REAL-TIME DIGITAL SYSTEMS DESIGN AND VERIFICATION WITH FPGAS
ECE 387 – LECTURE 4

PROF. DAVID ZARETSKY

DAVID.ZARETSKY@NORTHWESTERN.EDU

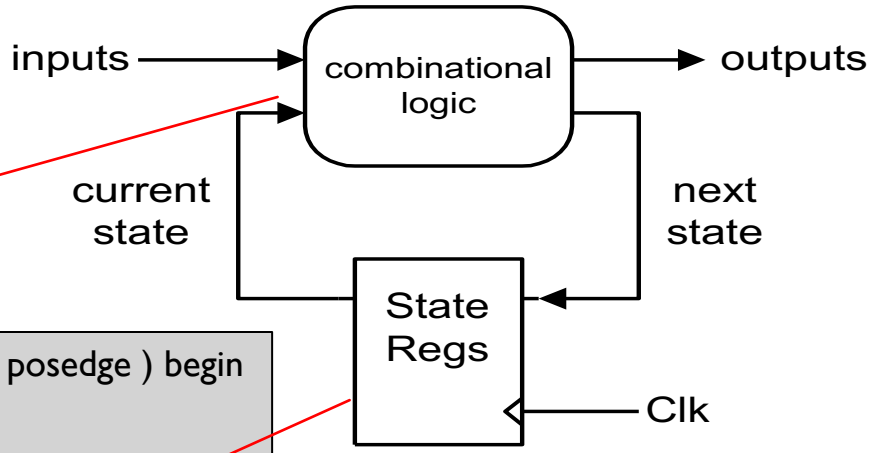
AGENDA

- Hierarchical Designs
- Simulation

HIERARCHICAL DESIGN SYNCHRONIZATION

- Separates the FSM into combinational and clocked processes
- Use **start** and **done** signals for synchronization
- Parent module initiates start / waits on done

```
always_comb begin
  next_state = state;
  z_c = z;
  done_c = done;
  case ( state ) begin
    S0 :
      if ( start = 1'b1 ) begin
          done_c = 1'b0;
          next_state = S1;
       end else begin
          next_state = S0;
       end
    S1:
       z_c = x + y + z;
       done_c = 1'b1;
       next_state = S0;
     :
    default:
       done_c <= X;
       next_state <= S0;
  end
end
```




```
always_ff( posedge clock, posedge ) begin
  if ( reset = 1'b1 ) then
     state <= S0;
     done <= '0';
     z <= 0;
  else begin
     state <= next_state;
     done <= done_c;
      z <= z_c;
  end
end
```

DESIGN EXAMPLE: VECTORSUM

- Without context, we don't know to what addresses X, Y, and Z point
  - Local Memory (Block RAMs / FIFOs)
  - Constant Memory (ROMs)
  - External Memory (DDR3)
  - Peripheral (Audio / Video)
  - Bus (SoC)

```
void add_n ( int *X, int *Y, int *Z, int n )
{
    for ( int i = 0; i < n; i++ )
    {
        Z[i] = X[i] + Y[i];
    }
}
```

Most architecture work is spent:

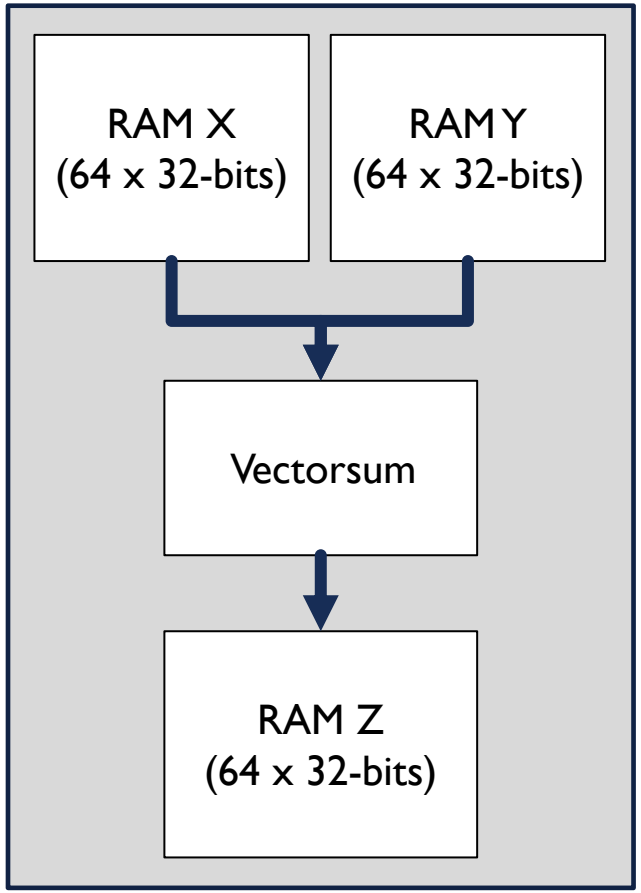
- Passing data between functions
- Optimizing memory addressing
- Accessing I/O data

VECTORSUM SOFTWARE MODEL

- Vectorsum in context:
  - Read arrays X and Y from RAMs
  - Compute vector addition of n-elements
  - Write output array Z to RAM

```
void vectorsum ( int *X, int *Y, int *Z, int n )
{
        for ( int i = 0; i < n; i++ )
        {
                Z[i] = X[i] + Y[i];
        }
}

int main()
{
        int X[64], Y[64], Z[64];
        vectorsum( X, Y, Z, 64 );
}
```

X, Y, and Z are pointing to local data stored on the stack.



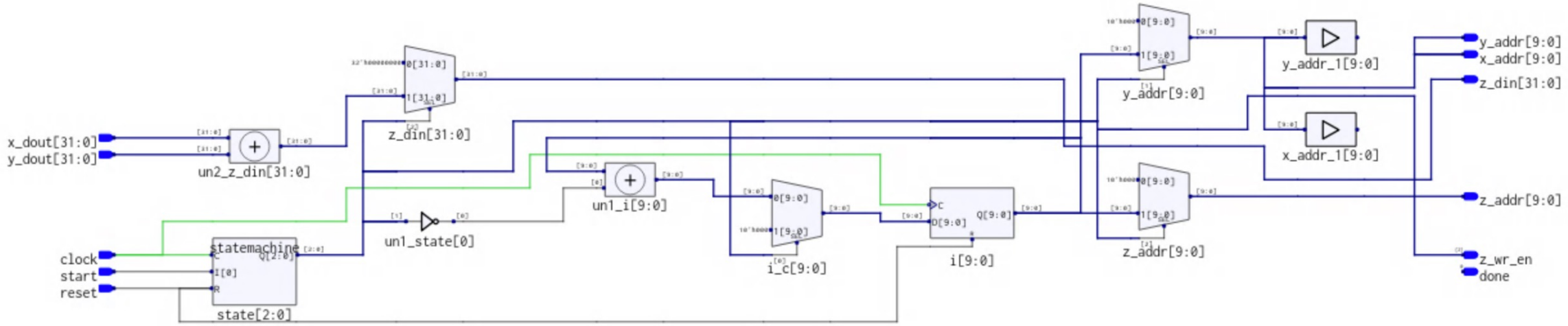
VECTORSUM IN SYSTEM VERILOG

```
module vectorsum
#( parameter DATA_WIDTH = 32,
   parameter ADDR_WIDTH = 10,
   parameter VECTOR_SIZE = 1024)
(
   input logic clock,
   input logic reset,
   input logic start,
   output logic done,
   input logic [DATA_WIDTH-1:0] x_dout,
   output logic [ADDR_WIDTH-1:0] x_addr,
   input logic [DATA_WIDTH-1:0] y_dout,
   output logic [ADDR_WIDTH-1:0] y_addr,
   output logic [DATA_WIDTH-1:0] z_din,
   output logic [ADDR_WIDTH-1:0] z_addr,
   output logic z_wr_en);

typedef enum logic [1:0] {s0, s1, s2} state_t;
state_t state, state_c;
logic [ADDR_WIDTH-1:0] i, i_c;
logic done_c;

always_ff @(posedge clock or posedge reset) begin
  if (reset) begin
    state <= s0;
    i <= '0;
    done <= 1'b0;
  end else begin
    state <= state_c;
    i <= i_c;
    done <= done_c;
  end
end
```

```
always_comb begin
   z_din = 'b0;
   z_wr_en = 'b0;
   z_addr = 'b0;
   x_addr = 'b0;
   y_addr = 'b0;
   state_c = state;
   i_c = i;
   done_c = done;

   case (state)
     s0: begin
       i_c = '0;
       if (start == 1'b1) begin
          state_c = s1;
          done_c = 1'b0;
       end
     end
     s1: begin
        if ($unsigned(i) < $unsigned(VECTOR_SIZE)) begin
           x_addr = i;
           y_addr = i;
           state_c = s2;
        end else begin
           done_c = 1'b1;
           state_c = s0;
        end
     end
```

```
     s2: begin
         z_din = $signed(y_dout) + $signed(x_dout);
         z_addr = i;
         z_wr_en = 1'b1;
         i_c = i + 'b1;
         state_c = s1;
     end

     default: begin
       z_din = 'x;
       z_wr_en = 'x;
       z_addr = 'x;
       x_addr = 'x;
       y_addr = 'x;
       state_c = s0;
       i_c = 'x;
       done_c = 'x;
     end
   endcase
end

endmodule
```

VECTORSUM ARCHITECTURE



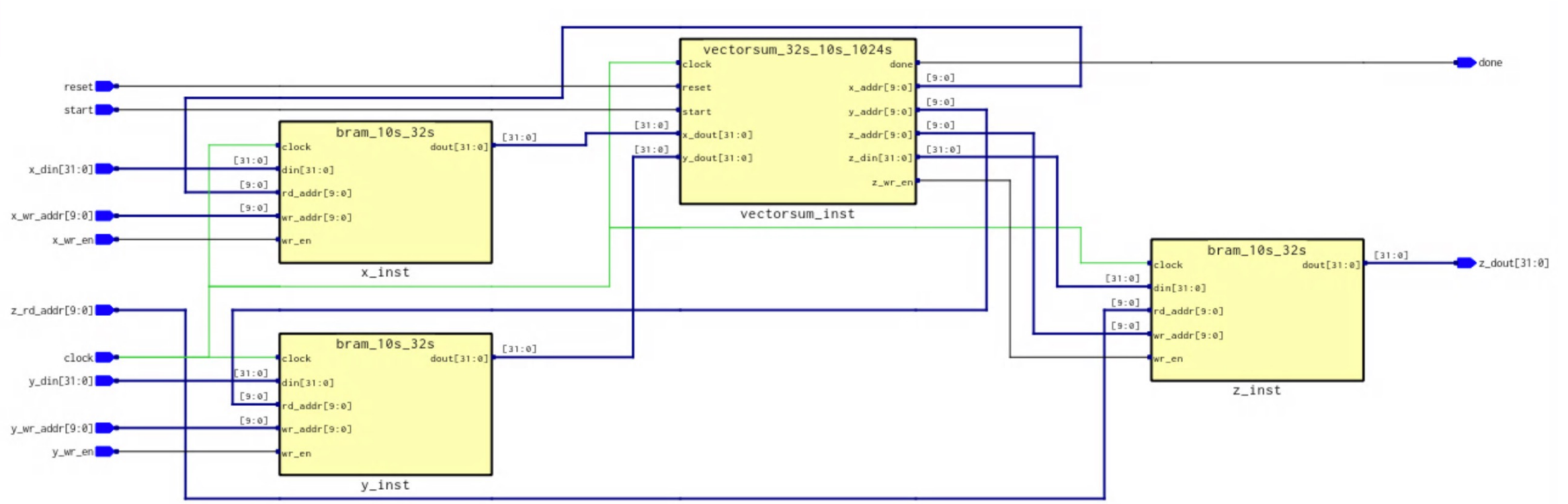
VECTORSUM TOP

```
module vectorsum_top
#( parameter DATA_WIDTH = 32,
   parameter ADDR_WIDTH = 10,
   parameter VECTOR_SIZE = 1024)
(
   input logic clock,
   input logic reset,
   input logic start,
   output logic done,
   input logic [DATA_WIDTH-1:0] x_din,
   input logic [ADDR_WIDTH-1:0] x_wr_addr,
   input logic x_wr_en,
   input logic [DATA_WIDTH-1:0] y_din,
   input logic [ADDR_WIDTH-1:0] y_wr_addr,
   input logic y_wr_en,
   output logic [DATA_WIDTH-1:0] z_dout,
   input logic [ADDR_WIDTH-1:0] z_rd_addr
);

logic [DATA_WIDTH-1:0] x_dout;
logic [ADDR_WIDTH-1:0] x_rd_addr;
logic [DATA_WIDTH-1:0] y_dout;
logic [ADDR_WIDTH-1:0] y_rd_addr;
logic [DATA_WIDTH-1:0] z_din;
logic [ADDR_WIDTH-1:0] z_wr_addr;
logic z_wr_en;
```

```
vectorsum #(
   .DATA_WIDTH(DATA_WIDTH),
   .ADDR_WIDTH(ADDR_WIDTH),
   .VECTOR_SIZE(VECTOR_SIZE)
) vectorsum_inst (
   .clock(clock),
   .reset(reset),
   .start(start),
   .done(done),
   .x_dout(x_dout),
   .x_addr(x_rd_addr),
   .y_dout(y_dout),
   .y_addr(y_rd_addr),
   .z_din(z_din),
   .z_addr(z_wr_addr),
   .z_wr_en(z_wr_en)
);

bram #(
  .BRAM_DATA_WIDTH(DATA_WIDTH),
  .BRAM_ADDR_WIDTH(ADDR_WIDTH)
) x_inst (
  .clock(clock),
  .rd_addr(x_rd_addr),
  .wr_addr(x_wr_addr),
  .wr_en(x_wr_en),
  .dout(x_dout),
  .din(x_din)
);
```

```
bram #(
  .BRAM_DATA_WIDTH(DATA_WIDTH),
  .BRAM_ADDR_WIDTH(ADDR_WIDTH)
) y_inst (
  .clock(clock),
  .rd_addr(y_rd_addr),
  .wr_addr(y_wr_addr),
  .wr_en(y_wr_en),
  .dout(y_dout),
  .din(y_din)
);

bram #(
  .BRAM_DATA_WIDTH(DATA_WIDTH),
  .BRAM_ADDR_WIDTH(ADDR_WIDTH)
) z_inst (
  .clock(clock),
  .rd_addr(z_rd_addr),
  .wr_addr(z_wr_addr),
  .wr_en(z_wr_en),
  .dout(z_dout),
  .din(z_din)
);

endmodule
```

VECTORSUM TOP ARCHITECTURE



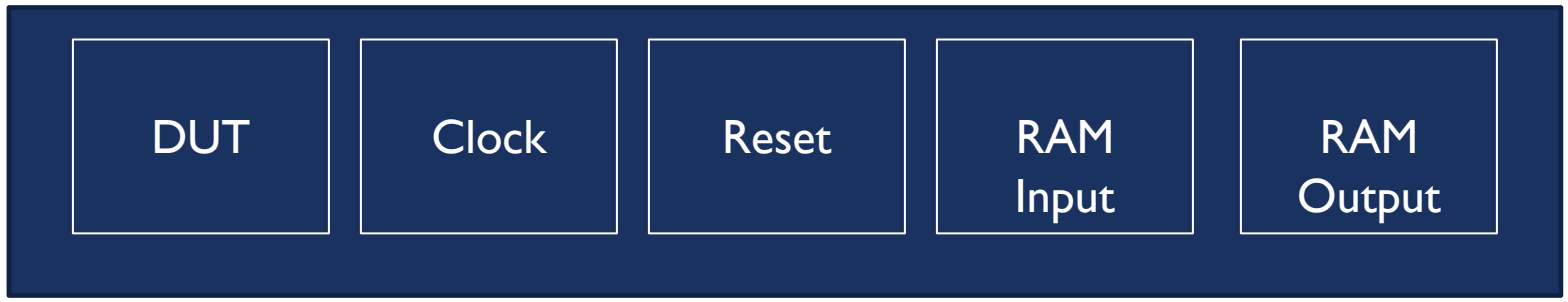
DESIGN VALIDATION

- Capture input and output data of a software model
- Use data to simulate and test your RTL model

```
void add_n ( int *X, int *Y, int *Z, int n )
{
        for ( int i = 0; i < n; i++ ) {
                Z[i] = X[i] + Y[i];
        }
}

int main()
{
        const int n = 64;
        int X[n], Y[n], Z[n];

        for (int i = 0; i < n; i++) {
                X[i] = rand();
                Y[i] = rand();
        }
        vectorsum( X, Y, Z, n );

        FILE * x_file = fopen("x.txt", "w");
        FILE * y_file = fopen("y.txt", "w");
        FILE * z_file = fopen("z.txt", "w");

        for (int i = 0; i < n; i++) {
                fprintf( x_file,"%08x\n", X[i] );
                fprintf( y_file,"%08x\n", Y[i] );
                fprintf( z_file,"%08x\n", Z[i] );
        }
        fclose( x_file );
        fclose( y_file );
        fclose( z_file );
        return 0;
}
```

TESTBENCH DESIGN

- Simulation processes
  - Design under test (DUT) – the design architecture
  - Clock - Instantiates the various clock signals for the different design components.
  - Reset - Asynchronous reset for the signals in the design
  - RAM Input – Reads input data from file and writes to the input memory.
  - RAM Output – Reads output data from memory and writes to file. Also compares output data to software data.



VECTORSUM TESTBENCH

```
`timescale 1 ns / 1 ns

module vectorsum_tb ();

localparam string X_NAME = "x.txt";
localparam string Y_NAME = "y.txt";
localparam string Z_NAME = "z.txt";
localparam DATA_WIDTH = 32;
localparam ADDR_WIDTH = 10;
localparam VECTOR_SIZE = 64;
localparam CLOCK_PERIOD = 10;

logic clock = 1'b0;
logic reset = 1'b0;
logic start = 1'b0;
logic done;

logic [DATA_WIDTH-1:0] x_din;
logic [ADDR_WIDTH-1:0] x_wr_addr;
logic x_wr_en;
logic [DATA_WIDTH-1:0] y_din;
logic [ADDR_WIDTH-1:0] y_wr_addr;
logic y_wr_en;
logic [DATA_WIDTH-1:0] z_dout;
logic [ADDR_WIDTH-1:0] z_rd_addr;

logic x_write_done = '0;
logic y_write_done = '0;
logic z_read_done = '0;
integer z_errors = '0;
```

```
vectorsum_top #(
   .DATA_WIDTH(DATA_WIDTH),
   .ADDR_WIDTH(ADDR_WIDTH),
   .VECTOR_SIZE(VECTOR_SIZE)
) vectorsum_top_inst (
  .clock(clock),
  .reset(reset),
  .start(start),
  .done(done),
  .x_wr_addr(x_wr_addr),
  .x_wr_en(x_wr_en),
  .x_din(x_din),
  .y_wr_addr(y_wr_addr),
  .y_wr_en(y_wr_en),
  .y_din(y_din),
  .z_rd_addr(z_rd_addr),
  .z_dout(z_dout)
);

// clock process
always begin
  #(CLOCK_PERIOD/2) clock = 1'b1;
  #(CLOCK_PERIOD/2) clock = 1'b0;
end

// reset process
initial begin
  #(CLOCK_PERIOD) reset = 1'b1;
  #(CLOCK_PERIOD) reset = 1'b0;
end
```

```
initial begin
  time start_time, end_time;

  @(negedge reset);
  wait(x_write_done && y_write_done);
  @(posedge clock);
  start_time = $time;
  $display("@ %0t: Beginning simulation...", start_time);

  @(posedge clock);
  #(CLOCK_PERIOD) start = 1'b1;
  #(CLOCK_PERIOD) start = 1'b0;
  wait(done);

  end_time = $time;
  $display("@ %0t: Simulation completed.", end_time);
  wait(z_read_done);
  $display("Total simulation cycle count: %0d",
         (end_time-start_time)/CLOCK_PERIOD);
  $display("Total error count: %0d", z_errors);

  $finish;
end
```

VECTORSUM TESTBENCH (CONT.)

```
initial begin : x_write
  integer fd, x, count;
  x_wr_addr = '0;
  x_wr_en = 1'b0;

  @(negedge reset);
  $display("@ %0t: Loading file %s...",
        $time, X_NAME);
  fd = $fopen(X_NAME, "r");

  for (x = 0; x < VECTOR_SIZE; x++) begin
     @(posedge clock);
     count = $fscanf(fd, "%h", x_din);
     x_wr_addr = x;
     x_wr_en = 1'b1;
  end

  @(posedge clock);
  x_wr_en = 1'b0;
  $fclose(fd);
  x_write_done = 1'b1;
end
```

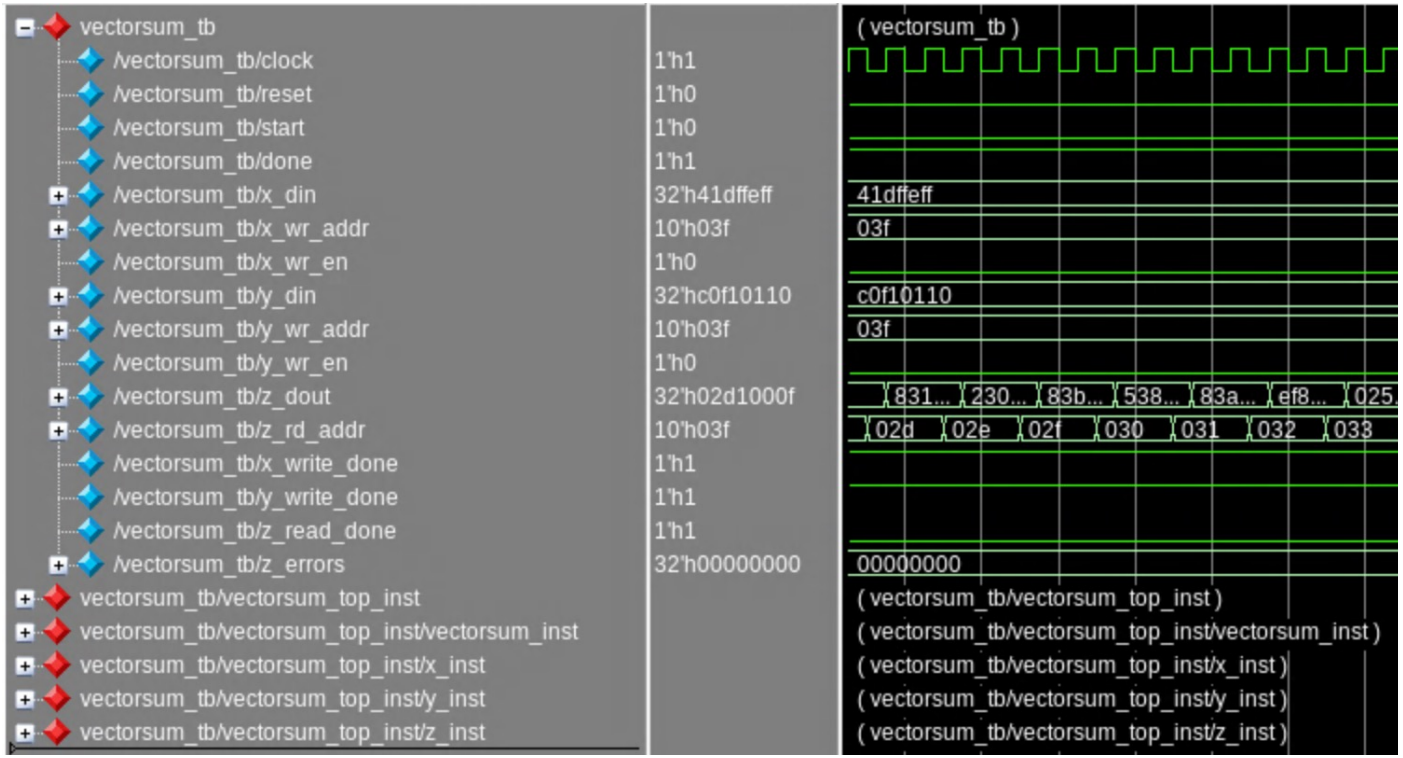
```
initial begin : y_write
  integer fd, y, count;
  y_wr_addr = '0;
  y_wr_en = 1'b0;
  @(negedge reset);
  $display("@ %0t: Loading file %s...",
        $time, Y_NAME);
  fd = $fopen(Y_NAME, "r");

  for (y = 0; y < VECTOR_SIZE; y++) begin
     @(posedge clock);
     count = $fscanf(fd, "%h", y_din);
     y_wr_addr = y;
     y_wr_en = 1'b1;
   end

   @(posedge clock);
   y_wr_en = 1'b0;
   $fclose(fd);
   y_write_done = 1'b1;
end
```

```
initial begin : z_write
   integer fd, z, count;
   logic [DATA_WIDTH-1:0] z_data_cmp, z_data_read;
   z_rd_addr = '0;

   @(negedge reset);
   wait(done);
   @(negedge clock);

   $display("@ %0t: Comparing file %s...", $time, Z_NAME);
   fd = $fopen(Z_NAME, "r");

   for (z = 0; z < VECTOR_SIZE; z++) begin
      @(negedge clock);
      z_rd_addr = z;
      @(negedge clock);
      count = $fscanf(fd, "%h", z_data_cmp);
      z_data_read = z_dout;
      if (z_data_read != z_data_cmp) begin
         z_errors++;
         $display("@ %0t: %s(%0d): ERROR: %h != %h at address 0x%h.",
            $time, Z_NAME, z+1, z_data_read, z_data_cmp, z);
      end
      @(posedge clock);
   end
   $fclose(fd);
   z_read_done = 1'b1;
end

endmodule
```

VECTORSUM SIMULATION

```
setenv LMC_TIMEUNIT -9
vlib work
vmap work work

vlog -work work "../sv/bram.sv"
vlog -work work "../sv/vectorsum.sv"
vlog -work work "../sv/vectorsum_top.sv"
vlog -work work "../sv/vectorsum_tb.sv"

vsim -classdebug -voptargs=+acc +notimingchecks -L work work.vectorsum_tb -wlf vectorsum.wlf

add wave -noupdate -group vectorsum_tb
add wave -noupdate -group vectorsum_tb -radix hexadecimal /vectorsum_tb/*
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst -radix hexadecimal /vectorsum_tb/vectorsum_top_inst/*
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/vectorsum_inst
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/vectorsum_inst -radix hexadecimal /vectorsum_tb/vectorsum_top_inst/vectorsum_inst/*
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/x_inst
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/x_inst -radix hexadecimal /vectorsum_tb/vectorsum_top_inst/x_inst/*
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/y_inst
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/y_inst -radix hexadecimal /vectorsum_tb/vectorsum_top_inst/y_inst/*
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/z_inst
add wave -noupdate -group vectorsum_tb/vectorsum_top_inst/z_inst -radix hexadecimal /vectorsum_tb/vectorsum_top_inst/z_inst/*

run -all
```

VECTORSUM SIMULATION WAVEFORM




```
# Top level modules:
# vectorsum_tb
# End time: 14:41:37 on Jan 11,2022, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -classdebug -voptargs="+acc" "+notimingchecks" -L work
work.vectorsum_tb -wlf vectorsum.wlf
# Start time: 14:41:37 on Jan 11,2022
# ** Note: (vsim-8009) Loading existing optimized design _opt
# Loading sv_std.std
# Loading work.vectorsum_tb(fast)
# Loading work.vectorsum_top(fast)
# Loading work.vectorsum(fast)
# Loading work.bram(fast)
# @ 20: Loading file y.txt...
# @ 20: Loading file x.txt...
# @ 675: Beginning simulation...
# @ 1995: Simulation completed.
# @ 2000: Comparing file z.txt...
# Total simulation cycle count: 132
# Total error count: 0
# ** Note: $finish : ../sv/vectorsum_tb.sv(84)
# Time: 3285 ns Iteration: 2 Instance: /vectorsum_tb
# 1
# Break in Module vectorsum_tb at ../sv/vectorsum_tb.sv line 84
```

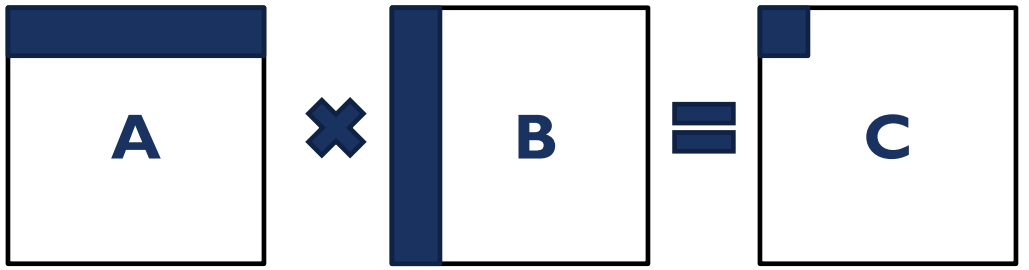
MATRIX MULTIPLICATION - APPLICATIONS

- 3D Projection
- Robotics (Position & Orientation)
- Transformation (Forward / Inverse Kinematics)
- Digital Signal Processing
- Computer Graphics
- Virtual Reality / Augmented Reality
- Machine Learning (Regression)

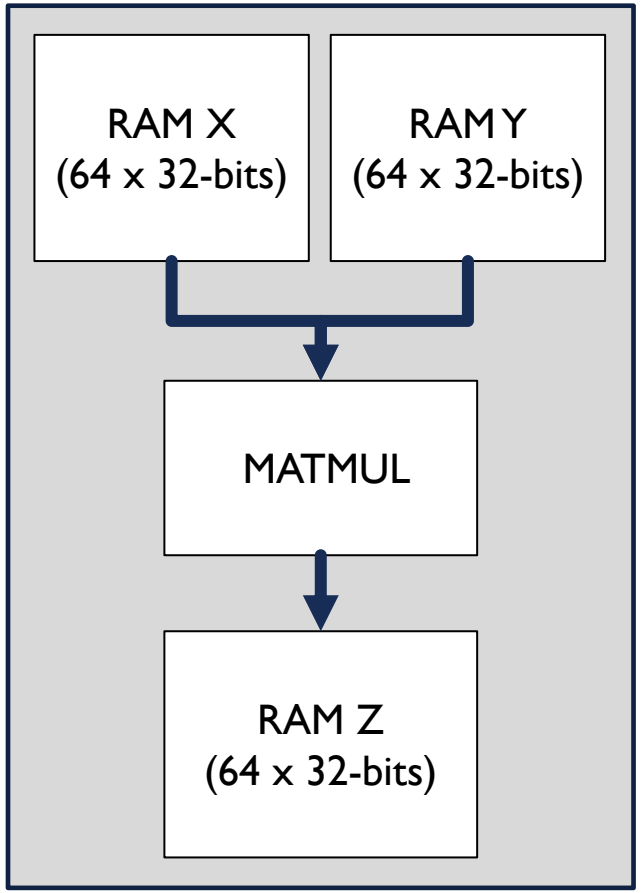
MATRIX MULTIPLICATION

```
void matmul( int *A, int *B, int *C, int N )
{
    for ( i=0; i<N; i++ ) {
        for ( j=0; j<N; j++ ) {
            C[i][j] = 0;
            for ( k=0; k<N; k++ ) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

Time complexity is $O(n^3)$



MATRIX MULTIPLICATION ARCHITECTURE



NEXT…

- Assignment #2: Matrix Multiply