Kevin Hoser and Alex Schendel

EE 435

Project 4

9 April 2020

**RAM**

`include "HEADER.vh"

module RAM(Data, clk, rdEn, wrEn, reset, Addr);

parameter MEMDEPTH = 256;

parameter DWIDTH = 32;

parameter AWIDTH = 8;

inout [DWIDTH-1:0] Data;

input clk;

input rdEn, wrEn; // active high enable, one-hot

input [AWIDTH-1:0] Addr;

input reset;

tri [DWIDTH-1:0] Data;

reg [DWIDTH-1:0] dataOut;

reg [DWIDTH-1:0] storage [MEMDEPTH-1:0];

integer i;

assign Data = (~wrEn) ? storage[Addr] : {DWIDTH{1'bz}};

//Resets the entire RAM by iterating through the registers

always @(negedge reset) begin

    for (i = 0; i < MEMDEPTH; i = i + 1) begin

        storage[i] = 32'b0;

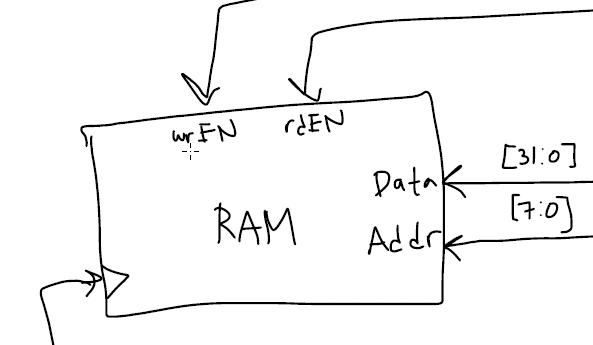
    end

end

//Initialize a DFF to set the data properly. See header.vh

`DFFE(storage[Addr], Data, wrEn, reset, clk)

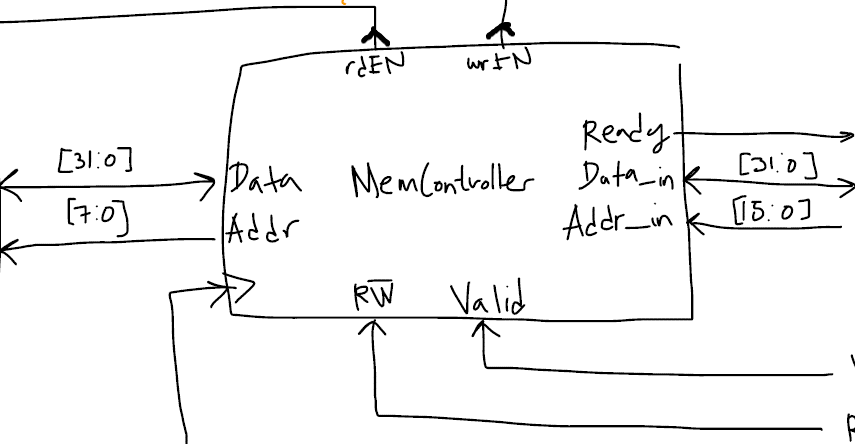
endmodule



**Memory Controller**

module MemControl(Data\_in, Data, rdEn, wrEn, Addr, Ready, clk, Addr\_in, RW, Valid);

parameter MEMDEPTH = 256;

parameter DWIDTH = 32;

parameter AWIDTH = 8;

inout Data\_in;

inout Data;

output reg rdEn, wrEn;

output reg [AWIDTH-1:0] Addr;

output reg Ready;

input clk;

input [15:0] Addr\_in;

input RW;

input Valid;

tri [DWIDTH-1:0] Data\_in, Data;

assign Data = (wrEn) ? Data\_in : {DWIDTH{1'bz}};

assign Data\_in = (RW) ? Data : {DWIDTH{1'bz}};

always @(posedge clk) begin

    // only run when the inputs valid

    if(Valid) begin

        Ready = 0;

        // wait one cycle to read the address

        wait(~clk);

        wait(clk);

        Addr = Addr\_in[AWIDTH-1:0];

        if (~RW)

            wrEn = 1; // write

        else

            rdEn = 1; // read

        // wait one cycle to write/read from ram

        wait(~clk);

        wait(clk);

        // add two cycles of arbitary delay

        wait(~clk);

        wait(clk);

        wait(~clk);

        wait(clk);

        // tell CPU we're done and disable RAM

        Ready = 1;

        rdEn = 0;

        wrEn = 0;

    end

end

endmodule

**Header File**

`ifndef HEADER

`define HEADER

//Macro to properly set the data of a reg in the RAM

`define DFFE(q, d, en, reset, clk) \

    always @(posedge clk) begin \

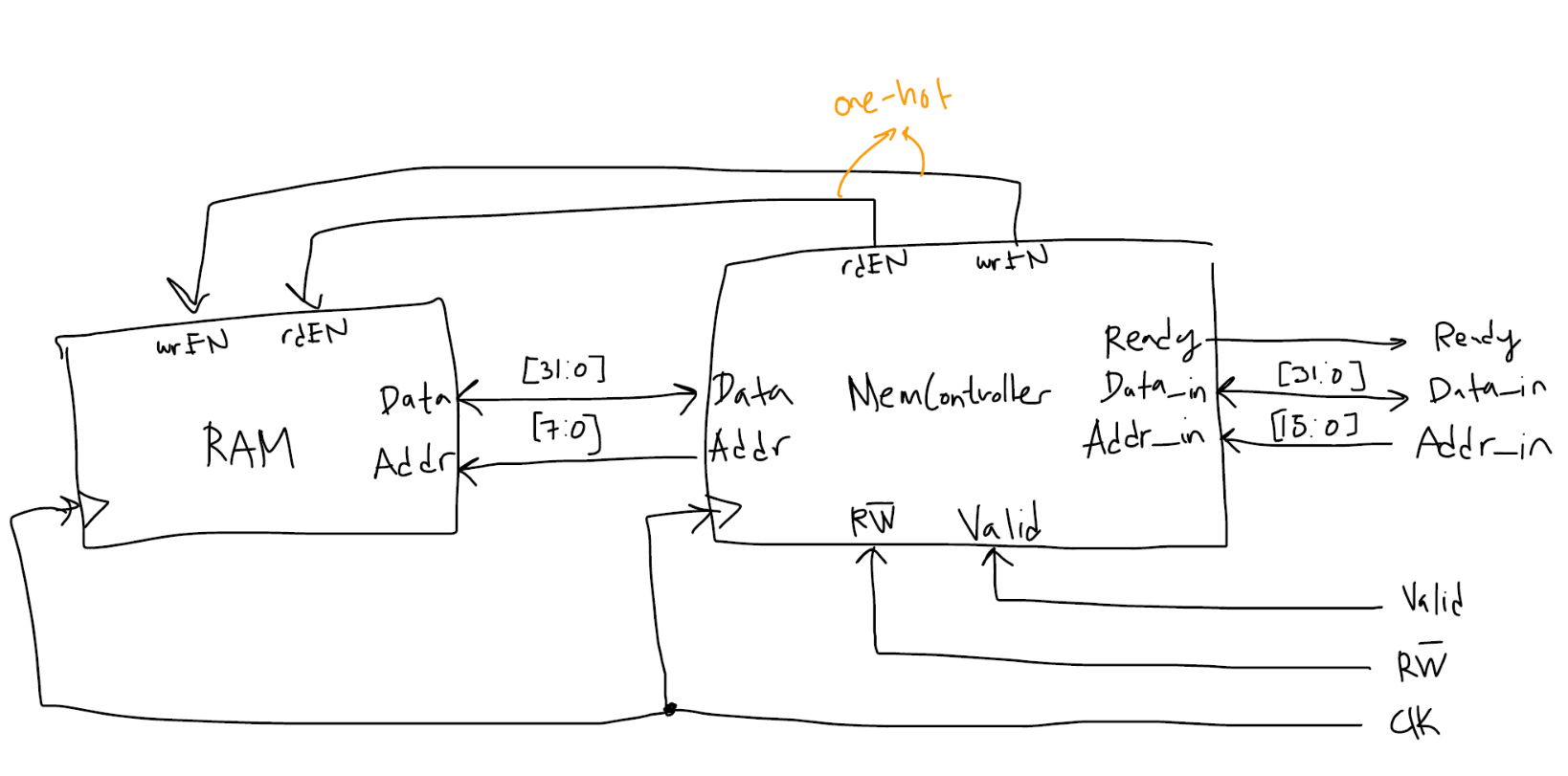
        if (reset) \

            q <= (en) ? d : q; \

    end

`endif

**RAM / Memory Controller Combined Block-Diagram**



**Memory Controller Testbench**

`timescale 1ns / 1ns

module MemControl\_tb();

parameter MEMDEPTH = 256;

parameter DWIDTH = 32;

parameter AWIDTH = 8;

// component variables

tri [DWIDTH-1:0] Data\_in, Data;

wire rdEn, wrEn;

wire [AWIDTH-1:0] Addr;

wire Ready;

reg clk;

reg reset;

reg [15:0] Addr\_in;

reg RW;

reg Valid;

MemControl dut(Data\_in, Data, rdEn, wrEn, Addr, Ready, clk, Addr\_in, RW, Valid);

RAM ram(Data, clk, rdEn, wrEn, reset, Addr);

// clock

always begin

    clk = 0;

    forever #10 clk = !clk;

end

reg [DWIDTH-1:0] myData;

assign Data\_in = (~RW) ? myData : {DWIDTH{1'bz}};

initial

begin

    // reset RAM

    reset = 1;

    #5 reset = 0;

    #5 reset = 1;

    // write to Addr 0

    myData = $random; Addr\_in = 16'h0000; RW = 0; Valid = 1;

    wait(~Ready);

    wait(Ready);

    Valid = 0;

    // write to Addr 1

    #20 myData = $random; Addr\_in = 16'h0001; RW = 0; Valid = 1;

    wait(~Ready);

    wait(Ready);

    Valid = 0;

    // write to Addr 2

    #20 myData = $random; Addr\_in = 16'h0002; RW = 0; Valid = 1;

    wait(~Ready);

    wait(Ready);

    Valid = 0;

    // read from Addr 0

    #20 myData = {DWIDTH{1'bz}}; Addr\_in = 16'h0000; RW = 1; Valid = 1;

    wait(~Ready);

    wait(Ready);

    Valid = 0;

    // read from Addr 2

    #20 myData = {DWIDTH{1'bz}}; Addr\_in = 16'h0002; RW = 1; Valid = 1;

    wait(~Ready);

    wait(Ready);

    Valid = 0;

    // read from Addr 1

    #20 myData = {DWIDTH{1'bz}}; Addr\_in = 16'h0001; RW = 1; Valid = 1;

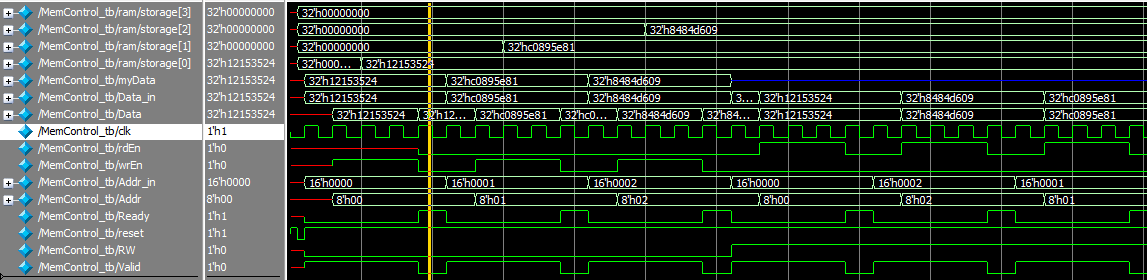
    wait(~Ready);

    wait(Ready);

    Valid = 0;

end

endmodule



**Multiplier**

module multiply\_32(prod, a, b);

parameter WIDTH = 32;

output [WIDTH-1:0] prod;

input [WIDTH-1:0] a, b;

assign prod = a \* b;

endmodule

**Divider**

module divide\_32(out, a, b);

parameter WIDTH = 32;

output [WIDTH-1:0] out;

input [WIDTH-1:0] a,b;

assign out = a / b;

endmodule

**Comparator**

module comp\_32(lt, eq, gt, a, b);

parameter WIDTH = 32;

output [WIDTH-1:0] lt, eq, gt;

input [WIDTH-1:0] a, b;

assign lt = {{31'b0}, {a < b}};

assign eq = {{31'b0}, {a == b}};

assign gt = {{31'b0}, {a > b}};

endmodule

**ALU**

module ALU(out, a, b, op);

output [31:0] out;

input [31:0] a, b;

input [3:0] op;

wire [31:0] add, sub, my\_and, my\_or, my\_xor, my\_xnor,

            my\_shl, my\_shr, my\_sra, my\_mult, my\_div,

            my\_lt, my\_gt, my\_eq;

full\_adder\_32 adder(, add, a, b, 1'b0);

full\_subtractor\_32 subber(, sub, a, b, 1'b0);

and32 ander(my\_and, a, b);

or32 orer(my\_or, a, b);

xor32 xorer(my\_xor, a, b);

xnor32 xnorer(my\_xnor, a, b);

shiftLeft32 shiftLefter(my\_shl, a, b);

shiftRight32 shiftRighter(my\_shr, a, b);

shiftRight32Arithmetic shiftRightAer(my\_sra, a, b);

multiply\_32 multiplierer(my\_mult, a, b);

divide\_32 dividerer(my\_div, a, b);

comp\_32 comparatorer(my\_lt, my\_eq, my\_gt, a, b);

mux16\_32 muxer(out, add, sub, my\_mult, my\_div, my\_lt, my\_eq, my\_gt, ,

    my\_and, my\_or, my\_xor, my\_xnor, my\_shl, my\_shr, my\_sra, , op);

endmodule

**ALU Testbench**

`timescale 1ns / 1ns

module ALU\_tb();

wire [31:0] out;

reg [31:0] a, b;

reg [3:0] op;

ALU dut(out, a, b, op);

initial // Test stimulus

  begin

    #10  a = 32'hBC157222; b =  32'h222751CB; op = 4'h0;

    #10  a = 32'h00000000; b =  32'h583bd1cc; op = 4'h1;

    #10  a = 32'h10101010; b =  32'h583bd1cc; op = 4'h8;

    #10  a = 32'h0189ABCD; b =  32'h1DC74A54; op = 4'h9;

    #10  a = 32'he9eec208; b =  32'h583bd1cc; op = 4'hA;

    #10  a = 32'h1fbc8148; b =  32'h20ce01ee; op = 4'hB;

    #10  a = 32'h2416e099; b =  32'h3ef9d5ec; op = 4'h2;

    #10  a = 32'h687f3b5a; b =  32'h71252c6f; op = 4'h3;

    #10  a = 32'h555984ab; b =  32'h6c886316; op = 4'h4;

    #10  a = 32'hFFFEFFFF; b =  32'h00000004; op = 4'hC;

    #10  a = 32'hFFFF7FFF; b =  32'h00000009; op = 4'hD;

    #10  a = 32'h80000000; b =  32'h0000001E; op = 4'hE;

    #10  a = 32'h23337af9; b =  32'h89bfe491; op = 4'hF; // empty

    #10  a = 32'h00000000; b =  32'h00000000; op = 4'h0;

    #10  a = 32'h62202dfd; b =  32'h15ff1963; op = 4'h2;

    #10  a = 32'hd9de66ad; b =  32'h2edd939c; op = 4'h3;

    #10  a = 32'h7fa3036c; b =  32'h3e845481; op = 4'h4;

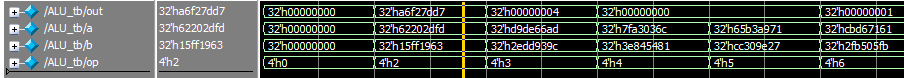
    #10  a = 32'h65b3a971; b =  32'hcc309e27; op = 4'h5;

    #10  a = 32'hcbd67161; b =  32'h2fb505fb; op = 4'h6;

    #10 $stop;

  end

endmodule



**CPU**

`include "header.vh"

module CPU;

parameter IWIDTH = 32;

parameter DWIDTH = 32;

parameter AWIDTH = 10;

reg clk;

// configure IR, MDR, MAR

reg [IWIDTH-1:0] IR;

reg [DWIDTH-1:0] MDR;

reg [AWIDTH-1:0] MAR;

wire [IWIDTH-1:0] IR\_nxt;

wire [DWIDTH-1:0] MDR\_nxt;

wire [AWIDTH-1:0] MAR\_nxt;

`DFFE(IR, IR\_nxt, 1'b1, 1'b1, clk);

`DFFE(MDR, MDR\_nxt, 1'b1, 1'b1, clk);

`DFFE(MAR, MAR\_nxt, 1'b1, 1'b1, clk);

// configure memory controller and RAM

tri [DWIDTH-1:0] Data\_in, Data;

wire rdEn, wrEn;

wire [7:0] Addr;

wire Ready;

reg reset;

reg [15:0] Addr\_in;

reg RW;

reg Valid;

assign Data\_in = (~RW) ? MDR : {DWIDTH{1'bz}};

MemControl m\_control(Data\_in, Data, rdEn, wrEn, Addr, Ready, clk, MAR[7:0], RW, Valid);

RAM ram(Data, clk, rdEn, wrEn, reset, Addr);

// clock

always begin

    clk = 0;

    forever #10 clk = !clk;

end

endmodule