**ECEN 323 – Winter 2020**

Lab #5: Arithmetic

Johnson, Ryan

Section 01

Preliminary

Complete the following table by determining the product of the given unsigned hexadecimal multiplicand and multiplier. Also, determine the decimal equivalent of each value.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Multiplicand** | | **Multiplier** | | **Product** | |
| **Hex** | **Decimal** | **Hex** | **Decimal** | **Hex (64-bits)** | **Decimal** |
| 0x0000013d | 317 | 0x000007e4 | 2020 | 0x9C554 | 640340 |
| 0x34E1CB0A | 887212810 | 0x786D4F31 | 2020429617 | 0x18E06BABF021F2EA | 1792551037905793770 |
| 0xA3102F73 | 2735746931 | 0xC412E59B | 3289572763 | 0x7CE46DB7975499A1 | 8999438590678440353 |

Demonstrate your understanding of the unsigned multiplication algorithm of Figure 3.4 by multiplying the multiplicand '0111' (7) by the multiplier '1101' (13). Provide a table like the one shown above.

| **Cycle** | **Multiplicand** | **Product** |
| --- | --- | --- |
| 0 | 0111 | 0000 1101 |
| 1 | 0111 | 0011 1110 |
| 2 | 0111 | 0001 1111 |
| 3 | 0111 | 0100 0111 |
| 4 | 0111 | 0101 1011 |

Complete the following table by determining the Quotient and Remainder of the given unsigned hexadecimal dividend and divisor. Also, determine the decimal equivalent of each value.

|  |
| --- |
|  |
| **Dividend** | | **Divisor** | | **Quotient** | | **Remainder** | |
| **Hex** | **Decimal** | **Hex** | **Decimal** | **Hex** | **Decimal** | **Hex** | **Decimal** |
| 0x00000134 | 5092 | 0x0000005D | 93 | 0x0036 | 54 | 0x00046 | 70 |
| 0x220E101A | 571346970 | 0x000FDC1A | 1039386 | 0x0225 | 549 | 0xB0C58 | 724056 |
| 0xB7342F81 | 3073650561 | 0x0003BC2E | 244782 | 0x310C | 12556 | 0x28F59 | 167769 |

Demonstrate your understanding of the unsigned division algorithm by dividing the unsigned value 1011 by the unsigned value 0010. Provide a table like the table shown above.

| **Cycle** | **Divisor** | **Remainder** |
| --- | --- | --- |
| 0 | 0010 | 0000 1011 |
| 1 | 0010 | 0001 0110 |
| 2 | 0010 | 0000 1101 |
| 3 | 0010 | 0001 1010 |
| 4 | 0010 | 0001 0101 |

Exercise #1

What is the multiplicand and the multiplier used in this testbench example. What is the correct product? Provide your answers in decimal.

2020 x 317 = 640340

Provide a copy of your new tcl script.

# Start execution of the second multiplication operation

add\_force start {1 0ns}

add\_force multiplier 786d4f31 -radix hex

add\_force multiplicand 34e1cb0a -radix hex

run 10 ns

# return the inputs to zero

add\_force start {0 0ns}

add\_force multiplier 00000000 -radix hex

add\_force multiplicand 00000000 -radix hex

run 360 ns

# At this point the operation should be done and provided correct answer

# Start execution of the third multiplication operation

add\_force start {1 0ns}

add\_force multiplier c412e59b -radix hex

add\_force multiplicand a3102f73 -radix hex

run 10 ns

# return the inputs to zero

add\_force start {0 0ns}

add\_force multiplier 00000000 -radix hex

add\_force multiplicand 00000000 -radix hex

run 360 ns

# At this point the operation should be done and provided correct answer

Provide a copy of your Verilog multiplier code.

`timescale 1ns / 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Module: seq\_multiplier

\*

\* Author: Ryan Johnson

\* Class: ECEN 323, Section 01, Winter 2020

\* Date: 3 Feb 2020

\*

\* Description: multiplies and writes the result to the product output

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module seq\_multiplier(

input wire logic clk,

input wire logic rst,

input wire logic start,

input wire logic [31:0] multiplicand,

input wire logic [31:0] multiplier,

output logic busy,

output logic [63:0] product

);

logic[4:0] mult\_cnt;

localparam CNT\_END = 5'b11111;

logic state;

localparam idle = 1'b0;

localparam running = 1'b1;

// increment counter when running

always\_ff@ (posedge clk)

begin

if (state == idle) mult\_cnt <= 5'b00000;

else mult\_cnt <= mult\_cnt + 1;

end

logic[63:0] product\_reg;

logic[32:0] multiplicand33;

logic[32:0] sum;

logic write;

assign write = product\_reg[0];

assign sum = multiplicand33 + {1'b0, product\_reg[63:32]};

//insert new multiplicand while idle;

always\_ff@ (posedge clk)

if (state == idle) multiplicand33 <= {1'b0, multiplicand};

else multiplicand33 <= multiplicand33;

//reset state or go to next state;

always\_ff@ (posedge clk)

if (rst) state <= idle;

else if (start) state <= running;

else if (mult\_cnt == CNT\_END) state <= idle;

else state <= state;

//assert busy when doing a multiplication

always\_ff@ (posedge clk)

if (rst) busy <= 1'b0;

else if (start) busy <= 1'b1;

if (state == idle) busy <= 1'b0;

else if (state == running) busy <= 1'b1;

else busy <= 1'b0;

//update product\_reg every clock cycle

always\_ff@ (posedge clk)

if (state == idle) product\_reg <= {32'b0, multiplier};

else if (state == running)

begin

if (write) product\_reg <= {sum, product\_reg[31:1]};

else product\_reg <= {1'b0, product\_reg[63:1]};

end

else product\_reg <= product\_reg;

//put product\_reg into product upon completion of calculation

always\_ff@ (posedge clk)

if (mult\_cnt == CNT\_END)

begin

if (write) product <= {sum, product\_reg[31:1]};

else product <= {1'b0, product\_reg[63:1]};

end

else product <= product;

endmodule

Exercise #2

What is the divisor and the dividend used in this testbench example. What is the correct quotient and remainder for this input? Provide your answers in decimal.

5092 / 93 = 54 Remainder 70

Provide a copy of your modified tcl script

# Initialize the simulation

restart

# Run for some time with unspecified inputs (indicated by blue)

run 50 ns

# Create 100 MHz clock and run for 5 clock cycles

# (This should put "red" messed up values in the internal registers)

add\_force clk {0} {1 5} -repeat\_every 10

run 50 ns

# Initialize the inputs and run for a few clock cycles (turns inputs from blue to green)

add\_force start 0

add\_force rst 0

add\_force divisor 00000000 -radix hex

add\_force dividend 00000000 -radix hex

run 30 ns

# Issue reset pulse (should turn register values in red to zero in green)

add\_force rst 1

run 20 ns

add\_force rst 0

run 10 ns

# At this point we are ready to perform a variety of division operations

add\_force start {1 0ns}

add\_force divisor 0000005D -radix hex

add\_force dividend 000013e4 -radix hex

run 10 ns

add\_force start {0 0ns}

add\_force divisor 00000000 -radix hex

add\_force dividend 00000000 -radix hex

run 360 ns

# At this point the operation should be done and provided correct answer

add\_force start {1 0ns}

add\_force divisor 000FDC1A -radix hex

add\_force dividend 220E101A -radix hex

run 10 ns

add\_force start {0 0ns}

add\_force divisor 00000000 -radix hex

add\_force dividend 00000000 -radix hex

run 360 ns

# At this point the operation should be done and provided correct answer

add\_force start {1 0ns}

add\_force divisor 0003BC2E -radix hex

add\_force dividend B7342F81 -radix hex

run 10 ns

add\_force start {0 0ns}

add\_force divisor 00000000 -radix hex

add\_force dividend 00000000 -radix hex

run 360 ns

# At this point the operation should be done and provided correct answer

Provide a copy of your Verilog divider code

`timescale 1ns / 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Module: seq\_divider

\*

\* Author: Ryan Johnson

\* Class: ECEN 323, Section 01, Winter 2020

\* Date: 3 Feb 2020

\*

\* Description: divides and writes the result to the quotient and remainder outputs

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module seq\_divider(

input wire clk,

input wire rst,

input wire start,

input wire [31:0] divisor,

input wire [31:0] dividend,

output logic busy,

output logic [31:0] quotient,

output logic [31:0] remainder

);

logic[4:0] divide\_cnt;

localparam CNT\_END = 5'b11111;

logic state;

localparam idle = 1'b0;

localparam running = 1'b1;

// increment counter when running

always\_ff@ (posedge clk)

begin

if (state == idle) divide\_cnt <= 5'b00000;

else divide\_cnt <= divide\_cnt + 1;

end

logic[63:0] remainder\_reg;

logic[31:0] divisor\_reg;

logic[31:0] update;

//insert new divisor while idle;

always\_ff@ (posedge clk)

if (state == idle) divisor\_reg <= divisor;

else divisor\_reg <= divisor\_reg;

assign update = remainder\_reg[62:31] - divisor\_reg;

//reset state or go to next state;

always\_ff@ (posedge clk)

if (rst) state <= idle;

else if (start) state <= running;

else if (divide\_cnt == CNT\_END) state <= idle;

else state <= state;

//assert busy when doing a division

always\_ff@ (posedge clk)

if (rst) busy <= 1'b0;

else if (start) busy <= 1'b1;

if (state == idle) busy <= 1'b0;

else if (state == running) busy <= 1'b1;

else busy <= 1'b0;

//update remainder\_reg every clock cycle

always\_ff@ (posedge clk)

if (state == idle) remainder\_reg <= {32'h0000, dividend};

else if (state == running)

begin

if (remainder\_reg[62:31] >= divisor\_reg)

remainder\_reg <= {update[31:0],remainder\_reg[30:0],1'b1};

else remainder\_reg <= {remainder\_reg[62:0],1'b0};

end

else remainder\_reg <= remainder\_reg;

//put remainder\_reg into remainder and quotient upon completion of calculation

//or set to zero at reset

always\_ff@ (posedge clk)

if (divide\_cnt == CNT\_END)

begin

if (remainder\_reg[62:31] >= divisor\_reg)

begin

remainder <= update[31:0];

quotient <= {remainder\_reg[30:0], 1'b1};

end

else

begin

remainder <= remainder\_reg[62:31];

quotient <= {remainder\_reg[30:0], 1'b0};

end

end

else if (rst)

begin

remainder <= 32'h0000;

quotient <= 32'h0000;

end

else

begin

remainder <= remainder;

quotient <= quotient;

end

endmodule

Exercise #3

Provide a copy of your top level design

`timescale 1ns / 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Module: seq\_multdiv

\*

\* Author: Ryan Johnson

\* Class: ECEN 323, Section 01, Winter 2020

\* Date: 7 Feb 2020

\*

\* Description: multiplies or divides and writes the result to the outputs

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module seq\_multdiv(

input wire clk,

input wire rst,

input wire start,

input wire op,

input wire [31:0] a,

input wire [31:0] b,

output logic busy,

output logic [31:0] r1,

output logic [31:0] r2

);

logic [31:0] multiplier;

logic [31:0] multiplicand;

logic [31:0] divisor;

logic [31:0] dividend;

assign multiplicand = a;

assign dividend = a;

assign multiplier = b;

assign divisor = b;

logic mult\_start;

logic div\_start;

logic mult\_busy;

logic div\_busy;

logic held\_op;

always\_ff@ (posedge clk)

begin

if(start && !busy) held\_op <= op;

end

assign mult\_start = (!op && !busy && start) ? 1'b1 : 1'b0;

assign div\_start = (op && !busy && start) ? 1'b1 : 1'b0;

logic [63:0] mult\_product;

logic [31:0] div\_quotient;

logic [31:0] div\_remainder;

// connecting to seq\_multiplier

seq\_multiplier m1(.clk(clk), .rst(rst), .start(mult\_start), .multiplicand(multiplicand),

.multiplier(multiplier), .busy(mult\_busy), .product(mult\_product));

// connecting to seq\_divider

seq\_divider d1(.clk(clk), .rst(rst), .start(div\_start), .dividend(dividend),

.divisor(divisor), .busy(div\_busy), .quotient(div\_quotient), .remainder(div\_remainder));

// assign busy the correct value

assign busy = held\_op ? div\_busy : mult\_busy;

// assign r1 and r2 the result of the operation

always\_comb

if (held\_op)

begin

r1 = div\_quotient;

r2 = div\_remainder;

end

else

begin

r1 = mult\_product[31:0];

r2 = mult\_product[63:32];

end

endmodule

Provide a copy of the simulation output from the testbench.

\*\*\* Start of Simulation \*\*\*

\*\*\* Correct: 34e1cb0a X 786d4f31=18e06babf021f2ea received at time 535 ns

\*\*\* Correct: 220e101a/000fdc1a=00000225 received at time 925 ns

\*\*\* Correct: 220e101a/000fdc1a = remainder 000b0c58 received at time 925 ns

\*\*\* Correct: f4e1cb0a X 886d4f31=828083c15021f2ea received at time 1315 ns

\*\*\* Correct: 8484d609/b1f05663=00000000 received at time 1705 ns

\*\*\* Correct: 8484d609/b1f05663 = remainder 8484d609 received at time 1705 ns

\*\*\* Correct: 12153524 X c0895e81=0d999be15676ff24 received at time 2095 ns

\*\*\* Correct: b1f05663/0484d609=00000027 received at time 2485 ns

\*\*\* Correct: b1f05663/0484d609 = remainder 01b3bb04 received at time 2485 ns

\*\*\* Correct: 06b97b0d X 46df998d=01dc97404d068b29 received at time 2875 ns

\*\*\* Correct: 89375212/32c28465=00000002 received at time 3265 ns

\*\*\* Correct: 89375212/32c28465 = remainder 23b24948 received at time 3265 ns

\*\*\* Correct: 00f3e301 X 06d7cd0d=000684e90501540d received at time 3655 ns

\*\*\* Correct: 1e8dcd3d/3b23f176=00000000 received at time 4045 ns

\*\*\* Correct: 1e8dcd3d/3b23f176 = remainder 1e8dcd3d received at time 4045 ns

\*\*\* Correct: 76d457ed X 462df78c=209366461f9ec09c received at time 4435 ns

\*\*\* Correct: e33724c6/3cfde9f9=00000003 received at time 4825 ns

\*\*\* Correct: e33724c6/3cfde9f9 = remainder 2c3d66db received at time 4825 ns

\*\*\* Correct: e2f784c5 X d513d2aa=bce9849c86e6c4d2 received at time 5215 ns

\*\*\* Correct: bbd27277/32aff7e5=00000003 received at time 5605 ns

\*\*\* Correct: bbd27277/32aff7e5 = remainder 23c28ac8 received at time 5605 ns

\*\*\* Correct: 8932d612 X 47ecdb8f=268c09e7771efa0e received at time 5995 ns

\*\*\* Correct: e77696ce/393069f2=00000004 received at time 6385 ns

\*\*\* Correct: e77696ce/393069f2 = remainder 02b4ef06 received at time 6385 ns

\*\*\* Correct: f4007ae8 X e2ca4ec5=d8293ff5b4e14488 received at time 6775 ns

\*\*\* Correct: de8e28bd/2e58495c=00000004 received at time 7165 ns

\*\*\* Correct: de8e28bd/2e58495c = remainder 252d034d received at time 7165 ns

\*\*\* Correct: 96ab582d X b2a72665=692583aeb00b77c1 received at time 7555 ns

\*\*\* Correct: 0573870a/31ef6263=00000000 received at time 7945 ns

\*\*\* Correct: 0573870a/31ef6263 = remainder 0573870a received at time 7945 ns

\*\*\* Simulation done with 0 errors \*\*\* 7995 ns

Exercise #4

Summarize any warnings you had during synthesis.

None

Summarize the number of “Slice LUTs” and “Slice Registers” used by your arithmetic unit.

273 Slice LUTs

334 Slice Registers

Summarize the estimated resources for your synthesized logic in the table below.

|  |  |
| --- | --- |
| **Resource** | **Estimation** |
| LUT | 273 |
| FF | 334 |
| BRAM | 0 |
| IO | 133 |
| BUFG | 1 |

How many hours did you work on the lab?

6 Hours

Please provide any suggestions for improving this lab in the future:

It would be helpful to clarify that the busy signal should be driven high starting the same cycle that start is driven high.