ASIC & FPGA HW(4)

Parham Gilani – 400101859

سوال 1)

Net Delay and Logic Delay are both important factors in determining the maximum working frequency of a digital circuit. Let's break down their differences and discuss how pipelining can improve Net Delay.

1. Net Delay:

- Net Delay refers to the time taken for a signal to propagate through the interconnect wires (metal traces) between different logic elements in a circuit.
- It includes the delay caused by the resistance, capacitance, and other electrical characteristics of the wires.
- Net Delay can vary depending on the routing of signals and the distance they need to travel between logic elements.
- In high-performance designs, reducing net delay is crucial for achieving higher clock frequencies.

2. Logic Delay:

- Logic Delay, on the other hand, refers to the time taken for a signal to propagate through the logic gates themselves.
- It includes the delay caused by the inherent propagation delay of logic gates, such as AND, OR, NOT gates, etc.
- Logic Delay is influenced by factors like gate size, transistor characteristics, and the complexity of the logic function being implemented.
- While reducing logic delay is important, it's often more predictable and easier to optimize compared to net delay.

Now, let's discuss how pipelining can help improve net delay:

Pipelining involves breaking down a sequential process into smaller, independent stages, with each stage executed in parallel. By inserting pipeline registers between these stages, the critical path of the circuit is shortened, leading to several benefits related to net delay:

1. Reduced Interconnect Length:

- Pipelining allows breaking long paths into shorter segments, reducing the distance signals need to travel between logic elements.
- Shorter interconnect lengths lead to lower net delay, as signals propagate faster over shorter distances due to reduced resistance and capacitance effects.

2. Balanced Timing Constraints:

- By dividing the circuit into pipeline stages, each stage can be optimized independently to meet timing constraints.
- This can help balance the timing across the entire circuit, reducing the overall net delay by distributing it more evenly.

3. Improved Clock Distribution:

- Pipelining facilitates easier clock distribution since each pipeline stage operates on a portion of the clock cycle.
- With shorter segments, clock skew issues are minimized, leading to more reliable clocking and reduced net delay related to clock distribution.

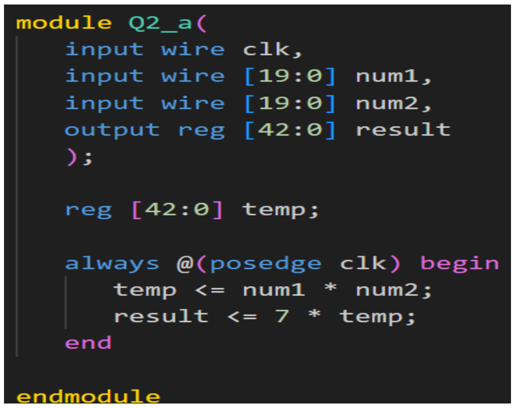
4. Parallel Processing:

- Pipelining enables parallel processing of multiple inputs, effectively increasing throughput.
- This can lead to a reduction in overall net delay as multiple operations can be executed concurrently, reducing the time taken to process each input.

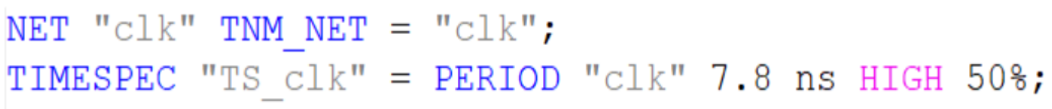
In summary, pipelining can significantly improve net delay by breaking down long paths into shorter segments, balancing timing constraints, improving clock distribution, and enabling parallel processing. By reducing net delay, pipelining helps in achieving higher clock frequencies and improving the overall performance of digital circuits.

سوال 2)

(a در این قسمت صرفا یک رجیستر temp تعریف کردم و مقدار حاصل ضرب را در ان ریختم و در مرحله بعد 7 برابر کردم. فرکانس کاری در اردر 128 MHz است.

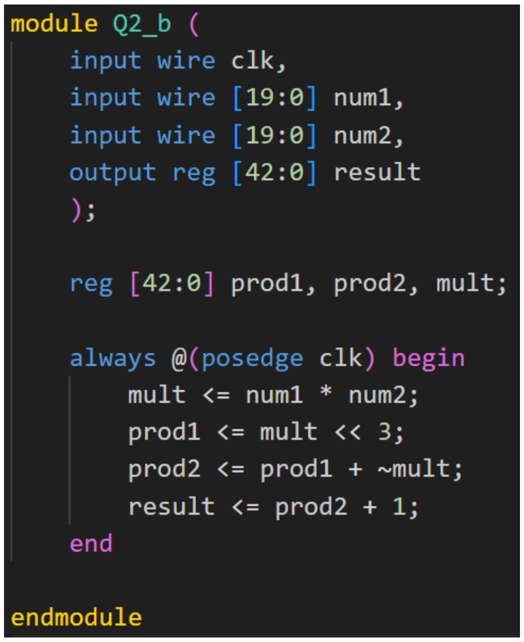


```
module Q2_a(
    input wire clk,
    input wire [19:0] num1,
    input wire [19:0] num2,
    output reg [42:0] result
    );

    reg [42:0] temp;

    always @(posedge clk) begin
        temp <= num1 * num2;
        result <= 7 * temp;
    end

endmodule
```

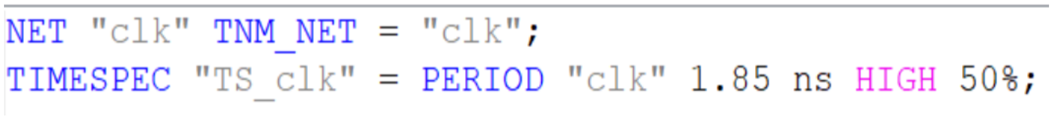


```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 7.8 ns HIGH 50%;
```

(b,c در این قسمت صرفا همان کد را پایپلاین کردم و به جای ضرب در 7 از شیفت 3 تایی و منها یه که به صورت نقیض به اضافه 1 است استفاده کردم. در عمل 4 لایه پایپلاین داریم. فرکانس کاری مدار در اردر 540MHz است چون در عمل دستورات بزرگ را به دستورات کوچکتر خورد کردم و بدیهی است فرکانس در این حالت بیشتر میشود. منابع مصرفی در حالت فوق برابر 134 رجیستر و 84 تا LUT و 105 فلیپ فلاپ و 81 تا IOB و 2 تا DSP48 است.

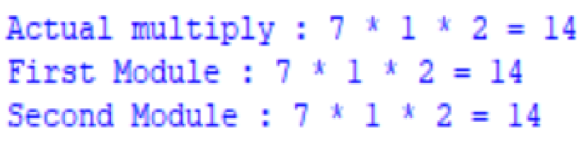


```
module Q2_b (
    input wire clk,
    input wire [19:0] num1,
    input wire [19:0] num2,
    output reg [42:0] result
    );

    reg [42:0] prod1, prod2, mult;

    always @(posedge clk) begin
        mult <= num1 * num2;
        prod1 <= mult << 3;
        prod2 <= prod1 + ~mult;
        result <= prod2 + 1;
    end

endmodule
```

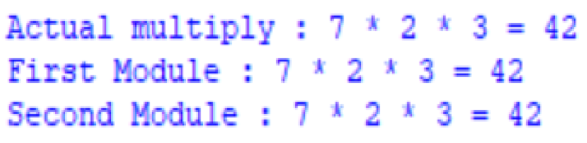


```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 1.85 ns HIGH 50%;
```

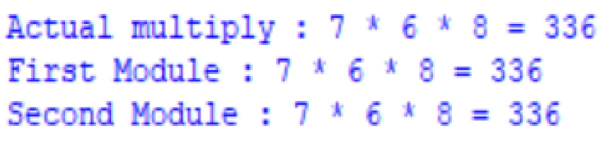
برای تست ماژول های مربوطه نیز تست بنچ نوشتم تا هر دو ماژول را با هم تست کند. همانطور که دیده میشود در هر ماژول به درستی عمل میکند.



```
Actual multiply : 7 * 1 * 2 = 14
First Module : 7 * 1 * 2 = 14
Second Module : 7 * 1 * 2 = 14
```



```
Actual multiply : 7 * 2 * 3 = 42
First Module : 7 * 2 * 3 = 42
Second Module : 7 * 2 * 3 = 42
```



```
Actual multiply : 7 * 6 * 8 = 336
First Module : 7 * 6 * 8 = 336
Second Module : 7 * 6 * 8 = 336
```

سوال 3)

a) Finite Impulse Response (FIR) filters are widely used in digital signal processing for various applications such as noise reduction, signal enhancement, and more. Two common FIR filter structures are the Direct Form and Transposed Form.

Direct Form FIR Filter:

1. Structure:
   - The direct form FIR filter is the most straightforward implementation.
   - It directly implements the convolution equation using the filter coefficients and input samples.

2. Equation:

   The output of a direct form FIR filter can be represented as:

$$y[n] = b_0\, x[n] + \; b_1\, x[n-1] + \cdots + \; b_{n-1}\, x[n-(N-1)]$$

   where:
   - y[n] is the output at time n,
   - x[n] is the input at time n,
   - b_0, b_1, ... , b_{N-1} are the filter coefficients,
   - N is the filter order.

3. Advantages:
   - Simple and straightforward to implement.
   - Easy to understand and analyze.

4. Disadvantages:
   - Requires a large number of adders and multipliers, especially for higher-order filters.
   - Not efficient in terms of hardware utilization.

Transposed Form FIR Filter:

1. Structure:
   - The transposed form FIR filter is a reordering of the direct form.
   - It rearranges the structure to reduce the number of adders and multipliers required.

2. Equation:

   The output of a transposed form FIR filter can be represented as:
   $y[n] = b_{n-1}\, x[n] + \; b_{n-2}\, x[n-1] + \cdots + \; b_0\, x[n-(N-1)]$

   where the coefficients are now in reverse order compared to the direct form.

3. Advantages:
   - Reduced hardware complexity: In the transposed form, the number of adders and multipliers is reduced because it exploits the symmetry of the coefficients.
   - Better suited for hardware implementation, as it reduces the critical path delay.

4. Disadvantages:
   - Slightly more complex mathematically than the direct form, but not significantly.
   - May be less intuitive to understand compared to the direct form for some.

Differences:

1. Implementation Complexity:
   - Direct Form: Simple to understand and implement but can be hardware-intensive.
   - Transposed Form: More complex conceptually but offers reduced hardware complexity.

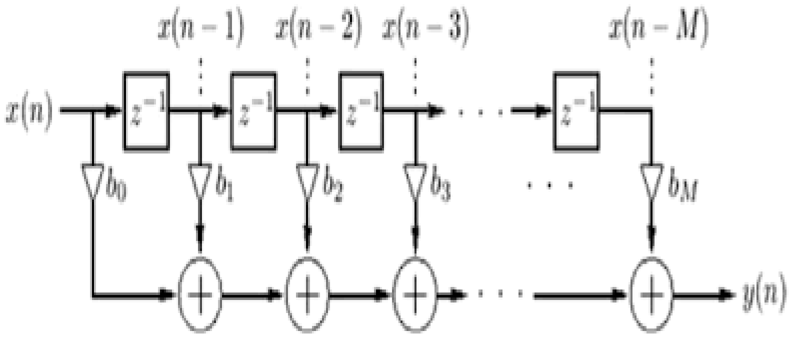
2. Hardware Efficiency:
   - Direct Form: Inefficient use of hardware due to a larger number of adders and multipliers.
   - Transposed Form: More efficient use of hardware due to the reduced number of operations.

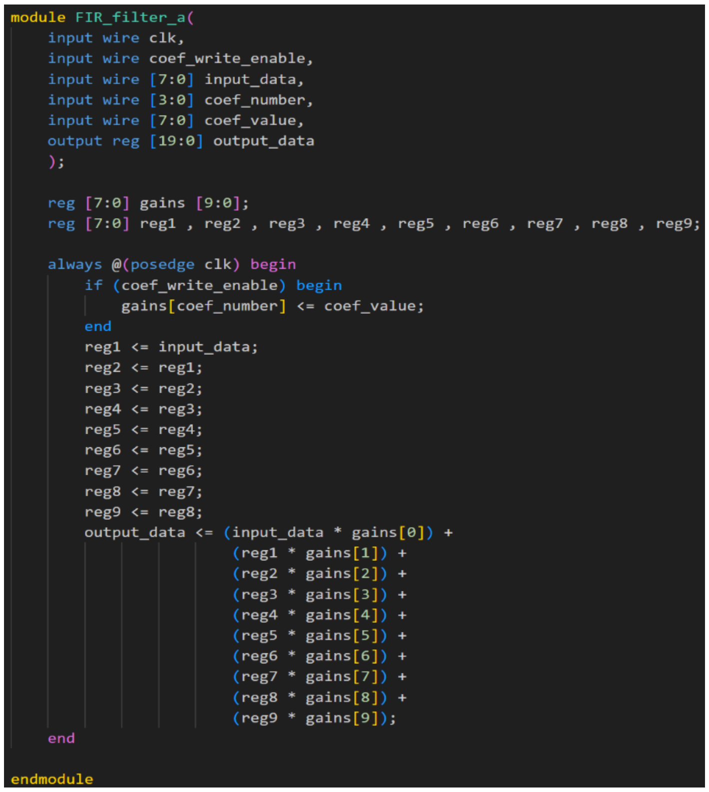
3. Coefficient Order:
   - Direct Form: Coefficients are in the same order as the original filter design.
   - Transposed Form: Coefficients are in reverse order compared to the original filter design.

4. Critical Path Delay:
   - Direct Form: Typically has a longer critical path delay due to its structure.
   - Transposed Form: Reduced critical path delay, making it more suitable for applications with strict timing requirements.

In summary, the direct form FIR filter is simpler to understand but may not be the most efficient in terms of hardware usage. The transposed form FIR filter, while slightly more complex conceptually, offers reduced hardware complexity and is often preferred for hardware implementations where efficiency and timing considerations are important.

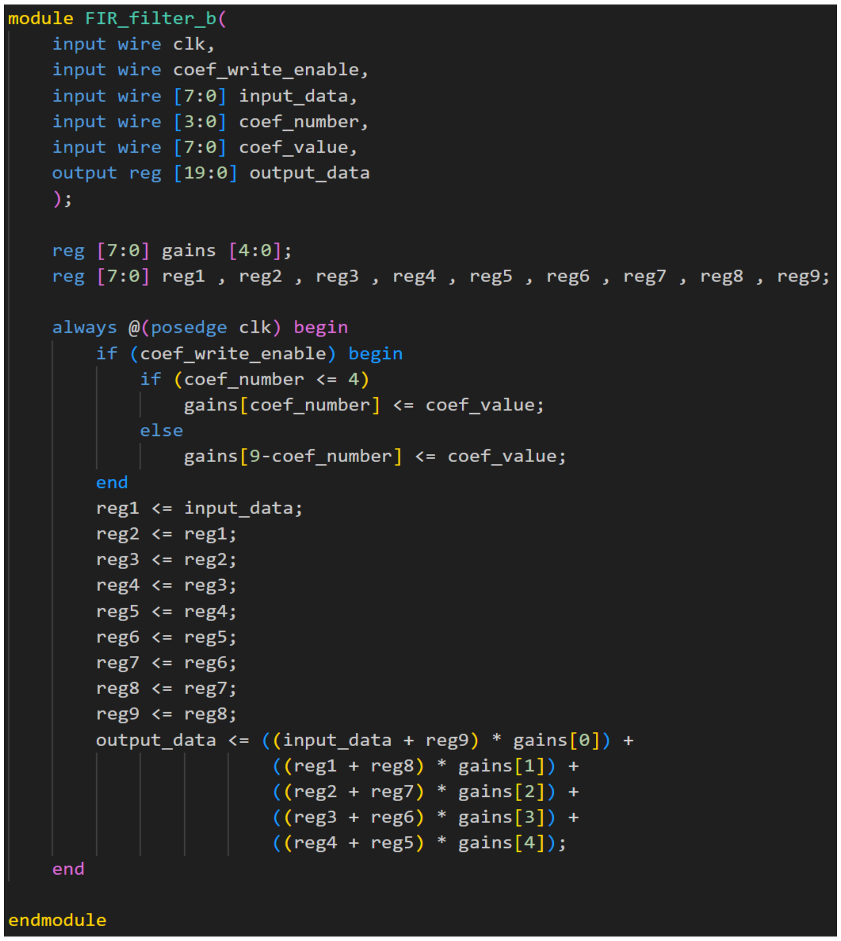
شمای مدار مورد نظر به صورت زیر است که چون مدار باید دارای 10 تپ باشد باید 10 تا رجیستر 8 بیتی برای ضرایب و 9 تا رجیستر برای ذخیره داده های قبلی گذاشت و در هر مرحله داده ها را شیفت داد و ضرایب را با سیگنال های کنترلی تغییر داد.



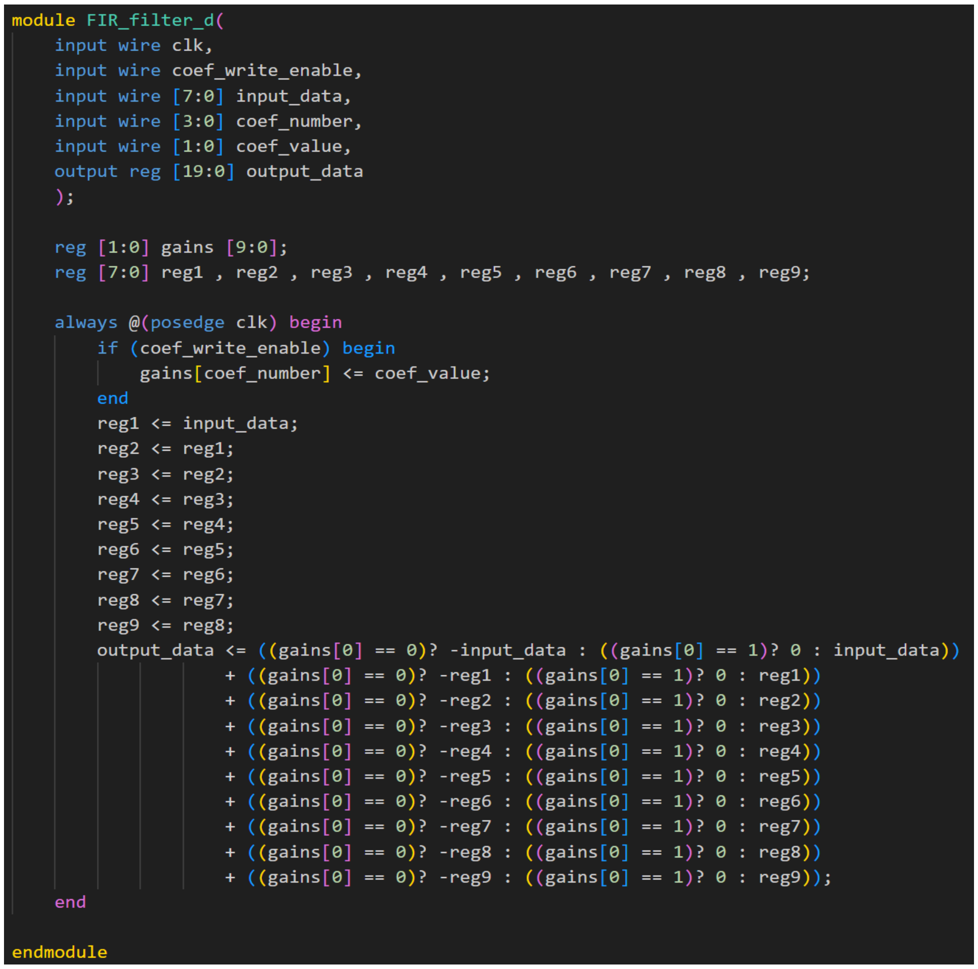


```
module FIR_filter_a(
    input wire clk,
    input wire coef_write_enable,
    input wire [7:0] input_data,
    input wire [3:0] coef_number,
    input wire [7:0] coef_value,
    output reg [19:0] output_data
    );

    reg [7:0] gains [9:0];
    reg [7:0] reg1 , reg2 , reg3 , reg4 , reg5 , reg6 , reg7 , reg8 , reg9;

    always @(posedge clk) begin
        if (coef_write_enable) begin
            gains[coef_number] <= coef_value;
        end
        reg1 <= input_data;
        reg2 <= reg1;
        reg3 <= reg2;
        reg4 <= reg3;
        reg5 <= reg4;
        reg6 <= reg5;
        reg7 <= reg6;
        reg8 <= reg7;
        reg9 <= reg8;
        output_data <= (input_data * gains[0]) +
                        (reg1 * gains[1]) +
                        (reg2 * gains[2]) +
                        (reg3 * gains[3]) +
                        (reg4 * gains[4]) +
                        (reg5 * gains[5]) +
                        (reg6 * gains[6]) +
                        (reg7 * gains[7]) +
                        (reg8 * gains[8]) +
                        (reg9 * gains[9]);
    end

endmodule
```

b,c) در این قسمت چون ضرایب symmetric هستند که یعنی متقارن هستند و همان مقدار ضریب در ضریب متقارنش است پس میتوان تعداد ضرایب را نصف کرد و ضریب نیمه دوم را به ضریب نیمه اول اشاره داد که با این روش مقدار رجیستر های استفاده شده برای ذخیره ضرایب کم میشود و مساحت کمتری اشغال میکند ولی در عوض کمی مدار پیچیده تر میشود.



```
module FIR_filter_b(
    input wire clk,
    input wire coef_write_enable,
    input wire [7:0] input_data,
    input wire [3:0] coef_number,
    input wire [7:0] coef_value,
    output reg [19:0] output_data
    );

    reg [7:0] gains [4:0];
    reg [7:0] reg1 , reg2 , reg3 , reg4 , reg5 , reg6 , reg7 , reg8 , reg9;

    always @(posedge clk) begin
        if (coef_write_enable) begin
            if (coef_number <= 4)
                gains[coef_number] <= coef_value;
            else
                gains[9-coef_number] <= coef_value;
        end
        reg1 <= input_data;
        reg2 <= reg1;
        reg3 <= reg2;
        reg4 <= reg3;
        reg5 <= reg4;
        reg6 <= reg5;
        reg7 <= reg6;
        reg8 <= reg7;
        reg9 <= reg8;
        output_data <= ((input_data + reg9) * gains[0]) +
                        ((reg1 + reg8) * gains[1]) +
                        ((reg2 + reg7) * gains[2]) +
                        ((reg3 + reg6) * gains[3]) +
                        ((reg4 + reg5) * gains[4]);
    end

endmodule
```



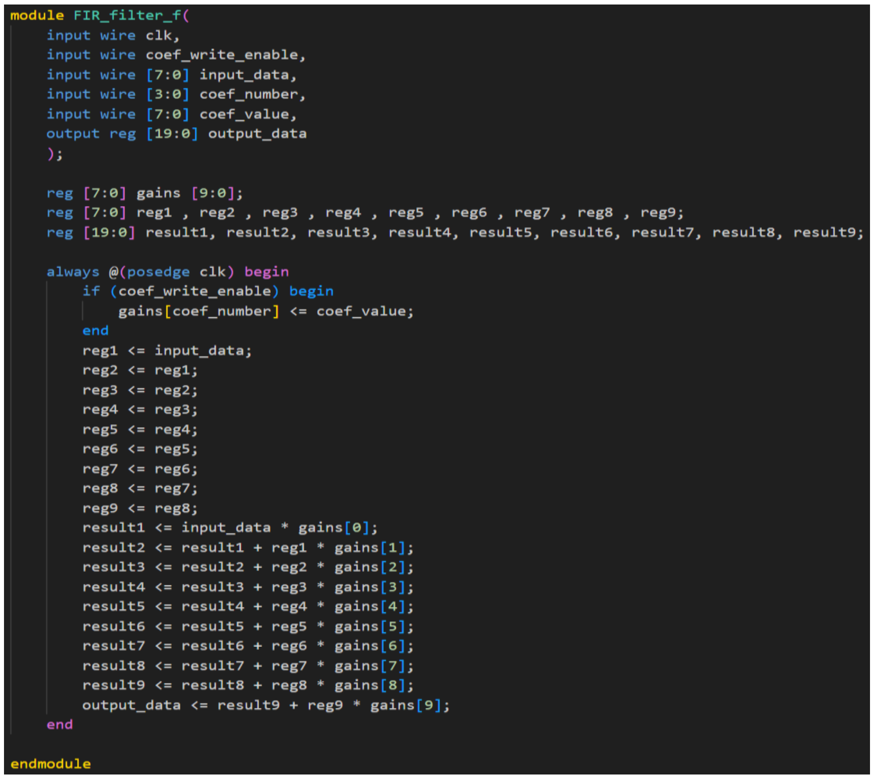
```
module FIR_filter_d(
    input wire clk,
    input wire coef_write_enable,
    input wire [7:0] input_data,
    input wire [3:0] coef_number,
    input wire [1:0] coef_value,
    output reg [19:0] output_data
    );

    reg [1:0] gains [9:0];
    reg [7:0] reg1 , reg2 , reg3 , reg4 , reg5 , reg6 , reg7 , reg8 , reg9;

    always @(posedge clk) begin
        if (coef_write_enable) begin
            gains[coef_number] <= coef_value;
        end
        reg1 <= input_data;
        reg2 <= reg1;
        reg3 <= reg2;
        reg4 <= reg3;
        reg5 <= reg4;
        reg6 <= reg5;
        reg7 <= reg6;
        reg8 <= reg7;
        reg9 <= reg8;
        output_data <= ((gains[0] == 0)? -input_data : ((gains[0] == 1)? 0 : input_data))
                     + ((gains[0] == 0)? -reg1 : ((gains[0] == 1)? 0 : reg1))
                     + ((gains[0] == 0)? -reg2 : ((gains[0] == 1)? 0 : reg2))
                     + ((gains[0] == 0)? -reg3 : ((gains[0] == 1)? 0 : reg3))
                     + ((gains[0] == 0)? -reg4 : ((gains[0] == 1)? 0 : reg4))
                     + ((gains[0] == 0)? -reg5 : ((gains[0] == 1)? 0 : reg5))
                     + ((gains[0] == 0)? -reg6 : ((gains[0] == 1)? 0 : reg6))
                     + ((gains[0] == 0)? -reg7 : ((gains[0] == 1)? 0 : reg7))
                     + ((gains[0] == 0)? -reg8 : ((gains[0] == 1)? 0 : reg8))
                     + ((gains[0] == 0)? -reg9 : ((gains[0] == 1)? 0 : reg9));
    end

endmodule
```

d) در این قسمت تعداد ضرایب ثابت و همان 10 تا است ولی عمق انها کمتر شده و میتوان با 2 بیت انها را هندل کرد برای مثال اگر 0 بود ضریب 1- و اگر 1 بود 0 و در غیر این صورت 1 بگیریم ضرایب را که باعث میشود رجیستر ها به شدت کاهش یابند ولی در عوض جمع انها پیچیده تر میشود.

e) در این قسمت با توجه به اینکه باید در هر مرحله 10 تا ضرب 8 بیتی و 10 تا جمع 16 بیتی انجام دهد که به نوبه خود زمان زیادی میبردبه همین دلیل فرکانس کاری ما در اردر 57.8 MHz میشود. (17.3 ns)

از 136 تا رجیستر و 112 تا LUT و 144 تا فلیپ فلاپ و 42 تا IOB و 10 تا DSP48 استفاده شده است به دلیل 10 تا ضرب.

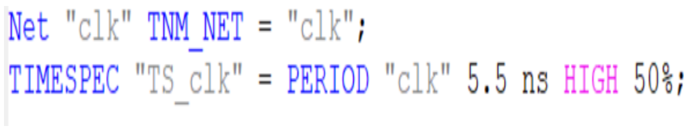
```
Net "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 17.3 ns HIGH 50%;
```



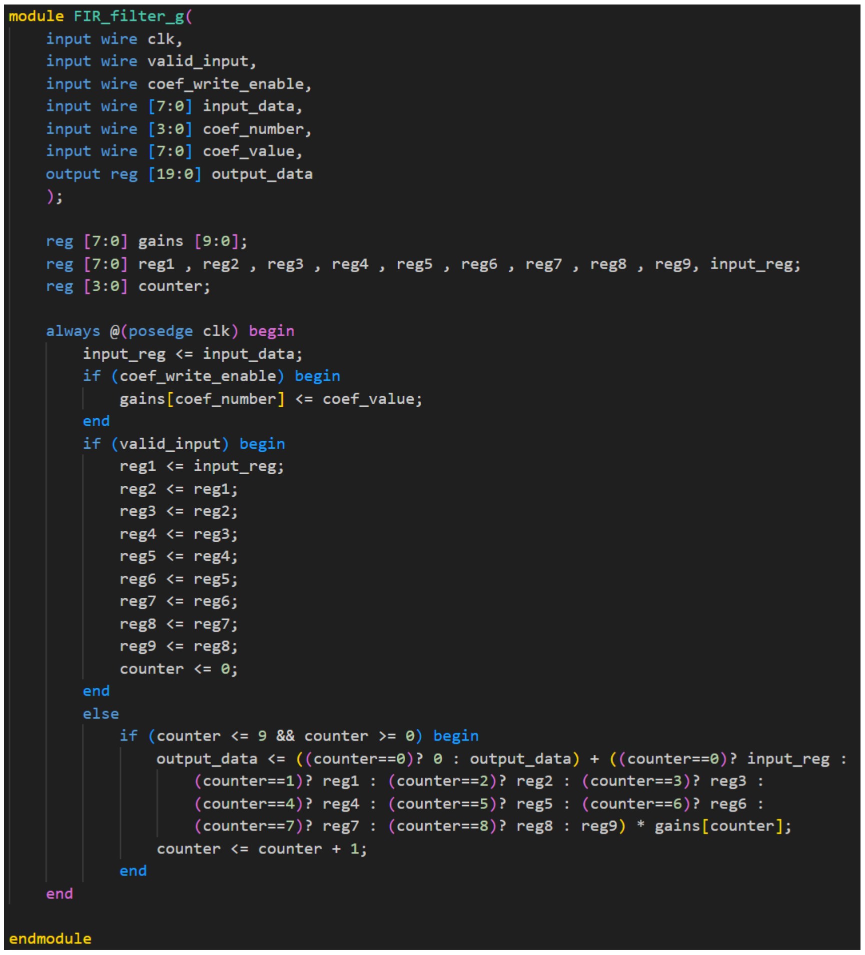
```
module FIR_filter_f(
    input wire clk,
    input wire coef_write_enable,
    input wire [7:0] input_data,
    input wire [3:0] coef_number,
    input wire [7:0] coef_value,
    output reg [19:0] output_data
    );

    reg [7:0] gains [9:0];
    reg [7:0] reg1 , reg2 , reg3 , reg4 , reg5 , reg6 , reg7 , reg8 , reg9;
    reg [19:0] result1, result2, result3, result4, result5, result6, result7, result8, result9;

    always @(posedge clk) begin
        if (coef_write_enable) begin
            gains[coef_number] <= coef_value;
        end
        reg1 <= input_data;
        reg2 <= reg1;
        reg3 <= reg2;
        reg4 <= reg3;
        reg5 <= reg4;
        reg6 <= reg5;
        reg7 <= reg6;
        reg8 <= reg7;
        reg9 <= reg8;
        result1 <= input_data * gains[0];
        result2 <= result1 + reg1 * gains[1];
        result3 <= result2 + reg2 * gains[2];
        result4 <= result3 + reg3 * gains[3];
        result5 <= result4 + reg4 * gains[4];
        result6 <= result5 + reg5 * gains[5];
        result7 <= result6 + reg6 * gains[6];
        result8 <= result7 + reg7 * gains[7];
        result9 <= result8 + reg8 * gains[8];
        output_data <= result9 + reg9 * gains[9];
    end

endmodule
```

f) در این قسمت برای پایپلاین کردن مدار باید جمع و ضرب خروجی را که تعداد زیادی دارد به جمع و ضرب های کوچک تر تقسیم کرد تا دیلی هر قسمت کم شود که به طبع باعث میشود خروجی در چند کلاک اماده شود(در این مثال 10 کلاک) ولی در عوض فرکانس کلاک بیشتر میشود.

همانطور که دیده میشود فرکانس برابر 181.8MHz است.(5.5ns)



```
Net "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 5.5 ns HIGH 50%;
```

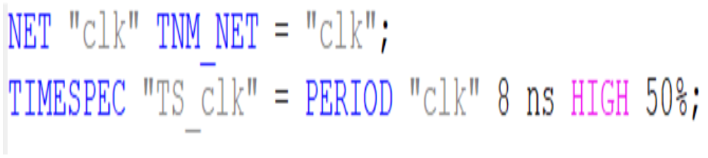


```
module FIR_filter_g(
    input wire clk,
    input wire valid_input,
    input wire coef_write_enable,
    input wire [7:0] input_data,
    input wire [3:0] coef_number,
    input wire [7:0] coef_value,
    output reg [19:0] output_data
    );

    reg [7:0] gains [9:0];
    reg [7:0] reg1 , reg2 , reg3 , reg4 , reg5 , reg6 , reg7 , reg8 , reg9, input_reg;
    reg [3:0] counter;

    always @(posedge clk) begin
        input_reg <= input_data;
        if (coef_write_enable) begin
            gains[coef_number] <= coef_value;
        end
        if (valid_input) begin
            reg1 <= input_reg;
            reg2 <= reg1;
            reg3 <= reg2;
            reg4 <= reg3;
            reg5 <= reg4;
            reg6 <= reg5;
            reg7 <= reg6;
            reg8 <= reg7;
            reg9 <= reg8;
            counter <= 0;
        end
        else
            if (counter <= 9 && counter >= 0) begin
                output_data <= ((counter==0)? 0 : output_data) + ((counter==0)? input_reg :
                    (counter==1)? reg1 : (counter==2)? reg2 : (counter==3)? reg3 :
                    (counter==4)? reg4 : (counter==5)? reg5 : (counter==6)? reg6 :
                    (counter==7)? reg7 : (counter==8)? reg8 : reg9) * gains[counter];
                counter <= counter + 1;
            end
    end

endmodule
```

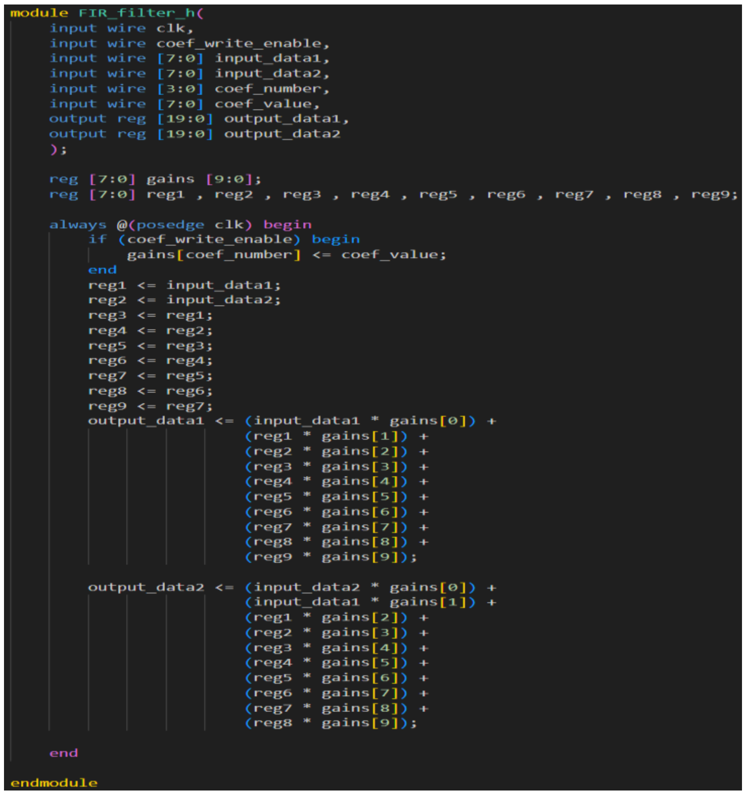
g) در این قسمت با انجام عمل resource sharing در هر کلاک از 1 جمع و 1 ضرب استفاده کردیم که به طبع نیاز به 10 کلاک برای اماده شدن جواب نیاز دارد.

با انجام این عمل رجیستر ها از 136 به 106 و LUT از 112 به 101 و فلیپ فلاپ ها از 144 به 131 و IOB از 42 به 43 و تعداد DSP48 ها از 10 به 2 تغییر کردند.

فرکانس کاری هم برابر 125MHz است.(8ns)



```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 8 ns HIGH 50%;
```

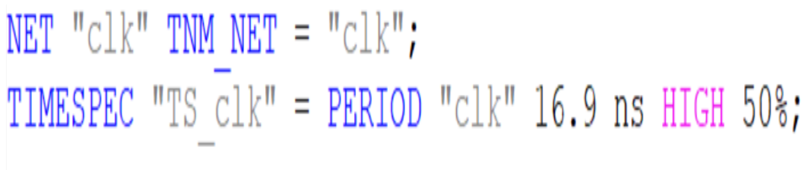


```
module FIR_filter_h(
    input wire clk,
    input wire coef_write_enable,
    input wire [7:0] input_data1,
    input wire [7:0] input_data2,
    input wire [3:0] coef_number,
    input wire [7:0] coef_value,
    output reg [19:0] output_data1,
    output reg [19:0] output_data2
);

    reg [7:0] gains [9:0];
    reg [7:0] reg1 , reg2 , reg3 , reg4 , reg5 , reg6 , reg7 , reg8 , reg9;

    always @(posedge clk) begin
        if (coef_write_enable) begin
            gains[coef_number] <= coef_value;
        end
        reg1 <= input_data1;
        reg2 <= input_data2;
        reg3 <= reg1;
        reg4 <= reg2;
        reg5 <= reg3;
        reg6 <= reg4;
        reg7 <= reg5;
        reg8 <= reg6;
        reg9 <= reg7;
        output_data1 <= (input_data1 * gains[0]) +
                        (reg1 * gains[1]) +
                        (reg2 * gains[2]) +
                        (reg3 * gains[3]) +
                        (reg4 * gains[4]) +
                        (reg5 * gains[5]) +
                        (reg6 * gains[6]) +
                        (reg7 * gains[7]) +
                        (reg8 * gains[8]) +
                        (reg9 * gains[9]);

        output_data2 <= (input_data2 * gains[0]) +
                        (input_data1 * gains[1]) +
                        (reg1 * gains[2]) +
                        (reg2 * gains[3]) +
                        (reg3 * gains[4]) +
                        (reg4 * gains[5]) +
                        (reg5 * gains[6]) +
                        (reg6 * gains[7]) +
                        (reg7 * gains[8]) +
                        (reg8 * gains[9]);

    end

endmodule
```

h) در این قسمت صرفا 2 تا ورودی و خروجی دادیم که به طبع باید 2 تا شیفت بدهیم.

در این قسمت تعداد رجیستر ها از 136 به 120 و LUT از 112 به 132 و فلیپ فلاپ ها از 144 به 148 و IOB از 42 به 70 و DSP48 ها از 10 به 20 تغییر کردند تا نرخ پردازش بیشتر شود.

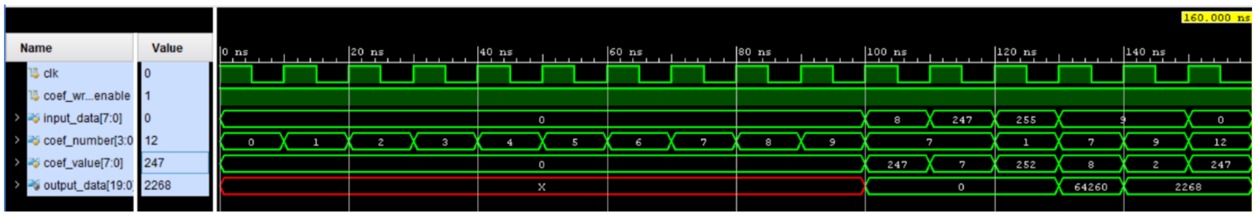
فرکانس کاری برابر 59.17MHz است. (16.9ns)



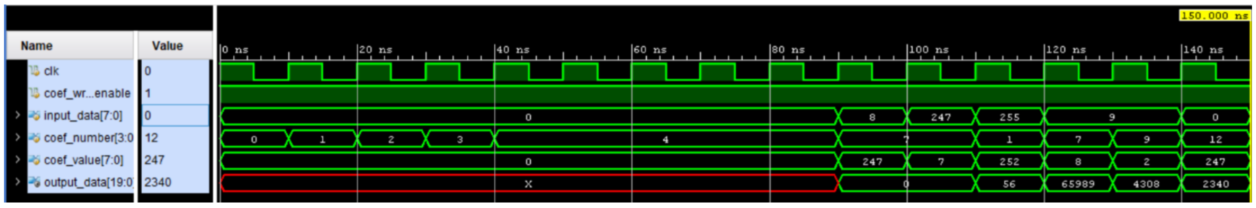
```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 16.9 ns HIGH 50%;
```

برای عملکرد درست کد ها هم Wave های انها به ترتیب نمایش داده شده است.

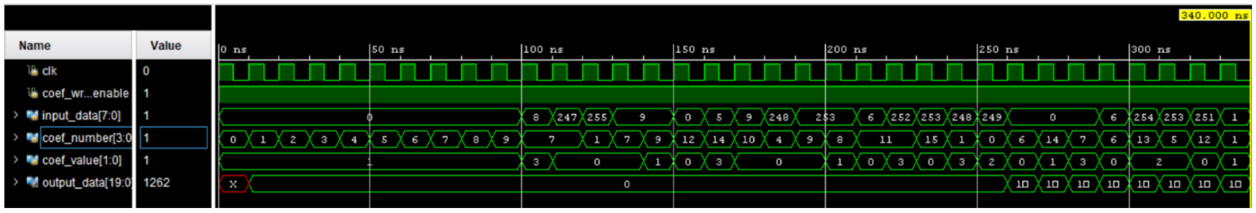
.a



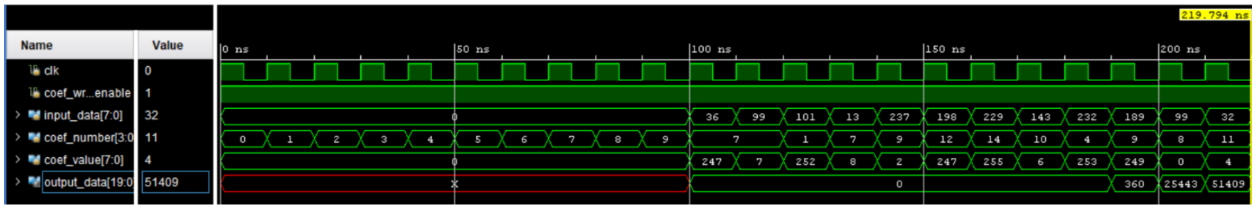
.b



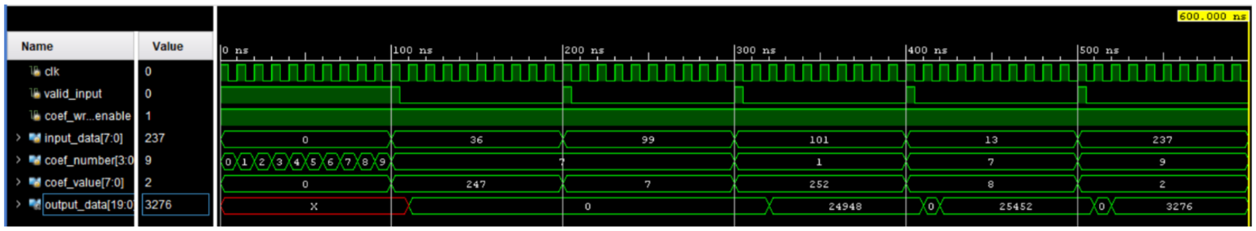
.d


340.000 ns
Name
Value
clk 0
coef_wr...enable 1
input_data[7:0] 1
coef_number[3:0] 1
coef_value[1:0] 1
output_data[19:0] 1262
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

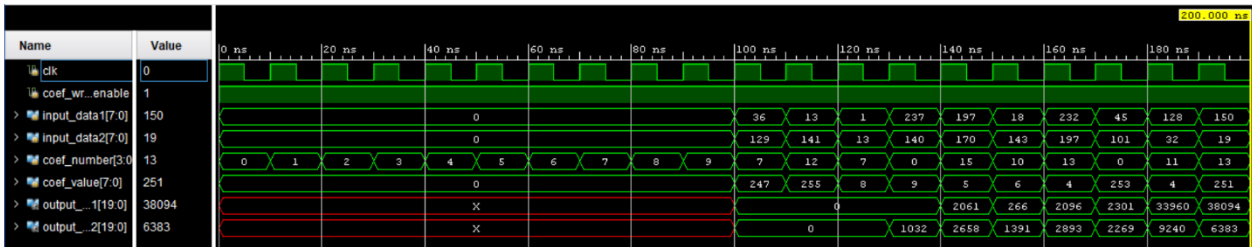
.f


219.794 ns
Name
Value
clk 0
coef_wr...enable 1
input_data[7:0] 32
coef_number[3:0] 11
coef_value[7:0] 4
output_data[19:0] 51409
0 ns
50 ns
100 ns
150 ns
200 ns

.g


600.000 ns
Name
Value
clk 0
valid_input 0
coef_wr...enable 1
input_data[7:0] 237
coef_number[3:0] 9
coef_value[7:0] 2
output_data[19:0] 3276
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns

.h


200.000 ns
Name
Value
clk 0
coef_wr...enable 1
input_data1[7:0] 150
input_data2[7:0] 19
coef_number[3:0] 13
coef_value[7:0] 251
output_...1[19:0] 38094
output_...2[19:0] 6383
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns