[**Executive Summary**](#_cv46q2o89n2m) **3**

[**User Operation**](#_821ly0t0n6i2) **4**

[**Module Table**](#_jb42o85umyu8) **5**

[**RTL Netlists**](#_pxr2y26cilb6) **7**

[Full Diagram](#_ucw9yuybctdu) **7**

[System](#_jq0dzclrx9e) 8

[Compute](#_vw5u22ax8giy) 8

[Count Control](#_7gsjc08swsre) 9

[Display](#_c52frgfnr73f) 10

[Memory](#_j3cz981muj3l) 11

[RAM](#_hirvi4j26zgf) 11

[ROM](#_4k3hbau2zi5l) 11

[**Modules, Testbenches, and Waveforms**](#_epi9aghcr31g) **12**

[MLP\_top](#_hfusmfqb8yux) 12

[testbench](#_6kfhxq7pnkk) 12

[waveform](#_loysgmqmvlmd) 14

[my\_ram](#_7xlz8nwg3xvt) 15

[testbench](#_9yjdrzymwfto) 15

[waveform](#_7hd8rzgf1pf0) 16

[my\_rom](#_u46ll9pkxdn5) 17

[testbench](#_qz1l02wnqmp1) 17

[waveform](#_3bqwb0kyc5d) 18

[multiplier](#_14g754eddqs6) 19

[testbench](#_r06mirv95oq4) 19

[waveform](#_gcpo4ypbrcky) 20

[adder](#_f0fvbrgg61vk) 21

[testbench](#_pacn3t23wg1) 21

[waveform](#_2i9xbtoezsmc) 22

[my\_dff](#_dpy5pdk2l9kw) 23

[testbench](#_o8rv868dsq96) 23

[waveform](#_7g1if7q0hsrh) 24

[mux2x1](#_j91i4mb34j8l) 24

[testbench](#_y2p3m33kimt4) 24

[waveform](#_iqm2g2o5aotg) 26

[compute](#_pha5sts1pucy) 27

[module compute\_tb();](#_5i861iqzgpyh) 27

[waveform](#_kypg30stbbas) 28

[count\_control](#_5ahhlhc59f6t) 28

[testbench](#_7dacbbfa2m2t) 28

[waveform](#_iqz3sojbe62o) 30

[display](#_3cnlognhfr5u) 31

[testbench](#_jdrderqk64y6) 31

[waveform](#_qgyx7y3la01m) 32

[**Video Demonstration Link**](#_dkayog13n8u2) **33**

# Executive Summary

For our midterm project, we have created a Machine Learning Processor. This processor has a 16 address ROM loaded with values from 0 to 15 loaded sequentially in memory. For example, address 0 has the value of 0 stored, address 1 has the value of 1 stored, and so on. The processor also has a RAM that is loaded with values from the user that is input through the keyboard using the UART\_RX module. Specifically, we pass in the input as a 4 bit binary representation of the number that the user pressed on the keyboard and pass it into RAM. The address that the user writes into is controlled by a counter that is initially set to 0 and increments every time it detects a positive edge of w\_RX\_DV and when SW[0] = 0. The user is allowed to input a decimal number from 0-9 and can store up to 16 values in RAM. We will assume that the user starts the program with SW[0] set to off.

After the user writes in all the values that they want, they must switch SW[0] on in order to put the system into computing mode. Once SW[0] = 1, the read address counter increments with the positive edge of CLOCK\_50, and is passed into ROM and RAM as raddr input. At each clock cycle, the 4 bit data from ROM and RAM at the counter address are multiplied and passed into a multiplier which outputs an 8 bit product of the two numbers. The product is passed into an adder, which sums the product with a value from a 2 by 1 mux. That 12 bit sum is then stored in a dff register which outputs a 12 bit wire into 3 hex displays. In order to accumulate the sums, the dff output is also passed into the 2 by 1 mux and once the mux detects output, the mux selector switches to 1 and outputs the output from the dff instead of the initial value of zero.

# User Operation

The user operates our project using both keyboard input and the FPGA board. The user ‘s job is to load the RAM with values. The system is in write mode when the input switch0 of the FPGA keyboard is off or 0. This value will show on hex4. When the system is in write mode the user can then enter values into the keyboard. The user should enter only single digit numbers from 0-9. The number of values entered will display on hex6, as it will increment the write address.. After the user has entered all the values they wish, they should flip the switch0 to on or 1. The system will then compute the dot product of the value pairs in ROM and RAM and output the result to the board at hex0-2. If the user wishes to reset the system, they should make sure the switch is set to write mode, then press the key0. This reset will clear the output and set the counters back to the 0 address.



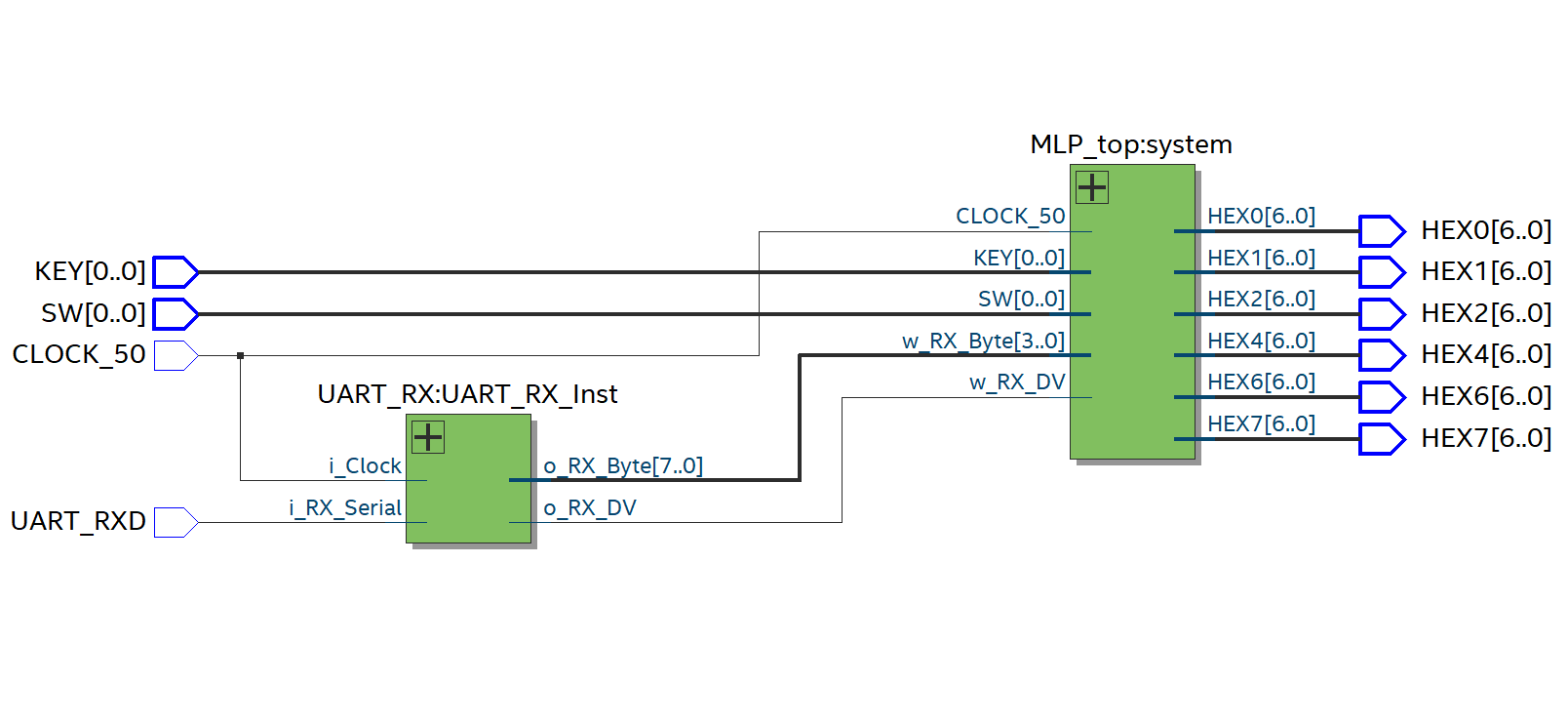
# Module Table

\*\*implies module was provided for us

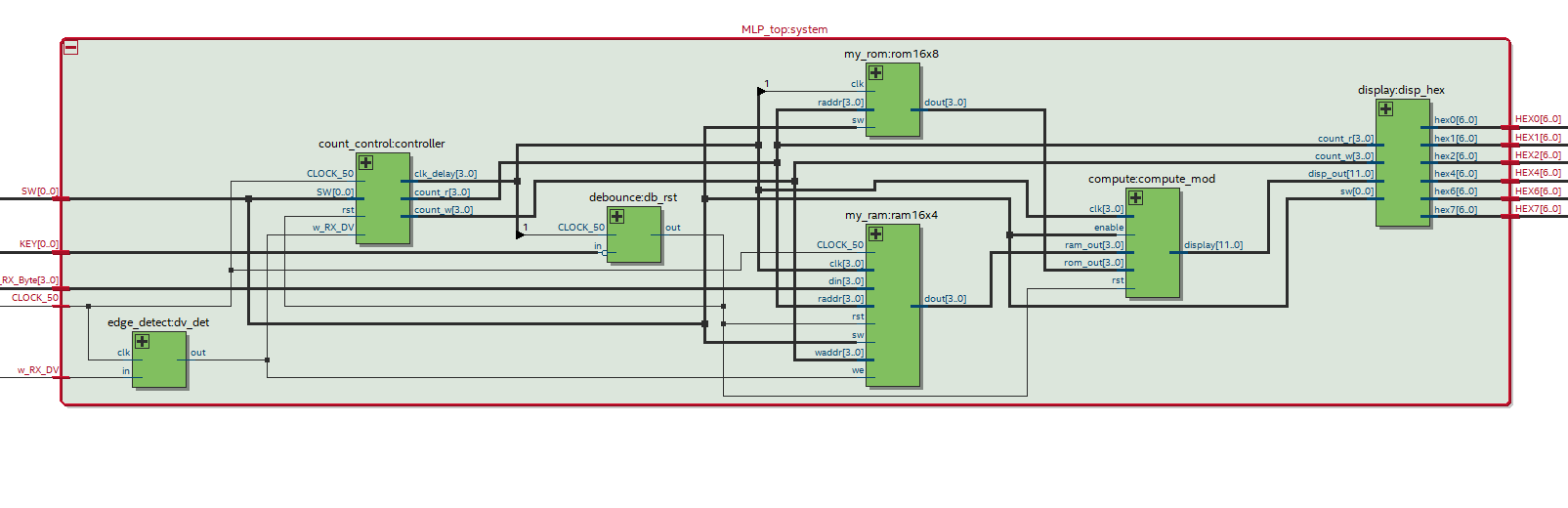
| **Module** | **Description** |
| --- | --- |
| MLP\_top\_de2 | A Module that combines all the modules in the system to create a machine learning processor that receives input from a keyboard and outputs the total sum of 16 pairs of numbers that have been multiplied together to an FPGA board. |
| MLP\_top | Top module of system that contains most of the logic and modules to create a machine learning processor, without the keyboard input |
| my\_ram | Random access memory module that reads and stores data inputted from the keyboard. |
| my\_rom | Read-only memory module that is preloaded with 16 values. |
| multiplier | Module that takes in two 4-bit numbers as inputs and outputs the numbers’ product. This module is used to multiply the values stored at RAM and ROM at the current count address. |
| adder | Module that takes in an 8-bit number and a 12-bit number and outputs the numbers’ sum. This module is used to add the most recent product from the multiplier with the current sum received from the multiplexor. |
| my\_dff | A d-flip-flop module that stores a 12-bit number. This module is used to store the current sum. |
| mux2x1 | A multiplexor module that takes in two inputs and one select bit to output the selected value. This is used to output a 0 for the first addition(because there is no current sum), and then the current sum for all the following additions. |
| compute | A module that instantiates the multiplier, adder, dff, and mux modules to create one hierarchical computing step. |
| debounce\*\* | A module that debounces keypresses on the FPGA board. This module is used for the reset button. |
| count\_control | A module that sets up the logic for the clock cycles and counters. |
| display | A module that sends the output from compute to the FPGA board. |
| edge\_detect\*\* | A module that takes in an input signal from the keyboard (w\_RX\_DV) on the positive edge of each clock cycle and outputs one at the positive edge of the input . This is used to ensure each key press is recognized. |
| hexdigit\*\* | A module that takes in a 4 bit input and displays the corresponding hexadecimal value on the seven-segment hex display. |
| UART\_RX\*\* | A module that receives input from the keyboard. |

# RTL Netlists

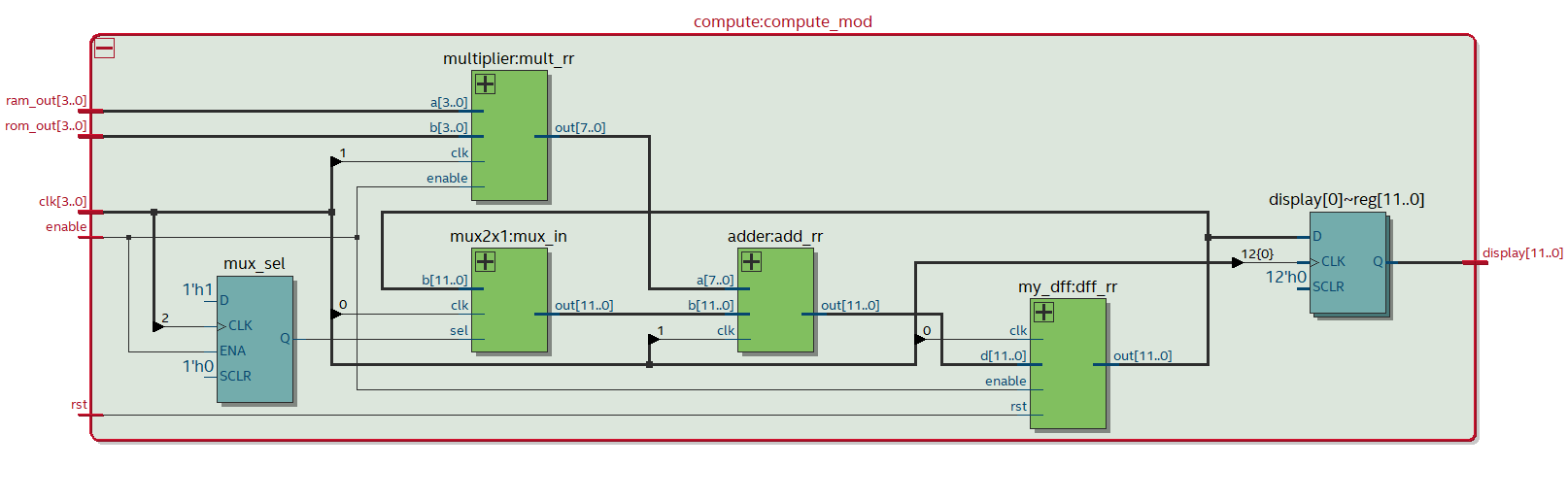
## Full Diagram



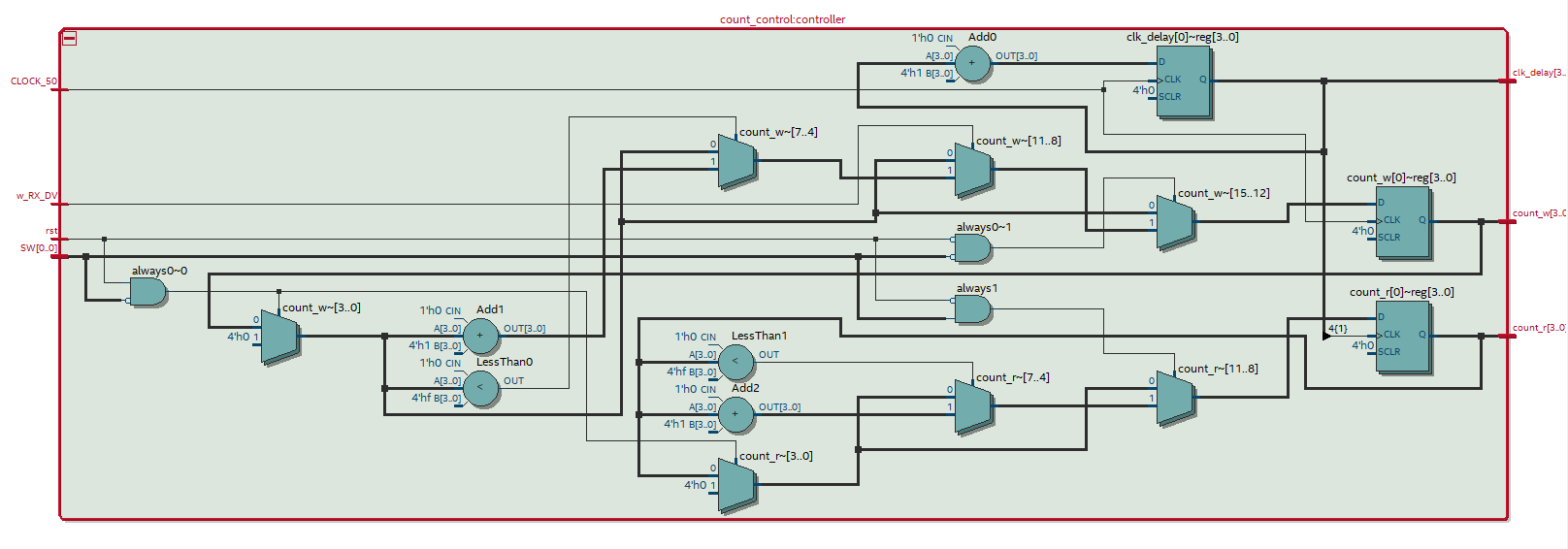
## System



## Compute



## Count Control



## 

## Display

# 

# 

# 

# 

# 

# 

# 

# 

# Memory

### RAM

# 

### ROM

# 

# 

# Modules, Testbenches, and Waveforms

## MLP\_top

### 

### testbench

`timescale 1ns/1ns

module MLP\_top\_tb();

reg t\_clk;

reg [0:0] t\_key;

reg [0:0] t\_sw;

reg [3:0] t\_RXD;

reg DV;

wire [6:0] hex0;

wire [6:0] hex1;

wire [6:0] hex2;

wire [6:0] hex4;

wire [6:0] hex6;

wire [6:0] hex7;

MLP\_top uut (

.CLOCK\_50 (t\_clk),

.KEY (t\_key),

.SW (t\_sw),

.w\_RX\_Byte (t\_RXD),

.w\_RX\_DV (DV),

.HEX0 (hex0),

.HEX1 (hex1),

.HEX2 (hex2),

.HEX4 (hex4),

.HEX6 (hex6),

.HEX7 (hex7)

);

always

#5 t\_clk = ~t\_clk;

initial begin

t\_key = 1; t\_clk = 1; t\_sw = 0;

t\_RXD = 4'b0001; DV = 1; // 1

#10; t\_RXD = 4'b0001; DV = 0;

#10; t\_RXD = 4'b0010; DV = 1; // 2

#10; t\_RXD = 4'b0010; DV = 0;

#10; t\_RXD = 4'b0011; DV = 1; // 3

#10; t\_RXD = 4'b0011; DV = 0;

#100; t\_RXD = 4'b1000; DV = 1; // 8

#10; t\_RXD = 4'b1000; DV = 0;

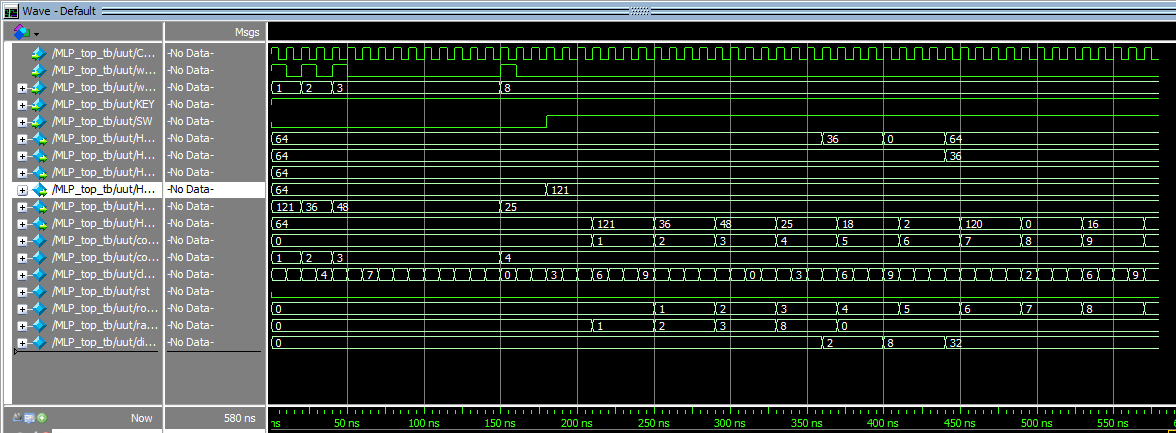
#20; t\_sw =1; DV = 0;

#400; $stop;

end

endmodule

### waveform



## 

## 

## 

## my\_ram

### 

### testbench

`timescale 1ns/1ns

module my\_ram\_tb();

reg [3:0] clk;

reg CLOCK\_50;

reg rst;

reg sw;

reg [3:0] raddr;

reg [3:0] waddr;

reg [3:0] din;

reg we;

wire [3:0] dout;

my\_ram uut (

.clk (clk),

.CLOCK\_50 (CLOCK\_50),

.rst (rst),

.sw (sw),

.raddr (raddr),

.waddr (waddr),

.din (din),

.we (we),

.dout (dout)

);

always @(posedge CLOCK\_50)

clk = clk + 4'b1;

always

#2 CLOCK\_50 = ~CLOCK\_50;

initial begin

clk = 0; CLOCK\_50 = 0; rst = 0; sw = 0; we = 1; raddr = 0; waddr = 0; din = 4'd15;

#8; waddr = 4'd1; din = 4'd3;

#8; waddr = 4'd2; din = 4'd7;

#8; sw = 1; we = 0;

#20; raddr = 4'd0;

#20; raddr = 4'd1;

#20; raddr = 4'd2;

#20; rst = 1; sw = 0;

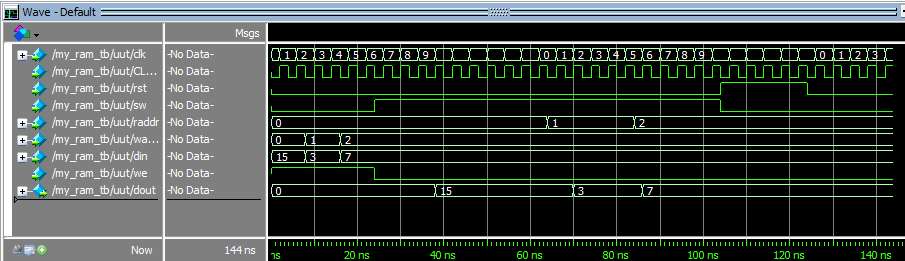
#20; rst = 0;

#20; $stop;

end

endmodule

### waveform



## 

## my\_rom

### 

### testbench

`timescale 1ns/1ns

module my\_rom\_tb();

reg [3:0] raddr;

reg clk;

reg sw;

wire [3:0] dout;

my\_rom uut (

.raddr (raddr),

.clk (clk),

.sw (sw),

.dout (dout)

);

always

#5 clk = ~clk;

initial begin

clk = 0; sw = 1; raddr = 4'd4;

#10; raddr = 4'd3;

#10; raddr = 4'd6;

#10; raddr = 4'd7;

#10; raddr = 4'd9;

#10; raddr = 4'd1;

#10; $stop;

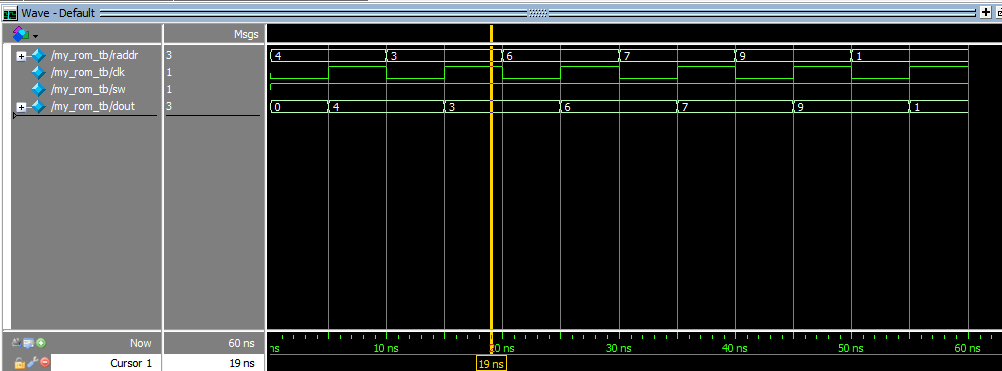
end

endmodule

### 

### 

### waveform



## 

## multiplier

### 

### testbench

`timescale 1ns/1ns

module multiplier\_tb();

reg clk;

reg enable;

reg [3:0] a;

reg [3:0] b;

wire [7:0] out;

multiplier uut (

.clk (clk),

.enable (enable),

.a (a),

.b (b),

.out (out)

);

always

#5 clk = ~clk;

initial begin

a = 4'd1; b = 4'd2; enable = 1; clk =0;

#10; a = 4'd3; b = 4'd3;

#10; a = 4'd15; b = 4'd11;

#10; a = 4'd7; b = 4'd4;

#10; a = 4'd20; b = 4'd5; enable = 0;

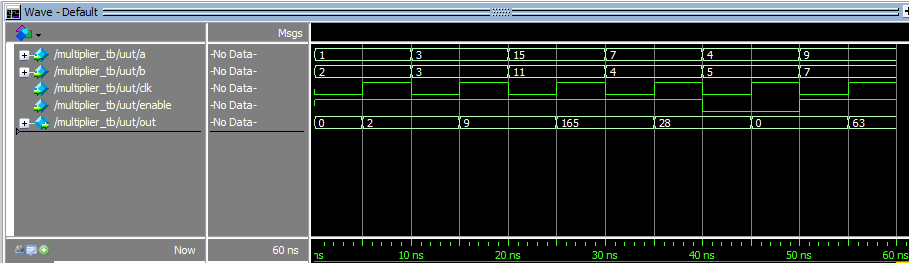
#10; a = 4'd9; b = 4'd7; enable = 1;

#10; $stop;

end

endmodule

### waveform



## 

## adder

### 

### testbench

`timescale 1ns/1ns

module adder\_tb();

reg clk;

reg [7:0] a;

reg [11:0] b;

wire [11:0] out;

adder uut (

.clk (clk),

.a (a),

.b (b),

.out (out)

);

always

#5 clk = ~clk;

initial begin

a = 8'd1; b = 12'd2; clk = 0;

#10; a = 8'd3; b = 12'd3;

#10; a = 8'd15; b = 12'd11;

#10; a = 8'd7; b = 12'd4;

#10; a = 8'd20; b = 12'd5;

#10; a = 8'd9; b = 12'd7;

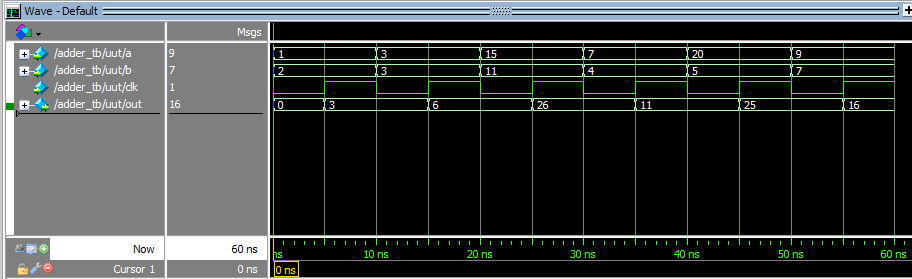
#10; $stop;

end

endmodule

### 

### waveform



## 

## my\_dff

### 

### testbench

`timescale 1ns/1ns

module my\_dff\_tb();

reg [11:0] d;

reg clk;

reg rst;

reg enable;

wire [11:0] out;

my\_dff uut (

.d (d),

.clk (clk),

.rst (rst),

.enable (enable),

.out (out)

);

always

#5 clk = ~clk;

initial begin

clk = 0; rst = 0; enable = 0; d= 12'h345;

#10; d = 12'h04a;

#10; d = 12'h111;

#10; d = 12'h400;

#10; d = 12'h876; rst = 1;

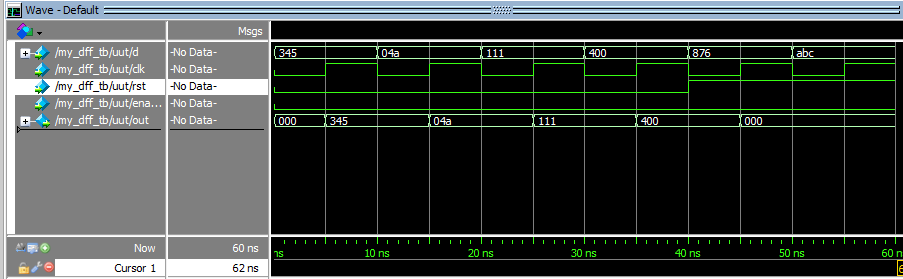
#10; d = 12'habc;

#10; $stop;

end

endmodule

### waveform



## 

## mux2x1

### 

### testbench

`timescale 1ns/1ns

module mux2x1\_tb();

reg clk;

reg sel;

reg [11:0] b;

wire [11:0] out;

mux2x1 uut (

.clk (clk),

.sel (sel),

.b (b),

.out (out)

);

always

#5 clk = ~clk;

initial begin

clk = 0; b= 4'd2; sel = 1;

#10; b = 4'd3;

#10; b = 4'd11;

#10; b = 4'd4;

#10; b = 4'd5; sel = 0;

#10; b = 4'd7; sel = 1;

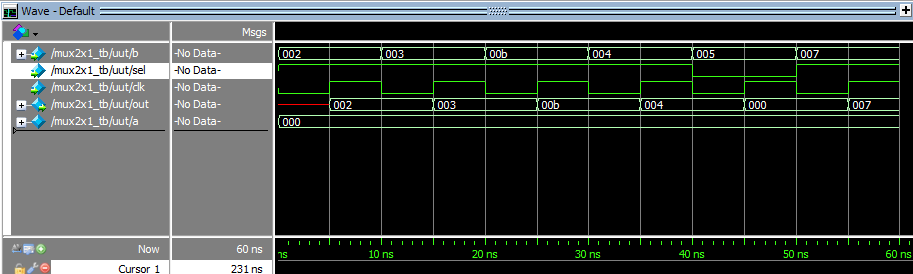
#10; $stop;

end

endmodule

### 

### waveform



## 

## compute

`timescale 1ns/1ns

### module compute\_tb();

reg [3:0] ram\_out;

reg [3:0] rom\_out;

reg [3:0] clk;

reg rst;

reg enable;

wire [11:0] display;

compute uut (

.ram\_out (ram\_out),

.rom\_out (rom\_out),

.clk (clk),

.rst (rst),

.enable (enable),

.display (display)

);

always

#5 clk = clk + 4'b1;

initial begin

clk = 0; rst= 0;

#20; rom\_out = 4'h9; ram\_out = 4'h1; enable = 1;

#20; ram\_out = 4'h3; rom\_out = 4'h2;

#20; ram\_out = 4'ha; rom\_out = 4'h3;

#20; ram\_out = 4'h5; rom\_out = 4'h4;

#20; ram\_out = 4'h4; rom\_out = 4'h5;

#20; ram\_out = 4'h3; rom\_out = 4'h6;

#20; ram\_out = 4'h0; rom\_out = 4'h7;

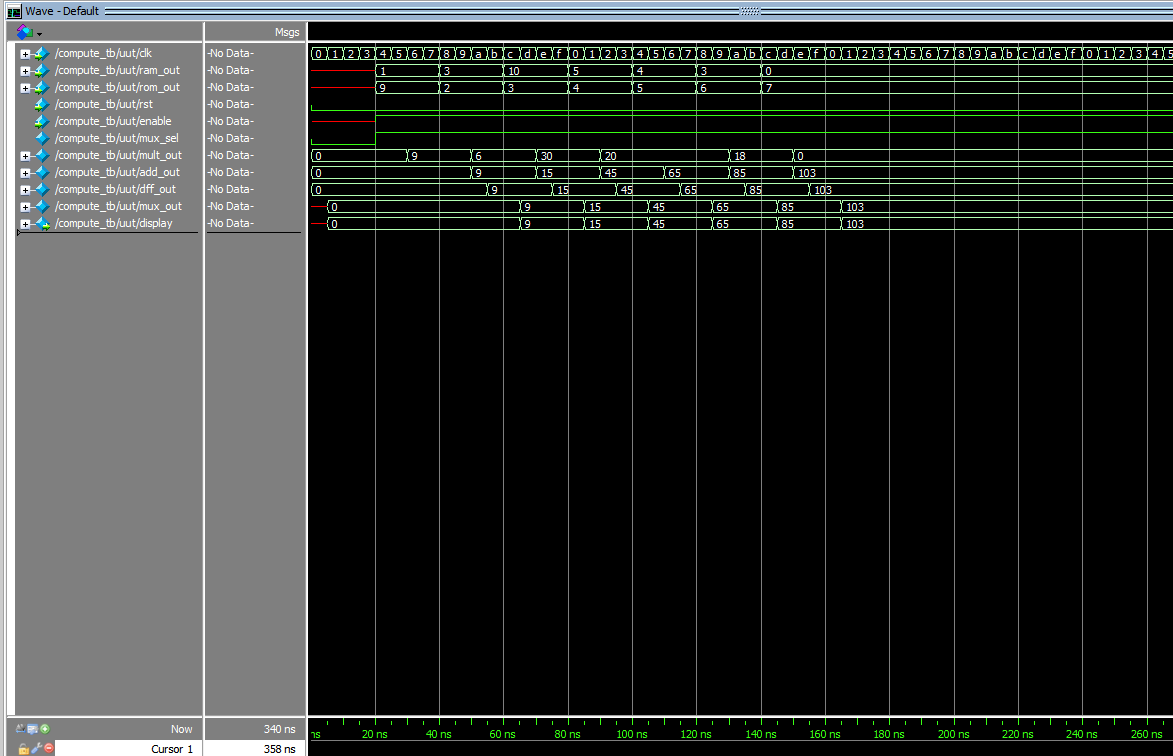
#200; $stop;

end

endmodule

### 

### waveform



## 

## count\_control

### 

### testbench

module count\_control\_tb();

reg t\_clk;

reg DV;

reg rst;

reg SW;

wire [3:0] clk\_delay;

wire [3:0] count\_w;

wire [3:0] count\_r;

count\_control uut (

.CLOCK\_50 (t\_clk),

.w\_RX\_DV (DV),

.rst (rst),

.SW (SW),

.clk\_delay (clk\_delay),

.count\_w (count\_w),

.count\_r (count\_r)

);

always

#5 t\_clk = ~t\_clk;

initial begin

t\_clk = 0; rst = 0; DV = 1; SW = 0;

#100; rst = 1;

#10; rst = 0;

#100; SW = 1; DV = 0;

#200; SW = 0; rst = 1;

#20; $stop;

end

endmodule

### waveform

## 

## 

## display

### 

### testbench

`timescale 1ns/1ns

module display\_tb();

reg [11:0] disp\_out;

reg [3:0] count\_r;

reg [3:0] count\_w;

reg [3:0] sw;

reg [3:0] ram\_out;

wire [6:0] hex0;

wire [6:0] hex1;

wire [6:0] hex2;

wire [6:0] hex4;

wire [6:0] hex6;

wire [6:0] hex7;

display uut (

.disp\_out (disp\_out),

.count\_r (count\_r),

.count\_w (count\_w),

.sw (sw),

.ram\_out (ram\_out),

.hex0 (hex0),

.hex1 (hex1),

.hex2 (hex2),

.hex4 (hex4),

.hex6 (hex6),

.hex7 (hex7)

);

initial begin

disp\_out = 12'h055; count\_r = 4'h2; count\_w = 4'd4; sw = 4'h0; ram\_out = 4'h3;

#10; disp\_out = 12'h343; count\_r = 4'h4; count\_w = 4'ha; sw = 4'h1; ram\_out = 4'h6;

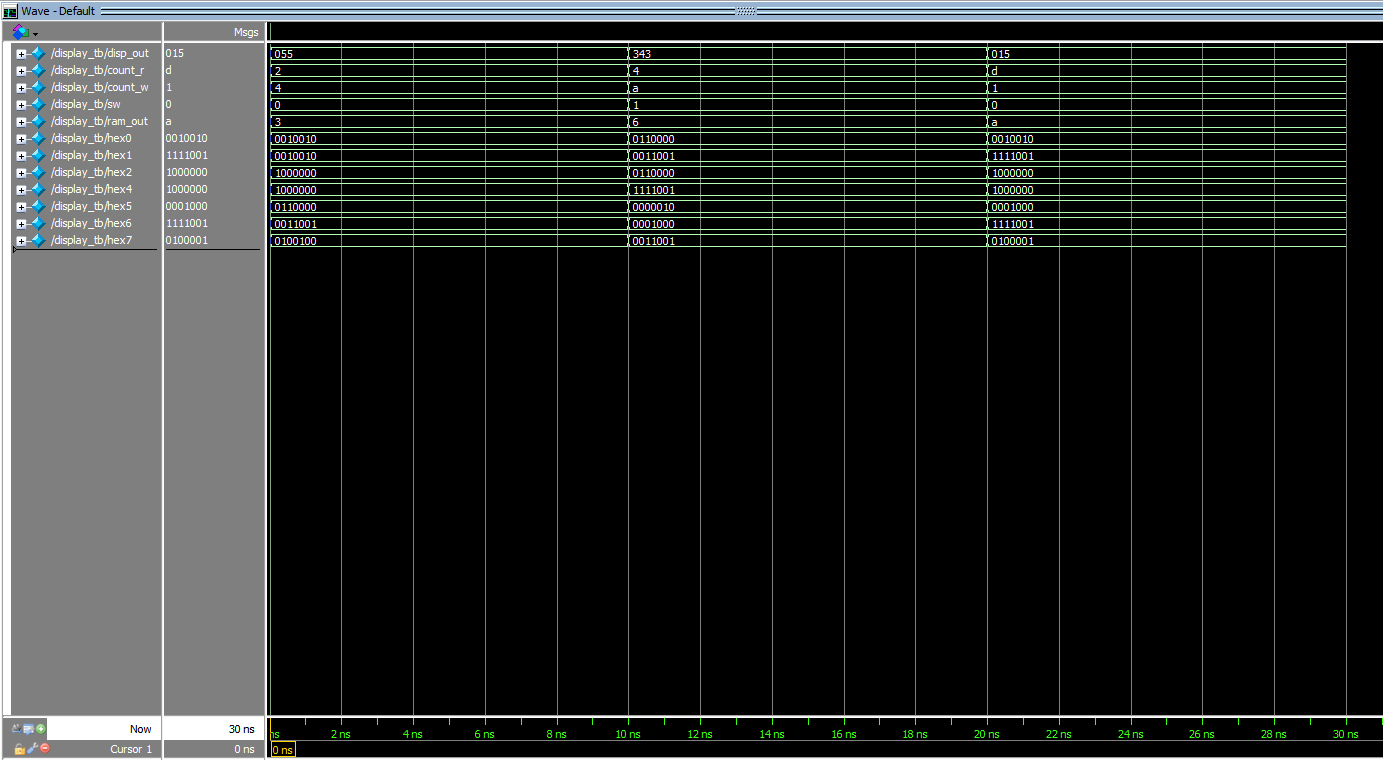
#10; disp\_out = 12'h015; count\_r = 4'hd; count\_w = 4'd1; sw = 4'h0; ram\_out = 4'ha;

#10; $stop;

end

endmodule

### waveform



# Video Demonstration Link

<https://youtu.be/4MrRCtvZbDk>