台科大

PA1

Computer Organization

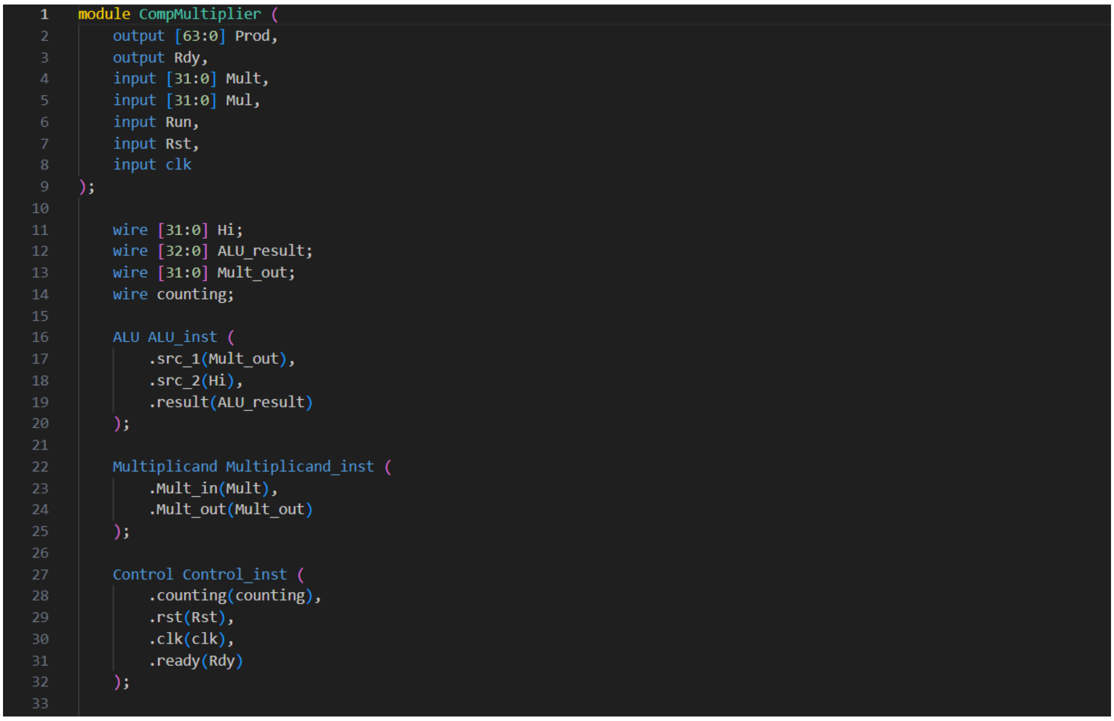
周 柏宇

2024/3/28

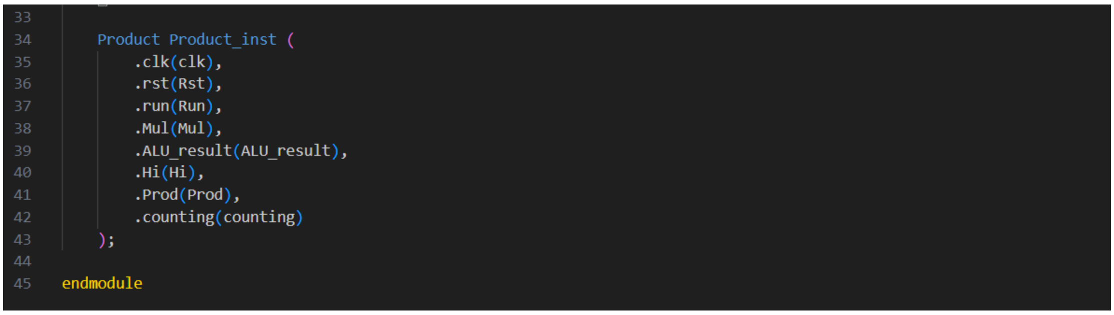
目錄

Multiplier

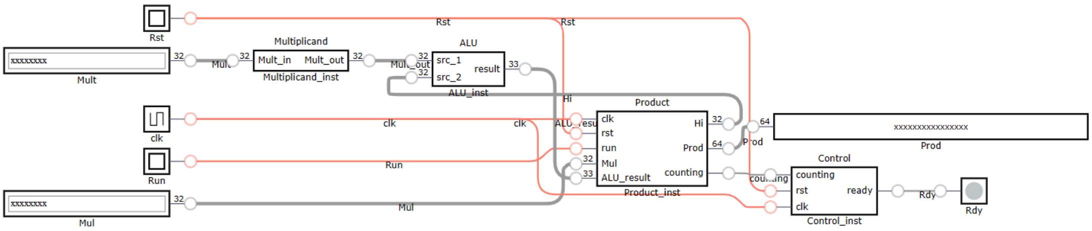
CompMultiplier



```
module CompMultiplier (
    output [63:0] Prod,
    output Rdy,
    input [31:0] Mult,
    input [31:0] Mul,
    input Run,
    input Rst,
    input clk
);

    wire [31:0] Hi;
    wire [32:0] ALU_result;
    wire [31:0] Mult_out;
    wire counting;

    ALU ALU_inst (
        .src_1(Mult_out),
        .src_2(Hi),
        .result(ALU_result)
    );

    Multiplicand Multiplicand_inst (
        .Mult_in(Mult),
        .Mult_out(Mult_out)
    );

    Control Control_inst (
        .counting(counting),
        .rst(Rst),
        .clk(clk),
        .ready(Rdy)
    );

```



```

    Product Product_inst (
        .clk(clk),
        .rst(Rst),
        .run(Run),
        .Mul(Mul),
        .ALU_result(ALU_result),
        .Hi(Hi),
        .Prod(Prod),
        .counting(counting)
    );

endmodule
```



Description

This module is to connect all the small components to form a functional multiplier.

Details

Line 1~9

I/O Interface.

Signal Symbol	Signal Name	Signal Description
Prod	64-bit calculation result	This signal is set as an output before the Rdy signal is activated to forbid the testbench to read the previous calculation result.
Rdy	completion signal	The "high" level represents the system has completed multiplication and maintained the result.
Mult	32-bit multiplicand	The signal is generated by the testbench and updated when the Rst signal is "high".
Mul	32-bit multiplier	The signal is generated by the testbench and updated when the Rst signal is "high".
Run	execution signal	The system performs multiplication as the Run signal is "high" level.
Rst	initialization signal	The "high" level indicates that the system is initialized before multiplication, and the output results are set to zero. This signal has the highest priority and is generated by the testbench.

clk	clock signal	Periodic square waves are generated by the testbench for synchronizing each signal with the drive system.

Line 11~14

Wires used in this module.

Wire Symbol	**Wire Name**	**Wire Description**
Hi	HI register	Upper 32 bits of the product. Used to pass data to ALU.
ALU_result	33 bits ALU result. This sees overflow as the 33th bit.	The computation result of ALU.
Mult_out	32-bit multiplicand	The signal is generated by the testbench and updated when the Rst signal is "high".
counting	Counting flag	Tell Control to start. This will avoid racing condition.

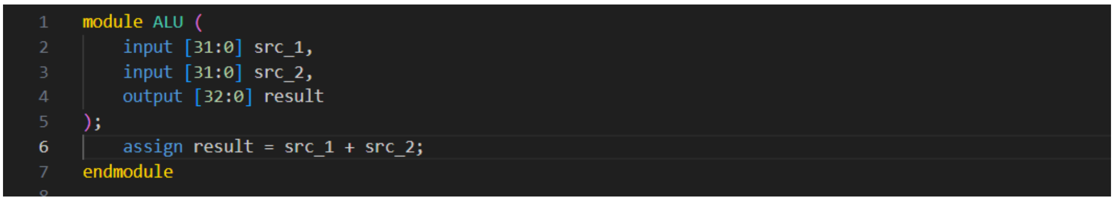
Line 16~43

Component instances.

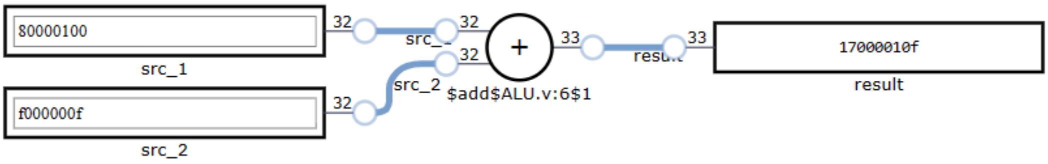
Instance Name	**Component Name**	**Instance Description**
ALU_inst	ALU instance	src_1 gets data from Mult_out, src_2 gets data from Hi, result is the output of this ALU, connected to ALU_result.
Multiplicand_inst	Multiplicand instance	Mult_in gets data from Mult, Mult_out sets output to Mult_out.
Control_inst	Control instance	counting gets data from Product, rst gets data from Rst, clk gets data from clk, ready sets output to Rdy.

Product_inst	Product instance	clk gets data from clk, rst gets data from Rst, run gets data from Run, Mul gets data from Mul on the posedge of Run, ALU_result gets data from ALU_result, Hi outputs data to Hi, Prod generates output to Prod. And last but not least it gives counting to Control.

ALU



```
1  module ALU (
2      input [31:0] src_1,
3      input [31:0] src_2,
4      output [32:0] result
5  );
6      assign result = src_1 + src_2;
7  endmodule
```



Description

This module provides basic arithmetic functions that fulfills multiplier’s needs.

Detail

Line 1~5

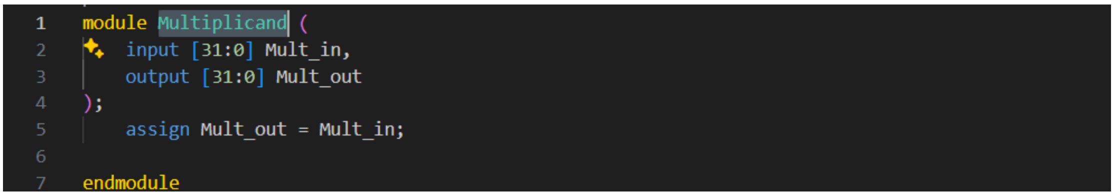
I/O Interface.

Signal Symbol	Signal Name	Signal Description
src_1	32-bits Source 1	addend
src_2	32-bits Source 2	augend
result	33-bits addition result	Sum with 1 overflow bit

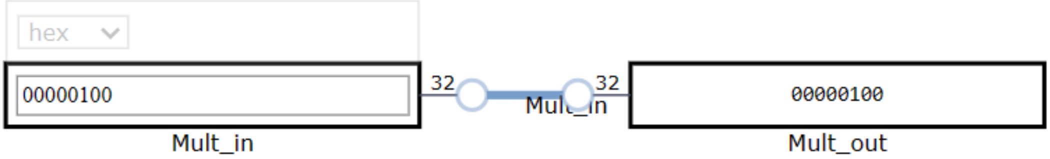
Line 6

Wire connection that represents result = src_1 + src_2.

Multiplicand



```
1  module Multiplicand (
2      input [31:0] Mult_in,
3      output [31:0] Mult_out
4  );
5      assign Mult_out = Mult_in;
6
7  endmodule
```



Description

Dummy module. Mult signal is given all along between the 2 posedge of Rst. There is no need to do any other operation.

Detail

Line 1~4

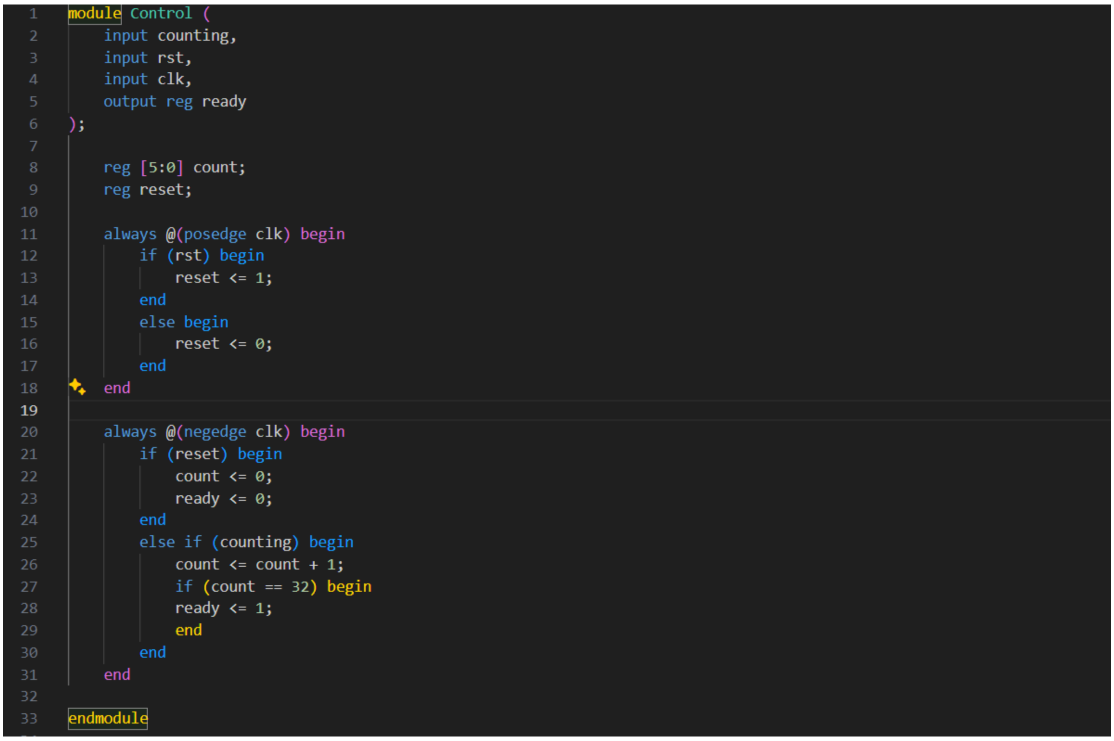
I/O Interface.

Signal Symbol	Signal Name	Signal Description
Mult_in	In direction of Mult	In
Mult_out	Out direction of Mult	out

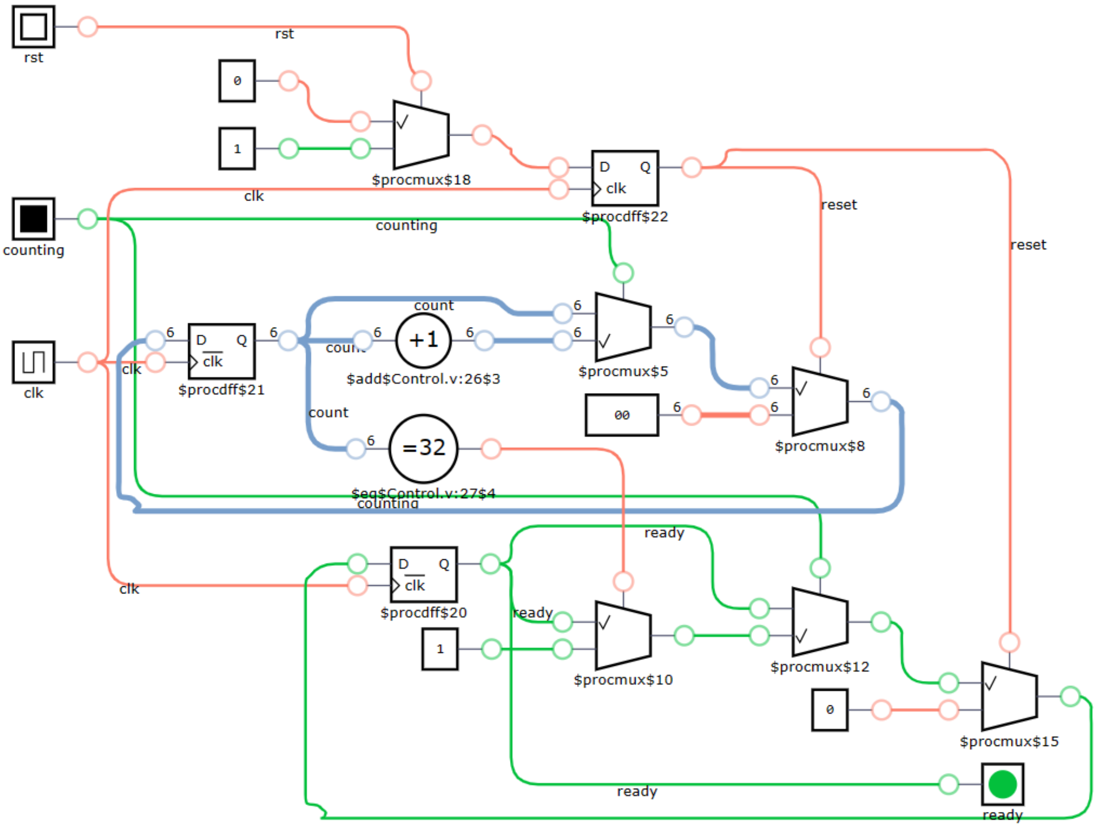
Line 5

Wire connection that denotes Mult_out = Mult_in.

Control



```verilog
module Control (
    input counting,
    input rst,
    input clk,
    output reg ready
);

    reg [5:0] count;
    reg reset;

    always @(posedge clk) begin
        if (rst) begin
            reset <= 1;
        end
        else begin
            reset <= 0;
        end
    end

    always @(negedge clk) begin
        if (reset) begin
            count <= 0;
            ready <= 0;
        end
        else if (counting) begin
            count <= count + 1;
            if (count == 32) begin
            ready <= 1;
            end
        end
    end

endmodule
```



Description

A counter with a flag.

Detail

Line 1~6

I/O Interface.

Signal Symbol	Signal Name	Signal Description
counting	Counting flag	The flag telling Control to count.
rst	initialization signal	The “high” level indicates that the system is initialized before multiplication, and the output results are set to zero. This signal has the highest priority and is generated by the testbench.
clk		Periodic square waves are generated by the testbench for synchronizing each signal with the drive system.
ready	Ready flag	Signify the end of the sequence of operation.

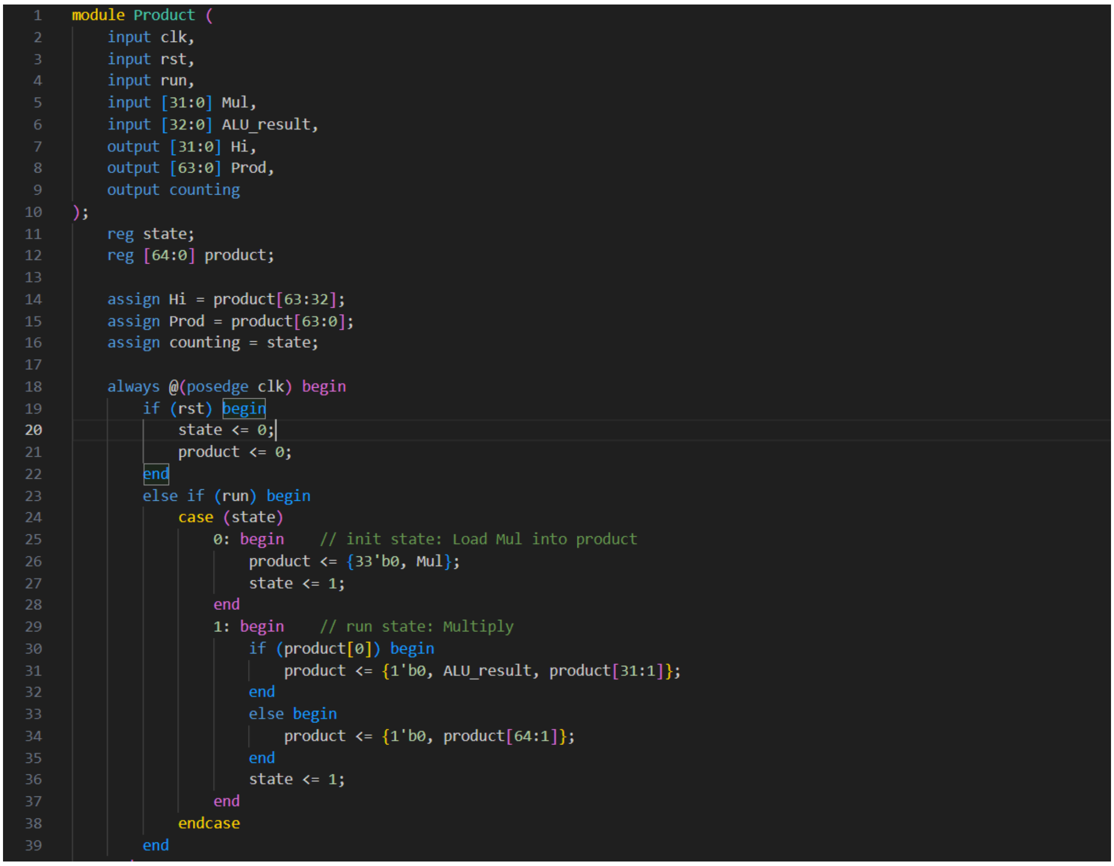
Line 11~18

Use reset register to hold rst signal between 2 posedge clk to avoid racing.

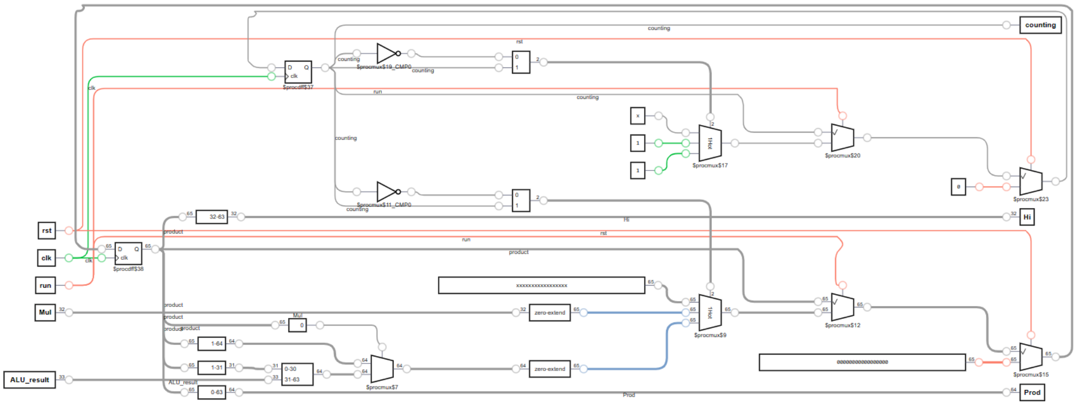
Line 20~31

When running count 32 negedges of clk then set ready flag to notify other component to stop or fetch data.

Product



```verilog
module Product (
    input clk,
    input rst,
    input run,
    input [31:0] Mul,
    input [32:0] ALU_result,
    output [31:0] Hi,
    output [63:0] Prod,
    output counting
);
    reg state;
    reg [64:0] product;

    assign Hi = product[63:32];
    assign Prod = product[63:0];
    assign counting = state;

    always @(posedge clk) begin
        if (rst) begin
            state <= 0;
            product <= 0;
        end
        else if (run) begin
            case (state)
                0: begin    // init state: Load Mul into product
                    product <= {33'b0, Mul};
                    state <= 1;
                end
                1: begin    // run state: Multiply
                    if (product[0]) begin
                        product <= {1'b0, ALU_result, product[31:1]};
                    end
                    else begin
                        product <= {1'b0, product[64:1]};
                    end
                    state <= 1;
                end
            endcase
        end
```



Description

Main module. I use 65 bits representing product in order to handle overflowing result from ALU. The computation is executed on posedge clk to avoid racing condition. And I use counting, which is a flag to tell Control to count. This flag will effective avoid racing.

Detail

Line 1~10

I/O Interface.

Signal Symbol	Signal Name	Signal Description
clk	clock	Clock
rst	reset	Reset the init state
run	Execution signal	Run operations when high
Mul	32-bits multiplicand	Init will put this to LO
ALU_result	33-bits ALU result	With 1 more carry bit
Hi	HI	Upper 32 bits
Prod	production	64-bits result
counting	Counting flag	Used to avoid racing

Line 11~12

Registers

Register Symbol	Register Name	Register Description
state	state	FSM state
product	65-bits product	1 more bit for overflow result generated by ALU.

Line 14~16

Wire connection of Hi, Prod and counting.

Line 19~22

Reset.

Line 24~38

FSM

State Symbol	State Name	State Description
0	Init	HI = 0, LO = Mul
1	run	Examine LSB to do predefined operations.

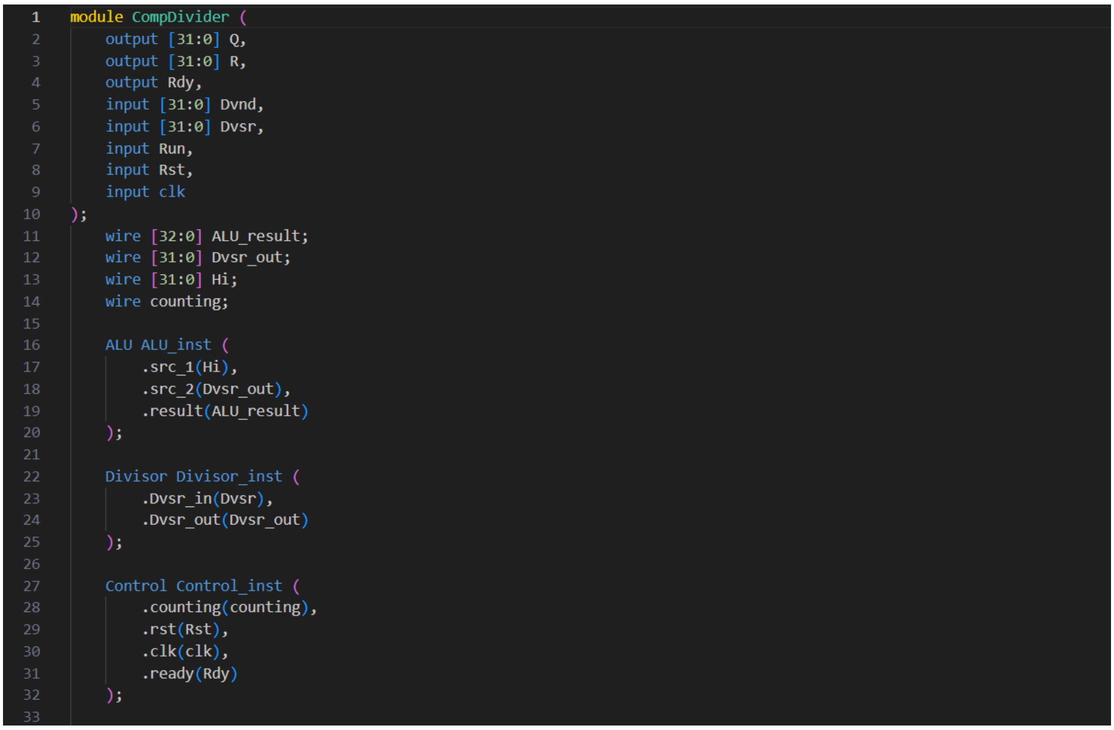
Line 30~36

If LSB is set, write ALU_result to HI and shift right 1 bit.

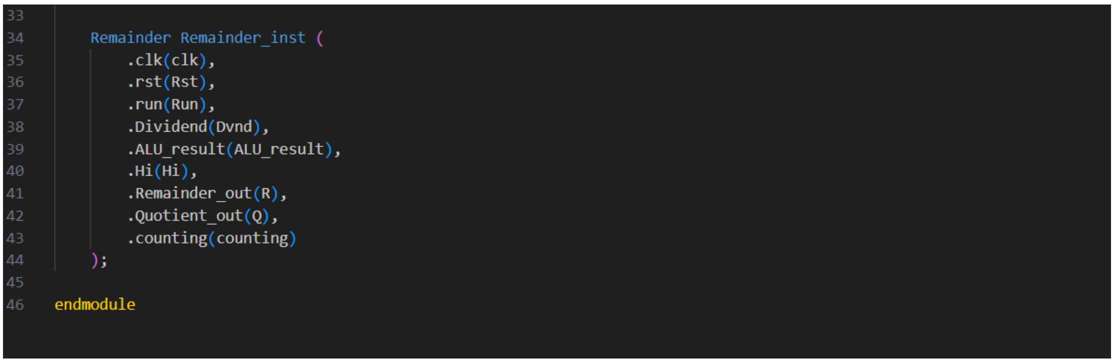
Else, shift right 1 bit.

Divider

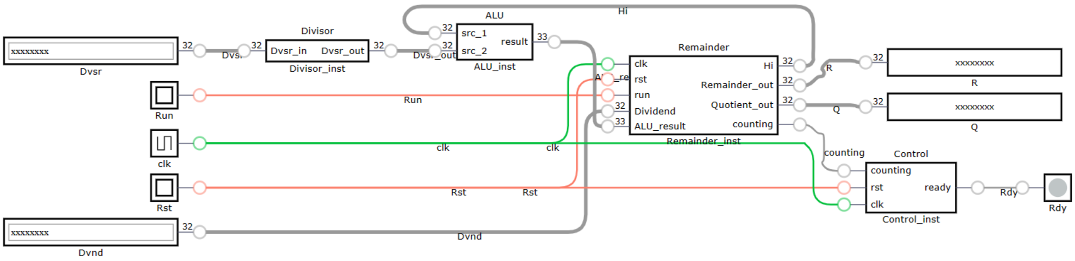
CompDivider



```
module CompDivider (
    output [31:0] Q,
    output [31:0] R,
    output Rdy,
    input [31:0] Dvnd,
    input [31:0] Dvsr,
    input Run,
    input Rst,
    input clk
);
    wire [32:0] ALU_result;
    wire [31:0] Dvsr_out;
    wire [31:0] Hi;
    wire counting;

    ALU ALU_inst (
        .src_1(Hi),
        .src_2(Dvsr_out),
        .result(ALU_result)
    );

    Divisor Divisor_inst (
        .Dvsr_in(Dvsr),
        .Dvsr_out(Dvsr_out)
    );

    Control Control_inst (
        .counting(counting),
        .rst(Rst),
        .clk(clk),
        .ready(Rdy)
    );

```



```

    Remainder Remainder_inst (
        .clk(clk),
        .rst(Rst),
        .run(Run),
        .Dividend(Dvnd),
        .ALU_result(ALU_result),
        .Hi(Hi),
        .Remainder_out(R),
        .Quotient_out(Q),
        .counting(counting)
    );

endmodule
```



Description

This module is to connect all the small components to form a functional divider.

Detail

Line 1~10

I/O Interface.

Signal Symbol	Signal Name	Signal Description
Q	32-bit Quotient	Quotient result
R	32-bit Reminder	Remainder result
Rdy	completion signal	Iff it's set the, result is valid
Dvnd	32-bit Dividend	Input
Dvsr	32-bit Divisor	Input
Run	execution signal	Start execute when high
Rst	initialization signal	Reset
clk	clock signal	clock

Line 11~14

Wires used in this module.

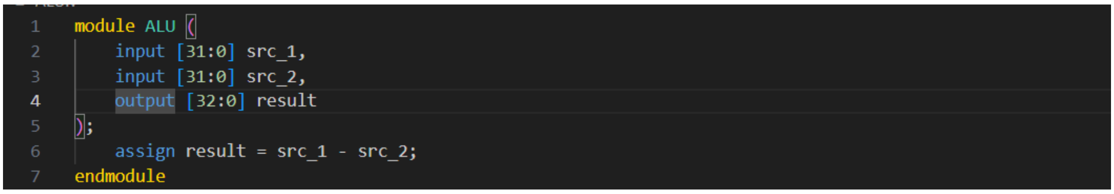
Wire Symbol	Wire Name	Wire Description
ALU_result	33-bits ALU out	With 1 carry bit
Dvsr_out	Divisor out	Dummy connection
Hi	HI register	Upper 32 bits
counting	Counting flag	Used to avoid racing.

Line 16~44

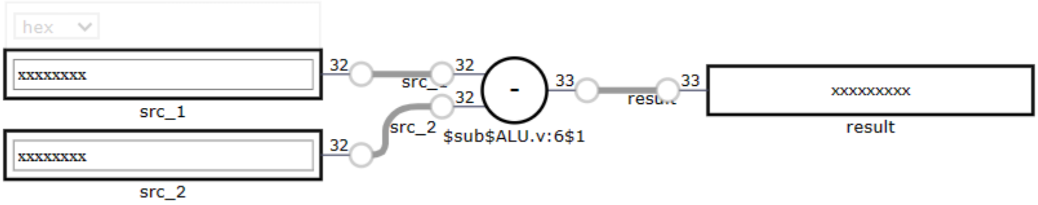
Component instances.

Instance Symbol	Instance Name	Instance Description
ALU_inst	ALU instance	Output to ALU_result considering R, Dvsr_out
Divisor_inst	Divisor instance	Dummy
Control_inst	Control instance	Counter with Rdy flag.
Remainder_inst	Remainder instance	1 clk, 2 flags, 2 in, 4 out. For more detail click me.

ALU



```
1  module ALU (
2      input [31:0] src_1,
3      input [31:0] src_2,
4      output [32:0] result
5  );
6      assign result = src_1 - src_2;
7  endmodule
```



Description

This module provides basic arithmetic functions that fulfills divider’s needs.

Detail

Line 1~5

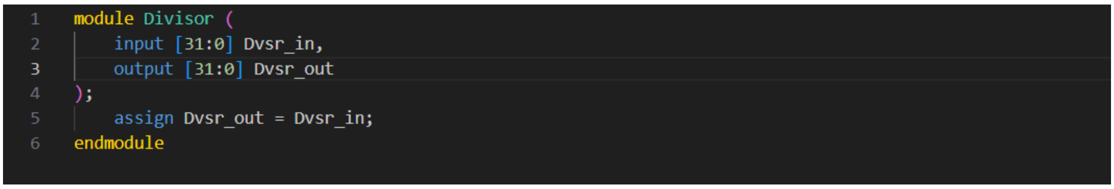
I/O Interface.

Signal Symbol	Signal Name	Signal Description
src_1	32-bits source 1	Minuend
src_2	32-bits source 2	subtrahend
result	33-bits result	Difference, 1 more borrow bit

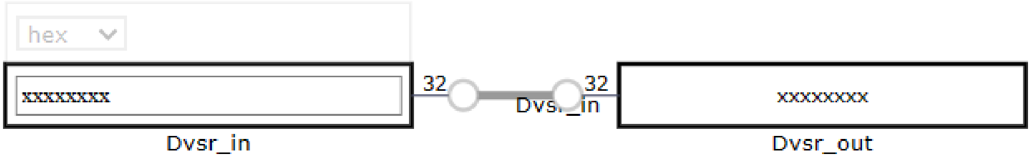
Line 6

Wire connection symbolize result = src_1 – src2.

Divisor



```
module Divisor (
    input [31:0] Dvsr_in,
    output [31:0] Dvsr_out
);
    assign Dvsr_out = Dvsr_in;
endmodule
```



Description

Dummy

Detail

Line 1~4

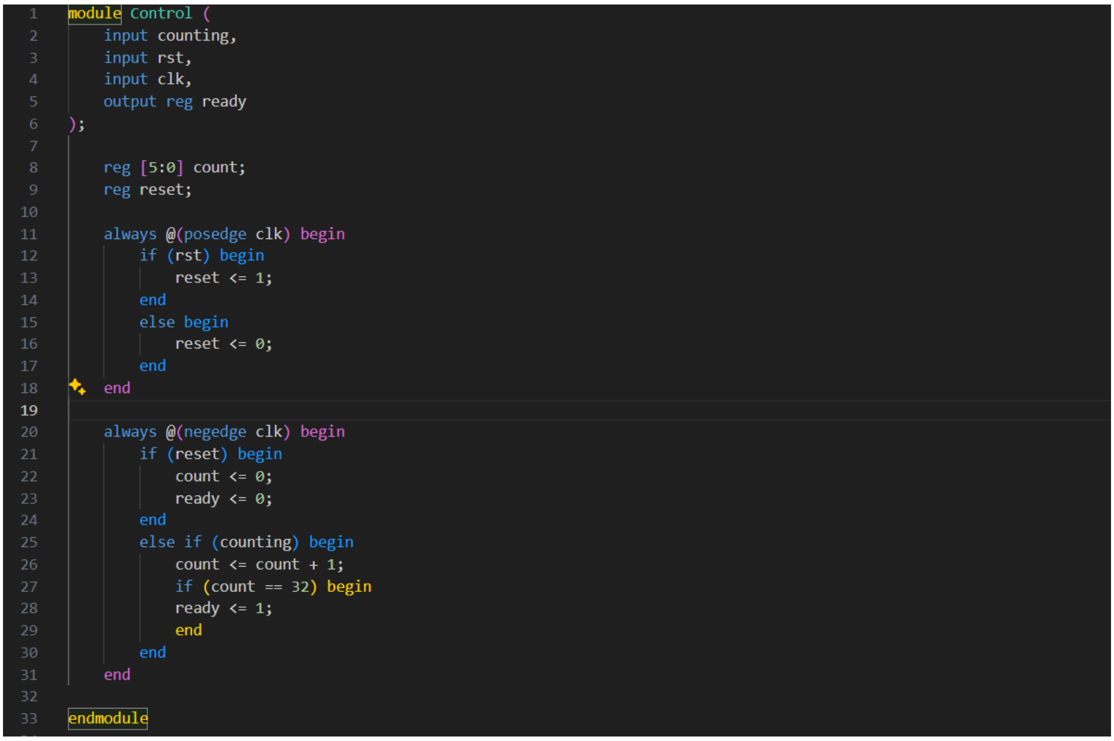
I/O Interface.

Signal Symbol	Signal Name	Signal Description
Dvsr_in	Divisor in	In
Dvsr_out	Divisor out	out

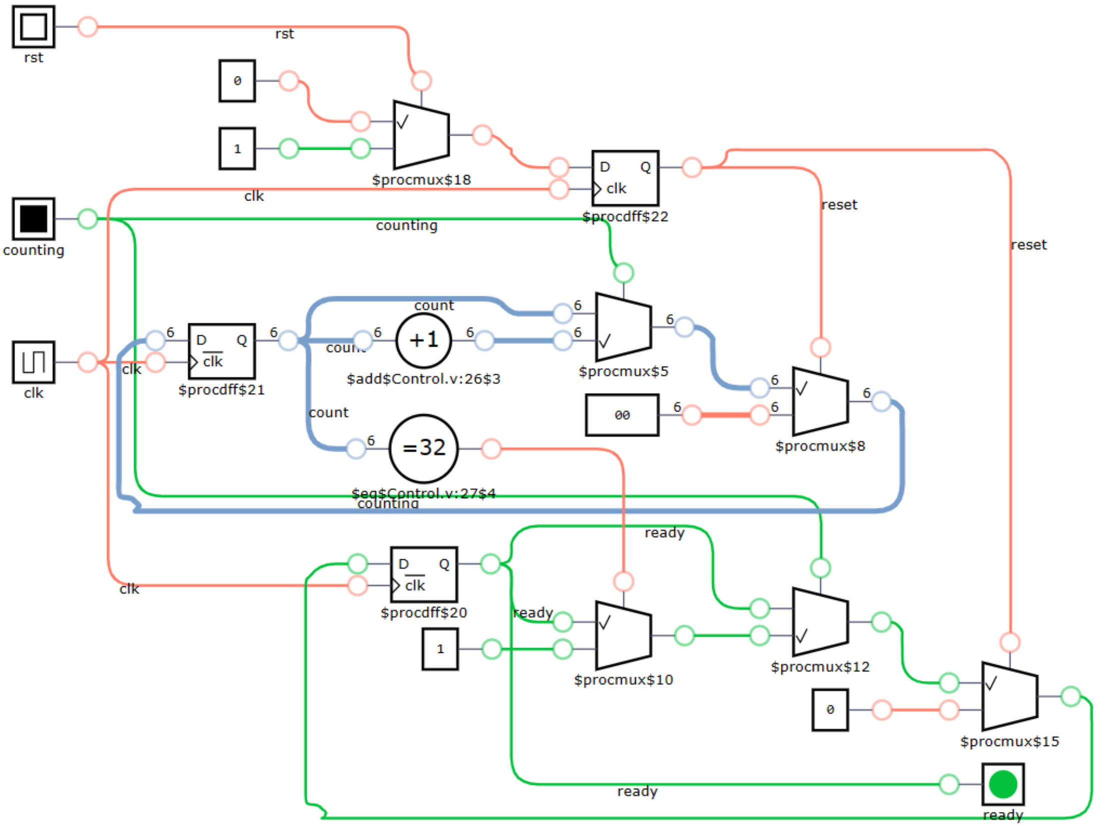
Line 5

Wire connection to make out = in.

Control



```
module Control (
    input counting,
    input rst,
    input clk,
    output reg ready
);

    reg [5:0] count;
    reg reset;

    always @(posedge clk) begin
        if (rst) begin
            reset <= 1;
        end
        else begin
            reset <= 0;
        end
    end

    always @(negedge clk) begin
        if (reset) begin
            count <= 0;
            ready <= 0;
        end
        else if (counting) begin
            count <= count + 1;
            if (count == 32) begin
            ready <= 1;
            end
        end
    end

endmodule
```



Description

A counter with 2 in flags, 1 out flag.

Detail

Line 1~6

I/O Interface.

Signal Symbol	Signal Name	Signal Description
counting	Counting flag	Input flag represent start
rst	Reset flag	Input flag represent reset
clk	Clock	Clock
ready	Ready flag	Signify the end of operation.

Line 8~9

Registers

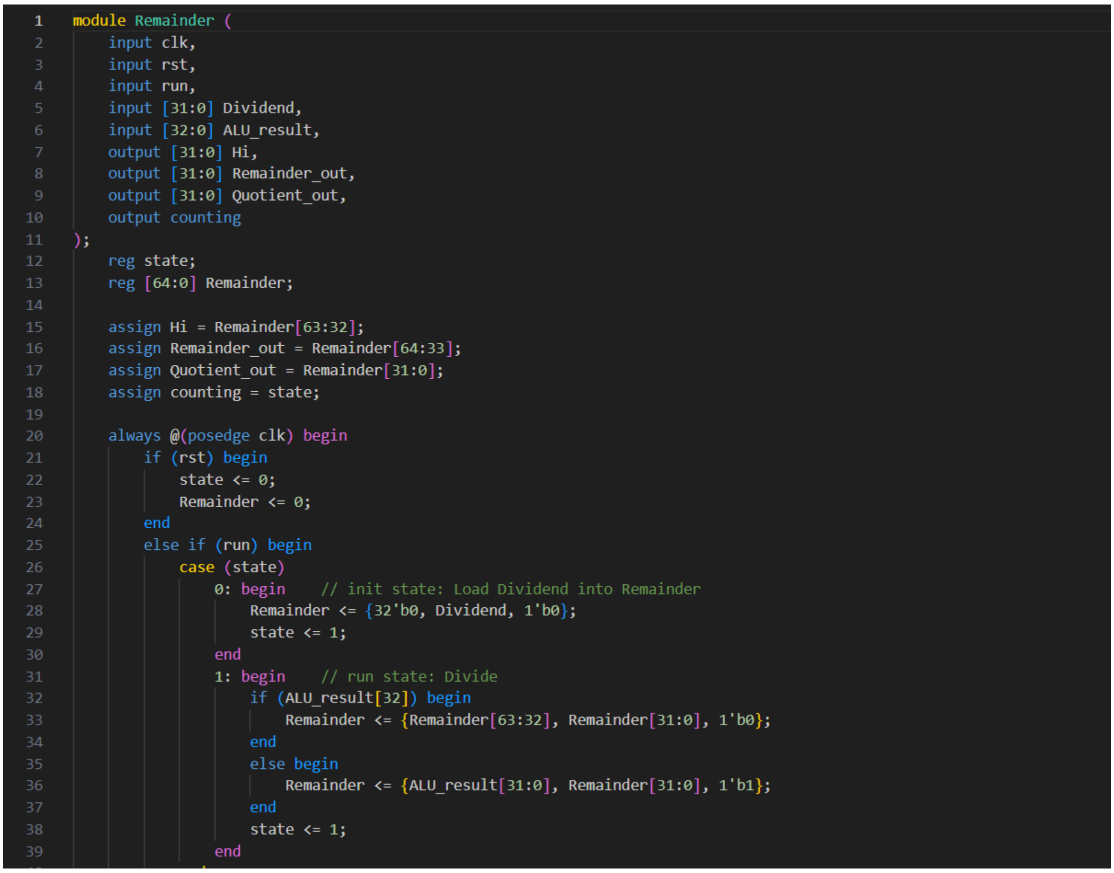
Line 11~18

Use reset register to hold rst signal between 2 posedge clk to avoid racing.

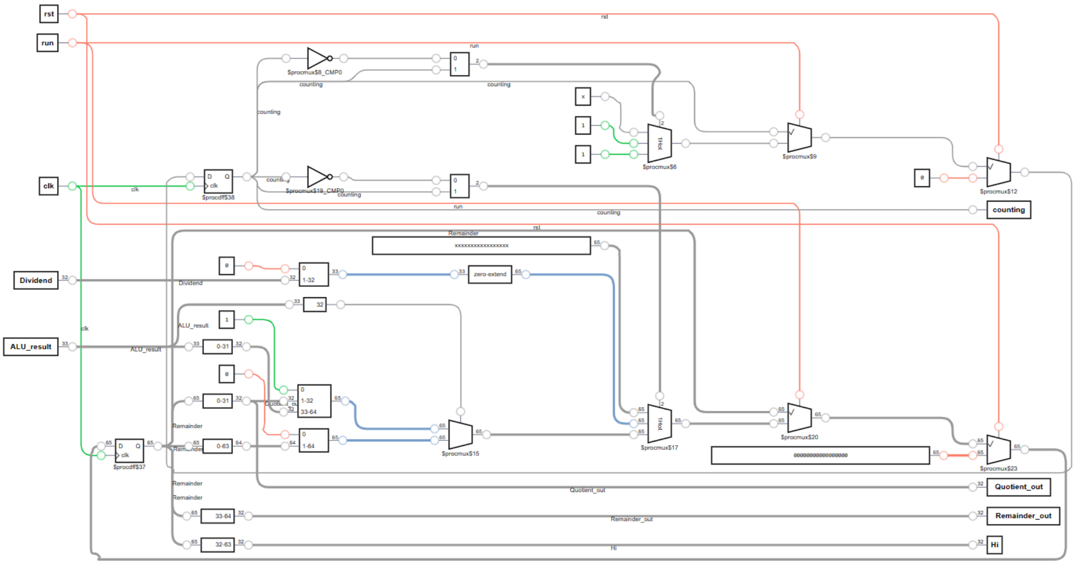
Line 20~31

When running count 32 negedges of clk then set ready flag to notify other component to stop or fetch data.

Remainder



```verilog
module Remainder (
    input clk,
    input rst,
    input run,
    input [31:0] Dividend,
    input [32:0] ALU_result,
    output [31:0] Hi,
    output [31:0] Remainder_out,
    output [31:0] Quotient_out,
    output counting
);
    reg state;
    reg [64:0] Remainder;

    assign Hi = Remainder[63:32];
    assign Remainder_out = Remainder[64:33];
    assign Quotient_out = Remainder[31:0];
    assign counting = state;

    always @(posedge clk) begin
        if (rst) begin
            state <= 0;
            Remainder <= 0;
        end
        else if (run) begin
            case (state)
                0: begin    // init state: Load Dividend into Remainder
                    Remainder <= {32'b0, Dividend, 1'b0};
                    state <= 1;
                end
                1: begin    // run state: Divide
                    if (ALU_result[32]) begin
                        Remainder <= {Remainder[63:32], Remainder[31:0], 1'b0};
                    end
                    else begin
                        Remainder <= {ALU_result[31:0], Remainder[31:0], 1'b1};
                    end
                    state <= 1;
                end
```



Description

Main module. I use 65 bits representing product in order to handle overflowing result from ALU. The computation is executed on posedge clk to avoid racing condition.

Detail

Line 1~11

I/O Interface.

Signal Symbol	Signal Name	Signal Description
clk	Clock	Clock
rst	Reset flag	Reset to init state
run	Run flag	Run when high
Dividend	32-bits dividend	In when init
ALU_result	33-bits ALU result	With 1 more borrow bit
Hi	Hi register	Upper 32 bits
Remainder_out	32-bits Remainder	34th bit to 65th bit to avoid the right shifting in last iteration
Quotient_out	32-bits Quotient	Output
counting	Counting flag	To avoid racing

Line 12~13

Registers.

Register Symbol	Register Name	Register Description
state	State	For FSM
Remainder	65-bits remainder	1 more bit to handle borrow bit from ALU

Line 15~18

Wire connection to Remainder_out, Quotient_out, HI, and counting.

Line 21~24

Reset.

Line 26~40

FSM

State Symbol	State Name	State Description
0	Init	Init Remainer by put Dividend to LO and shift left 1 bit.
1	run	Try to divide.

Line 31~39

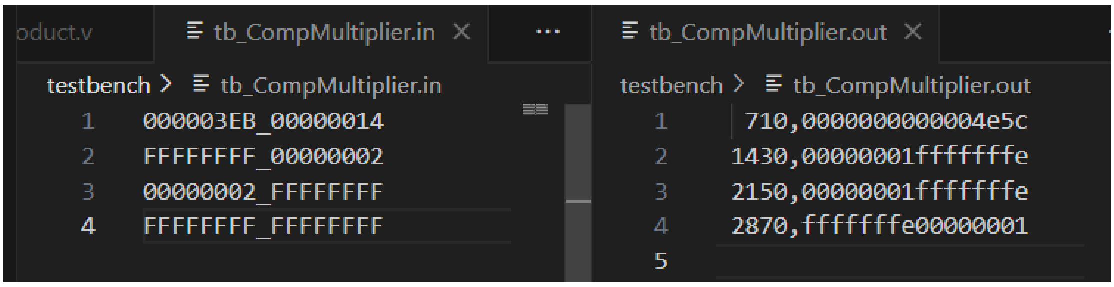
If ALU_result is negative, shift left 1 bit.

Else, HI = ALU_result, then shift left 1 bit. Finally set Remainder[0] to 1.

Test

Multiplier

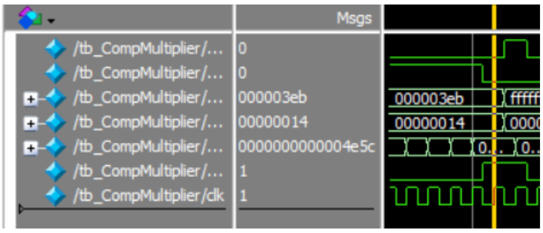
tb_CompMultiplier


oduct.v
tb_CompMultiplier.in
tb_CompMultiplier.out
testbench > tb_CompMultiplier.in
testbench > tb_CompMultiplier.out

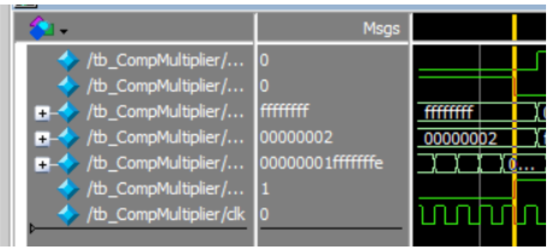
```
000003EB_00000014
FFFFFFFF_00000002
00000002_FFFFFFFF
FFFFFFFF_FFFFFFFF
```

```
710,0000000000004e5c
1430,00000001fffffffe
2150,00000001fffffffe
2870,fffffffe00000001
```

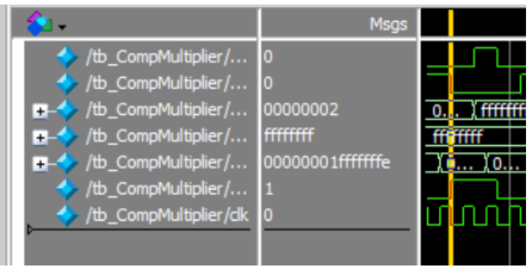
Multiplicand	Multiplier	Description
000003EB	00000014	Common case
FFFFFFFF	00000002	Multiplicand is extreme value.
00000002	FFFFFFFF	Multiplier is extreme value.
FFFFFFFF	FFFFFFFF	Both are extreme value.



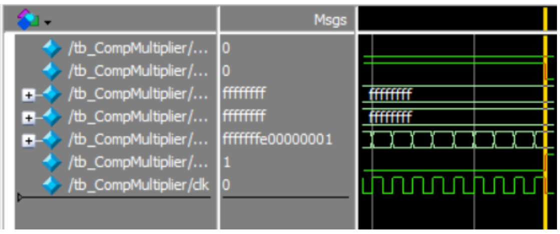
```
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  000003eb
/tb_CompMultiplier/...  00000014
/tb_CompMultiplier/...  0000000000004e5c
/tb_CompMultiplier/...  1
/tb_CompMultiplier/clk  1
```



```
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  ffffffff
/tb_CompMultiplier/...  00000002
/tb_CompMultiplier/...  00000001fffffffe
/tb_CompMultiplier/...  1
/tb_CompMultiplier/clk  0
```

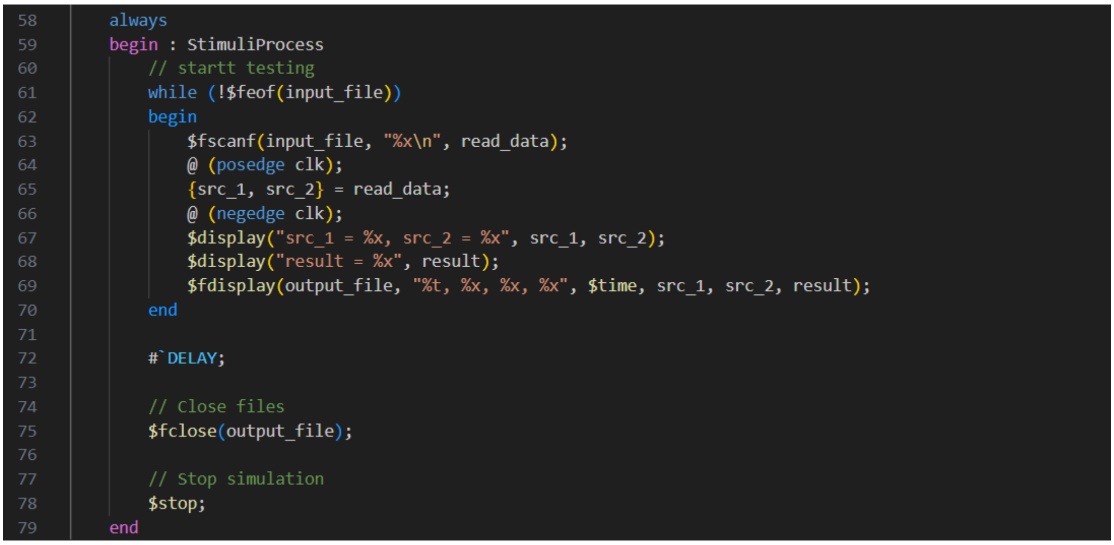


```
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  00000002
/tb_CompMultiplier/...  ffffffff
/tb_CompMultiplier/...  00000001fffffffe
/tb_CompMultiplier/...  1
/tb_CompMultiplier/clk  0
```

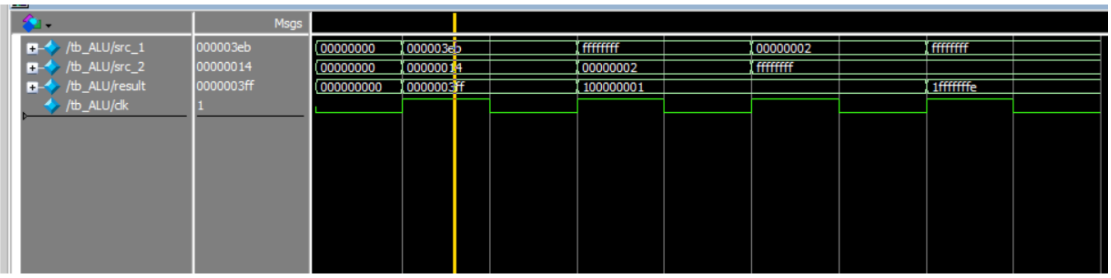


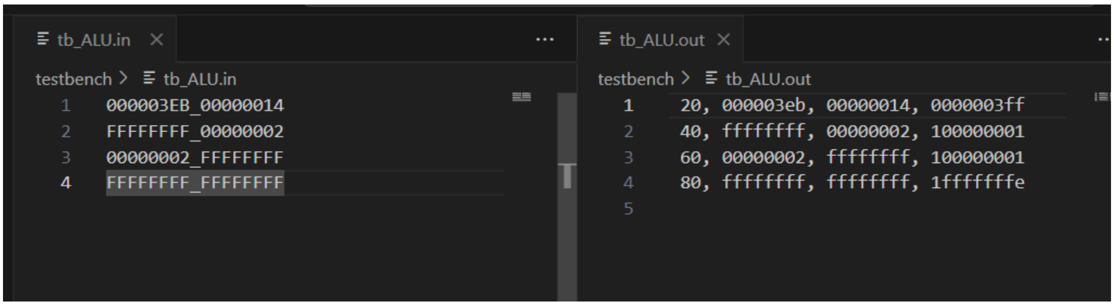
```
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  0
/tb_CompMultiplier/...  ffffffff
/tb_CompMultiplier/...  ffffffff
/tb_CompMultiplier/...  fffffffe00000001
/tb_CompMultiplier/...  1
/tb_CompMultiplier/clk  0
```

tb_ALU



```
always
begin : StimuliProcess
    // startt testing
    while (!$feof(input_file))
    begin
        $fscanf(input_file, "%x\n", read_data);
        @ (posedge clk);
        {src_1, src_2} = read_data;
        @ (negedge clk);
        $display("src_1 = %x, src_2 = %x", src_1, src_2);
        $display("result = %x", result);
        $fdisplay(output_file, "%t, %x, %x, %x", $time, src_1, src_2, result);
    end

    #`DELAY;

    // Close files
    $fclose(output_file);

    // Stop simulation
    $stop;
end
```



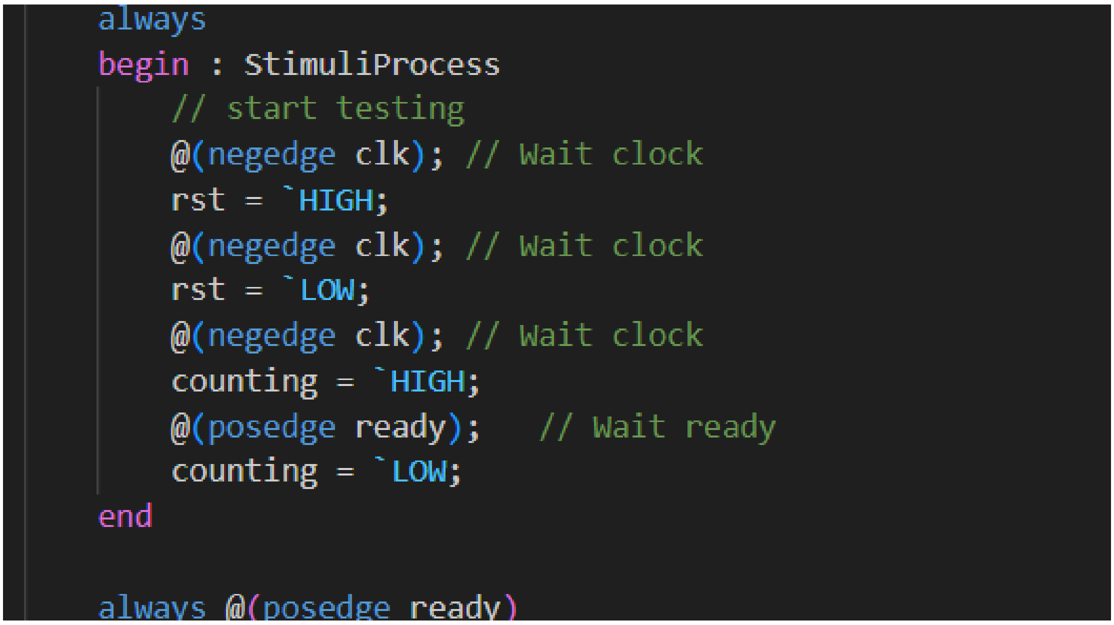


```
000003EB_00000014
FFFFFFFF_00000002
00000002_FFFFFFFF
FFFFFFFF_FFFFFFFF
```

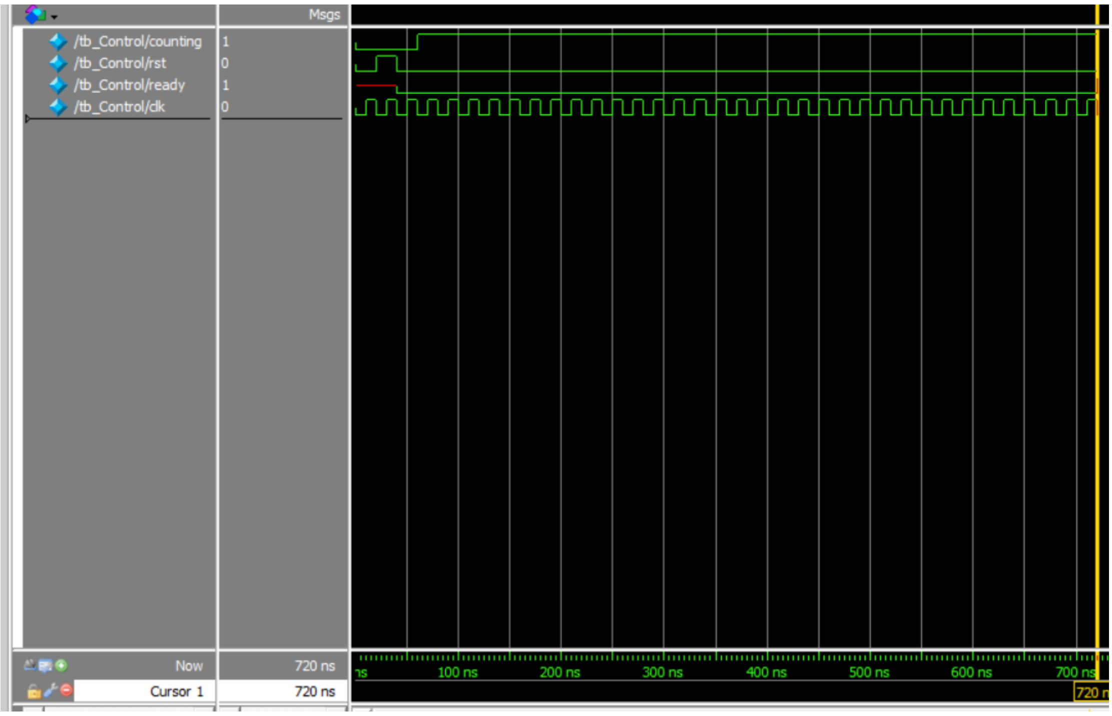
```
20, 000003eb, 00000014, 0000003ff
40, ffffffff, 00000002, 100000001
60, 00000002, ffffffff, 100000001
80, ffffffff, ffffffff, 1fffffffe
```

Test for functionality of ALU(only addition).

tb_Control



```
always
begin : StimuliProcess
    // start testing
    @(negedge clk); // Wait clock
    rst = `HIGH;
    @(negedge clk); // Wait clock
    rst = `LOW;
    @(negedge clk); // Wait clock
    counting = `HIGH;
    @(posedge ready);   // Wait ready
    counting = `LOW;
end

always @(posedge ready)
```



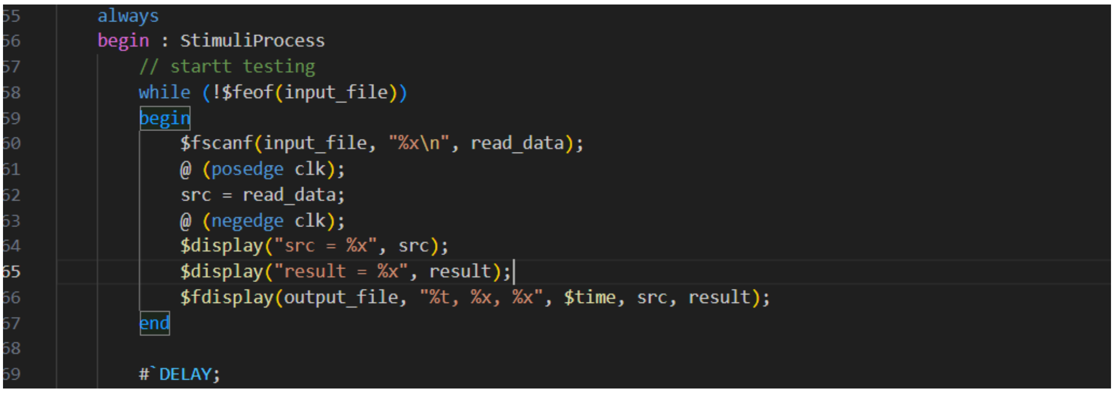
/tb_Control/counting 1
/tb_Control/rst 0
/tb_Control/ready 1
/tb_Control/clk 0

Now 720 ns
Cursor 1 720 ns

Test for functionality of Control.

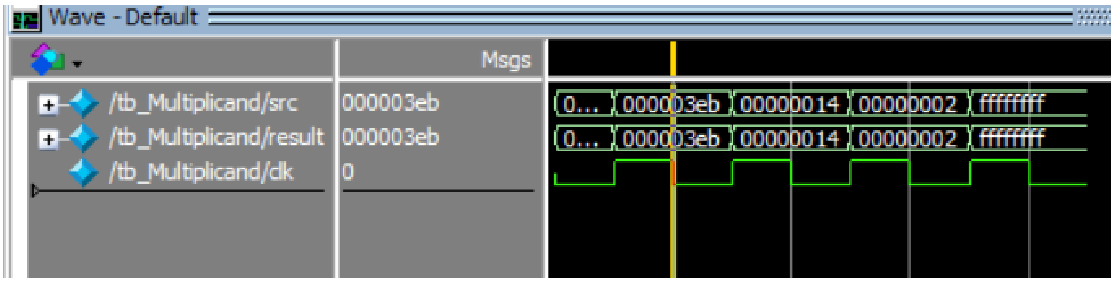
(720-60)/20 = 33, 0.5 more for start and 0.5 more for end to avoid racing.

tb_Multiplicand



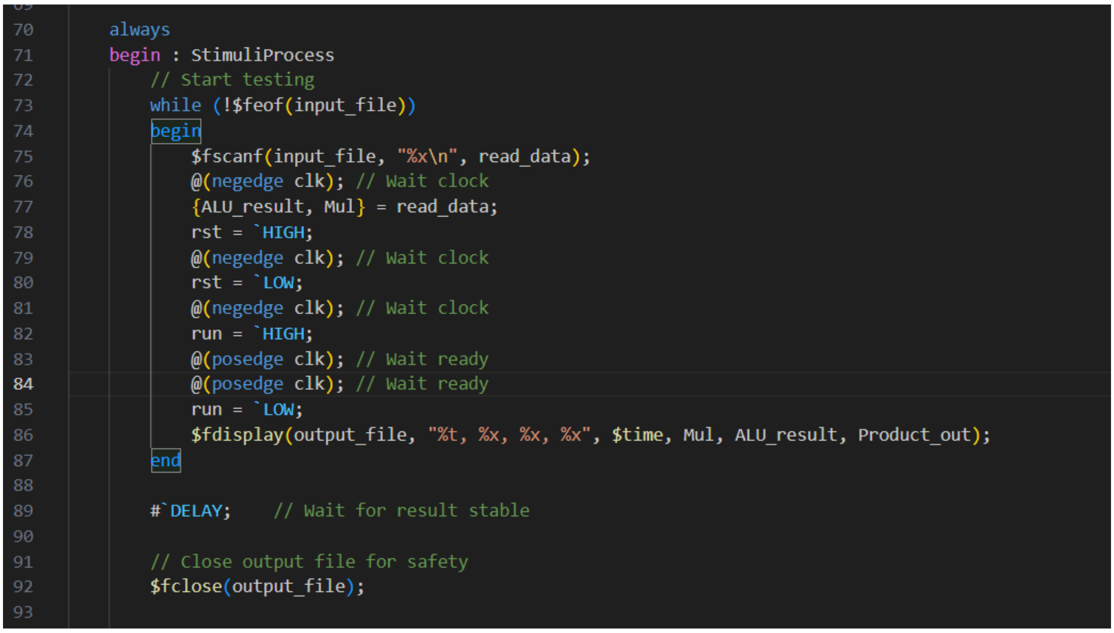
```
always
begin : StimuliProcess
    // startt testing
    while (!$feof(input_file))
    begin
        $fscanf(input_file, "%x\n", read_data);
        @ (posedge clk);
        src = read_data;
        @ (negedge clk);
        $display("src = %x", src);
        $display("result = %x", result);
        $fdisplay(output_file, "%t, %x, %x", $time, src, result);
    end

    #`DELAY;
```

Test for the functionality of Multiplicand.



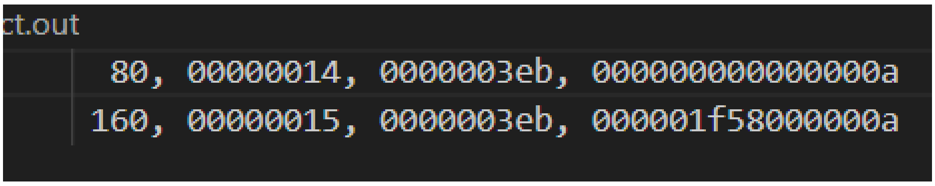
	Msgs	
/tb_Multiplicand/src	000003eb	0... 000003eb 00000014 00000002 ffffffff
/tb_Multiplicand/result	000003eb	0... 000003eb 00000014 00000002 ffffffff
/tb_Multiplicand/clk	0	

tb_Product

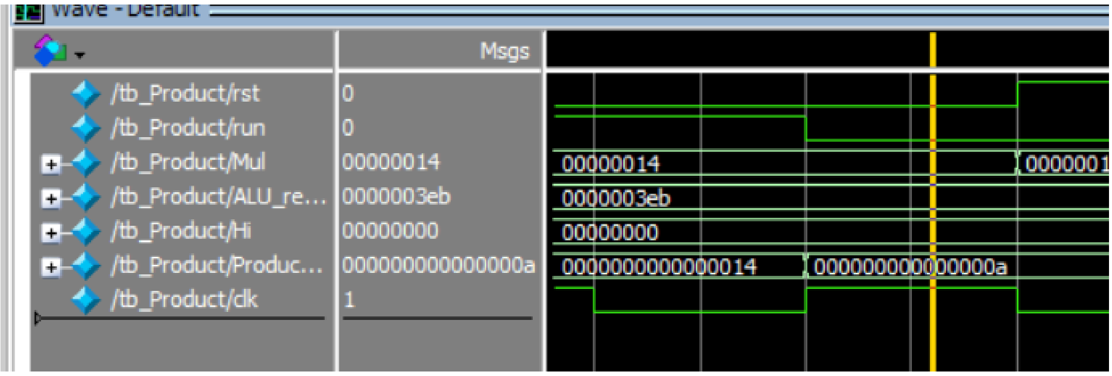


```
always
begin : StimuliProcess
    // Start testing
    while (!$feof(input_file))
    begin
        $fscanf(input_file, "%x\n", read_data);
        @(negedge clk); // Wait clock
        {ALU_result, Mul} = read_data;
        rst = `HIGH;
        @(negedge clk); // Wait clock
        rst = `LOW;
        @(negedge clk); // Wait clock
        run = `HIGH;
        @(posedge clk); // Wait ready
        @(posedge clk); // Wait ready
        run = `LOW;
        $fdisplay(output_file, "%t, %x, %x, %x", $time, Mul, ALU_result, Product_out);
    end

    #`DELAY;    // Wait for result stable

    // Close output file for safety
    $fclose(output_file);
```

Test for functionality.

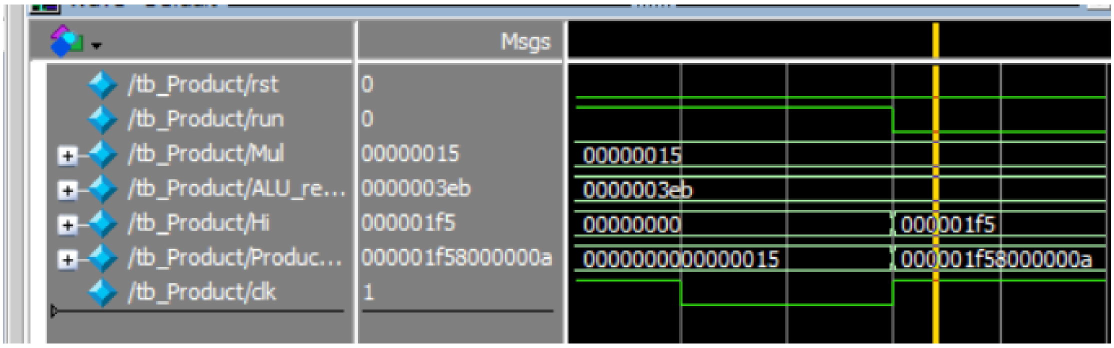


```
ct.out
  80, 00000014, 0000003eb, 000000000000000a
 160, 00000015, 0000003eb, 000001f58000000a
```



```
                              Msgs
/tb_Product/rst               0
/tb_Product/run               0
/tb_Product/Mul               00000014
/tb_Product/ALU_re...         0000003eb
/tb_Product/Hi                00000000
/tb_Product/Produc...         000000000000000a
/tb_Product/clk               1
```

14h LSB is not 1, shift right.

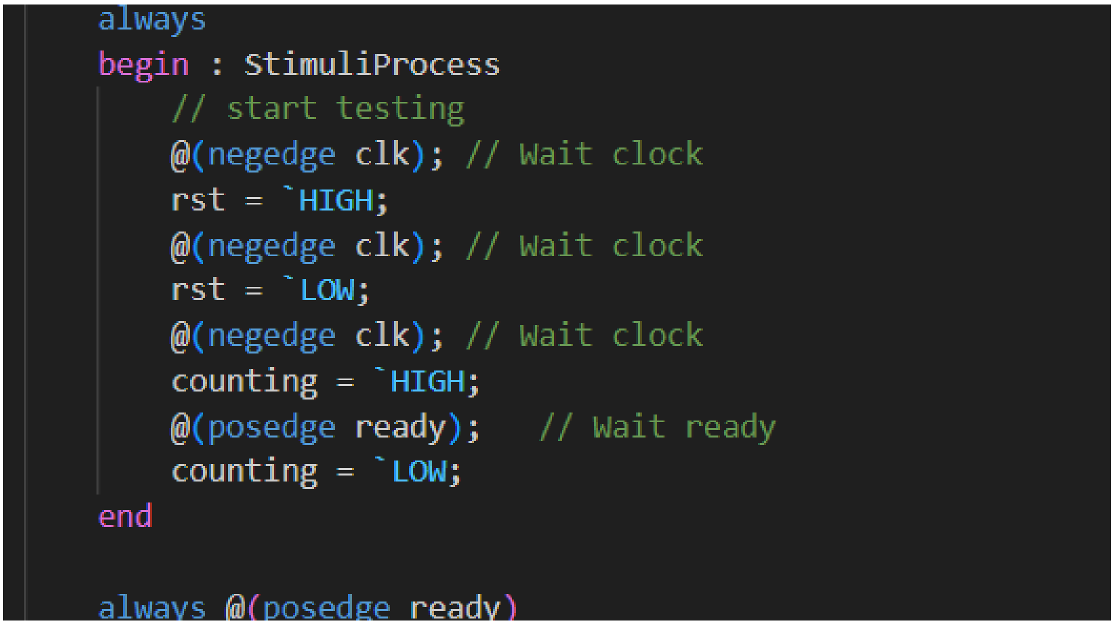


```
                              Msgs
/tb_Product/rst               0
/tb_Product/run               0
/tb_Product/Mul               00000015
/tb_Product/ALU_re...         0000003eb
/tb_Product/Hi                000001f5
/tb_Product/Produc...         000001f58000000a
/tb_Product/clk               1
```

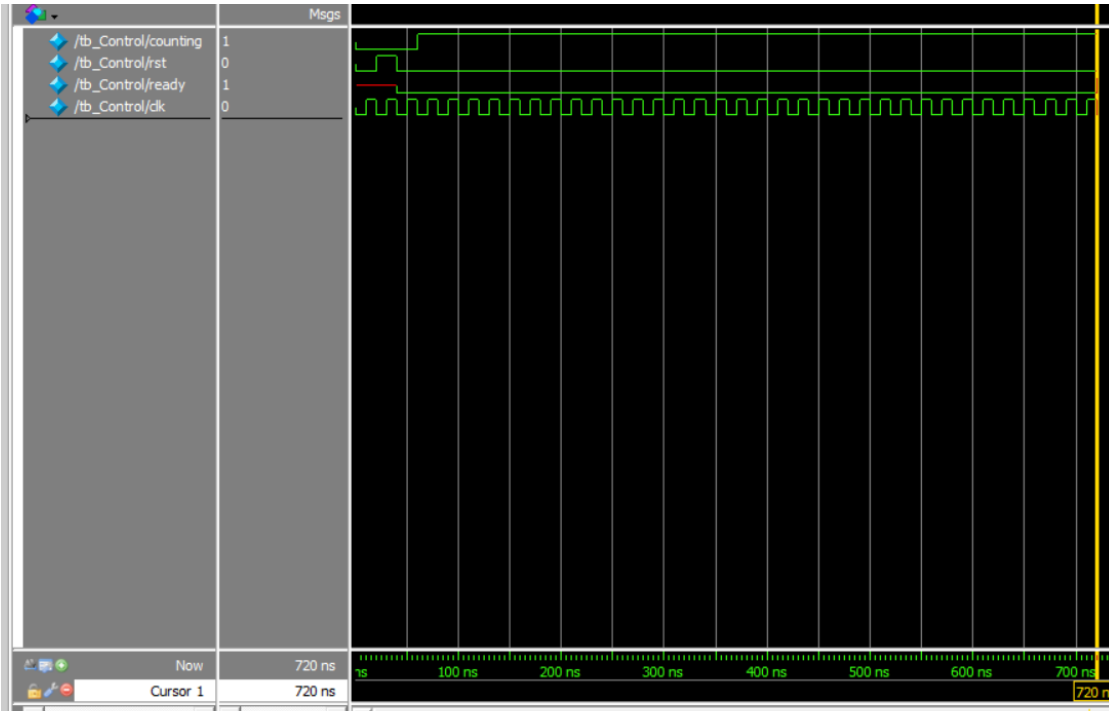
15h LSB is 1, add to HI, then shift right.

Divider

tb_Control



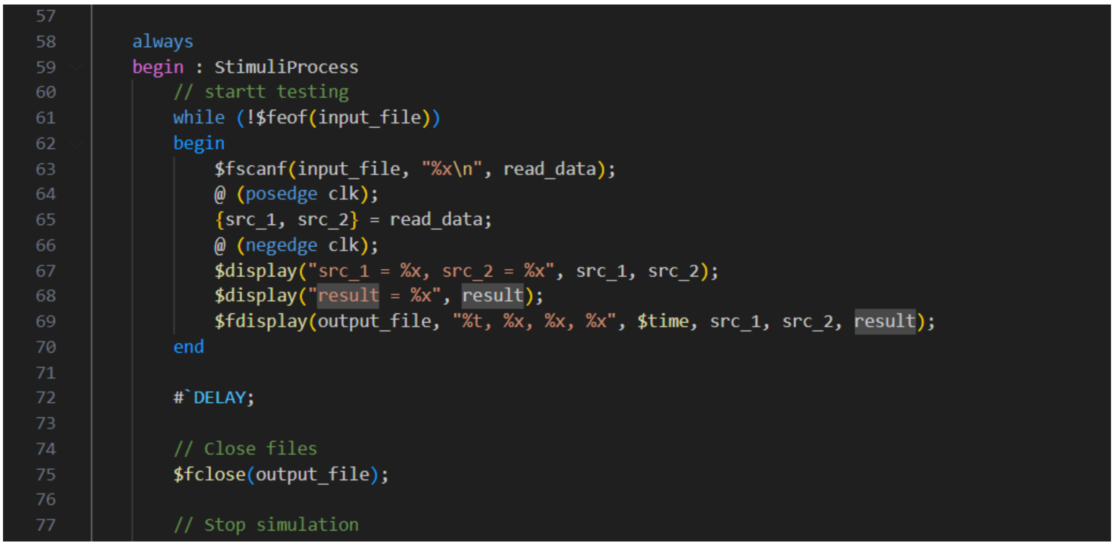
```
    always
    begin : StimuliProcess
        // start testing
        @(negedge clk); // Wait clock
        rst = `HIGH;
        @(negedge clk); // Wait clock
        rst = `LOW;
        @(negedge clk); // Wait clock
        counting = `HIGH;
        @(posedge ready);    // Wait ready
        counting = `LOW;
    end

    always @(posedge ready)
```



Test for functionality of Control.

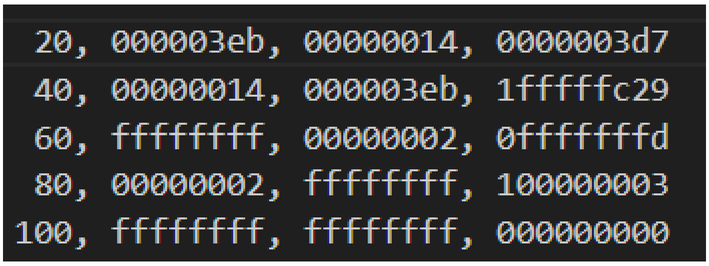
(720-60)/20 = 33, 0.5 more for start and 0.5 more for end to avoid racing.

tb_ALU

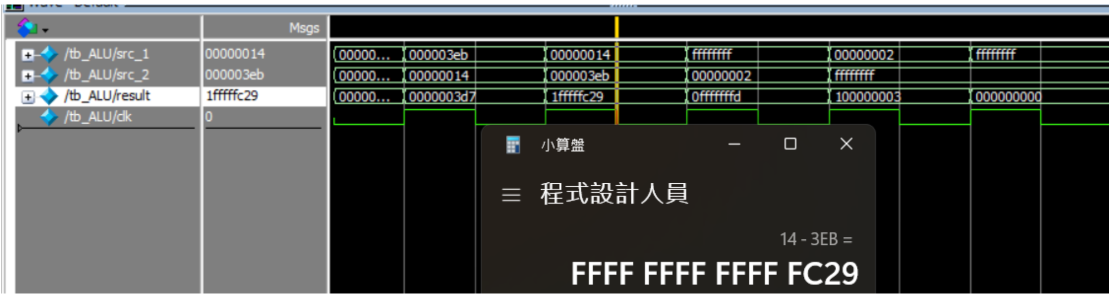


```
    always
    begin : StimuliProcess
        // startt testing
        while (!$feof(input_file))
        begin
            $fscanf(input_file, "%x\n", read_data);
            @ (posedge clk);
            {src_1, src_2} = read_data;
            @ (negedge clk);
            $display("src_1 = %x, src_2 = %x", src_1, src_2);
            $display("result = %x", result);
            $fdisplay(output_file, "%t, %x, %x, %x", $time, src_1, src_2, result);
        end

        #`DELAY;

        // Close files
        $fclose(output_file);

        // Stop simulation
```

Test for functionality



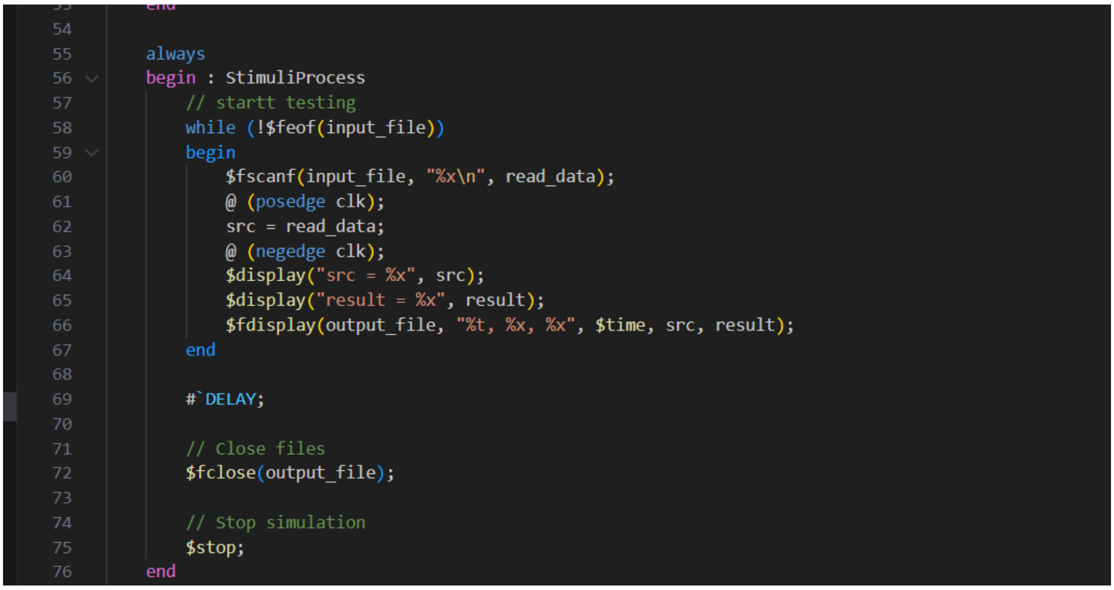
```
 20, 000003eb, 00000014, 0000003d7
 40, 00000014, 000003eb, 1fffffc29
 60, ffffffff, 00000002, 0fffffffd
 80, 00000002, ffffffff, 100000003
100, ffffffff, ffffffff, 000000000
```



```
14 - 3EB =
FFFF FFFF FFFF FC29
```

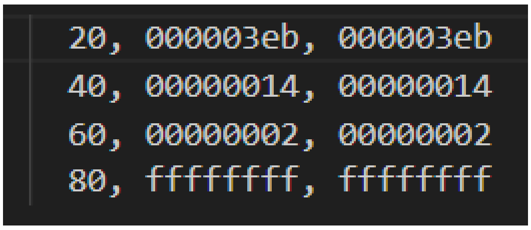
Note that result is 33-bits.

tb_Divisor

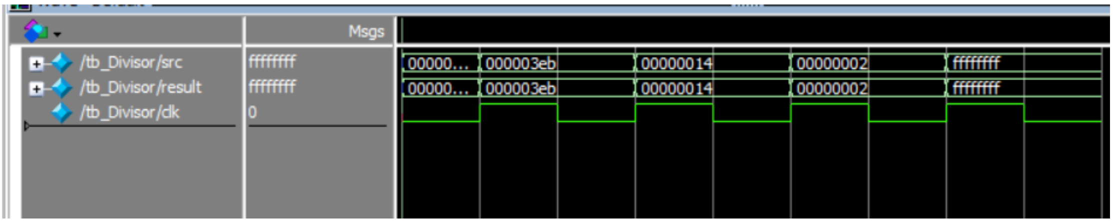


```
end

always
begin : StimuliProcess
    // startt testing
    while (!$feof(input_file))
    begin
        $fscanf(input_file, "%x\n", read_data);
        @ (posedge clk);
        src = read_data;
        @ (negedge clk);
        $display("src = %x", src);
        $display("result = %x", result);
        $fdisplay(output_file, "%t, %x, %x", $time, src, result);
    end

    #`DELAY;

    // Close files
    $fclose(output_file);

    // Stop simulation
    $stop;
end
```

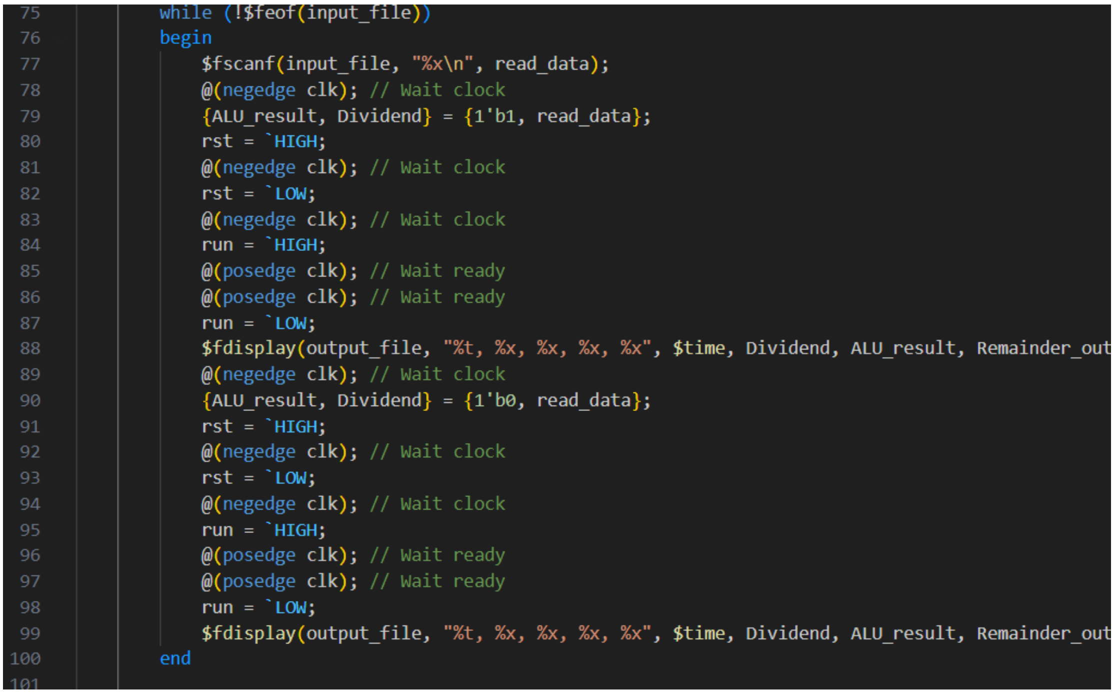
Test for functionality.



```
20, 000003eb, 000003eb
40, 00000014, 00000014
60, 00000002, 00000002
80, ffffffff, ffffffff
```

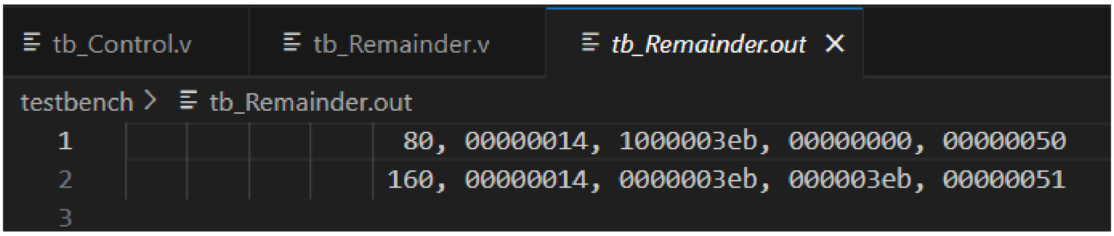


tb_Remainder

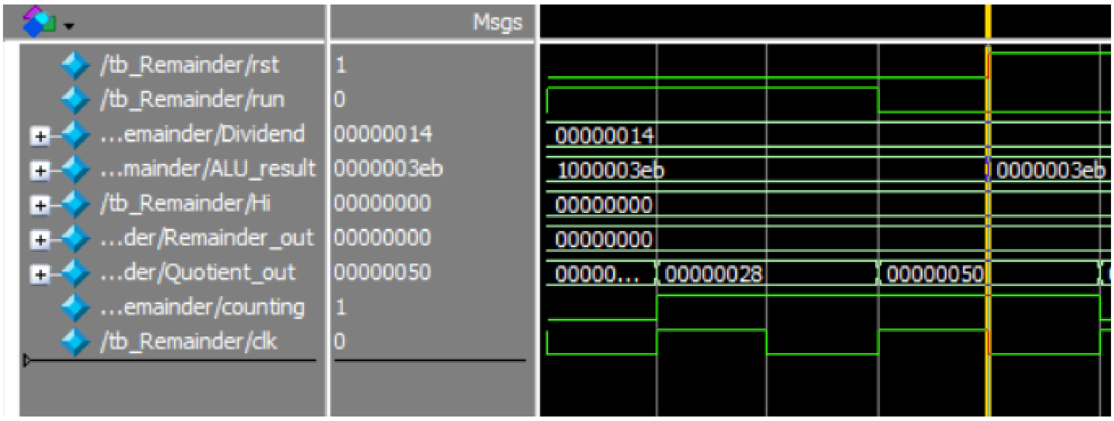


```verilog
        while (!$feof(input_file))
        begin
            $fscanf(input_file, "%x\n", read_data);
            @(negedge clk); // Wait clock
            {ALU_result, Dividend} = {1'b1, read_data};
            rst = `HIGH;
            @(negedge clk); // Wait clock
            rst = `LOW;
            @(negedge clk); // Wait clock
            run = `HIGH;
            @(posedge clk); // Wait ready
            @(posedge clk); // Wait ready
            run = `LOW;
            $fdisplay(output_file, "%t, %x, %x, %x, %x", $time, Dividend, ALU_result, Remainder_out
            @(negedge clk); // Wait clock
            {ALU_result, Dividend} = {1'b0, read_data};
            rst = `HIGH;
            @(negedge clk); // Wait clock
            rst = `LOW;
            @(negedge clk); // Wait clock
            run = `HIGH;
            @(posedge clk); // Wait ready
            @(posedge clk); // Wait ready
            run = `LOW;
            $fdisplay(output_file, "%t, %x, %x, %x, %x", $time, Dividend, ALU_result, Remainder_out
        end
```

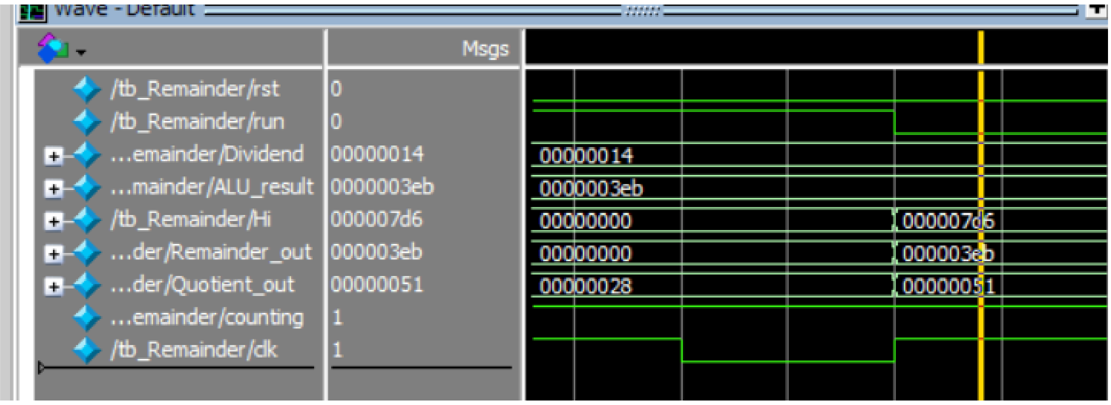
Test for functionality.



```
 80, 00000014, 1000003eb, 00000000, 00000050
160, 00000014, 0000003eb, 000003eb, 00000051
```



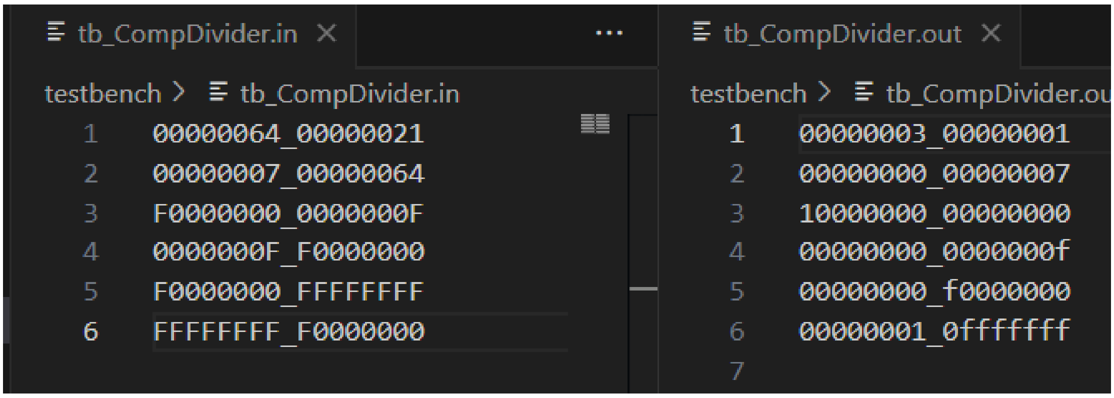
The first time ALU_result[32] = 0 -> 28h * 2h = 50h



```
/tb_Remainder/rst            0
/tb_Remainder/run            0
...emainder/Dividend         00000014
...mainder/ALU_result        0000003eb
/tb_Remainder/Hi             000007d6
...der/Remainder_out         000003eb
...der/Quotient_out          00000051
...emainder/counting         1
/tb_Remainder/clk            1
```

The first time ALU_result[32] = 1 -> 28h * 2h + 1h = 51h

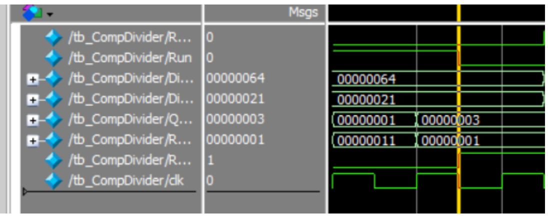
tb_CompDivider



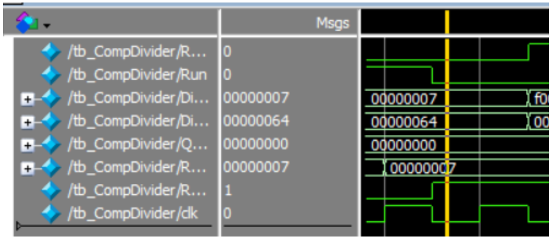
```
00000064_00000021
00000007_00000064
F0000000_0000000F
0000000F_F0000000
F0000000_FFFFFFFF
FFFFFFFF_F0000000
```

```
00000003_00000001
00000000_00000007
10000000_00000000
00000000_0000000f
00000000_f0000000
00000001_0fffffff

```

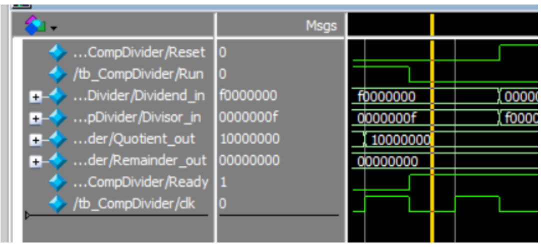
Dividend	Divisor	Description
00000064	00000021	Common case
00000007	00000064	Common case
F0000000	0000000F	Dividend is extreme value
0000000F	F0000000	Divisor is extreme value
F0000000	FFFFFFFF	Both are extreme
FFFFFFFF	F0000000	Both are extreme



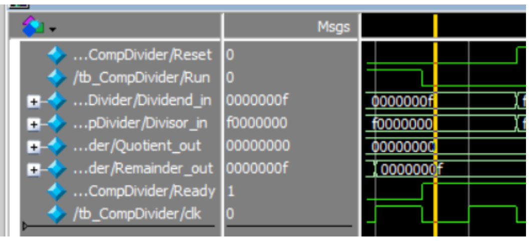
```
/tb_CompDivider/R...     0
/tb_CompDivider/Run      0
/tb_CompDivider/Di...    00000064
/tb_CompDivider/Di...    00000021
/tb_CompDivider/Q...     00000003
/tb_CompDivider/R...     00000001
/tb_CompDivider/R...     1
/tb_CompDivider/clk      0
```



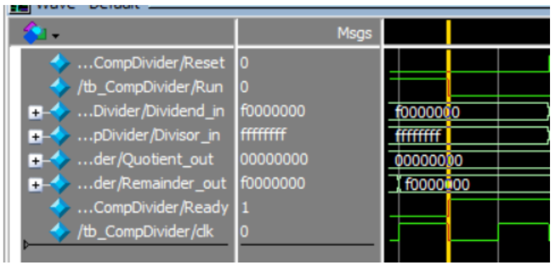
```
/tb_CompDivider/R...     0
/tb_CompDivider/Run      0
/tb_CompDivider/Di...    00000007
/tb_CompDivider/Di...    00000064
/tb_CompDivider/Q...     00000000
/tb_CompDivider/R...     00000007
/tb_CompDivider/R...     1
/tb_CompDivider/clk      0
```



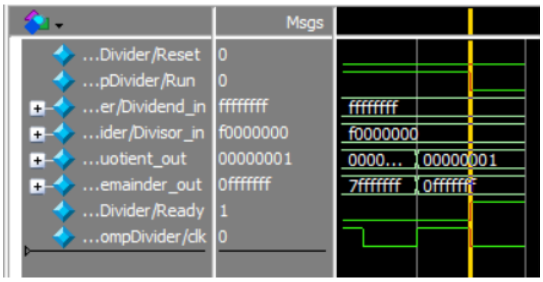
	Msgs
…CompDivider/Reset	0
/tb_CompDivider/Run	0
…Divider/Dividend_in	f0000000
…pDivider/Divisor_in	0000000f
…der/Quotient_out	10000000
…der/Remainder_out	00000000
…CompDivider/Ready	1
/tb_CompDivider/clk	0



	Msgs
…CompDivider/Reset	0
/tb_CompDivider/Run	0
…Divider/Dividend_in	0000000f
…pDivider/Divisor_in	f0000000
…der/Quotient_out	00000000
…der/Remainder_out	0000000f
…CompDivider/Ready	1
/tb_CompDivider/clk	0



	Msgs
…CompDivider/Reset	0
/tb_CompDivider/Run	0
…Divider/Dividend_in	f0000000
…pDivider/Divisor_in	ffffffff
…der/Quotient_out	00000000
…der/Remainder_out	f0000000
…CompDivider/Ready	1
/tb_CompDivider/clk	0



	Msgs
…Divider/Reset	0
…pDivider/Run	0
…er/Dividend_in	ffffffff
…ider/Divisor_in	f0000000
…uotient_out	00000001
…emainder_out	0fffffff
…Divider/Ready	1
…ompDivider/clk	0

Conclusion and Insights

The most valuable lesson I learnt from this assignment is making report is the most tedious thing in the world. I think whom it may concern will feel the same when reading our naïve reports.

Since Run flag will be set on negedge of clk, it'll be better if the mul/div operation is on posedge of clk because of the racing problem. Also, because of the counter should be count on negedge to make the last iteration's output stable (setup time constraint,) this will require 1 more clk to make this happen. Considering the above observation, I make Product/Remainder to tell the control when to start counting. **<u>This will 100% make no racing.</u>** My structure is not the best in performance, but I think it'll be relative stabler among all the other students' structures in this assignment.

During the implementation, I found out that control can just sit there doing nothing but counting, so I did. It'll need modification if considering merge multiplication and division or even merge all component to a complete ALU, but whatever. The requirement of this assignment doesn't forbid us to do this.

And there is one more trick I used. I make multiplication/division specific ALU to further simply the control path. As a result, the control just sitting there waiting for set up flags.