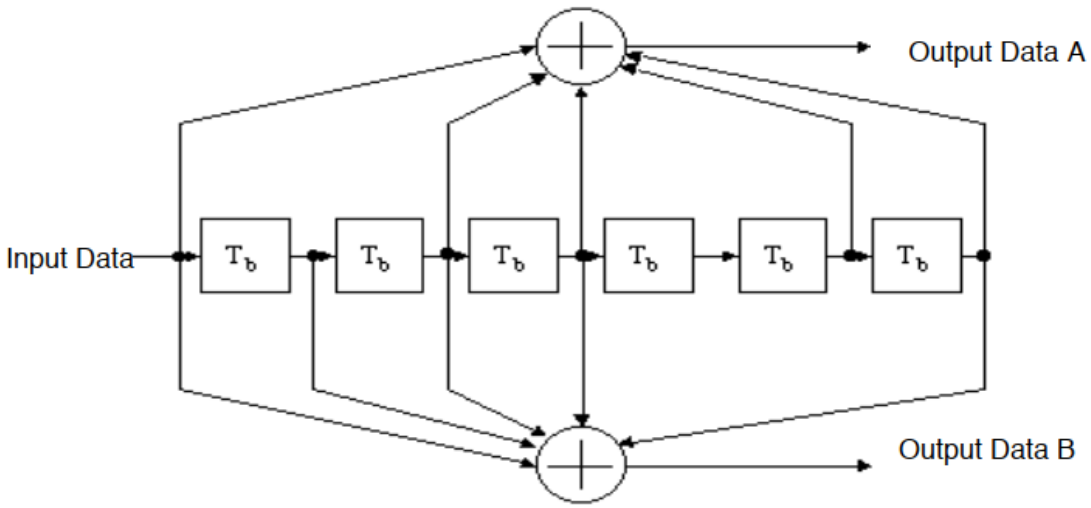
```
26 -    data = [zeros(1,16) ,psdu ,zeros(1,6) ,  zeros(1 , N_PAD)];%DATA Frame that should be scrambled
27 -    dataout = [zeros(1,16) ,psdu ,zeros(1,6) , zeros(1 , N_PAD)];%Scrambled DATA Frame that should send to encoder
28      %Scramble
29 -    for i=1:length(data)
30 -        fprintf(f ,'%d \n' , data(i));
31 -        scramble_data = xor(scramble_state(1),scramble_state(4));
32 -        scramble_state = [scramble_state(2:7) ,scramble_data];
33 -        dataout(i) = xor(data(i) ,scramble_data);
34 -    end
35 -    fclose(f);
36 -    fclose(x);
```

در بخش بالا به Scramble کردن بخش DATA میپردازیم و سپس فایل های مربوط به تولید ورودی را میبندیم چون در ادامه به آن ها نیازی نیست.

```
38      %Encoder
39 -    rate = 0;
40 -    if(rate == 0)%%0: rate = 1/2
41 -    encode_state = [0 , 0 , 0 , 0 , 0 , 0 , 0];
42 -    encode_data = zeros(2*length(dataout) , 1);
43 -    for i=1:length(dataout)
44 -          encode_data(2*i-1) = mod(encode_state(2)+ encode_state(3)+ encode_state(5)+encode_state(6) + dataout(i) , 2);
45 -          encode_data(2*i) = mod(encode_state(1)+encode_state(2) + encode_state(3) + encode_state(6) + dataout(i) , 2);
46 -           encode_state(2:6) = encode_state(1:5);
47 -           encode_state(1) = dataout(i);
48 -    end
49 -    end
```

پیاده سازی انجام شده برای یک Coding Rate = ½ انجام شده است.کد بالا با توجه به Encoder که در استاندارد به آن اشاره شده است پیاده سازی شده است که شکل آن را در زیر میبینیم.

داده های موجود که در بخش قبل Scramble شده اند وارد Encoder میشود و خروجی آن آماده فرستاده شدن به آخرین بخش یعنی interleaving میشود.



```
%Interleaver
N_CBPS = rate_param.NCBPS; %As we said in Phase3 these two parts came from Datarate of dataframe
N_BPSC = rate_param.NBPSC;

if(N_BPSC/2 < 1)
    S = 1;
else
    S = N_BPSC/2;
end
step = length(encode_data)/N_CBPS;
for i=1:step
    Interleave_stage1 = zeros(N_CBPS , 1);
    for j=0:N_CBPS-1
        index =((N_CBPS/16)*(mod(j , 16)) + floor(j/16)+1);
        Interleave_stage1(S*floor(index/S) + mod((index + N_CBPS - floor(16*index/N_CBPS)) , S)) = encode_data(j+1 + (i-
        Interleave(S*floor(index/S) + mod((index + N_CBPS - floor(16*index/N_CBPS)) , S) + (i-1)*N_CBPS) = encode_data(j
        disp([encode_data(j+1 + (i-1)*48), S*floor(index/S) + mod((index + N_CBPS - floor(16*index/N_CBPS)) , S)-1] );
    end
    fprintf(o , "%d\n" , Interleave_stage1);
end

fclose(o);
```

حال در بالا همانطور که در فاز3 اشاره شد مقادیر N_{CBPS} و N_{BPSC} که برای تست کردن درستی کد به یک عدد دلخواه قرار داده شده بودند در اینجا با توجه به ورودی فریم تعیین میشوند.

سپس دو مرحله ای که برای جا به جایی بیت های ورودی نیاز است انجام میشود و بخش DATA فریم نیز آماده میشود تا در کنار بخش SIGNAL و PREAMBLE آماده فرستاده شدن به گیرنده شود.

(تابع هایی که عمل جا به جایی بیت ها را انجام میدهند در زیر آورده شده است)

The first permutation is defined by the rule

$$i = (N_{CBPS}/16)\,(k \bmod 16) + \text{floor}(k/16) \qquad k = 0,1,\ldots,N_{CBPS} - 1 \tag{15}$$

The function floor (.) denotes the largest integer not exceeding the parameter.

The second permutation is defined by the rule

$$j = s \times \text{floor}(i/s) + (i + N_{CBPS} - \text{floor}(16 \times i/N_{CBPS})) \bmod s \qquad i = 0,1,\ldots N_{CBPS} - 1 \tag{16}$$

The value of s is determined by the number of coded bits per subcarrier, N_{BPSC}, according to

$$s = \max(N_{BPSC}/2,1) \tag{17}$$

The deinterleaver, which performs the inverse relation, is also defined by two permutations.

حال فایل تولید شده In.txt داده های ورودی ابتدایی به بخش فرستنده در آن ذخیره شده است و فایل Out_Transmitter.txt داده های خروجی که از فرستنده خارج میشود حال به سراغ پیاده سازی وریلاگ می رویم.

پیاده سازی وریلاگ:

در این بخش از پیاده سازی هایی که در بخش های قبل انجام شده است استفاده کرده ایم و ماژول هایی که از قبل پیاده سازی شده اند را دیگر به تفکیک توضیح نمیدهیم و در صورت نیاز میتوان به گزارش های موجود فاز های قبل که در فایل ارسالی قرار داده شده است مراجعه کرد تا از نحوه پیاده سازی آن مطلع شد.

```
21  module transmitter(
22       data_in,
23       n_pad,
24       Clk,
25       reset,
26       data_out,
27       ready
28      );
29      parameter frame_size = 180;
30      input data_in;
31      input Clk;
32      input reset;
33      input [5:0] n_pad;
34      output [frame_size-1:0]data_out;
35      output  ready;
36
```

ورودی های فرستنده داده های ورودی، کلاک ، Reset و تعداد بیت های Pad می باشد.

خروجی های فرستنده نیز یک فریم کامل به همراه یک بیت برای مشخص کردن معتبر بودن خروجی است.

در بخش بعدی ابتدا داده های ورودی را با توجه به تعداد آن ها جدا کرده و بخش SIGNAL و PREAMBLE را ابتدا دریافت میکنیم. سپس بخش DATA را توسط تابع Scrambler تبدیل میکنیم.

خروجی های تابع Scramble به عنوان ورودی های ماژول Encoder داده میشود تا داده ها کدگذاری شوند و در نهایت خروجی های بخش Encoder نیز تحت عنوان ورودی به Interleaver داده میشود تا فریم نهایی خروجی آماده شود.

کد بخش های توضیح داده شده در صفحه بعد آمده است و توضیحات مرتبط به هر ماژول نیز به آن اشاره شده است که در کدام گزارش آمده است.

[ماژول Scramble: گزارش فاز 1 با نام phase1.pdf صفحه 6 تا 9 ، ماژول encoder: گزارش فاز 2 با نام Phase2.pdf صفحه 6 و 7 ، ماژول interleaver: گزارش فاز 3 با نام phase3.pdf صفحه 4 تا 6]

```
    assign data_out = ready_final ? {out_stage3,signal_frame } : 0;
    assign ready = ready_final;

    scrambler data_scramble (
    .in_data(data_in),
    .Clk(Clk),
    .scrambler_seed(scrambler_seed),
    .reset(rst_scrambler),
    .en(en_scrambler),
    .out_data(out_scrambler)
    );


    always@(posedge Clk)begin
        if(reset)begin
            C_state <= signal_preamble;
            Counter <= 7'd0;
            out <= 0;
        end
        else
        begin
            case(C_state)

                signal_preamble:begin
                    if(Counter < 35)begin
                        out <= data_in;
                        Counter <= Counter + 1'b1;
                        signal_frame <= {data_in , signal_frame[35:1]};
                        signal_field <= {signal_field[22:0] , data_in};
                        en_scrambler <= 1'b0;
                        rst_scrambler <= 1'b1;
                    end
                    else begin
                        out <= data_in;
                        Counter <= 0;
                        signal_frame <= {data_in , signal_frame[35:1] };
                        signal_field <= {signal_field[22:0] , data_in};
                        en_scrambler <= 1'b1;
                        rst_scrambler <= 1'b0;
                        in_scrambler <= data_in;
                        C_state <= service;
                    end
                end

                service:begin
                    if(Counter < 15)begin
                        phase1_ready <= 1;
                        out <= out_scrambler;
                        Counter <= Counter + 1'b1;
                        in_scrambler <= data_in;
                        en_scrambler <= 1'b1;
                        rst_scrambler <= 1'b0;
                    end
                    else begin
                        out <= out_scrambler;
                        Counter <= 0;
                        in_scrambler <= data_in;
                        en_scrambler <= 1'b1;
                        rst_scrambler <= 1'b0;
                        C_state <= psdu;
                    end
                end

                psdu:begin
                    if(Counter < {length , 3'b000}) begin// 8bit * Num of frames
                        out <= out_scrambler;
                        Counter <= Counter + 1'b1;
                        in_scrambler <= data_in;
                        en_scrambler <= 1'b1;
                        rst_scrambler <= 1'b0;
                    end
                    else begin
                        out <= out_scrambler;
                        Counter <= 0;
                        in_scrambler <= data_in;
                        en_scrambler <= 1'b1;
                        rst_scrambler <= 1'b0;
                        C_state <= tail_pad;
                    end
                end

                tail_pad:begin
                    if(Counter < (6 + n_pad))begin
                        out <= out_scrambler;
                        Counter <= Counter + 1'b1;
                        en_scrambler <= 1'b1;
                        rst_scrambler <= 1'b0;
                    end
                end
            endcase
        end
    end

    encoder stage2 (.data_in(out), .data_out(out_stage2),.Clk(Clk),.reset(1'b1),.rate(1'b0),.en(phase1_ready));
    interleaver stage3( .in_data({out_stage2}), .out_data(out_stage3),  .Clk(Clk), .ready(ready_final), .en(phase1_ready));

```

حال برای شبیه سازی و چک کردن درستی کد پیاده سازی شده از یک تست بنچ استفاده میکنیم.

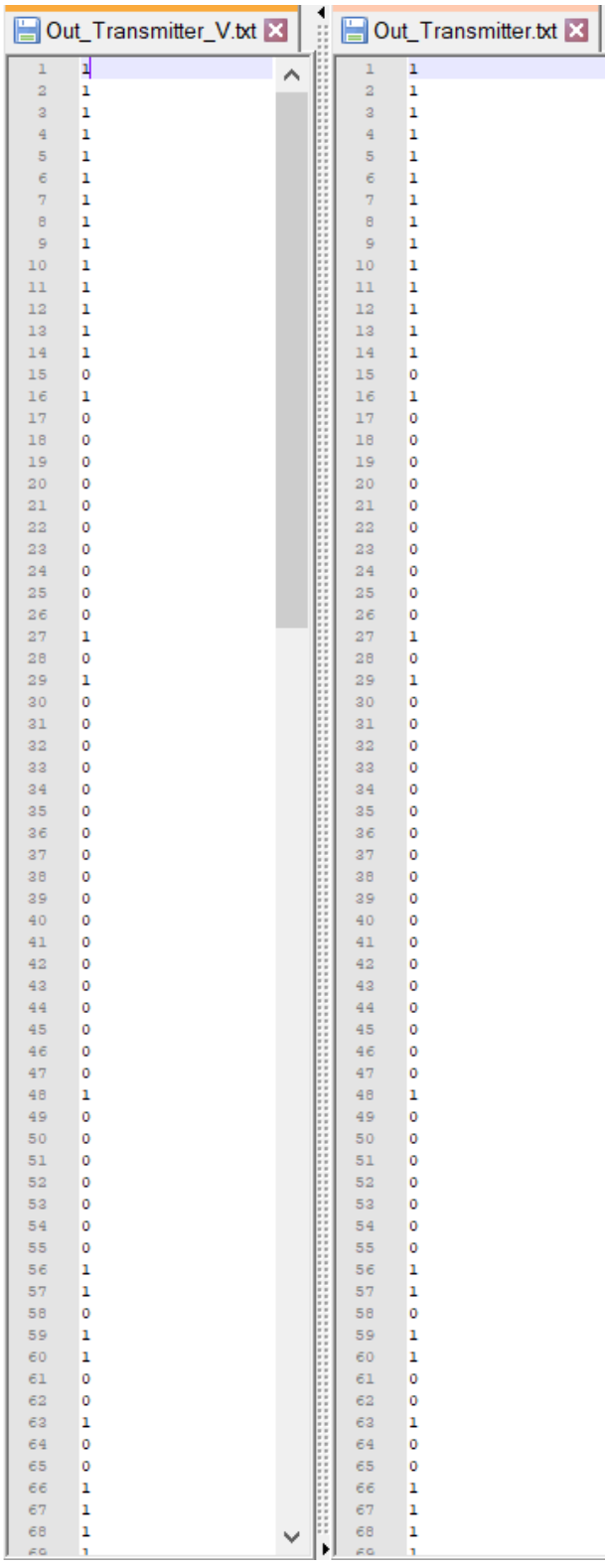
```
`timescale 1ns / 1ps
module transmitter_tb();

    reg Clk;
    reg reset;
    reg in;
    reg [5:0]pad_in;
    wire [179:0]out;
    integer op1, op_out ,op2;
    integer i , tempin , temp , j;
    wire ready;
    reg inreg[230:0];

transmitter DUT(
     .data_in(in),
     .n_pad(pad_in),
     .Clk(Clk),
     .reset(reset),
     .data_out(out),
     .ready(ready)
    );

    initial
    begin
        Clk = 0;
        reset = 1;
        op1 = $fopen("In.txt" , "r");
        op2 = $fopen("N_pad.txt" , "r");
        op_out = $fopen ("Out_Transmitter_V.txt" , "w");
        #20;
        reset = 0;
        for(i=0 ; i < 2000 ; i = i+1)
          @(posedge Clk);
        $stop;
    end

    always #10 Clk = ~Clk;

    always@(posedge Clk)
    begin
        temp = $fscanf(op2 , "%b" , pad_in);
        tempin = $fscanf(op1 , "%b" , in);
        if(ready)
            begin
            for(j=0 ; j<180; j = j+1)
            $fwrite(op_out , "%b\n" , out[j]);
            $stop;
            end
    end
endmodule
```

در این تست بنچ همانطور که مشاهده میشود فایل In.txt که در بخش پیاده سازی متلب ساخته شده است و به عنوان ورودی به متلب داده شد به ورودی وریلاگ داده میشود و خروجی پیاده سازی وریلاگ را در یک فایل با نام Out_Transmitter_V.txt ذخیره میکنیم.

حال برای چک کردن درستی کارکرد فرستنده کافی است که فایل خروجی کد فرستنده متلب را با خروجی فرستنده وریلاگ با یکدیگر مقایسه کنیم.(دو فایل Out_Transmitter.txt (خروجی شبیه سازی متلب) و Out_Transmitter_V.txt (خروجی شبیه سازی وریلاگ))

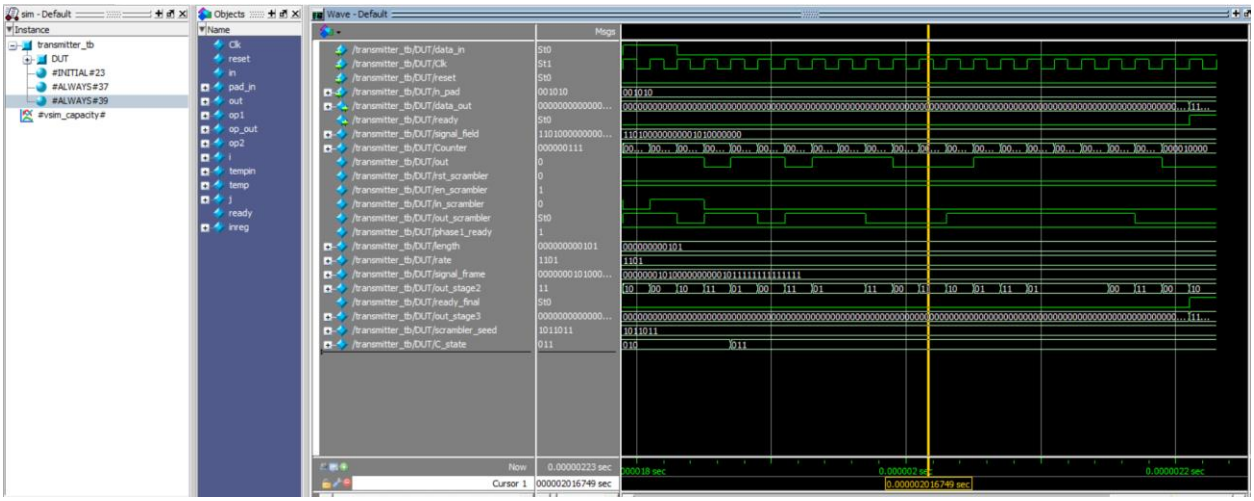
توجه: همانطور که در پیاده سازی متلب گفته شد چون تولید داده های ورودی به صورت رندوم است در صورت اجرای کد باید فایل In.txt موجود در پوشه ای که در آن شبیه سازی وریلاگ انجام میشود آپدیت شود.

با مقایسه مقادیر این دو فایل در صفحه بعد به این نتیجه خواهیم رسید که پیاده سازی ما کاملا درست است.



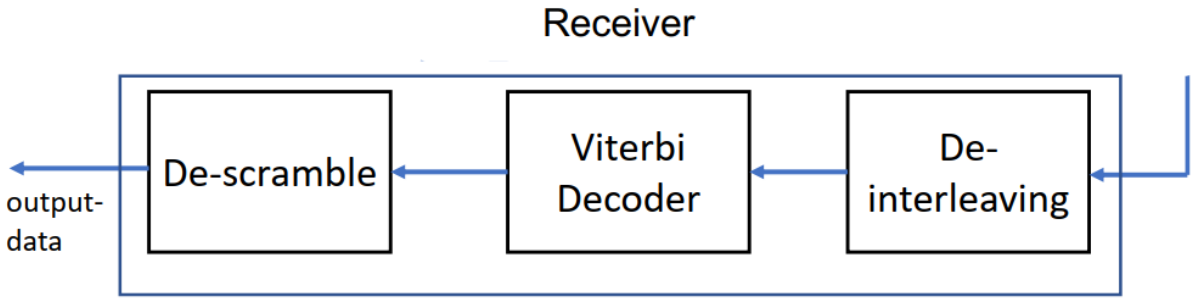
با مقایسه ادامه بیت ها نیز از تطابق خروجی های متلب و وریلاگ میتوان اطمینان حاصل کرد و این نشان دهنده درست بودن پیاده سازی های ما میدهد.

شکل سیگنال های خروجی در شبیه ساز وریلاگ:



مقدمه:

بخش گیرنده:



برای بخش گیرنده باید عملیاتی که در فرستنده انجام شده اند را به ترتیب عکس تکرار کنیم تا به داده های ورودی دست پیدا کنیم.

مراحلی که باید طی کنیم به ترتیب Deinterleaving ، Viterbi Decoder و Descramble می باشد.

بخش های مختلف هر کدام در فاز های قبلی توضیحات کامل راجع به جزئیات آن ها گفته شده است که گزارش های مربوط به آن ها در فایل ارسال شده قرار دارد.(برای بخش DeScramble گزارش با نام Phase1

، برای بخش Viterbi decoder گزارش با نام Phase2 و برای بخش Deinterleaver گزارش با نام Phase3 در فایل ارسالی موجود است که میتوان به آن ها مراجعه کرد).

در ابتدا داده های خروجی فرستنده را به مکان های درست آن باز میگردانیم. سپس با استفاده از Decoder آن ها را از حالت کدگذاری شده در می آوریم که الگوریتم Viterbi (در گزارش فاز 2 کامل به آن اشاره شده است) باعث جبران سازی خطا های احتمالی کانال ارتباطی نیز میشود و در نهایت با Descramble کردن به داده های ورودی دست پیدا میکنیم.

حال به سراغ پیاده سازی فرستنده در متلب و وریلاگ میپردازیم.

پیاده سازی متلب:

```
clear;
clc;
f = fopen('Out_Transmitter.txt' , 'r');%Read data that transmitter send
o = fopen('Out_Receiver.txt' , 'w+');%Save output of receiver for verification
data_in = fscanf(f , "%d");
flag = 1;
counter = 1;
while(flag)%Check when a new frame come
    preamble(counter) = data_in(counter);
    if(preamble(counter) == 1)
        counter = counter + 1;
    end
    if(counter == 12)
        flag = 0;
        preamble(counter) = data_in(counter);
    end
end
for i=1:24 %Save SIGNAL Frame
    signal_d(i) = data_in(i+counter);
end
index = i + counter;
rate = signal_d(1:4);
rate_bin = dec2bin(rate);
length_d = signal_d(6:17);
rate_param = SET_RATE_PARAMS_R(rate_bin);%Calculate different parameters of Data rate
N_SYM = ceil((16 + 8*5+6)/rate_param.NDBPS);
N_DATA = N_SYM * rate_param.NDBPS;
N_PAD = N_DATA - (16 + 8 * 5 + 6);
n_pad = dec2bin(N_PAD , 6);
in = data_in(index+1:index+(N_DATA*2));
scramble_state = [1 ,0 ,1 ,1 ,0 ,1 ,1 ];
```

همانطور که مشخص است خروجی فرستنده که در قبل توضیحات آن داده شد را به عنوان ورودی بخش گیرنده قرار میدهیم(فایل Out_Transmitter.txt به عنوان ورودی گرفته میشود که خود خروجی فرستنده است)

برای چک کردن و درستی پیاده سازی گیرنده خروجی گیرنده را تحت عنوان Out_Receiver.txt ذخیره میکنیم که در انتها آن را با ورودی فرستنده که In.txt است مقایسه کنیم.

در ادامه کد با توجه به بخش اول ورودی و PREAMBLE تصمیم میگیریم که آیا فریم جدید آمده است یا خیر.

پس از آن بخش های مختلف DATA و SIGNAL را جدا میکنیم.

میدانیم عملیات هایی که در فرستنده انجام شده است همگی بر روی بخش DATA بوده است پس عملیات بازگشت را باید بر روی آن انجام دهیم.

```
34      %%Deinterleaver
35 -    N_CBPS = rate_param.NCBPS; %This part in Phase4 will be one of inputs that come from
36                  % Signal field of input frame
37 -    N_BPSC = rate_param.NBPSC; %This part in Phase4 will be one of inputs that come from
38                  % Signal field of input frame
39 -    if(N_BPSC/2 < 1)
40 -        S = 1;
41 -    else
42 -        S = N_BPSC/2;
43 -    end
44 -    step = length(in)/N_CBPS;
45 -    out_deinterleaver = zeros(2*N_DATA , 1);
46 -  for i=1:step
47 -      deInterleave_stage1 = zeros(N_CBPS , 1);
48 -      for j=0:N_CBPS-1
49 -          index =(S*floor(j/S) + mod((j + floor(16 * j/N_CBPS)) , S));
50 -          deInterleave_stage1(16*index - (N_CBPS - 1)*floor(16*index/N_CBPS) + 1) = in(j+1 + (i-1)*N_CBPS);
51 -          out_deinterleaver(16*index - (N_CBPS - 1)*floor(16*index/N_CBPS) + 1 + (i-1)*N_CBPS) = in(j+1 + (i-1)*N_CBPS);
52 -      end
53 -  end
```

سپس با توجه به فریمی که گیرنده دریافت کرده است و تشخیص Rate و باقی پارامتر های مدنظر دومرحله برای برگشتن از مکان هایی که در فرستنده داده ها در آن قرار گرفته است با توجه به استاندارد انجام شده است که روابط آن به شکل زیر است.(جزئیات دقیق تر این بخش در گزارش فاز 3 با نام phase3.pdf در صفحه 8 به بعد موجود است)

The first permutation is defined by the rule

$$i = s \times \text{floor}(j/s) + (j + \text{floor}(16 \times j/N_{CBPS})) \bmod s \qquad j = 0,1,\ldots N_{CBPS} - 1$$

where

s is defined in Equation (17).

This permutation is the inverse of the permutation described in Equation (16).

The second permutation is defined by the rule

$$k = 16 \times i - (N_{CBPS} - 1)\text{floor}(16 \times i/N_{CBPS}) \qquad i = 0,1,\ldots N_{CBPS} - 1$$

This permutation is the inverse of the permutation described in Equation (15).

پس از آن به سراغ decoder میرویم:

```
55      %%Viterbi_Decoder
56 -    Num_data_in = N_DATA*2;
57 -    state_Num = 64;
58 -    state = 1:64;
59 -    node_Cost = zeros(state_Num , Num_data_in/2+1);
60 -    node = zeros(state_Num , Num_data_in/2+1);
61 -    in = out_deinterleaver;
62 -    for i=1:Num_data_in/2
63 -        a = dec2bin(in(2*i-1));
64 -        b = dec2bin(in(2*i));
65 -        data(i) = bin2dec([a , b])+1;
66 -    end
67 -    node(1,1) = 1;%%Mark node as visited
68 -    trace = zeros(state_Num , Num_data_in/2 );
69 -    trace_back = zeros(state_Num , Num_data_in/2 );
```

در کد بالا ابتدا هزینه هر کدام از گره ها را در ابتدا مقدار دهی میکنیم و همچنین داده های سریال ورودی را به صورت دوبیت دوبیت تبدیل میکنیم.

دو آرایه دیگر برای ذخیره سازی مسیر و برگشت آن نیز در نظر میگیریم. از آنجایی که میدانیم مقدار دهی اولیه encoder با 6'b000000 است پس گره اول را تحت عنوان دیده شده در نظر میگیریم.

با توجه به encoder ما 64 حالت مختلف داریم.

```
for j=1:state_Num
    if(node(j , i) == 1)
        [Next_state0 , in0 , out0 , Next_state1 , in1 , out1] = Calc_state(state(j));
        if(cost_calc(data(i) , out0) + node_Cost(j , i) < node_Cost(Next_state0 , i+1)  || node(Next_state0 , i+1)==0)
            node(Next_state0 , i+1) = 1;
            node_Cost(Next_state0 , i+1) = cost_calc(data(i) , out0) + node_Cost(j , i);
            if(i==1)
                trace(Next_state0 , i) = in0;
            else
                trace(Next_state0 , 1:i) = [trace_back(j , 1:i-1) , in0];
            end
        end
        if(cost_calc(data(i) , out1) + node_Cost(j , i) < node_Cost(Next_state1 , i+1)  || node(Next_state1 , i+1)==0)
            node(Next_state1 , i+1) = 1;
            node_Cost(Next_state1 , i+1) = cost_calc(data(i) , out1) + node_Cost(j , i);
            if(i==1)
                trace(Next_state1 , i) = in1;
            else
                trace(Next_state1 , 1:i) = [trace_back(j , 1:i-1) , in1];
            end
        end
```

در قسمت بالا با توجه به ورودی ها ابتدا با استفاده از تابع cost_calc هزینه hamming را محاسبه میکنیم سپس برای هر گره میتوان از دو مسیر به آن رسید:مسیر اول اگر ورودی به encoder 0 بوده باشد و مسیر دوم اگر ورودی به encoder 1 بوده باشد.

حال با توجه به هزینه های انباشته تصمیم گیری را انجام میدهیم.(in0 = 0 و in1 = 1)

سپس به سراغ مرحله بعد و ستون بعدی می رویم.

در ابتدا باید تشخیص دهیم که به چه گره و حالتی خواهیم رفت که آن نیز با یک تابع تحت عنوان calc_state بدست می آید.

قطعه کد calc_state:

```
function[state0 , in0 , out0 , state1 , in1 , out1]= Calc_state(in)
    CS = dec2bin(in-1 , 6);
    in0 = 0;
    in1 = 1;
    state0 = 1 + bin2dec(['0',CS(1,1:5)]);
    state1 = 1 + bin2dec(['1',CS(1,1:5)]);
```

با توجه به کد بالا حالت گره های بعدی در صورت ورودی 0(state0) و ورودی 1 (state1) است. خروجی های out0 و out1 نیز از طریق XOR حالت گره ها بدست می آید.

قطعه کد cost_calc:

```
function [out]= cost_calc(in1,in2)
    if(in1==in2)
      out=0;
    end
    if(in1==1&&in2==2||in2==1&&in1==2)
      out=1;
    end
    if(in1==1&&in2==3||in2==1&&in1==3)
      out=1;
    end
        if(in1==1&&in2==4||in2==1&&in1==4)
      out=2;
        end
        if(in1==2&&in2==3||in2==2&&in1==3)
      out=2;
        end
    if(in1==2&&in2==4||in2==2&&in1==4)
      out=1;
        end
     if(in1==3&&in2==4||in2==3&&in1==4)
      out=1;
    end
    end
```

کد بالا نیز با توجه به ورودی های دوبیتی که داده میشود هزینه hamming را محاسبه میکند

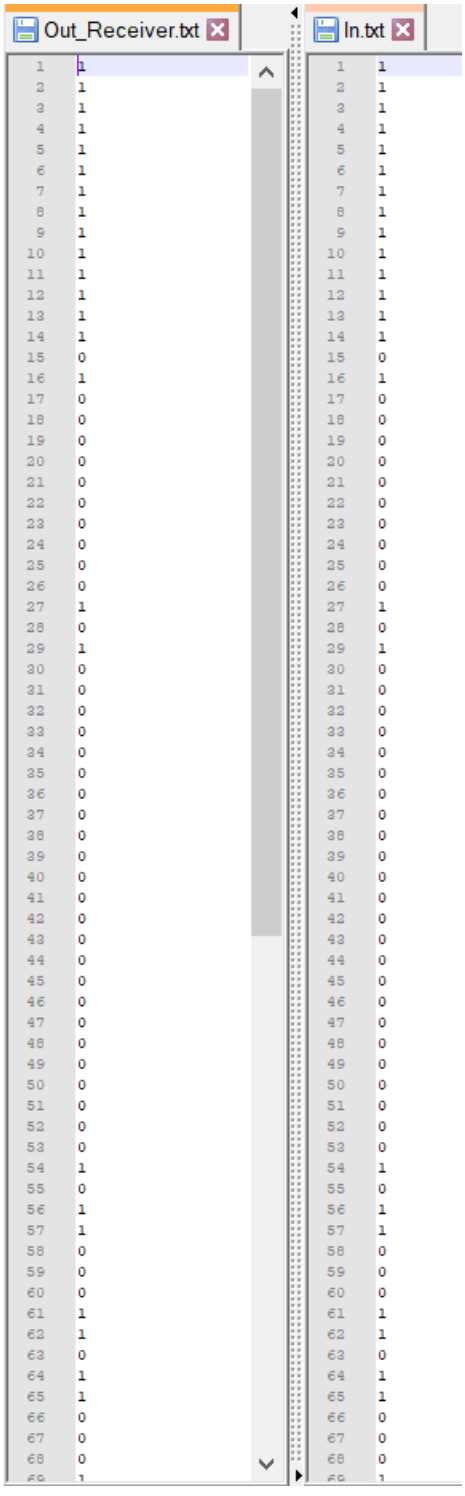
```
97      %Compare last iteration for find node that we should trace back
98 -    min = 1;
99 -    for i=2 : state_Num
100 -       if(node_Cost(i ,Num_data_in/2+1) < node_Cost(min ,Num_data_in/2+1))
101 -           min = i;
102 -       end
103 -   end
104 -   decode = zeros(Num_data_in/2 , 1);
105 -   for i=1:Num_data_in/2
106 -          decode(i) = trace(min,i);
107 -   end
```

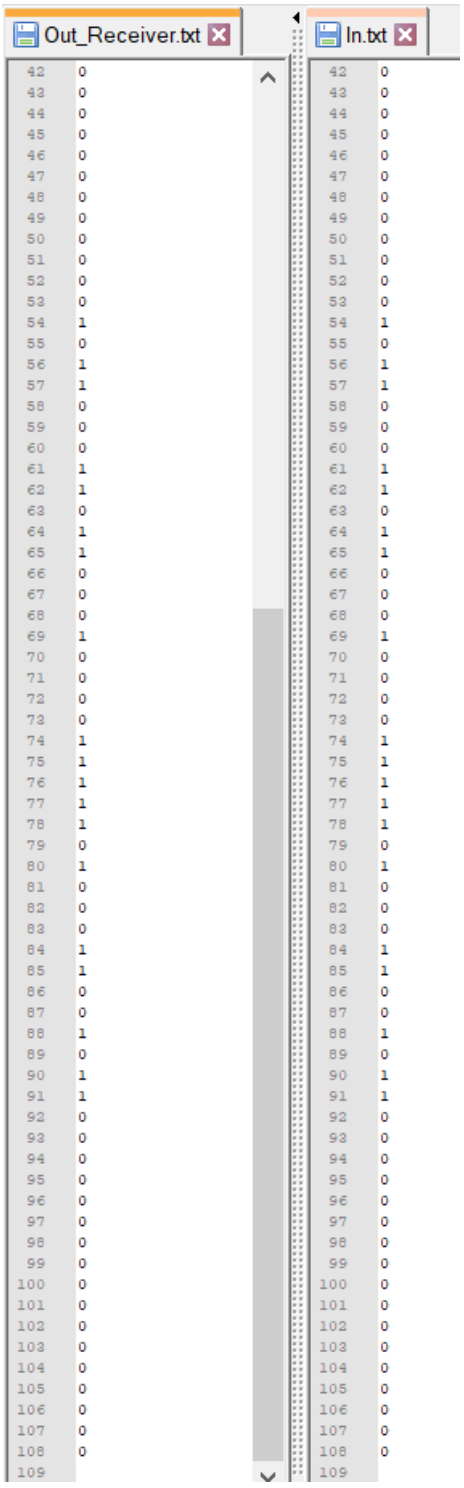
در نهایت مشابه بالا کوچکترین مسیر را پیدا میکنیم و عملیات بازیابی را بر روی آن انجام می دهیم. (جزئیات دقیق تر این بخش در گزارش فاز 2 با نام phase2.pdf در صفحه 15 به بعد موجود است)

حال در انتها باید مقادیر بدست آمده از decoder را descramble کنیم تا نتیجه مورد نظر یافت شود.

```
109     %Descramble
110 -   data_d = zeros(Num_data_in/2 , 1);
111 -   for i=1:Num_data_in/2
112 -       scramble_data = xor(scramble_state(1),scramble_state(4));
113 -       scramble_state = [scramble_state(2:7) ,scramble_data];
114 -       data_d(i) = xor(decode(i) ,scramble_data);
115 -   end
116
117 -   for i=1:counter
118 -      fprintf(o , "%d \n" ,   preamble(i));
119 -   end
120 -   for i=1:24
121 -      fprintf(o , "%d \n" ,   signal_d(i));
122 -   end
123 -   for i=1:Num_data_in/2
124 -      fprintf(o , "%d \n" ,   data_d(i));
125 -   end
126 -   fclose(f);
127 -   fclose(o);
```

حال برای چک کردن درستی بخش گیرنده کافی است خروجی گیرنده Out_Receiver.txt را با ورودی فرستنده In.txt مقایسه کنیم، زیرا هدف ما نیز همین بود که پس از تغییر داده های ورودی و فرستادن آن روی کانال در گیرنده همان داده های ابتدایی را دریافت کنیم.


Out_Receiver.txt
In.txt



همانطور که از تصاویر بالا مشخص است خروجی گیرنده کاملا مطابق ورودی فرستنده است و پیاده سازی گیرنده به طور کامل صحیح است و دقیقا ورودی را به ما میدهد.