

Nov 12, 13 0:49

code.txt

Page 1/17

```
file name: define.v
1  /**
2  /* The 1st IPSJ SIG-ARC High-Performance Processor Design Contest          Version 2013-10 */
3  /* From MieruPC2010: MipsCore                                           ArchLab. TOKYO TECH */
4  /**
5
6  'define PROG_SIZE (512*1024) // MEMORY IMAGE FILE Load Size in byte (default 512KB)
7  'define IMEM_SIZE ( 64*1024) // Instruction Memory Size in byte (default 64KB)
8
9  /**
10
11 /* Clock Interval Definition
12 /**
13 'define SYS_CLOCK 100 // board input clock in MHz
14 'define DCM_CLKFX_MULTIPLY 3 // CLK = SYS_CLOCK * DCM_CLKFX_MULTIPLY / DCM_CLKFX_DIVIDE
15 'define DCM_CLKFX_DIVIDE 10 // note: CLKFX_DIVIDE must be 1~32, CLKFX_MULTIPLY must be 2~32
16
17 'define DCM_DRAMC_MULTIPLY 6 // for DRAM CLK
18 'define DCM_DRAMC_DIVIDE 10 //
19
20 'define SERIAL_WCNT ('SYS_CLOCK * 'DCM_CLKFX_MULTIPLY / 'DCM_CLKFX_DIVIDE) // 1M baud UART wait count
21 /**
22 'define WORD 31:0 // 1 word is 32 bit
23 'define ADDR 31:0 // Address Range of 32bit
24
25 /* Instruction Attribute Definition
26 /**
27 'define ATTR 7:0 // type of instruction Attribute
28
29 'define LD_1B 8'b00000001
30 'define LD_2B 8'b00000010
31 'define LD_4B 8'b00000100
32 'define ST_1B 8'b00001000
33 'define ST_2B 8'b00010000
34 'define ST_4B 8'b00100000
35 'define BRANCH 8'b01000000
36 'define LD_ANY 8'b00000111 // MASK
37 'define ST_ANY 8'b00111000 // MASK
38 'define LDST_ANY 8'b00111111 // MASK
39
40 /* Unique instruction operation number or OPN (unique instruction id)
41 /**
42 'define OPNT 5:0 // type of operation number
43
44 'define NOP 6'd00 // NOP must be zero!
45 'define ERROR 6'd01
46 'define SLL 6'd02
47 'define SRL 6'd03
48 'define SRA 6'd04
49 'define SLLV 6'd05
50 'define SRLV 6'd06
51 'define SRAV 6'd07
52 'define JR 6'd08
53 'define JALR 6'd09
54 'define MFHI 6'd10
55 'define MFLO 6'd11
56 'define MULT 6'd12
57 'define MULTU 6'd13
58 'define ADD 6'd14
59 'define ADDU 6'd15
60 'define SUB 6'd16
61 'define SUBU 6'd17
62 'define AND 6'd18
63 'define OR 6'd19
64 'define XOR 6'd20
65 'define NOR 6'd21
66 'define SLT 6'd22
67 'define SLTU 6'd23
68 'define J 6'd24
69 'define JAL 6'd25
70 'define BEQ 6'd26
71 'define BNE 6'd27
72 'define ADDI 6'd28
73 'define ADDIU 6'd29
74 'define SLTI 6'd30
75 'define SLTIU 6'd31
76 'define ANDI 6'd32
77 'define ORI 6'd33
78 'define XORI 6'd34
79 'define LUI 6'd35
80 'define LW 6'd38
81 'define SW 6'd43
82 'define BLEZ 6'd44
83 'define BGTZ 6'd45
84 'define BLTZ 6'd46
85 'define BGEZ 6'd47
86 'define DIV 6'd48
87 'define DIVU 6'd49
88 /**
```

Tuesday November 12, 2013

code.txt

Nov 12, 13 0:49

code.txt

Page 2/17

1/9

Nov 12, 13 0:49

code.txt

Page 3/17

```

file name: MieruSys.v
1  /*****
2  /* The 1st IPSJ SIG-ARC High-Performance Processor Design Contest          Version 2013-10 */
3  /* From MieruPC 2010: Top Level Module                                ArchLab. TOKYO TECH */
4  /*****
5  'include "../rtl/define.v"
6  'default_nettype none
7
8  /*****/
9  module MieruSys(CLK_IN, RST_X_IN, RXD, TXD, ULED,
10     DDR2CLK0, DDR2CLK1, DDR2CKE, DDR2RASN, DDR2CASN, DDR2WEN, DDR2RZQ, DDR2BA, DDR2A, DDR2DQ,
11     DDR2UDQS, DDR2UDQSN, DDR2LDQS, DDR2LDQSN, DDR2LDM, DDR2UDM, DDR2ODT, DDR2ZIO);
12
13     output DDR2CLK0, DDR2CLK1, DDR2CKE, DDR2RASN, DDR2CASN, DDR2WEN, DDR2RZQ, DDR2ZIO;
14     output DDR2LDM, DDR2UDM, DDR2ODT;
15     output [2:0] DDR2BA;
16     output [12:0] DDR2A;
17     inout  [15:0] DDR2DQ;
18     inout  DDR2UDQS, DDR2UDQSN, DDR2LDQS, DDR2LDQSN;
19
20     input      CLK_IN, RST_X_IN, RXD;
21     output reg TXD;
22     output reg [7:0] ULED;
23
24     wire      CLK, DRAM_CLK, RST_X;
25     wire [2:0] CORE_STAT;
26
27     wire [31:0] MM_ADDR;
28     wire [7:0]  MM_DATA;
29     wire        MM_WE;
30     wire [7:0]  MM_DOUT;
31
32     wire [31:0] INIT_ADDR;
33     wire [31:0] INIT_DATA;
34     wire        INIT_WE;
35     wire        INIT_DONE;
36
37     wire [31:0] CORE_I_ADDR, CORE_D_ADDR;
38     wire [31:0] CORE_I_IN, CORE_D_IN, CORE_D_OUT;
39     wire [3:0]  CORE_D_WE;
40     wire        CORE_D_OE;
41
42     wire clk_t, locked, locked2;
43     IBUFG ibuf (.I(CLK_IN), .O(clk_t));
44     clockgen clockgen01(clk_t, CLK, locked);
45     clockgen2 clockgen02(clk_t, DRAM_CLK, locked2);
46     resetgen resetgen01(CLK, (RST_X_IN & locked & locked2), RST_X);
47
48     /*****/
49     wire [31:0] mem_adr; //
50     wire        mem_we, mem_re; //
51     wire [31:0] mem_wr_dat; //
52     wire [31:0] mem_rd_dat; //
53     wire        mem_busy; //
54     wire        calib_done; //
55
56     dram_con mem0(.CLK(CLK), .DRAM_CLK(DRAM_CLK), .RST_X(RST_X), .calib_done(calib_done),
57     .D_ADDR(mem_adr), .D_DIN(mem_wr_dat), .D_WE(mem_we),
58     .D_RE(mem_re), .D_DOUT(mem_rd_dat), .D_BUSY(mem_busy),
59     // DRAM interface
60     .DDR2CLK0 (DDR2CLK0),
61     .DDR2CLK1 (DDR2CLK1),
62     .DDR2CKE (DDR2CKE),
63     .DDR2RASN (DDR2RASN),
64     .DDR2CASN (DDR2CASN),
65     .DDR2WEN (DDR2WEN),
66     .DDR2RZQ (DDR2RZQ),
67     .DDR2ZIO (DDR2ZIO),
68     .DDR2BA (DDR2BA),
69     .DDR2A (DDR2A),
70     .DDR2DQ (DDR2DQ),
71     .DDR2UDQS (DDR2UDQS),
72     .DDR2UDQSN (DDR2UDQSN),
73     .DDR2LDQS (DDR2LDQS),
74     .DDR2LDQSN (DDR2LDQSN),
75     .DDR2LDM (DDR2LDM),
76     .DDR2UDM (DDR2UDM),
77     .DDR2ODT (DDR2ODT)
78     );
79
80     /*****/
81     PLOADER loader(.CLK(CLK), .RST_X(RST_X), .RXD(RXD),
82     .ADDR(INIT_ADDR), .DATA(INIT_DATA), .WE(INIT_WE), .DONE(INIT_DONE));
83
84     /*****/
85     wire core_rst_x = RST_X & INIT_DONE;
86     wire [31:0] WORD_A, WORD_D;
87     wire        WORD_W;
88
89     MIPSORE core(.CLK(CLK), .RST_X(core_rst_x), .STALL(mem_busy), .STAT(CORE_STAT), .IO_IN(0),

```

Nov 12, 13 0:49

code.txt

Page 4/17

```

90     .I_ADDR(CORE_I_ADDR), .I_IN(CORE_I_IN),
91     .D_ADDR(CORE_D_ADDR), .D_IN(CORE_D_IN), .D_OE(CORE_D_OE),
92     .D_OUT(CORE_D_OUT), .D_WE(CORE_D_WE));
93
94     IMEM imem(CLK, RST_X, INIT_ADDR, INIT_DATA, INIT_WE, CORE_I_ADDR, CORE_I_IN);
95
96     assign mem_adr  = (!INIT_DONE) ? INIT_ADDR : CORE_D_ADDR;
97     assign mem_wr_dat = (!INIT_DONE) ? INIT_DATA : CORE_D_OUT;
98     assign mem_we    = (!INIT_DONE) ? INIT_WE : (CORE_D_WE!=0 && CORE_D_ADDR>32'hff); // Note!
99     assign mem_re    = CORE_D_OE;
100    assign CORE_D_IN = mem_rd_dat;
101
102    /*****/
103    reg [25:0] cnt_t; // just for ULED
104    always @(posedge CLK) cnt_t <= cnt_t + 1;
105
106    always @(posedge CLK) ULED[7] <= cnt_t[25];
107    always @(posedge CLK) ULED[6] <= CORE_STAT[1]; // Processor Memory Alignment Error
108    always @(posedge CLK) ULED[5] <= CORE_STAT[0]; // Processor Decode Error
109    always @(posedge CLK) ULED[4] <= mem_busy; // DRAM is working
110    always @(posedge CLK) ULED[3] <= ~TXD; // UART TXD
111    always @(posedge CLK) ULED[2] <= ~RXD; // UART RXD
112    always @(posedge CLK) ULED[1] <= ~calib_done; // DRAM calibration done
113    always @(posedge CLK) ULED[0] <= ~INIT_DONE; // MEMORY IMAGE transfer is done
114
115    /*****/
116    wire tx_ready, txd_w;
117    always @(posedge CLK) TXD <= txd_w;
118    UartTx send(CLK, RST_X, CORE_D_OUT[7:0], (CORE_D_WE!=0 & CORE_D_ADDR==0), txd_w, tx_ready);
119 endmodule
120 /*****/

```

Nov 12, 13 0:49

code.txt

Page 5/17

```

file name: clock.v
1  /*****
2  /* The 1st IPSJ SIG-ARC High-Performance Processor Design Contest          Version 2013-09 */
3  /* From MieruPC 2010: Top Level Module                                ArchLab. TOKYO TECH */
4  /*****/
5  `include "../rtl/define.v"
6  `default_nettype none
7
8  /*****/
9  module clockgen(CLK_IN, CLK_OUT, LOCKED);
10     input  CLK_IN;
11     output CLK_OUT, LOCKED;
12
13     wire   CLK_OBUF, CLK_IBUF, CLK0, CLK0_OUT;
14
15     BUFG obuf (.I(CLK_OBUF), .O(CLK_OUT));
16     BUFG fbuf (.I(CLK0), .O(CLK0_OUT));
17     //IBUFG ibuf (.I(CLK_IN), .O(CLK_IBUF));
18     assign CLK_IBUF = CLK_IN;
19
20     DCM_SP dcm (.CLKIN(CLK_IBUF), .CLKFX(CLK_OBUF), .CLK0(CLK0), .CLKFB(CLK0_OUT),
21               .LOCKED(LOCKED), .RST(1'b0), .DSSEN(1'b0), .PSCLK(1'b0),
22               .PSEN(1'b0), .PSINDEC(1'b0));
23     defparam dcm.CLKFX_DIVIDE = 'DCM_CLKFX_DIVIDE;
24     defparam dcm.CLKFX_MULTIPLY = 'DCM_CLKFX_MULTIPLY;
25     defparam dcm.CLKIN_PERIOD = 10.000; // 100MHz input clock
26 endmodule
27
28 /*****/
29 module clockgen2(CLK_IN, CLK_OUT, LOCKED);
30     input  CLK_IN;
31     output CLK_OUT, LOCKED;
32
33     wire   CLK_OBUF, CLK_IBUF, CLK0, CLK0_OUT;
34
35     BUFG obuf (.I(CLK_OBUF), .O(CLK_OUT));
36     BUFG fbuf (.I(CLK0), .O(CLK0_OUT));
37     //IBUFG ibuf (.I(CLK_IN), .O(CLK_IBUF));
38     assign CLK_IBUF = CLK_IN;
39
40     DCM_SP dcm (.CLKIN(CLK_IBUF), .CLKFX(CLK_OBUF), .CLK0(CLK0), .CLKFB(CLK0_OUT),
41               .LOCKED(LOCKED), .RST(1'b0), .DSSEN(1'b0), .PSCLK(1'b0),
42               .PSEN(1'b0), .PSINDEC(1'b0));
43     defparam dcm.CLKFX_DIVIDE = 'DCM_DRAMC_DIVIDE;
44     defparam dcm.CLKFX_MULTIPLY = 'DCM_DRAMC_MULTIPLY;
45     defparam dcm.CLKIN_PERIOD = 10.000; // 100MHz input clock
46 endmodule
47
48 /* Reset Generator : generate about 1000 cycle reset signal */
49 /*****/
50 module resetgen(CLK, RST_X_I, RST_X_O);
51     input  CLK, RST_X_I;
52     output RST_X_O;
53
54     reg [11:0] cnt;
55     assign RST_X_O = cnt[11];
56
57     always @(posedge CLK or negedge RST_X_I) begin
58         if (!RST_X_I) cnt <= 0;
59         else if (~RST_X_O) cnt <= cnt + 1;
60     end
61 endmodule
62 /*****/

```

Nov 12, 13 0:49

code.txt

Page 6/17

```

file name: system.v
1  /*****
2  /* The 1st IPSJ SIG-ARC High-Performance Processor Design Contest          Version 2013-09 */
3  /* From MieruPC 2010: Top Level Module                                ArchLab. TOKYO TECH */
4  /*****/
5  `include "../rtl/define.v"
6  `default_nettype none
7
8  /* Program Loader: Initialize the main memory, copy memory image to the main memory */
9  /*****/
10 module PLOADER (CLK, RST_X, RXD, ADDR, DATA, WE, DONE);
11     input  CLK, RST_X, RXD;
12     output reg [31:0] ADDR;
13     output reg [31:0] DATA;
14     output reg WE;
15     output reg DONE; // program load is done
16
17     reg [31:0] waddr; // memory write address
18
19     wire SER_EN;
20     wire [7:0] SER_DATA;
21     serialc serc (CLK, RST_X, RXD, SER_DATA, SER_EN);
22
23     always @(posedge CLK or negedge RST_X) begin
24         if(!RST_X) begin
25             {ADDR, DATA, WE, waddr, DONE} <= 0;
26         end else begin
27             if(DONE==0 && SER_EN) begin
28                 ADDR <= waddr;
29                 //ADDR <= (waddr<32'h40000) ? waddr : {8'h04, 6'd0, waddr[17:0]};
30                 DATA <= {SER_DATA, DATA[31:8]};
31                 WE <= (waddr[1:0]==3);
32                 waddr <= waddr + 1;
33             end else begin
34                 WE <= 0;
35                 if(waddr>='PROG_SIZE) DONE <= 1;
36             end
37         end
38     end
39 endmodule
40
41 /*****/
42 module UartTx(CLK, RST_X, DATA, WE, TXD, READY);
43     input  CLK, RST_X, WE;
44     input [7:0] DATA;
45     output reg TXD, READY;
46
47     reg [8:0] cmd;
48     reg [11:0] waitnum;
49     reg [3:0] cnt;
50
51     always @(posedge CLK or negedge RST_X) begin
52         if(!RST_X) begin
53             TXD <= 1'b1;
54             READY <= 1'b1;
55             cmd <= 9'hfff;
56             waitnum <= 0;
57             cnt <= 0;
58         end else if( READY ) begin
59             TXD <= 1'b1;
60             waitnum <= 0;
61             if( WE )begin
62                 READY <= 1'b0;
63                 cmd <= {DATA, 1'b0};
64                 cnt <= 10;
65             end
66         end else if( waitnum >= 'SERIAL_WCNT ) begin
67             TXD <= cmd[0];
68             READY <= (cnt == 1);
69             cmd <= {1'b1, cmd[8:1]};
70             waitnum <= 1;
71             cnt <= cnt - 1;
72         end else begin
73             waitnum <= waitnum + 1;
74         end
75     end
76 endmodule
77
78 /*****/
79 /* RS232C serial controller (deserializer): */
80 /*****/
81 `define SS_SER_WAIT 'd0 // RS232C deserializer, State WAIT
82 `define SS_SER_RCV0 'd1 // RS232C deserializer, State Receive 0th bit
83 // States Receive 1st bit to 7th bit are not used
84 `define SS_SER_DONE 'd9 // RS232C deserializer, State DONE
85 /*****/
86 module serialc(CLK, RST_X, RXD, DATA, EN);
87     input  CLK, RST_X, RXD; // clock, reset, RS232C input
88     output [7:0] DATA; // 8bit output data
89     output reg EN; // 8bit output data enable

```

Nov 12, 13 0:49

code.txt

Page 7/17

```

90
91     reg    [7:0]   DATA;
92     reg    [3:0]   stage;
93     reg    [12:0]  cnt;          // counter to latch D0, D1, ..., D7
94     reg    [11:0]  cnt_start;    // counter to detect the Start Bit
95     wire   [12:0]  waitcnt;
96
97     assign waitcnt = 'SERIAL_WCNT;
98
99     always @(posedge CLK or negedge RST_X)
100         if (!RST_X) cnt_start <= 0;
101         else cnt_start <= (RXD) ? 0 : cnt_start + 1;
102
103     always @(posedge CLK or negedge RST_X)
104         if (!RST_X) begin
105             EN <= 0;
106             stage <= 'SS_SER_WAIT;
107             cnt <= 1;
108             DATA <= 0;
109         end else if (stage == 'SS_SER_WAIT) begin // detect the Start Bit
110             EN <= 0;
111             stage <= (cnt_start == (waitcnt >> 1)) ? 'SS_SER_RCV0 : stage;
112         end else begin
113             if (cnt != waitcnt) begin
114                 cnt <= cnt + 1;
115                 EN <= 0;
116             end else begin // receive 1bit data
117                 stage <= (stage == 'SS_SER_DONE) ? 'SS_SER_WAIT : stage + 1;
118                 EN <= (stage == 8) ? 1 : 0;
119                 DATA <= {RXD, DATA[7:1]};
120                 cnt <= 1;
121             end
122         end
123     endmodule
124     /*****

```

Nov 12, 13 0:49

code.txt

Page 8/17

```

file name: memory.v
1  /****
2  /* MieruPC 2010: Main Memory (SRAM Controller) ArchLab. TOKYO TECH */
3  /* See the README file on the parent directory for license information.
4  /****
5  'include "../rtl/define.v"
6  'default_nettype none
7
8  /****
9  module IMEM(CLK, RST_X, A_IN, D_IN, WE_IN, INST_ADDR, INST_OUT);
10     input    CLK, RST_X;
11     input    [31:0] A_IN;
12     input    [31:0] D_IN;
13     input    WE_IN;
14     input    [31:0] INST_ADDR;
15     output   [31:0] INST_OUT;
16
17     reg [31:0] DATA, waddr;
18     reg        WE;
19
20     wire [31:0] a = (waddr < 'IMEM_SIZE/4) ? A_IN : INST_ADDR;
21     IMEM_CORE(CLK, WE, a, DATA, INST_OUT);
22
23     always @(posedge CLK or negedge RST_X) begin
24         if (!RST_X) begin
25             {DATA, WE, waddr} <= 0;
26         end else begin
27             if (WE_IN) begin
28                 WE <= (waddr < 'IMEM_SIZE/4) ? 1 : 0; // note
29                 DATA <= D_IN;
30                 waddr <= waddr + 1;
31             end else begin
32                 WE <= 0;
33             end
34         end
35     end
36 endmodule
37
38 /****
39 module IMEM_CORE(CLK, WE_IN, A_IN, D_IN, D_OUT);
40     input    CLK, WE_IN;
41     input    [31:0] A_IN, D_IN;
42     output   reg [31:0] D_OUT;
43
44     reg [31:0] mem['IMEM_SIZE/4-1:0];
45
46     always @(negedge CLK) begin // use negedge CLK
47         if (WE_IN) mem[A_IN[15:2]] <= D_IN;
48         D_OUT <= mem[A_IN[15:2]];
49     end
50 endmodule
51 /****

```

Nov 12, 13 0:49

code.txt

Page 9/17

```

file name: MipsCore.v
1  /*****
2  /* The 1st IPSJ SIG-ARC High-Performance Processor Design Contest          Version 2013-10 */
3  /* from MipsCore pipeline95 2013-10-13                                  ArchLab. TOKYO TECH */
4  /*****
5  'include "../rtl/define.v"
6  'default_nettype none
7
8  /*****
9  /* 32bit-32cycle divider (signed or unsigned), Just for Pipelining Version */
10 /*****
11 module DIVUNIT(CLK, RST_X, INIT, SIGNED, A, B, RSLT, BUSY);
12     input          CLK, RST_X, INIT, SIGNED;
13     input [31:0]   A, B;
14     output [63:0]  RSLT;
15     output         BUSY;
16
17     wire [31:0]    uint_a = (SIGNED & A[31]) ? ~A + 1 : A;
18     wire [31:0]    uint_b = (SIGNED & B[31]) ? ~B + 1 : B;
19     wire [63:0]    uint_rslt;
20     reg            sign, sign_a, sign_b;
21
22     reg            init_t;
23     always @(posedge CLK) init_t <= INIT;
24
25     DIVUNITCORE divcore(.CLK(CLK), .RST_X(RST_X), .INIT(INIT), .A(uint_a), .B(uint_b),
26                         .RSLT(uint_rslt), .BUSY(BUSY));
27
28     assign RSLT[63:32] = (~sign) ?          uint_rslt[63:32] :
29                         ((sign_a, sign_b) == 2'b00) ? uint_rslt[63:32] :
30                         ((sign_a, sign_b) == 2'b01) ?          :
31                         ((sign_a, sign_b) == 2'b10) ? ~uint_rslt[63:32] + 1 : ~uint_rslt[63:32] + 1;
32
33     assign RSLT[31: 0] = (~sign) ?          uint_rslt[31: 0] :
34                         ((sign_a, sign_b) == 2'b00) ? uint_rslt[31: 0] :
35                         ((sign_a, sign_b) == 2'b01) ? ~uint_rslt[31: 0] + 1 :
36                         ((sign_a, sign_b) == 2'b10) ? ~uint_rslt[31: 0] + 1 : uint_rslt[31: 0];
37
38     always @(posedge CLK or negedge RST_X)
39     if(!RST_X) (sign, sign_a, sign_b) <= 0;
40     else if (init_t==1 && INIT==0) begin
41         sign <= SIGNED;
42         sign_a <= A[31];
43         sign_b <= B[31];
44     end
45 endmodule
46
47 /*****
48 module DIVUNITCORE(CLK, RST_X, INIT, A, B, RSLT, BUSY);
49     input          CLK, RST_X, INIT;
50     input [31:0]   A, B;
51     output reg [63:0] RSLT;
52     output         BUSY;
53
54     reg [31:0]     divisor;
55     reg [5:0]      count;
56     wire [32:0]    differ = RSLT[63:31] - {1'b0, divisor};
57
58     reg            init_t;
59     always @(posedge CLK) init_t <= INIT;
60
61     assign BUSY = count!=0 || (init_t==1 && INIT==0);
62
63     always @(posedge CLK or negedge RST_X) begin
64         if(!RST_X) {divisor, count, RSLT} <= 0;
65         else begin
66             if(init_t==1 && INIT==0) begin
67                 divisor <= B;
68                 RSLT <= {32'h0, A};
69                 count <= 1;
70             end else if(count!=0 && count<33) begin
71                 divisor <= divisor;
72                 RSLT <= (differ[32]) ? {RSLT[62:0], 1'h0} : {differ[31:0], RSLT[30:0], 1'h1};
73                 count <= count + 1;
74             end else begin
75                 count <= 0;
76             end
77         end
78     end
79 endmodule
80
81 /*****
82 /* MipsCore: 32bit-MIPS 5-stage pipelining processor
83 /*****
84 module MIPS_CORE(CLK, RST_X, STALL, STALL_I_ADDR, I_IN, D_ADDR, D_IN, D_OUT, D_OE, D_WE, IO_IN);
85     input          CLK, RST_X, STALL, I_ADDR, I_IN, D_ADDR, D_IN, D_OUT, D_OE, D_WE, IO_IN;
86     output reg [2:0] STALL;
87     output [I_ADDR] I_ADDR, D_ADDR;
88     input [31:0]    I_IN, D_IN, IO_IN;
89     output [31:0]   D_OUT;

```

Nov 12, 13 0:49

code.txt

Page 10/17

```

90     output [3:0]    D_WE;
91     output          D_OE;
92
93     reg [I_ADDR]    pc;          // program counter
94     reg [31:0]      hi, lo;      // high and low register for multiply and divide inst.
95
96     assign I_ADDR = pc;
97
98     /*****
99     reg [I_ADDR]    IfId_npc;      // IF-ID pipeline reg: next pc, pc + 4
100    reg [31:0]      IfId_ir;       // IF-ID pipeline reg: instruction
101
102    reg [I_ADDR]    IdEx_npc;      // ID-EX pipeline reg: next pc, pc + 4
103    reg [31:0]      IdEx_rrs;      // ID-EX pipeline reg: fetched operand of R[rs]
104    reg [31:0]      IdEx_rrt;      // ID-EX pipeline reg: fetched operand of R[rt]
105    reg [4:0]       IdEx_dst;      // ID-EX pipeline reg: decoded destination reg number
106    reg [31:0]      IdEx_ir;       // ID-EX pipeline reg: instruction
107    reg ['OPNT]     IdEx_opn;      // ID-EX pipeline reg: decoded operation ID
108    reg ['ATTR]     IdEx_attr;     // ID-EX pipeline reg: decoded instruction attribute
109
110    reg [4:0]       ExMa_dst;      // EX-MA pipeline reg: decoded destination reg number
111    reg [31:0]      ExMa_rslt;     // EX-MA pipeline reg: execution result
112    reg [3:0]       ExMa_mwe;      // EX-MA pipeline reg: mem write enable
113    reg             ExMa_oe;       // EX-MA pipeline reg: data memory output enable
114    reg [4:0]       ExMa_lds;      // EX-MA pipeline reg: load selector
115    reg [31:0]      ExMa_std;      // EX-MA pipeline reg: store data
116
117    reg [4:0]       MaWb_dst;      // MA-WB pipeline reg: decoded destination reg number
118    reg [31:0]      MaWb_rslt;     // MA-WB pipeline reg: execution result
119
120    /*****
121    reg [I_ADDR]    IdTPC;         // wire, calculated branch Taken address (Taken Program Counter)
122    reg            IdC;            // wire, Calculated branch result, branch Taken or Untaken
123    reg            IdB;            // wire, branch/jump instruction ?
124    wire [31:0]     MaRSLT;        // wire, execution result on MA stage
125    wire            bstall;        // stall signal by branch instruction
126    wire            DIVBUSY;
127    wire            PSTALL = (STALL || DIVBUSY); // pipeline stall
128
129    /*****
130    /* Stage 1 : IF, instruction fetch
131    /*****
132    wire [I_ADDR]   IFNPC = pc + 4;
133
134    always @(posedge CLK or negedge RST_X) begin ///// update program counter
135        if(!RST_X) pc <= 0;
136        else if(!PSTALL && !bstall) pc <= (IdC) ? IdTPC : IFNPC; // If branch taken, uses taken PC
137    end
138
139    always @(posedge CLK or negedge RST_X) begin ///// update pipeline registers
140        if(!RST_X) {IfId_npc, IfId_ir} <= 0;
141        else if(!PSTALL && !bstall) begin
142            IfId_npc <= IFNPC;
143            IfId_ir <= I_IN; // if branch taken then NOP else instruction from imem.
144        end
145    end
146
147    /*****
148    /* Stage 2 : ID, instruction decode & operand fetch
149    /*****
150    wire [5:0] IdOP = IfId_ir[31:26]; // opcode field of instruction
151    wire [4:0] IdRS = IfId_ir[25:21]; // rs field of instruction
152    wire [4:0] IdRT = IfId_ir[20:16]; // rt field of instruction
153    wire [4:0] IdRD = IfId_ir[15:11]; // rd field of instruction
154    wire [5:0] IdFCT = IfId_ir[5:0]; // funct field of instruction
155    wire [31:0] IdRRS, IdRRT; // register value of rs and rt
156
157    assign bstall = IdB &&
158                ((IdRS!=0 && (IdRS==IdEx_dst || IdRS==ExMa_dst)) ||
159                 (IdRT!=0 && (IdRT==IdEx_dst || IdRT==ExMa_dst))); // branch stall
160
161    GPR gpr(.CLK(CLK), .REGNUM0(IdRS), .REGNUM1(IdRT), .DOUT0(IdRRS), .DOUT1(IdRRT), // register file
162            .REGNUM2(ExMa_dst), .DINO(MaRSLT), .WEO(!PSTALL));
163
164    reg [6:0] IdOPN; // instruction operation number (unique operation ID)
165    reg [4:0] IdDST; // destination register
166    reg ['ATTR] IdATTR; // instruction attribute
167    always @(IfId_ir) begin
168        IdOPN = 'ERROR____;
169        IdDST = 0;
170        IdATTR = 0;
171        case (IdOP) // OP
172            6'h00: case (IdFCT) // FUNCT
173                6'h00: begin IdOPN= (IdRD) ? 'SLL : 'NOP; IdDST=IdRD; end
174                6'h02: begin IdOPN='SRL; IdDST=IdRD; end
175                6'h03: begin IdOPN='SRA; IdDST=IdRD; end
176                6'h04: begin IdOPN='SLLV; IdDST=IdRD; end
177                6'h06: begin IdOPN='SRLV; IdDST=IdRD; end
178                6'h07: begin IdOPN='SRV; IdDST=IdRD; end
179                6'h08: begin IdOPN='JR; IdDST=0; end

```

Nov 12, 13 0:49

code.txt

Page 11/17

```

180      6'h09: begin IdOPN='JALR____; IdDST=31; end
181      6'h10: begin IdOPN='MFHI____; IdDST=IdRD; end
182      6'h12: begin IdOPN='MFLO____; IdDST=IdRD; end
183      6'h18: begin IdOPN='MULT____; IdDST=0; end
184      6'h19: begin IdOPN='MULTU____; IdDST=0; end
185      6'h20: begin IdOPN='ADD____; IdDST=IdRD; end
186      6'h21: begin IdOPN='ADDU____; IdDST=IdRD; end
187      6'h22: begin IdOPN='SUB____; IdDST=IdRD; end
188      6'h23: begin IdOPN='SUBU____; IdDST=IdRD; end
189      6'h24: begin IdOPN='AND____; IdDST=IdRD; end
190      6'h25: begin IdOPN='OR____; IdDST=IdRD; end
191      6'h26: begin IdOPN='XOR____; IdDST=IdRD; end
192      6'h27: begin IdOPN='NOR____; IdDST=IdRD; end
193      6'h2a: begin IdOPN='SLT____; IdDST=IdRD; end
194      6'h2b: begin IdOPN='SLTU____; IdDST=IdRD; end
195      6'h1a: begin IdOPN='DIV____; IdDST=IdRD; end
196      6'h1b: begin IdOPN='DIVU____; IdDST=IdRD; end
197  endcase
198  6'h01: case (IdRT)
199      5'h00: begin IdOPN='BLTZ____; IdDST=0; end
200      5'h01: begin IdOPN='BGEZ____; IdDST=0; end
201  endcase
202  6'h02: begin IdOPN='J____; IdDST=0; end
203  6'h03: begin IdOPN='JAL____; IdDST=31; end
204  6'h04: begin IdOPN='BEQ____; IdDST=0; end
205  6'h05: begin IdOPN='BNE____; IdDST=0; end
206  6'h06: begin IdOPN='BLEZ____; IdDST=0; end
207  6'h07: begin IdOPN='BGZT____; IdDST=0; end
208  6'h08: begin IdOPN='ADDI____; IdDST=IdRT; end
209  6'h09: begin IdOPN='ADDIU____; IdDST=IdRT; end
210  6'h0a: begin IdOPN='SLTI____; IdDST=IdRT; end
211  6'h0b: begin IdOPN='SLTIU____; IdDST=IdRT; end
212  6'h0c: begin IdOPN='ANDI____; IdDST=IdRT; end
213  6'h0d: begin IdOPN='ORI____; IdDST=IdRT; end
214  6'h0e: begin IdOPN='XORI____; IdDST=IdRT; end
215  6'h0f: begin IdOPN='LUI____; IdDST=IdRT; end
216  6'h23: begin IdOPN='LW____; IdDST=IdRT; IdATTR='LD_4B; end
217  6'h2b: begin IdOPN='SW____; IdDST=0; IdATTR='ST_4B; end
218  endcase
219  end
220
221  wire ['ADDR] IdBPC = IfId_npc + ({16{IfId_ir[15]}}, IfId_ir[15:0]) << 2);
222  always @(*) begin // branch & jump resolution unit
223      {IdTPC, IdC, IdB} = 0;
224      case (IdOP)
225          6'h04: begin IdB=1; IdTPC = IdBPC; IdC = (IdRRS == IdRRT); end // BEQ
226          6'h05: begin IdB=1; IdTPC = IdBPC; IdC = (IdRRS != IdRRT); end // BNE
227          6'h06: begin IdB=1; IdTPC = IdBPC; IdC = (IdRRS[31] || (IdRRS==0)); end // BLEZ
228          6'h07: begin IdB=1; IdTPC = IdBPC; IdC = (~IdRRS[31] && (IdRRS!=0)); end // BGZT
229          6'h01: begin IdB=1; IdTPC = IdBPC; IdC = (IdRT) ? ~IdRRS[31] : IdRRS[31]; end // BGEZ,BLTZ
230          6'h02: begin IdB=1; IdTPC = {IfId_npc[31:28], IfId_ir[25:0], 2'b0}; IdC=1; end // J
231          6'h03: begin IdB=1; IdTPC = {IfId_npc[31:28], IfId_ir[25:0], 2'b0}; IdC=1; end // JAL
232          6'h00: if (IdFCT==6'h08) begin IdB=1; IdTPC = IdRRS; IdC = 1; end // JR
233          else if (IdFCT==6'h09) begin IdB=1; IdTPC = IdRRS; IdC = 1; end // JALR
234      endcase
235  end
236
237  always @(posedge CLK or negedge RST_X) begin // update pipeline registers
238      if(!RST_X) {IdEx_npc, IdEx_rrs, IdEx_rrt, IdEx_dst, IdEx_ir, IdEx_opn, IdEx_attr} <= 0;
239      else if(!PSTALL) begin
240          IdEx_npc <= (bstall) ? 0 : IfId_npc;
241          IdEx_rrs <= (bstall) ? 0 : IdRRS; // data from general-purpose register file
242          IdEx_rrt <= (bstall) ? 0 : IdRRT; // data from general-purpose register file
243          IdEx_dst <= (bstall) ? 0 : IdDST;
244          IdEx_ir <= (bstall) ? 0 : IfId_ir;
245          IdEx_opn <= (bstall) ? 0 : IdOPN;
246          IdEx_attr <= (bstall) ? 0 : IdATTR;
247      end
248  end
249
250  /*
251  /* Stage 3 : EX, execute & address generation for LD/ST
252  /*
253  wire [31:0] RRS_U, RRT_U; // output of data forwarding unit
254  wire signed [31:0] RRS_S = RRS_U; // signed wire
255  wire signed [31:0] RRT_S = RRT_U; // signed wire
256  wire [4:0] SHAMT = IdEx_ir[10:6]; // Shift amount
257  wire [15:0] IMM = IdEx_ir[15:0]; // Immediate
258  wire [31:0] SET32I = {{16{IMM[15]}}, IMM}; // Sign Extended Imm.
259  wire ['ADDR] SETADI = SET32I['ADDR] << 2; // shifted immediate of address bit width
260  wire ['ADDR] JADDR = IdEx_ir['ADDR] << 2; // Juma address
261  wire ['ADDR] ExA = RRS_U['ADDR] + SET32I['ADDR]; // mem address of LD/ST
262  reg [31:0] ExRSLT; // execution result
263  reg [3:0] ExWE; // memory write enable vector
264
265  wire DIVINIT = IdOPN=='DIV____ || IdOPN=='DIVU____;
266  wire EXSIGNED = IdEx_opn=='DIV____;
267  wire [63:0] DURSLT;
268
269  DIVUNIT divu(.CLK(CLK), .RST_X(RST_X), .INIT(DIVINIT), .SIGNED(EXSIGNED),

```

Nov 12, 13 0:49

code.txt

Page 12/17

```

270      .A(RRS_U), .B(RRT_U), .RSLT(DURSLT), .BUSY(DIVBUSY));
271
272  FORWARDING frs(.SRC(IdEx_ir[25:21]), .DIN0(IdEx_rrs), .DOUT(RRS_U),
273      .DST1(ExMa_dst), .DIN1(ExMa_rslt), .DST2(MaWb_dst), .DIN2(MaWb_rslt));
274  FORWARDING frt(.SRC(IdEx_ir[20:16]), .DIN0(IdEx_rrt), .DOUT(RRT_U),
275      .DST1(ExMa_dst), .DIN1(ExMa_rslt), .DST2(MaWb_dst), .DIN2(MaWb_rslt));
276
277  always @(*) begin
278      {ExRSLT, ExWE} = 0;
279      case ( IdEx_opn )
280          'ADD : begin ExRSLT = RRS_U + RRT_U; end
281          'ADDI : begin ExRSLT = RRS_U + SET32I; end
282          'ADDIU : begin ExRSLT = RRS_U + SET32I; end
283          'ADDU : begin ExRSLT = RRS_U + RRT_U; end
284          'SUB : begin ExRSLT = RRS_U - RRT_U; end
285          'SUBU : begin ExRSLT = RRS_U - RRT_U; end
286          'AND : begin ExRSLT = RRS_U & RRT_U; end
287          'ANDI : begin ExRSLT = RRS_U & {16'h0, IMM}; end
288          'NOR : begin ExRSLT = ~(RRS_U | RRT_U); end
289          'OR : begin ExRSLT = RRS_U | RRT_U; end
290          'ORI : begin ExRSLT = RRS_U | {16'h0, IMM}; end
291          'XOR : begin ExRSLT = RRS_U ^ RRT_U; end
292          'XORI : begin ExRSLT = RRS_U ^ {16'h0, IMM}; end
293          'SLL : begin ExRSLT = RRT_U << SHAMT; end
294          'SRL : begin ExRSLT = RRT_U >> SHAMT; end
295          'SRA : begin ExRSLT = RRT_S >> SHAMT; end
296          'SLLV : begin ExRSLT = RRT_U << RRS_U[4:0]; end
297          'SRLV : begin ExRSLT = RRT_U >> RRS_U[4:0]; end
298          'SRAV : begin ExRSLT = RRT_S >> RRS_U[4:0]; end
299          'SLT : begin ExRSLT = (RRS_U[31] ^ RRT_U[31]) ? RRS_U[31] : (RRS_U < RRT_U); end
300          'SLTI : begin ExRSLT = (RRS_U[31] ^ IMM[15]) ? RRS_U[31] : (RRS_U < SET32I); end
301          'SLTIU : begin ExRSLT = (RRS_U < SET32I); end
302          'SLTU : begin ExRSLT = (RRS_U < RRT_U); end
303          'JAL : begin ExRSLT = IdEx_npc + 4; end
304          'JALR : begin ExRSLT = IdEx_npc + 4; end
305          'LUI : begin ExRSLT = {IMM, 16'h0}; end
306          'LW : begin ExRSLT = {ExA[31:21], 2'b00}; end
307          'SW : begin ExRSLT = {ExA[31:21], 2'b00}; ExWE = 4'b1111; end
308          'MFHI : begin ExRSLT = hi; end
309          'MFLO : begin ExRSLT = lo; end
310      endcase
311  end
312
313  always @( posedge CLK or negedge RST_X ) begin // update hi and lo register
314      if(!RST_X) {hi, lo} <= 0;
315      else if(!PSTALL) begin
316          if(IdEx_opn == 'MULT) {hi, lo} <= RRS_S * RRT_S;
317          if(IdEx_opn == 'MULTU) {hi, lo} <= RRS_U * RRT_U;
318          if(IdEx_opn == 'DIV) {hi, lo} <= (RRT_U) ? DURSLT : 0;
319          if(IdEx_opn == 'DIVU) {hi, lo} <= (RRT_U) ? DURSLT : 0;
320      end
321  end
322
323  always @( posedge CLK or negedge RST_X ) begin // update pipeline registers
324      if(!RST_X) {ExMa_rslt, ExMa_dst, ExMa_mwe, ExMa_oe, ExMa_lds, ExMa_std} <= 0;
325      else if(!PSTALL) begin
326          ExMa_rslt <= ExRSLT;
327          ExMa_dst <= IdEx_dst;
328          ExMa_std <= RRT_U;
329          ExMa_mwe <= ExWE;
330          ExMa_oe <= (IdEx_attr & 'LDST_ANY) ? 1 : 0;
331          ExMa_lds <= (IdEx_opn=='LW) ? 1 : 0;
332      end
333  end
334
335  /*
336  /* Stage 4 : MA, memory access for LD/ST
337  /*
338  assign D_ADDR = (ExMa_oe) ? ExMa_rslt : 0;
339  assign D_OUT = (ExMa_oe) ? ExMa_std : 0;
340  assign D_WE = (ExMa_oe && !PSTALL) ? ExMa_mwe : 0;
341  assign D_OE = ExMa_oe;
342
343  // wire [31:0] MaLD = (ExMa_rslt[23]) ? IO_IN : D_IN; // use I/O input if the high address
344  wire [31:0] MaLD = D_IN; // This edition does not use I/O.
345  wire [31:0] MaLDD = MaLD >> (8*ExMa_rslt[1:0]); // loaded data, align data here
346
347  assign MaRSLT = (ExMa_lds) ? MaLDD : ExMa_rslt;
348
349  always @( posedge CLK or negedge RST_X ) begin // update pipeline registers
350      if(!RST_X) {MaWb_rslt, MaWb_dst} <= 0;
351      else if(!PSTALL) begin
352          MaWb_rslt <= MaRSLT;
353          MaWb_dst <= ExMa_dst;
354      end
355  end
356
357  /* Update STATUS_LED (ERROR_LED)
358  /*
359  wire ['OPNT] ExOPN = IdEx_opn;

```

Nov 12, 13 0:49

code.txt

Page 13/17

```
360     always @(posedge CLK or negedge RST_X) begin
361         if(!RST_X) STAT <= 0;
362         else if(!PSTALL) begin
363             if(!OPN=="ERROR") STAT[0] <= 1; // instruction decode error!
364             if((!OPN=="LW" || ExOPN=="SW") && ExA[1:0] != 0))
365                 STAT[1] <= 1; // memory access alignment error!
366         end
367     end
368 endmodule
369
370 /***** 32bitx32 2R/1W General Purpose Registers (Register File) */
371 /***** 32bitx32 2R/1W General Purpose Registers (Register File) */
372 /***** 32bitx32 2R/1W General Purpose Registers (Register File) */
373 module GPR(CLK, REGNUM0, REGNUM1, REGNUM2, DIN0, WE0, DOUT0, DOUT1);
374     input        CLK;
375     input        [4:0] REGNUM0, REGNUM1, REGNUM2;
376     input        [31:0] DIN0;
377     input        WE0;
378     output reg [31:0] DOUT0, DOUT1;
379
380     reg [31:0] r[0:31];
381
382     always @(negedge CLK) DOUT0 <= (REGNUM0==0) ? 0 : r[REGNUM0];
383     always @(negedge CLK) DOUT1 <= (REGNUM1==0) ? 0 : r[REGNUM1];
384     always @(posedge CLK) if(WE0) r[REGNUM2] <= DIN0;
385 endmodule
386
387 /**** data forwarding unit ****/
388 /**** data forwarding unit ****/
389 module FORWARDING(SRC, DIN0, DST1, DIN1, DST2, DIN2, DOUT);
390     input  [4:0] SRC, DST1, DST2; // register number
391     input  [31:0] DIN0, DIN1, DIN2; // 32bit value
392     output [31:0] DOUT; // 32bit data output
393
394     assign DOUT = (SRC!=0 && DST1==SRC) ? DIN1 : (SRC!=0 && DST2==SRC) ? DIN2 : DIN0;
395 endmodule
396
397 /***** 32bitx32 2R/1W General Purpose Registers (Register File) *****/
```

Nov 12, 13 0:49

code.txt

Page 14/17

```
file name: dram_mem.v
1  /***** 32bitx32 2R/1W General Purpose Registers (Register File) *****/
2  /* The 1st IPSJ SIG-ARC High-Performance Processor Design Contest          Version 2013-09 */
3  /* From MieruPC 2010: Top Level Module                                     ArchLab. TOKYO TECH */
4  /***** 32bitx32 2R/1W General Purpose Registers (Register File) *****/
5  'default_nettype none
6
7  /***** 32bitx32 2R/1W General Purpose Registers (Register File) *****/
8  'define DRAM_CMD_WRITE      3'b000
9  'define DRAM_CMD_WRITE_AP   3'b010
10 'define DRAM_CMD_READ       3'b001
11 'define DRAM_CMD_READ_AP    3'b011
12 /***** 32bitx32 2R/1W General Purpose Registers (Register File) *****/
13 module dram_con(CLK, DRAM_CLK, RST_X, calib_done,
14     D_ADR, D_DIN, D_DOUT, D_WE, D_RE, D_BUSY,
15     DDR2CLK0, DDR2CLK1, DDR2CKE, DDR2RASN, DDR2CASN, DDR2WEN, DDR2RZQ, DDR2ZIO,
16     DDR2BA, DDR2A, DDR2DQ,
17     DDR2UDQS, DDR2UDQSN, DDR2LDQS, DDR2LDM, DDR2UDM, DDR2ODT,
18     DDR2RZM);
19
20     input CLK, DRAM_CLK, RST_X;
21     output calib_done;
22     output DDR2CLK0, DDR2CLK1, DDR2CKE, DDR2RASN, DDR2CASN, DDR2WEN, DDR2RZQ, DDR2ZIO, DDR2RZM;
23     output DDR2LDM, DDR2UDM, DDR2ODT;
24     output [2:0] DDR2BA;
25     output [12:0] DDR2A;
26     inout  [15:0] DDR2DQ;
27     inout  DDR2UDQS, DDR2UDQSN, DDR2LDQS, DDR2LDM;
28
29     input [31:0] D_ADR, D_DIN;
30     input        D_WE, D_RE;
31     output reg [31:0] D_DOUT;
32     output        D_BUSY;
33
34     // port 0 //////////////////////////////////////
35     wire c3_p0_cmd_empty, c3_p0_cmd_full;
36     reg [31:0] c3_p0_cmd_byte_addr;
37     reg c3_p0_cmd_en; // command enqueue
38     reg [2:0] c3_p0_cmd_inst; // command
39     wire c3_p0_wr_full, c3_p0_wr_empty;
40     wire [6:0] c3_p0_wr_count;
41     wire c3_p0_wr_underrun, c3_p0_wr_error; // not enough data in write FIFO, or FIFO sync miss
42     reg c3_p0_wr_en;
43     reg [31:0] c3_p0_wr_data;
44     wire [31:0] c3_p0_rd_data; // head of read fifo
45     wire c3_p0_rd_empty;
46     wire c3_p0_rd_full;
47     wire c3_p0_rd_overflow, c3_p0_rd_error;
48     wire [6:0] c3_p0_rd_count; // read fifo entry
49
50     //----- Begin Cut here for INSTANTIATION Template ---// INST_TAG
51     dram # (
52         .C3_P0_MASK_SIZE(4),
53         .C3_P0_DATA_PORT_SIZE(32),
54         .C3_P1_MASK_SIZE(4),
55         .C3_P1_DATA_PORT_SIZE(32),
56         .DEBUG_EN(0),
57         .C3_MEMCLK_PERIOD(3000),
58         .C3_CALIB_SOFT_IP("TRUE"),
59         .C3_SIMULATION("FALSE"),
60         .C3_RST_ACT_LOW(0),
61         .C3_INPUT_CLK_TYPE("SINGLE_ENDED"),
62         .C3_MEM_ADDR_ORDER("ROW_BANK_COLUMN"),
63         .C3_NUM_DQ_PINS(16),
64         .C3_MEM_ADDR_WIDTH(13),
65         .C3_MEM_BANKADDR_WIDTH(3)
66     )
67     u_dram (
68         .c3_sys_clk (DRAM_CLK),
69         .c3_sys_rst_i (0), // (~RST_X),
70
71         .mcb3_dram_dq (DDR2DQ),
72         .mcb3_dram_a (DDR2A),
73         .mcb3_dram_ba (DDR2BA),
74         .mcb3_dram_ras_n (DDR2RASN),
75         .mcb3_dram_cas_n (DDR2CASN),
76         .mcb3_dram_we_n (DDR2WEN),
77         .mcb3_dram_odt (DDR2ODT),
78         .mcb3_dram_cke (DDR2CKE),
79         .mcb3_dram_ck (DDR2CLK0),
80         .mcb3_dram_ck_n (DDR2CLK1),
81         .mcb3_dram_dqs (DDR2LDQS),
82         .mcb3_dram_dqs_n (DDR2LDQSN),
83         .mcb3_dram_udqs (DDR2UDQS),
84         .mcb3_dram_udqs_n (DDR2UDQSN),
85         .mcb3_dram_udm (DDR2UDM),
86         .mcb3_dram_dm (DDR2LDM),
87         .c3_clk0 (),
88         .c3_rst0 (),
89         .c3_calib_done (calib_done),
```

Nov 12, 13 0:49

code.txt

Page 15/17

```

90 .mcb3_rzq          (DDR2RZQ),
91 .mcb3_zio          (DDR2ZIO),
92
93 /*****/
94 .c3_p0_cmd_clk      (CLK),
95 .c3_p0_cmd_en       (c3_p0_cmd_en),
96 .c3_p0_cmd_instr    (c3_p0_cmd_instr),
97 .c3_p0_cmd_bl       (0),
98 .c3_p0_cmd_byte_addr (c3_p0_cmd_byte_addr),
99 .c3_p0_cmd_empty     (c3_p0_cmd_empty),
100 .c3_p0_cmd_full     (c3_p0_cmd_full),
101
102 .c3_p0_wr_clk       (CLK),
103 .c3_p0_wr_en        (c3_p0_wr_en),
104 .c3_p0_wr_mask      (4'b0000),
105 .c3_p0_wr_data      (c3_p0_wr_data),
106 .c3_p0_wr_full      (c3_p0_wr_full),
107 .c3_p0_wr_empty     (c3_p0_wr_empty),
108 .c3_p0_wr_count     (c3_p0_wr_count),
109 .c3_p0_wr_underrun  (c3_p0_wr_underrun),
110 .c3_p0_wr_error     (c3_p0_wr_error),
111
112 .c3_p0_rd_clk       (CLK),
113 .c3_p0_rd_en        (1),
114 .c3_p0_rd_data      (c3_p0_rd_data),
115 .c3_p0_rd_full      (c3_p0_rd_full),
116 .c3_p0_rd_empty     (c3_p0_rd_empty),
117 .c3_p0_rd_count     (c3_p0_rd_count),
118 .c3_p0_rd_overflow  (c3_p0_rd_overflow),
119 .c3_p0_rd_error     (c3_p0_rd_error)
120 );
121
122 // INST_TAG_END ----- End INSTANTIATION Template -----
123
124 /*****/
125 parameter INIT      = 0; // INIT must be 0
126 parameter WAIT_REQ  = 1;
127 parameter WRITE_FIFO = 2;
128 parameter WRITE_CMD  = 3;
129 parameter READ_CMD   = 4;
130 parameter READ_FIFO  = 5;
131
132 ///// READ & WRITE PORT CONTROL //////////////////////////////////////
133 ///// negedge sensitive
134 assign D_BUSY = (state != WAIT_REQ);
135
136 reg [4:0] state;
137 always @(negedge CLK or negedge RST_X)begin
138     if(RST_X==0)begin
139         state      <= 0;
140         c3_p0_cmd_instr <= 0;
141         c3_p0_cmd_en   <= 0;
142         c3_p0_cmd_byte_addr <= 0;
143         c3_p0_wr_en    <= 0;
144         c3_p0_wr_data  <= 0;
145     end else begin
146         if(state==0)begin // initialize
147             if(calib_done) state <= WAIT_REQ;
148         end else if(state==WAIT_REQ) begin // wait request
149             c3_p0_cmd_en <= 0;
150             c3_p0_cmd_byte_addr <= D_ADR ;
151             c3_p0_wr_data <= D_DIN;
152
153             if(D_WE) state <= WRITE_FIFO;
154             else if(D_RE) state <= READ_CMD;
155         end
156         else if(state==WRITE_FIFO) begin
157             if(c3_p0_wr_empty) begin
158                 c3_p0_wr_en <= 1;
159                 state <= WRITE_CMD;
160             end
161         end else if(state==WRITE_CMD) begin
162             c3_p0_wr_en <= 0;
163             if(c3_p0_cmd_empty) begin
164                 c3_p0_cmd_instr <= 'DRAM_CMD_WRITE_AP;
165                 c3_p0_cmd_en <= 1;
166                 state <= WAIT_REQ;
167             end
168         end else if(state==READ_CMD) begin
169             if(c3_p0_cmd_empty && c3_p0_rd_empty) begin
170                 c3_p0_cmd_instr <= 'DRAM_CMD_READ_AP;
171                 c3_p0_cmd_en <= 1;
172                 state <= READ_FIFO;
173             end
174         end else if(state==READ_FIFO) begin
175             c3_p0_cmd_en <= 0;
176             if(!c3_p0_rd_empty && c3_p0_cmd_empty) begin
177                 D_DOUT <= c3_p0_rd_data;
178                 state <= WAIT_REQ;
179             end
180         end
181     end
182 end
183 endmodule
184 /*****/

```

Nov 12, 13 0:49

code.txt

Page 16/17

```

180         end
181     end
182 end
183 endmodule
184 /*****/

```


Nov 12, 13 0:49

code.txt

Page 17/17

```

file name: main.ucf
1 #####
2 # UCF file for ATLYS Board V0.1 2013-09-01 ArchLab. TOKYO TECH #
3 #####
4 NET CLK_IN LOC="L15" | PERIOD=100 MHz; #
5 NET RST_X_IN LOC="T15" ; # RESET
6 NET RXD LOC="A16" ; # P1, UART Serial RXD
7 NET TXD LOC="B16" ; # GPIO, UART Serial TXD
8 NET ULED<0> LOC="U18" ; # LD0 (LED, Light Emitting Diode 0)
9 NET ULED<1> LOC="M14" ; # LD1 (LED, Light Emitting Diode 1)
10 NET ULED<2> LOC="N14" ; # LD2 (LED, Light Emitting Diode 2)
11 NET ULED<3> LOC="L14" ; # LD3 (LED, Light Emitting Diode 3)
12 NET ULED<4> LOC="M13" ; # LD4 (LED, Light Emitting Diode 4)
13 NET ULED<5> LOC="D4" ; # LD5 (LED, Light Emitting Diode 5)
14 NET ULED<6> LOC="P16" ; # LD6 (LED, Light Emitting Diode 6)
15 NET ULED<7> LOC="N12" ; # LD7 (LED, Light Emitting Diode 7)
16
17 ## from AtlysGeneral.ucf
18 #####
19 # DDR2
20 NET "DDR2CLK0" LOC = "G3" | IOSTANDARD = DIFF_SSTL18_II;
21 NET "DDR2CLK1" LOC = "G1" | IOSTANDARD = DIFF_SSTL18_II;
22 NET "DDR2CKE" LOC = "H7" | IOSTANDARD = SSTL18_II;
23 NET "DDR2RASN" LOC = "L5" | IOSTANDARD = SSTL18_II;
24 NET "DDR2CASN" LOC = "K5" | IOSTANDARD = SSTL18_II;
25 NET "DDR2WEN" LOC = "E3" | IOSTANDARD = SSTL18_II;
26 NET "DDR2RZQ" LOC = "L6" | IOSTANDARD = SSTL18_II;
27 NET "DDR2ZIO" LOC = "C2" | IOSTANDARD = SSTL18_II;
28 #####
29 NET "DDR2BA<0>" LOC = "F2" | IOSTANDARD = SSTL18_II;
30 NET "DDR2BA<1>" LOC = "F1" | IOSTANDARD = SSTL18_II;
31 NET "DDR2BA<2>" LOC = "E1" | IOSTANDARD = SSTL18_II;
32
33 NET "DDR2A<0>" LOC = "J7" | IOSTANDARD = SSTL18_II;
34 NET "DDR2A<1>" LOC = "J6" | IOSTANDARD = SSTL18_II;
35 NET "DDR2A<2>" LOC = "H5" | IOSTANDARD = SSTL18_II;
36 NET "DDR2A<3>" LOC = "L7" | IOSTANDARD = SSTL18_II;
37 NET "DDR2A<4>" LOC = "F3" | IOSTANDARD = SSTL18_II;
38 NET "DDR2A<5>" LOC = "H4" | IOSTANDARD = SSTL18_II;
39 NET "DDR2A<6>" LOC = "H3" | IOSTANDARD = SSTL18_II;
40 NET "DDR2A<7>" LOC = "H6" | IOSTANDARD = SSTL18_II;
41 NET "DDR2A<8>" LOC = "D2" | IOSTANDARD = SSTL18_II;
42 NET "DDR2A<9>" LOC = "D1" | IOSTANDARD = SSTL18_II;
43 NET "DDR2A<10>" LOC = "F4" | IOSTANDARD = SSTL18_II;
44 NET "DDR2A<11>" LOC = "D3" | IOSTANDARD = SSTL18_II;
45 NET "DDR2A<12>" LOC = "G6" | IOSTANDARD = SSTL18_II;
46
47 NET "DDR2DQ<0>" LOC = "L2" | IOSTANDARD = SSTL18_II;
48 NET "DDR2DQ<1>" LOC = "L1" | IOSTANDARD = SSTL18_II;
49 NET "DDR2DQ<2>" LOC = "K2" | IOSTANDARD = SSTL18_II;
50 NET "DDR2DQ<3>" LOC = "K1" | IOSTANDARD = SSTL18_II;
51 NET "DDR2DQ<4>" LOC = "H2" | IOSTANDARD = SSTL18_II;
52 NET "DDR2DQ<5>" LOC = "H1" | IOSTANDARD = SSTL18_II;
53 NET "DDR2DQ<6>" LOC = "J3" | IOSTANDARD = SSTL18_II;
54 NET "DDR2DQ<7>" LOC = "J1" | IOSTANDARD = SSTL18_II;
55 NET "DDR2DQ<8>" LOC = "M3" | IOSTANDARD = SSTL18_II;
56 NET "DDR2DQ<9>" LOC = "M1" | IOSTANDARD = SSTL18_II;
57 NET "DDR2DQ<10>" LOC = "N2" | IOSTANDARD = SSTL18_II;
58 NET "DDR2DQ<11>" LOC = "N1" | IOSTANDARD = SSTL18_II;
59 NET "DDR2DQ<12>" LOC = "T2" | IOSTANDARD = SSTL18_II;
60 NET "DDR2DQ<13>" LOC = "T1" | IOSTANDARD = SSTL18_II;
61 NET "DDR2DQ<14>" LOC = "U2" | IOSTANDARD = SSTL18_II;
62 NET "DDR2DQ<15>" LOC = "U1" | IOSTANDARD = SSTL18_II;
63
64 NET "DDR2UDQS" LOC = "P2" | IOSTANDARD = DIFF_SSTL18_II;
65 NET "DDR2UDQSN" LOC = "P1" | IOSTANDARD = DIFF_SSTL18_II;
66 NET "DDR2LDQS" LOC = "L4" | IOSTANDARD = DIFF_SSTL18_II;
67 NET "DDR2LDQSN" LOC = "L3" | IOSTANDARD = DIFF_SSTL18_II;
68 NET "DDR2LDM" LOC = "K3" | IOSTANDARD = SSTL18_II;
69 NET "DDR2UDM" LOC = "K4" | IOSTANDARD = SSTL18_II;
70 NET "DDR2ODT" LOC = "K6" | IOSTANDARD = SSTL18_II;
71 #####

```