```
/* $Author: Ramkumar Ravi */
/* $LastChangedDate: 2012-05-04 */
/* NOTE: Some sections are incomplete. Please look into the logic and complete them before
using the design */
/* Direct Mapped Cache */

module mem_system(/*AUTOARG*/
   // Outputs
   DataOut, Done, Stall, CacheHit, err,
   // Inputs
   Addr, DataIn, Rd, Wr, createdump, clk, rst
   );

   input [15:0] Addr;
   input [15:0] DataIn;
   input        Rd;
   input        Wr;
   input        createdump;
   input        clk;
   input        rst;

   output [15:0] DataOut;
   output Done;
   output Stall;
   output CacheHit;
   output err;

   /* data_mem = 1, inst_mem = 0 *
    * needed for cache parameter */
   parameter mem_type = 0;

   // your code here

   // Intermediate wires or buses
   wire [15:0] mem_dout, cache_dout;
   wire [4:0] tag_out;
   wire cache_err, mem_stall, mem_err;
   wire [3:0] busy;
   wire [2:0] coffset;

   // Control Signals
   reg tag_from_cache, data_from_mem;
   wire [15:0] addr_to_mem, data_to_cache;
   reg en, comp , write, V_in, mem_wr, mem_rd, cont_err, done, stall;
   wire hit, validsig, dirty;
   reg incr_count, clear_count, cword_from_count;

   // State variables
   wire qidle, qcomprd, qmemrd, qwbmem, qinstall_cache, qdone;
   wire qcompwr, qerr, qwaitstate, qwrmissdone, qprewbmem;
   wire didle, dcomprd, dmemrd, dwbmem, dinstall_cache, ddone;
   wire dcompwr, derr, dwaitstate, dwrmissdone, dprewbmem;
   wire [10:0] state;
   assign state = {qprewbmem, qwrmissdone, qwaitstate, qerr, qcompwr, qdone, qinstall_cache,
   qwbmem, qmemrd, qcomprd, qidle};
```

```
// Initialize state machine to the idle state
wire idle_ns;
assign idle_ns = rst ? 1'b1 : didle;

// Counter status for reading and writing an entire block
wire count11;
wire [1:0] count;
assign count11 = count[1] & count[0];
// Instantiate the counter here
counter cntr (.count(count), .incr(incr_count), .clear(clear_count), .clk(clk), .rst(rst));

// Mux to cache data_in and mem addr
wire [1:0] wib;

// Word to writeback or read from mem depending on count
assign wib = (count[1]) ? (count[0] ? 2'b11: 2'b10) : (count[0] ? 2'b01 : 2'b00);

// Logic to determine which address is supplied to mem
assign addr_to_mem = tag_from_cache ? {tag_out[4:0], Addr[10:3], wib, Addr[0]} : {Addr[15:3],
 wib, Addr[0]};
assign data_to_cache[15:0] =  data_from_mem ? mem_dout[15:0] : DataIn[15:0];

// What to use as word input to cache
assign coffset = cword_from_count ? {wib,1'b0} : Addr[2:0];

// Keep track of cache hit signal
wire dch, qch;
dff ch (.q(qch), .d(dch), .clk(clk), .rst(rst));
assign dch = (state[1]|state[6]) ? (hit & validsig) : qch;

//Assign mem system outputs
assign DataOut = cache_dout;
assign Done = done;
assign Stall = stall | mem_stall; //stalls entire mem system if not done
assign CacheHit = qch;
assign err = mem_err | cache_err | cont_err;


//*************************************************************************************
**********************
//****************************************Next-state logic (Section Partially
Complete)*************************

     //state[10] = PREWBMEM
     //state[9] = WRMISSDONE
     //state[8] = WAITSTATE
     //state[7] = ERR
     //state[6] = COMPWR
     //state[5] = DONE
     //state[4] = ISNTALL_CACHE
     //state[3] = WBMEM
     //state[2] = MEMRD
     //state[1] = COMPRD
     //state[0] = IDLE

// (Current state is DONE) OR (Rd = 0 and Wr = 0 AND current state is IDLE) OR (Current
```

```
   state is WRMISSDONE)
   assign didle = state[5] | (~Rd & ~Wr & state[0]) | state[9];

   assign dcomprd = //Rd = 1 AND Wr = 0 AND currently in IDLE state

   assign dcompwr = //Rd = 0 AND Wr = 1 AND currently in IDLE state

   assign dmemrd = ((~validsig | (~hit & validsig & ~dirty )) & (state[1]|state[6])) | (state[2]
    & mem_stall) | (state[4] & ~count11) | (state[3] & ~mem_stall & count11);

   assign dwaitstate = // mem_stall=0 AND currently in MEMRD state

   assign dwbmem = state[10] |
                 state[3] & (mem_stall | ~count11);

   assign dinstall_cache = // Current state is WAITSTATE

   assign ddone = // [hit AND valid AND current state is COMPRD] OR [hit AND valid AND current
   state is COMPWR] OR [Rd=1 AND Wr=0 AND current state is INSTALL_CACHE AND count11]

   assign derr = // Rd=1 AND Wr=1

   assign dwrmissdone = // Current state is INSTALL_CACHE AND Wr=1 AND Rd=0 AND count11

   assign dprewbmem = (~hit & validsig & dirty & (state[1] | state[6]));


  //*************************************************************************************
  ********************
  //*********************************************MemSystem
  Control**************************************************

always@(*) begin
case(state)

//IDLE
11'b000_0000_0001: begin en=1'b1; comp= 1'b0 ;  write= 1'b0; V_in= 1'b0; mem_wr=1'b0 ; mem_rd=
1'b0  ;
                    cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'bx; data_from_mem= 1'b0; stall
                     = 1'b0;
                    incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
                  end
//COMPRD
11'b000_0000_0010: begin en=1'b1; comp= 1'b1;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b0;   mem_rd=
  1'b0;
                    cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'b0;data_from_mem= 1'b0; stall
                    = 1'b1;
                     incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
                  end

//MEMRD
11'b000_0000_0100: begin en=1'b1; comp= 1'b0;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b0;  mem_rd=
1'b1;
                   cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'b0; data_from_mem= 1'b0; stall
                   = 1'b1;
                    incr_count = 1'b0; clear_count = 1'b0;cword_from_count =1'b0;
```

```
                    end
//WBMEM
11'b000_0000_1000: begin en=1'b1; comp= 1'b0;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b1;  mem_rd=
1'b0;
                     cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'b1; data_from_mem= 1'b0; stall
                     = 1'b1;
                      incr_count = 1'b1; clear_count = 1'b0; cword_from_count =1'b1;
                   end
//INSTALL_CACHE
11'b000_0001_0000: begin en=1'b1; comp= 1'b0;  write= 1'b1; V_in= 1'b1; mem_wr= 1'b0;   mem_rd=
  1'b0;
                    cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'b0; data_from_mem= 1'b1; stall =
                     1'b1;
                     incr_count = 1'b1; clear_count = 1'b0; cword_from_count =1'b1;
                    end
//DONE
11'b000_0010_0000: begin en=1'b1; comp= 1'b0;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b0;   mem_rd=
  1'b0;
                    cont_err= 1'b0; done= 1'b1; tag_from_cache= 1'b0; data_from_mem= 1'b0; stall =
                     1'b1;
                     incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
                    end
//COMPWR
11'b000_0100_0000: begin en=1'b1; comp= 1'b1;  write= 1'b1; V_in= 1'b0; mem_wr= 1'b0;  mem_rd=
1'b0;
                     cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'b0; data_from_mem= 1'b0; stall
                      = 1'b1;
                      incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
                    end

//ERR
11'b000_1000_0000: begin en=1'b1; comp= 1'b1;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b0;  mem_rd=
1'b0;
                    cont_err= 1'b1; done= 1'b0; tag_from_cache= 1'b0;data_from_mem= 1'b1; stall =
                    1'b1;
                     incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
                    end

 //WATISTATE
11'b001_0000_0000: begin en=1'b1; comp= 1'b0;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b0;  mem_rd=
1'b0;
                     cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'b0; data_from_mem= 1'b0; stall
                     = 1'b1;
                     incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
                    end

//WRMISSDONE
11'b010_0000_0000: begin en=1'b1; comp= 1'b1;  write= 1'b1; V_in= 1'b1; mem_wr= 1'b0;   mem_rd=
  1'b0;
                    cont_err= 1'b0; done= 1'b1; tag_from_cache= 1'b0; data_from_mem= 1'b0; stall =
                     1'b1;
                     incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
                    end
//PREWBMEM
11'b100_0000_0000: begin en=1'b1; comp= 1'b0;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b1;  mem_rd=
1'b0;
```

```
                cont_err= 1'b0; done= 1'b0; tag_from_cache= 1'b1; data_from_mem= 1'b0; stall
                = 1'b1;
                 incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b1;
              end

default :      begin   en=1'b1; comp= 1'b1;  write= 1'b0; V_in= 1'b0; mem_wr= 1'b0;   mem_rd=
1'b0;
              cont_err= 1'b1; done= 1'b0; tag_from_cache= 1'b0; data_from_mem= 1'b0; stall =
              1'b0;
               incr_count = 1'b0; clear_count = 1'b0; cword_from_count =1'b0;
            end
endcase
end

//*************************************************************************************************
******************
//***********************************Instantiate memory and cache
modules************************************

     cache #(mem_type) c0 (
              .enable(en),
              .clk(clk),
              .rst(rst),
              .createdump(createdump),
              .tag_in(Addr[15:11]),
              .index(Addr[10:3]),
              .offset(coffset),
              .data_in (data_to_cache),
              .comp(comp),
              .write(write),
              .valid_in(V_in),
              .tag_out(tag_out),
              .data_out(cache_dout),
              .hit(hit),
              .dirty(dirty),
              .valid(validsig),
              .err(cache_err)
              );

          // Instantiate four_bank_mem here with parameters
          // clk, rst, createdump, .addr(addr_to_mem), .data_in(cache_dout), .wr(mem_wr),
          .rd(mem_rd), .data_out(mem_dout), .stall(mem_stall), busy, .err(mem_err)


   // Instantiate dffs for states --> This section is incomplete
   dff idle(.q(qidle), .d(idle_ns), .clk(clk), .rst(1'b0));
   dff comprd(.q(qcomprd), .d(dcomprd), .clk(clk), .rst(rst));
   // DFF with d-dmemrd and q-qmemrd
   // DFF with d-dwbmem and q-qwbmem
   // DFF with d-dinstall_cache and q-qinstall_cache
   // DFF with d-ddone and q-qdone
   // Similarly DFFs for dcompwr, derr, dwaitstate, dwrmissdone and dprewbmem

endmodule // mem_system
```

```
module counter (count, incr, clear, clk, rst);

input clk, rst, incr, clear;
output [1:0] count;

wire c1,c2;
wire qb0, qb1, db0, db1;

dff b1 (.q(qb1), .d(db1), .clk(clk), .rst(rst));
// Another DFF with D - db0 and Q - qb0

wire b0_ns, b1_ns;
assign b0_ns = incr & ~clear ? ~qb0: qb0;
assign b1_ns = incr & ~clear ? (~qb1&qb0)|(qb1&~qb0): qb1;

assign db0 = clear ? 1'b0 : b0_ns;
assign db1 = clear ? 1'b0 : b1_ns;
assign count = {qb1,qb0};

endmodule
```