Jack Mravunac

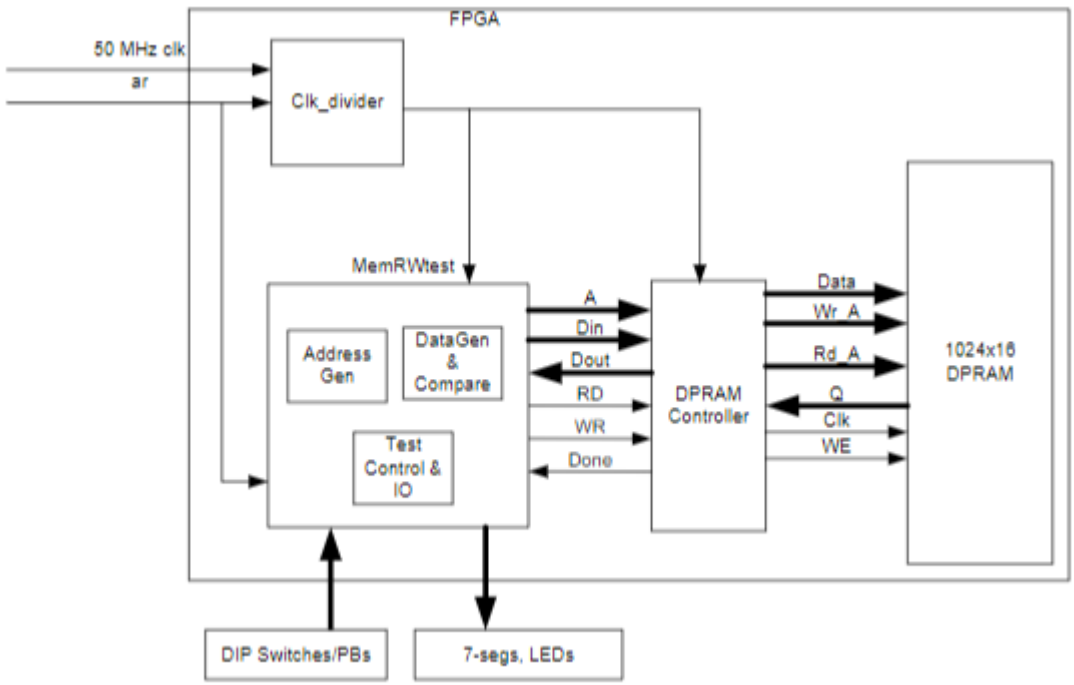
Project 3 – DE2-115 Internal Memory Controller

Scope:

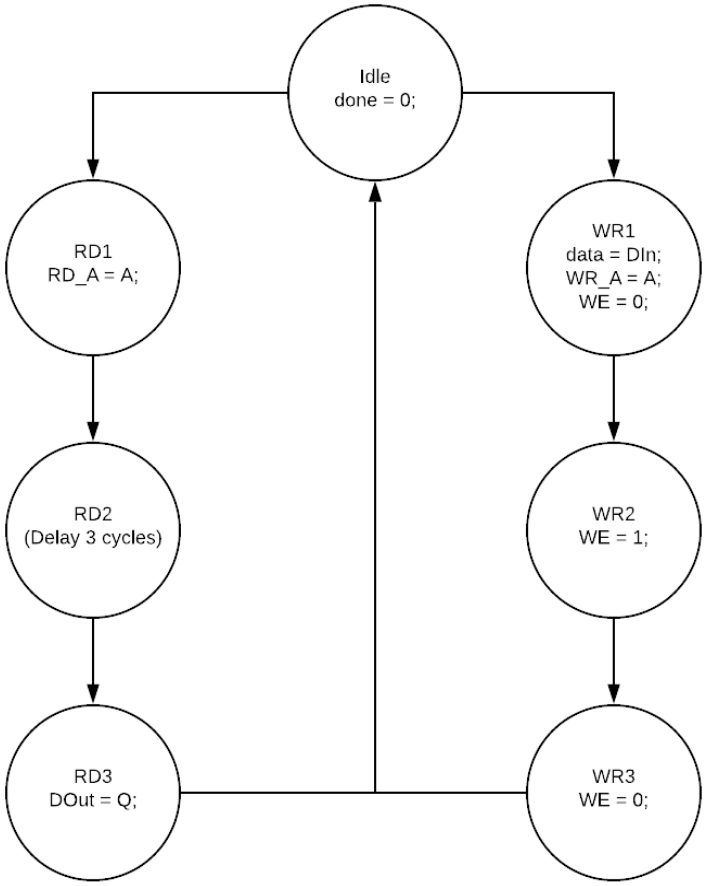
The goal of this project was to design, implement, and test a memory controller for an internal DPRAM. The project contained 3 modules, the DPRAM, controller, and a test module. The goal of the test module was to perform an internal test and also offer the user I/O to load and read data from any of the memory locations.

Design Strategy:

The Project was built to the specifications in the diagram below excluding the clock divider which was not needed.



The DPRAM module was created in Quartus through the IP setup wizard. The controller was implemented following the state machine design we went over in class. State diagram shown below. The MemRWtest module implemented two similar state machines, one for the internal test, and one for the I/O read and write. The active state machine is controlled by a switch on the board. When in the internal test mode, the button on the far left is pushed to start the test, at the end of the test the green LED will light up if the test passes or the red LED will light up if the test fails. In use I/O mode the buttons from left to right are address, read, write, and reset. The address button will take in the 10 rightmost switches as the address and the write uses the 16 rightmost to take in the data to be written.

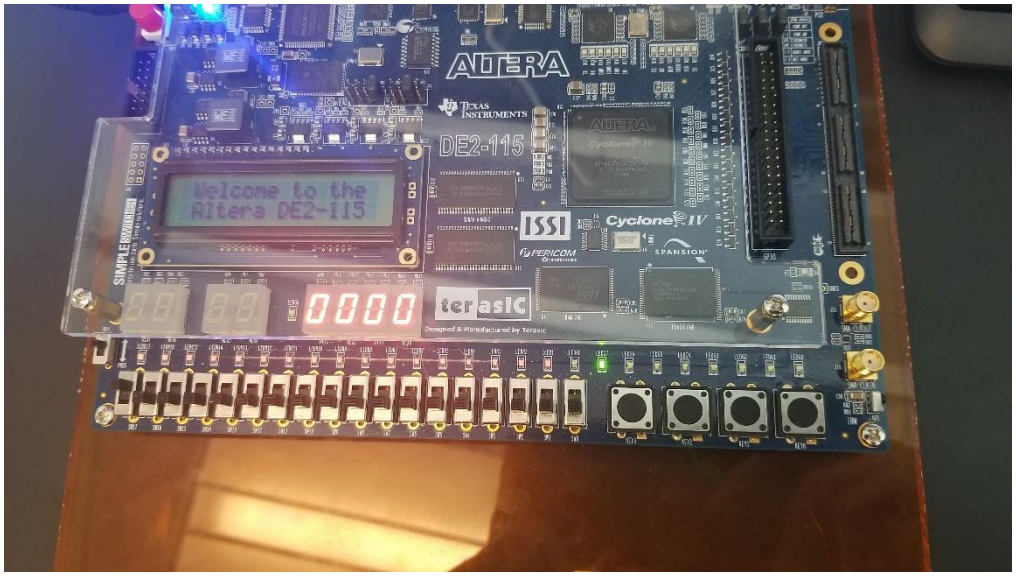


Results:

The following link contains video showing the I/O working as expected. The hex value F is stored in memory location 1 and the hex value E is store in location 2.

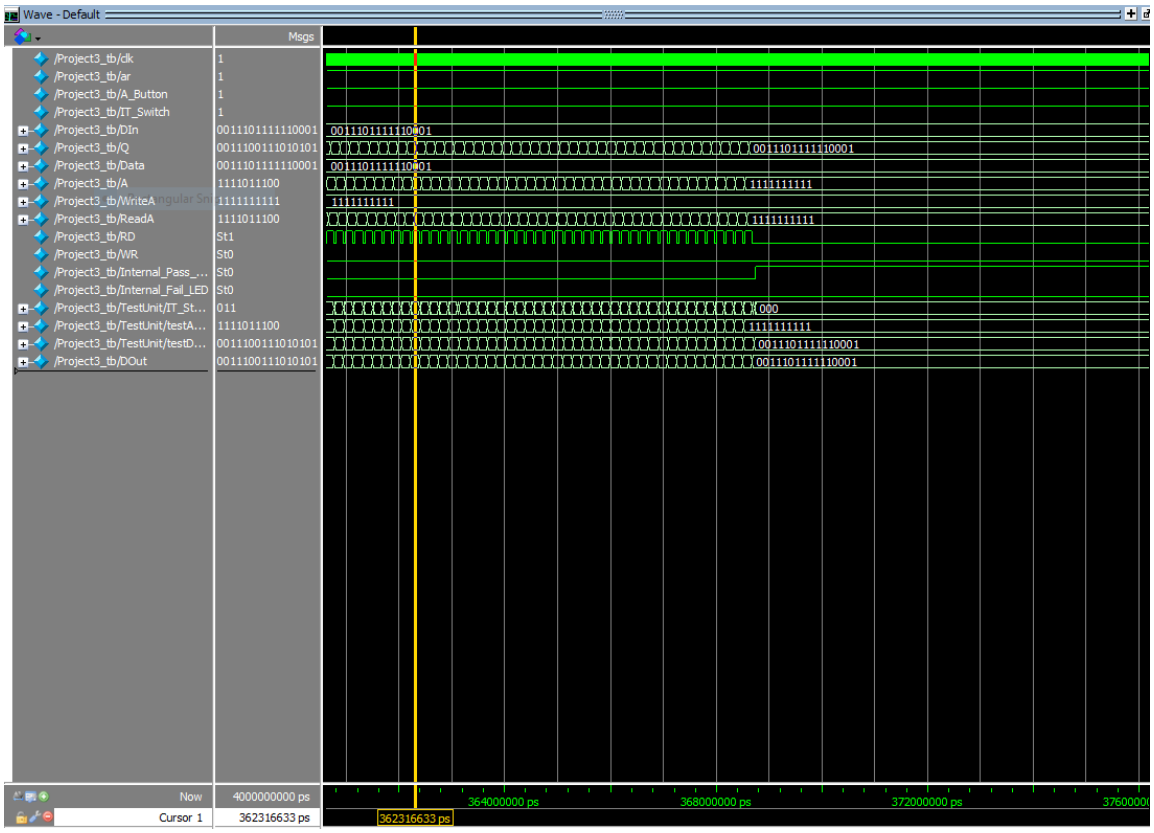
https://photos.app.goo.gl/BiKGWyWHXTih5EFc8

The picture below shows the board in internal test mode with the green LED on showing it has passed the test.



From Quartus the Fmax is 99.39 MHz and the board uses 232 logic elements which is <1%. The DPRAM module is clearly seen as implemented with Quartus showing 16,384 bits which is equivalent to 1024 X 16.

The screenshot below shows the internal test running in model sim where it compares the data written which was the address * 15 to show all 16 bits working, to the address being read at the time * 15 as well. When all 1024 addresses are read the internal pass LED turns on which is also shown.



What I Learned:

In doing this project I learned more about the importance of planning out the code before you write it. Having created the state diagrams before I even started writing any code, I found that I was able to easily translate the diagrams into code quickly and was able to have success after only fixing a couple of syntax errors.

Conclusions:

This project went very smoothly after initial planning. The hardest thing I actually had to deal with was modelsim not working on my personal computer. I spent more than a couple hours trying to debug it before attempting on a lab computer where it worked fine on my first attempt.

I feel like this project helped me to increase my understanding of how memory works at a level I had not seen before. This will definitely be a project I pull from in the future as I work on other digital design projects.

Appendix:

Project_3_top:

```
/*
/  Jack Mravunac
/  Project3_top.v
/  Top file for DPRAM controller project
/  27 February 2020
*/

`timescale 100 ns / 1 ns

module Project3_top(clk, ar, universalIn, A_Button, Rd_Button, Wr_Button,
IT_Switch,
    Done_LED, Internal_Pass_LED, Internal_Fail_LED, SevenSeg_Zero_Out,
SevenSeg_One_Out, SevenSeg_Two_Out, SevenSeg_Three_Out);

input clk, ar, A_Button, Rd_Button, Wr_Button, IT_Switch;
input [15:0] universalIn;

output Done_LED, Internal_Pass_LED, Internal_Fail_LED;
output [6:0] SevenSeg_Zero_Out, SevenSeg_One_Out, SevenSeg_Two_Out,
SevenSeg_Three_Out;

wire [3:0] SevenSeg_Zero, SevenSeg_One, SevenSeg_Two, SevenSeg_Three;
wire [15:0] DOut, DIn, Q, Data;
wire [9:0] A, WriteA, ReadA;
wire Done, RD, WR, WE;


MemRWTest TestUnit(.clk(clk), .ar(ar), .UniversalIn(universalIn),
.DOut(DOut), .Done(Done), .A_Button(A_Button), .Rd_Button(Rd_Button),
.Wr_Button(Wr_Button), .IT_Switch(IT_Switch),
    .A(A), .DIn(DIn), .RD(RD), .WR(WR), .Done_LED(Done_LED),
.Internal_Pass_LED(Internal_Pass_LED), .Internal_Fail_LED(Internal_Fail_LED),
```

```
    .SevenSeg_Zero(SevenSeg_Zero), .SevenSeg_One(SevenSeg_One),
.SevenSeg_Two(SevenSeg_Two), .SevenSeg_Three(SevenSeg_Three));

sevseg_dec SegZero(.x_in(SevenSeg_Zero), .segs(SevenSeg_Zero_Out));

sevseg_dec SegOne(.x_in(SevenSeg_One), .segs(SevenSeg_One_Out));

sevseg_dec SegTwo(.x_in(SevenSeg_Two), .segs(SevenSeg_Two_Out));

sevseg_dec SegThree(.x_in(SevenSeg_Three), .segs(SevenSeg_Three_Out));

DPRAM_Controller Controller(.clk(clk), .ar(ar), .RD(RD), .WR(WR), .A(A),
.DIn(DIn), .Q(Q),
    .DOut(DOut), .Data(Data), .Wr_A(WriteA), .Rd_A(ReadA), .Done(Done),
.WE(WE));

DPRAM Ram(.clock(clk), .data(Data), .rdaddress(ReadA), .wraddress(WriteA),
.wren(WE),
    .q(Q));

endmodule
```

MemRWTest:

```
/*
/  Jack Mravunac
/  MemRWTest.v
/  Contains I/O and self tests for the DPRAM
/  27 February 2020
*/

`timescale 100 ns / 1 ns

module MemRWTest(clk, ar, UniversalIn, DOut, Done, A_Button, Rd_Button,
Wr_Button, IT_Switch,
    A, DIn, RD, WR, Done_LED, Internal_Pass_LED, Internal_Fail_LED,
SevenSeg_Zero, SevenSeg_One, SevenSeg_Two, SevenSeg_Three);

input clk, ar, Done, A_Button, Rd_Button, Wr_Button, IT_Switch;
input [15:0] UniversalIn, DOut;

output reg [3:0] SevenSeg_Zero, SevenSeg_One, SevenSeg_Two, SevenSeg_Three;
output reg [9:0] A;
output reg [15:0] DIn;
output reg RD, WR, Done_LED, Internal_Pass_LED, Internal_Fail_LED;

parameter [2:0]   Idle = 3'b0, Address = 3'b001, Wr1 = 3'b010, Wr2 = 3'b111,
Wr3 = 3'b011, Rd1 = 3'b100, Rd2 = 3'b101, Rd3 = 3'b110;
parameter [2:0]   Idle2 = 3'b0, AddrImp = 3'b001, Write = 3'b010, Read =
3'b011, Check = 3'b100, Success = 3'b101, Fail = 3'b110;
    reg [2:0] User_State;
    reg [2:0] IT_State;
    reg [2:0] delay, delay2, delay3;
    reg [9:0] testAddr;
    reg [15:0] testData;
```

```
always @(negedge ar or posedge clk)
    if(~ar)
     begin
        A = 10'b0;
        DIn = 16'b0;
        RD = 1'b0;
        WR = 1'b0;
        testAddr = 10'b0;
        testData = 16'b0;
        Done_LED = 1'b0;
        Internal_Pass_LED = 1'b0;
        Internal_Fail_LED = 1'b0;
        SevenSeg_Zero = 4'b0;
        SevenSeg_One = 4'b0;
        SevenSeg_Two = 4'b0;
        SevenSeg_Three = 4'b0;
     end
    else
     begin
            if(IT_Switch)
             begin
                case(IT_State)
                    Idle2:
                     begin
                        if(~A_Button)
                            IT_State = AddrImp;
                     end

                    AddrImp:
                     begin
                        testAddr = testAddr + 1;
                        testData = testAddr * 15;
                        delay2 = 3'b0;
                        delay3 = 3'b0;
                        A = testAddr;
                        DIn = testData;
                        IT_State = Write;
                     end

                    Write:
                     begin
                        WR = 1'b1;
                        if(delay2 > 3'b110)
                        begin
                            if(testAddr > 1022)
                            begin
                                testAddr = 0;
                                testData = 0;

                                IT_State = Read;
                            end
                            else
                                IT_State = AddrImp;
                        end
                        else
                            delay2 = delay2 + 1;
                     end
```

```
            Read:
             begin
                WR = 1'b0;
                delay2 = 3'b0;
                A = testAddr;
                RD = 1'b1;
                if(delay3 > 3'b110)
                begin
                    RD = 1'b0;
                    delay3 = 3'b0;
                    IT_State = Check;
                end
                else
                    delay3 = delay3 + 1;

             end

            Check:
             begin

                testData = testAddr * 15;
                if(testData != DOut)
                    IT_State = Fail;
                else
                begin
                    if(testAddr > 1022)
                        IT_State = Success;
                    else
                    begin
                        testAddr = testAddr + 1;
                        IT_State = Read;
                    end
                end

             end

            Success:
             begin
                Internal_Pass_LED = 1'b1;
                Internal_Fail_LED = 1'b0;
                IT_State = Idle2;
             end

            Fail:
             begin
                Internal_Pass_LED = 1'b0;
                Internal_Fail_LED = 1'b1;
                IT_State = Idle2;
             end

            default:
             begin
                IT_State = Idle2;
             end
            endcase
        end
```

```
else
 begin
    case(User_State)
        Idle:
        begin
            if(~A_Button)
                User_State = Address;
            else if(~Rd_Button)
                User_State = Rd1;
            else if(~Wr_Button)
                User_State = Wr1;
        end

        Address:
        begin
            A = UniversalIn[9:0];
            User_State = Idle;
        end

        Rd1:
        begin
            Done_LED = 1'b0;
            delay = 3'b0;
            RD = 1'b1;
            User_State = Rd2;
        end

        Rd2:
        begin
            if(delay > 3'b110)
                User_State = Rd3;
            else
                delay = delay + 1;
        end

        Rd3:
        begin
            RD = 1'b0;
            SevenSeg_Zero = DOut[3:0];
            SevenSeg_One = DOut[7:4];
            SevenSeg_Two = DOut[11:8];
            SevenSeg_Three = DOut[15:12];
            Done_LED = 1'b1;
            User_State = Idle;
        end

        Wr1:
        begin
            Done_LED = 1'b0;
            DIn = UniversalIn[15:0];
            User_State = Wr2;
        end

        Wr2:
        begin
            WR = 1'b1;
            User_State = Wr3;
```

```
                    end

                    Wr3:
                    begin
                        WR = 1'b0;
                        Done_LED = 1'b1;
                        User_State = Idle;
                    end

                    default:
                    begin
                        User_State = Idle;
                    end
                endcase
            end

     end


endmodule
```

DPRAM_Controller:

```
/*
/  Jack Mravunac
/  DPRAM_Controller.v
/  Controller for the DPRAM
/  27 February 2020
*/

`timescale 100 ns / 1 ns

module DPRAM_Controller(clk, ar, RD, WR, A, DIn, Q,
    DOut, Data, Wr_A, Rd_A, Done, WE);

    input clk, RD, WR, ar;
    input [9:0] A;
    input [15:0] DIn, Q;

    output reg [15:0] DOut, Data;
    output reg [9:0] Wr_A, Rd_A;
    output reg Done, WE;

    parameter [2:0]   Idle = 3'b0, Wr1 = 3'b001, Wr2 = 3'b010, Wr3 = 3'b011,
Rd1 = 3'b100, Rd2 = 3'b101, Rd3 = 3'b110;
    reg [2:0] state;
    reg [1:0] delay;


    always @(negedge ar or posedge clk)
        if(~ar)
         begin
            state = Idle;
            DOut = 16'b0;
            Data = 16'b0;
```

```
    Wr_A = 10'b0;
    Rd_A = 10'b0;
    Done = 1'b0;
    WE = 1'b0;
 end
else
 begin
        case(state)
            Idle:
            begin
                Done = 1'b0;
                if(RD)
                    state = Rd1;
                  else if(WR)
                    state = Wr1;


            end

            Wr1:
            begin
                Data = DIn;
                Wr_A = A;
                WE = 1'b0;
                state = Wr2;
            end

            Wr2:
            begin
                WE = 1'b1;
                state = Wr3;
            end

            Wr3:
            begin
                WE = 1'b0;
                state = Idle;
            end

            Rd1:
            begin
                Rd_A = A;
                delay = 2'b0;
                state = Rd2;
            end

            Rd2:
            begin
                if(delay > 2'b10)
                    state = Rd3;
                else
                    delay = delay + 1;
            end

            Rd3:
            begin
                DOut = Q;
```

```
                    delay = 2'b0;
                    state = Idle;
                end

                default:
                begin
                    state = Idle;
                end
            endcase
        end

endmodule
```

Project3_tb:

```
/*
/  Jack Mravunac
/  Project3_tb.v
/  Contains test bence for the internal test
/  27 February 2020
*/

`timescale 1 ns / 1 ns

module Project3_tb;
    reg clk, ar, A_Button, Rd_Button, Wr_Button, IT_Switch;
    reg [15:0] universalIn;

    wire [3:0] SevenSeg_Zero, SevenSeg_One, SevenSeg_Two, SevenSeg_Three;
    wire [15:0] DOut, DIn, Q, Data;
    wire [9:0] A, WriteA, ReadA;
    wire Done, RD, WR, WE;


MemRWTest TestUnit(.clk(clk), .ar(ar), .UniversalIn(universalIn),
.DOut(DOut), .Done(Done), .A_Button(A_Button), .Rd_Button(Rd_Button),
.Wr_Button(Wr_Button), .IT_Switch(IT_Switch),
    .A(A), .DIn(DIn), .RD(RD), .WR(WR), .Done_LED(Done_LED),
.Internal_Pass_LED(Internal_Pass_LED), .Internal_Fail_LED(Internal_Fail_LED),
    .SevenSeg_Zero(SevenSeg_Zero), .SevenSeg_One(SevenSeg_One),
.SevenSeg_Two(SevenSeg_Two), .SevenSeg_Three(SevenSeg_Three));

sevseg_dec SegZero(.x_in(SevenSeg_Zero), .segs(SevenSeg_Zero_Out));

sevseg_dec SegOne(.x_in(SevenSeg_One), .segs(SevenSeg_One_Out));

sevseg_dec SegTwo(.x_in(SevenSeg_Two), .segs(SevenSeg_Two_Out));

sevseg_dec SegThree(.x_in(SevenSeg_Three), .segs(SevenSeg_Three_Out));

DPRAM_Controller Controller(.clk(clk), .ar(ar), .RD(RD), .WR(WR), .A(A),
.DIn(DIn), .Q(Q),
    .DOut(DOut), .Data(Data), .Wr_A(WriteA), .Rd_A(ReadA), .Done(Done),
.WE(WE));
```

```
DPRAM Ram(.clock(clk), .data(Data), .rdaddress(ReadA), .wraddress(WriteA),
.wren(WE),
    .q(Q));

initial
  begin
    clk = 1'b0; // set to 0 so toggling can occur

    ar = 1'b1;   // Start reset at 1

    #1 ar = 1'b0; // Set reset to 0 after 5 ns
    #20 ar = 1'b1; // Set reset to 1

    #20 IT_Switch = 1'b1;
    #100 A_Button = 1'b1;
    #100 A_Button = 1'b0;
    #20 A_Button = 1'b1;
  end


// Controls the test clock
always
    #10 clk = ~clk;     // For 20 ns period (50 MHz)

endmodule
```

sevseg_dec:

```
// Seven-segment decoder for hexadecimal values
// seven_seg_decoder.v
// Don M. Gruenbacher
// Jan. 23, 2006

module sevseg_dec(x_in, segs);
    input [3:0]   x_in;
    output [6:0] segs;

assign segs =
        (x_in == 4'h0) ? 7'b1000000 :
        (x_in == 4'h1) ? 7'b1111001 :
        (x_in == 4'h2) ? 7'b0100100 :
        (x_in == 4'h3) ? 7'b0110000 :
        (x_in == 4'h4) ? 7'b0011001 :
        (x_in == 4'h5) ? 7'b0010010 :
        (x_in == 4'h6) ? 7'b0000010 :
        (x_in == 4'h7) ? 7'b1111000 :
        (x_in == 4'h8) ? 7'b0000000 :
        (x_in == 4'h9) ? 7'b0010000 :
        (x_in == 4'ha) ? 7'b0001000 :
        (x_in == 4'hb) ? 7'b0000011 :
        (x_in == 4'hc) ? 7'b1000110 :
        (x_in == 4'hd) ? 7'b0100001 :
        (x_in == 4'he) ? 7'b0000110 :
                         7'b0001110 ;
```

```
endmodule
```

DPRAM:

```
// megafunction wizard: %RAM: 2-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: DPRAM.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.1.0 Build 590 10/25/2017 SJ Lite Edition
// ************************************************************


//Copyright (C) 2017  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module DPRAM (
    clock,
    data,
    rdaddress,
    wraddress,
    wren,
    q);

    input     clock;
    input   [15:0]  data;
    input   [9:0]  rdaddress;
    input   [9:0]  wraddress;
    input     wren;
    output  [15:0]  q;
```

```
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri1      clock;
    tri0      wren;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [15:0] sub_wire0;
    wire [15:0] q = sub_wire0[15:0];

    altsyncram  altsyncram_component (
                .address_a (wraddress),
                .address_b (rdaddress),
                .clock0 (clock),
                .data_a (data),
                .wren_a (wren),
                .q_b (sub_wire0),
                .aclr0 (1'b0),
                .aclr1 (1'b0),
                .addressstall_a (1'b0),
                .addressstall_b (1'b0),
                .byteena_a (1'b1),
                .byteena_b (1'b1),
                .clock1 (1'b1),
                .clocken0 (1'b1),
                .clocken1 (1'b1),
                .clocken2 (1'b1),
                .clocken3 (1'b1),
                .data_b ({16{1'b1}}),
                .eccstatus (),
                .q_a (),
                .rden_a (1'b1),
                .rden_b (1'b1),
                .wren_b (1'b0));
    defparam
        altsyncram_component.address_aclr_b = "NONE",
        altsyncram_component.address_reg_b = "CLOCK0",
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_input_b = "BYPASS",
        altsyncram_component.clock_enable_output_b = "BYPASS",
        altsyncram_component.intended_device_family = "Cyclone IV E",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 1024,
        altsyncram_component.numwords_b = 1024,
        altsyncram_component.operation_mode = "DUAL_PORT",
        altsyncram_component.outdata_aclr_b = "NONE",
        altsyncram_component.outdata_reg_b = "CLOCK0",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.read_during_write_mode_mixed_ports =
"DONT_CARE",
        altsyncram_component.widthad_a = 10,
        altsyncram_component.widthad_b = 10,
        altsyncram_component.width_a = 16,
        altsyncram_component.width_b = 16,
        altsyncram_component.width_byteena_a = 1;
```

```
endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLRdata NUMERIC "0"
// Retrieval info: PRIVATE: CLRq NUMERIC "0"
// Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
// Retrieval info: PRIVATE: CLRrren NUMERIC "0"
// Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
// Retrieval info: PRIVATE: CLRwren NUMERIC "0"
// Retrieval info: PRIVATE: Clock NUMERIC "0"
// Retrieval info: PRIVATE: Clock_A NUMERIC "0"
// Retrieval info: PRIVATE: Clock_B NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MEMSIZE NUMERIC "16384"
// Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING ""
// Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
// Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "1"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
// Retrieval info: PRIVATE: REGdata NUMERIC "1"
// Retrieval info: PRIVATE: REGq NUMERIC "1"
// Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
// Retrieval info: PRIVATE: REGrren NUMERIC "1"
// Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
// Retrieval info: PRIVATE: REGwren NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
// Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
// Retrieval info: PRIVATE: VarWidth NUMERIC "0"
```

```
// Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "16"
// Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "16"
// Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "16"
// Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "16"
// Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: enable NUMERIC "0"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_B STRING "NONE"
// Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "1024"
// Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "1024"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "DUAL_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_B STRING "CLOCK0"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING
"DONT_CARE"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "10"
// Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "10"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "16"
// Retrieval info: CONSTANT: WIDTH_B NUMERIC "16"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 16 0 INPUT NODEFVAL "data[15..0]"
// Retrieval info: USED_PORT: q 0 0 16 0 OUTPUT NODEFVAL "q[15..0]"
// Retrieval info: USED_PORT: rdaddress 0 0 10 0 INPUT NODEFVAL
"rdaddress[9..0]"
// Retrieval info: USED_PORT: wraddress 0 0 10 0 INPUT NODEFVAL
"wraddress[9..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
// Retrieval info: CONNECT: @address_a 0 0 10 0 wraddress 0 0 10 0
// Retrieval info: CONNECT: @address_b 0 0 10 0 rdaddress 0 0 10 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 16 0 data 0 0 16 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 16 0 @q_b 0 0 16 0
// Retrieval info: GEN_FILE: TYPE_NORMAL DPRAM.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL DPRAM.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL DPRAM.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL DPRAM.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL DPRAM_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL DPRAM_bb.v FALSE
// Retrieval info: LIB_FILE: altera_mf
```