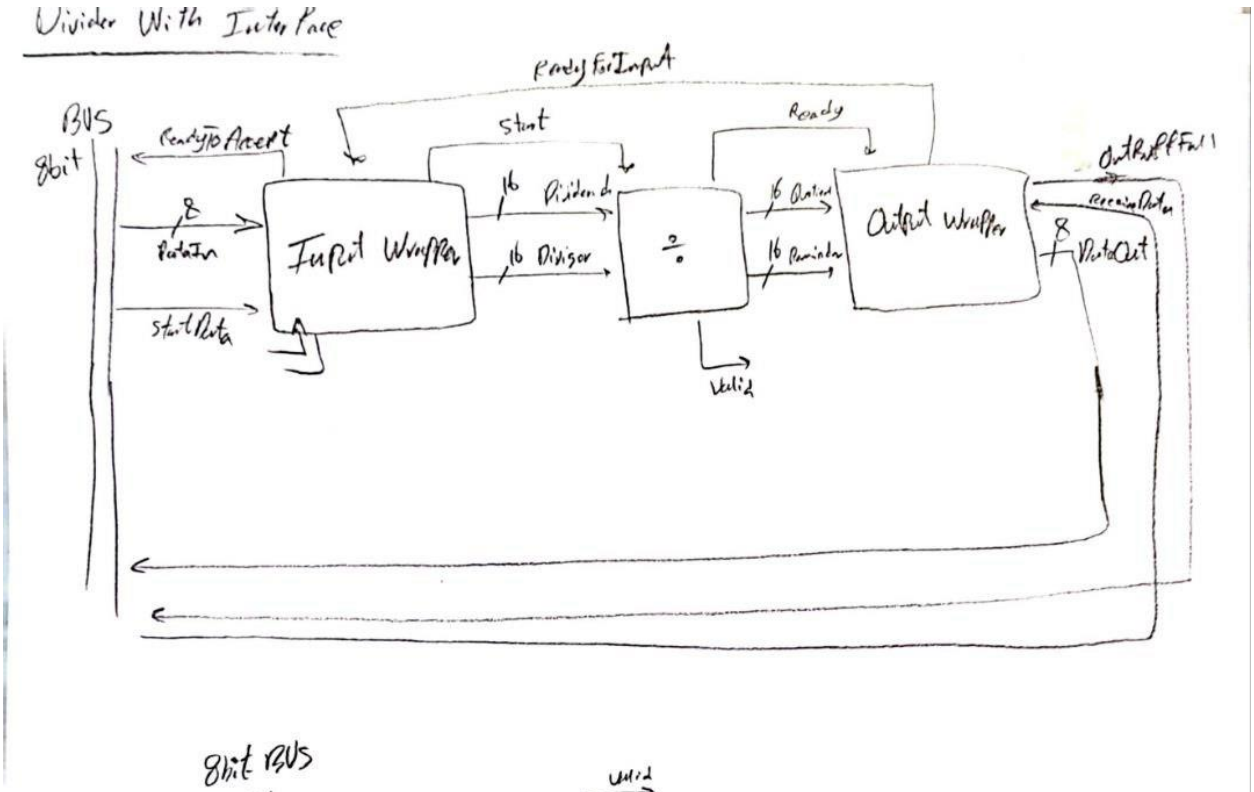
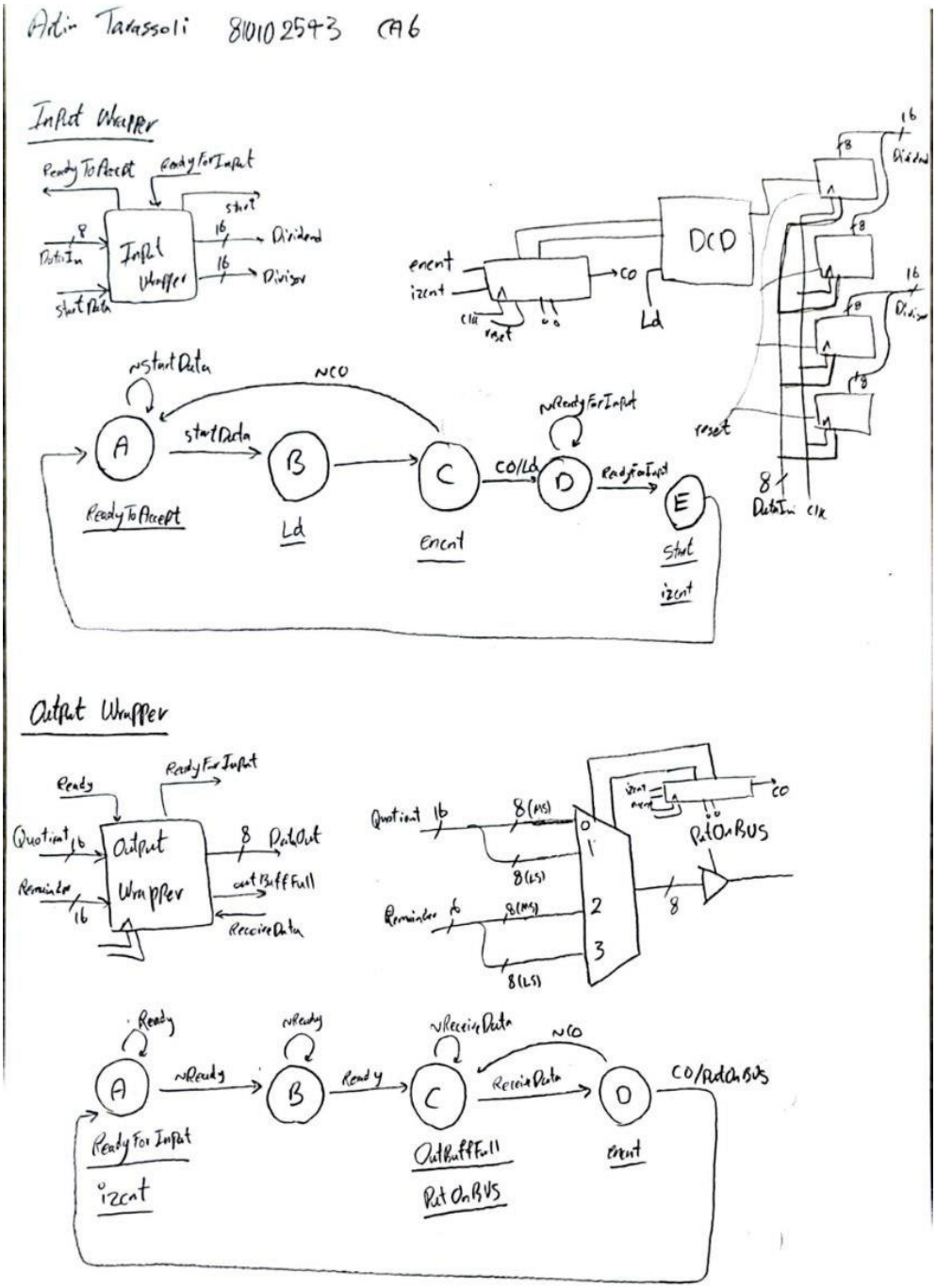
Artin Tavassoli 810102543 CA6

OVERVIEW:



DESIGN- DATA PATH AND CONTROLLER



INPUT WRAPPER:

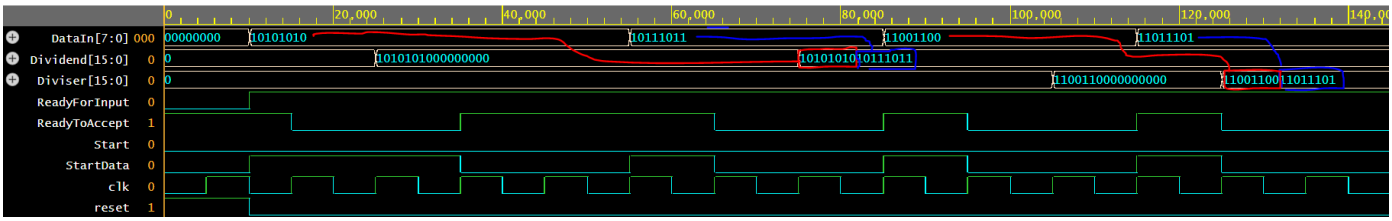
RESULTS

First 8-bit: Dividend Most Significant

Second 8-bit: Dividend Least Significant

Third 8-bit: Divisor Most Significant

Forth 8-bit: Divisor Least Significant



Code:

```
module Counter2bit (input enCnt, reset,izcnt, clk, output logic [1:0] cnt, output Co);
    assign Co = &cnt;
    always @(posedge clk or posedge reset) begin
        if (reset) cnt <= 2'b0;
        else if(izcnt) cnt <= 2'b0;
        else if (enCnt) cnt <= cnt + 1'b1;
    end
endmodule

module Dcd2to4 (input en,reset, input [1:0] in, output logic[3:0] out);
    always @(reset, en, in)begin
        out = 4'b0;
        if (reset)  out = 4'b0;
        else if(en)begin
            case(in)
                2'd0:  out = 4'b0001;
                2'd1:  out = 4'b0010;
                2'd2:  out = 4'b0100;
                2'd3:  out = 4'b1000;
                default: out = 4'b0;
            endcase
        end
        else  out = 4'b0;
    end
endmodule

module Register8bit (input [7:0] in, input clk, load, reset, output logic [7:0] out);
    always @(posedge clk or posedge reset) begin
        if (reset) out <= 8'b0;
        else if (load) out <= in;
    end
endmodule
```

```
module Input_Wrapper_Datapath(input encnt,izcnt,reset,load,clk, input [7:0]DataIn,
        output [15:0] Dividend,Diviser, output Co);
    wire [1:0] cnt;
    Counter2bit counter(.enCnt(encnt), .reset(reset), .izcnt(izcnt), .clk(clk), .cnt(cnt), .Co(Co));
    wire [3:0] registers_load;
    Dcd2to4 dcd(.en(load), .reset(reset), .in(cnt), .out(registers_load));
    wire [7:0] Dividend_8bitMS, Dividend_8bitLS, Diviser_8bitMS, Diviser_8bitLS;
    Register8bit rg1(.in(DataIn), .clk(clk), .load(registers_load[0]), .reset(reset), .out(Dividend_8bitMS));
    Register8bit rg2(.in(DataIn), .clk(clk), .load(registers_load[1]), .reset(reset), .out(Dividend_8bitLS));
    Register8bit rg3(.in(DataIn), .clk(clk), .load(registers_load[2]), .reset(reset), .out(Diviser_8bitMS));
    Register8bit rg4(.in(DataIn), .clk(clk), .load(registers_load[3]), .reset(reset), .out(Diviser_8bitLS));
    assign Dividend = {Dividend_8bitMS, Dividend_8bitLS};
    assign Diviser = {Diviser_8bitMS, Diviser_8bitLS};
endmodule

module Input_Wrapper_Controller(input ReadyForInput,StartData,Co,clk,reset,
            output logic ReadyToAccept, Start, encnt,izcnt, output load);
        typedef enum {A,B,C,D,E} states;
        states state,next_state;
        always @(posedge clk or posedge reset) begin
            if(reset) state <= A;
            else state <= next_state;
        end
        assign load = (Co || (state == B)) ? 1'b1 : 1'b0;
        always @(state, StartData, Co, ReadyForInput) begin
        {ReadyToAccept, encnt, Start, izcnt} = 0;
        //next_state = state;
        case (state)
            A: begin
                next_state = StartData ? B : A;
                ReadyToAccept = 1;
            end
            B: begin
                next_state = C;
            end
            C: begin
                encnt = 1;
                next_state = Co ? D : A;
            end
            D: begin
                next_state = ReadyForInput ? E : D;
            end
            E: begin
                Start = 1;
                izcnt = 1;
                next_state = A;
            end
        endcase
    end
endmodule
```

```
module Input_Wrapper(input ReadyForInput,StartData, reset,clk, input [7:0]DataIn,
        output [15:0] Dividend,Diviser, output ReadyToAccept, Start);

        wire encnt, izcnt, load, Co;
        Input_Wrapper_Datapath Dp(.encnt(encnt), .izcnt(izcnt), .reset(reset), .load(load), .clk(clk),
                .DataIn(DataIn), .Dividend(Dividend), .Diviser(Diviser), .Co(Co));

        Input_Wrapper_Controller controller(.ReadyForInput(ReadyForInput), .StartData(StartData), .Co(Co), .clk(clk), .reset(reset),
         .ReadyToAccept(ReadyToAccept), .Start(Start), .encnt(encnt), .izcnt(izcnt), .load(load));
endmodule
```

Testbench:

```
module Input_Wrapper_tb;
    reg [7:0] DataIn;
    wire [15:0] Dividend, Diviser;
    wire ReadyToAccept, Start;
    Input_Wrapper dut (
        .ReadyForInput(ReadyForInput),
        .StartData(StartData),
        .reset(reset),
        .clk(clk),
        .DataIn(DataIn),
        .Dividend(Dividend),
        .Diviser(Diviser),
        .ReadyToAccept(ReadyToAccept),
        .Start(Start)
    );

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        ReadyForInput = 0; StartData = 0; reset = 1; DataIn = 8'h00;
        #10 reset = 0;

        ReadyForInput = 1; StartData = 1; DataIn = 8'hAA;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0; #20;

        StartData = 1; DataIn = 8'hBB;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0; #20;

        StartData = 1; DataIn = 8'hCC;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0; #20;

        StartData = 1; DataIn = 8'hDD;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0; #20;
        $stop;
    end
endmodule
```

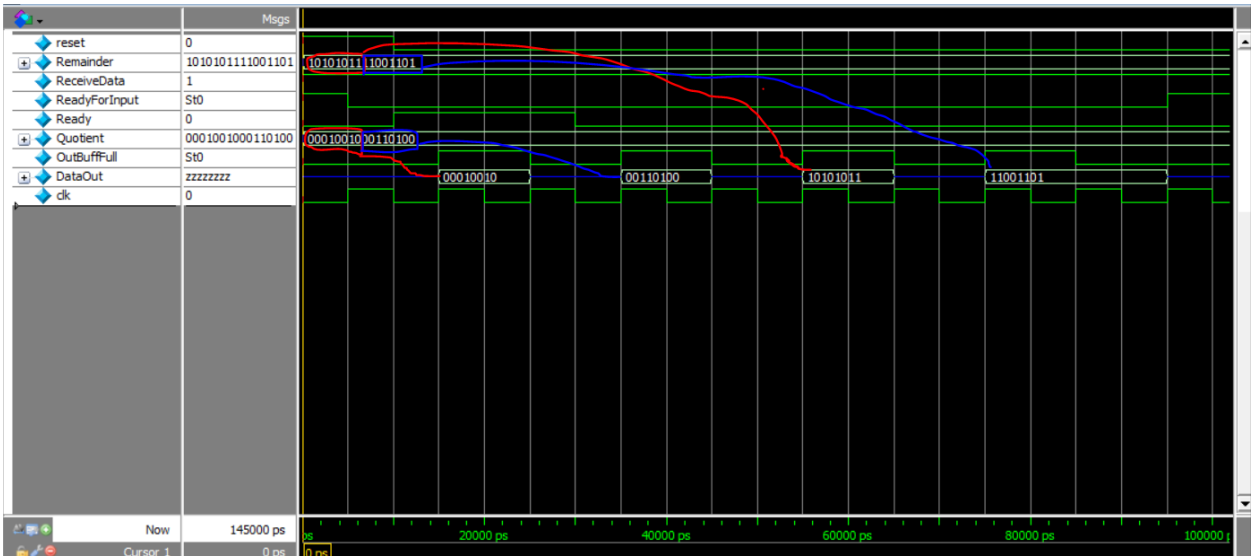
OUTPUT WRAPPER:

First 8-bit: Quotient Most Significant

Second 8-bit: Quotient Least Significant

Third 8-bit: Remainder Most Significant

Forth 8-bit: Remainder Least Significant



Code:

```
module Counter2bit (input enCnt, reset,izcnt, clk, output logic [1:0] cnt, output Co);
    assign Co = &cnt;
    always @(posedge clk or posedge reset) begin
        if (reset) cnt <= 2'b0;
        else if(izcnt) cnt <= 2'b0;
        else if (enCnt) cnt <= cnt + 1'b1;
    end
endmodule

module Multiplexer4to1(input[1:0] select, input[31:0] in, output logic [7:0] out);
    always @(select, in)begin
            out = 7'b0;
            case(select)
                2'd0:  out = in[31:24];
                2'd1:  out = in[23:16];
                2'd2:  out = in[15:8];
                2'd3:  out = in[7:0];
                default: out = 7'b0;
            endcase
        end
endmodule

module Buffif1(input [7:0] in, input select, output [7:0] out);
    assign out = select ? in : 8'bzzzzzzzz;
endmodule

module Output_Wrapper_Datapath(input [15:0] Quotient, Remainder, input PutOnBus,encnt,izcnt,reset,clk, output [7:0] out, output Co);
    wire [1:0] cnt;
    Counter2bit counter(.enCnt(encnt), .reset(reset), .izcnt(izcnt), .clk(clk), .cnt(cnt), .Co(Co));
    wire [7:0] multiplexer_output;
    Multiplexer4to1 multiplexer(.select(cnt), .in({Quotient,Remainder}), .out(multiplexer_output));
    Buffif1 buffer(.in(multiplexer_output), .select(PutOnBus), .out(out));
endmodule
```

```
module Output_Wrapper_Controller(input Ready,ReceiveData,Co,clk,reset,
            output logic ReadyForInput, OutBuffFull, encnt,izcnt,output PutOnBus);
        typedef enum {A,B,C,D} states;
        states state,next_state;
         always @(posedge clk or posedge reset) begin
            if(reset) state <= A;
            else state <= next_state;
        end
        assign PutOnBus = (Co || (state == C)) ? 1'b1 : 1'b0;
        always @(state, Ready, ReceiveData, Co) begin
        {ReadyForInput, OutBuffFull, encnt, izcnt} = 0;
        case (state)
            A: begin
                ReadyForInput = 1;
                izcnt = 1;
                next_state = Ready ? A : B;
            end
            B: begin
                next_state = Ready ? C : B;
            end
            C: begin
                OutBuffFull = 1;
                next_state = ReceiveData ? D : C;
            end
            D: begin
                encnt = 1;
                next_state = Co ? A : C;
            end
        endcase
    end
endmodule

module Output_Wrapper(input Ready, ReceiveData, reset,clk,input [15:0] Quotient,Remainder,
        output ReadyForInput,OutBuffFull, output [7:0]DataOut);
        wire encnt, izcnt, PutOnBus, Co;
        Output_Wrapper_Datapath Dp(.Quotient(Quotient), .Remainder(Remainder),.encnt(encnt),
            .izcnt(izcnt),.reset(reset),.clk(clk), .PutOnBus(PutOnBus),.out(DataOut), .Co(Co));
        Output_Wrapper_Controller controller(.Ready(Ready), .ReceiveData(ReceiveData), .Co(Co), .clk(clk),
            .reset(reset), .ReadyForInput(ReadyForInput), .OutBuffFull(OutBuffFull),
             .encnt(encnt), .izcnt(izcnt), .PutOnBus(PutOnBus));
endmodule
```

Testbench:

```
`timescale 1ps/1ps
module Divider_With_Interface_tb;
    reg StartData, reset, clk, ReceiveData;
    reg [7:0] DataIn;
    wire ReadyToAccept1, Valid1, OutBuffFull1;
    wire [7:0] DataOut1;
    wire ReadyToAccept, Valid, OutBuffFull;
    wire [7:0] DataOut;

    Divider_With_Interface_post DUT (
        .StartData(StartData),
        .reset(reset),
        .clk(clk),
        .ReceiveData(ReceiveData),
        .DataIn(DataIn),
        .ReadyToAccept(ReadyToAccept1),
        .Valid(Valid1),
        .OutBuffFull(OutBuffFull1),
        .DataOut(DataOut1)
    );

     Divider_With_Interface_pre DUT_pre (
        .StartData(StartData),
        .reset(reset),
        .clk(clk),
        .ReceiveData(ReceiveData),
        .DataIn(DataIn),
        .ReadyToAccept(ReadyToAccept),
        .Valid(Valid),
        .OutBuffFull(OutBuffFull),
        .DataOut(DataOut)
    );
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
```

```
2   module Divider_With_Interface_tb;
37
38      initial begin
39          StartData = 0; reset = 1; DataIn = 8'b0;
40          #10 reset = 0;
41          ReceiveData = 0;
42          #20
43
44          StartData = 1; DataIn = 8'b01010110;
45          #10;
46          wait (ReadyToAccept == 1);
47          StartData = 0; #20;
48
49          StartData = 1; DataIn = 8'b10011101;
50          #10;
51          wait (ReadyToAccept == 1);
52          StartData = 0; #20;
53
54          StartData = 1; DataIn = 8'b00000101;
55          #10;
56          wait (ReadyToAccept == 1);
57          StartData = 0; #20;
58
59          StartData = 1; DataIn = 8'b10000101;
60          #10;
61          wait (ReadyToAccept == 1);
62          StartData = 0;
63          #50;
64
65          repeat (4) begin
66              wait(OutBuffFull);
67              ReceiveData = 1;
68              #20;
69              ReceiveData = 0;
70              #20;
71          end
72
73          #50;
74          $stop;
```

DIVIDER WITH RTL INTERACTION:

Suppose we want to divide 22173 by 1413:

01010110 10011101 = 22173

00000101 10000101 = 1413

Q = 15 = 00000000 00001111

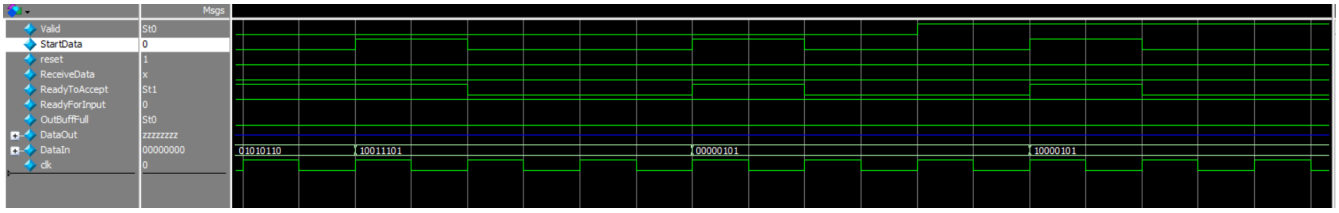
R = 978 = 00000011 11010010

DataIn: 01010110, 10011101, 00000101, 10000101

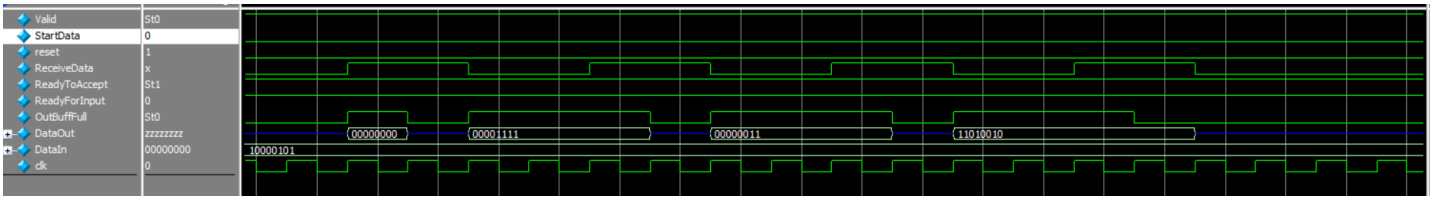
DataOut: 00000000, 00001111, 00000011, 11010010 (and z between them)

RESULTS

Sending Data:



Receiving Data:



Code:

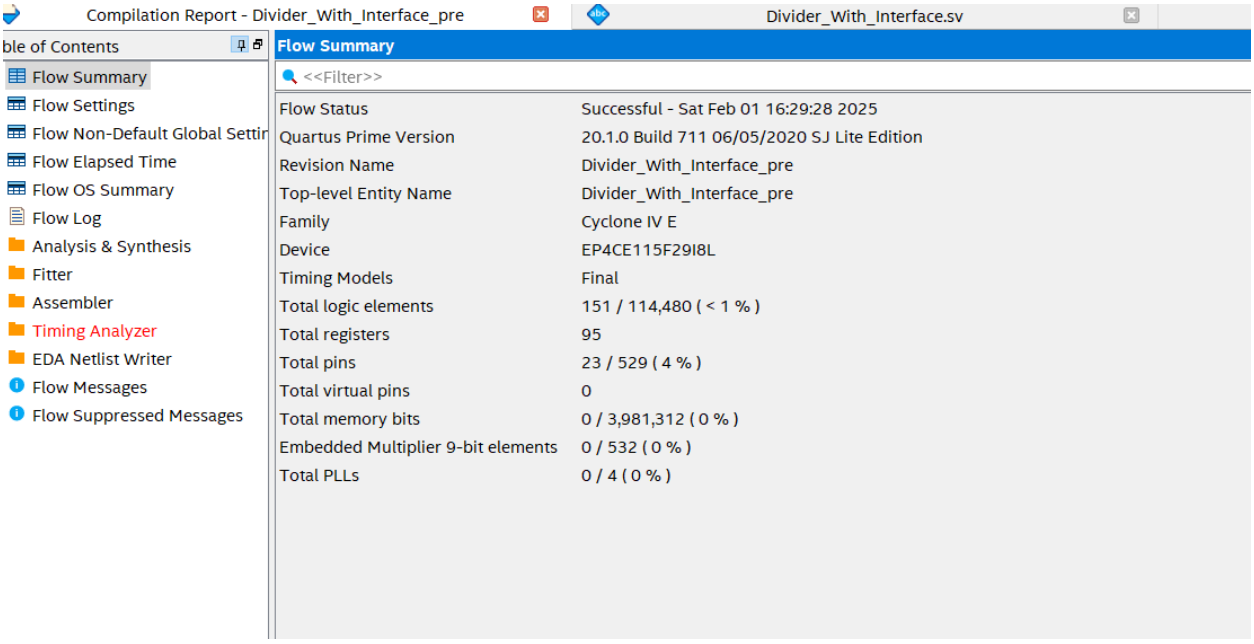
```
module Divider_With_Interface(input StartData, reset, clk, ReceiveData,  input [7:0]DataIn,
    output ReadyToAccept, Valid, OutBuffFull, output [7:0]DataOut);
    wire ReadyForInput, Start, Ready;
    wire [15:0]Dividend, Divisor, Quotient, Remainder;
    Input_Wrapper input_wrapper(.ReadyForInput(ReadyForInput), .StartData(StartData), .clk(clk), .reset(reset),
    .DataIn(DataIn), .Dividend(Dividend), .Diviser(Divisor), .ReadyToAccept(ReadyToAccept), .Start(Start));
    Divider divider(.Dividend(Dividend), .Divisor(Divisor), .clk(clk), .start(Start),
    .rst(reset), .R(Remainder), .Q(Quotient), .ready(Ready), .valid(Valid));
    Output_Wrapper output_wrapper(.Ready(Ready), .ReceiveData(ReceiveData), .reset(reset), .clk(clk),
    .Quotient(Quotient), .Remainder(Remainder), .ReadyForInput(ReadyForInput), .OutBuffFull(OutBuffFull), .DataOut(DataOut));
endmodule
```

Testbench:

```
module Divider_With_Interface_tb;
    wire ReadyToAccept1, Valid1, OutBuffFull1;
    wire [7:0] DataOut1;
    wire ReadyToAccept, Valid, OutBuffFull;
    wire [7:0] DataOut;

    Divider_With_Interface_post DUT (
        .StartData(StartData),
        .reset(reset),
        .clk(clk),
        .ReceiveData(ReceiveData),
        .DataIn(DataIn),
        .ReadyToAccept(ReadyToAccept1),
        .Valid(Valid1),
        .OutBuffFull(OutBuffFull1),
        .DataOut(DataOut1)
    );

     Divider_With_Interface_pre DUT_pre (
        .StartData(StartData),
        .reset(reset),
        .clk(clk),
        .ReceiveData(ReceiveData),
        .DataIn(DataIn),
        .ReadyToAccept(ReadyToAccept),
        .Valid(Valid),
        .OutBuffFull(OutBuffFull),
        .DataOut(DataOut)
    );
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        StartData = 0; reset = 1; DataIn = 8'b0;
        #10 reset = 0;
        ReceiveData = 0;
        #20

        StartData = 1; DataIn = 8'b01010110;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0; #20;

        StartData = 1; DataIn = 8'b10011101;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0; #20;

        StartData = 1; DataIn = 8'b00000101;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0; #20;

        StartData = 1; DataIn = 8'b10000101;
        #10;
        wait (ReadyToAccept == 1);
        StartData = 0;
        #50;

        repeat (4) begin
            wait(OutBuffFull);
            ReceiveData = 1;
            #20;
            ReceiveData = 0;
            #20;
        end

        #50;
```

SYNTHESIS:

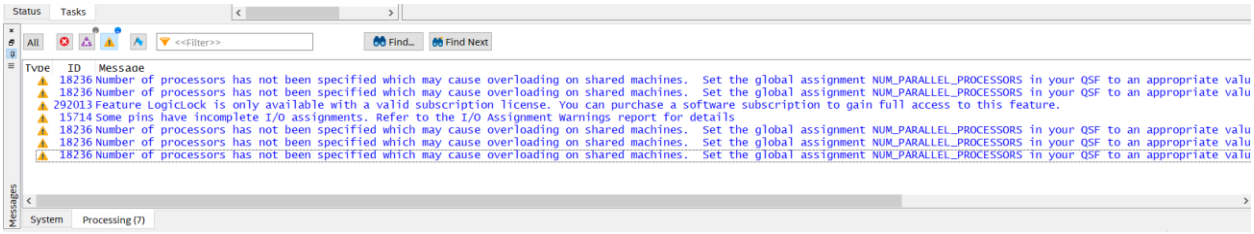
Total of 95 registers:



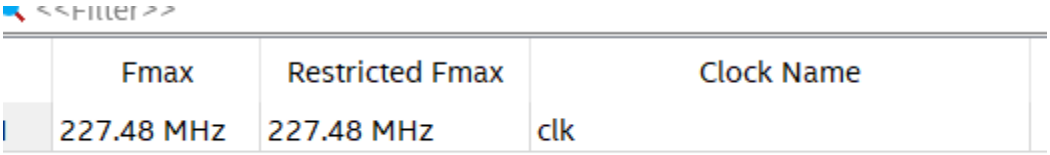
Utilization of these registers:

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtua
1	|Divider_With_Interface_pre	138 (1)	95 (0)	0	0	0	0	23	0
1	|Divider:divider|	96 (0)	54 (0)	0	0	0	0	0	0
1	|Divider_Controller:controller_module|	11 (6)	6 (2)	0	0	0	0	0	0
1	|Counter:count_module|	5 (5)	4 (4)	0	0	0	0	0	0
2	|Divider_Datapath:datapath_module|	85 (5)	48 (0)	0	0	0	0	0	0
1	|Absolute:Aabs|	18 (18)	0 (0)	0	0	0	0	0	0
2	|Left_Shift_Register:Areg|	14 (14)	16 (16)	0	0	0	0	0	0
3	|Left_Shift_Register:Temp_reg|	16 (16)	16 (16)	0	0	0	0	0	0
4	|Register:Breg|	16 (16)	16 (16)	0	0	0	0	0	0
5	|Subtract:sub|	16 (16)	0 (0)	0	0	0	0	0	0
2	|Input_Wrapper:input_wrapper|	15 (0)	37 (0)	0	0	0	0	0	0
1	|Input_Wrapper_Controller:controller|	8 (8)	3 (3)	0	0	0	0	0	0
2	|Input_Wrapper_Datapath:Dp|	7 (0)	34 (0)	0	0	0	0	0	0
1	|Counter2bit:counter|	3 (3)	2 (2)	0	0	0	0	0	0
2	|Dcd2to4:dcd|	4 (4)	0 (0)	0	0	0	0	0	0
3	|Register8bit:rg1|	0 (0)	8 (8)	0	0	0	0	0	0
4	|Register8bit:rg2|	0 (0)	8 (8)	0	0	0	0	0	0
5	|Register8bit:rg3|	0 (0)	8 (8)	0	0	0	0	0	0
6	|Register8bit:rg4|	0 (0)	8 (8)	0	0	0	0	0	0
3	|Output_Wrapper:output_wrapper|	26 (0)	4 (0)	0	0	0	0	0	0
1	|Output_Wrapper_Controller:controller|	7 (7)	2 (2)	0	0	0	0	0	0

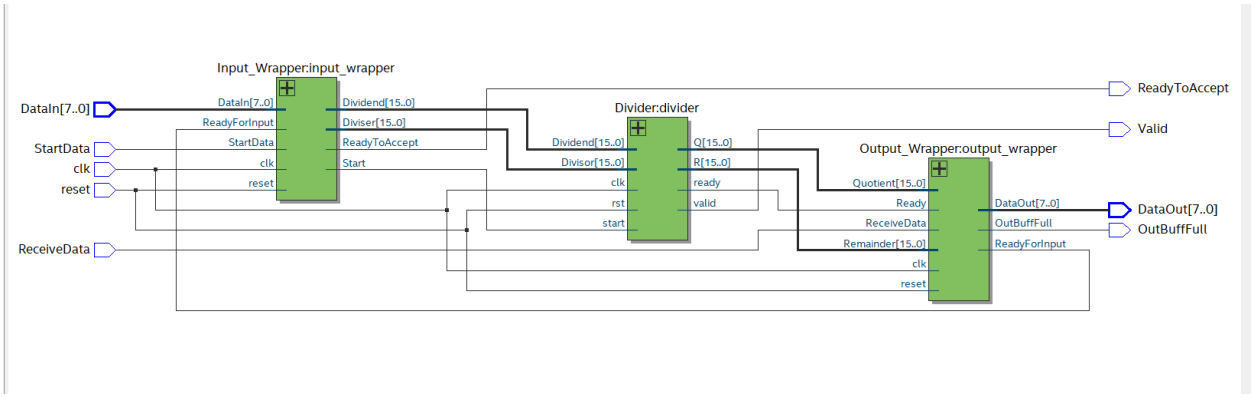
Zero important warnings



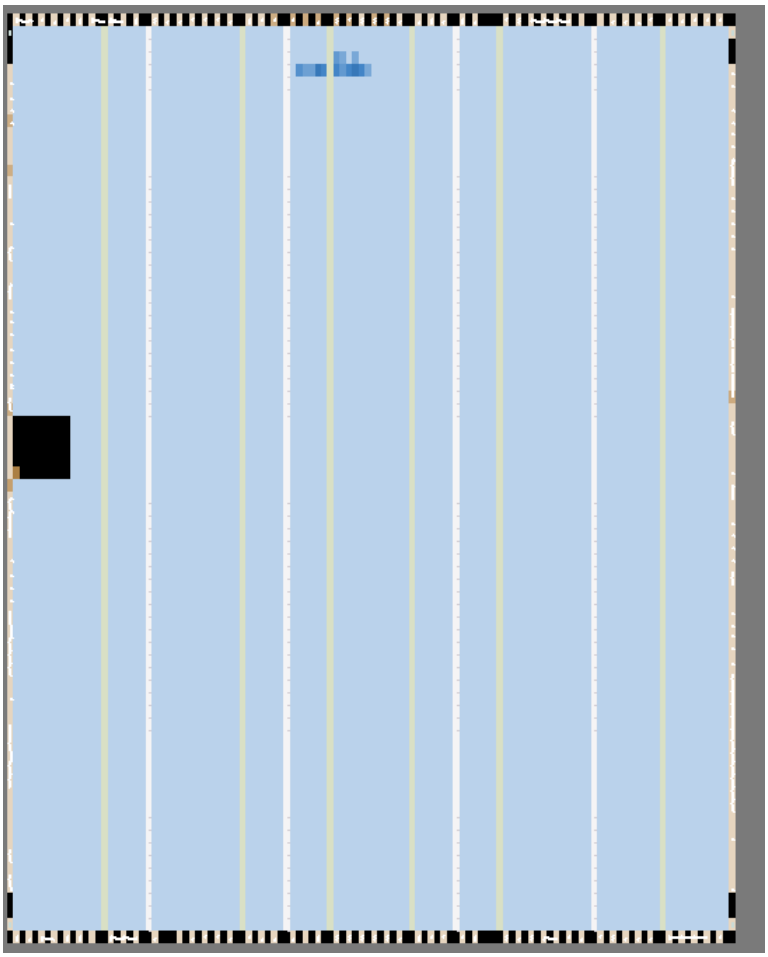
Maximum Frequency of clock:



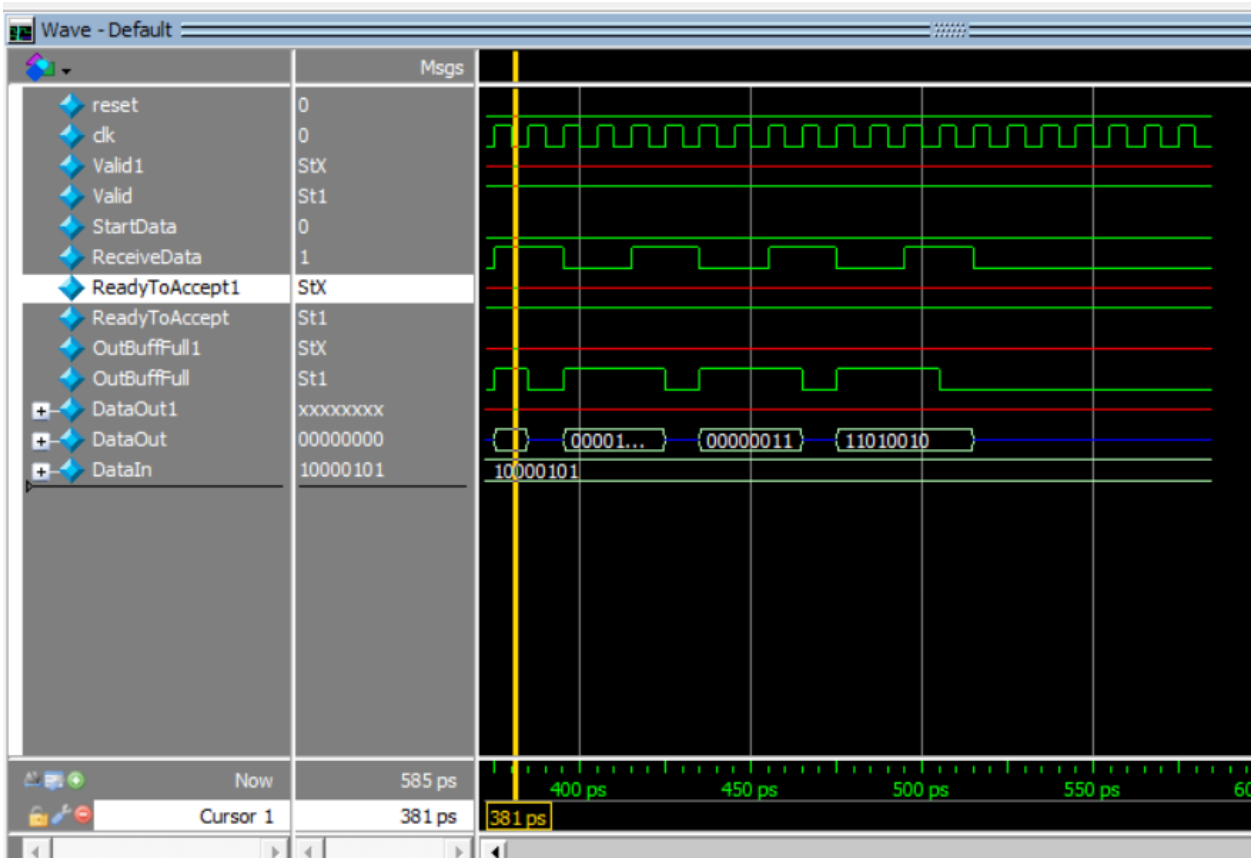
RTL OVERVIEW:



CHIP UTILIZATION:



POST SYNTHESIS SIMULATION:



For whatever reason I can't seem to find why the output is x, presynthesis is correct and no warnings in the syntheses part, Ms.Zahra Mahdavi and I spent 5 hours debugging it and couldn't find out why 😊

Synthesis and post synthesis simulation of the divider(CA5) was correct.