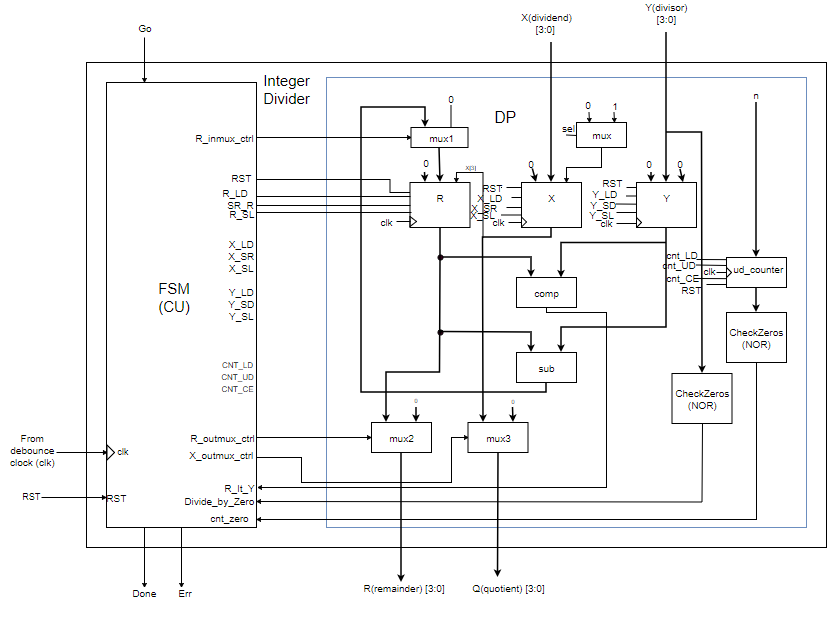


**Introduction**

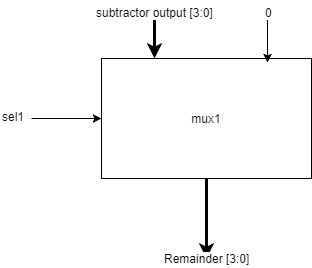
The purpose of this lab was to enhance our understanding in system-level design, verification, and FPGA validation, as well as to be familiar with the Engineering Design Automation tools used in the lab. In this lab, we designed a 4-bit unsigned integer divider using a datapath and a finite state machine. The finite state machine was sued as a control unit that generated control signals used for steering the datapath’s operation. Self-checking testbenches were designed for both the control unit and the datapath. A self-checking testbench was also designed for the whole system (control unit and datapath). The system was implemented on the Nexys4 DDR FPGA board.

**Design Methodology**

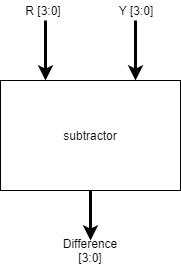
The control unit and datapath that were designed in this lab can be seen in the image below.



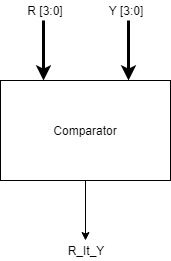
*Figure 1*: complete system block diagram, with the unsigned integer divider, finite state machine (control unit), and datapath.

**

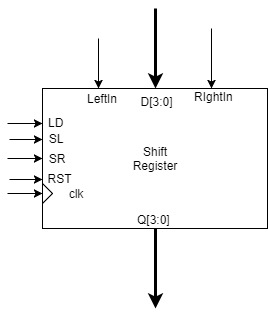
*Figure 2*: First multiplexer of the datapath. The output goes to the remainder’s shift register.



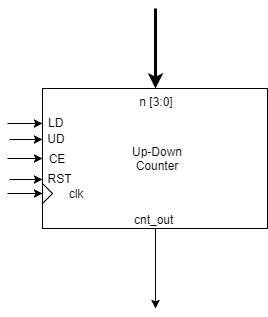
*Figure 3*: subtractor for obtaining the difference between the remainder and the divisor



*Figure 4*: comparator used for comparing the remainder and divisor



*Figure 5*: 4- bit shift register used for the remainder, dividend, and divisor



*Figure 6*: counter used for cnt\_out signal

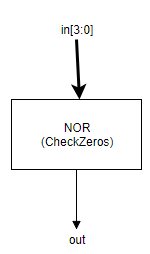
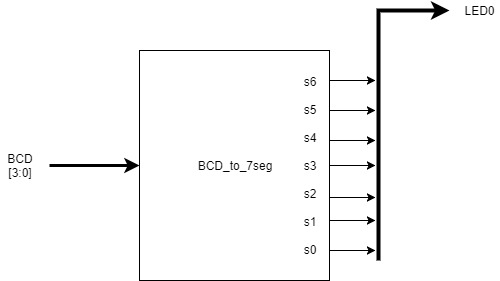


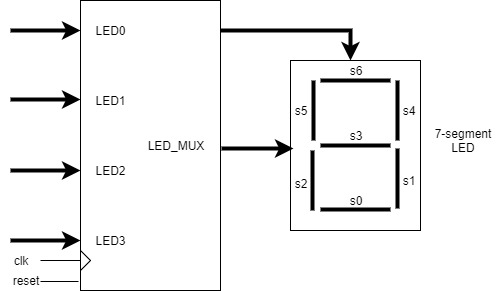
Figure 7: NOR module used to check for zeros



*Figure 8*: BCD to 7-segment module block diagram

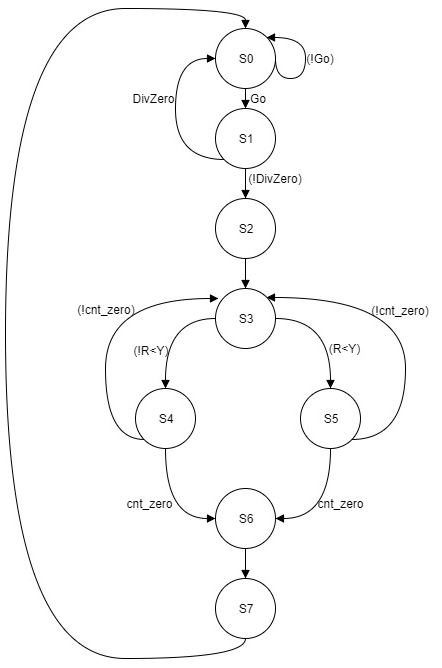


*Figure 9*: clock generator module block diagram

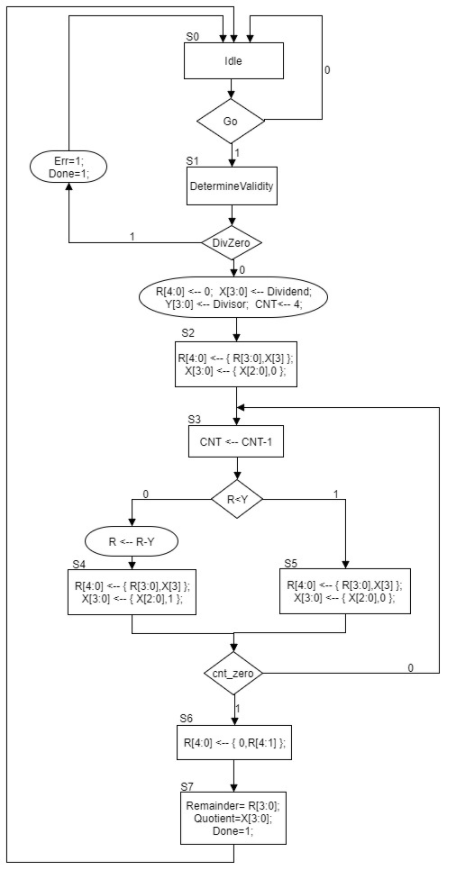


*Figure 10*: LED\_MUX module block diagram with one 7-segment LED

The datapath contains three shift registers: one for the remainder (remainder is selected through a mux), another for the dividend (dividend is also selected through a mux), and the last one for the divisor. The remainder and divisor go through a comparator, which outputs a flag indicating which value is greater. The same values go through a subtractor, whose output goes to the mux connected to the remainder’s shift register. The remainder and dividend are the outputs of the whole system, which the dividend outputting as the quotient after it goes through the shift register. The datapath also includes a counter that outputs a status signal to the control unit.



*Figure 11*: Next State Transition Diagram for the Finite State Machine



*Figure 12*: ASM diagram

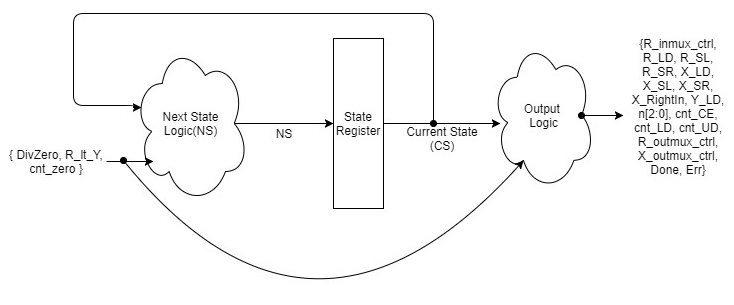


Figure 13: FSM structural diagram

Initially, the 5-bit remainder is set to 0. CNT, which is the value of the counter, is set to 4. Then the first four bits of the remainder are concatenated with the fourth bit of the dividend. This value is now the new remainder. At the same time, the first three bits of the original dividend are concatenated with 0. This value is now the new dividend. CNT is counted down by 1. Then, if the remainder is less than the divisor, the same concatenation that occurred with the original remainder and divisor will occur with this remainder and dividend. If the remainder is greater than the divisor, the same concatenation that occurred with the original remainder will occur with this remainder, and the first three bits of the dividend will be concatenated with 1. This value will be the new dividend. After this, if the value of CNT is not 0, the process will start again from comparing the remainder and divisor. If the value of CNT is 0, the last 5 bits of the remainder will be concatenated with 0, and this value will be the new remainder. Finally, the remainder and dividend (quotient) will be outputted, as well as a “done” flag. We can see the structure of how this finite state machine works in Figure 13

Table 1: ASM output chart

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Inputs** | | | **Outputs** | | | | | | | | | | | | | | | | |
| **CS** | **DivZero** | **R\_lt\_Y** | **R\_inmux\_ctrl** | **R\_LD** | **R\_SL** | **R\_SR** | **X\_LD** | **X\_SL** | **X\_SR** | **X\_RightIn** | **Y\_LD** | **N** | **cnt\_CE** | **cnt\_LD** | **cnt\_UD** | **R\_outmux\_ctrl** | **X\_outmux\_ctrl** | **Done** | **Err** |
| S0 | - | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S1 | 0 | - | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 100 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 000 | 0 | - | - | 0 | 0 | 1 | 1 |
| S2 | - | - | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 000 | 0 | - | - | 0 | 0 | 0 | 0 |
| S3 | - | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| - | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| S4 | - | - | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 000 | 0 | - | - | 0 | 0 | 0 | 0 |
| S5 | - | - | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 000 | 0 | - | - | 0 | 0 | 0 | 0 |
| S6 | - | - | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 000 | 0 | - | - | 0 | 0 | 0 | 0 |
| S7 | - | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 000 | 0 | - | - | 1 | 1 | 1 | 0 |

Table 2: List of modules used

|  |  |
| --- | --- |
| File Name | Description |
| *DP.v* | datapath design code for the four bit integer divider |
| *ShiftReg.v* | parameterized Shift register module for the datapath registers R,X, and Y |
| *comp.v* | Comparator module used to compare R to Y |
| *sub.v* | Subtractor module for R[3:0]-Y[3:0] |
| *MUX2.v* | parameterized 2 to 1 mux for the datapath output |
| *NOR.v* | NOR module used to check if results are zero |
| *UD\_counter.v* | Up-Down Counter module used for the loop |
| *DP\_tb.v* | self-checking testbench for the integer divider datapath |
| *CU.v* | Finite State Machine or Control Unit of the integer divider |
| *CU\_tb.v* | self-checking testbench for the control unit of the integer divider |
| *Integer\_divider.v* | design module for the small calculator --connecting DP with CU |
| *Int\_div\_tb.v* | self-checking testbench for the integer divider |
| *Int\_div\_fpga.v* | top-level module for the integer divider FPGA implementation |
| *clk\_gen.v* | utility module to generate the clock for the FPGA |
| *button\_debouncer.v* | utility module for the debouncer on the FPGA |
| *bcd\_to\_7seg.v* | utility module to determine the 7-segment display output |
| *led\_mux.v* | utility module for signals to be displayed on the 7-segment display |
| *Int\_div\_fpga.xdc* | constraint file for the unsigned integer division FPGA validation |

**Simulation Results**

The behavioral simulations of the datapath design, the control unit design, and the integer divider design were all successful. From eyeball testing, we were able to observe that the quotients and remainders were correct on the datapath and integer divider simulations. The states and the control unit outputs also seemed to be changing according to the ASM chart correctly. From the self-checking messages, we were able to see that there were no errors in the outputs of the datapath, control unit, and the integer divider.

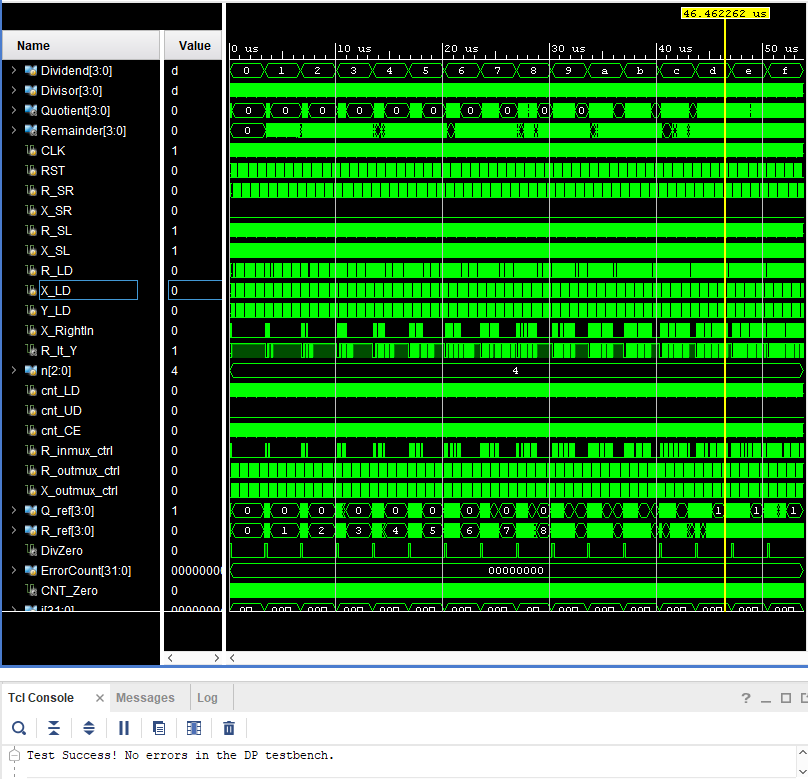


Figure 14:Datapath full simulation waveform with self-checking message displayed

From eyeball testing the datapath simulation, we can see that the outputs of the datapath are correct, where the output muxes are only enabled when the correct answer is finalized. 13/11 gives a quotient of 1 and a remainder of 2 and 13/12 gives a quotient of 1 and a remainder of 1, which is reflected in the testbench below.

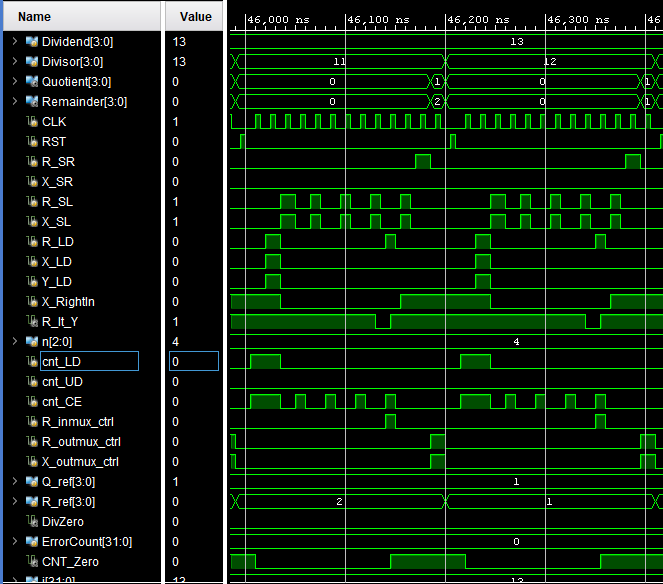


Figure 15:Close view of the datapath simulation

From the self-checking messages of the control unit simulation, we can see that the outputs of the control unit at each state appear to be correct. From eyeball testing, the states seem to be correctly changing according to the ASM chart in figure 12. When the divisor is zero, then Err will be activated at S1 and go back to S0, otherwise it will continue on down the ASM chart. Overall, the control unit simulation was a success.

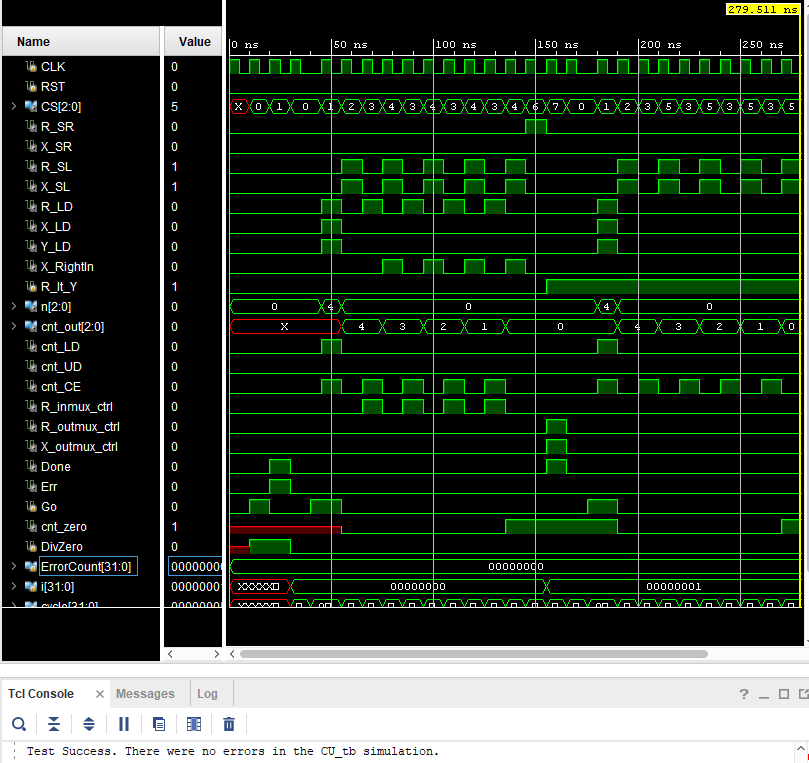


Figure 16: Control unit simulation with self-checking messages displayed

From the self-checking message of the integer divider simulation, we can see that the quotient and remainder were all outputted correctly to their corresponding dividends and divisors.

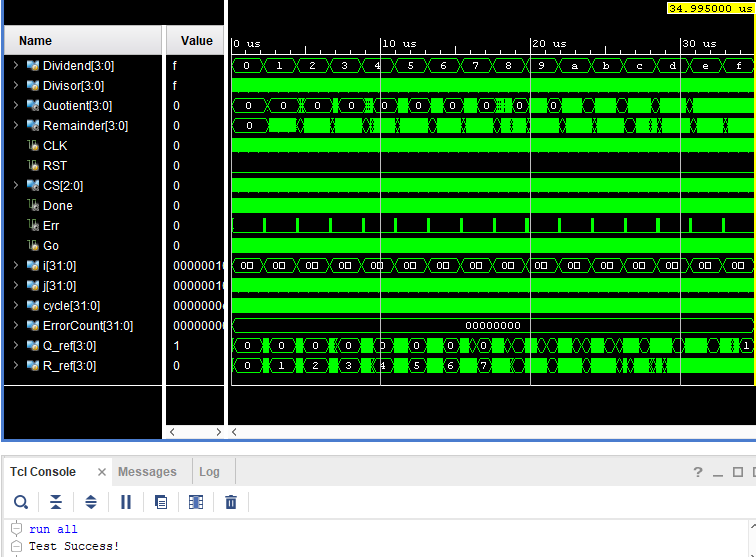


Figure 17:Integer Division full simulation waveforms with self-checking message displayed

From the figure below, we can see that the states are changing according to the ASM chart and that the quotients and remainders are outputting the correct result.

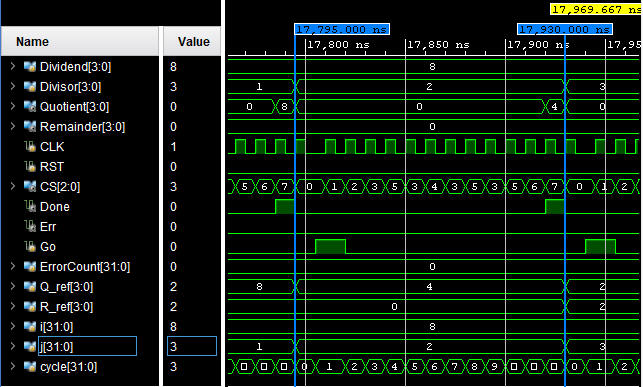


Figure 18: Close view of Integer Division simulation

**FPGA Validation**

The FPGA validation for the integer divider was a success. The inputs *Go, Reset, and Debouncer\_button* were connected to buttons on the Nexys4 while the input *Dividend* and *Divisor* inputs were connected to the switches on the Nexys4 as shown in the figure below. When the *Reset* button was pressed, the system would go back to S0 and the outputs would be cleared. Each time the *Dividend* and *Divisor* are done being configured, the user can then commence with pressing *Go* and the *Debouncer\_button* at the same time so it will switch to S1, which is reflected in the ASM chart provided in figure 12.

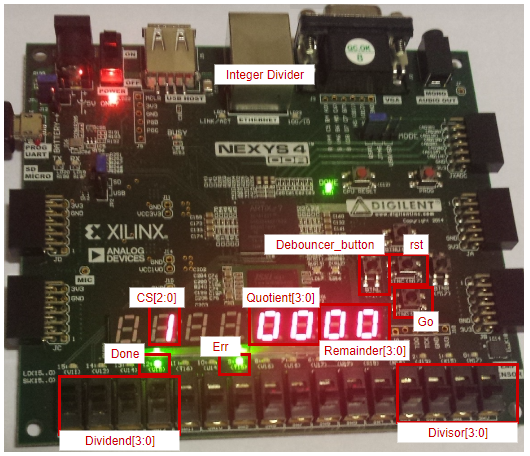


Figure 19: Integer Divider FPGA implementation with input/output labels

The FPGA correctly goes through all the relevant states of the ASM diagram from figure 12. Images of the system being at different states are included below.

When the system is not at the final state, or State 7, the Quotient and Remainder will always display 0, because the output muxes are designed for this purpose.

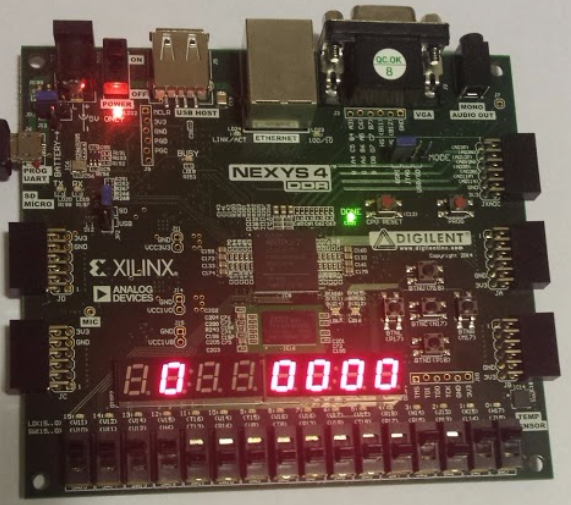


Figure 20: State 0, where the output muxes are at 0

Depending on whether the Divisor is equal to 0, the system would be different at State 1, where it would either output Done and Err, or it would load the relevant inputs to the registers according to the ASM chart. The two different State 1’s will be shown below.

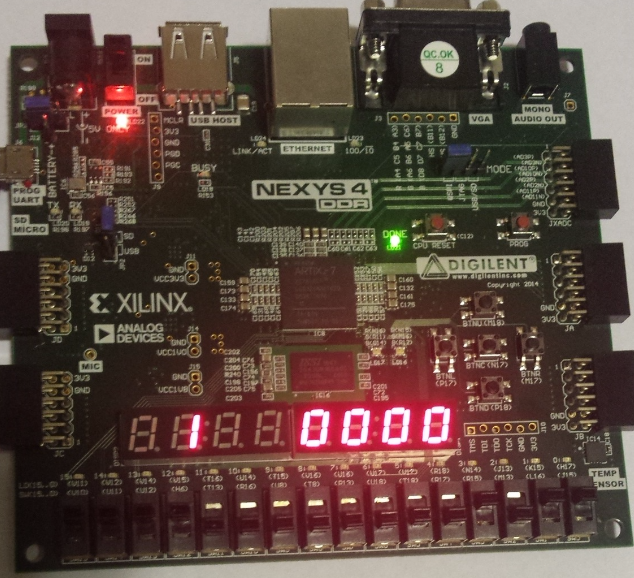


Figure 21: State 1, when Divisor is not 0



Figure 22: State 1, when the Divisor is zero

The figures provided next are provided to show what the system would look like at different states of the FSM. The output muxes are still disabled since we are not at State 7 yet.

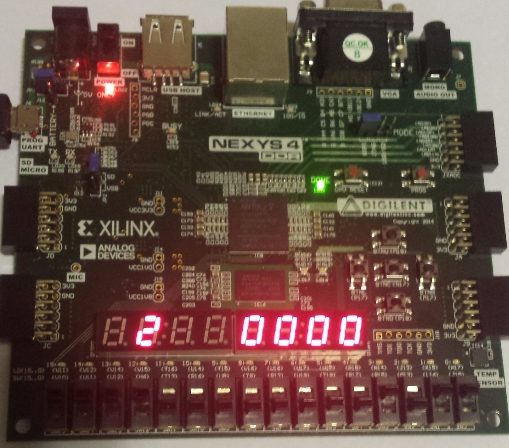


Figure 23: State 2, with output muxes not enabled

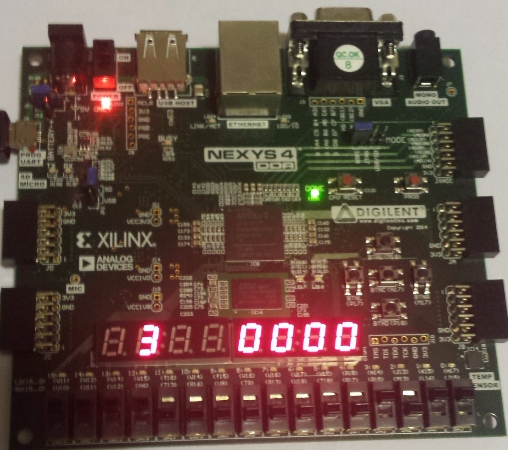


Figure 24: State 3, with output muxes not enabled

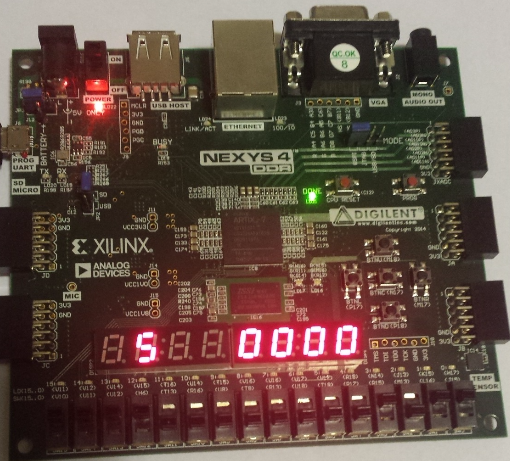


Figure 25: State 5, with output muxes not enabled

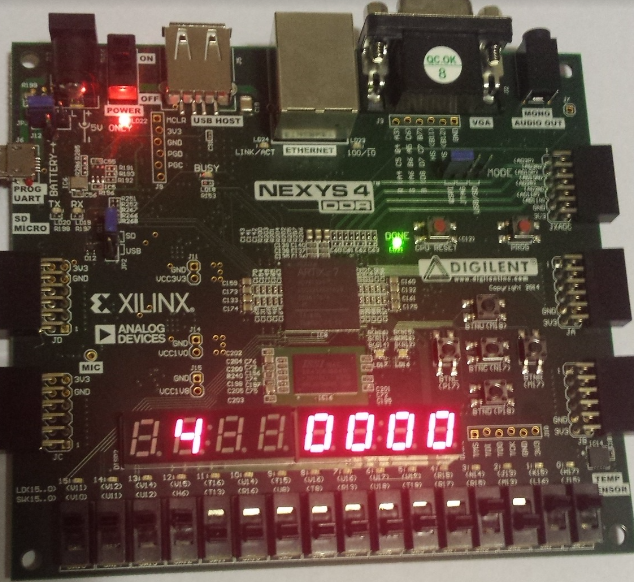


Figure 26: State 4, with output muxes not enabled

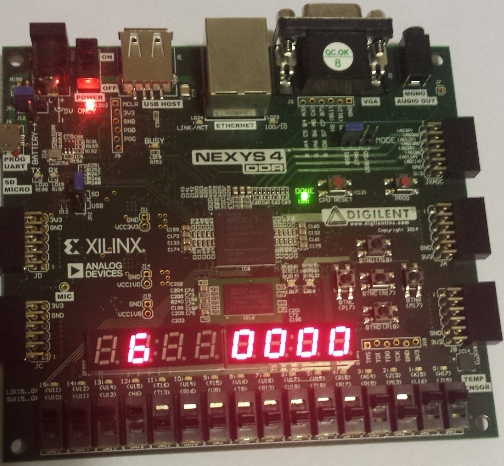


Figure 27: State 6, with output muxes not enabled

The figure below provides an example of what State 7 should look like. The Done flag is activated, the 7-segment display shows that the system is at state 7, and the quotient and remainder are correct. 15/11 is equal to a quotient of 1 and a remainder of 4, which is reflected on the 7-segment display.

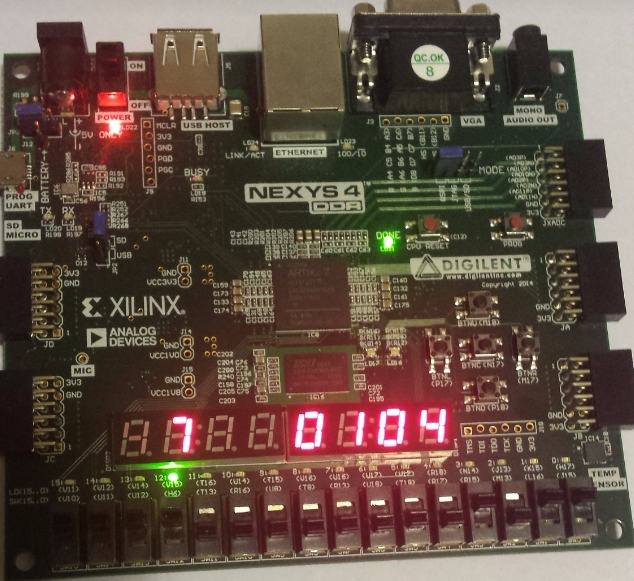


Figure 28: Dividend=15, Divisor=11 outputting Quotient=1, Remainder=4 at state7

**Conclusion**

The design of the datapath, control unit, and the integrated system of the unsigned integer divider was a success. The testbenches were all completed and ran with a self-checking “Test Success” message included after the simulation was done running without errors. From eyeball testing, the testbench results also seem correct. The FPGA validation of the integer division was also a success, with the outputs displayed on the 7-segment display or on the LEDs which the inputs were switches or buttons.

There were some issues with Vivado where “initial” was used in a NOR module and Vivado ignored the module during synthesis; however it was fixed by replacing it with an “always@(input)”. This might be because initial only goes through the code once, which is usually used for testbenches and not the actual hardware.

Overall, the lab was a success and everything ran accordingly to the ASM and FSM design.

**Appendix**

1. **File *DP.v*- datapath design code for the four bit integer divider**

module DP(Dividend, Divisor, Quotient, Remainder,

CLK, RST, R\_SR, X\_SR, R\_SL, X\_SL,

R\_LD, X\_LD, Y\_LD, X\_RightIn,

R\_lt\_Y, CNT\_Zero,

n, cnt\_LD, cnt\_UD, cnt\_CE,

R\_inmux\_ctrl, R\_outmux\_ctrl, X\_outmux\_ctrl, DivZero

);

input [3:0] Dividend, Divisor;

output wire [3:0]Quotient, Remainder;

input CLK, RST;

input wire R\_SR, X\_SR,R\_SL, X\_SL;

input wire R\_LD, X\_LD, Y\_LD;

input X\_RightIn;

output wire R\_lt\_Y; //flag

input wire [2:0]n;

input wire cnt\_LD, cnt\_UD, cnt\_CE; //Counter ctrl

input wire R\_inmux\_ctrl, R\_outmux\_ctrl, X\_outmux\_ctrl;

output DivZero, CNT\_Zero;

wire [2:0] cnt\_out;

wire [4:0] R\_D;

wire [3:0] X\_D,Y\_D;

wire [4:0] R\_Q;

wire [3:0] X\_Q, Y\_Q;

wire [3:0]R\_minus\_Y;

wire X\_Right\_In;

NOR #4 Divisor0(.in(Divisor),.out(DivZero));

MUX2to1 #5 R\_inmux(.in1({1'b0,R\_minus\_Y}), .in2(5'd0), .s2(R\_inmux\_ctrl), .m2out(R\_D));

ShiftReg #5 R( .CLK(CLK), .RST(RST), .SR(R\_SR), .SL(R\_SL), .LD(R\_LD),

.LeftIn(1'b0), .RightIn(X\_Q[3]), .D(R\_D), .Q(R\_Q));

ShiftReg #4 X ( .CLK(CLK), .RST(RST), .SR(X\_SR), .SL(X\_SL), .LD(X\_LD),

.LeftIn(0), .RightIn(X\_Right\_In), .D(Dividend), .Q(X\_Q));

MUX2to1 #1 X\_Rin(.in1(1'b1), .in2(1'b0), .s2(X\_RightIn), .m2out(X\_Right\_In));

ShiftReg #4 Y ( .CLK(CLK), .RST(RST), .SR(0), .SL(0), .LD(Y\_LD),

.LeftIn(0), .RightIn(0), .D(Divisor), .Q(Y\_Q));

comparator #4 comp(.A(R\_Q[3:0]), .B(Y\_Q), .lt(R\_lt\_Y));

sub #4 R\_sub\_Y(.A(R\_Q[3:0]), .B(Y\_Q), .C(R\_minus\_Y));

UD\_counter #3 UD\_cnt ( .D(n), .LD(cnt\_LD), .UD(cnt\_UD),

.CE(cnt\_CE), .CLK(CLK), .RST(RST), .Q(cnt\_out));

NOR #3 CNT0(.in(cnt\_out),.out(CNT\_Zero));

MUX2to1 #4 R\_outmux(.in1(R\_Q[3:0]), .in2(4'd0), .s2(R\_outmux\_ctrl), .m2out(Remainder));

MUX2to1 #4 X\_outmux(.in1(X\_Q), .in2(4'd0), .s2(X\_outmux\_ctrl), .m2out(Quotient));

endmodule

1. **File *ShiftReg.v* - parameterized Shift register module for the datapath registers R,X, and Y**

`timescale 1ns / 1ps

module ShiftReg (CLK, RST, SL, SR, LD, LeftIn, RightIn, D, Q);

parameter Data\_width = 4;

input CLK, RST, SL, SR, LD, LeftIn, RightIn;

input [Data\_width-1:0] D;

output reg [Data\_width-1:0] Q;

always @(posedge CLK)

begin

if (RST)

Q = 0;

else if (LD)

Q = D;

else if (SL) // shift left

begin

Q [Data\_width-1:1] = Q [Data\_width-2:0];

Q [0] = RightIn;

end

else if (SR) // shift right

begin

Q [Data\_width-2:0] = Q [Data\_width-1:1];

Q [Data\_width -1] = LeftIn;

end

else Q [Data\_width-1:0] = Q [Data\_width-1:0];

end

endmodule

1. **File *comp.v* - Comparator module used to compare R to Y**

module comparator(A, B, lt);

parameter Data\_width = 4;

input [Data\_width - 1:0] A;

input [Data\_width - 1:0] B;

output reg lt;

always @ (A or B)

begin

lt <= 1'b0;

if (A < B)

lt <= 1'b1;

end

endmodule

1. **File *sub.v* - Subtractor module for R[3:0]-Y[3:0]**

`timescale 1ns / 1ps

module sub(A, B, C);

parameter Data\_width = 4;

input [Data\_width - 1:0] A;

input [Data\_width - 1:0] B;

output reg [Data\_width - 1:0]C;

always @ (A or B)

begin

C = A - B;

end

endmodule

1. **File *MUX2.v* - parameterized 2 to 1 mux for the datapath input/outputs**

module MUX2to1 (in1, in2, s2, m2out);

parameter Data\_width = 4;

input[Data\_width-1:0] in1, in2;

input s2;

output reg[Data\_width-1:0] m2out;

always@ (in1, in2, s2)

begin

if (s2)

m2out = in1;

else

m2out = in2;

end

endmodule//MUX2

1. **File *NOR.v* - NOR module used to check if results are zero**

`timescale 1ns / 1ps

module NOR(in, out );

parameter Data\_width=4;

input [Data\_width-1:0] in;

output reg out;

always@(in)

begin

out=~|in;

end

endmodule

1. **File *UD\_counter.v* -Up-Down Counter module used for the loop**

`timescale 1ns / 1ps

module UD\_counter (D, LD, UD, CE, CLK, RST, Q);

parameter Data\_width = 4; // data size

parameter UP = 1, DOWN = 0;

input LD, UD, CE, CLK, RST;

input [Data\_width-1:0] D;

output reg [Data\_width-1:0] Q;

always @(posedge CLK)

begin

if (RST)

Q = 0;

else if (CE)

begin

if (LD)

Q = D;

else

begin

case(UD)

DOWN: Q = Q - 1;

UP: Q = Q + 1;

endcase

end

end

else

Q = Q;

end

endmodule

1. **File *DP\_tb.v* - self-checking testbench for the integer divider datapath**

`timescale 1ns / 1ps

module DP\_tb();

reg [3:0] Dividend, Divisor;

wire [3:0]Quotient, Remainder;

reg CLK, RST;

reg R\_SR, X\_SR, R\_SL, X\_SL;

reg R\_LD, X\_LD, Y\_LD;

reg X\_RightIn;

wire R\_lt\_Y; //flag

reg [2:0]n;

//wire [2:0] cnt\_out; //Counter in/out

wire CNT\_zero;

reg cnt\_LD, cnt\_UD, cnt\_CE; //Counter ctrl

reg R\_inmux\_ctrl, R\_outmux\_ctrl, X\_outmux\_ctrl;

reg [3:0] Q\_ref,R\_ref;

wire DivZero;

integer i,j,k,ErrorCount;

DP DP\_DUT(Dividend, Divisor, Quotient, Remainder,

CLK, RST, R\_SR, X\_SR, R\_SL, X\_SL,

R\_LD, X\_LD, Y\_LD, X\_RightIn,

R\_lt\_Y, CNT\_Zero,

n, cnt\_LD, cnt\_UD, cnt\_CE,

R\_inmux\_ctrl, R\_outmux\_ctrl, X\_outmux\_ctrl, DivZero

);

initial

begin

ErrorCount=0;

for(i=0;i<16;i=i+1)

begin

Dividend=i;

for(j=0;j<16;j=j+1)

begin

Divisor=j;

//Dividend=15;

//Divisor=5;

Q\_ref=Dividend/Divisor;

R\_ref=Dividend%Divisor;

CLK=0;

//S0

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0; cnt\_UD=0;

R\_outmux\_ctrl=0; X\_outmux\_ctrl=0;

n=4;

RST=0;#5;RST=1;#5;RST=0;#5;

cnt\_CE=1; cnt\_LD=1;

#5; CLK=1;#5; CLK=0;#5;

//S1

CLK=0;

R\_inmux\_ctrl=0; R\_LD=1;

Y\_LD=1; X\_LD=1;

cnt\_CE=1; cnt\_LD=1;

#5; CLK=1;#5; CLK=0;#5;

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0;

//S2

R\_SL=1; R\_SR=0; R\_LD=0;

X\_SL=1; X\_SR=0; X\_LD=0;

X\_RightIn=1'b0;

#5; CLK=1;#5; CLK=0;#5;

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0;

for(k=3; k>=0; k=k-1)

begin

//S3

if(R\_lt\_Y)

begin

R\_LD=0;

X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0;

cnt\_CE=1; cnt\_LD=0; cnt\_UD=0;

#5; CLK=1;#5;

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0;

R\_inmux\_ctrl=0;

CLK=0;#5;

//S5

R\_SL=1; R\_SR=0; R\_LD=0;

X\_SL=1; X\_SR=0; X\_LD=0;

X\_RightIn=1'b0;

#5; CLK=1;#5;

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0;

R\_inmux\_ctrl=0;

CLK=0;#5;

end

else

begin //S4

R\_LD=1; R\_inmux\_ctrl=1;

X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0;

cnt\_CE=1; cnt\_LD=0; cnt\_UD=0;

#5; CLK=1;#5;

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0;

R\_inmux\_ctrl=0;

CLK=0;#5;

R\_SL=1; R\_SR=0;

X\_SL=1; X\_SR=0; X\_LD=0;

X\_RightIn=1'b1;

#5; CLK=1;#5;

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0;

R\_inmux\_ctrl=0;

CLK=0;#5;

end

end//endk

R\_LD=0; R\_SL=0; R\_SR=1;

#5; CLK=1;#5; CLK=0;#5;

R\_LD=0; X\_LD=0; Y\_LD=0;

R\_SR=0; X\_SR=0;

R\_SL=0; X\_SL=0; cnt\_CE=0; cnt\_LD=0;

R\_outmux\_ctrl=1; X\_outmux\_ctrl=1;

#5; CLK=1;#5;

if(Quotient!=Q\_ref || Remainder!=R\_ref)

begin

$display("%d/%d (Qref%d,Rref%d)does not equal %d R%d at %d", Dividend, Divisor, Q\_ref, R\_ref, Quotient,Remainder, $time);

ErrorCount=ErrorCount+1;

end

if(Divisor==0 && DivZero!=1)

begin

$display("Divisor is 0, DivZero should be active at %d", $time);

ErrorCount=ErrorCount+1;

end

CLK=0;#5;

end//end j

end//end i

if(ErrorCount!=0)

$display("%d total errors.",ErrorCount);

else

$display("Test Success! No errors in the DP testbench.");

end//end initial

endmodule

1. **File *CU.v* - Control Unit of the integer divider**

`timescale 1ns / 1ps

module CU(input CLK, rst, Go, R\_lt\_Y, DivZero, cnt\_zero, output reg Done, Err, reg [2:0]CS,

output reg R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD,output reg [2:0] n, output reg cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, X\_RightIn);

reg [2:0]NS;

parameter S0=3'b000,

S1=3'b001,

S2=3'b010,

S3=3'b011,

S4=3'b100,

S5=3'b101,

S6=3'b110,

S7=3'b111;

initial begin

CS=S0;

end

always@(CS,Go,cnt\_zero,DivZero,R\_lt\_Y)

begin

case(CS)

S0: begin

if(Go) NS<=S1;

else NS<=S0;

end

S1: begin

if(DivZero) NS<=S0;

else NS<=S2;

end

S2: NS<=S3;

S3:begin

if(R\_lt\_Y) NS<=S5;

else NS<=S4;

end

S4: begin

if(cnt\_zero) NS<=S6;

else NS<=S3;

end

S5:begin

if(cnt\_zero) NS<=S6;

else NS<=S3;

end

S6: NS<=S7;

S7: NS<=S0;

endcase

end

always@(posedge CLK,posedge rst)

begin

if(rst) CS<=S0;

else CS<=NS;

end

always@(CS,DivZero,R\_lt\_Y)

begin

case(CS)

S0:begin

X\_RightIn=0; Err=0;

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_0\_0\_0\_0\_0\_0\_000\_0\_0\_0\_0\_0\_0;

end

S1:begin

if(DivZero) begin

X\_RightIn=0;Err=1;

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_0\_0\_0\_0\_0\_0\_000\_0\_0\_0\_0\_0\_1;

Err=1;

end

else

begin

X\_RightIn=0;Err=0;

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

<=17'b0\_1\_0\_0\_1\_0\_0\_1\_100\_1\_1\_0\_0\_0\_0;

end

end

S2:begin

X\_RightIn=0; Err=0;

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_1\_0\_0\_1\_0\_0\_000\_0\_0\_0\_0\_0\_0;

end

S3:

begin

if(R\_lt\_Y) begin

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_0\_0\_0\_0\_0\_0\_000\_1\_0\_0\_0\_0\_0;

Err=0; X\_RightIn=0;

end

else begin

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b1\_1\_0\_0\_0\_0\_0\_0\_000\_1\_0\_0\_0\_0\_0;

Err=0; X\_RightIn=0;

end

end

S4: begin

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_1\_0\_0\_1\_0\_0\_000\_0\_0\_0\_0\_0\_0;

X\_RightIn=1;Err=0;

end

S5:begin

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_1\_0\_0\_1\_0\_0\_000\_0\_0\_0\_0\_0\_0;

X\_RightIn=0;Err=0;

end

S6:begin

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_0\_1\_0\_0\_0\_0\_000\_0\_0\_0\_0\_0\_0;

X\_RightIn=0;Err=0;

end

S7:begin

{R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done}

=17'b0\_0\_0\_0\_0\_0\_0\_0\_000\_0\_0\_0\_1\_1\_1;

X\_RightIn=0;Err=0;

end

endcase

end

endmodule

1. **File *CU\_tb.v* - self-checking testbench for the control unit of the integer divider**

`timescale 1ns / 1ps

module CU\_testbench();

reg CLK, RST;

wire [2:0] CS;

wire R\_SR, X\_SR, R\_SL, X\_SL;

wire R\_LD, X\_LD, Y\_LD;

wire X\_RightIn;

reg R\_lt\_Y; //flag

wire [2:0]n;

wire [2:0] cnt\_out; //Counter in/out

wire cnt\_LD, cnt\_UD, cnt\_CE; //Counter ctrl

wire R\_inmux\_ctrl, R\_outmux\_ctrl, X\_outmux\_ctrl;

wire Done,Err;

reg Go;

integer i, cycle, ErrorCount;

reg DivZero;

CU CU\_DUT(CLK, RST, Go, R\_lt\_Y, DivZero, cnt\_zero, Done, Err, CS,

R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, X\_RightIn);

UD\_counter #3 UD\_cu ( .D(n), .LD(cnt\_LD), .UD(cnt\_UD),

.CE(cnt\_CE), .CLK(CLK), .RST(RST), .Q(cnt\_out));

NOR #3 cu\_nor(cnt\_out, cnt\_zero);

initial

begin

CLK=0;

ErrorCount=0;

Go=0;

R\_lt\_Y=0;

RST=0;

CLK=1; #5; CLK=0; #5;

DivZero=1;

Go=1;

CLK=1; #5; CLK=0; #5;

Go=0;

// RST=1;#5; RST=0; #5;

CLK=1; #5; CLK=0; #5;

DivZero=0;

for(i=0;i<2;i=i+1)

begin

R\_lt\_Y=i;

for(cycle=0;cycle<12;cycle=cycle+1)

begin

if(CS==0)begin Go=1;#5; end

else Go=0;

CLK=1;#5;

case(CS)

0: begin

if({R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_0\_0\_0\_0\_0\_0\_0\_000\_0\_0\_0\_0\_0\_0\_0 )//S0

begin

$display("Error at S0 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

1: begin

if(!DivZero && {R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_1\_0\_0\_1\_0\_0\_0\_1\_100\_1\_1\_0\_0\_0\_0\_0)//S1

begin

$display("Error at S1 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

else if(DivZero && {R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_0\_0\_0\_0\_0\_0\_0\_000\_0\_0\_0\_0\_0\_1\_1)

begin

$display("Error at S1 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

2: begin

if({R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_1\_0\_0\_1\_0\_0\_0\_000\_0\_0\_0\_0\_0\_0\_0)//S2

begin

$display("Error at S2 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

3: begin

if(R\_lt\_Y==0 && {R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b1\_1\_0\_0\_0\_0\_0\_0\_0\_000\_1\_0\_0\_0\_0\_0\_0)//S3\_0

begin

$display("Error at S3 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

else if(R\_lt\_Y==1 && {R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_0\_0\_0\_0\_0\_0\_0\_000\_1\_0\_0\_0\_0\_0\_0)//S3\_1

begin

$display("Error at S3 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

4: begin

if({R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_1\_0\_0\_1\_0\_1\_0\_000\_0\_0\_0\_0\_0\_0\_0)//S4

begin

$display("Error at S4 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

5: begin

if({R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_1\_0\_0\_1\_0\_0\_0\_000\_0\_0\_0\_0\_0\_0\_0)//S5

begin

$display("Error at S5 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

6: begin

if({R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_0\_1\_0\_0\_0\_0\_0\_000\_0\_0\_0\_0\_0\_0\_0)//S6

begin

$display("Error at S6 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

7: begin

if({R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,Err} !=19'b0\_0\_0\_0\_0\_0\_0\_0\_0\_000\_0\_0\_0\_1\_1\_1\_0)//S7

begin

$display("Error at S7 %b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b,%b time%d",R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR,X\_RightIn, Y\_LD, n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, Done,$time);

ErrorCount=ErrorCount+1;

end

end

endcase

CLK=0; #5;

end

end//end R\_lt\_Y

if(ErrorCount==0)

$display("Test Success. There were no errors in the CU\_tb simulation.");

else $display("%d total errors",ErrorCount);

$stop;

end//end initial

endmodule

1. **File Integer\_divider.v - design module for the integer divider --connecting DP with CU**

`timescale 1ns / 1ps

module Integer\_divider(CLK, RST, Go, Done, Err, CS,

Dividend, Divisor, Quotient, Remainder

);

input wire CLK, RST;

input wire Go;

output wire [2:0] CS;

input wire [3:0] Dividend, Divisor;

output wire [3:0]Quotient, Remainder;

output wire Done, Err;

wire R\_SR, X\_SR, R\_SL, X\_SL;

wire R\_LD, X\_LD, Y\_LD;

wire X\_RightIn;

wire R\_lt\_Y; //flag

wire [2:0]n;

wire [2:0] cnt\_out; //Counter in/out

wire cnt\_zero;

wire cnt\_LD, cnt\_UD, cnt\_CE; //Counter ctrl

wire R\_inmux\_ctrl, R\_outmux\_ctrl, X\_outmux\_ctrl;

wire DivZero;

DP DP\_Div(Dividend, Divisor, Quotient, Remainder,

CLK, RST, R\_SR, X\_SR, R\_SL, X\_SL,

R\_LD, X\_LD, Y\_LD, X\_RightIn,

R\_lt\_Y, cnt\_zero,

n, cnt\_LD, cnt\_UD, cnt\_CE,

R\_inmux\_ctrl, R\_outmux\_ctrl, X\_outmux\_ctrl, DivZero

);

CU CU\_div(CLK, RST, Go, R\_lt\_Y, DivZero, cnt\_zero, Done, Err, CS,

R\_inmux\_ctrl, R\_LD, R\_SL, R\_SR, X\_LD, X\_SL, X\_SR, Y\_LD,

n, cnt\_CE, cnt\_LD, cnt\_UD, R\_outmux\_ctrl, X\_outmux\_ctrl, X\_RightIn);

endmodule

1. **File *Int\_div\_tb.v* - self-checking testbench for the integer divider**

`timescale 1ns / 1ps

module Int\_div\_tb();

reg [3:0] Dividend, Divisor;

wire [3:0]Quotient, Remainder;

reg CLK, RST;

wire [2:0] CS;

wire Done,Err;

reg Go;

integer i,j,cycle,ErrorCount;

reg [3:0] Q\_ref,R\_ref;

Integer\_divider div\_DUT(CLK, RST, Go, Done, Err, CS,

Dividend, Divisor, Quotient, Remainder

);

initial

begin

ErrorCount=0;

CLK=0;

RST=0;

for(i=0;i<16;i=i+1)

begin

Dividend=i;

for(j=0;j<16;j=j+1)

begin

Divisor=j;

Q\_ref=Dividend/Divisor;

R\_ref=Dividend%Divisor;

for(cycle=0;cycle<13;cycle=cycle+1)

begin

if(CS==0)begin Go=1;#5; end

else Go=0;

CLK=1;#5;

CLK=0;#5;

if(Done && (Quotient!=Q\_ref || Remainder!=R\_ref))

begin

$display("%d/%d (Qref%d,Rref%d)does not equal %d R%d at %d", Dividend, Divisor, Q\_ref, R\_ref, Quotient,Remainder, $time);

ErrorCount=ErrorCount+1;

end

end//end cycle

end//end j

end//end i

if(ErrorCount!=0)

$display("%d total errors.",ErrorCount);

else

$display("Test Success! ");

end//end initial

endmodule

1. **File *Int\_div\_fpga.v* - top-level module for the small calculator FPGA implementation**

module Int\_div\_fpga(input clk100MHz, rst, Debouncer\_button,

input [3:0]Dividend, [3:0]Divisor,

input Go, output Done,Err,

output [7:0] LEDSEL, [7:0] LEDOUT); //display Product on 7seg

wire [2:0] CS;

wire [7:0] seg\_R\_ones,seg\_R\_tens,seg\_Q\_ones,seg\_Q\_tens, seg\_CS;

wire [3:0]Quotient, Remainder;

reg [4:0] Quotient1,Quotient10, Remainder1, Remainder10;

supply1 [7:0] vcc;

wire DONT\_USE, clk\_5KHz;

wire Debouncer\_out;

always@(Quotient,Remainder)

begin

Remainder1=Remainder%10;

Remainder10=Remainder/10;

Quotient1=Quotient%10;

Quotient10=Quotient/10;

end

Integer\_divider div\_DUT(Debouncer\_out, rst, Go, Done, Err, CS,

Dividend, Divisor, Quotient, Remainder );

clk\_gen autoClock(.clk100MHz( clk100MHz ), .rst( rst ), .clk\_4sec( DONT\_USE ), .clk\_5KHz( clk\_5KHz ));

button\_debouncer #8 Debouncer(.clk( clk\_5KHz),

.button( Debouncer\_button ), /\* Input button from constraints \*/

.debounced\_button( Debouncer\_out ) );

bcd\_to\_7seg BCD\_Q1s (Quotient1, seg\_Q\_ones);

bcd\_to\_7seg BCD\_Q10s (Quotient10, seg\_Q\_tens);

bcd\_to\_7seg BCD\_R1s (Remainder1, seg\_R\_ones);

bcd\_to\_7seg BCD\_R10s (Remainder10, seg\_R\_tens);

bcd\_to\_7seg BCD\_CS (CS, seg\_CS);

led\_mux LED\_mux (clk\_5KHz, rst,

vcc, seg\_CS, vcc, vcc, seg\_Q\_tens, seg\_Q\_ones, seg\_R\_tens, seg\_R\_ones, //activelow so vcc

LEDSEL, LEDOUT);

endmodule

1. **File *clk\_gen.v* - utility module to generate the clock for the FPGA**

module clk\_gen(clk100MHz, rst, clk\_4sec, clk\_5KHz);

input clk100MHz, rst;

output clk\_4sec, clk\_5KHz;

reg clk\_4sec, clk\_5KHz;

integer count, count1;

always@(posedge clk100MHz)

begin

if(rst)

begin

count = 0;

count1 = 0;

clk\_4sec = 0;

clk\_5KHz =0;

end

else

begin

if(count == 200000000)

begin

clk\_4sec = ~clk\_4sec;

count = 0;

end

if(count1 == 10000)

begin

clk\_5KHz = ~clk\_5KHz;

count1 = 0;

end

count = count + 1;

count1 = count1 + 1;

end

end

endmodule // endclk\_gen

1. **File *button\_debouncer.v* - utility module for the debouncer on the FPGA**

module button\_debouncer #(parameter depth = 16) (

input wire clk, /\* 5 KHz clock \*/

input wire button, /\* Input button from constraints \*/

output reg debounced\_button);

localparam history\_max = (2\*\*depth)-1; /\* History of sampled input button \*/

reg [depth-1:0] history;

always @ (posedge clk)

begin

/\* Move history back one sample and insert new sample \*/

history <= { button, history[depth-1:1] };

/\* Assert debounced button if it has been in a consistent state throughout history \*/

debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;

end

endmodule

1. **File *bcd\_to\_7seg.v* - utility module to determine the 7-segment display output**

module bcd\_to\_7seg (input [3:0] BCD, output reg [7:0] s\_1s);

always @ (BCD)

begin

case (BCD)

0: s\_1s = 8'b10001000; //active low outputs

1: s\_1s = 8'b11101101; //s6 s5 s4 ... s0

2: s\_1s = 8'b10100010;

3: s\_1s = 8'b10100100; //display 3 on 7-seg

4: s\_1s = 8'b11000101; //display 4 on 7-seg

5: s\_1s = 8'b10010100; //display 5 on 7-seg

6: s\_1s = 8'b10010000; //and so on...

7: s\_1s = 8'b10101101;

8: s\_1s = 8'b10000000;

9: s\_1s = 8'b10000100;

default: s\_1s = 8'b01111111;

endcase

end

endmodule

1. **File *led\_mux.v* - utility module for signals to be displayed on the 7-segment display**

`timescale 1ns / 1ps

//for multiplexing signals to be displayed on 7seg

module led\_mux(input clk, rst,

input [7:0] LED7, LED6, LED5, LED4, LED3, LED2, LED1, LED0,

output [7:0] LEDSEL, LEDOUT); //which 7 seg and what digit to display

reg [2:0] index;

reg [15:0] led\_ctrl;

assign {LEDSEL, LEDOUT} = led\_ctrl;

always @ (posedge clk) index <= (rst) ? 3'b0 : (index + 3'd1);

always @ (index, LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7)

begin

case (index)

0: led\_ctrl <= {8'b11111110, LED0};

1: led\_ctrl <= {8'b11111101, LED1};

2: led\_ctrl <= {8'b11111011, LED2};

3: led\_ctrl <= {8'b11110111, LED3};

4: led\_ctrl <= {8'b11101111, LED4};

5: led\_ctrl <= {8'b11011111, LED5};

6: led\_ctrl <= {8'b10111111, LED6};

7: led\_ctrl <= {8'b01111111, LED7};

default: led\_ctrl <= {8'b11111111, 8'hFF};

endcase

end

endmodule

1. **File *Int\_div\_fpga.xdc* - constraint file for the integer division FPGA validation**

#Clock

set\_property -dict {PACKAGE\_PIN E3 IOSTANDARD LVCMOS33} [get\_ports {clk100MHz}];

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk100MHz}];

set\_property -dict {PACKAGE\_PIN N17 IOSTANDARD LVCMOS33} [get\_ports {rst}];#BTNC

#Input Debouncer Button

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { Debouncer\_button }];#BTNL

#input Go

set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { Go }]; #btnd

#input Dividend

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { Dividend[0] }]; #sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { Dividend[1] }]; #sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { Dividend[2] }]; #sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { Dividend[3] }]; #sw[15]

#input In2

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { Divisor[0] }]; #sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { Divisor[1] }]; #sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { Divisor[2] }]; #sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { Divisor[3] }]; #sw[3]

#output Done

set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { Done }]; #led[12]

#output Err

set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { Err }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#Which 7-segment display to use

set\_property -dict {PACKAGE\_PIN J17 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[0]}]; #right-most 7-seg display1

set\_property -dict {PACKAGE\_PIN J18 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[1]}];

set\_property -dict {PACKAGE\_PIN T9 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[2]}];

set\_property -dict {PACKAGE\_PIN J14 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[3]}];

set\_property -dict {PACKAGE\_PIN P14 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[4]}];

set\_property -dict {PACKAGE\_PIN T14 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[5]}];

set\_property -dict {PACKAGE\_PIN K2 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[6]}];

set\_property -dict {PACKAGE\_PIN U13 IOSTANDARD LVCMOS33} [get\_ports {LEDSEL[7]}]; #left-most 7-seg display2

#What to display at the 7-seg displays ALL ACTIVE LOW

set\_property -dict {PACKAGE\_PIN K13 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[0]}];

set\_property -dict {PACKAGE\_PIN K16 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[1]}];

set\_property -dict {PACKAGE\_PIN P15 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[2]}];

set\_property -dict {PACKAGE\_PIN L18 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[3]}];

set\_property -dict {PACKAGE\_PIN R10 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[4]}];

set\_property -dict {PACKAGE\_PIN T11 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[5]}];

set\_property -dict {PACKAGE\_PIN T10 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[6]}];

set\_property -dict {PACKAGE\_PIN H15 IOSTANDARD LVCMOS33} [get\_ports {LEDOUT[7]}];