file name: define.v

```
/*******************************************************************************************************/
/* MieruPC2010 Version 1.4.0                                                       ArchLab. TOKYO TECH */
/*******************************************************************************************************/

/* Clock Interval Definition                                                                           */
/*******************************************************************************************************/
‘define DCM_CLKFX_MULTIPLY  7      // you can modify this value, (40MHz / 8) * 7 = 35MHz
‘define DCM_CLKFX_DIVIDE    8      // CLKFX_DIVIDE must be 1~32, CLKFX_MULTIPLY must be 2~32

/*******************************************************************************************************/
‘define SYS_CLOCK 40                                                           // 27MHz system clock
‘define TIMER_CNT_WAIT (‘SYS_CLOCK/‘DCM_CLKFX_DIVIDE*‘DCM_CLKFX_MULTIPLY*1000) // for 1kHz timer
‘define CP0_CNT_WAIT   (‘SYS_CLOCK/‘DCM_CLKFX_DIVIDE*‘DCM_CLKFX_MULTIPLY*10)   // for 100kHz CP0 timer
‘define LCD_WAIT       (‘SYS_CLOCK/‘DCM_CLKFX_DIVIDE*‘DCM_CLKFX_MULTIPLY)      // for LCD 1Mbps

/* Memory Controller / Initializer Constants                                                           */
/*******************************************************************************************************/
‘define ADDR 23:0                  // Address Range 24bit
‘define JADDR 21:0                 // Address Range for J format (ADDR − 2bit, 26bit max.)
‘define HALT_ADDR 24’h0000         // Branch Address to Halt

‘define MMC_START_BLOCK   81       // MMC Image Start Physical Address: 0x0000a200
‘define MMC_LAST_BLOCK  1104       // MMC Image End   Physical Address: 0x00089fff

/* Instruction Processing Stage definition                                                             */
/*******************************************************************************************************/
‘define CPU_START 5’h00
‘define CPU_IF0   5’h01
‘define CPU_IF1   5’h02
‘define CPU_IF2   5’h03
‘define CPU_IF3   5’h04
‘define CPU_IF4   5’h05
‘define CPU_ID    5’h06
‘define CPU_RF    5’h07
‘define CPU_EX    5’h08
‘define CPU_MA0   5’h09
‘define CPU_MA1   5’h0a
‘define CPU_MA2   5’h0b
‘define CPU_MA3   5’h0c
‘define CPU_MA4   5’h0d
‘define CPU_WB    5’h0e
‘define CPU_WCNT  5’h0f
‘define CPU_HALT  5’h10
‘define CPU_ERROR 5’h11

/* Instruction Attribute Definition                                                                    */
/*******************************************************************************************************/
‘define READ_NONE            19’h00000
‘define READ_RS              19’h00000
‘define READ_RT              19’h00000
‘define READ_RD              19’h00000
‘define READ_HI              19’h00000
‘define READ_LO              19’h00000
‘define READ_HILO            19’h00000
‘define WRITE_NONE           19’h00000
‘define WRITE_RS             19’h00000
‘define WRITE_RT             19’h00001
‘define WRITE_RD             19’h00002
‘define WRITE_HI             19’h00004
‘define WRITE_LO             19’h00008
‘define WRITE_HILO           19’h0000c
‘define WRITE_RD_COND        19’h00010
‘define WRITE_RRA            19’h00020
‘define LOAD_1B              19’h00040
‘define LOAD_2B              19’h00080
‘define LOAD_4B_ALIGN        19’h00100
‘define LOAD_4B_UNALIGN      19’h00200
‘define LOAD_4B              19’h00300
‘define LOAD_ANY             19’h003c0
‘define STORE_1B             19’h00400
‘define STORE_2B             19’h00800
‘define STORE_4B_ALIGN       19’h01000
‘define STORE_4B_UNALIGN     19’h02000
‘define STORE_4B             19’h03000
‘define STORE_ANY            19’h03c00
‘define LDST                 19’h03fc0
‘define LOADSTORE_4B_UNALIGN 19’h02200
‘define BRANCH               19’h04000
‘define BRANCH_LIKELY        19’h08000
‘define READ_CP0             19’h00000
‘define WRITE_CP0            19’h10000
‘define SYSTEM_CALL          19’h20000
‘define BRANCH_ERET          19’h40000

/*******************************************************************************************************/
```

file name: MieruEMB.v

```
/*******************************************************************************************************/
/* MieruEMB System Version 1.0 since 2011−09                                       ArchLab. TOKYO TECH */
/*  See the README file on the parent directory for license information.                               */
/*******************************************************************************************************/
/*******************************************************************************************************/
/* MieruPC2010 Version 1.4.0, 2011−10−05: Top Level Module                         ArchLab. TOKYO TECH */
/*******************************************************************************************************/
‘include "../rtl/define.v"
‘default_nettype none

/*******************************************************************************************************/
module MieruEMB(CLK, SW, LCD, ULED, GPIO, GPI, MMC_CS_X, MMC_DIN, MMC_SCLK, MMC_DOUT,
               SRAM_A, SRAM_D, SRAM_OE_X, SRAM_WE_X,
               LCD_CS0, LCD_CD, LCD_WR, LCD_RSTB, LCD_D);
    input             CLK, GPI;     // input clock & general purpose input
    input [2:0]       SW;           // input switch
    input             MMC_DOUT;     // MMC Data Output
    output            LCD;          // LCD Serial Output
    output            MMC_CS_X;     // MMC Chip Select
    output            MMC_DIN;      // MMC Data Input
    output            MMC_SCLK;     // MMC Clock
    output [2:0]      ULED;         // User LED
    output reg [18:0] SRAM_A;       // SRAM Address
    output reg        SRAM_OE_X;    // SRAM Output Enable
    output            SRAM_WE_X;    // SRAM Write Enable
    inout [1:0]       GPIO;         // General Purpose I/O
    inout [7:0]       SRAM_D;       // SRAM Data Bus

    output        LCD_CS0, LCD_CD, LCD_WR, LCD_RSTB;
    output [7:0]  LCD_D;

    wire              FCLK, RST_OX;                  // system clock and reset signal
    wire              WE, CORE_WE, MEM_WE, PLOAD_WE; // Write Enable signals
    wire              PLOAD_DONE;                    // Program load is done
    wire [‘ADDR]      ADDR, PLOAD_ADDR, CORE_ADDR;   // memory address
    wire [7:0]        DATA_IN, DATA_OUT, CORE_DATA, PLOAD_DATA; // data

    CLKGEN clkgen(.CLK_I(CLK), .RST_X_I(SW[0] | SW[1] | SW[2]), .CLK_O(FCLK), .RST_X_O(RST_OX));

    MIPSCORE core(.CLK(FCLK), .RST_X(RST_OX & PLOAD_DONE),
                  .ADDR(CORE_ADDR), .DATA_IN(DATA_IN), .DATA_OUT(CORE_DATA), .WE(CORE_WE));

    PROGLOADER pload(.CLK(FCLK), .RST_X(RST_OX),
                     .DATA_IN(DATA_IN),
                     .ADDR(PLOAD_ADDR), .DATA_OUT(PLOAD_DATA), .WE(PLOAD_WE), .DONE(PLOAD_DONE));

    IOCON iocon(.CLK(FCLK), .RST_X(RST_OX),
                .LCD(LCD), .GPIO(GPIO), .SW(SW), .GPI(GPI),
                .ADDR(ADDR), .DATA_IN(DATA_IN), .DATA_OUT(DATA_OUT),
                .WE(WE), .MEM_WE(MEM_WE), .MEM_Q(SRAM_D),
                .MMC_CSX(MMC_CS_X), .MMC_DIN(MMC_DIN), .MMC_DOUT(MMC_DOUT), .MMC_SCLK(MMC_SCLK));

    reg lcd_reg;
    always @(posedge FCLK) lcd_reg <= ~LCD;

    assign ULED[0]  = (!PLOAD_DONE) ? 1          : lcd_reg;
    assign ADDR     = (!PLOAD_DONE) ? PLOAD_ADDR : CORE_ADDR;
    assign DATA_OUT = (!PLOAD_DONE) ? PLOAD_DATA : CORE_DATA;
    assign WE       = (!PLOAD_DONE) ? PLOAD_WE   : CORE_WE;

    reg [7:0] D_KEEP;
    always @(posedge FCLK or negedge RST_OX) begin
         if (!RST_OX) begin
             SRAM_A    <= 0;
             D_KEEP    <= 0;
             SRAM_OE_X <= 0;
         end else begin
             SRAM_A    <= ADDR[18:0];
             D_KEEP    <= DATA_OUT;
             SRAM_OE_X <= MEM_WE;
         end
    end

    assign SRAM_D = (SRAM_OE_X) ? D_KEEP : 8’hzz;
    assign SRAM_WE_X = ~FCLK | ~SRAM_OE_X;

    reg [24:0] cnt_t;
    always @(posedge FCLK) cnt_t <= cnt_t + 1;
    assign ULED[2] = cnt_t[24];
    assign ULED[1] = (~SW[0]) | (~SW[1]) | (~SW[2]);

    minilcd_con lcdcon (FCLK, (RST_OX & PLOAD_DONE), CORE_ADDR[13:0], CORE_DATA,
                        (CORE_WE & CORE_ADDR[23:16]==8’h90),
                        LCD_CS0, LCD_CD, LCD_RSTB, LCD_D, LCD_WR);
endmodule

/*******************************************************************************************************/
```

```
file name: minilcd.v
/******************************************************************************************************/
/* MieruEMB System Version 1.0 since 2011−09                                      ArchLab. TOKYO TECH */
/******************************************************************************************************/
‘default_nettype none

/******************************************************************************************************/
/***** MiniLCD controller,  128x128 pixel LCD                                                    *****/
/******************************************************************************************************/
module minilcd_con(CLK, RST_X, VRAM_ADDR, VRAM_DATA, VRAM_WE,
                   LCD_CS0, LCD_CD, LCD_RSTB, LCD_D, LCD_WR);
    input            CLK, RST_X;
    input [13:0]     VRAM_ADDR;
    input [7:0]      VRAM_DATA;
    input            VRAM_WE;
    output           LCD_WR;
    output reg       LCD_CS0, LCD_CD, LCD_RSTB;
    output reg [7:0] LCD_D;

    reg          init;
    reg [14:0]   cmdcnt;
    reg [2:0]    writecnt;
    reg [23:0]   waitcnt;
    wire [15:0]  dout;
    wire [7:0]   D;

    assign LCD_WR = ~writecnt[2];

    minilcd_initmem mlcdmem(CLK, cmdcnt[5:0], dout);

    minilcd_vram vram(.CLK(CLK), .DIN(VRAM_DATA[3:0]), .WADDR(VRAM_ADDR), .WE(VRAM_WE),
                      .DOUT(D), .RADDR(cmdcnt[14:1]));

    always @(posedge CLK or negedge RST_X) begin
      if (!RST_X) begin
        LCD_CS0  <= 0;
        LCD_CD   <= 0;
        LCD_RSTB <= 0;
        LCD_D    <= 0;
        init     <= 1;
        cmdcnt   <= 0;
        writecnt <= 0;
        waitcnt  <= 0;
      end
      else if (writecnt != 0) writecnt <= writecnt − 1;
      else if (waitcnt  != 0) waitcnt  <= waitcnt  − 1;
      else if (init == 1) begin
        waitcnt  <= {dout[15:11], 18’h00000};
        LCD_RSTB <= ~dout[10];
        LCD_CS0  <= dout[9];
        LCD_CD   <= dout[8];
        LCD_D    <= dout[7:0];
        init     <= (cmdcnt != ’h3a);
        cmdcnt   <= (cmdcnt == ’h3a) ? 0 : cmdcnt + 1;
        writecnt <= 7;
      end
      else begin // after initialized
        LCD_RSTB <= 1;
        LCD_CS0  <= 0;
        LCD_CD   <= 1;
        LCD_D    <= (cmdcnt[0]) ? // convert 3 bit to 16 bit color
                     {D[1], D[0], D[0], D[0], D[0], D[0], D[0], D[0]} :
                     {D[2], D[2], D[2], D[2], D[1], D[1], D[1], D[1]} ;
        cmdcnt   <= (cmdcnt == ’h7fff) ? 0 : cmdcnt + 1;
        writecnt <= 7;
      end
    end
endmodule

/******************************************************************************************************/
/* MiniLCD Video Memory 8KB (128x128=16K pixel, 4bit per pixel = 8KB)                                 */
/******************************************************************************************************/
module minilcd_vram(CLK, DIN, DOUT, RADDR, WADDR, WE);
    input        CLK, WE;
    input [3:0]  DIN;
    input [13:0] RADDR, WADDR;
    output reg [3:0] DOUT;

    reg [3:0] mem [16383:0]; /* 4bit x 16K pixel = 8KB VRAM */

    always @(posedge CLK) begin
        if (WE) mem[WADDR] <= DIN;
        DOUT <= mem[RADDR];
    end
endmodule

/******************************************************************************************************/
/* MiniLCD initialize memory, [WAIT(5)] [RST] [CSX] [DC] [DATA(8)]                                    */
/******************************************************************************************************/
module minilcd_initmem(CLK, ADDR, DATA);
```

```
    input             CLK;
    input       [5:0] ADDR;
    output reg [15:0] DATA;

    always @(posedge CLK) begin
        case (ADDR)
        ’h00: DATA = 16’h1200;
        ’h01: DATA = 16’h1600; // Hardware Reset
        ’h02: DATA = 16’h6200;
        ’h03: DATA = 16’h2801; // Software Reset
        ’h04: DATA = 16’ha011; // Sleep Out
        ’h05: DATA = 16’h00ff; // VCOM 4 Level Control
        ’h06: DATA = 16’h0140; //   − TC2 4clk, TC3 3clk delay
        ’h07: DATA = 16’h0103;
        ’h08: DATA = 16’h011a;
        ’h09: DATA = 16’h00b1; // Frame Rate Control (Normal)
        ’h0a: DATA = 16’h0104; //   − RTN         24
        ’h0b: DATA = 16’h0125; //   − Front Porch 37
        ’h0c: DATA = 16’h0118; //   − Back Porch  24
        ’h0d: DATA = 16’h00b4; // Display Inversion Control
        ’h0e: DATA = 16’h0103; //   − Line Inversion
        ’h0f: DATA = 16’h00b6; // Display Function set 5
        ’h10: DATA = 16’h0105; //   (Output waveform configure)
        ’h11: DATA = 16’h0102;
        ’h12: DATA = 16’h00c1; // Power Control 2
        ’h13: DATA = 16’h0107; //   − VGHH 14.7V, VGLL −12.25V
        ’h14: DATA = 16’h00fc; // Power Control 6
        ’h15: DATA = 16’h0111;
        ’h16: DATA = 16’h0117;
        ’h17: DATA = 16’h00c5; // VCOM Control 1
        ’h18: DATA = 16’h013c; //   − VCOMH +4V
        ’h19: DATA = 16’h014f; //   − VCOML −0.525V
        ’h1a: DATA = 16’h0036; // Memory Data Access Control
        ’h1b: DATA = 16’h01c8; //   − XY mirror, BGR order
        ’h1c: DATA = 16’h003a; // Interface Pixel Format
        ’h1d: DATA = 16’h0105; //   − 16−bit per pixel
        ’h1e: DATA = 16’h00e1; // Gamma Correction Negative
        ’h1f: DATA = 16’h0101; //   − VRH
        ’h20: DATA = 16’h011c; //   − VRL
        ’h21: DATA = 16’h0105; //   − PK0(V3)
        ’h22: DATA = 16’h0111; //   − PK1(V6)
        ’h23: DATA = 16’h0117; //   − PK2(V11)
        ’h24: DATA = 16’h011a; //   − PK3(V19)
        ’h25: DATA = 16’h011C; //   − PK4(V27)
        ’h26: DATA = 16’h0121; //   − PK5(V36)
        ’h27: DATA = 16’h011F; //   − PK6(V44)
        ’h28: DATA = 16’h011D; //   − PK7(V52)
        ’h29: DATA = 16’h0127; //   − PK8(V57)
        ’h2a: DATA = 16’h012F; //   − PK9(V60)
        ’h2b: DATA = 16’h0105; //   − SELV0
        ’h2c: DATA = 16’h0103; //   − SELV1
        ’h2d: DATA = 16’h0100; //   − SELV62
        ’h2e: DATA = 16’h013F; //   − SELV63
        ’h2f: DATA = 16’h002a; // Column Address Set
        ’h30: DATA = 16’h0100; //   − start   2
        ’h31: DATA = 16’h0102;
        ’h32: DATA = 16’h0100; //   − end   129
        ’h33: DATA = 16’h0181;
        ’h34: DATA = 16’h002b; // Row Address Set
        ’h35: DATA = 16’h0100; //   − start   3
        ’h36: DATA = 16’h0103;
        ’h37: DATA = 16’h0100; //   − end   130
        ’h38: DATA = 16’h0182;
        ’h39: DATA = 16’h5029; // Display On
        ’h3a: DATA = 16’h002c; // Memory Write
        ’h3b: DATA = 16’h0000;
        ’h3c: DATA = 16’h0000;
        ’h3d: DATA = 16’h0000;
        ’h3e: DATA = 16’h0000;
        ’h3f: DATA = 16’h0000;
        endcase
    end
endmodule
/******************************************************************************************************/
```

```
file name: MipsCore.v
/******************************************************************************************************/
/* MieruPC2010: MipsCore                       Version 1.4.0-test05 (2011-10-05)  ArchLab. TOKYO TECH */
/******************************************************************************************************/
‘include "../rtl/define.v"
‘include "../rtl/instn.v"
‘default_nettype none

/******************************************************************************************************/
/* 32bit-32cycle mutiplier (signed or unsigned)                                                      */
/******************************************************************************************************/
module MULUNIT(CLK, RST_X, INIT, SIGNED, A, B, RSLT, BUSY);
    input         CLK;
    input         RST_X;
    input         INIT;
    input         SIGNED;
    input  [31:0] A, B;
    output [63:0] RSLT;
    output        BUSY;

    wire [31:0]   uint_a, uint_b;
    wire [63:0]   uint_rslt;
    reg           sign;

    MULUNITCORE mulcore(.CLK(CLK), .RST_X(RST_X), .INIT(INIT),
                        .A(uint_a), .B(uint_b), .RSLT(uint_rslt), .BUSY(BUSY));

    assign uint_a = (SIGNED & A[31])? ~A + 1 : A;
    assign uint_b = (SIGNED & B[31])? ~B + 1 : B;
    assign RSLT   = (SIGNED & sign)? ~uint_rslt + 1 : uint_rslt;

    always @( posedge CLK or negedge RST_X )
      if( !RST_X ) sign <= 0;
      else         sign <= (INIT)? A[31]^B[31] : sign;

endmodule

/******************************************************************************************************/
module MULUNITCORE(CLK, RST_X, INIT, A, B, RSLT, BUSY);
    input         CLK;
    input         RST_X;
    input         INIT;
    input  [31:0] A, B;
    output [63:0] RSLT;
    output        BUSY;

    reg [31:0]    multiplicand;
    reg [5:0]     count;
    wire [32:0]   sum;
    reg [63:0]    RSLT;

    assign BUSY   = (count < 32);
    assign sum    = RSLT[63:32] + multiplicand;

    always @( posedge CLK or negedge RST_X ) begin
        if( !RST_X ) begin
            multiplicand   <= 0;
            RSLT           <= 0;
            count          <= 0;
        end else if( INIT )begin
            multiplicand   <= A;
            RSLT           <= {32’h0, B};
            count          <= 0;
        end else begin
            multiplicand   <= multiplicand;
            RSLT           <= (RSLT[0]) ? {sum, RSLT[31:1]} : {1’h0, RSLT[63:1]};
            count          <= count + 1;
        end
    end
endmodule

/******************************************************************************************************/
/* 32bit-32cycle divider (signed or unsigned)                                                        */
/******************************************************************************************************/
module DIVUNIT(CLK, RST_X, INIT, SIGNED, A, B, RSLT, BUSY);
    input         CLK, RST_X, INIT, SIGNED;
    input  [31:0] A, B;
    output [63:0] RSLT;
    output        BUSY;

    wire [31:0]   uint_a, uint_b;
    wire [63:0]   uint_rslt;
    reg           sign_a, sign_b;

    DIVUNITCORE divcore(.CLK(CLK), .RST_X(RST_X), .INIT(INIT), .A(uint_a), .B(uint_b),
                        .RSLT(uint_rslt), .BUSY(BUSY));

    assign uint_a = (SIGNED & A[31]) ? ~A + 1 : A;
    assign uint_b = (SIGNED & B[31]) ? ~B + 1 : B;
    assign RSLT[63:32] =
```

```
        (~SIGNED)?                     uint_rslt[63:32]           :
        ({sign_a, sign_b} == 2’b00) ?  uint_rslt[63:32]           :
        ({sign_a, sign_b} == 2’b01) ?  uint_rslt[63:32]           :
        ({sign_a, sign_b} == 2’b10) ? ~uint_rslt[63:32] + 1       : ~uint_rslt[63:32] + 1;
    assign RSLT[31: 0] =
        (~SIGNED)?                     uint_rslt[31: 0]     :
        ({sign_a, sign_b} == 2’b00) ?  uint_rslt[31: 0]     :
        ({sign_a, sign_b} == 2’b01) ? ~uint_rslt[31: 0] + 1 :
        ({sign_a, sign_b} == 2’b10) ? ~uint_rslt[31: 0] + 1 : uint_rslt[31: 0];

    always @( posedge CLK or negedge RST_X )
      if(!RST_X) begin
          sign_a  <= 0;
          sign_b  <= 0;
      end else begin
          sign_a  <= (INIT) ? A[31] : sign_a;
          sign_b  <= (INIT) ? B[31] : sign_b;
      end
endmodule

/******************************************************************************************************/
module DIVUNITCORE(CLK, RST_X, INIT, A, B, RSLT, BUSY);
    input         CLK, RST_X, INIT;
    input  [31:0] A, B;
    output [63:0] RSLT;
    output        BUSY;

    reg [63:0]    RSLT;
    reg [31:0]    divisor;
    reg [5:0]     count;
    wire [32:0]   differ;

    assign BUSY   = (count < 32);
    assign differ = RSLT[63:31] - {1’b0, divisor};

    always @( posedge CLK or negedge RST_X ) begin
        if(!RST_X) begin
            divisor   <= 0;
            RSLT      <= 0;
            count     <= 0;
        end else if( INIT ) begin
            divisor   <= B;
            RSLT      <= {32’h0, A};
            count     <= 0;
        end else begin
            divisor   <= divisor;
            RSLT      <= (differ[32]) ? {RSLT[62:0], 1’h0} : {differ[31:0], RSLT[30:0], 1’h1};
            count     <= count + 1;
        end
    end
endmodule

/******************************************************************************************************/
/* 32bitx32 2R/W General Purpose Registers                                                           */
/******************************************************************************************************/
module GPR(CLK, REGNUM0, REGNUM1, DIN0, DIN1, WE0, WE1, DOUT0, DOUT1);
    input         CLK;
    input [4:0]   REGNUM0, REGNUM1;
    input [31:0]  DIN0, DIN1;
    input         WE0, WE1;
    output [31:0] DOUT0, DOUT1;

    reg [31:0]    r[0:31];
    reg [31:0]    DOUT0, DOUT1;

    always @(posedge CLK) DOUT0 <= (REGNUM0==0) ? 0 : r[REGNUM0];
    always @(posedge CLK) DOUT1 <= (REGNUM1==0) ? 0 : r[REGNUM1];
    always @(posedge CLK) if(WE0) r[REGNUM0] <= DIN0;
    always @(posedge CLK) if(WE1) r[REGNUM1] <= DIN1;
endmodule

/******************************************************************************************************/
/* MipsCore: 32bit-MIPS multicycle processor                                                         */
/******************************************************************************************************/
module MIPSCORE(CLK, RST_X, ADDR, DATA_IN, DATA_OUT, WE);
    input              CLK, RST_X;
    input  [7:0]       DATA_IN;
    output [7:0]       DATA_OUT;
    output [‘ADDR]     ADDR;
    output             WE;
    /**************************************************************************************************/
    /***** internal register                                                                     *****/
    reg [4:0]          state;
    reg [‘ADDR]        pc, delay_npc, inst_pc, inst_npc;
    reg [‘ADDR]        inst_eaddr;
    reg                exec_delay;
    reg [31:0]         inst_ir;
    reg [4:0]          inst_rt, inst_rd, inst_dst;
    reg [4:0]          inst_shamt;
    reg [15:0]         inst_imm;
reg [‘JADDR]       inst_addr;
reg [18:0]         inst_attr;
reg [6:0]          inst_op;
reg [31:0]         inst_rrs, inst_rrt, inst_cpr, inst_rslt, inst_rslthi;
reg [31:0]         hi, lo;           // high and low register for multiply and divide inst.
reg                inst_cond;
reg [3:0]          inst_datamask;
reg [31:0]         inst_loaddata;
/**************************************************************************************************/
/***** internal wire                                                                          *****/
reg  [4:0]         IDRS, IDRT, IDRD;
reg  [4:0]         IDSHAMT;
reg  [15:0]        IDIMM;
reg  [‘JADDR]      IDADDR;
reg  [18:0]        IDATTR;
reg  [6:0]         IDOP;
wire [4:0]         RFDST;
reg  [31:0]        EXRSLT;
reg  [31:0]        EXRSLTHI;
reg  [‘ADDR]       EXEADDR;
reg  [‘ADDR]       EXNPC;
reg                EXC;
reg  [3:0]         EXMASK;
reg  [1:0]         EXDATAEXT;
reg                EXBUSY;
wire               EXSIGNED;
wire [31:0]        MALOADDATA, MALOADDATARAW, MALOADMASK;
wire [63:0]        DURSLT, MURSLT;
wire               DUBUSY, MUBUSY;
wire               DIVMULINIT;
wire [4:0]         GPRNUM0, GPRNUM1, CPRNUM;
wire [31:0]        GPRREADDT0,  GPRREADDT1,  CPRREADDT;
wire [31:0]        GPRWRITEDT0, GPRWRITEDT1, CPRWRITEDT;
wire               GPRWE0, GPRWE1, CPRWE;
wire               CPEXCSET, CPEXCCLR, CPEXCACK, CPEXCOCCUR, CPEXCBD;
wire [3:0]         CPEXCCODE;
wire [‘ADDR]       CPEXCEPC, CPEXCNPC;
wire [31:0]        SET32I; // 32bit sign extended of 16bit immediate
wire [‘ADDR]       SETADI; // sign extended address of 16bit immediate
wire [31:0]        RRT_U, RRS_U;
wire signed [31:0] RRT_S;
wire               CPUEXE = 1; // signal to stall the processor!

/**************************************************************************************************/
/* Sub module declaration                                                                         */
/**************************************************************************************************/
DIVUNIT du(.CLK(CLK), .RST_X(RST_X), .INIT(DIVMULINIT), .SIGNED(EXSIGNED),
           .A(GPRREADDT0), .B(GPRREADDT1), .RSLT(DURSLT), .BUSY(DUBUSY));

MULUNIT mu(.CLK(CLK), .RST_X(RST_X), .INIT(DIVMULINIT), .SIGNED(EXSIGNED),
           .A(GPRREADDT0), .B(GPRREADDT1), .RSLT(MURSLT), .BUSY(MUBUSY));

GPR gpr(.CLK(CLK), .REGNUM0(GPRNUM0), .REGNUM1(GPRNUM1), .DIN0(GPRWRITEDT0), .DIN1(GPRWRITEDT1),
        .WE0(GPRWE0), .WE1(GPRWE1), .DOUT0(GPRREADDT0), .DOUT1(GPRREADDT1));

MIPSCP0 cp0(.CLK(CLK), .RST_X(RST_X), .REG_NUM(CPRNUM), .REG_IN(CPRWRITEDT), .REG_WE(CPRWE),
            .REG_OUT(CPRREADDT), .EXC_SET(CPEXCSET), .EXC_CLR(CPEXCCLR), .EXC_ACK(CPEXCACK),
            .EXC_CODE(CPEXCCODE), .EXC_EPC(CPEXCEPC), .EXC_BD(CPEXCBD),
            .EXC_OCCUR(CPEXCOCCUR), .EXC_NPC(CPEXCNPC));

/**************************************************************************************************/
/* Mips::proceedstate()                                                                           */
/**************************************************************************************************/
always @(posedge CLK or negedge RST_X) begin
    if(!RST_X)                                   state <= ‘CPU_START;
    else if(~CPUEXE || state==‘CPU_HALT)         state <= state;
    else if(exec_delay && delay_npc==‘HALT_ADDR) state <= ‘CPU_HALT;
    else if(state==‘CPU_EX &&  EXBUSY)           state <= state;
    else if(state==‘CPU_EX && ~EXBUSY)           state <= (inst_attr & ‘LDST) ? ‘CPU_MA0 : ‘CPU_WB;
    else if(state==‘CPU_WB)                      state <= ‘CPU_IF0;
    else                                         state <= state + 1;
end

/**************************************************************************************************/
/* Mips:: instruction fetch stage and memory access address generation                            */
/**************************************************************************************************/
// inst_eaddr is incremented on MA0, MA1, and MA2 stage
assign ADDR = (state == ‘CPU_IF0)                      ? (pc & ~’h3)        :
              (state == ‘CPU_IF1)                      ? (pc & ~’h3) | 2’h1 :
              (state == ‘CPU_IF2)                      ? (pc & ~’h3) | 2’h2 :
              (state == ‘CPU_IF3)                      ? (pc & ~’h3) | 2’h3 :
              (state >= ‘CPU_MA0 && state <= ‘CPU_MA3) ? inst_eaddr : 0;

always @( posedge CLK or negedge RST_X ) begin
    if(!RST_X) inst_pc <= 0;
    else if(CPUEXE) begin
        if     (state==‘CPU_IF0)  inst_pc         <= pc;
        if     (state==‘CPU_IF1)  inst_ir[ 7: 0]  <= DATA_IN;
        else if(state==‘CPU_IF2)  inst_ir[15: 8]  <= DATA_IN;
        else if(state==‘CPU_IF3)  inst_ir[23:16]  <= DATA_IN;
```

```
        else if(state==‘CPU_IF4)  inst_ir[31:24]  <= DATA_IN;
    end
end

/**************************************************************************************************/
/* Mips:: instruction decode stage                                                                */
/**************************************************************************************************/
always@ (inst_ir) begin
    IDRS    = inst_ir[25:21];
    IDRT    = inst_ir[20:16];
    IDRD    = inst_ir[15:11];
    IDSHAMT = inst_ir[10:6];
    IDIMM   = inst_ir[15:0];
    IDADDR  = inst_ir[25:0];
    IDOP    = ‘NOP______;
    IDATTR  = ‘WRITE_NONE;

    case (inst_ir[31:26]) // OP
      6’d00: case (inst_ir[5:0]) // FUNCT
               6’d0: begin
                   if      ((IDRT | IDRD |        IDSHAMT) == 0) begin IDOP=‘NOP______;        end
                   else if ((IDRT | IDRD) == 0 && IDSHAMT  == 1) begin IDOP=‘NOP______;        end
                   else begin IDOP=‘SLL______; IDATTR=‘WRITE_RD;                               end
               end
               6’d2: begin IDOP=‘SRL______; IDATTR=‘WRITE_RD;                                  end
               6’d3: begin IDOP=‘SRA______; IDATTR=‘WRITE_RD;                                  end
               6’d4: begin IDOP=‘SLLV_____; IDATTR=‘WRITE_RD;                                  end
               6’d6: begin IDOP=‘SRLV_____; IDATTR=‘WRITE_RD;                                  end
               6’d7: begin IDOP=‘SRAV_____; IDATTR=‘WRITE_RD;                                  end
               6’d8: begin
                   if      (IDSHAMT==0    ) begin IDOP=‘JR_______; IDATTR=‘BRANCH;             end
                   else if (IDSHAMT==5’d16) begin IDOP=‘JR_HB____; IDATTR=‘BRANCH;             end
               end
               6’d9: begin
                   if      (IDSHAMT==0    ) begin IDOP=‘JALR_____; IDATTR=‘BRANCH | ‘WRITE_RD; end
                   else if (IDSHAMT==5’d16) begin IDOP=‘JALR_HB__; IDATTR=‘BRANCH | ‘WRITE_RD; end
               end
               6’d10: begin IDOP=‘MOVZ_____; IDATTR=‘WRITE_RD_COND;                            end
               6’d11: begin IDOP=‘MOVN_____; IDATTR=‘WRITE_RD_COND;                            end
               6’d12: begin IDOP=‘SYSCALL__; IDATTR=‘SYSTEM_CALL;                              end
               6’d16: begin IDOP=‘MFHI_____; IDATTR=‘WRITE_RD;                                 end
               6’d17: begin IDOP=‘MTHI_____; IDATTR=‘WRITE_HI;                                 end
               6’d18: begin IDOP=‘MFLO_____; IDATTR=‘WRITE_RD;                                 end
               6’d19: begin IDOP=‘MTLO_____; IDATTR=‘WRITE_LO;                                 end
               6’d24: begin IDOP=‘MULT_____; IDATTR=‘WRITE_HILO;                               end
               6’d25: begin IDOP=‘MULTU____; IDATTR=‘WRITE_HILO;                               end
               6’d26: begin IDOP=‘DIV______; IDATTR=‘WRITE_HILO;                               end
               6’d27: begin IDOP=‘DIVU_____; IDATTR=‘WRITE_HILO;                               end
               6’d32: begin IDOP=‘ADD______; IDATTR=‘WRITE_RD;                                 end
               6’d33: begin IDOP=‘ADDU_____; IDATTR=‘WRITE_RD;                                 end
               6’d34: begin IDOP=‘SUB______; IDATTR=‘WRITE_RD;                                 end
               6’d35: begin IDOP=‘SUBU_____; IDATTR=‘WRITE_RD;                                 end
               6’d36: begin IDOP=‘AND______; IDATTR=‘WRITE_RD;                                 end
               6’d37: begin IDOP=‘OR_______; IDATTR=‘WRITE_RD;                                 end
               6’d38: begin IDOP=‘XOR______; IDATTR=‘WRITE_RD;                                 end
               6’d39: begin IDOP=‘NOR______; IDATTR=‘WRITE_RD;                                 end
               6’d42: begin IDOP=‘SLT______; IDATTR=‘WRITE_RD;                                 end
               6’d43: begin IDOP=‘SLTU_____; IDATTR=‘WRITE_RD;                                 end
             endcase
      6’d01: case ( IDRT )
               6’d00: begin IDOP=‘BLTZ_____; IDATTR=‘BRANCH;                                   end
               6’d01: begin IDOP=‘BGEZ_____; IDATTR=‘BRANCH;                                   end
               6’d02: begin IDOP=‘BLTZL____; IDATTR=‘BRANCH_LIKELY;                            end
               6’d03: begin IDOP=‘BGEZL____; IDATTR=‘BRANCH_LIKELY;                            end
               6’d16: begin IDOP=‘BLTZAL___; IDATTR=‘BRANCH        | ‘WRITE_RRA;               end
               6’d17: begin IDOP=‘BGEZAL___; IDATTR=‘BRANCH        | ‘WRITE_RRA;               end
               6’d18: begin IDOP=‘BLTZALL__; IDATTR=‘BRANCH_LIKELY | ‘WRITE_RRA;               end
               6’d19: begin IDOP=‘BGEZALL__; IDATTR=‘BRANCH_LIKELY | ‘WRITE_RRA;               end
            endcase
      6’d02: begin IDOP=‘J________; IDATTR=‘BRANCH;                                       end
      6’d03: begin IDOP=‘JAL______; IDATTR=‘BRANCH | ‘WRITE_RRA;                          end
      6’d04: begin IDOP=‘BEQ______; IDATTR=‘BRANCH;                                       end
      6’d05: begin IDOP=‘BNE______; IDATTR=‘BRANCH;                                       end
      6’d06: begin IDOP=‘BLEZ_____; IDATTR=‘BRANCH;                                       end
      6’d07: begin IDOP=‘BGTZ_____; IDATTR=‘BRANCH;                                       end
      6’d08: begin IDOP=‘ADDI_____; IDATTR=‘WRITE_RT;                                     end
      6’d09: begin IDOP=‘ADDIU____; IDATTR=‘WRITE_RT;                                     end
      6’d10: begin IDOP=‘SLTI_____; IDATTR=‘WRITE_RT;                                     end
      6’d11: begin IDOP=‘SLTIU____; IDATTR=‘WRITE_RT;                                     end
      6’d12: begin IDOP=‘ANDI_____; IDATTR=‘WRITE_RT;                                     end
      6’d13: begin IDOP=‘ORI______; IDATTR=‘WRITE_RT;                                     end
      6’d14: begin IDOP=‘XORI_____; IDATTR=‘WRITE_RT;                                     end
      6’d15: begin IDOP=‘LUI______; IDATTR=‘WRITE_RT;                                     end
      6’d16:
        if      (IDRS == 6’d00) begin IDOP=‘MFC0_____; IDATTR=‘WRITE_RT;                  end
        else if (IDRS == 6’d04) begin IDOP=‘MTC0_____; IDATTR=‘WRITE_CP0;                 end
        else if (IDRS == 6’d16 && inst_ir[5:0] == 6’d24)
             begin IDOP=‘ERET_____; IDATTR=‘BRANCH_ERET;                                  end
      6’d20: begin IDOP=‘BEQL_____; IDATTR=‘BRANCH_LIKELY;                                end
      6’d21: begin IDOP=‘BNEL_____; IDATTR=‘BRANCH_LIKELY;                                end
```

```
          6’d22: begin IDOP=‘BLEZL____; IDATTR=‘BRANCH_LIKELY;                                end
          6’d23: begin IDOP=‘BGTZL____; IDATTR=‘BRANCH_LIKELY;                                end
          6’d28: begin IDOP=‘MUL______; IDATTR=‘WRITE_RD;                                     end
          6’d32: begin IDOP=‘LB_______; IDATTR=‘WRITE_RT | ‘LOAD_1B;                          end
          6’d33: begin IDOP=‘LH_______; IDATTR=‘WRITE_RT | ‘LOAD_2B;                          end
          6’d34: begin IDOP=‘LWL______; IDATTR=‘WRITE_RT | ‘LOAD_4B_UNALIGN;                  end
          6’d35: begin IDOP=‘LW_______; IDATTR=‘WRITE_RT | ‘LOAD_4B_ALIGN;                    end
          6’d36: begin IDOP=‘LBU______; IDATTR=‘WRITE_RT | ‘LOAD_1B;                          end
          6’d37: begin IDOP=‘LHU______; IDATTR=‘WRITE_RT | ‘LOAD_2B;                          end
          6’d38: begin IDOP=‘LWR______; IDATTR=‘WRITE_RT | ‘LOAD_4B_UNALIGN;                  end
          6’d40: begin IDOP=‘SB_______; IDATTR=‘STORE_1B;                                     end
          6’d41: begin IDOP=‘SH_______; IDATTR=‘STORE_2B;                                     end
          6’d42: begin IDOP=‘SWL______; IDATTR=‘STORE_4B_UNALIGN;                             end
          6’d43: begin IDOP=‘SW_______; IDATTR=‘STORE_4B_ALIGN;                               end
          6’d46: begin IDOP=‘SWR______; IDATTR=‘STORE_4B_UNALIGN;                             end
        endcase
    end

    always @( posedge CLK or negedge RST_X ) begin
        if(!RST_X) {inst_rt, inst_rd, inst_shamt, inst_imm, inst_addr, inst_attr, inst_op} <= 0;
        else if(CPUEXE && state==‘CPU_ID) begin
            inst_rt    <= IDRT;
            inst_rd    <= IDRD;
            inst_shamt <= IDSHAMT;
            inst_imm   <= IDIMM;
            inst_addr  <= IDADDR;
            inst_attr  <= IDATTR;
            inst_op    <= IDOP;
        end
    end

    /**************************************************************************************************/
    /* Mips:: register file access                                                                    */
    /**************************************************************************************************/
    assign GPRNUM0 = (state==‘CPU_ID) ? IDRS : inst_dst; // use IDRS if reg_read else write to inst_dst
    assign GPRNUM1 = (state==‘CPU_ID) ? IDRT : ’d7;      // use IDRT if reg_read else SYSCALL $7<=0
    assign CPRNUM  = IDRD;
    assign RFDST   = ((inst_attr & ‘WRITE_RD ) || (inst_attr & ‘WRITE_RD_COND)) ? inst_rd :
                      (inst_attr & ‘WRITE_RT ) ? inst_rt :
                      (inst_attr & ‘WRITE_RRA) ? ’d31 : (inst_attr & ‘SYSTEM_CALL) ? ’d2 : ’d0;

    always @( posedge CLK or negedge RST_X ) begin
        if(!RST_X) {inst_rrs, inst_rrt, inst_cpr, inst_dst} <= 0;
        else if(CPUEXE && state==‘CPU_RF) begin
            inst_rrs   <= GPRREADDT0;
            inst_rrt   <= GPRREADDT1;
            inst_cpr   <= CPRREADDT;
            inst_dst   <= RFDST;
        end
    end

    assign DIVMULINIT = (CPUEXE && state==‘CPU_RF); // start signal for div & mul

    /**************************************************************************************************/
    /* Mips:: execute                                                                                 */
    /**************************************************************************************************/
    function [3:0] MASKUA; // MASK_UN_ALIGN
        input [24:0] eaddr;
        input        left;
        MASKUA = (left)? 4’b1111 << (3 − eaddr[1:0]) : 4’b1111 >> eaddr[1:0];
    endfunction

    assign SET32I = (inst_imm[15]) ? {16’hffff, inst_imm} : {16’h0000, inst_imm};
    assign SETADI = SET32I[‘ADDR];
    assign RRT_S = inst_rrt;
    assign RRS_U = inst_rrs;
    assign RRT_U = inst_rrt;
    assign EXSIGNED = (inst_op == ‘MULT_____ || inst_op == ‘DIV______);

    always @(DUBUSY or DURSLT or MUBUSY or MURSLT or inst_shamt or inst_imm or inst_addr or
             inst_op or inst_pc or inst_rrs or inst_rrt or SET32I or SETADI or
             RRT_S or RRS_U or RRT_U or hi or lo or inst_cpr) begin
        EXRSLT     = 0;
        EXRSLTHI   = 0;
        EXNPC      = 0;
        EXC        = 0;
        EXEADDR    = 0;
        EXMASK     = 0;
        EXDATAEXT  = 0;
        EXBUSY     = 0;
        case ( inst_op )
          ‘SYSCALL__ : begin EXRSLT   = 1;                                                           end
          ‘ERET_____ : begin EXRSLT   = 1;                                                           end
          ‘NOP______ : begin EXRSLT   = 1;                                                           end
          ‘SLL______ : begin EXRSLT   = RRT_U <<  inst_shamt;                                        end
          ‘SRL______ : begin EXRSLT   = RRT_U >>  inst_shamt;                                        end
          ‘SRA______ : begin EXRSLT   = RRT_S >>> inst_shamt;                                        end
          ‘SLLV_____ : begin EXRSLT   = RRT_U <<  RRS_U[4:0];                                        end
          ‘SRLV_____ : begin EXRSLT   = RRT_U >>  RRS_U[4:0];                                        end
          ‘SRAV_____ : begin EXRSLT   = RRT_S >>> RRS_U[4:0];                                        end
```

```
          ‘JR_______ : begin EXNPC    = RRS_U;                      EXC = 1;                         end
          ‘JR_HB____ : begin EXNPC    = RRS_U;                      EXC = 1;                         end
          ‘JALR_____ : begin EXRSLT   = inst_pc + 8; EXNPC = RRS_U; EXC = 1;                         end
          ‘JALR_HB__ : begin EXRSLT   = inst_pc + 8; EXNPC = RRS_U; EXC = 1;                         end
          ‘MOVZ_____ : begin EXRSLT   = RRS_U;                      EXC  = (RRT_U == 0);             end
          ‘MOVN_____ : begin EXRSLT   = RRS_U;                      EXC  = (RRT_U != 0);             end
          ‘MFHI_____ : begin EXRSLT   = hi;                                                          end
          ‘MTHI_____ : begin EXRSLTHI = RRS_U;                                                       end
          ‘MFLO_____ : begin EXRSLT   = lo;                                                          end
          ‘MTLO_____ : begin EXRSLT   = RRS_U;                                                       end
          ‘MULT_____ : begin {EXRSLTHI, EXRSLT} = MURSLT;               EXBUSY = MUBUSY;             end
          ‘MULTU____ : begin {EXRSLTHI, EXRSLT} = MURSLT;               EXBUSY = MUBUSY;             end
          ‘MUL______ : begin {EXRSLTHI, EXRSLT} = MURSLT;               EXBUSY = MUBUSY;             end
          ‘DIV______ : begin {EXRSLTHI, EXRSLT} = (RRT_U) ? DURSLT : 0; EXBUSY = DUBUSY;             end
          ‘DIVU_____ : begin {EXRSLTHI, EXRSLT} = (RRT_U) ? DURSLT : 0; EXBUSY = DUBUSY;             end
          ‘ADD______ : begin EXRSLT   = RRS_U + RRT_U;                                               end
          ‘ADDU_____ : begin EXRSLT   = RRS_U + RRT_U;                                               end
          ‘SUB______ : begin EXRSLT   = RRS_U − RRT_U;                                               end
          ‘SUBU_____ : begin EXRSLT   = RRS_U − RRT_U;                                               end
          ‘AND______ : begin EXRSLT   = RRS_U & RRT_U;                                               end
          ‘OR_______ : begin EXRSLT   = RRS_U | RRT_U;                                               end
          ‘XOR______ : begin EXRSLT   = RRS_U ^ RRT_U;                                               end
          ‘NOR______ : begin EXRSLT   = ~(RRS_U | RRT_U);                                            end
          ‘SLT______ : begin EXRSLT   = (RRS_U[31] ^ RRT_U[31]) ? RRS_U[31] :
                                        (RRS_U < RRT_U)         ? 1 : 0;                             end
          ‘SLTU_____ : begin EXRSLT   = (RRS_U < RRT_U) ? 32’h1 : 32’h0;                             end
          ‘BLTZ_____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC =  RRS_U[31];               end
          ‘BGEZ_____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC = ~RRS_U[31];               end
          ‘BLTZL____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC =  RRS_U[31];               end
          ‘BGEZL____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC = ~RRS_U[31];               end
          ‘BLTZAL___ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC =  RRS_U[31];
                             EXRSLT   = inst_pc + 8;                                                 end
          ‘BGEZAL___ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC = ~RRS_U[31];
                             EXRSLT   = inst_pc + 8;                                                 end
          ‘BLTZALL__ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC =  RRS_U[31];
                             EXRSLT   = inst_pc + 8;                                                 end
          ‘BGEZALL__ : begin EXNPC    = inst_pc + (SETADI << 2) + 4; EXC = ~RRS_U[31];
                             EXRSLT   = inst_pc + 8;                                                 end
          ‘J________ : begin EXNPC    = (inst_pc & 32’hf0000000) | (inst_addr << 2); EXC = 1;        end
          ‘JAL______ : begin EXNPC    = (inst_pc & 32’hf0000000) | (inst_addr << 2); EXC = 1;
                             EXRSLT   = inst_pc + 8;                                                 end
          ‘BEQ______ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = (RRS_U == RRT_U);                                            end
          ‘BEQL_____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = (RRS_U == RRT_U);                                            end
          ‘BNE______ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = (RRS_U != RRT_U);                                            end
          ‘BNEL_____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = (RRS_U != RRT_U);                                            end
          ‘BLEZ_____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = ( RRS_U[31] || (RRS_U == 0)) ? 1 : 0;                        end
          ‘BLEZL____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = ( RRS_U[31] || (RRS_U == 0)) ? 1 : 0;                        end
          ‘BGTZ_____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = (~RRS_U[31] && (RRS_U != 0)) ? 1 : 0;                        end
          ‘BGTZL____ : begin EXNPC    = inst_pc + (SETADI << 2) + 4;
                             EXC      = (~RRS_U[31] && (RRS_U != 0)) ? 1 : 0;                        end
          ‘ADDI_____ : begin EXRSLT   = RRS_U + SET32I;                                              end
          ‘ADDIU____ : begin EXRSLT   = RRS_U + SET32I;                                              end
          ‘SLTI_____ : begin EXRSLT   = (RRS_U[31] ^ inst_imm[15]) ? RRS_U[31] :
                                        (RRS_U < SET32I)           ? 1 : 0;                          end
          ‘SLTIU____ : begin EXRSLT   = (RRS_U < SET32I)           ? 1 : 0;                          end
          ‘ANDI_____ : begin EXRSLT   = RRS_U & {16’h0, inst_imm};                                   end
          ‘ORI______ : begin EXRSLT   = RRS_U | {16’h0, inst_imm};                                   end
          ‘XORI_____ : begin EXRSLT   = RRS_U ^ {16’h0, inst_imm};                                   end
          ‘LUI______ : begin EXRSLT   = {inst_imm, 16’h0};                                           end
          ‘MFC0_____ : begin EXRSLT   = inst_cpr;                                                    end
          ‘MTC0_____ : begin EXRSLT   = RRT_U;                                                       end
          ‘LW_______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b1111;
                             EXRSLT   = 0;                                                           end
          ‘LB_______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b0001;
                             EXRSLT   = 0;                  EXDATAEXT = 2’b11;                       end
          ‘LBU______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b0001;
                             EXRSLT   = 0;                                                           end
          ‘LH_______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b0011;
                             EXRSLT   = 0;                  EXDATAEXT = 2’b10;                       end
          ‘LHU______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b0011;
                             EXRSLT   = 0;                                                           end
          ‘LWL______ : begin EXEADDR  = RRS_U + SETADI − 3; EXMASK = MASKUA(RRS_U + SETADI,1);
                             EXRSLT   = RRT_U;                                                       end
          ‘LWR______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = MASKUA(RRS_U + SETADI,0);
                             EXRSLT   = RRT_U;                                                       end
          ‘SWR______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = MASKUA(RRS_U + SETADI,0);
                             EXRSLT   = RRT_U;                                                       end
          ‘SWL______ : begin EXEADDR  = RRS_U + SETADI − 3; EXMASK = MASKUA(RRS_U + SETADI,1);
                             EXRSLT   = RRT_U;                                                       end
          ‘SB_______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b0001;
                             EXRSLT   = RRT_U;                                                       end
          ‘SH_______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b0011;
                             EXRSLT   = RRT_U;                                                       end
          ‘SW_______ : begin EXEADDR  = RRS_U + SETADI;     EXMASK = 4’b1111;
                             EXRSLT   = RRT_U;                                                        end
          default    : begin end
        endcase
    end

    always @( posedge CLK or negedge RST_X ) begin
        if (!RST_X)
            {inst_rslt, inst_rslthi, inst_eaddr, inst_cond, inst_npc, inst_datamask} <= 0;
        else if (CPUEXE && state==‘CPU_EX ) begin
            inst_rslt     <= EXRSLT;
            inst_rslthi   <= EXRSLTHI;
            inst_eaddr    <= EXEADDR;
            inst_cond     <= EXC;
            inst_npc      <= EXNPC;
            inst_datamask <= EXMASK;
        end else if (CPUEXE && state >= ‘CPU_MA0 && state <= ‘CPU_MA2) begin
            inst_eaddr    <= inst_eaddr + 1;
        end
    end

    /***************************************************************************************************/
    /* Mips:: memory access                                                                            */
    /***************************************************************************************************/
    ///// memory store
    assign DATA_OUT = (state == ‘CPU_MA0) ? inst_rslt[ 7: 0] :
                      (state == ‘CPU_MA1) ? inst_rslt[15: 8] :
                      (state == ‘CPU_MA2) ? inst_rslt[23:16] :
                      (state == ‘CPU_MA3) ? inst_rslt[31:24] : 0;

    assign WE       = (CPUEXE && (inst_attr & ‘STORE_ANY) &&
                       ((state == ‘CPU_MA0 && inst_datamask[0]) ||
                        (state == ‘CPU_MA1 && inst_datamask[1]) ||
                        (state == ‘CPU_MA2 && inst_datamask[2]) ||
                        (state == ‘CPU_MA3 && inst_datamask[3])));

    always @( posedge CLK or negedge RST_X ) begin  ///// memory load
        if (!RST_X) inst_loaddata <= 0;
        else if (CPUEXE)
          if      (state == ‘CPU_MA1) inst_loaddata[ 7: 0] <= DATA_IN;
          else if (state == ‘CPU_MA2) inst_loaddata[15: 8] <= DATA_IN;
          else if (state == ‘CPU_MA3) inst_loaddata[23:16] <= DATA_IN;
          else if (state == ‘CPU_MA4) inst_loaddata[31:24] <= DATA_IN;
    end

    /***************************************************************************************************/
    /* Mips:: write back                                                                               */
    /***************************************************************************************************/
    assign MALOADDATARAW = inst_loaddata[31:0];
    assign MALOADMASK = {{8{inst_datamask[3]}}, {8{inst_datamask[2]}},
                         {8{inst_datamask[1]}}, {8{inst_datamask[0]}}};
    assign MALOADDATA = (EXDATAEXT[0]) ? {{24{MALOADDATARAW[ 7]}}, MALOADDATARAW[ 7:0]} :
                        (EXDATAEXT[1]) ? {{16{MALOADDATARAW[15]}}, MALOADDATARAW[15:0]} :
                        (MALOADDATARAW & MALOADMASK) | (inst_rslt & ~MALOADMASK);

    assign GPRWE0 = ((CPUEXE && state==‘CPU_WB && ~CPEXCOCCUR) &&
                     ~((inst_attr & ‘WRITE_RD_COND) && inst_cond==0));
    assign GPRWE1 = ((CPUEXE && state==‘CPU_WB && ~CPEXCOCCUR) &&   (inst_attr & ‘SYSTEM_CALL));
    assign CPRWE  = ((CPUEXE && state==‘CPU_WB && ~CPEXCOCCUR) &&   (inst_attr & ‘WRITE_CP0));
    assign GPRWRITEDT0 = (inst_attr & ‘LDST) ? MALOADDATA : inst_rslt;
    assign GPRWRITEDT1 = 32’h0;  // 2nd write port is just for SYSCALL (REG_A3 <= 0)
    assign CPRWRITEDT  = inst_rslt;


    always @( posedge CLK or negedge RST_X ) begin
        if(!RST_X) {hi, lo} <= 0;
        else if(CPUEXE && state==‘CPU_WB && ~CPEXCOCCUR) begin
             if(inst_attr & ‘WRITE_HI) hi <= inst_rslthi;
             if(inst_attr & ‘WRITE_LO) lo <= inst_rslt;
        end
    end

    /***************************************************************************************************/
    /* Mips:: setnpc()                                                                                 */
    /***************************************************************************************************/
    assign CPEXCSET  = (CPUEXE && state==‘CPU_EX && (inst_attr & ‘SYSTEM_CALL));
    assign CPEXCCLR  = (CPUEXE && state==‘CPU_EX && (inst_attr & ‘BRANCH_ERET));
    assign CPEXCACK  = (CPUEXE && state==‘CPU_WB && CPEXCOCCUR);
    assign CPEXCCODE = 4’d8; // EXC_SYSCALL
    assign CPEXCEPC  = (exec_delay) ? pc − 4 : pc;
    assign CPEXCBD   = exec_delay;

    always @( posedge CLK or negedge RST_X ) begin
        if(!RST_X) {pc, delay_npc, exec_delay} <= 0;
        else if(CPUEXE && state == ‘CPU_WB) begin
            if (CPEXCOCCUR) begin
                pc          <= CPEXCNPC;
                exec_delay  <= 0;
            end else if( exec_delay ) begin
                pc          <= delay_npc;
```

```
                delay_npc   <= 0;
                exec_delay  <= 0;
            end else if( ((inst_attr & ‘BRANCH) || (inst_attr & ‘BRANCH_LIKELY)) && inst_cond ) begin
                pc          <= pc + 4;
                delay_npc   <= inst_npc;
                exec_delay  <= 1;
            end else if  (inst_attr & ‘BRANCH_ERET) begin
                pc          <= CPEXCNPC;
            end else if( (inst_attr & ‘BRANCH_LIKELY) && !inst_cond ) begin
                pc          <= pc + 8;
            end else begin
                pc          <= pc + 4;
            end
        end
    end
endmodule

/*******************************************************************************************************/
```

```
file name: init.v
/******************************************************************************************************/
/* MieruPC2010: Program Loader                                                    ArchLab. TOKYO TECH */
/******************************************************************************************************/
‘include "../rtl/define.v"
‘default_nettype none

/* Program Loader: Initialize the main memory, copy memory image to SRAM                              */
/******************************************************************************************************/
module PROGLOADER (CLK, RST_X, ADDR, DATA_OUT, WE, DATA_IN, DONE);
    input          CLK, RST_X;
    input  [7:0]   DATA_IN;
    output [‘ADDR] ADDR;
    output [7:0]   DATA_OUT;
    output         WE, DONE;

    reg [‘ADDR]    ADDR, writeaddr;
    reg [7:0]      DATA_OUT;
    reg            WE, DONE;

    reg [3:0]      phase;
    reg [10:0]     block;
    reg [26:0]     timeout;
    reg [1:0]      cnt;

    always @(posedge CLK or negedge RST_X) begin
        if (!RST_X) begin
            timeout <= 0;
            DONE    <= 0;
        end else if (!DONE) begin
            timeout <= timeout + 1;
            DONE    <= (phase == 8 || timeout == 100000000);
        end
    end

    always @(posedge CLK or negedge RST_X) begin
        if(!RST_X) begin
            ADDR      <= 0;
            writeaddr <= 0;
            DATA_OUT  <= 0;
            WE        <= 0;
            phase     <= 0;
            block     <= ‘MMC_START_BLOCK;
            cnt       <= 0;
        end else if (!DONE) begin
            if (phase == 0) begin // write block address
                ADDR      <= ’h800109;
                DATA_OUT  <= {block[6:0], 1’b0};
                WE        <= 1;
                phase     <= 1;
            end else if (phase == 1) begin
                ADDR      <= ’h80010a;
                DATA_OUT  <= {4’h0, block[10: 7]};
                WE        <= 1;
                phase     <= 2;
            end else if (phase == 2) begin
                ADDR      <= ’h80010b;
                DATA_OUT  <= 0;
                WE        <= 1;
                phase     <= 3;
            end else if (phase == 3) begin // wait for ready signal
                ADDR      <= ’h800108;
                WE        <= 0;
                phase     <= (cnt == 3) ? 4 : phase;
                cnt       <= cnt + 1;
            end else if (phase == 4) begin
                phase     <= (DATA_IN == 8’h01) ? 5 : phase;
            end else if (phase == 5) begin // copy to main memory
                ADDR      <= ’h800200 + writeaddr[8:0];
                WE        <= 0;
                phase     <= (cnt == 1) ? 6 : phase;
                cnt       <= (cnt == 1) ? 0 : 1;
            end else if (phase == 6) begin
                ADDR      <= writeaddr;
                writeaddr <= writeaddr + 1;
                DATA_OUT  <= DATA_IN;
                WE        <= 1;
                phase     <= (writeaddr[8:0] == 9’h1ff) ? 7 : 5;
            end else if (phase == 7) begin // increment block
                WE        <= 0;
                block     <= block + 1;
                phase     <= (block == ‘MMC_LAST_BLOCK) ? 8 : 0;
            end
        end
    end
endmodule

/******************************************************************************************************/
```

```
file name: iocon.v
/******************************************************************************************************/
/* MieruPC2010                                                                    ArchLab. TOKYO TECH */
/******************************************************************************************************/
‘include "../rtl/define.v"
‘default_nettype none

/** I/O Controller                                                                                    */
/******************************************************************************************************/
module IOCON(CLK, RST_X, KEY_CLK, KEY_IN, LCD, GPIO, ADDR, DATA_IN, DATA_OUT, WE, MEM_WE, MEM_Q,
             MMC_CSX, MMC_DIN, MMC_DOUT, MMC_SCLK, SW, GPI);
    input         CLK, RST_X, GPI; // clock & reset
    input         KEY_CLK, KEY_IN; // keyboard block and keyboard data
    input [‘ADDR] ADDR;
    input [7:0]   DATA_OUT;
    input         WE;
    input [7:0]   MEM_Q;
    input         MMC_DOUT;
    input [2:0]   SW;
    output        LCD;
    output [7:0]  DATA_IN;
    output        MEM_WE;
    output        MMC_CSX, MMC_DIN, MMC_SCLK;
    inout [15:0]  GPIO;

    wire          kbcon_ack;
    reg           kbcon_busy;
    wire [7:0]    vk_on;
    wire [7:0]    vk_off;
    wire          lcd_rdy, lcd_txd;
    reg  [7:0]    lcd_val;
    reg           lcd_we;
    wire [31:0]   counter;

    reg           ior_we;
    reg [5:0]     ior_ain;
    reg  [31:0]   ior_din;
    wire [31:0]   ior_dout;
    wire [7:0]    ior_q;

    wire [8:0]    cc_ram_addr, cr_addr;
    wire [7:0]    cc_ram_d, cr_din, cr_dout;
    wire          cc_ram_we, cc_we, cc_rdy, cr_we;
    reg           cc_dirty;
    wire [22:0]   cc_block;

    wire targ_kc  = (ADDR >> 1 == ’h400080); // ’h800100 − ’h800101, Keyboard
    wire targ_cnt = (ADDR >> 2 == ’h200043); // ’h80010c − ’h80010f, 1kHz Counter
    wire targ_cc  = (ADDR >> 2 == ’h200042); // ’h800108 − ’h80010b, MMC Control
    wire targ_cr  = (ADDR >> 9 == ’h4001);   // ’h800200 − ’h8003ff, MMC Data

    /**************************************************************************************************/
    gpiocon gpiocon(CLK, RST_X, GPIO, WE, ADDR, DATA_OUT[0]);
    counter1kHz cnt1kHz(.CLK(CLK), .RST_X(RST_X), .CNT_OUT(counter));

    /* Keyboard controller */
    /**************************************************************************************************/
//  kbcon_mieru kcb(.CLK(CLK), .RST_X(RST_X), .KEY_CLK(KEY_CLK), .KEY_IN(KEY_IN), .ACK(kbcon_ack),
//                  .VK_ON(vk_on), .VK_OFF(vk_off));

    assign kbcon_ack = (ADDR == ’h800100) && kbcon_busy;

    always @(posedge CLK or negedge RST_X) begin
        if (!RST_X)                      kbcon_busy <= 0;
        else if (ior_ain == 0 && ior_we) kbcon_busy <= (ior_din != 0);
    end

    /* LCD controller */
    /**************************************************************************************************/
//  lcdcon lc(.CLK(CLK), .RST_X(RST_X), .VALUE(lcd_val), .WE(lcd_we), .TXD(lcd_txd), .READY(lcd_rdy));

    assign LCD  = ~lcd_txd;

    always @(posedge CLK or negedge RST_X) begin
        if (!RST_X) begin
            lcd_val <= 0;
            lcd_we  <= 0;
        end else begin
            lcd_val <= DATA_OUT;
            lcd_we  <= WE & (ADDR == ’h800104);
        end
    end

    /* MMC Controller & MMC Sector RAM                                                                */
    /**************************************************************************************************/
    mmccon mmcc(.CLK(CLK), .RST_X(RST_X),
                .MMC_CSX(MMC_CSX), .MMC_DIN(MMC_DIN), .MMC_DOUT(MMC_DOUT), .MMC_SCLK(MMC_SCLK),
                .RAM_ADDR(cc_ram_addr), .RAM_D(cc_ram_d), .RAM_WE(cc_ram_we), .RAM_Q(cr_dout),
                .RAM_DIRTY(cc_dirty), .CORE_WE(cc_we), .CORE_D(DATA_OUT), .CORE_ADDR(ADDR[1:0]),
                .BLOCK(cc_block), .READY(cc_rdy));

    mmcram mmcr (.CLK(CLK), .WE(cr_we), .ADDR(cr_addr), .DIN(cr_din), .DOUT(cr_dout));

    assign cc_we     = WE & targ_cc;
    assign cr_we     = (cc_rdy) ? WE & targ_cr : cc_ram_we;
    assign cr_addr   = (cc_rdy) ? ADDR[8:0]    : cc_ram_addr;
    assign cr_din    = (cc_rdy) ? DATA_OUT     : cc_ram_d;
    always @(posedge CLK or negedge RST_X) begin
        if (!RST_X)            cc_dirty <= 0;
        else if (cc_ram_we)    cc_dirty <= 0;
        else if (WE & targ_cr) cc_dirty <= 1;
    end

    /***************************************************************************************************/
    ioram ior(.CLK(CLK), .WE(ior_we),
              .AIN(ior_ain), .DIN(ior_din), .AOUT(ADDR[7:2]), .DOUT(ior_dout));

    reg  [2:0] ior_state;
    always @(posedge CLK or negedge RST_X) begin
        if(!RST_X) ior_state <= 0;
        else       ior_state <= (lcd_we) ? 1 :(WE & targ_cc) ? 2 : ior_state + 1;
    end

    always @(ior_state or vk_off or vk_on or targ_kc or lcd_rdy or cc_block or cc_rdy or
             cc_dirty or counter or targ_cnt or GPIO) begin
        ior_we = 1;
        if (ior_state == 0) begin
            ior_ain    = 6’h00;     // 0x800100
            ior_din    = {16’h0000, vk_off, vk_on};
            ior_we     = ~targ_kc; // don’t update while reading
        end else if (ior_state == 1) begin
            ior_ain    = 6’h01;     // 0x800104, LCD
            ior_din    =  {31’b0, lcd_rdy};
        end else if (ior_state == 2) begin
            ior_ain    = 6’h02;     // 0x800108, MMC control
            ior_din    =  {cc_block, 7’h00, cc_dirty, cc_rdy};
        end else if (ior_state == 3) begin
            ior_ain    = 6’h03;     // 0x80010c, 1kHz counter
            ior_din    = counter;
            ior_we     = ~targ_cnt;   // don’t update while reading
        end else if (ior_state == 4) begin
            ior_ain    = 6’h3c;     // 0x8001f0, GPIO
            ior_din    = {7’b0, GPIO[ 3], 7’b0, GPIO[ 2], 7’b0, GPIO[ 1], 7’b0, GPIO[ 0]};
        end else if (ior_state == 5) begin
            ior_ain    = 6’h3d;     // 0x8001f4, GPIO
            ior_din    = {7’b0, GPIO[ 7], 7’b0, GPIO[ 6], 7’b0, GPIO[ 5], 7’b0, GPIO[ 4]};
        end else if (ior_state == 6) begin
            ior_ain    = 6’h3e;     // 0x8001f8, GPIO
            ior_din    = {7’b0, GPIO[11], 7’b0, GPIO[10], 7’b0, GPIO[ 9], 7’b0, GPIO[ 8]};
        end else if (ior_state == 7) begin
            ior_ain    = 6’h3f;     // 0x8001fc, GPIO
            ior_din    = {7’b0,      GPI, 7’b0,   SW[ 2], 7’b0,   SW[ 1], 7’b0,   SW[ 0]};
        end
    end

    reg [1:0] byte_select;
    reg  targr_io, targr_mm;
    always @(posedge CLK or negedge RST_X) begin
        if(!RST_X) begin
            targr_io    <= 0;
            targr_mm    <= 0;
            byte_select <= 0;
        end else begin
            targr_io    <= (ADDR >> 8 == ’h8001); // 0x800100 − 0x8001ff
            targr_mm    <= targ_cr;               // 0x800200 − 0x8003ff
            byte_select <= ADDR[1:0];
        end
    end

    assign ior_q = (byte_select==0) ? ior_dout[7:0]   :
                   (byte_select==1) ? ior_dout[15:8]  :
                   (byte_select==2) ? ior_dout[23:16] : ior_dout[31:24];

    /***************************************************************************************************/
    assign MEM_WE  = WE && ((ADDR >> 19) == 0); // SRAM Write Enable, higher bits must be zero
    assign DATA_IN = (targr_io) ? ior_q   :     // general memory mapped I/O, 0x800100 − 0x8001ff
                     (targr_mm) ? cr_dout :     // MMC data,                 0x800200 − 0x8003ff
                     MEM_Q;                     // SRAM data                 0x000000 − 0x7fffff
endmodule

/*******************************************************************************************************/
module counter1kHz(CLK, RST_X, CNT_OUT);
    input         CLK, RST_X;
    output [31:0] CNT_OUT;

    reg [31:0]    CNT_OUT;
    reg [15:0]    cntwait;

    always @(posedge CLK or negedge RST_X) begin
        if (!RST_X) begin
```

```
            CNT_OUT <= 0;
            cntwait <= 1;
        end else begin
            if(cntwait >= ’TIMER_CNT_WAIT) begin
                CNT_OUT <= CNT_OUT + 1;
                cntwait <= 1;
            end
            else cntwait <= cntwait + 1;
        end
    end
endmodule

/*******************************************************************************************************/
module ioram(CLK, WE, AIN, DIN, AOUT, DOUT);
    input              CLK, WE;
    input [5:0]        AIN, AOUT;
    input [31:0]       DIN;
    output [31:0]      DOUT;

    reg [31:0]         DOUT;
    reg [31:0]         mem[0:63]; // 256B

    always @(posedge CLK) begin // synthesis attribute ram_style of mem is block;
        if (WE) mem[AIN] <= DIN;
        DOUT <= mem[AOUT];
    end
endmodule

/*******************************************************************************************************/
module mmcram(CLK, WE, ADDR, DIN, DOUT);
    input         CLK, WE;
    input [8:0]   ADDR;
    input [7:0]   DIN;
    output [7:0]  DOUT;

    reg [7:0]     DOUT;
    reg [7:0]     mem[0:511]; // 512B

    always @(posedge CLK) begin // synthesis attribute ram_style of mem is block;
        if (WE) mem[ADDR] <= DIN;
        DOUT <= mem[ADDR];
    end
endmodule

/*******************************************************************************************************/
```

file name: MieruEmb.ucf

```
##############################################################################
# UCF file for MieruEMB System Board V1.1  2011−09−24    ArchLab. TOKYO TECH #
##############################################################################
#### SYSTEM ##################################################################
NET CLK        LOC="P36" | PERIOD=40 MHz ; #
NET ULED<0>    LOC="P32" ;                 # D2 (Light Emitting Diode 2)
NET ULED<1>    LOC="P33" ;                 # D3 (Light Emitting Diode 3)
NET ULED<2>    LOC="P34" ;                 # D4 (Light Emitting Diode 4)
NET SW<0>      LOC="P68" ;                 # SW1 (Switch 1)
NET SW<1>      LOC="P70" ;                 # SW2 (Switch 2)
NET SW<2>      LOC="P71" ;                 # SW3 (Switch 3)
NET GPIO<0>    LOC="P40" ;                 # General Purpose I/O
NET GPIO<1>    LOC="P35" ;                 # General Purpose I/O / ULED[3]
NET GPI        LOC="P38" ;                 # General Purpose Input
#### On board SRAM ###########################################################
NET SRAM_A<0>  LOC="P10" ;                 #
NET SRAM_A<1>  LOC="P9"  ;                 #
NET SRAM_A<2>  LOC="P5"  ;                 #
NET SRAM_A<3>  LOC="P4"  ;                 #
NET SRAM_A<4>  LOC="P3"  ;                 #
NET SRAM_A<5>  LOC="P91" ;                 #
NET SRAM_A<6>  LOC="P90" ;                 #
NET SRAM_A<7>  LOC="P86" ;                 #
NET SRAM_A<8>  LOC="P85" ;                 #
NET SRAM_A<9>  LOC="P84" ;                 #
NET SRAM_A<10> LOC="P78" ;                 #
NET SRAM_A<11> LOC="P79" ;                 #
NET SRAM_A<12> LOC="P83" ;                 #
NET SRAM_A<13> LOC="P11" ;                 #
NET SRAM_A<14> LOC="P12" ;                 #
NET SRAM_A<15> LOC="P23" ;                 #
NET SRAM_A<16> LOC="P27" ;                 #
NET SRAM_A<17> LOC="P26" ;                 #
NET SRAM_A<18> LOC="P24" ;                 #
NET SRAM_D<0>  LOC="P2"  | PULLDOWN ;      #
NET SRAM_D<1>  LOC="P98" | PULLDOWN ;      #
NET SRAM_D<2>  LOC="P95" | PULLDOWN ;      #
NET SRAM_D<3>  LOC="P94" | PULLDOWN ;      #
NET SRAM_D<4>  LOC="P15" | PULLDOWN ;      #
NET SRAM_D<5>  LOC="P16" | PULLDOWN ;      #
NET SRAM_D<6>  LOC="P17" | PULLDOWN ;      #
NET SRAM_D<7>  LOC="P18" | PULLDOWN ;      #
NET SRAM_OE_X  LOC="P22" ;                 #
NET SRAM_WE_X  LOC="P92" ;                 #
##############################################################################
NET LCD        LOC="P67" ;                 # P1
#### MultiMediaCard ##########################################################
NET MMC_CS_X   LOC="P41" ;                 #
NET MMC_DIN    LOC="P42" ;                 #
NET MMC_SCLK   LOC="P43" ;                 #
NET MMC_DOUT   LOC="P47" | PULLUP ;        #
#### Mini−LCD (Liquid Crystal Display) #######################################
NET LCD_CS0    LOC="P48" ;                 #
NET LCD_CD     LOC="P49" ;                 #
NET LCD_WR     LOC="P53" ;                 #
NET LCD_RSTB   LOC="P54" ;                 #
NET LCD_D<0>   LOC="P57" ;                 #
NET LCD_D<1>   LOC="P58" ;                 #
NET LCD_D<2>   LOC="P60" ;                 #
NET LCD_D<3>   LOC="P61" ;                 #
NET LCD_D<4>   LOC="P62" ;                 #
NET LCD_D<5>   LOC="P63" ;                 #
NET LCD_D<6>   LOC="P65" ;                 #
NET LCD_D<7>   LOC="P66" ;                 #
##############################################################################
```