به نام خدا

دانشکده مهندسی برق

گزارش فاز 3 پروژه

درس سیستم های مبتنی بر FPGA/ASIC

محمدامین حاجی خداوردیان

97101518

استاد:دکتر مهدی شعبانی

نیمسال دوم 99-00

در این فاز به پیاده سازی Interleaver و deinterleaver میپردازیم.

مقدمه

بخش Interleaver :

در پیاده سازی استاندارد مربوطه خروجی های Encoder باید تحت تاثیر Interleaving قرار بگیرند. عمل Interleaving در دو مرحله انجام می شود و در هر مرحله با جا به جایی مکان بیت های ورودی، بیت های خروجی بدست می آید.

در مرحله نخست باید بیت های ورودی را با توجه به رابطه زیر پخش کنیم:

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

این مرحله این تضمین را به ما میدهد که بیت های کدشده مجاور در حامل های غیرمجاور قرار گیرند.

سپس مرحله دوم باید با توجه به رابطه زیر بیت ها پراکنده شوند:

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

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

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

مرحله دوم باعث می شود که بیت های کدشده مجاور به طور متناوب بر روی بیت های کمتر قرار گیرند بنابراین از بیت های با قابلیت اطمینان پایین جلوگیری میشود.

برای مدولاسیون های مختلف نحوه پخش کردن بیت های ورودی به گونه مختلفی انجام میشود که آن را متناسب با جدول زیر انجام میدهیم.

در این فاز پروژه ما با BPSK با Datarate = 6Mbit/s و Coding Rate = ½ کار خواهیم کرد.

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

پیاده سازی متلب:

در آدرس Matlab/interleaver/interleaver.m فایل متلب وجود دارد.

برای این بخش ابتدا خروجی Encoder از فاز قبل با نام Out_rate0.txt خوانده میشود.

سپس با توجه به length داده شده در فریم میتوانیم تعداد عملیاتی که باید انجام شود را پیدا کنیم در اینجا فرض شده است ورودی های ما 96 بیت هستند پس باید در دو مرحله 48 بیتی انجام شود.(N_{CBPS} = 48)

```
clear;
clc;
%%%tx
f = fopen('Out_rate0.txt' , 'r');
o = fopen('Out_interleaver_m.txt' , 'w+');
in = fscanf(f , "%d");
N_CBPS = 48; %This part in Phase4 will be one of inputs that come from
             % Signal field of input frame
N_BPSC = 1; %This part in Phase4 will be one of inputs that come from
             % Signal field of input frame
if(N_BPSC/2 < 1)
    S = 1;
else
    S = N_BPSC/2;
end
step = length(in)/N_CBPS;
```

سپس با توجه به مقدار N_{CBPS} تعداد مراحلی که باید تکرار شود را پیدا میکنیم.

```
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)) = in(j+1 + (i-1)*48);
        disp([in(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
```

سپس با توجه به روابط داده شده در استاندارد دو مرحله را پیاده سازی میکنیم و بیت های ورودی را متناسب با آن پخش میکنیم.

خروجی ها را در یک فایل با نام Out_interleaver_m.txt ذخیره میکنیم تا در تست کردن پیاده سازی توسط وریلاگ از آن استفاده کنیم.

پیاده سازی وریلاگ:

در آدرس Verilog/interleaver فایل interleaver.v و interleaver_tb.v وجود دارد.

ابتدا به توضیح کد اصلی میپردازیم.

ورودی های ماژول interleaver به شکل زیر است:

```
module interleaver(
    in_data,
    out_data,
    Clk,
    ready,
    en
);
    parameter N_CBPS = 48;
    parameter N_BPSC = 1;
    parameter length = 96;
    input [1:0]in_data;
    input Clk;
    input en;
    output [length-1:0]out_data;
    output reg ready;

    wire[2:0] S_interleave;
    reg [47:0] counter = 0;
    reg [10:0]step_counter = 0;
    wire [10:0]step;
    wire [10:0]first_permutation0 , first_permutation1;
    wire [10:0]second_permutation0 , second_permutation1;
    reg [length-1:0]interleave_stage2;
```

یک داده ورودی دوبیتی که از Encoder می آید به آن داده میشود.(در فاز قبل پیاده ساز encoder با rate= ½ به این صورت بود که در هر کلاک دو بیت خروجی میدهد).

یک کلاک و یک فعال ساز(en) برای فعال کردن و غیر فعال کردن ماژول وجود دارد.

خروجی ها نیز شامل یک خروجی با تعداد بیت مشخص که چه تعدادی بیت کدشده را میخواهیم Interleaving روی آن انجام دهیم که با توجه به length و تعداد pad_bit مشخص میشود.(در اینجا به عنوان مثال 96 بیت گرفته شده است).

یک خروجی 1 بیت ready هم وجود دارد که مشخص میکند در چه زمانی داده خروجی ما معتبر است.

```
25    assign S_interleave = (N_BPSC/2 < 1) ? 1 : N_BPSC/2;
26    assign step = length/N_CBPS;
27
28    assign first_permutation0 = ((N_CBPS/16)*(counter%16) + (counter/16));
29    assign first_permutation1 = ((N_CBPS/16)*((counter + 1)%16) + ((counter + 1)/16));
30    assign second_permutation0 = S_interleave*(first_permutation0/S_interleave) + ((first_permutation0 + N_CBPS - 16*first_permutation0/N_CBPS))%S_interleave;
31    assign second_permutation1 = S_interleave*(first_permutation1/S_interleave) + ((first_permutation1 + N_CBPS - 16*first_permutation1/N_CBPS))%S_interleave;
32
33    assign out_data = (ready) ? interleave_stage2 : 0;
34
35    always @(counter) begin
36
37
38        if(counter == N_CBPS)begin
39
40            step_counter <= step_counter + 1;
41            counter <= 0;
42
43        end
44
45    end
```

در کد بالا ابتدا بخش ترکیبی مدار آورده شده است که شامل حساب کردن تعداد مراحل، مقدار S و دومرحله ای که رابطه آن به ما داده شده است را حساب میکنیم و در هر مرحله چک میکنیم که آیا به تعداد N_{CBPS} ورودی گرفته ایم اگر جواب مثبت باشد به سراغ دسته های بعدی ورودی ها به تعداد N_{CBPS} میرویم.

```
47    always @(posedge Clk) begin
48
49        if(en)begin
50
51            counter <= counter + 2 ;
52            interleave_stage2[step_counter * N_CBPS + second_permutation0] <= in_data[0];
53            interleave_stage2[step_counter * N_CBPS + second_permutation1] <= in_data[1];
54
55            if(step_counter == step)begin
56                ready <= 1'b1;
57                step_counter <= 0;
58                counter <= 0;
59            end
60            else
61                ready <= 1'b0;
62        end
63    end
```

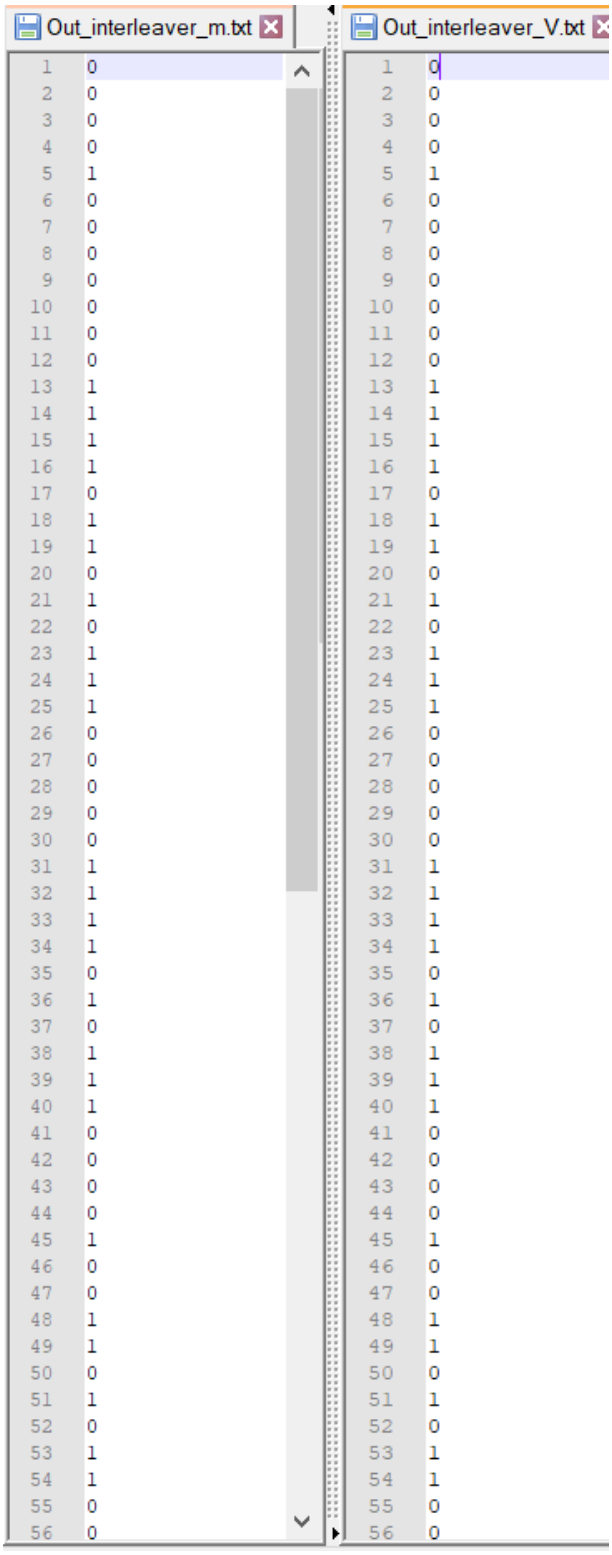
بخش ترتیبی مدار نیز به این صورت است که در هر کلاک ساعت ورودی ها را با توجه به خانه ای که باید در آن قرار بگیرد به عنوان خروجی میدهد(مکان آن ها را در بخش ترکیبی با استفاده از روابط استاندارد به دست آورده ایم) و چون در هر کلاک دو بیت را مورد بررسی قرار میدهیم شمارنده خود را به گونه ای در نظر گرفته ایم که دوگام به جلو حرکت کند.

کد تست بنچ نیز به صورت زیر است:

```
module interleaver_tb;
    reg [1:0] in;
    reg Clk=0;
    wire [95:0]out;
    wire ready;
    reg en = 0;

    integer In_data , Out_data , f1;
    integer i , j;
    always #5 Clk = ~Clk;

    initial
    begin

        In_data  = $fopen("Out_rate0.txt", "r");
        Out_data = $fopen("Out_interleaver_V.txt", "w");
        for(i = 0 ; i < 300 ; i = i + 1)
            @(posedge Clk);
        $stop;

    end
    always @(posedge Clk) begin
        en = 1;
        f1 = $fscanf(In_data, "%b\n%b\n", in[0],in[1]);
        if(ready)begin
            for(j=0 ; j<96; j = j+1)
            $fwrite(Out_data , "%b\n" , out[j]);
            $stop;
        end
    end

     interleaver dut ( .in_data(in), .out_data(out),.Clk(Clk),.ready(ready) , .en(en));

endmodule
```

در این کد خروجی های Encoder (فاز قبل) که در پیاده سازی متلب از آن استفاده کردیم را به عنوان ورودی میدهیم و خروجی Interleaver را در یک فایل با نام Out_interleaver_v.txt ذخیره میکنیم.

برای چک کردن درستی باید خروجی شبیه سازی با متلب که با نام Out_interleaver_m.txt است را با Out_interleaver_v.txt مقایسه کنیم.



همانطور که در شکل بالا دیده میشود این دو خروجی دقیقا با هم یکسان هستند بنابراین پیاده سازی وریلاگ صحیح است.(خروجی متلب و وریلاگ دقیقا یکسان شده اند.)

مقدمه

بخش Deinterleaver :

در این بخش میخواهیم بیت هایی که در Interleaver درمکان های مختلف پخش کرده ایم را بازیابی کنیم.

برای اینکار دو مرحله عملیاتی که در Interleaver انجام میشود را باید عکس آن را انجام داد تا به ورودی هایی که توسط Encoder به ماژول ما داده شده است یافت شود.

در استاندارد دو روابطی که ما را به این هدف می رساند موجود است و آن ها عبارت هستند از:

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

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

کافی است در پیاده سازی هایی که برای Interleaver انجام داده ایم مراحلی که طی میکردیم را تغییر دهیم تا به نتیجه مطلوب دست پیدا کنیم بنابراین در بخش Deinterleaver ما تفاوت زیادی به جز در روابط بالا نخواهیم داشت.

پیاده سازی متلب:

در آدرس Matlab/deinterleaver/deinterleaver.m فایل متلب وجود دارد.

در این بخش خروجی پیاده سازی Interleaver را با نام Out_interleaver_m.txt را به عنوان ورودی دریافت میکنیم.

```
clear;
clc;
%%%rx
f = fopen('Out_interleaver_m.txt' , 'r');
o = fopen('Out_deinterleaver_m.txt' , 'w+');
in = fscanf(f , "%d");
N_CBPS = 48; %This part in Phase4 will be one of inputs that come from
             % Signal field of input frame
N_BPSC = 1; %This part in Phase4 will be one of inputs that come from
             % Signal field of input frame
if(N_BPSC/2 < 1)
    S = 1;
else
    S = N_BPSC/2;
end
step = length(in)/N_CBPS;
for i=1:step
    deInterleave_stage1 = zeros(N_CBPS , 1);
    for j=0:N_CBPS-1
        index =(S*floor(j/S) + mod((j + floor(16 * j/N_CBPS)) , S));
        deInterleave_stage1(16*index - (N_CBPS - 1)*floor(16*index/N_CBPS) + 1) = in(j+1 + (i-1)*48);
    end
    fprintf(o , "%d\n" , deInterleave_stage1);
end
```

یافتن تعداد دفعاتی که به تعداد N_{CBPS} بیت باید این روند انجام شود و مشخص کردن مقادیر آن ها مشابه Interleaver پیاده سازی شده است.

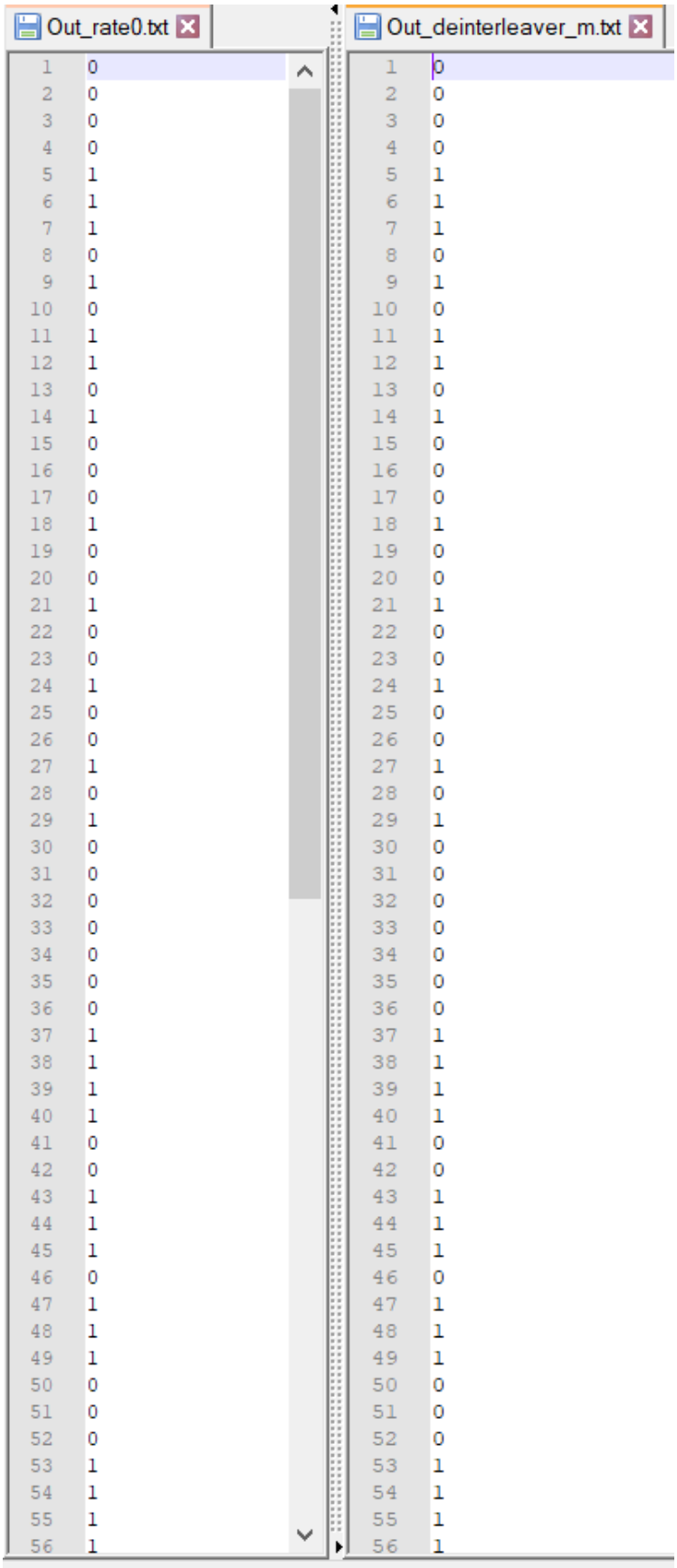
حال در بخش آخر(که در باکس قرمز رنگ مشخص شده است) به جای روابط مربوط به Interleaver از روابطی که برای deinterleaver در اختیار داریم استفاده میکنیم.

خروجی این بخش را با عنوان Out_deinterleaver_m.txt ذخیره میکنیم.

برای چک کردن درستی پیاده سازی این بخش کافی است که خروجی deinterleaver را با ورودی interleaver(که ورودی آن داده Encode شده در فاز قبلی پروژه بود) با نام Out_rate0.txt مقایسه کنیم.

همانطور که در تصویر صفحه بعد دیده میشود خروجی deinterleaver دقیقا برابر ورودی interleaver است و این به آن معنا است که شبیه سازی های انجام شده در متلب به طور کامل صحیح است.

حال در بخش بعد به پیاده سازی وریلاگ میپردازیم.


Out_rate0.txt
Out_deinterleaver_m.txt

پیاده سازی وریلاگ:

در آدرس Verilog/deinterleaver فایل deinterleaver.v و deinterleaver_tb.v وجود دارد.

ابتدا به توضیح کد اصلی میپردازیم.

ورودی های ماژول deinterleaver به شکل زیر است:

```
module deinterleaver(
    in_data,
    out_data,
    Clk,
    ready,
    en
);
    parameter N_CBPS = 48;
    parameter N_BPSC = 1;
    parameter length = 96;
    input [1:0]in_data;
    input Clk;
    input en;
    output [length-1:0]out_data;
    output reg ready;

    wire[2:0] S_deinterleave;
    reg [47:0] counter = 0;
    reg [10:0]step_counter = 0;
    wire [10:0]step;
    wire [10:0]first_permutation0 , first_permutation1;
    wire [10:0]second_permutation0 , second_permutation1;
    reg [length-1:0]deinterleave_stage2;
```

یک داده ورودی دوبیتی که از خروجی interleaver می آید به آن داده میشود.

یک کلاک و یک فعال ساز(en) برای فعال کردن و غیر فعال کردن ماژول وجود دارد.

خروجی ها نیز شامل یک خروجی با تعداد بیت مشخص که چه تعدادی بیت کدشده را که interleave شده است را میخواهیم deInterleaving روی آن انجام دهیم که با توجه به length و تعداد pad_bit مشخص میشود.(در اینجا به عنوان مثال 96 بیت گرفته شده است).

یک خروجی 1 بیت ready هم وجود دارد که مشخص میکند در چه زمانی داده خروجی ما معتبر است.

```
assign S_deinterleave = (N_BPSC/2 < 1) ? 1 : N_BPSC/2;
assign step = length/N_CBPS;

assign first_permutation0 = S_deinterleave * (counter/S_deinterleave) + (counter + 16 * counter/N_CBPS)%S_deinterleave;
assign first_permutation1 = S_deinterleave * ((counter + 1)/S_deinterleave) + ((counter + 1) + 16 * (counter + 1)/N_CBPS)%S_deinterleave;
assign second_permutation0 = 16*first_permutation0 - (N_CBPS - 1)*(16*first_permutation0/N_CBPS);
assign second_permutation1 = 16*first_permutation1 - (N_CBPS - 1)*(16*first_permutation1/N_CBPS);

assign out_data = (ready) ? deinterleave_stage2 : 0;

always @(counter) begin


    if(counter == N_CBPS)begin

        step_counter <= step_counter + 1;
        counter <= 0;

    end

end
```

در کد بالا ابتدا بخش ترکیبی مدار آورده شده است که شامل حساب کردن تعداد مراحل، مقدار S و دومرحله ای که رابطه آن به ما داده شده است را حساب میکنیم و در هر مرحله چک میکنیم که آیا به تعداد N_{CBPS} ورودی گرفته ایم اگر جواب مثبت باشد به سراغ دسته های بعدی ورودی ها به تعداد N_{CBPS} میرویم.(تفاوت این بخش با interleaver همانطور که گفته شد فقط در روابط استفاده شده برای یافتن مکان هر بیت است. که در باکس قرمز مشخص شده است)

```
always @(posedge Clk) begin

    if(en)begin

        counter <= counter + 2 ;
        deinterleave_stage2[step_counter * N_CBPS + second_permutation0] <= in_data[0];
        deinterleave_stage2[step_counter * N_CBPS + second_permutation1] <= in_data[1];

        if(step_counter == step)begin
            ready <= 1'b1;
            step_counter <= 0;
            counter <= 0;
        end
        else
            ready <= 1'b0;
    end
end
```

این بخش ترتیبی نیز دقیقا مشابه بخش Interleaver است.

حال به سراغ بررسی تست بنچ میپردازیم.

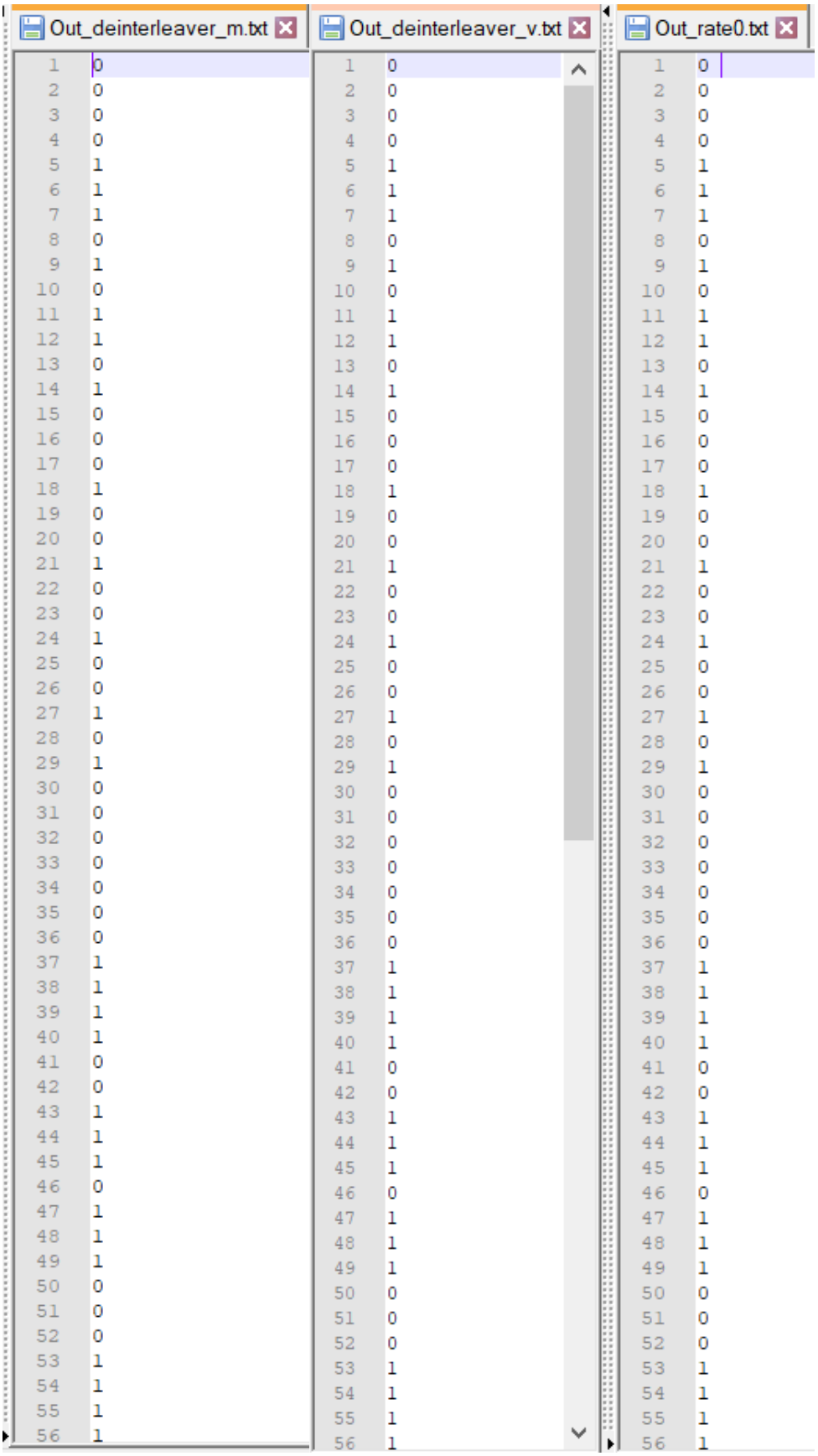
تست بنچ این بخش دقیقا مشابه قبل است با این تفاوت که این بار ورودی ما خروجی بخش Interleaver با نام Out_interleaver_m.txt است تا ببینیم آیا مشابه شبیه سازی متلب خروجی ها برابر ورودی بخش Interleaver که داده های کدشده به دست آمده از فاز قبلی است میشود یا خیر.

```
module deinterleaver_tb;
    reg [1:0] in;
    reg Clk=0;
    wire [95:0]out;
    wire ready;
    reg en = 0;

    integer In_data , Out_data , f1;
    integer i , j;
    always #5 Clk = ~Clk;

    initial
    begin

        In_data  = $fopen("Out_interleaver_m.txt", "r");
        Out_data = $fopen("Out_deinterleaver_v.txt", "w");
        for(i = 0 ; i < 300 ; i = i + 1)
            @(posedge Clk);
        $stop;

    end
    always @(posedge Clk) begin
        en = 1;
        f1 = $fscanf(In_data, "%b\n%b\n", in[0],in[1]);
        if(ready)begin
            for(j=0 ; j<96; j = j+1)
            $fwrite(Out_data , "%b\n" , out[j]);
            $stop;
        end
    end

     deinterleaver dut ( .in_data(in), .out_data(out),.Clk(Clk),.ready(ready) , .en(en));

endmodule
```

خروجی های deinterleaver را تست بنچ بالا در فایلی با نام Out_deinterleaver_v.txt ذخیره میکند.

حال برای چک کردن درستی پیاده سازی وریلاگ کافی است مقادیر خروجی ذخیره شده در فایل بالا را با خروجی deinterleaver متلب با نام Out_deinterleaver_m.txt و ورودی بخش interleaver که نام آن Out_rate0.txt است مقایسه کنیم.

همانطور که در شکل صفحه بعد دیده میشود هر سه فایل مقادیر یکسانی را به ما نشان میدهند که این نشان دهنده آن است که پیاده سازی ها به درستی انجام شده است.


Out_deinterleaver_m.txt
Out_deinterleaver_v.txt
Out_rate0.txt
1 0
2 0
3 0
4 0
5 1
6 1
7 1
8 0
9 1
10 0
11 1
12 1
13 0
14 1
15 0
16 0
17 0
18 1
19 0
20 0
21 1
22 0
23 0
24 1
25 0
26 0
27 1
28 0
29 1
30 0
31 0
32 0
33 0
34 0
35 0
36 0
37 1
38 1
39 1
40 1
41 0
42 0
43 1
44 1
45 1
46 0
47 1
48 1
49 1
50 0
51 0
52 0
53 1
54 1
55 1
56 1