The George Washington University
School of Engineering and Applied Science
Department of Electrical and Computer Engineering
ECE 4150
Spring 2022

Final Project Report

TINYMIPS CPU PROJECT

Jaret Williams – Electrical Engineering
Nathan Pen – Electrical Engineering

Professor Jerry Wu
May 8, 2022

Contents

I. Introduction 4
II. Architecture 5
III. HDL Code 7
MIPS Module 8
Controller 9
Datapath 18
ALU 25
ALU Control 26
MIPS Memory 27
External Memory 28
Reg File 30
IV. Synthesis 32
Pre DFT-Synthesis Schematic 33
Post-DFT Synthesis Schematic 34
Synthesis Reports 37
V. Testing 38
Functional Testing 38
CPU Test Bench 39
Controller Test Bench 42
Datapath Test Bench 46
RAM Verilog File 51
RAM Data 53
DFT 55
VI. Layout 57
P&R 57
CCAR 58
DRC/LVS 63
VII. Enhancements 64
VIII. Problems/Debugging 64
IX. Conclusions 65
Appendix A 66

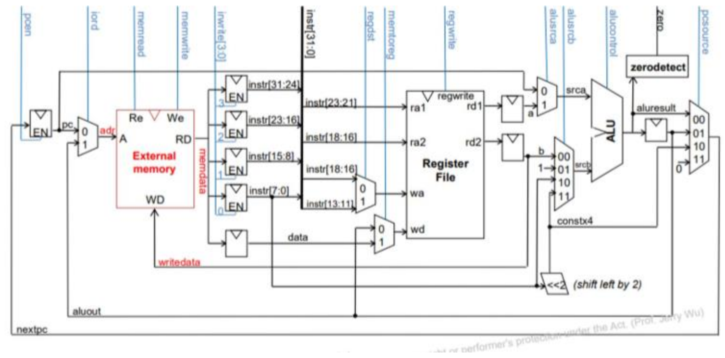
Dc_syn.tcl 66
Dc_test.tcl 77
Appendix B 94
Mips_scan.cell 94
Mips_scan.dft_drc 95
Mips_scan.scan_path 104
Mips_scan.violators 105
Mips_syn.area 114
Mips_syn.design 118
Mips_syn.net 126
Mips_syn.ports 134
Mips_syn.pow 142
Mips_syn.timing 145
Appendix C 157
Mips_syn.v 157
Mips_scan.v 210
Appendix D 265
tmax_atpg.tcl 265
Appendix E 269
maxtb.dat 269
Appendix F 431
Do.do file 431
Default.view 431

I. Introduction

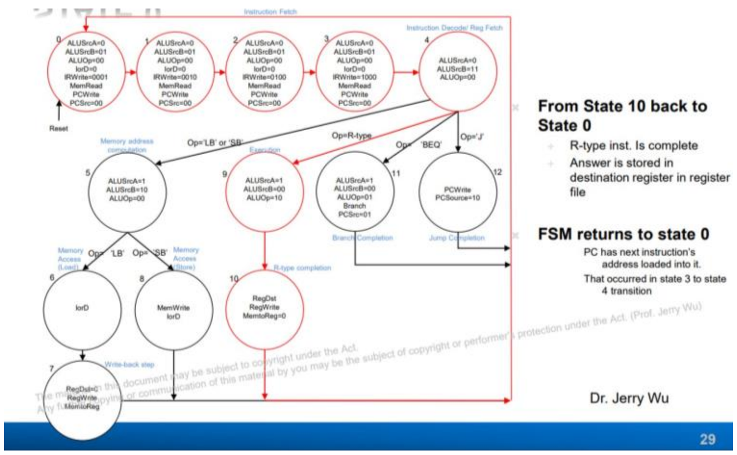
In this project the students used all the steps in the ASIC design flow to design, test, and build the Tiny MIPS CPU. The ASIC design flow starts with the specification/architecture step which is where for this project, the specification of the Tiny MIPS CPU and the block diagrams were given to the team from Professor Wu in the project description. Next, the team performed the RTL coding and Simulation which includes writing all the Verilog code and creating the test benches which are shown in HDL Code and Testing sections. The following step is the logic synthesis where the team used Synopsis Design Vision to realize the Verilog code into logic gate cells. Then right after, in the optimization and DFT insertion step, the team used a script in Synopsis Design Vision to replace the D Flip Flops with D Flip Flops with scan cells which completes the scan cell insertion process. The both the gate level simulation and the static timing analysis are performed in Tetramax with an ATPG script the team used. Then the team used Cadence Innovus, Chip Assembly Router, and Virtuoso to place and route the Tiny MIPS CPU synthesized schematic in a layout of a Pad frame. Finally, the team performed a DRC and LVS on the CPU in the pad frame with the schematic. Now the design is done and is ready to be fabricated. This project was a challenge for the whole team so completing this project while finishing senior design and other final assignments, the team was not able to add an enhancement.

II. Architecture

The diagrams below show the overall architecture of the CPU, these were used directly to design the Datapath and controller modules which drive the core functionality of the CPU.



Tiny MIPS CPU Datapath Block Diagram


Instruction Fetch
Instruction Decode/ Reg Fetch
ALUSrcA=0 ALUSrcB=01 ALUOp=00 IorD=0 IRWrite=0001 MemRead PCWrite PCSrc=00
ALUSrcA=0 ALUSrcB=01 ALUOp=00 IorD=0 IRWrite=0010 MemRead PCWrite PCSrc=00
ALUSrcA=0 ALUSrcB=01 ALUOp=00 IorD=0 IRWrite=0100 MemRead PCWrite PCSrc=00
ALUSrcA=0 ALUSrcB=01 ALUOp=00 IorD=0 IRWrite=1000 MemRead PCWrite PCSrc=00
ALUSrcA=0 ALUSrcB=11 ALUOp=00
Reset
Memory address computation
Op='LB' or 'SB'
Op=R-type
Op='BEQ'
Op='J'
Execution
ALUSrcA=1 ALUSrcB=10 ALUOp=00
ALUSrcA=1 ALUSrcB=00 ALUOp=10
ALUSrcA=1 ALUSrcB=00 ALUOp=01 Branch PCSrc=01
PCWrite PCSource=10
Branch Completion
Jump Completion
Memory Access (Load)
Memory Access (Store)
Op='LB'
Op='SB'
R-type completion
IorD
MemWrite IorD
RegDst RegWrite MemtoReg=0
Write-back step
RegDst=0 RegWrite MemtoReg

From State 10 back to State 0

- R-type inst. Is complete
- Answer is stored in destination register in register file

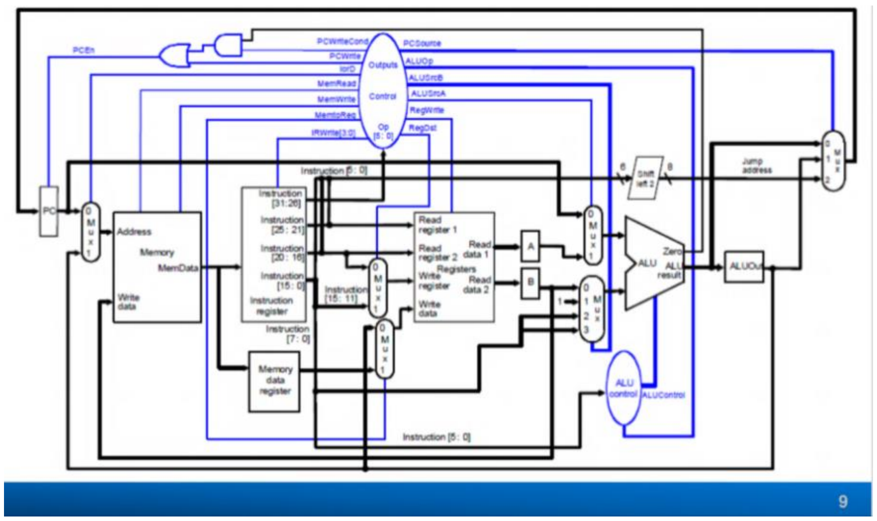
FSM returns to state 0

PC has next instruction's address loaded into it.
That occurred in state 3 to state 4 transition

Dr. Jerry Wu

29

Tiny MIPS CPU Controller Finite State Machine



Visualization of Controller in Tiny MIPS CPU

Instruction	Function	Encoding	OP (decimal)	OP (binary)	Func (decimal)	Func (binary)
add a, b, c	addition: a = b + c	R	0	000000	32	100000
sub a, b, c	subtraction: a = b - c	R	0	000000	34	100010
and a, b, c	bitwise AND: a = b and c	R	0	000000	36	100100
or a, b, c	bitwise OR: a = b + c	R	0	000000	37	100101
slt a, b, c	set less than: a=1 if b<c a = 0 otheriwse	R	0	000000	42	101010
addi a, b, #	add immediate: a = b + #	I	8	001000	n/a	n/a
beq a, b, addr	branch if equal: PC = PC + addr	I	4	000100	n/a	n/a
j addr	jump: PC = addr	J	2	000010	n/a	n/a
lb a, offset(b)	load byte: a = mem[b+offset]	I	32	100000	n/a	n/a
sb a, offset(b)	store byte: mem[b+offset] = a	I	48	110000	n/a	n/a

Tiny MIPS Instruction Set

III. HDL Code

Commented [JW1]: Add additional comments/intro to each module?

The following pages include all the Verilog HDL code used to implement the project. Each module will have its own header, be in landscape orientation, and have its own page as stated in the report template.

MIPS Module

```
//`timescale 1ns/10ps

module mips #(parameter WIDTH = 8, REGBITS = 3)

             (input              clk, reset, const_gnd,

              input  [WIDTH-1:0] memdata,

              output             memread, memwrite,

              output [WIDTH-1:0] adr, writedata);

   wire [31:0] instr;

   wire        zero, alusrca, memtoreg, iord, pcen, regwrite, regdst;

   wire [1:0]  aluop,pcsource,alusrcb;

   wire [3:0]  irwrite;

   wire [2:0]  alucont;

   controller  cont(.clk(clk), .reset(reset), .op(instr[31:26]), .zero(zero), .memread(memread),
.memwrite(memwrite),.alusrca(alusrca), .memtoreg(memtoreg), .iord(iord), .pcen(pcen), .regwrite(regwrite),
.regdst(regdst),.pcsource(pcsource), .alusrcb(alusrcb), .aluop(aluop), .irwrite(irwrite));

   alucontrol  ac(.aluop(aluop), .funct(instr[5:0]), .alucont(alucont));

   datapath #(WIDTH, REGBITS)  dp(.clk(clk), .reset(reset), .const_gnd(const_gnd), .memdata(memdata),
.alusrca(alusrca), .memtoreg(memtoreg), .iord(iord), .pcen(pcen),.regwrite(regwrite), .regdst(regdst),
.pcsource(pcsource), .alusrcb(alusrcb), .irwrite(irwrite), .alucont(alucont),.zero(zero), .instr(instr),
.adr(adr), .writedata(writedata));


endmodule
```

Controller

```
module controller(alusrca, alusrcb, aluop, pcen, iord, irwrite,

                  memread, memwrite, memtoreg,

              pcsource, regwrite, regdst,

                  op, clk, reset, zero) ;


   // INPUTS

   input  [5:0]   op          ;      // OPCODE

   input          clk, reset, zero ;


   // OUTPUTS

   output         alusrca    ;      // ALUSrcA

   output [1:0]   alusrcb    ;      // ALUSrcB

   output [1:0]   aluop      ;      // ALUOp

   output         pcen     ;        // Program Counter enable

   output         iord       ;      // IorD

   output [3:0]   irwrite    ;      // irwrite [0] = IRWrite0, irwrite[1] = IRWrite1, etc.

   output         memread    ;      // MemRead

   output         memwrite   ;      // MemWrite

   output         memtoreg   ;      // MemtoReg

   output [1:0]   pcsource   ;      // PCSrc

   output         regwrite   ;      // RegWrite

   output     regdst     ;     // RegDst
```

```
// REGISTERS
reg             alusrca ;
reg [1:0]      alusrcb    ;      // ALUSrcB
reg [1:0]      aluop      ;      // ALUOp
reg            branch     ;      // Branch
reg            iord       ;      // IorD
reg [3:0]      irwrite    ;      // irwrite [0] = IRWrite0, irwrite[1] = IRWrite1, etc.
reg            memread    ;      // MemRead
reg            memwrite   ;      // MemWrite
reg            memtoreg   ;      // MemtoReg
reg            pcwrite    ;      // PCWrite
reg [1:0]      pcsource   ;      // PCSrc
reg            regwrite   ;      // RegWrite
reg            regdst     ;      // RegDst



// STATES (12)
//
parameter      FETCH1  =  4'b0001;     // state 0
parameter      FETCH2  =  4'b0010;     // state 1
parameter      FETCH3  =  4'b0011;     // state 2
parameter      FETCH4  =  4'b0100;     // state 3
parameter      DECODE  =  4'b0101;     // state 4
parameter      MEMADR  =  4'b0110;     // state 5
parameter      LBRD    =  4'b0111;     // state 6
parameter      LBWR    =  4'b1000;     // state 7
parameter      SBWR    =  4'b1001;     // state 8
parameter      RTYPEEX =  4'b1010;     // state 9
parameter      RTYPEWR =  4'b1011;     // state 10
parameter      BEQEX   =  4'b1100;     // state 11
parameter      JEX     =  4'b1101;     // state 12
parameter        ADDIWR  =  4'b1110;   // state 13

// OPCODES  -- Make input decoding simpler
parameter      LB      =  6'b100000;   // load byte
parameter      SB      =  6'b101000; // store byte
parameter      RTYPE   =  6'b0;  // Register type instruction
parameter      BEQ     =  6'b000100; // Branch if Equal instruction i-type
parameter      J       =  6'b000010; //  Jump instruction
parameter      ADDI    =  6'b001000; // addi operation

// STATE REGISTERS
reg [3:0] state, nextstate;
```

```
   // state register
   always @(posedge clk)
      if(reset) state <= FETCH1;
      else state <= nextstate;

   // next state logic
   always @(*)
      begin
      case(state)
      FETCH1: nextstate <= FETCH2;
      FETCH2: nextstate <= FETCH3;
      FETCH3: nextstate <= FETCH4;
      FETCH4: nextstate <= DECODE;
      DECODE: case(op)
            LB: nextstate <= MEMADR;
            SB: nextstate <= MEMADR;
            RTYPE: nextstate <= RTYPEEX;
            BEQ: nextstate <= BEQEX;
            J: nextstate <= JEX;
            ADDI: nextstate <= MEMADR;
            default: nextstate <= FETCH1;
            endcase
      MEMADR: case(op)
```

```
            LB: nextstate <= LBRD;
            SB: nextstate <= SBWR;
            ADDI: nextstate <= ADDIWR;
            default: nextstate <= FETCH1;
            endcase
      LBRD: nextstate <= LBWR;
      RTYPEEX: nextstate <= RTYPEWR;
      LBWR: nextstate <= FETCH1;
      SBWR: nextstate <= FETCH1;
      RTYPEWR: nextstate <= FETCH1;
      BEQEX: nextstate <= FETCH1;
      JEX: nextstate <= FETCH1;
      ADDIWR: nextstate <= FETCH1;
      default: nextstate <= FETCH1;
      endcase
     end
   // registered outputs
   always @(*)
      begin
      irwrite <= 4'b0000;
      alusrca <= 0;
      alusrcb <= 2'b00;
      aluop <= 2'b00;
```

```
  iord <= 0;
memread <= 0;
memwrite <= 0;
  memtoreg <= 0;
  pcwrite <= 0;
pcsource <= 2'b00;
  regwrite <= 0;
regdst <= 0;
branch <= 0;
case(state)
FETCH1:
   begin
      memread <= 1;
      irwrite <= 4'b0001;
      alusrcb <= 2'b01;
      pcwrite <= 1;
   end
FETCH2:
   begin
      memread <= 1;
      irwrite <= 4'b0010;
      alusrcb <= 2'b01;
      pcwrite <= 1;
   end
FETCH3:
   begin
      memread <= 1;
      irwrite <= 4'b0100;
      alusrcb <= 2'b01;
      pcwrite <= 1;
   end
FETCH4:
   begin
      memread <= 1;
      irwrite <= 4'b1000;
      alusrcb <= 2'b01;
      pcwrite <= 1;
   end
DECODE:alusrcb <= 2'b11;
MEMADR:
   begin
      alusrca <= 1;
      alusrcb <= 2'b10;
   end
LBRD:
   begin
```

```
        memread <= 1;
        iord <= 1;
     end
  LBWR:
     begin
        regwrite <= 1;
        memtoreg <= 1;
     end
  SBWR:
     begin
        memwrite <= 1;
        iord <= 1;
     end
  RTYPEEX:
     begin
        alusrca <= 1;
        aluop <= 2'b10;
     end
  RTYPEWR:
     begin
        regdst <= 1;
        regwrite <= 1;
     end
```

```
      BEQEX:
         begin
            alusrca <= 1;
            aluop <= 2'b01;
            branch <= 1;
            pcsource <= 2'b01;
         end
      JEX:
         begin
            pcwrite <= 1;
            pcsource <= 2'b10;
         end
      ADDIWR:
         begin
            regdst <= 1;
            iord <= 1;
         end
      endcase
     end
  assign pcen = pcwrite | (branch & zero); //pc enable
endmodule
```

Datapath

```
// Datapath, including register file, ALU, muxes, and other registers
module datapath #(parameter WIDTH = 8, REGBITS = 3)
                 (input              clk, reset,
              input                const_gnd,
                  input  [WIDTH-1:0] memdata,
                  input              alusrca, memtoreg, iord,
                  input              pcen, regwrite, regdst,
                  input  [1:0]       pcsource, alusrcb,
                  input  [3:0]       irwrite,
                  input  [2:0]       alucont,
                  output             zero,
                  output [31:0]      instr,
                  output [WIDTH-1:0] adr, writedata);

   wire [REGBITS-1:0] ra1, ra2, wa;
   wire [WIDTH-1:0]   pc, nextpc, md, rd1, rd2, wd, a, src1, src2, aluresult, aluout, constx4;

   // shift left constant field by 2
   assign constx4 = {instr[WIDTH-3:0], {2{const_gnd}}};

   // register file address fields
   assign ra1 = instr[REGBITS+20:21];
   assign ra2 = instr[REGBITS+15:16];
   mux2       #(REGBITS) regmux(instr[REGBITS+15:16], instr[REGBITS+10:11], regdst, wa);


   // load instruction into four 8-bit registers over four cycles
   dffen      #(8)      ir0(clk, irwrite[3], memdata[7:0], instr[7:0]);
   dffen      #(8)      ir1(clk, irwrite[2], memdata[7:0], instr[15:8]);
   dffen      #(8)      ir2(clk, irwrite[1], memdata[7:0], instr[23:16]);
   dffen      #(8)      ir3(clk, irwrite[0], memdata[7:0], instr[31:24]);


   // datapath
   dffenr     #(WIDTH)  pcreg(clk, reset, pcen, nextpc, pc);
   dff        #(WIDTH)  mdr(clk, memdata, md);
   dff        #(WIDTH)  areg(clk, rd1, a);
   dff        #(WIDTH)  wrd(clk, rd2, writedata);
   dff        #(WIDTH)  res(clk, aluresult, aluout);
   mux2       #(WIDTH)  adrmux(pc, aluout, iord, adr);
   mux2       #(WIDTH)  src1mux(pc, a, alusrca, src1);
   mux4       #(WIDTH)  src2mux(writedata, {{7{const_gnd}},{~{const_gnd}}}, instr[WIDTH-1:0], constx4,
alusrcb, src2);
   mux4       #(WIDTH)  pcmux(aluresult, aluout, constx4, {8{const_gnd}}, pcsource, nextpc);
   mux2       #(WIDTH)  wdmux(aluout, md, memtoreg, wd);
   regfile    #(WIDTH,REGBITS) rf(clk, regwrite, ra1, ra2, wa, wd, rd1, rd2);
   alu        #(WIDTH) alunit(src1, src2, alucont, aluresult);
   zerodetect #(WIDTH) zd(aluresult, zero);
endmodule

module alu #(parameter WIDTH = 8)
            (input      [WIDTH-1:0] a, b,
             input      [2:0]       alucont,
             output reg [WIDTH-1:0] result);

   wire     [WIDTH-1:0] b2, sum, slt;

   assign b2 = alucont[2] ? ~b:b;
   assign sum = a + b2 + alucont[2];
   // slt should be 1 if most significant bit of sum is 1
   assign slt = sum[WIDTH-1];

   always@(*)
      case(alucont[1:0])
         2'b00: result <= a & b;
         2'b01: result <= a | b;
         2'b10: result <= sum;
         2'b11: result <= slt;
      endcase
endmodule
```

```
//call other modules
module regfile #(parameter WIDTH = 8, REGBITS = 3)
                (input                clk,
                 input                regwrite,
                 input  [REGBITS-1:0] ra1, ra2, wa,
                 input  [WIDTH-1:0]   wd,
                 output [WIDTH-1:0]   rd1, rd2);

   reg  [WIDTH-1:0] RAM [(1<<REGBITS)-1:0];

   always @(posedge clk)
      if (regwrite) RAM[wa] <= wd;

   assign rd1 = ra1 ? RAM[ra1] : 0;
   assign rd2 = ra2 ? RAM[ra2] : 0;
endmodule

module zerodetect #(parameter WIDTH = 8)
                   (input [WIDTH-1:0] a,
                    output            y);

   assign y = (a==0);
endmodule
```

```
module dff #(parameter WIDTH = 8)
             (input                  clk,
              input      [WIDTH-1:0] d,
              output reg [WIDTH-1:0] q);

   always @(posedge clk)
      q <= d;
endmodule

module dffen #(parameter WIDTH = 8)
               (input                  clk, en,
                input      [WIDTH-1:0] d,
                output reg [WIDTH-1:0] q);

   always @(posedge clk)
      if (en) q <= d;
endmodule

module dffenr #(parameter WIDTH = 8)
                (input                  clk, reset, en,
                 input      [WIDTH-1:0] d,
                 output reg [WIDTH-1:0] q);
   always @(posedge clk)
      if      (reset) q <= 0;
      else if (en)    q <= d;
endmodule

module mux2 #(parameter WIDTH = 8)
             (input  [WIDTH-1:0] d0, d1,
              input              s,
              output [WIDTH-1:0] y);

   assign y = s ? d1 : d0;
endmodule

module mux4 #(parameter WIDTH = 8)
             (input      [WIDTH-1:0] d0, d1, d2, d3,
              input      [1:0]       s,
              output reg [WIDTH-1:0] y);

   always @(*)
      case(s)
         2'b00: y <= d0;
         2'b01: y <= d1;
```

```
        2'b10: y <= d2;
        2'b11: y <= d3;
     endcase
endmodule
```

ALU

```
`timescale 1ns/10ps

module alu (result, a, b, alucont) ;


   input  [7:0] a, b ;

   input  [2:0] alucont ;

   output [7:0] result ;

   reg    [7:0] result ;

   wire   [7:0] b2, sum, slt ;

   assign b2 = alucont[2] ? ~b:b ;

   assign sum = a + b2 + alucont[2] ;

   // slt should be 1 if most significant bit of sum is 1

   assign slt = sum[7] ;

   always@(*)

      case(alucont[1:0])

         2'b00: result <= a & b ;

         2'b01: result <= a | b ;

         2'b10: result <= sum ;

         2'b11: result <= slt ;

      endcase


endmodule
```

ALU Control

```
module alucontrol( alucont, aluop, funct ) ;

// decodes 'funct' field from the assembly instruction, determines the type of instruction it is: R, I, J

// creates 3-bit control line (alucont) for the ALU

   input  [1:0] aluop   ;

   input  [5:0] funct   ;

   output [2:0] alucont ;

   reg    [2:0] alucont ;

   always @(*)

      case(aluop)

         2'b00: alucont <= 3'b010;  // add for lb/sb

         2'b01: alucont <= 3'b110;  // sub (for beq)

         default: case(funct)       // R-Type instructions

                     6'b100000: alucont <= 3'b010; // add (for add)

                     6'b100010: alucont <= 3'b110; // subtract (for sub)

                     6'b100100: alucont <= 3'b000; // logical and (for and)

                     6'b100101: alucont <= 3'b001; // logical or (for or)

                     6'b101010: alucont <= 3'b111; // set on less (for slt)

                     default:   alucont <= 3'b101; // should never happen

                  endcase

      endcase


endmodule
```

Jaret Williams
Nathan Pen

MIPS Memory

```
module mips_mem #(parameter WIDTH = 8, REGBITS = 3)(clk, reset, const_gnd);

   input clk, reset,const_gnd;


   wire    memread, memwrite;

   wire    [WIDTH-1:0] adr, writedata;

   wire    [WIDTH-1:0] memdata;


   // instantiate the mips processor

   ram memory (.memdata(memdata), .memwrite(memwrite), .adr(adr), .writedata(writedata), .clk(clk)) ;

   mips cpu(.clk(clk), .reset(reset), .const_gnd(const_gnd), .memdata(memdata), .memread(memread),
.memwrite(memwrite), .adr(adr), .writedata(writedata));


endmodule
```

External Memory

```
//connects the memory to the provided testing data

module exmem #(parameter WIDTH = 8, RAM_ADDR_BITS = 8)

   (input clk, en, memwrite,

    input [7:0] adr,

    input [7:0] writedata,

    output reg [7:0] memdata

    );


      integer i;

   reg [7:0] mips_ram [0:256];


 initial

      begin

      for(i=0; i<256; i=i+1)

      begin

            mips_ram[i]=8'b0;

            end

      $display("memory scrubbed");

      $readmemh("fib.dat", mips_ram); //hex

//    $readmemb("fib.dat", mips_ram); //binary

      $display ("File loaded.");

      end
      integer k;
      initial begin
            $display("Contents of Mem after reading data file:");
            for (k=0; k<256; k=k+1) $display("%d:%h",k,mips_ram[k]);
      end

   always @(negedge clk)
      if (en) begin
         if (memwrite)
            mips_ram[adr] <= writedata;
         memdata <= mips_ram[adr];
      end
endmodule
```

Reg File

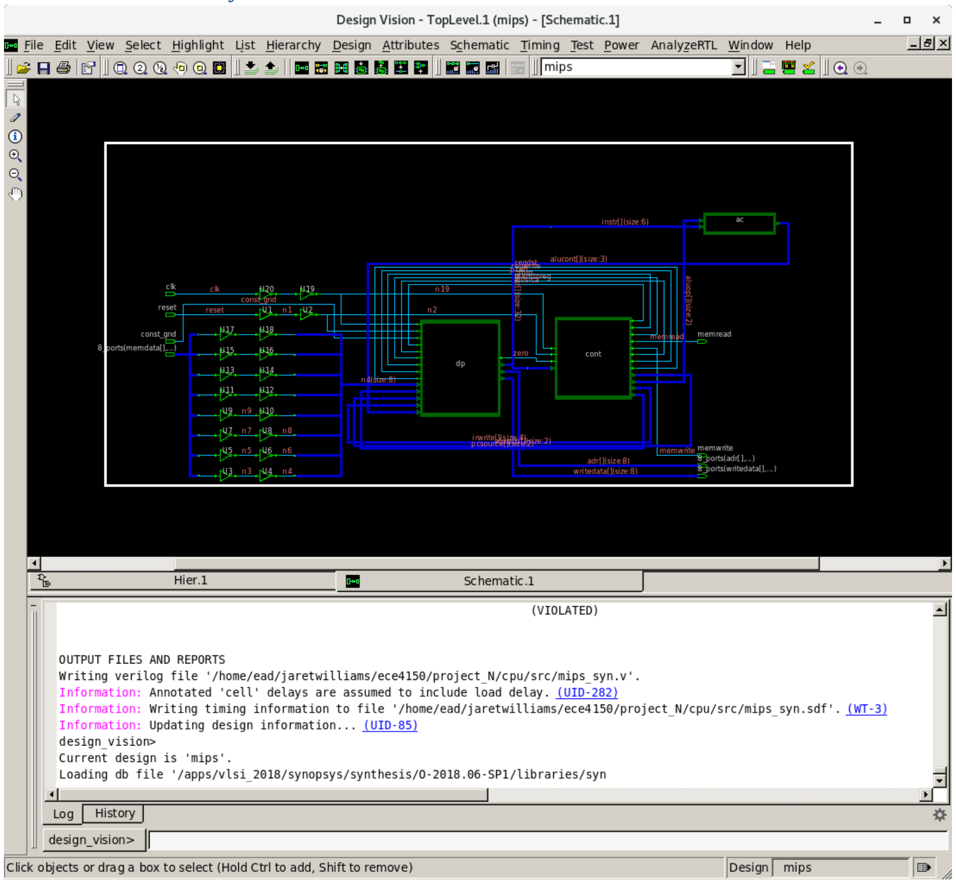
```
`timescale 1ns/10ps

module regfile (rd1, rd2, clk, regwrite, ra1, ra2, wa, wd) ;


// defines register operation for read and write operations


   input         clk ;

   input         regwrite ;  // signal to command regfile to 'write' to a reg

   input  [2:0] ra1 ;       // 3-bit address of $s0 --> $s7 (source reg: RS)

   input  [2:0] ra2 ;       // 3-bit address of $s0 --> $s7 (source reg: RT)

   input  [2:0] wa  ;       // 3-bit address of $s0 --> $s7 (destin reg: RD)


   input  [7:0] wd  ;       // 8-bit data to write to RD

   output [7:0]  rd1, rd2 ; // 8-bit data to read from RS and RT


   // 2-dimensional register (8x8) -- holds actual registers $s0 through $s7

   reg  [7:0] REGS [7:0];



   // WRITE

   always @(posedge clk)

      if (regwrite) REGS[wa] <= wd;
   // READ
   assign rd1 = ra1 ? REGS[ra1] : 0;   // looks up address ra1 in reg array
   assign rd2 = ra2 ? REGS[ra2] : 0;   // looks up address ra2 in reg array
                                // note: register 0 hardwired to 0

endmodule
```

IV. Synthesis

For the synthesis of the CPU the team utilized Synopsis Design Vision and two synthesis scripts provided to the students in the labs. The students followed the lab 5 manual which had the students first run the dc_syn.tcl (Appendix A) script that was adapted for the MIPS CPU project. This first allowed the HDL code to be synthesized into osu5_stdcells so the Verilog code be realized as logic gates down to the transistor level. Next the dc_test.tcl (Appendix A) script was run to have DFT incorporated into the design. This was incorporated by replacing the standard D Flip Flops with scan cell D Flip Flop's. This added two new pins to the design, test_si and test_se which allows TetraMax to utilize these pins when performing ATPG testing on the design. The resulting design was optimized for area and not for area, with the following clock parameters: clock network latency of 0.3 ns, 2.0 ns delay from clock to valid inputs, 1.65 ns delay from clock to valid output, setting max fanout load for input pins, set default strobe time in a test cycle for output ports to 40, and setting the default length of a test vector cycle to 100.

Fortunately, the team did not run into too many errors during synthesis. The only issue that occurred was when first completing the DFT insertion by hand and not using the dc_test.tcl script, the team forgot to write the command to create the mip_scan.sdc file that was later used in the place and route of the CPU. The fix was to run the dc_test.tcl script for the MIPS CPU Verilog file which automatically generated the .sdc file and got the project back on track.

Pre DFT-Synthesis Schematic

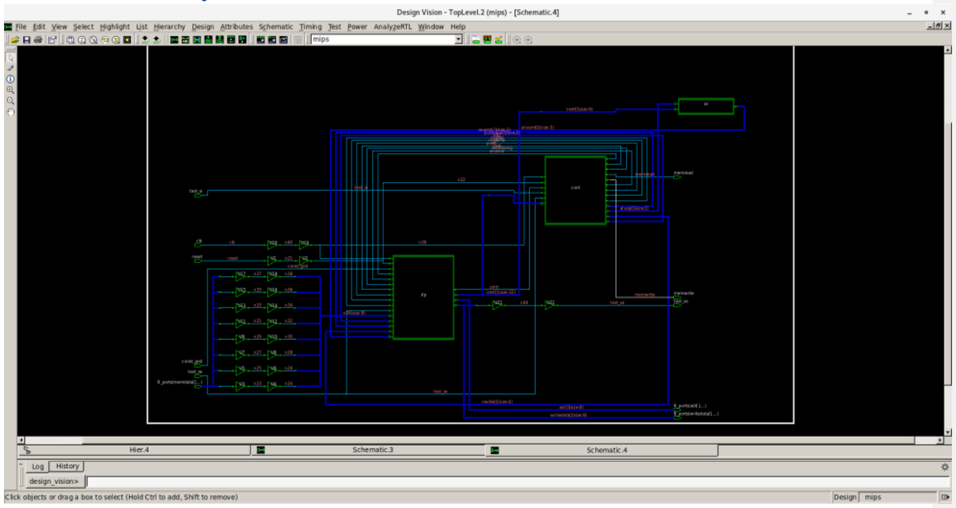

Design Vision - TopLevel.1 (mips) - [Schematic.1]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
mips
ac
dp
cont
clk
reset
const_gnd
memread
memwrite
Hier.1
Schematic.1
Log
History
design_vision>

```
                                                    (VIOLATED)


OUTPUT FILES AND REPORTS
Writing verilog file '/home/ead/jaretwilliams/ece4150/project_N/cpu/src/mips_syn.v'.
Information: Annotated 'cell' delays are assumed to include load delay. (UID-282)
Information: Writing timing information to file '/home/ead/jaretwilliams/ece4150/project_N/cpu/src/mips_syn.sdf'. (WT-3)
Information: Updating design information... (UID-85)
design_vision>
Current design is 'mips'.
Loading db file '/apps/vlsi_2018/synopsys/synthesis/O-2018.06-SP1/libraries/syn
```

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove) Design mips

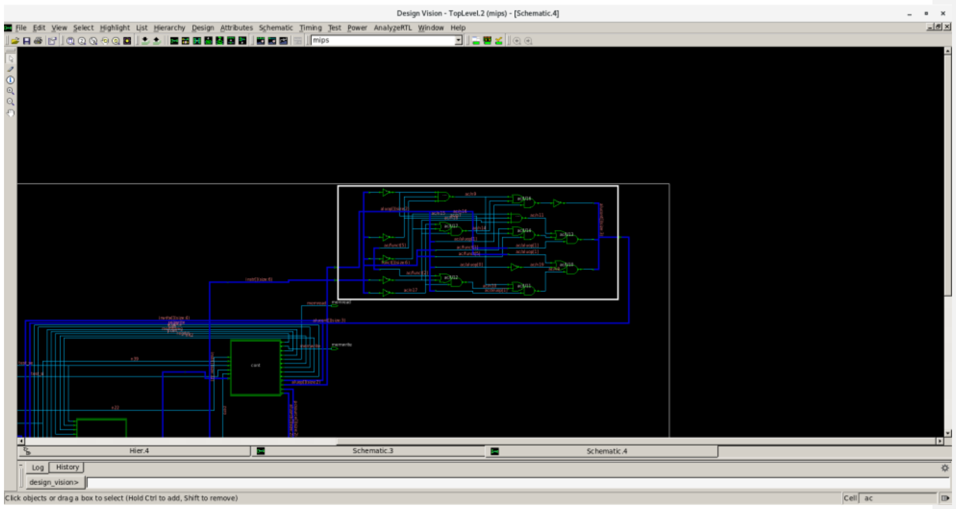
TinyMIPS CPU Pre DFT Synthesis

Post-DFT Synthesis Schematic

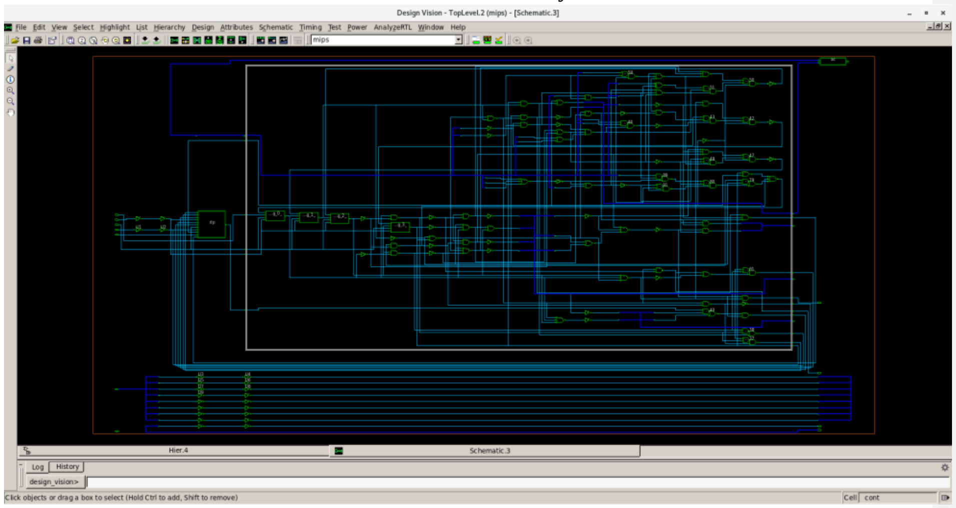

Design Vision - TopLevel.2 (mips) - [Schematic.4]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
mips
Hier.4
Schematic.3
Schematic.4
Log History
design_vision>

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

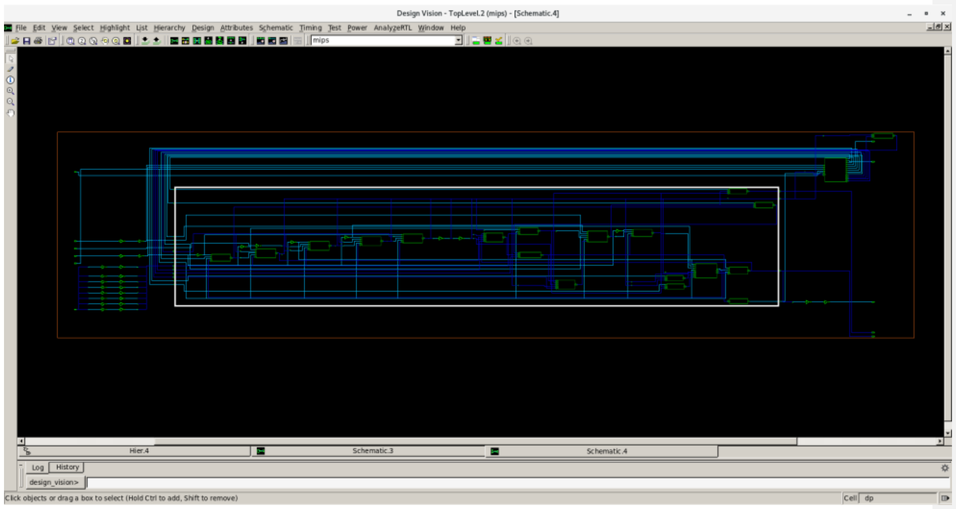
TinyMIPS Schematic Post-DFT Synthesis


Design Vision - TopLevel.2 (mips) - [Schematic.4]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Hier.4
Schematic.3
Schematic.4
Log
History
design_vision>
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
Cell ac

ALU Controller Post-DFT Synthesis

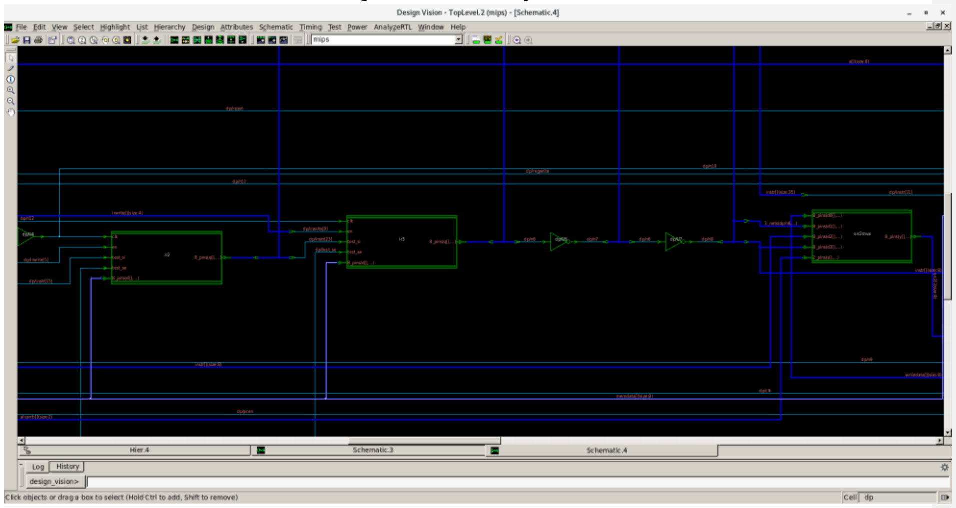

Design Vision - TopLevel.2 (mips) - [Schematic.3]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Hier.4
Schematic.3
Log
History
design_vision>
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
Cell cont

Controller Post-DFT Synthesis


Design Vision - TopLevel.2 (mips) - [Schematic.4]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Hier.4
Schematic.3
Schematic.4
Log
History
design_vision>

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

Datapath Pos- DFT Synthesis


Design Vision - TopLevel.2 (mips) - [Schematic.4]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Hier.4
Schematic.3
Schematic.4
Log
History
design_vision>

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

Single Flip Flop Post-DFT Synthesis

Jaret Williams
Nathan Pen

Synthesis Reports

Throughout the synthesis process, many reports were created at each step to give the user the desired values and information of the resultant synthesized circuit. After scan cell insertion the total area of the MIPS CPU is 362,835.00 µm^2, a total dynamic power of 1.9390 mW, cell leakage power of 85.2062 nW, and a data arrival time of 0.37. All synthesis reports in are Appendix B and all the synthesized code is in Appendix C.

V. Testing

Functional Testing

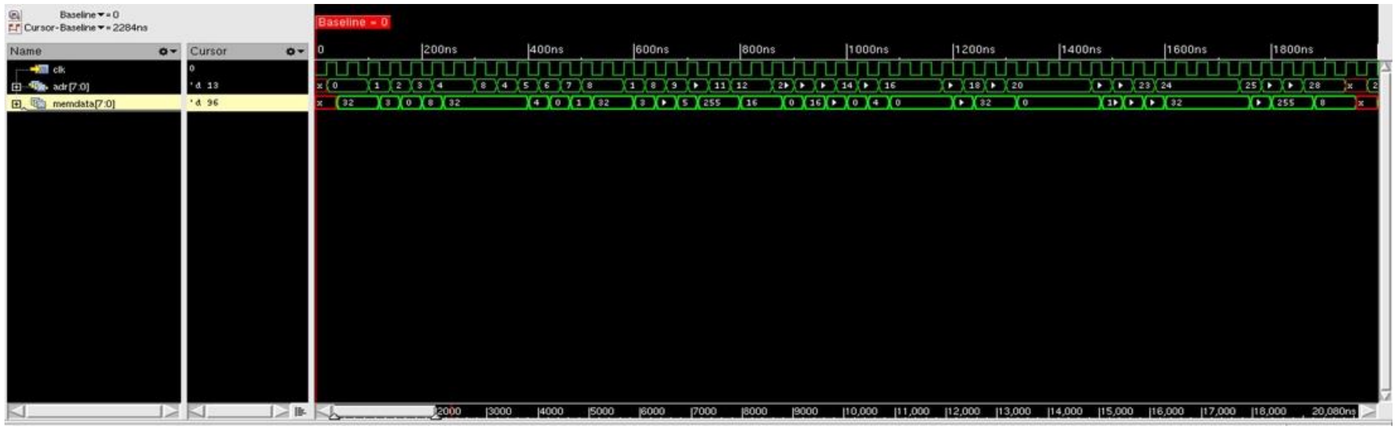
Throughout the HDL design process multiple test benches were created and verified to ensure that each portion of the CPU worked properly to minimize errors when compiling everything into the final CPU. For the controller test bench, shown below, the team used a clock with a 10ns period and calculated how long it would take to go through every state, and changed the operation code to ensure that the controller runs through all the necessary states with the correct outputs. Also, the reset function on the controller was tested. For the Datapath testbench, shown below, the test bench loads a test code into the input of the Datapath. Then the output is checked with what the expected output and if it matches then the Datapath was coded correctly. The Datapath test bench allowed the team to find an error with naming of the ir flip-flops. The final test bench that the team utilized is the CPU test bench, shown below. In the CPU test bench, the team loaded the ram.dat file with the correct instructions to run the Fibonacci sequence and then connected the ram.v file to the mips.v file to get a complete working system. The CPU test bench outputted the states and results, the team was able to see the waveform and then see the final output of the test bench to be decimal value 3 which would be at the memory address 255. This output confirmed that the team coded the TinyMIPS CPU correctly and continue with the project following the ASIC Design flow.

CPU Test Bench

```
// top level testing

module top_tb #(parameter WIDTH = 8, REGBITS = 3)();

   parameter FINISHTIME = 20000;

   parameter CLKPERIOD  = 20;

   parameter const_gnd = 1'b0;

   reg clk = 0;

   reg reset = 1;

   // testing mips memory module

   mips_mem2 #(WIDTH,REGBITS) dut(.clk(clk), .reset(reset), .const_gnd(const_gnd));

   // initialize

   initial

      begin

         reset <= 1;

         #(4*CLKPERIOD) reset <= 0;

         #FINISHTIME

       $display("Finishing Simulation due to simulation constraint.");

      $finish;

      end

   always #CLKPERIOD clk <= ~clk;    // clock gen

   // test Fibonacci Simulation

   always@(negedge clk)
```

```
begin
   if(dut.memwrite)
      if(dut.adr == 8'hFF & dut.writedata == 8'h0D)
      begin
               $display("Fibonacci Simulation was successful!!!");
                 #(4*CLKPERIOD)
             $display("Ending Simulation.");
               $finish;
      end
      else
          begin
               $display("Fibonacci Simulation has failed...");
               $display("Data at address FF should be 0D");
           #(4*CLKPERIOD)
             $display("Ending Simulation.");
               $finish;
          end
end
initial
begin
      $shm_open("top_tb.db");
      $shm_probe(top_tb,"AS");
end
```

```
endmodule
```



Output of CPU Test Bench

Controller Test Bench

```
`timescale 1ns/10ps

module controller_tb();

// Verifies proper controller functionality

reg [5:0]         op;

reg         reset,clk;


wire        alusrca;

wire [1:0]      alusrcb;      // ALUSrcB

wire [1:0]      aluop;  // ALUOp

wire            branch; // Branch

wire            iord;   // IorD

wire [3:0]      irwrite;      // irwrite [0] = IRWrite0, irwrite[1] = IRWrite1, etc.

wire            memread;      // MemRead

wire            memwrite;     // MemWrite

wire            memtoreg;     // MemtoReg

wire            pcwrite;      // PCWrite

wire [1:0]      pcsource;     // PCSrc

wire            regwrite;     // RegWrite

wire          regdst;   // RegDst

wire [3:0]  state;


// OPCODES
```

```
parameter      LB      =  6'b100000;      // load byte

parameter      SB      =  6'b101000; // store byte

parameter      RTYPE   =  6'b0; // Regsiter type instuction

parameter      BEQ     =  6'b000100; // Branch if Equal instruction i-type

parameter      J       =  6'b000010; //  Jump

parameter   ADDI  =  6'b001000; // addi op


controller  cont(.clk(clk), .reset(reset), .op(op), .zero(zero), .memread(memread),
.memwrite(memwrite),.alusrca(alusrca), .memtoreg(memtoreg), .iord(iord), .pcen(pcen), .regwrite(regwrite),
.regdst(regdst),.pcsource(pcsource), .alusrcb(alusrcb), .aluop(aluop), .irwrite(irwrite));

//test each state of FSM changing the op code

initial

    begin

    op <= LB;

    reset <= 0;

    $monitor("op: %6b, state: %d", op, state-1);

    #90

    op <= SB;

    #80

    op <= RTYPE;

    #80

    op <= BEQ;

    #70
    op <= J;
    #70
    op <= ADDI;
    #90
    reset <= 1;
    #30
    reset <= 0;
    #40
    $finish;
    end
initial
    begin
    clk=1'b0;
    end
always #5
    begin
    clk=~clk;
    if (clk) $display("posedge clk");
    end

initial
 begin
    // Open a db file for saving simulation data
```

```
  $shm_open ("controller_tb.db");
    // Collect all signals (hierarchically) from the module "controller_tb"
  $shm_probe (controller_tb,"AS");
 end
endmodule
```

Datapath Test Bench

```
`timescale 1ns/10ps

module test #(parameter WIDTH = 8, REGBITS = 3) ;

// Verifies proper data flow in datapath

   parameter CLK_H = 20 ;    // half clock period

   parameter CLK_P = 40 ;    // full clock period


   // INPUTS (13)

   reg  [2:0]  alucontrol ;  // control signal for ALU

   reg         alusrca    ;  // control signal for 2:1 mux for ALU's srca input

   reg  [1:0]  alusrcb    ;  // control signal for 4:1 mux for ALU's srcb input

   reg         iord       ;  // control signal for 2:1 mux from Program counter

   reg  [3:0]  irwrite    ;  // control signal for the 4 DFF's holding the instruction

   reg  [7:0]  memdata    ;  // 8-bit line coming from memory's RD line

   reg         memtoreg   ;  // control signal for the 2:1 mux for memory's WD line

   reg         pcen       ;  // control signal for PC's DFF

   reg         regdst     ;  // control signal for 2:1 mux for memory's WA line

   reg         regwrite   ;  // control signal for regfile

   reg  [1:0]  pcsource   ;  // control signal for 4:1 mux leading to PC register

   reg         clk, reset ;


   // OUTPUTS (4)

   wire [ 7:0] adr        ;  // output coming from 2:1 mux from program counter
   wire [31:0] instr      ;  // output coming from all 4 DFF's holding the instruction
   wire [ 7:0] writedata  ;  // output leading to WD line on memory
   wire        zero       ;  // output coming from zero detect module
assign const_gnd=1'b0;

   datapath    #(WIDTH, REGBITS) dp(.clk(clk), .reset(reset), .const_gnd(const_gnd), .memdata(memdata),
.alusrca(alusrca), .memtoreg(memtoreg), .iord(iord), .pcen(pcen),.regwrite(regwrite), .regdst(regdst),
.pcsource(pcsource), .alusrcb(alusrcb), .irwrite(irwrite), .alucont(alucont),.zero(zero), .instr(instr),
.adr(adr), .writedata(writedata));

   initial begin
      $monitor ("CLK= %b, instruction= %b", clk, instr ) ;

      clk <= 0 ; reset <= 0 ; alucontrol <=3'b0 ; alusrca <=0 ;  alusrcb <= 2'b0 ; iord <=0 ; irwrite <=
4'b0 ;
      memtoreg <=0 ; pcen <=0 ; regdst <=0 ; regwrite <=0 ; pcsource <=2'b0 ; memdata<=8'b0 ;

      #CLK_P $display ("reset now clocked in" ) ;
      reset <= 1 ;




      // add $s1 $s2 $s3: 0000 0000 0100 0011 0000 1000 0010 0000
      // send in first byte of instruction:
```

```
      irwrite <= 4'b0001 ;
      memdata <= 8'b00000000 ;

      // send in 2nd byte of instruction:
      #CLK_P irwrite <=4'b0010 ;
      memdata <= 8'b01000011;

      // send in 3rd byte of instruction:
      #CLK_P irwrite <=4'b0100 ;
      memdata <= 8'b00001000;

      // send in 4th byte of instruction:
      #CLK_P irwrite <=4'b1000 ;
      memdata <= 8'b00100000;
#CLK_P if (instr=={8'b00000000,8'b01000011,8'b00001000,8'b00100000})
   $display("INSTRUCTION load succeed.");
else $display("INSTRUCTION load fail.");

         alusrca <=0;
        alusrcb <=2'b11;
        iord    <=0;
        irwrite <=4'b0000;
        memtoreg<=0;
```

```
pcen <=0;
pcsource<=2'b00;
regwrite<=0;
regdst  <=0;
 #CLK_P
 alucontrol<=3'b010;
 alusrca <=1;
alusrcb <=2'b00;
iord    <=0;
irwrite <=4'b0000;
memtoreg<=0;
pcen <=0;
pcsource<=2'b00;
regwrite<=0;
regdst  <=0;

#CLK_P
 alucontrol<=3'b000;
 alusrca <=0;
alusrcb <=2'b00;
iord    <=0;
irwrite <=4'b0000;
memtoreg<=0;
```

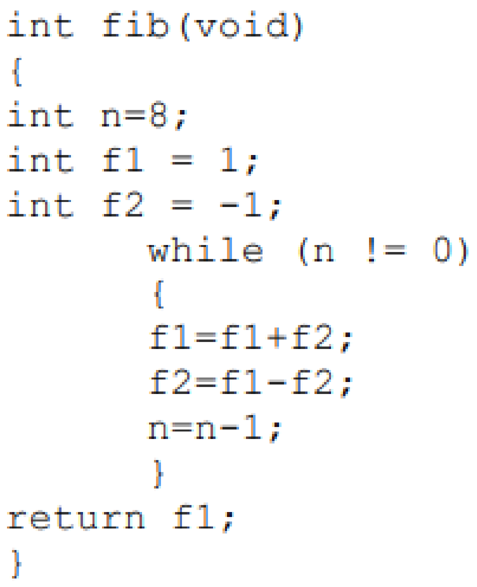
```
        pcen <=0;
        pcsource<=2'b00;
        regwrite<=1;
        regdst  <=1;

      #CLK_P $finish ;

   end



   always
      #CLK_H clk = ~clk;



   initial begin
      $shm_open("datapath.db");
      $shm_probe(test, "AS");
      $shm_save;
   end


endmodule
```

RAM Verilog File

```
module ram (memdata, memwrite, adr, writedata, clk);

// ram file given in lab

   output [7:0] memdata   ;

   input        memwrite  ;

   input  [7:0] adr       ;

   input  [7:0] writedata ;

   input        clk       ;


   reg    [7:0] memdata   ;

   reg    [7:0] mips_ram [0:255] ;  // actual memory 2D array


   integer i, k ;


   // The following $readmemh statement initializes the RAM contents

   // via an external file (use $readmemb for binary data). The ram.dat

   // file is a list of bytes, one per line, starting at address 0.


   initial begin


      // reset's memory upon startup

      for(i=0; i<256; i=i+1)

            mips_ram[i]=8'b0;
      $display("RAM: Memory initialized to 0");

      // loads contents of memory from a file called: ram.dat
      $readmemb("ram2.dat", mips_ram);          // if ram.dat is binary
      //$readmemh("ram2.dat", mips_ram);  // if ram.dat is hex
      $display ("RAM: External Memory File: ram.dat loaded into RAM.");

      // displays contents of memory after file load
      $display("RAM: Contents of Mem after reading data file:");
      for (k=0; k<256; k=k+1)
            $display("%d:%b",k,mips_ram[k]);
   end

   // The behavioral description of the RAM - note clocked behavior
   always @(negedge clk) begin
         if (memwrite) begin      // must be a write
            mips_ram[adr] = writedata;
          $writememb("ramsyn.after.dat", mips_ram) ; //write out contents of ram to file
       end
         memdata <= mips_ram[adr];  // must be a read
   end
endmodule
```

RAM Data

```
// -------------------------------------------------------------------------
// BINARY INSTRUCTION:              MEM. ADR:       INSTR:
// -------------------------------------------------------------------------
00100000 00000011 00000000 00001000   // 0->3         addi $3, $0, 8
00100000 00000100 00000000 00000001   // 4->7         addi $s4,$s0,1
00100000 00000101 11111111 11111111   // 8->11        addi $s5,$s0,-1
00010000 01100000 00000000 00000100   // 12->15       Loop: Beq $s3,$s0, Exit
00000000 10000101 00100000 00100000   // 16->19       add $s4, $s4, $s5
00000000 10000101 00101000 00100010   // 20->23       sub $s5, $s4, $s5
00100000 01100011 11111111 11111111   // 24->27       addi $s3,$s3,-1
00001000 00000000 00000000 00000011   // 28->31       J Loop
10100000 00000100 00000000 11111111   // 32->3        Exit:Sb $s4,255($s0)
```



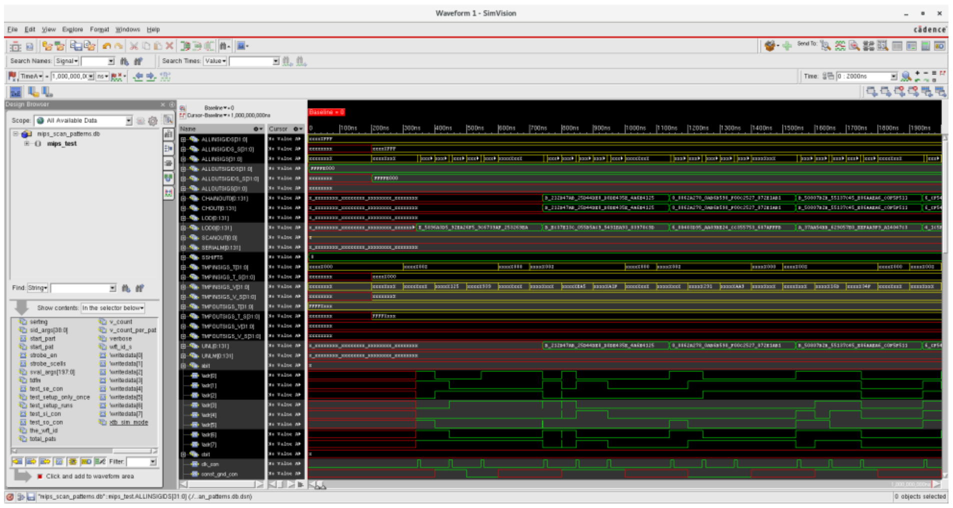
```
int fib(void)
{
int n=8;
int f1 = 1;
int f2 = -1;
      while (n != 0)
      {
      f1=f1+f2;
      f2=f1-f2;
      n=n-1;
      }
return f1;
}
```

RAM.dat (Fibonacci Sequence) in C

DFT

The team used the full scan automatic test pattern generation (ATPG) design for test (DFT) strategy for the project. This involves using the tool Tetramax which runs through the synthesized, scan-cell inserted MIPS CPU design and then develops numerous input patterns, known as test vectors, to recognize locations where potential faults in the design could arise. This process was done for specifically stuck-at faults to see if a gate input is stuck at a value of either 0 or 1. The team performed ATPG for the MIPS CPU using the tmax_atpg.tcl script (Appendix D) then the test vectors are outputted to a Verilog test bench (image shown below) and into a data file (Appendix E). After the tmax_atpg.tcl script ran, the program created 271 test vectors, one of the test vectors used is called pattern 267 which is shown below. The way test vector works is that it inputs the Proc load_unload value and then compares the allclock_launch value to the allclock_capture value and if they are different, it shows that there could be a potential fault at the certain position that the program is checking.

If the chip was fabricated, there are two different processes that one could use to verify the functionality of a chip. One way for a small one-off chip, would be using a probe station to see if the output of a test input from one of the test vectors matches the desired output. This result would verify if the chip is functioning correctly or not. Another option that is used in large manufacturing would be an ATE (Automatic Test Equipment) machine that is preloaded with the ATPG data created from a program like Tetramax and automatically compare the outputs of the test inputs to verify the proper functionality of the chip. The ATE machines are much more accurate than doing by hand using a probe station but are extremely expensive, therefore they are only practical for industrial use.


Successful Screen Output of DFT Testbench

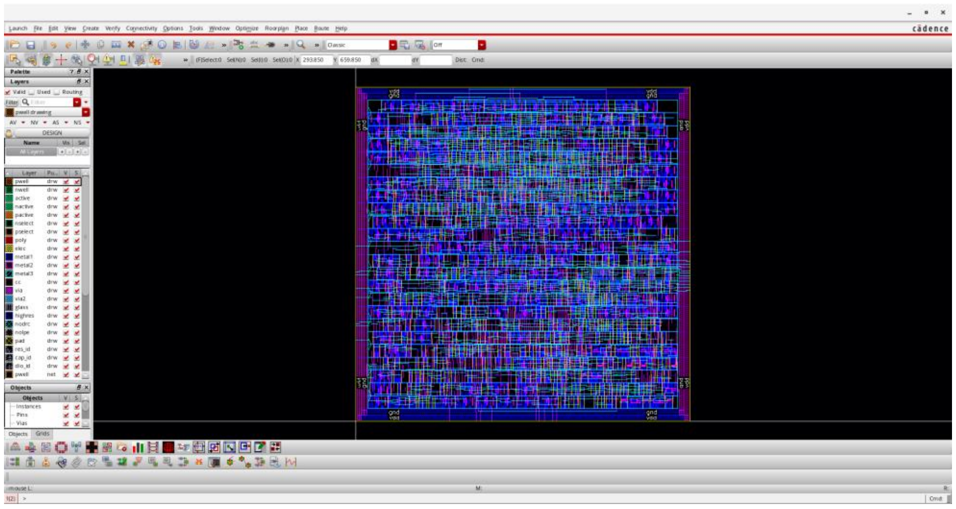
```
//Pattern #267
0000000000000001
// Proc load_unload
000010_0000000000001111
10111000110110111110111000010000000101111101111010111000101101111111001111001011111001100001111000001101000000000100011100000000001011_011111
01100000001100111100000011111111110100111110011010010111000000100000100000110000100001011000011110110101010001001001010101011001101_001011
// Proc allclock_launch
000001_0000000000010000
Z000000000000_1101011110101_100000
// Proc allclock_capture
000001_0000000000010001
Z000000000000_1000101000010_100000
```

Test Vector Pattern #267 from ATPG

VI. Layout

P&R

One of the final steps of the ASIC design flow is the place and route of the synthesized design. To do this, the team used the Innovus Digital Implementation System from Cadence. Following the steps in lab 7, the team was able to successfully place and route the MIPS layout. The floorplan was set at the automatically generated value generated for our design. Gnd and vdd 'rings' were used around the outside of the design with vertical and horizontal power rails. The default settings were mostly used during the process. The picture below shows our MIPS chip after P&R



CCAR

The final step in the layout process was to take the layout that team created in Innovus, and now connect it to the pad frame. This was done using the Cadence Chip Assembly Router (CCAR). This made it much faster to connect our many inputs from the main module to the outer pad frame, compared to doing it by hand like it ECE 4140.

One of the first step was to set up the pad frame being used (Frame1_38: 1 Tiny Chip unit frame, 900x900 microns interior area), we were able to use the smallest size without any problems. Each pad was adjusted to reflect the input or output needed and pins were added on both sides. These additional internal signals are shown in the table below and are necessary to connect the pad frame to the appropriate MIPS pins during the CCAR process.

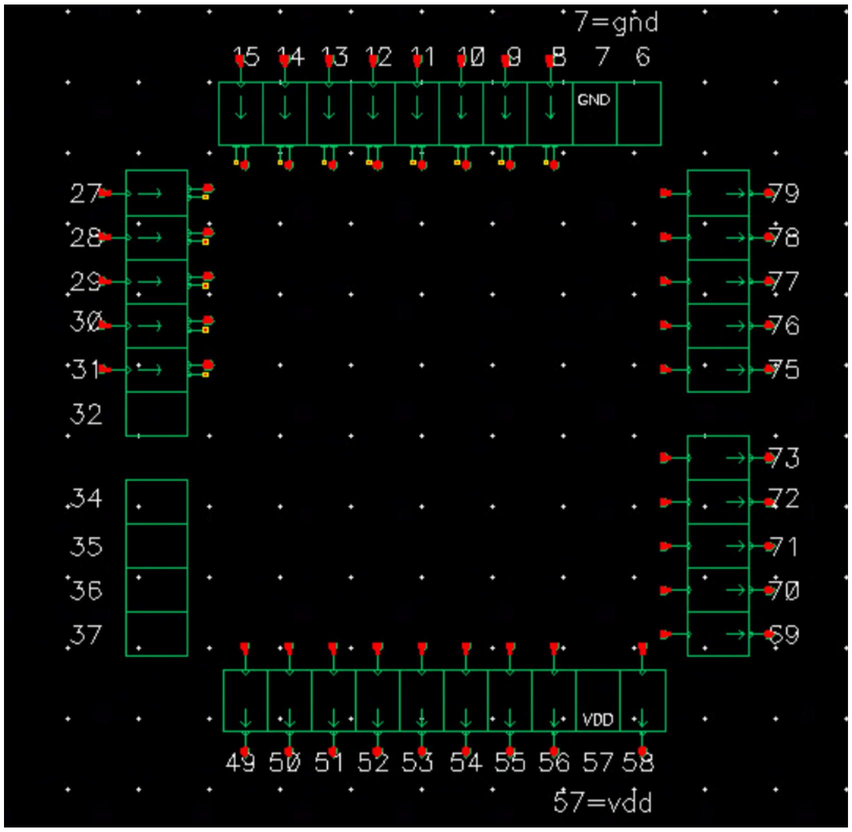
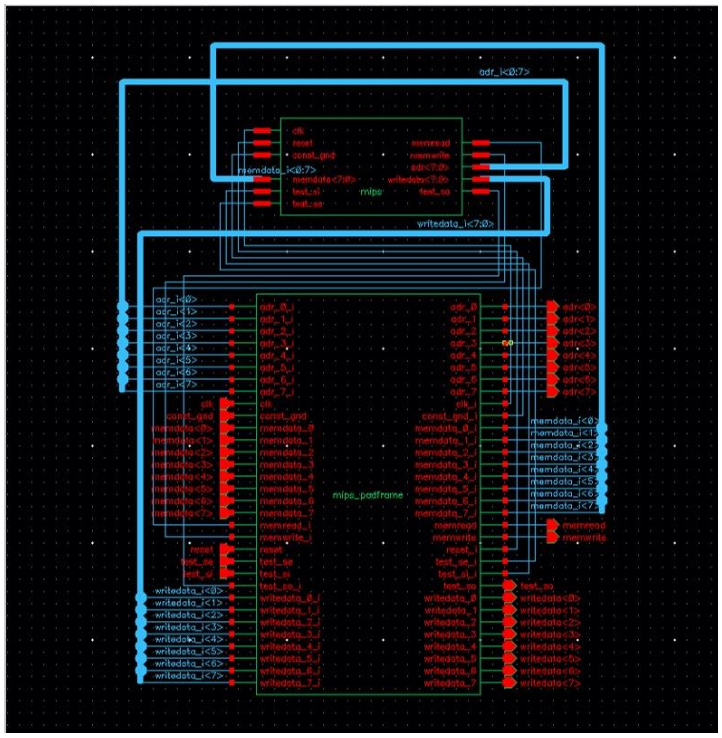
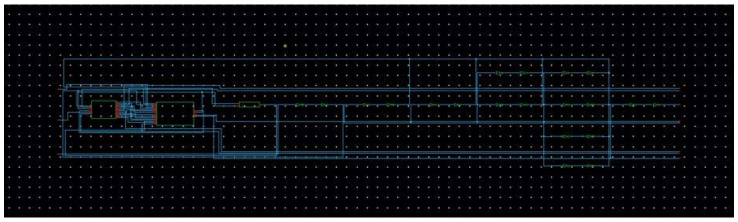


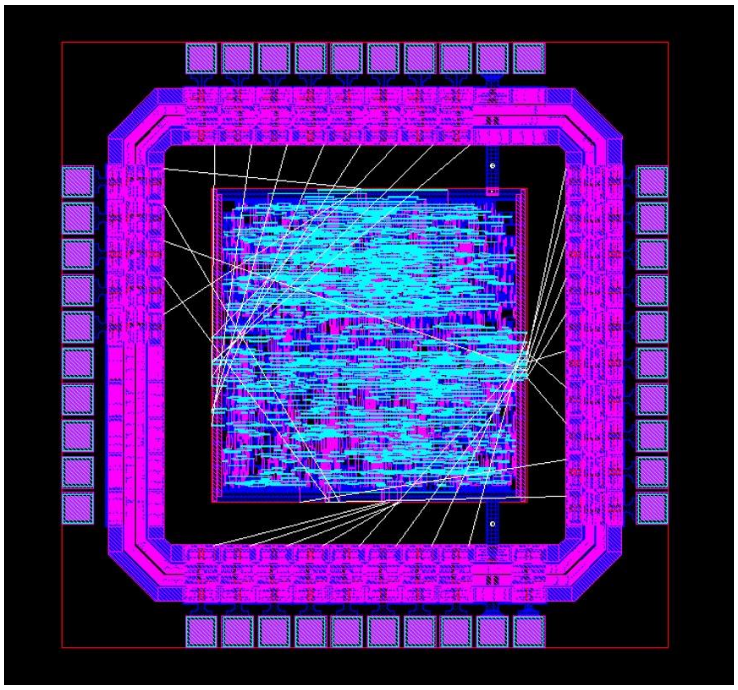
Table mapping PAD frame pads to processor I/O

MIPS		**MIPS Pad Frame Internal Signals**	
Input	Output	Input Pad	Output Pad
clk	memread	Clk_i	Memread_i
reset	Memwrite	Reset_i	Memwrite_i
Const_gnd	Adr<7:0>	Const_gnd_i	Adr_i<7:0>
Memdata<7:0>	Writedata<7:0>	Memdata_i<7:0>	Writedata_i<7:0>
Test_si	Test_so	Test_si_i	Test_so_i
Test_se		Test_se_i	

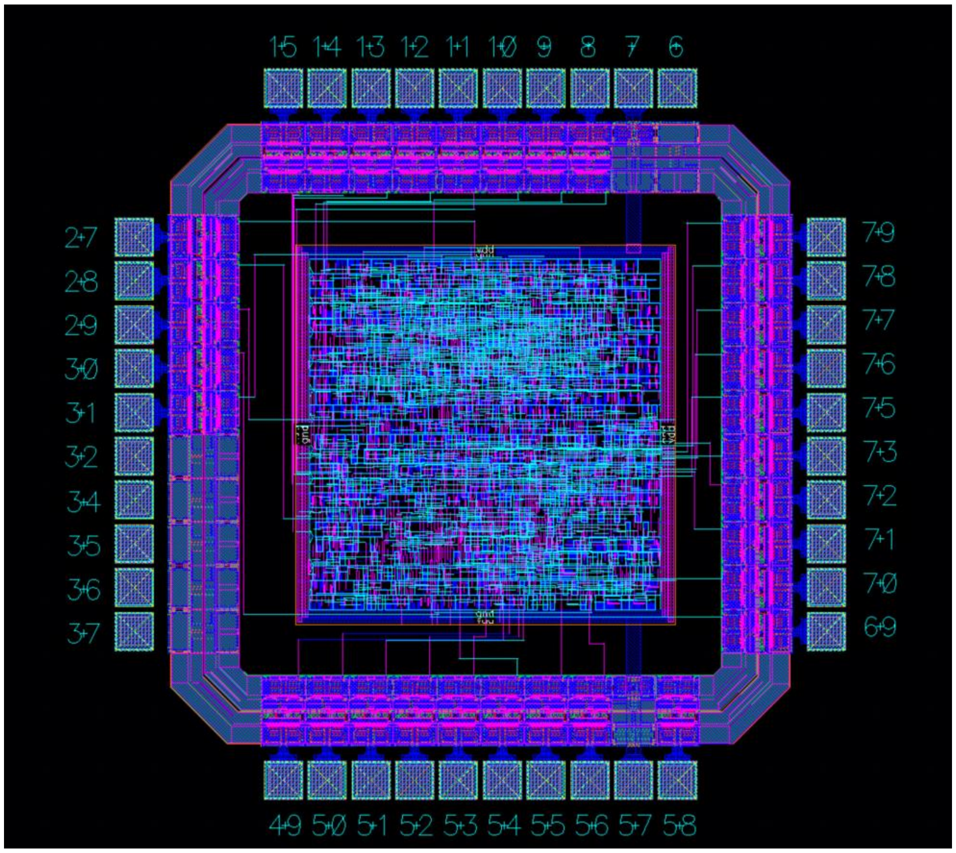
A schematic view for both the MIPS module and the pad frame were made and the signals were wired together, making sure to connect the internal signals correctly.



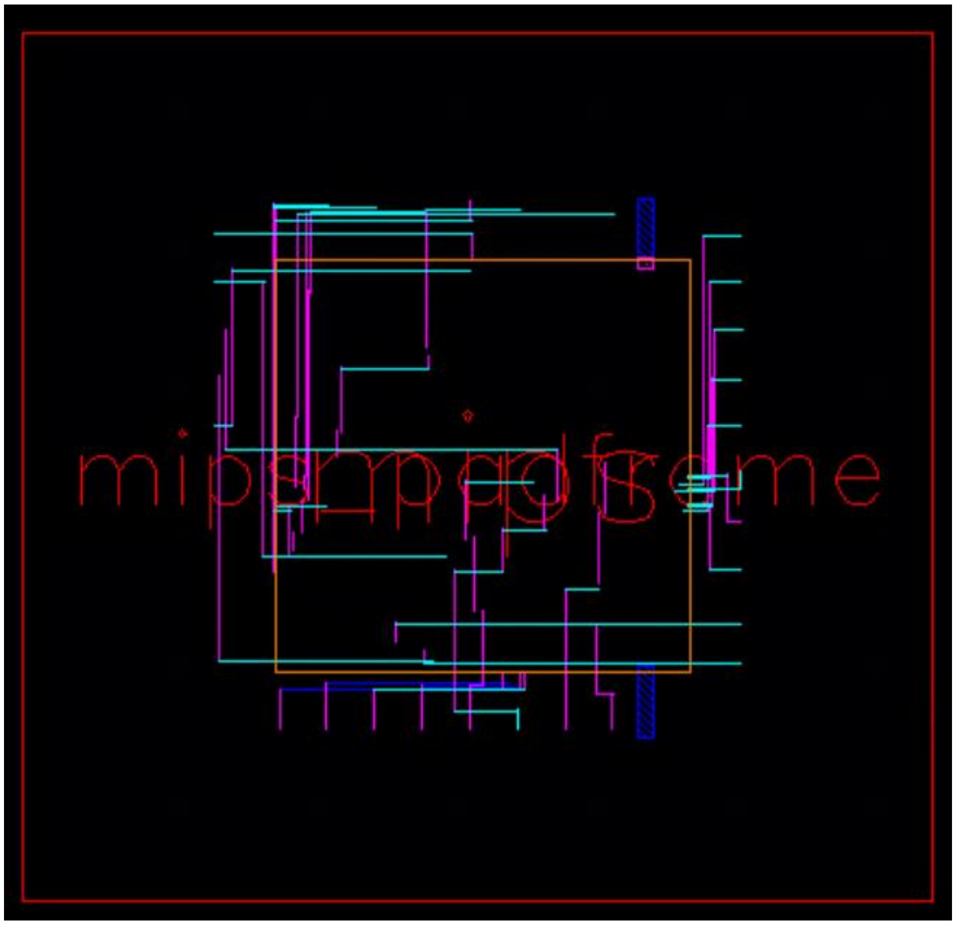

Generated Schematic of the Connected Modules



This screenshot from the CCAR window shows all the connections that need to be completed during the autoroute. The power and ground lines were done manually, making sure to use the appropriate metal width. Then after executing the do.do file (Appendix F) and setting the rules, we ran the autorouter.

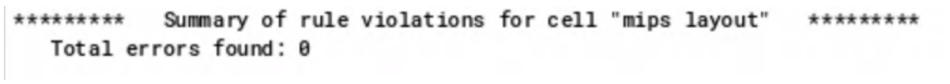


The autoroute results are shown here. The process was executed quickly and made all the correct connections as set up in the beginning.



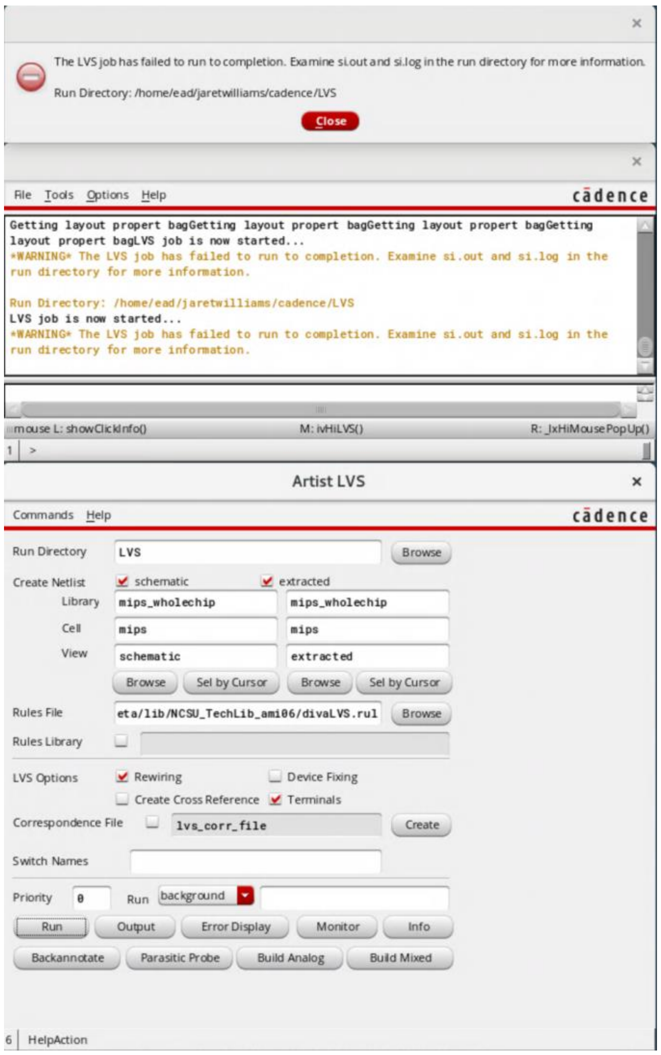
This view helps visualize the connections that were routed during the process.

DRC/LVS



```
*********    Summary of rule violations for cell "mips layout"    *********
   Total errors found: 0
```

DRC passed successfully on all the different modules.



The LVS job has failed to run to completion. Examine si.out and si.log in the run directory for more information.

Run Directory: /home/ead/jaretwilliams/cadence/LVS

```
Getting layout propert bagGetting layout propert bagGetting layout propert bagGetting
layout propert bagLVS job is now started...
*WARNING* The LVS job has failed to run to completion. Examine si.out and si.log in the
run directory for more information.

Run Directory: /home/ead/jaretwilliams/cadence/LVS
LVS job is now started...
*WARNING* The LVS job has failed to run to completion. Examine si.out and si.log in the
run directory for more information.
```

mouse L: showClickInfo() M: ivHiLVS() R: _lxHiMousePopUp()

Artist LVS

Run Directory	LVS	
Create Netlist	[✓ schematic]	[✓ extracted]
Library	mips_wholechip	mips_wholechip
Cell	mips	mips
View	schematic	extracted
Rules File	eta/lib/NCSU_TechLib_ami06/divaLVS.rul	
Rules Library	☐	
LVS Options	[✓ Rewiring]	☐ Device Fixing
	☐ Create Cross Reference	[✓ Terminals]
Correspondence File	☐ lvs_corr_file	
Switch Names		
Priority	0	Run: background

However, we encountered challenges with the LVS and was unable to run the check.

VII. Enhancements

Due to time constraints from completing the base project and other final assignments, the team was not able to implement any enhancements to the tiny MIPS CPU.

VIII. Problems/Debugging

As all projects go, the team ran into a few issues finalizing the CPU and the steps leading to the final product. Initially the team ran into some compiling errors with the Verilog code but that was an easy fix because most of it was just syntax errors or typos. The next issue occurred when running the MIPS CPU Verilog test bench and the output was incorrect. So, the team went back and re-tested the major components of the CPU to find that the Datapath had an issue with how the ir flip flops with labeled. Once that issue was resolved the MIPS CPU Verilog test bench outputted the correct simulation. Now that the Verilog issues have been resolved, the next issue arose when manually inserting the scan cells into the D Flip Flops. When manually inserting the scan cells, the team forgot to run the command that writes the mip_scan.sdc file so once the team recognized the issue, they ran the dc_test.tcl script (Appendix A) and tmax_atpg.tcl script (Appendix D).

The team was very fortunate to have a smooth process with the HDL, synthesis, and DFT processes of the ASIC design flow, the same cannot be said about the place and route step. The first issue arose when the team attempted to use the default.view file from lab 7 without changing the contents to align with the final project, once the team recognized this the quick change was made and no problems arose until place the layout into the pad frame. The biggest issue that arose when placing the layout in the padframe was getting the software to recognize the connections that needed to be made during the CCAR process. At first, the cells for the pad frame and MIPS were in different libraries and this caused CCAR to be unable to match the pins and create the connections. The files were all copied to a single whole chip library which was used a single location for all the files used for the creation of the overall layout. Once this was done, the CCAR process was restarted, and the team was immediately able to see the white lines on the display representing the remaining connections needed. Finally, the autorouter generated these connections and provided us with the final chip.

Commented [NP2]: Jaret, can you finish this paragraph with the issues you had with the pad frame

Commented [JW3R2]: Got it

IX. Conclusions

Final Stats:

- Pin count: 29
- Power draw: 1.9390 mW
- Total Area: 304,542 um
- Processor Speed: 2.7 MHz

Note that the final number of transistors was not included to do the team being unable to get an LVS report of the design.

Overall, this was a successful project that allowed the team to apply knowledge of the full ASIC design flow practiced in lab and the testing procedures learned in lecture, which is an experience that one cannot find in the field anymore. The team put a lot of work in this project and were able to produce a working Tiny MIPS CPU that met the project specifications. This work can be directly applied to real world work in the field and was a very valuable experience for both members of the team.

With more time the team would have implemented an enhancement to improve and streamline the Tiny MIPS CPU. This would have allowed the team to have a more unique but difficult project that they could have marketed to employers and stand out a little more from the rest of the class who did not do an enhancement. Also if GW IT was able to fix the issue with the scan cell D Flip Flops library, the team would be able to have the LVS and have a full transistor count.

Appendix A

Synthesis scripts

Dc_syn.tcl

```
##################################################################

#### Design Compiler Script for ECE 128

#### Performs Synthesis only to AMI .5 technology

#### author: wgibb

#### note: this is a TCL script

#### modified from work done by tjf and eb

##################################################################


####################################

# ITEMS YOU WILL NEED TO SET FOR EACH DESIGN

# 1) myFiles - LIST OF YOUR FILES TO SYNTHESIZE

# 2) basename - TOP LEVEL MODULE IN YOUR DESIGN

# 3) myClk - NAME OF YOUR CLOCK SIGNAL

# 4) virtual - USE A REAL CLOCK (SEQUENTIAL DESIGNS) OR A VIRTUAL

#           CLOCK (COMBINATORIAL DESIGNS)

# 5) myPeriod - SETS THE CLOCK SPEED, THUS DEFINING THE SYNTHESIS SPEED GOAL

####################################


# list of all HDL files in the design

set myFiles [list ./src/mips.v ./src/controller.v ./src/alucontrol.v ./src/datapath.v ] ;
set basename mips         ;# Top-level module name
set myClk clk                    ;# The name of your clock
set virtual 1                       ;# 1 if virtual clock, 0 if real clock
set myPeriod_ns 40               ;# desired clock period (in ns) (sets speed goal)

####################################
# Some runtime options, change only if needed
set runname syn                ;# Name appended to output files
set exit_dc 0                 ;# 1 to exit DC after running, 0 to keep DC running
# set the target library
set target_library [list osu05_stdcells.db] ;
####################################

####################################
# Control the writing of result files
####################################
set verbose 0                 ;# 1 Write reports to screen, 0 do not write reports to screen

####################################
# Timing and loading information
####################################

set myClkLatency_ns 0.3       ; # clock network latency
set myInDelay_ns 2.0              ;# delay from clock to inputs valid
set myOutDelay_ns 1.65             ;# delay from clock to output valid
set myInputBuf INVX1           ;# name of cell driving the inputs
set myLoadLibrary [file rootname $target_library]    ;# name of library the cell comes from
set myLoadPin A             ;# name of pin that the outputs drive
set myMaxFanout   1     ;# max fanout load for input pins
set myOutputLoad 0.1    ;# output pin loading

##################
# compiler switches...
##################
set optimizeArea 1                ;# 1 for area, 0 for speed
set useUltra 0                      ;# 1 for compile_ultra, 0 for compile
                                     # mapEffort, useUngroup are for
                                     # non-ultra compile...
set useUngroup 0                    ;# 0 if no flatten, 1 if flatten

####################################
# Set some system-level things that RARELY change...
####################################
# synthetic_library is set in .synopsys_dc.setup to be
# the dw_foundation library.
set link_library [concat  [concat  "*" $target_library] $synthetic_library]
```

```
####################################
set fileFormat verilog              ;# verilog or VHDL

##############################################################
##############################################################
### YOU SHOULD NOT NEED TO CHANGE ANYTHING BELOW THIS LINE ###
##############################################################
##############################################################

##############################################################
#### read in, link to standard cells, and uniquify design ####
##############################################################

####################################
# remove any other designs from design compiler's memory
####################################
remove_design -all

echo IMPORTING DESIGN
####################################
# analyzer & elaborate verilog source files
####################################
analyze -format $fileFormat -lib WORK $myFiles
```

```
elaborate $basename -lib WORK -update

####################################
# set design to 'highest' module level
####################################
current_design $basename

####################################
# link to standard cell libraries and uniquify
####################################
link
uniquify

####################################################
#### setup clock & all input/output constraints ####
####################################################
echo SETTING CONSTRAINTS

####################################
# now you can create clocks for the design
# and set other constraints
####################################
if {  $virtual == 0 } {
   create_clock -period $myPeriod_ns $myClk
} else {
   create_clock -period $myPeriod_ns -name $myClk
}
set_clock_latency $myClkLatency_ns $myClk
####################################
# set delays on all inputs & outputs with respect to the clock (in ns)
# set the input and output delay relative to myClk
####################################
if {  $virtual == 0 } {
    set_input_delay $myInDelay_ns -clock $myClk [all_inputs]
} else {
    set_input_delay $myInDelay_ns -clock $myClk [remove_from_collection [all_inputs] $myClk]
}
set_output_delay $myOutDelay_ns -clock $myClk [all_outputs]
####################################
# Set the driving cell for all inputs except the clock
# The clock has infinite drive by default. This is usually
# what you want for synthesis because you will use other
# tools (like SOC Encounter) to build the clock tree
# (or define it by hand).
####################################
if {  $virtual == 0 } {
    set_driving_cell  -library $myLoadLibrary -lib_cell $myInputBuf [all_inputs]
} else {
   set_driving_cell  -library $myLoadLibrary -lib_cell $myInputBuf [remove_from_collection [all_inputs]
$myClk]
}
####################################
# set load/fanin/fanout for all inputs/outputs
####################################
set_load $myOutputLoad [all_outputs]

####################################
# check value of fanout
####################################
set_max_fanout $myMaxFanout [all_inputs]
set_fanout_load 8 [all_outputs]

echo DONE SETTING CONSTRAINTS

####################################
# This command will fix the problem of having
# assign statements left in your structural file.
# But, it will insert pairs of inverters for feedthroughs!
set_fix_multiple_port_nets -all -buffer_constants
```

```
####################################


echo BEGIN COMPILING DESIGN

####################################

# optimize for area

####################################

if { $optimizeArea == 1} {

      set_max_area 0

}

####################################

# now compile the design with given mapping effort

# and do a second compile with incremental mapping

# or use the compile_ultra meta-command

####################################

if {  $useUltra == 1 } {

   compile_ultra

} else {

   if {  $useUngroup == 1 } {

      compile -ungroup_all -map_effort medium

  } else {

      compile -map_effort medium -exact_map

  }

}
```

```
check_design

echo VIOLATIONS

report_constraint -all_violators


#####################################################

#### generate verilog code for synthesized module ###

#### sdc files, sdf files, design compiler project###

#### and write out reports                  ###

#####################################################

echo OUTPUT FILES AND REPORTS

set filebase [format "%s%s" [format "%s%s" $basename "_"] $runname]


####################################

# structural (synthesized) file as verilog

####################################

set filename [format "%s%s%s" ./src/ $filebase ".v"]

redirect change_names { change_names -rules verilog -hierarchy -verbose }

write -format verilog -hierarchy -output $filename


####################################

# write out the sdf file for back-annotated verilog sim

# This file can be large!

####################################
```

```
set filename [format "%s%s%s" ./src/ $filebase ".sdf"]
write_sdf -version 1.0 $filename

####################################
# this is the timing constraints file generated from the
# conditions above - used in the place and route program
####################################
set filename [format "%s%s%s" ./src/ $filebase ".sdc"]
write_sdc $filename

####################################
# generate reports for user to view
####################################

if { $verbose == 1 } {
      report_design
      report_hierarchy
      report_timing -path full -delay max -nworst 3 -significant_digits 2 -sort_by group
      report_timing -path full -delay min -nworst 3 -significant_digits 2 -sort_by group
      report_area
      report_cell
      report_net
      report_port -v
      report_power -analysis_effort low
}

# Design and Hierarchy reports
set filename [format "%s%s%s" ./reports/ $filebase ".design"]
redirect $filename { report_design }
set filename [format "%s%s%s" ./reports/ $filebase ".design"]
redirect -append $filename { report_hierarchy }

# Timing reports
set filename [format "%s%s%s" ./reports/ $filebase ".timing"]
redirect $filename { report_timing -path full -delay max -nworst 5 -significant_digits 2 -sort_by group }
set filename [format "%s%s%s" ./reports/ $filebase ".timing"]
redirect -append $filename { report_timing -path full -delay min -nworst 5 -significant_digits 2 -sort_by
group }

# Report_cell and report_area
set filename [format "%s%s%s" ./reports/ $filebase ".area"]
redirect $filename { report_area }
set filename [format "%s%s%s" ./reports/ $filebase ".area"]
redirect -append $filename { report_cell }

# Report port
```

```
set filename [format "%s%s%s" ./reports/ $filebase ".ports"]
redirect $filename { report_port -v}

#report net
set filename [format "%s%s%s" ./reports/ $filebase ".net"]
redirect $filename { report_net }

# report power
set filename [format "%s%s%s" ./reports/ $filebase ".pow"]
redirect $filename { report_power -analysis_effort low }

####################################
# quit dc
####################################
if { $exit_dc == 1} {
      exit
}
```

Dc_test.tcl

```
##################################################################
#### Design Compiler Script for ECE 128
#### Performs Synthesis only to AMI .5 technology
#### author: wgibb
```

```
#### note: this is a TCL script
#### modified from work done by tjf and eb
##################################################################
####################################
# ITEMS YOU WILL NEED TO SET FOR EACH DESIGN
# 1) myFiles - LIST OF YOUR FILES TO SYNTHESIZE
# 2) basename - TOP LEVEL MODULE IN YOUR DESIGN
# 3) myClk - NAME OF YOUR CLOCK SIGNAL
# 4) virtual - USE A REAL CLOCK (SEQUENTIAL DESIGNS) OR A VIRTUAL
#           CLOCK (COMBINATORIAL DESIGNS)
# 5) myPeriod - SETS THE CLOCK SPEED, THUS DEFINING THE SYNTHESIS SPEED GOAL
####################################

# list of all HDL files in the design
set myFiles [list ./src/mips.v ./src/controller.v ./src/alucontrol.v ./src/datapath.v] ;
set basename mips       ;# Top-level module name
set myClk clk                    ;# The name of your clock
set virtual 0                       ;# 1 if virtual clock, 0 if real clock
set myPeriod_ns 40               ;# desired clock period (in ns) (sets speed goal)

####################################
# Some runtime options, change only if needed
set runname syn                ;# Name appended to output files
set exit_dc 0                 ;# 1 to exit DC after running, 0 to keep DC running
# set the target library
set target_library [list osu05_stdcells.db] ;
####################################

####################################
# Control the printing of result files
####################################
set verbose 0                 ;# 1 Write reports to screen, 0 do not write reports to screen
set verbose_dft 0       ;# 1 Write reports to screen, 0 do not write reports to screen
####################################
# Timing and loading information
####################################

set myClkLatency_ns 0.3       ; # clock network latency
set myInDelay_ns 2.0              ;# delay from clock to inputs valid
set myOutDelay_ns 1.65             ;# delay from clock to output valid
set myInputBuf INVX1           ;# name of cell driving the inputs
set myLoadLibrary [file rootname $target_library]    ;# name of library the cell comes from
set myLoadPin A             ;# name of pin that the outputs drive
set myMaxFanout   1     ;# max fanout load for input pins
set myOutputLoad 0.1    ;# output pin loading
```

```
##################
# compiler switches...
##################
set optimizeArea 1                ;# 1 for area, 0 for speed
set useUltra 0                      ;# 1 for compile_ultra, 0 for compile
                                     # mapEffort, useUngroup are for
                                     # non-ultra compile...
set useUngroup 0                    ;# 0 if no flatten, 1 if flatten


####################################
# DFT Switches                    #
####################################
set dft_runname scan             ; # name appended to output files
set scan_library [list osu_scan.db] ; # Library with scan chain cells
set scancell      DFFPOSX1_SCAN ; # Name of ScanFF Cell

# Setup timing variables for dft_drc command

set test_default_delay 0      ; # define time when values are applied to input ports
set test_default_bidir_delay 0      ; # Defines  the  default switching time of bidirectional
                          # ports in a tester cycle.
set test_default_strobe 40    ; # default strobe time in a test cycle for output ports
```

```
                                # and bidirectional ports in output mode
set test_default_period 100   ; # Defines the default length of a test vector cycle

# Setup scan chain for insert_dft

set test_default_scan_style multiplexed_flip_flop;
# Defines  the  default scan style for the insert_dft command.
# type "man test_default_scan_style" for more information

####################################
# Set some system-level things that RARELY change...
####################################
# synthetic_library is set in .synopsys_dc.setup to be
# the dw_foundation library.
set link_library [concat  [concat  "*" $target_library] $synthetic_library]
####################################
set fileFormat verilog              ;# verilog or VHDL

###############################################################
###############################################################
### YOU SHOULD NOT NEED TO CHANGE ANYTHING BELOW THIS LINE ###
###############################################################
###############################################################
```

```
##############################################################
#### read in, link to standard cells, and uniquify design ####
##############################################################

####################################
# remove any other designs from design compiler's memory
####################################
remove_design -all

echo IMPORTING DESIGN
####################################
# analyzer & elaborate verilog source files
####################################
analyze -format $fileFormat -lib WORK $myFiles
elaborate $basename -lib WORK -update

####################################
# set design to 'highest' module level
####################################
current_design $basename

####################################
```

```
# link to standard cell libraries and uniquify
####################################
link
uniquify

####################################################
#### setup clock & all input/output constraints ####
####################################################
echo SETTING CONSTRAINTS

####################################
# now you can create clocks for the design
# and set other constraints
####################################
if {  $virtual == 0 } {
   create_clock -period $myPeriod_ns $myClk
} else {
   create_clock -period $myPeriod_ns -name $myClk
}
set_clock_latency $myClkLatency_ns $myClk
####################################
# set delays on all inputs & outputs with respect to the clock (in ns)
# set the input and output delay relative to myClk
```

```
####################################
if {  $virtual == 0 } {
    set_input_delay $myInDelay_ns -clock $myClk [all_inputs]
} else {
    set_input_delay $myInDelay_ns -clock $myClk [remove_from_collection [all_inputs] $myClk]
}
set_output_delay $myOutDelay_ns -clock $myClk [all_outputs]
####################################
# Set the driving cell for all inputs except the clock
# The clock has infinite drive by default. This is usually
# what you want for synthesis because you will use other
# tools (like SOC Encounter) to build the clock tree
# (or define it by hand).
####################################
if {  $virtual == 0 } {
    set_driving_cell  -library $myLoadLibrary -lib_cell $myInputBuf [all_inputs]
} else {
   set_driving_cell  -library $myLoadLibrary -lib_cell $myInputBuf [remove_from_collection [all_inputs]
$myClk]
}
####################################
# set load/fanin/fanout for all inputs/outputs
####################################
```

```
set_load $myOutputLoad [all_outputs]

####################################
# check value of fanout
####################################
set_max_fanout $myMaxFanout [all_inputs]
set_fanout_load 8 [all_outputs]

echo DONE SETTING CONSTRAINTS

####################################
# This command will fix the problem of having
# assign statements left in your structural file.
# But, it will insert pairs of inverters for feedthroughs!
set_fix_multiple_port_nets -all -buffer_constants
####################################

echo BEGIN COMPILING DESIGN
####################################
# optimize for area
####################################
if { $optimizeArea == 1} {
      set_max_area 0
}
####################################
# now compile the design with given mapping effort
# and do a second compile with incremental mapping
# or use the compile_ultra meta-command
####################################
if {  $useUltra == 1 } {
   compile_ultra
} else {
   if {  $useUngroup == 1 } {
      compile -ungroup_all -map_effort medium
  } else {
      compile -map_effort medium -exact_map
  }
}
check_design
echo VIOLATIONS
report_constraint -all_violators

#####################################################
#### generate verilog code for synthesized module ###
#### sdc files, sdf files, design compiler project###
#### and write out reports                  ###
```

```
#####################################################
echo OUTPUT FILES AND REPORTS
set filebase [format "%s%s" [format "%s%s" $basename "_"] $runname]

####################################
# structural (synthesized) file as verilog
####################################
set filename [format "%s%s%s" ./src/ $filebase ".v"]
redirect change_names { change_names -rules verilog -hierarchy -verbose }
write -format verilog -hierarchy -output $filename

####################################
# write out the sdf file for back-annotated verilog sim
# This file can be large!
####################################
set filename [format "%s%s%s" ./src/ $filebase ".sdf"]
write_sdf -version 1.0 $filename

####################################
# this is the timing constraints file generated from the
# conditions above - used in the place and route program
####################################
set filename [format "%s%s%s" ./src/ $filebase ".sdc"]
```

```
write_sdc $filename

####################################
# generate reports for user to view
####################################

if { $verbose == 1 } {
      report_design
      report_hierarchy
      report_timing -path full -delay max -nworst 3 -significant_digits 2 -sort_by group
      report_timing -path full -delay min -nworst 3 -significant_digits 2 -sort_by group
      report_area
      report_cell
      report_net
      report_port -v
      report_power -analysis_effort low
}

# Design and Hierarchy reports
set filename [format "%s%s%s" ./reports/ $filebase ".design"]
redirect $filename { report_design }
set filename [format "%s%s%s" ./reports/ $filebase ".design"]
redirect -append $filename { report_hierarchy }
```

```
# Timing reports

set filename [format "%s%s%s" ./reports/ $filebase ".timing"]

redirect $filename { report_timing -path full -delay max -nworst 5 -significant_digits 2 -sort_by group }

set filename [format "%s%s%s" ./reports/ $filebase ".timing"]

redirect -append $filename { report_timing -path full -delay min -nworst 5 -significant_digits 2 -sort_by
group }


# Report_cell and report_area

set filename [format "%s%s%s" ./reports/ $filebase ".area"]

redirect $filename { report_area }

set filename [format "%s%s%s" ./reports/ $filebase ".area"]

redirect -append $filename { report_cell }


# Report port

set filename [format "%s%s%s" ./reports/ $filebase ".ports"]

redirect $filename { report_port -v}


#report net

set filename [format "%s%s%s" ./reports/ $filebase ".net"]

redirect $filename { report_net }


# report power
```

```
set filename [format "%s%s%s" ./reports/ $filebase ".pow"]
redirect $filename { report_power -analysis_effort low }

##########################################
#### Insert Test Structures ###
##########################################
# Update filebase
set filebase [format "%s%s" [format "%s%s" $basename "_"] $dft_runname]
# Update target library
set target_library [list $target_library $scan_library]
# Set the scan cells to use in the design
#set_scan_register_type -type {DFFPOSX1_SCAN} ;
set_scan_register_type -type ${scancell} ;

# Make sure to add a test_out port
set_scan_configuration -create_dedicated_scan_out_ports true

# Infer clock and reset lines
create_test_protocol -infer_async -infer_clock

dft_drc -verbose

# Replace flip flops with multiplexed flipflops
```

```
compile -scan

# Check for constraint violations
report_constraint -all_violators

#########################################
### Building Scan Chains              ###
#########################################

# connects all scan-enabled ff's together into scan-chain
# note, it creates two new ports: test_si & test_se
insert_dft

# set drive strength of the test ports to 2 (so it isn't assumed to be infinite)
set_drive 2 test_si
set_drive 2 test_se

# since you've already inserted scan-ff's, we don't want that to happen again,
# when we run insert_dft
set_scan_configuration -replace false

# run insert_scan again to set drive-strength constraints
insert_dft
```

```
# report any constraints that may have been violated by inserting the test
# structures

if { $verbose_dft == 1 } {
      report_constraint -all_violators
      dft_drc -verbose -coverage_estimate
      report_scan_path -view existing -chain all
      report_cell
}

# report dft_drc
set filename [format "%s%s%s" ./reports/ $filebase ".violators"]
redirect $filename { report_constraint -all_violators }

# report dft_drc
set filename [format "%s%s%s" ./reports/ $filebase ".dft_drc"]
redirect $filename { dft_drc -verbose -coverage_estimate }

# report scan path
set filename [format "%s%s%s" ./reports/ $filebase ".scan_path"]
redirect $filename { report_scan_path -view existing -chain all }
```

```
# report cells
set filename [format "%s%s%s" ./reports/ $filebase ".cell"]
redirect $filename { report_cell }

# Write out protocol
set filename [format "%s%s%s" ./src/ $filebase ".spf"]
write_test_protocol -output $filename

# Write out scan chain design
set filename [format "%s%s%s" ./src/ $filebase ".v"]
redirect change_names { change_names -rules verilog -hierarchy -verbose }
write -format verilog -hierarchy -output $filename

####################################
# this is the timing constraints file generated from the
# conditions above - used in the place and route program
####################################
set filename [format "%s%s%s" ./src/ $filebase ".sdc"]
write_sdc $filename

####################################
# quit dc
####################################
```

```
if { $exit_dc == 1} {
      exit
}
```

Appendix B

Synthesis Reports

Mips_scan.cell

```
****************************************
Report : constraint
        -all_violators
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 13:27:25 2022
****************************************


   max_area

                             Required        Actual
   Design                        Area          Area            Slack
   -----------------------------------------------------------------
   mips                            0.00       362835.00      -362835.00
                                                                        (VIOLATED)
```

```
1
```

Mips_scan.dft_drc

```
In mode: Internal_scan...

  Design has scan chains in this mode

  Design is scan routed

  Post-DFT DRC enabled


Information: Starting test design rule checking. (TEST-222)

  Loading test protocol

  ...basic checks...

  ...basic sequential cell checks...

  ...checking vector rules...

  ...checking clock rules...

  ...checking scan chain rules...

  ...checking scan compression rules...

  ...checking X-state rules...

  ...checking tristate rules...

  ...extracting scan details...


-----------------------------------------------------------------
```

```
  DRC Report

  Total violations: 0

-----------------------------------------------------------------



Test Design rule checking did not find violations


-----------------------------------------------------------------
  Sequential Cell Report

  0 out of 132 sequential cells have violations

-----------------------------------------------------------------

SEQUENTIAL CELLS WITHOUT VIOLATIONS
      * 132 cells are valid scan cells
         dp/areg/q_reg_7_
         dp/areg/q_reg_6_
         dp/areg/q_reg_5_
         dp/areg/q_reg_4_
         dp/areg/q_reg_3_
```

```
dp/areg/q_reg_2_
dp/areg/q_reg_1_
dp/areg/q_reg_0_
dp/ir0/q_reg_7_
dp/ir0/q_reg_6_
dp/ir0/q_reg_5_
dp/ir0/q_reg_4_
dp/ir0/q_reg_3_
dp/ir0/q_reg_2_
dp/ir0/q_reg_1_
dp/ir0/q_reg_0_
dp/ir1/q_reg_7_
dp/ir1/q_reg_6_
dp/ir1/q_reg_5_
dp/ir1/q_reg_4_
dp/ir1/q_reg_3_
dp/ir1/q_reg_2_
dp/ir1/q_reg_1_
dp/ir1/q_reg_0_
dp/ir2/q_reg_7_
dp/ir2/q_reg_6_
dp/ir2/q_reg_5_
dp/ir2/q_reg_4_
```

```
dp/ir2/q_reg_3_

dp/ir2/q_reg_2_

dp/ir2/q_reg_1_

dp/ir2/q_reg_0_

dp/ir3/q_reg_7_

dp/ir3/q_reg_6_

dp/ir3/q_reg_5_

dp/ir3/q_reg_4_

dp/ir3/q_reg_3_

dp/ir3/q_reg_2_

dp/ir3/q_reg_1_

dp/ir3/q_reg_0_

dp/mdr/q_reg_7_

dp/mdr/q_reg_6_

dp/mdr/q_reg_5_

dp/mdr/q_reg_4_

dp/mdr/q_reg_3_

dp/mdr/q_reg_2_

dp/mdr/q_reg_1_

dp/mdr/q_reg_0_

dp/pcreg/q_reg_7_

dp/pcreg/q_reg_6_

dp/pcreg/q_reg_5_
```

```
dp/pcreg/q_reg_4_
dp/pcreg/q_reg_3_
dp/pcreg/q_reg_2_
dp/pcreg/q_reg_1_
dp/pcreg/q_reg_0_
dp/res/q_reg_7_
dp/res/q_reg_6_
dp/res/q_reg_5_
dp/res/q_reg_4_
dp/res/q_reg_3_
dp/res/q_reg_2_
dp/res/q_reg_1_
dp/res/q_reg_0_
dp/rf/RAM_reg_7__7_
dp/rf/RAM_reg_7__6_
dp/rf/RAM_reg_7__5_
dp/rf/RAM_reg_7__4_
dp/rf/RAM_reg_7__3_
dp/rf/RAM_reg_7__2_
dp/rf/RAM_reg_7__1_
dp/rf/RAM_reg_7__0_
dp/rf/RAM_reg_6__7_
dp/rf/RAM_reg_6__6_
```

```
dp/rf/RAM_reg_6__5_
dp/rf/RAM_reg_6__4_
dp/rf/RAM_reg_6__3_
dp/rf/RAM_reg_6__2_
dp/rf/RAM_reg_6__1_
dp/rf/RAM_reg_6__0_
dp/rf/RAM_reg_5__7_
dp/rf/RAM_reg_5__6_
dp/rf/RAM_reg_5__5_
dp/rf/RAM_reg_5__4_
dp/rf/RAM_reg_5__3_
dp/rf/RAM_reg_5__2_
dp/rf/RAM_reg_5__1_
dp/rf/RAM_reg_5__0_
dp/rf/RAM_reg_4__7_
dp/rf/RAM_reg_4__6_
dp/rf/RAM_reg_4__5_
dp/rf/RAM_reg_4__4_
dp/rf/RAM_reg_4__3_
dp/rf/RAM_reg_4__2_
dp/rf/RAM_reg_4__1_
dp/rf/RAM_reg_4__0_
dp/rf/RAM_reg_3__7_
```

```
dp/rf/RAM_reg_3__6_
dp/rf/RAM_reg_3__5_
dp/rf/RAM_reg_3__4_
dp/rf/RAM_reg_3__3_
dp/rf/RAM_reg_3__2_
dp/rf/RAM_reg_3__1_
dp/rf/RAM_reg_3__0_
dp/rf/RAM_reg_2__7_
dp/rf/RAM_reg_2__6_
dp/rf/RAM_reg_2__5_
dp/rf/RAM_reg_2__4_
dp/rf/RAM_reg_2__3_
dp/rf/RAM_reg_2__2_
dp/rf/RAM_reg_2__1_
dp/rf/RAM_reg_2__0_
dp/rf/RAM_reg_1__7_
dp/rf/RAM_reg_1__6_
dp/rf/RAM_reg_1__5_
dp/rf/RAM_reg_1__4_
dp/rf/RAM_reg_1__3_
dp/rf/RAM_reg_1__2_
dp/rf/RAM_reg_1__1_
dp/rf/RAM_reg_1__0_
```

```
         dp/wrd/q_reg_7_
         dp/wrd/q_reg_6_
         dp/wrd/q_reg_5_
         dp/wrd/q_reg_4_
         dp/wrd/q_reg_3_
         dp/wrd/q_reg_2_
         dp/wrd/q_reg_1_
         dp/wrd/q_reg_0_
         cont/state_reg_0_
         cont/state_reg_3_
         cont/state_reg_2_
         cont/state_reg_1_

....Inferring feed-through connections....
Information: Test design rule checking completed. (TEST-123)
  Running test coverage estimation...
 5930 faults were added to fault list.
 ATPG performed for stuck fault model using internal pattern source.
 ----------------------------------------------------------
 #patterns     #faults     #ATPG faults  test      process
 stored     detect/active  red/au/abort  coverage  CPU time
 ---------  -------------  ------------  --------  --------
 Begin deterministic ATPG: #uncollapsed_faults=4964, abort_limit=10...
```

```
0            4178    786         0/0/2    86.68%      0.01
0             441    344         1/0/2    94.17%      0.01
0             206    137         2/0/2    97.68%      0.03
0              57     74         7/0/4    98.74%      0.03
0              66      4        10/0/4    99.93%      0.03
0               4      0        10/0/4   100.00%      0.03

           Pattern Summary Report
-----------------------------------------------
#internal patterns                           0
-----------------------------------------------


    Uncollapsed Stuck Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT       5891
Possibly detected                PT          0
Undetectable                     UD         39
ATPG untestable                  AU          0
Not detected                     ND          0
-----------------------------------------------
```

```
 total faults                              5930
 test coverage                           100.00%
 -----------------------------------------------
  Information: The test coverage above may be inferior
               than the real test coverage with customized
               protocol and test simulation library.
```

Mips_scan.scan_path

```
****************************************
Report : Scan path
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 13:27:43 2022
****************************************

========================================
TEST MODE: Internal_scan
VIEW     : Existing DFT
========================================

========================================
AS SPECIFIED BY USER
```

```
========================================


========================================
AS BUILT BY insert_dft
========================================

Scan_path    Len   ScanDataIn  ScanDataOut ScanEnable  MasterClock SlaveClock
-----------  ----- ----------- ----------- ----------- ----------- -----------
I 1          132   test_si     test_so     test_se     clk         -

1
```

Mips_scan.violators

```
In mode: Internal_scan...
  Design has scan chains in this mode
  Design is scan routed
  Post-DFT DRC enabled

Information: Starting test design rule checking. (TEST-222)
  Loading test protocol
  ...basic checks...
  ...basic sequential cell checks...
  ...checking vector rules...
  ...checking clock rules...
  ...checking scan chain rules...
  ...checking scan compression rules...
  ...checking X-state rules...
  ...checking tristate rules...
  ...extracting scan details...

-----------------------------------------------------------------
  DRC Report

  Total violations: 0

-----------------------------------------------------------------


Test Design rule checking did not find violations

-----------------------------------------------------------------
  Sequential Cell Report

  0 out of 132 sequential cells have violations
```

```
-----------------------------------------------------------------

SEQUENTIAL CELLS WITHOUT VIOLATIONS
      * 132 cells are valid scan cells
         dp/areg/q_reg_7_
         dp/areg/q_reg_6_
         dp/areg/q_reg_5_
         dp/areg/q_reg_4_
         dp/areg/q_reg_3_
         dp/areg/q_reg_2_
         dp/areg/q_reg_1_
         dp/areg/q_reg_0_
         dp/ir0/q_reg_7_
         dp/ir0/q_reg_6_
         dp/ir0/q_reg_5_
         dp/ir0/q_reg_4_
         dp/ir0/q_reg_3_
         dp/ir0/q_reg_2_
         dp/ir0/q_reg_1_
         dp/ir0/q_reg_0_
         dp/ir1/q_reg_7_
         dp/ir1/q_reg_6_
         dp/ir1/q_reg_5_
```

```
dp/ir1/q_reg_4_
dp/ir1/q_reg_3_
dp/ir1/q_reg_2_
dp/ir1/q_reg_1_
dp/ir1/q_reg_0_
dp/ir2/q_reg_7_
dp/ir2/q_reg_6_
dp/ir2/q_reg_5_
dp/ir2/q_reg_4_
dp/ir2/q_reg_3_
dp/ir2/q_reg_2_
dp/ir2/q_reg_1_
dp/ir2/q_reg_0_
dp/ir3/q_reg_7_
dp/ir3/q_reg_6_
dp/ir3/q_reg_5_
dp/ir3/q_reg_4_
dp/ir3/q_reg_3_
dp/ir3/q_reg_2_
dp/ir3/q_reg_1_
dp/ir3/q_reg_0_
dp/mdr/q_reg_7_
dp/mdr/q_reg_6_
```

```
dp/mdr/q_reg_5_

dp/mdr/q_reg_4_

dp/mdr/q_reg_3_

dp/mdr/q_reg_2_

dp/mdr/q_reg_1_

dp/mdr/q_reg_0_

dp/pcreg/q_reg_7_

dp/pcreg/q_reg_6_

dp/pcreg/q_reg_5_

dp/pcreg/q_reg_4_

dp/pcreg/q_reg_3_

dp/pcreg/q_reg_2_

dp/pcreg/q_reg_1_

dp/pcreg/q_reg_0_

dp/res/q_reg_7_

dp/res/q_reg_6_

dp/res/q_reg_5_

dp/res/q_reg_4_

dp/res/q_reg_3_

dp/res/q_reg_2_

dp/res/q_reg_1_

dp/res/q_reg_0_

dp/rf/RAM_reg_7__7_
```

```
dp/rf/RAM_reg_7__6_
dp/rf/RAM_reg_7__5_
dp/rf/RAM_reg_7__4_
dp/rf/RAM_reg_7__3_
dp/rf/RAM_reg_7__2_
dp/rf/RAM_reg_7__1_
dp/rf/RAM_reg_7__0_
dp/rf/RAM_reg_6__7_
dp/rf/RAM_reg_6__6_
dp/rf/RAM_reg_6__5_
dp/rf/RAM_reg_6__4_
dp/rf/RAM_reg_6__3_
dp/rf/RAM_reg_6__2_
dp/rf/RAM_reg_6__1_
dp/rf/RAM_reg_6__0_
dp/rf/RAM_reg_5__7_
dp/rf/RAM_reg_5__6_
dp/rf/RAM_reg_5__5_
dp/rf/RAM_reg_5__4_
dp/rf/RAM_reg_5__3_
dp/rf/RAM_reg_5__2_
dp/rf/RAM_reg_5__1_
dp/rf/RAM_reg_5__0_
```

```
dp/rf/RAM_reg_4__7_
dp/rf/RAM_reg_4__6_
dp/rf/RAM_reg_4__5_
dp/rf/RAM_reg_4__4_
dp/rf/RAM_reg_4__3_
dp/rf/RAM_reg_4__2_
dp/rf/RAM_reg_4__1_
dp/rf/RAM_reg_4__0_
dp/rf/RAM_reg_3__7_
dp/rf/RAM_reg_3__6_
dp/rf/RAM_reg_3__5_
dp/rf/RAM_reg_3__4_
dp/rf/RAM_reg_3__3_
dp/rf/RAM_reg_3__2_
dp/rf/RAM_reg_3__1_
dp/rf/RAM_reg_3__0_
dp/rf/RAM_reg_2__7_
dp/rf/RAM_reg_2__6_
dp/rf/RAM_reg_2__5_
dp/rf/RAM_reg_2__4_
dp/rf/RAM_reg_2__3_
dp/rf/RAM_reg_2__2_
dp/rf/RAM_reg_2__1_
```

```
         dp/rf/RAM_reg_2__0_

         dp/rf/RAM_reg_1__7_

         dp/rf/RAM_reg_1__6_

         dp/rf/RAM_reg_1__5_

         dp/rf/RAM_reg_1__4_

         dp/rf/RAM_reg_1__3_

         dp/rf/RAM_reg_1__2_

         dp/rf/RAM_reg_1__1_

         dp/rf/RAM_reg_1__0_

         dp/wrd/q_reg_7_

         dp/wrd/q_reg_6_

         dp/wrd/q_reg_5_

         dp/wrd/q_reg_4_

         dp/wrd/q_reg_3_

         dp/wrd/q_reg_2_

         dp/wrd/q_reg_1_

         dp/wrd/q_reg_0_

         cont/state_reg_0_

         cont/state_reg_3_

         cont/state_reg_2_

         cont/state_reg_1_


....Inferring feed-through connections....
```

```
Information: Test design rule checking completed. (TEST-123)
  Running test coverage estimation...
 6010 faults were added to fault list.
 ATPG performed for stuck fault model using internal pattern source.
 ----------------------------------------------------------
 #patterns     #faults     #ATPG faults  test      process
 stored     detect/active  red/au/abort  coverage  CPU time
 ---------  -------------  ------------  --------  --------
 Begin deterministic ATPG: #uncollapsed_faults=4964, abort_limit=10...
 0            4148    816         0/0/0    86.36%      0.01
 0             503    313         0/0/0    94.77%      0.01
 0             169    141         2/0/0    97.64%      0.02
 0              81     53         8/0/0    99.11%      0.02
 0              50      1        10/0/0    99.98%      0.02
 0               1      0        10/0/0   100.00%      0.02

            Pattern Summary Report
 -----------------------------------------------
 #internal patterns                           0
 -----------------------------------------------



     Uncollapsed Stuck Fault Summary Report
 -----------------------------------------------
 fault class                     code   #faults
 ------------------------------  ----  ---------
 Detected                         DT       5971
 Possibly detected                PT          0
 Undetectable                     UD         39
 ATPG untestable                  AU          0
 Not detected                     ND          0
 -----------------------------------------------
 total faults                              6010
 test coverage                           100.00%
 -----------------------------------------------
  Information: The test coverage above may be inferior
               than the real test coverage with customized
               protocol and test simulation library.
Current design is 'mips'.
Current design is 'mips'.
Current design is 'mips'.
1
```

Mips_syn.area

```
****************************************
Report : area
```

```
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************

Library(s) Used:

    osu05_stdcells (File: /apps/design_kits/osu_stdcells_v2p7/synopsys/lib/ami05/osu05_stdcells.db)

Number of ports:                          569
Number of nets:                          1407
Number of cells:                          892
Number of combinational cells:            730
Number of sequential cells:               140
Number of macros/black boxes:               0
Number of buf/inv:                        205
Number of references:                       4

Combinational area:             186246.000000
Buf/Inv area:                    30024.000000
Noncombinational area:          120960.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)
```

```
Total cell area:                307206.000000

Total area:                 undefined

1


****************************************

Report : cell

Design : mips

Version: O-2018.06-SP1

Date   : Sat Apr 30 12:55:30 2022

****************************************


Attributes:

    b - black box (unknown)

    h - hierarchical

    n - noncombinational

    p - parameterized

    r - removable

    u - contains unmapped logic


Cell                      Reference       Library             Area  Attributes

--------------------------------------------------------------------------------

U1                        INVX2           osu05_stdcells  144.000000
```

```
U2                        INVX2          osu05_stdcells  144.000000
U3                        INVX2          osu05_stdcells  144.000000
U4                        INVX2          osu05_stdcells  144.000000
U5                        INVX2          osu05_stdcells  144.000000
U6                        INVX2          osu05_stdcells  144.000000
U7                        INVX2          osu05_stdcells  144.000000
U8                        INVX2          osu05_stdcells  144.000000
U9                        INVX2          osu05_stdcells  144.000000
U10                       INVX2          osu05_stdcells  144.000000
U11                       INVX2          osu05_stdcells  144.000000
U12                       INVX2          osu05_stdcells  144.000000
U13                       INVX2          osu05_stdcells  144.000000
U14                       INVX2          osu05_stdcells  144.000000
U15                       INVX2          osu05_stdcells  144.000000
U16                       INVX2          osu05_stdcells  144.000000
U17                       INVX2          osu05_stdcells  144.000000
U18                       INVX2          osu05_stdcells  144.000000
U19                       INVX2          osu05_stdcells  144.000000
U20                       INVX2          osu05_stdcells  144.000000
ac                        alucontrol                     3654.000000
                                                                   h
cont                      controller                     22878.000000
                                                                   h, n
```

```
dp                          datapath_WIDTH8_REGBITS3        277794.000000
                                                                      h, n, p
--------------------------------------------------------------------------------
Total 23 cells                                          307206.000000
1
```

Mips_syn.design

```
****************************************
Report : design
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************


Design allows ideal nets on clock nets.


Library(s) Used:


    osu05_stdcells (File: /apps/design_kits/osu_stdcells_v2p7/synopsys/lib/ami05/osu05_stdcells.db)


Local Link Library:


    {osu05_stdcells.db}
```

```
Flip-Flop Types:

    No flip-flop types specified.

Latch Types:

    No latch types specified.

Operating Conditions:


    Operating Condition Name : typical
    Library : osu05_stdcells
    Process :   1.00
    Temperature :  25.00
    Voltage :   5.00

Wire Loading Model:

    No wire loading specified.
```

```
Wire Loading Model Mode: top.

Timing Ranges:

    No timing ranges specified.

Pin Input Delays:

    None specified.

Pin Output Delays:

    None specified.

Disabled Timing Arcs:

    No arcs disabled.

Required Licenses:

    None Required

Design Parameters:
    None specified.
1

****************************************
Report : hierarchy
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************

mips
    INVX2                                osu05_stdcells
    alucontrol
        INVX2                            osu05_stdcells
        NAND2X1                          osu05_stdcells
        NAND3X1                          osu05_stdcells
        OAI21X1                          osu05_stdcells
    controller
        AND2X2                           osu05_stdcells
        AOI21X1                          osu05_stdcells
        AOI22X1                          osu05_stdcells
        DFFPOSX1                         osu05_stdcells
        INVX2                             osu05_stdcells
        NAND2X1                           osu05_stdcells
        NAND3X1                           osu05_stdcells
        NOR2X1                            osu05_stdcells
        OAI21X1                           osu05_stdcells
        OAI22X1                           osu05_stdcells
        OR2X1                             osu05_stdcells
    datapath_WIDTH8_REGBITS3
        BUFX2                             osu05_stdcells
        INVX2                             osu05_stdcells
        alu_WIDTH8
            AND2X2                        osu05_stdcells
            AOI22X1                       osu05_stdcells
            INVX2                         osu05_stdcells
            NAND2X1                       osu05_stdcells
            NOR2X1                        osu05_stdcells
            OAI21X1                       osu05_stdcells
            OR2X1                         osu05_stdcells
            XNOR2X1                       osu05_stdcells
            XOR2X1                        osu05_stdcells
            alu_WIDTH8_DW01_add_0
                FAX1                      osu05_stdcells
        dff_WIDTH8_0
            DFFPOSX1                      osu05_stdcells
        dff_WIDTH8_1
            DFFPOSX1                      osu05_stdcells
            INVX2                         osu05_stdcells
        dff_WIDTH8_2
            DFFPOSX1                      osu05_stdcells
            INVX2                         osu05_stdcells
        dff_WIDTH8_3
            DFFPOSX1                      osu05_stdcells
        dffen_WIDTH8_0
            AOI22X1                       osu05_stdcells
            DFFPOSX1                      osu05_stdcells
            INVX2                         osu05_stdcells
        dffen_WIDTH8_1
            AOI22X1                       osu05_stdcells
            DFFPOSX1                      osu05_stdcells
            INVX2                         osu05_stdcells
        dffen_WIDTH8_2
            AOI22X1                       osu05_stdcells
            DFFPOSX1                      osu05_stdcells
            INVX2                         osu05_stdcells
        dffen_WIDTH8_3
            AOI22X1                       osu05_stdcells
            DFFPOSX1                      osu05_stdcells
            INVX2                         osu05_stdcells
        dffenr_WIDTH8
            AOI22X1                       osu05_stdcells
            DFFPOSX1                      osu05_stdcells
            INVX2                         osu05_stdcells
            NOR2X1                        osu05_stdcells
        mux2_WIDTH3
            AOI22X1                       osu05_stdcells
            INVX2                         osu05_stdcells
        mux2_WIDTH8_0
            AOI22X1                       osu05_stdcells
            INVX2                         osu05_stdcells
        mux2_WIDTH8_1
            AOI22X1                       osu05_stdcells
            INVX2                         osu05_stdcells
        mux2_WIDTH8_2
            AOI22X1                       osu05_stdcells
            INVX2                         osu05_stdcells
        mux4_WIDTH8_0
            AND2X2                        osu05_stdcells
            AOI22X1                       osu05_stdcells
            INVX2                         osu05_stdcells
            NAND2X1                        osu05_stdcells
            NOR2X1                         osu05_stdcells
        mux4_WIDTH8_1
            AND2X2                         osu05_stdcells
            AOI22X1                        osu05_stdcells
            INVX2                          osu05_stdcells
            NAND2X1                        osu05_stdcells
            NOR2X1                         osu05_stdcells
        regfile_WIDTH8_REGBITS3
            AND2X2                         osu05_stdcells
            AOI21X1                        osu05_stdcells
            AOI22X1                        osu05_stdcells
            BUFX2                          osu05_stdcells
            DFFPOSX1                       osu05_stdcells
            INVX2                          osu05_stdcells
            NAND2X1                        osu05_stdcells
            NAND3X1                        osu05_stdcells
            NOR2X1                         osu05_stdcells
            NOR3X1                         osu05_stdcells
            OAI21X1                        osu05_stdcells
            OR2X1                          osu05_stdcells
        zerodetect_WIDTH8
            NAND2X1                        osu05_stdcells
```

```
            NOR2X1                        osu05_stdcells
1
```

Mips_syn.net

```
****************************************
Report : net
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************


Operating Conditions: typical   Library: osu05_stdcells
Wire Load Model Mode: top


Net
 Fanout     Fanin      Load       LoadUR     LoadUF     LoadLR     LoadLF     Resist     Pins       Attr
-----------------------------------------------------------------------------------------------
adr[0]
      1          1         0.10       0.10       0.10       0.10       0.10       0.00         2
adr[1]
      1          1         0.10       0.10       0.10       0.10       0.10       0.00         2
```

```
adr[2]
      1         1         0.10      0.10      0.10      0.10      0.10      0.00         2
adr[3]
      1         1         0.10      0.10      0.10      0.10      0.10      0.00         2
adr[4]
      1         1         0.10      0.10      0.10      0.10      0.10      0.00         2
adr[5]
      1         1         0.10      0.10      0.10      0.10      0.10      0.00         2
adr[6]
      1         1         0.10      0.10      0.10      0.10      0.10      0.00         2
adr[7]
      1         1         0.10      0.10      0.10      0.10      0.10      0.00         2
alucont[0]
      3         1         0.08      0.08      0.08      0.08      0.08      0.00         4
alucont[1]
      9         1         0.28      0.28      0.28      0.28      0.28      0.00        10
alucont[2]
      5         1         0.29      0.29      0.29      0.29      0.29      0.00         6
aluop[0]
      3         1         0.10      0.10      0.10      0.10      0.10      0.00         4
aluop[1]
      5         1         0.13      0.13      0.13      0.13      0.13      0.00         6
alusrca
```

```
      9         1        0.30      0.30      0.30      0.30      0.30      0.00       10
alusrcb[0]
      3         1        0.08      0.08      0.08      0.08      0.08      0.00        4
alusrcb[1]
      4         1        0.10      0.10      0.10      0.10      0.10      0.00        5
clk
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
const_gnd
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
instr[0]
      5         1        0.16      0.16      0.16      0.16      0.16      0.00        6
instr[1]
      5         1        0.16      0.16      0.16      0.16      0.16      0.00        6
instr[2]
      6         1        0.20      0.20      0.20      0.20      0.20      0.00        7
instr[3]
      6         1        0.19      0.19      0.19      0.19      0.19      0.00        7
instr[4]
      5         1        0.16      0.16      0.16      0.16      0.16      0.00        6
instr[5]
      7         1        0.20      0.20      0.20      0.20      0.20      0.00        8
instr[26]
      2         1        0.06      0.06      0.06      0.06      0.06      0.00        3
```

```
instr[27]
      4         1        0.13      0.13      0.13      0.13      0.13      0.00        5
instr[28]
      2         1        0.07      0.07      0.07      0.07      0.07      0.00        3
instr[29]
      3         1        0.09      0.09      0.09      0.09      0.09      0.00        4
instr[30]
      2         1        0.06      0.06      0.06      0.06      0.06      0.00        3
instr[31]
      4         1        0.11      0.11      0.11      0.11      0.11      0.00        5
iord
      9         1        0.30      0.30      0.30      0.30      0.30      0.00       10
irwrite[0]
     10         1        0.32      0.32      0.32      0.32      0.32      0.00       11
irwrite[1]
     10         1        0.32      0.32      0.32      0.32      0.32      0.00       11
irwrite[2]
      9         1        0.29      0.29      0.29      0.29      0.29      0.00       10
irwrite[3]
      9         1        0.29      0.29      0.29      0.29      0.29      0.00       10
memdata[0]
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
memdata[1]
```

```
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
memdata[2]
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
memdata[3]
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
memdata[4]
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
memdata[5]
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
memdata[6]
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
memdata[7]
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
memread
      1          1        0.10       0.10       0.10       0.10       0.10       0.00        2
memtoreg
     10          1        0.33       0.33       0.33       0.33       0.33       0.00       11
memwrite
      2          1        0.13       0.13       0.13       0.13       0.13       0.00        3
n1
      1          1        0.03       0.03       0.03       0.03       0.03       0.00        2
n2
      4          1        0.12       0.12       0.12       0.12       0.12       0.00        5
```

```
n3
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n4
      5         1        0.15      0.14      0.15      0.14      0.15      0.00        6
n5
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n6
      5         1        0.14      0.14      0.14      0.14      0.14      0.00        6
n7
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n8
      5         1        0.14      0.14      0.14      0.14      0.14      0.00        6
n9
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n10
      5         1        0.14      0.14      0.14      0.14      0.14      0.00        6
n11
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n12
      5         1        0.14      0.14      0.14      0.14      0.14      0.00        6
n13
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n14
```

```
      5         1        0.14      0.14      0.14      0.14      0.14      0.00        6
n15
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n16
      5         1        0.14      0.14      0.14      0.14      0.14      0.00        6
n17
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
n18
      5         1        0.14      0.14      0.14      0.14      0.14      0.00        6
n19
      7         1        0.25      0.25      0.25      0.25      0.25      0.00        8
n20
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
pcen
      1         1        0.03      0.03      0.03      0.03      0.03      0.00        2
pcsource[0]
      3         1        0.08      0.08      0.08      0.08      0.08      0.00        4
pcsource[1]
      5         1        0.13      0.13      0.13      0.13      0.13      0.00        6
regdst
      4         1        0.13      0.13      0.13      0.13      0.13      0.00        5
regwrite
      2         1        0.05      0.05      0.05      0.05      0.05      0.00        3
```

```
reset
      1          1         0.03       0.03       0.03       0.03       0.03       0.00        2
writedata[0]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
writedata[1]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
writedata[2]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
writedata[3]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
writedata[4]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
writedata[5]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
writedata[6]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
writedata[7]
      2          1         0.13       0.13       0.13       0.13       0.13       0.00        3
zero
      1          1         0.03       0.03       0.03       0.03       0.03       0.00        2
-----------------------------------------------------------------------------------------------
Total 81 nets
    264         81         9.58       9.55       9.58       9.55       9.58       0.00      345
```

```
Maximum
     10          1         0.33       0.33       0.33       0.33       0.33       0.00       11
Average
   3.26       1.00         0.12       0.12       0.12       0.12       0.12       0.00     4.26
1
```

Mips_syn.ports

```
****************************************
Report : port
        -verbose
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************



                       Pin      Wire     Max     Max     Connection
Port           Dir      Load     Load     Trans   Cap     Class      Attrs
--------------------------------------------------------------------------------
clk            in       0.0000   0.0000   --      --      --
const_gnd      in       0.0000   0.0000   --       0.41   --
memdata[0]     in       0.0000   0.0000   --       0.41   --
memdata[1]     in       0.0000   0.0000   --       0.41   --
```

```
memdata[2]     in      0.0000   0.0000   --       0.41   --
memdata[3]     in      0.0000   0.0000   --       0.41   --
memdata[4]     in      0.0000   0.0000   --       0.41   --
memdata[5]     in      0.0000   0.0000   --       0.41   --
memdata[6]     in      0.0000   0.0000   --       0.41   --
memdata[7]     in      0.0000   0.0000   --       0.41   --
reset          in      0.0000   0.0000   --       0.41   --
adr[0]         out     0.1000   0.0000   --      --      --
adr[1]         out     0.1000   0.0000   --      --      --
adr[2]         out     0.1000   0.0000   --      --      --
adr[3]         out     0.1000   0.0000   --      --      --
adr[4]         out     0.1000   0.0000   --      --      --
adr[5]         out     0.1000   0.0000   --      --      --
adr[6]         out     0.1000   0.0000   --      --      --
adr[7]         out     0.1000   0.0000   --      --      --
memread        out     0.1000   0.0000   --      --      --
memwrite       out     0.1000   0.0000   --      --      --
writedata[0]   out     0.1000   0.0000   --      --      --
writedata[1]   out     0.1000   0.0000   --      --      --
writedata[2]   out     0.1000   0.0000   --      --      --
writedata[3]   out     0.1000   0.0000   --      --      --
writedata[4]   out     0.1000   0.0000   --      --      --
writedata[5]   out     0.1000   0.0000   --      --      --
```

```
writedata[6]   out     0.1000   0.0000   --      --      --
writedata[7]   out     0.1000   0.0000   --      --      --


              External  Max             Min                Min       Min
              Number    Wireload        Wireload           Pin       Wire
Port          Points    Model           Model              Load      Load
--------------------------------------------------------------------------------
clk                1      --              --               --        --
const_gnd          1      --              --               --        --
memdata[0]         1      --              --               --        --
memdata[1]         1      --              --               --        --
memdata[2]         1      --              --               --        --
memdata[3]         1      --              --               --        --
memdata[4]         1      --              --               --        --
memdata[5]         1      --              --               --        --
memdata[6]         1      --              --               --        --
memdata[7]         1      --              --               --        --
reset              1      --              --               --        --
adr[0]             1      --              --               --        --
adr[1]             1      --              --               --        --
adr[2]             1      --              --               --        --
adr[3]             1      --              --               --        --
```

```
adr[4]            1      --               --               --         --
adr[5]            1      --               --               --         --
adr[6]            1      --               --               --         --
adr[7]            1      --               --               --         --
memread           1      --               --               --         --
memwrite          1      --               --               --         --
writedata[0]      1      --               --               --         --
writedata[1]      1      --               --               --         --
writedata[2]      1      --               --               --         --
writedata[3]      1      --               --               --         --
writedata[4]      1      --               --               --         --
writedata[5]      1      --               --               --         --
writedata[6]      1      --               --               --         --
writedata[7]      1      --               --               --         --


                    Input Delay
                  Min             Max       Related   Max
Input Port    Rise    Fall    Rise    Fall   Clock  Fanout
--------------------------------------------------------------------------------
clk           --      --      --      --      --      1.00
const_gnd     2.00    2.00    2.00    2.00  clk       1.00
memdata[0]    2.00    2.00    2.00    2.00  clk       1.00
memdata[1]    2.00    2.00    2.00    2.00  clk       1.00
```

```
memdata[2]    2.00    2.00    2.00    2.00  clk       1.00
memdata[3]    2.00    2.00    2.00    2.00  clk       1.00
memdata[4]    2.00    2.00    2.00    2.00  clk       1.00
memdata[5]    2.00    2.00    2.00    2.00  clk       1.00
memdata[6]    2.00    2.00    2.00    2.00  clk       1.00
memdata[7]    2.00    2.00    2.00    2.00  clk       1.00
reset         2.00    2.00    2.00    2.00  clk       1.00


                    Driving Cell
Input Port   Rise(min/max)      Fall(min/max)      Mult(min/max)  Attrs(min/max)
--------------------------------------------------------------------------------
const_gnd    osu05_stdcells/INVX1
                                osu05_stdcells/INVX1
                                                     -- /  --
memdata[0]   osu05_stdcells/INVX1
                                osu05_stdcells/INVX1
                                                     -- /  --
memdata[1]   osu05_stdcells/INVX1
                                osu05_stdcells/INVX1
                                                     -- /  --
memdata[2]   osu05_stdcells/INVX1
                                osu05_stdcells/INVX1
```

```
                                                    -- /  --
memdata[3]   osu05_stdcells/INVX1
                                  osu05_stdcells/INVX1
                                                    -- /  --
memdata[4]   osu05_stdcells/INVX1
                                  osu05_stdcells/INVX1
                                                    -- /  --
memdata[5]   osu05_stdcells/INVX1
                                  osu05_stdcells/INVX1
                                                    -- /  --
memdata[6]   osu05_stdcells/INVX1
                                  osu05_stdcells/INVX1
                                                    -- /  --
memdata[7]   osu05_stdcells/INVX1
                                  osu05_stdcells/INVX1
                                                    -- /  --
reset        osu05_stdcells/INVX1
                                  osu05_stdcells/INVX1
                                                    -- /  --


               Max Drive      Min Drive      Resistance    Min     Min       Cell
Input Port    Rise     Fall    Rise     Fall    Max      Min     Cap     Fanout     Deg
```

```
--------------------------------------------------------------------------------
clk           --      --     --      --     --      --     --     --         --
const_gnd     --      --     --      --     --      --     --     --         --
memdata[0]    --      --     --      --     --      --     --     --         --
memdata[1]    --      --     --      --     --      --     --     --         --
memdata[2]    --      --     --      --     --      --     --     --         --
memdata[3]    --      --     --      --     --      --     --     --         --
memdata[4]    --      --     --      --     --      --     --     --         --
memdata[5]    --      --     --      --     --      --     --     --         --
memdata[6]    --      --     --      --     --      --     --     --         --
memdata[7]    --      --     --      --     --      --     --     --         --
reset         --      --     --      --     --      --     --     --         --


               Max Tran        Min Tran
Input Port    Rise    Fall    Rise    Fall
--------------------------------------------------------------------------------
clk           --      --      --      --
const_gnd     --      --      --      --
memdata[0]    --      --      --      --
memdata[1]    --      --      --      --
memdata[2]    --      --      --      --
memdata[3]    --      --      --      --
```

```
memdata[4]    --      --      --      --

memdata[5]    --      --      --      --

memdata[6]    --      --      --      --

memdata[7]    --      --      --      --

reset         --      --      --      --



                    Output Delay

                  Min             Max      Related  Fanout

Output Port   Rise    Fall    Rise    Fall  Clock     Load

--------------------------------------------------------------------------------

adr[0]        1.65    1.65    1.65    1.65  clk       8.00

adr[1]        1.65    1.65    1.65    1.65  clk       8.00

adr[2]        1.65    1.65    1.65    1.65  clk       8.00

adr[3]        1.65    1.65    1.65    1.65  clk       8.00

adr[4]        1.65    1.65    1.65    1.65  clk       8.00

adr[5]        1.65    1.65    1.65    1.65  clk       8.00

adr[6]        1.65    1.65    1.65    1.65  clk       8.00

adr[7]        1.65    1.65    1.65    1.65  clk       8.00

memread       1.65    1.65    1.65    1.65  clk       8.00

memwrite      1.65    1.65    1.65    1.65  clk       8.00

writedata[0]

              1.65    1.65    1.65    1.65  clk       8.00
```

```
writedata[1]
               1.65     1.65     1.65     1.65  clk        8.00
writedata[2]
               1.65     1.65     1.65     1.65  clk        8.00
writedata[3]
               1.65     1.65     1.65     1.65  clk        8.00
writedata[4]
               1.65     1.65     1.65     1.65  clk        8.00
writedata[5]
               1.65     1.65     1.65     1.65  clk        8.00
writedata[6]
               1.65     1.65     1.65     1.65  clk        8.00
writedata[7]
               1.65     1.65     1.65     1.65  clk        8.00

1
```

Mips_syn.pow

```
Loading db file '/apps/design_kits/osu_stdcells_v2p7/synopsys/lib/ami05/osu05_stdcells.db'
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)

****************************************
```

```
Report : power
        -analysis_effort low
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************


Library(s) Used:

    osu05_stdcells (File: /apps/design_kits/osu_stdcells_v2p7/synopsys/lib/ami05/osu05_stdcells.db)


Operating Conditions: typical   Library: osu05_stdcells
Wire Load Model Mode: top


Global Operating Voltage = 5
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =   1.4057 mW   (72%)
  Net Switching Power  = 533.2949 uW   (28%)
                         ---------
Total Dynamic Power    =   1.9390 mW  (100%)

Cell Leakage Power     =  85.2062 nW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.0000            0.0000            0.0000  (   0.00%)
register           0.0000            0.0000            0.0000            0.0000  (   0.00%)
sequential         1.0795        2.0166e-02           34.0571            1.0997  (  56.71%)
combinational      0.3262            0.5131           51.1491            0.8394  (  43.29%)
--------------------------------------------------------------------------------------------------
```

```
Total              1.4057 mW         0.5333 mW        85.2063 nW         1.9391 mW
1
```

Mips_syn.timing

```
****************************************
Report : timing
        -path full
        -delay max
        -nworst 5
        -max_paths 5
        -sort_by group
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************

Operating Conditions: typical   Library: osu05_stdcells
Wire Load Model Mode: top

  Startpoint: cont/state_reg_1_
              (rising edge-triggered flip-flop)
  Endpoint: adr[7] (output port clocked by clk)
  Path Group: (none)
  Path Type: max

  Point                                                        Incr       Path
  ---------------------------------------------------------------------------
  cont/state_reg_1_/CLK (DFFPOSX1)                             0.00       0.00 r
  cont/state_reg_1_/Q (DFFPOSX1)                               0.54       0.54 f
  cont/U33/Y (INVX2)                                           0.16       0.71 r
  cont/U53/Y (NAND2X1)                                         0.22       0.93 f
  cont/U46/Y (NOR2X1)                                          0.28       1.21 r
  cont/U45/Y (NOR2X1)                                          0.22       1.43 f
  cont/U44/Y (OAI21X1)                                         0.63       2.06 r
  cont/iord (controller)                                       0.00       2.06 r
  dp/iord (datapath_WIDTH8_REGBITS3)                           0.00       2.06 r
  dp/adrmux/s (mux2_WIDTH8_2)                                  0.00       2.06 r
  dp/adrmux/U9/Y (INVX2)                                       0.43       2.49 f
  dp/adrmux/U10/Y (AOI22X1)                                    0.18       2.67 r
  dp/adrmux/U1/Y (INVX2)                                       0.17       2.85 f
  dp/adrmux/y[7] (mux2_WIDTH8_2)                               0.00       2.85 f
  dp/adr[7] (datapath_WIDTH8_REGBITS3)                         0.00       2.85 f
  adr[7] (out)                                                 0.00       2.85 f
  data arrival time                                                       2.85
  ---------------------------------------------------------------------------
  (Path is unconstrained)
```

```
Startpoint: cont/state_reg_1_
            (rising edge-triggered flip-flop)
Endpoint: adr[6] (output port clocked by clk)
Path Group: (none)
Path Type: max

Point                                                    Incr       Path
---------------------------------------------------------------------------
cont/state_reg_1_/CLK (DFFPOSX1)                         0.00       0.00 r
cont/state_reg_1_/Q (DFFPOSX1)                           0.54       0.54 f
cont/U33/Y (INVX2)                                       0.16       0.71 r
cont/U53/Y (NAND2X1)                                     0.22       0.93 f
cont/U46/Y (NOR2X1)                                      0.28       1.21 r
cont/U45/Y (NOR2X1)                                      0.22       1.43 f
cont/U44/Y (OAI21X1)                                     0.63       2.06 r
cont/iord (controller)                                   0.00       2.06 r
dp/iord (datapath_WIDTH8_REGBITS3)                       0.00       2.06 r
dp/adrmux/s (mux2_WIDTH8_2)                              0.00       2.06 r
dp/adrmux/U9/Y (INVX2)                                   0.43       2.49 f
dp/adrmux/U11/Y (AOI22X1)                                0.18       2.67 r
dp/adrmux/U2/Y (INVX2)                                   0.17       2.84 f
```

```
dp/adrmux/y[6] (mux2_WIDTH8_2)                         0.00       2.84 f
dp/adr[6] (datapath_WIDTH8_REGBITS3)                   0.00       2.84 f
adr[6] (out)                                           0.00       2.84 f
data arrival time                                                 2.84
--------------------------------------------------------------------------
(Path is unconstrained)



Startpoint: cont/state_reg_1_
            (rising edge-triggered flip-flop)
Endpoint: adr[5] (output port clocked by clk)
Path Group: (none)
Path Type: max

Point                                                  Incr       Path
--------------------------------------------------------------------------
cont/state_reg_1_/CLK (DFFPOSX1)                       0.00       0.00 r
cont/state_reg_1_/Q (DFFPOSX1)                         0.54       0.54 f
cont/U33/Y (INVX2)                                     0.16       0.71 r
cont/U53/Y (NAND2X1)                                   0.22       0.93 f
cont/U46/Y (NOR2X1)                                    0.28       1.21 r
cont/U45/Y (NOR2X1)                                    0.22       1.43 f
cont/U44/Y (OAI21X1)                                   0.63       2.06 r
```

```
  cont/iord (controller)                                  0.00       2.06 r
  dp/iord (datapath_WIDTH8_REGBITS3)                      0.00       2.06 r
  dp/adrmux/s (mux2_WIDTH8_2)                             0.00       2.06 r
  dp/adrmux/U9/Y (INVX2)                                  0.43       2.49 f
  dp/adrmux/U12/Y (AOI22X1)                               0.18       2.67 r
  dp/adrmux/U3/Y (INVX2)                                  0.17       2.84 f
  dp/adrmux/y[5] (mux2_WIDTH8_2)                          0.00       2.84 f
  dp/adr[5] (datapath_WIDTH8_REGBITS3)                    0.00       2.84 f
  adr[5] (out)                                            0.00       2.84 f
  data arrival time                                                  2.84
  ---------------------------------------------------------------------------
  (Path is unconstrained)


  Startpoint: cont/state_reg_1_
              (rising edge-triggered flip-flop)
  Endpoint: adr[4] (output port clocked by clk)
  Path Group: (none)
  Path Type: max

  Point                                                   Incr       Path
  ---------------------------------------------------------------------------
  cont/state_reg_1_/CLK (DFFPOSX1)                        0.00       0.00 r
  cont/state_reg_1_/Q (DFFPOSX1)                          0.54       0.54 f
  cont/U33/Y (INVX2)                                      0.16       0.71 r
  cont/U53/Y (NAND2X1)                                    0.22       0.93 f
  cont/U46/Y (NOR2X1)                                     0.28       1.21 r
  cont/U45/Y (NOR2X1)                                     0.22       1.43 f
  cont/U44/Y (OAI21X1)                                    0.63       2.06 r
  cont/iord (controller)                                  0.00       2.06 r
  dp/iord (datapath_WIDTH8_REGBITS3)                      0.00       2.06 r
  dp/adrmux/s (mux2_WIDTH8_2)                             0.00       2.06 r
  dp/adrmux/U9/Y (INVX2)                                  0.43       2.49 f
  dp/adrmux/U13/Y (AOI22X1)                               0.18       2.67 r
  dp/adrmux/U4/Y (INVX2)                                  0.17       2.84 f
  dp/adrmux/y[4] (mux2_WIDTH8_2)                          0.00       2.84 f
  dp/adr[4] (datapath_WIDTH8_REGBITS3)                    0.00       2.84 f
  adr[4] (out)                                            0.00       2.84 f
  data arrival time                                                  2.84
  --------------------------------------------------------------------------
  (Path is unconstrained)



  Startpoint: cont/state_reg_1_
              (rising edge-triggered flip-flop)
  Endpoint: adr[3] (output port clocked by clk)
```

```
Path Group: (none)
Path Type: max

Point                                                    Incr       Path
---------------------------------------------------------------------------
cont/state_reg_1_/CLK (DFFPOSX1)                         0.00       0.00 r
cont/state_reg_1_/Q (DFFPOSX1)                           0.54       0.54 f
cont/U33/Y (INVX2)                                       0.16       0.71 r
cont/U53/Y (NAND2X1)                                     0.22       0.93 f
cont/U46/Y (NOR2X1)                                      0.28       1.21 r
cont/U45/Y (NOR2X1)                                      0.22       1.43 f
cont/U44/Y (OAI21X1)                                     0.63       2.06 r
cont/iord (controller)                                   0.00       2.06 r
dp/iord (datapath_WIDTH8_REGBITS3)                       0.00       2.06 r
dp/adrmux/s (mux2_WIDTH8_2)                              0.00       2.06 r
dp/adrmux/U9/Y (INVX2)                                   0.43       2.49 f
dp/adrmux/U14/Y (AOI22X1)                                0.18       2.67 r
dp/adrmux/U5/Y (INVX2)                                   0.17       2.84 f
dp/adrmux/y[3] (mux2_WIDTH8_2)                           0.00       2.84 f
dp/adr[3] (datapath_WIDTH8_REGBITS3)                     0.00       2.84 f
adr[3] (out)                                             0.00       2.84 f
data arrival time                                                   2.84
---------------------------------------------------------------------------
```

```
  (Path is unconstrained)


1

****************************************
Report : timing
        -path full
        -delay min
        -nworst 5
        -max_paths 5
        -sort_by group
Design : mips
Version: O-2018.06-SP1
Date   : Sat Apr 30 12:55:30 2022
****************************************

Operating Conditions: typical   Library: osu05_stdcells
Wire Load Model Mode: top

  Startpoint: dp/wrd/q_reg_7_
              (rising edge-triggered flip-flop)
  Endpoint: writedata[7]
```

```
          (output port clocked by clk)
  Path Group: (none)
  Path Type: min

  Point                                                  Incr       Path
  --------------------------------------------------------------------------
  dp/wrd/q_reg_7_/CLK (DFFPOSX1)                         0.00       0.00 r
  dp/wrd/q_reg_7_/Q (DFFPOSX1)                           0.37       0.37 r
  dp/wrd/q[7] (dff_WIDTH8_1)                             0.00       0.37 r
  dp/writedata[7] (datapath_WIDTH8_REGBITS3)             0.00       0.37 r
  writedata[7] (out)                                     0.00       0.37 r
  data arrival time                                                 0.37
  --------------------------------------------------------------------------
  (Path is unconstrained)



  Startpoint: dp/wrd/q_reg_6_
              (rising edge-triggered flip-flop)
  Endpoint: writedata[6]
            (output port clocked by clk)
  Path Group: (none)
  Path Type: min
  Point                                                    Incr       Path
  ---------------------------------------------------------------------------
  dp/wrd/q_reg_6_/CLK (DFFPOSX1)                           0.00       0.00 r
  dp/wrd/q_reg_6_/Q (DFFPOSX1)                             0.37       0.37 r
  dp/wrd/q[6] (dff_WIDTH8_1)                               0.00       0.37 r
  dp/writedata[6] (datapath_WIDTH8_REGBITS3)               0.00       0.37 r
  writedata[6] (out)                                       0.00       0.37 r
  data arrival time                                                   0.37
  ---------------------------------------------------------------------------
  (Path is unconstrained)



  Startpoint: dp/wrd/q_reg_5_
              (rising edge-triggered flip-flop)
  Endpoint: writedata[5]
            (output port clocked by clk)
  Path Group: (none)
  Path Type: min

  Point                                                    Incr       Path
  ---------------------------------------------------------------------------
  dp/wrd/q_reg_5_/CLK (DFFPOSX1)                           0.00       0.00 r
  dp/wrd/q_reg_5_/Q (DFFPOSX1)                             0.37       0.37 r
```

```
dp/wrd/q[5] (dff_WIDTH8_1)                              0.00       0.37 r
dp/writedata[5] (datapath_WIDTH8_REGBITS3)               0.00       0.37 r
writedata[5] (out)                                       0.00       0.37 r
data arrival time                                                   0.37
--------------------------------------------------------------------------
(Path is unconstrained)



Startpoint: dp/wrd/q_reg_4_
            (rising edge-triggered flip-flop)
Endpoint: writedata[4]
          (output port clocked by clk)
Path Group: (none)
Path Type: min

Point                                                    Incr       Path
--------------------------------------------------------------------------
dp/wrd/q_reg_4_/CLK (DFFPOSX1)                           0.00       0.00 r
dp/wrd/q_reg_4_/Q (DFFPOSX1)                             0.37       0.37 r
dp/wrd/q[4] (dff_WIDTH8_1)                               0.00       0.37 r
dp/writedata[4] (datapath_WIDTH8_REGBITS3)               0.00       0.37 r
writedata[4] (out)                                       0.00       0.37 r
data arrival time                                                   0.37
```

```
  ---------------------------------------------------------------------------
  (Path is unconstrained)


  Startpoint: dp/wrd/q_reg_3_
              (rising edge-triggered flip-flop)
  Endpoint: writedata[3]
            (output port clocked by clk)
  Path Group: (none)
  Path Type: min

  Point                                                    Incr       Path
  ---------------------------------------------------------------------------
  dp/wrd/q_reg_3_/CLK (DFFPOSX1)                           0.00       0.00 r
  dp/wrd/q_reg_3_/Q (DFFPOSX1)                             0.37       0.37 r
  dp/wrd/q[3] (dff_WIDTH8_1)                               0.00       0.37 r
  dp/writedata[3] (datapath_WIDTH8_REGBITS3)               0.00       0.37 r
  writedata[3] (out)                                       0.00       0.37 r
  data arrival time                                                   0.37
  ---------------------------------------------------------------------------
  (Path is unconstrained)
```

Appendix C

Mips_syn.v

```
/////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : O-2018.06-SP1
// Date      : Sat Apr 30 16:39:35 2022
/////////////////////////////////////////////////////////////



module controller ( alusrca, alusrcb, aluop, pcen, iord, irwrite, memread,
        memwrite, memtoreg, pcsource, regwrite, regdst, op, clk, reset, zero
 );
  output [1:0] alusrcb;
  output [1:0] aluop;
  output [3:0] irwrite;
  output [1:0] pcsource;
  input [5:0] op;
  input clk, reset, zero;
  output alusrca, pcen, iord, memread, memwrite, memtoreg, regwrite, regdst;
  wire   N45, n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39, n40,
         n41, n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53, n54,
         n55, n56, n57, n58, n59, n60, n61, n62, n63, n64, n65, n66, n67, n68,
         n69, n70, n71, n72, n73, n74, n75, n1, n3, n4, n5, n6, n7, n8, n9,
         n10, n11, n12, n13, n14, n16, n18, n19, n20, n21, n23, n24, n25, n26,
         n27, n76;
  wire   [3:0] state;

  DFFPOSX1 state_reg_0_ ( .D(N45), .CLK(clk), .Q(state[0]) );
  DFFPOSX1 state_reg_3_ ( .D(n6), .CLK(clk), .Q(state[3]) );
  DFFPOSX1 state_reg_2_ ( .D(n4), .CLK(clk), .Q(state[2]) );
  DFFPOSX1 state_reg_1_ ( .D(n3), .CLK(clk), .Q(state[1]) );
  INVX2 U4 ( .A(n34), .Y(irwrite[3]) );
  INVX2 U5 ( .A(n35), .Y(irwrite[2]) );
  AND2X2 U8 ( .A(state[0]), .B(state[2]), .Y(n51) );
  AND2X2 U9 ( .A(n11), .B(op[5]), .Y(n66) );
  OAI21X1 U37 ( .A(n28), .B(n29), .C(n20), .Y(regwrite) );
  AOI21X1 U38 ( .A(n18), .B(n28), .C(n29), .Y(regdst) );
  NAND2X1 U39 ( .A(n30), .B(n16), .Y(pcen) );
  AOI21X1 U40 ( .A(zero), .B(aluop[0]), .C(pcsource[1]), .Y(n30) );
  NAND3X1 U41 ( .A(n25), .B(state[0]), .C(state[2]), .Y(n31) );
  NOR2X1 U42 ( .A(n32), .B(n21), .Y(memtoreg) );
  NAND2X1 U43 ( .A(n16), .B(n33), .Y(memread) );
  OAI21X1 U44 ( .A(n29), .B(n18), .C(n36), .Y(iord) );
  NOR2X1 U45 ( .A(memwrite), .B(n14), .Y(n36) );
  NOR2X1 U46 ( .A(n32), .B(n28), .Y(memwrite) );
  NAND2X1 U47 ( .A(n37), .B(n38), .Y(alusrcb[1]) );
  NAND2X1 U48 ( .A(n16), .B(n37), .Y(alusrcb[0]) );
  NAND3X1 U49 ( .A(n34), .B(n35), .C(n40), .Y(n39) );
  NOR2X1 U50 ( .A(irwrite[0]), .B(irwrite[1]), .Y(n40) );
  NAND3X1 U51 ( .A(n1), .B(n19), .C(n38), .Y(alusrca) );
  NOR2X1 U52 ( .A(n18), .B(n32), .Y(aluop[0]) );
  NAND2X1 U53 ( .A(state[3]), .B(n27), .Y(n32) );
  OAI21X1 U54 ( .A(n45), .B(n46), .C(n76), .Y(n44) );
  OAI21X1 U55 ( .A(n10), .B(n11), .C(n47), .Y(n46) );
  NAND3X1 U56 ( .A(n23), .B(n48), .C(n49), .Y(n47) );
  OAI21X1 U57 ( .A(op[1]), .B(n7), .C(n50), .Y(n48) );
  NAND2X1 U58 ( .A(n33), .B(n19), .Y(n45) );
  NAND3X1 U59 ( .A(state[1]), .B(n24), .C(n51), .Y(n33) );
  OAI21X1 U60 ( .A(n53), .B(n54), .C(n76), .Y(n52) );
  OAI21X1 U61 ( .A(n55), .B(n37), .C(n34), .Y(n54) );
  NAND3X1 U62 ( .A(n27), .B(n24), .C(n43), .Y(n34) );
  AOI22X1 U63 ( .A(n49), .B(n56), .C(n9), .D(n7), .Y(n55) );
  OAI21X1 U64 ( .A(n12), .B(n58), .C(n50), .Y(n56) );
  NAND3X1 U65 ( .A(n57), .B(n12), .C(op[1]), .Y(n50) );
  NAND2X1 U66 ( .A(n57), .B(n13), .Y(n58) );
  NAND2X1 U67 ( .A(n35), .B(n59), .Y(n53) );
  NAND2X1 U68 ( .A(n60), .B(state[1]), .Y(n35) );
  OAI21X1 U69 ( .A(n62), .B(n63), .C(n76), .Y(n61) );
  OAI21X1 U70 ( .A(n37), .B(n64), .C(n41), .Y(n63) );
  NAND3X1 U71 ( .A(state[1]), .B(n24), .C(n65), .Y(n41) );
  NAND3X1 U72 ( .A(n59), .B(n19), .C(n42), .Y(n62) );
  NAND2X1 U73 ( .A(n60), .B(n27), .Y(n42) );
  NOR2X1 U74 ( .A(n28), .B(state[3]), .Y(n60) );
  NAND2X1 U75 ( .A(state[0]), .B(n26), .Y(n28) );
  NOR2X1 U76 ( .A(n21), .B(n29), .Y(aluop[1]) );
  NAND2X1 U77 ( .A(state[3]), .B(state[1]), .Y(n29) );
  OAI21X1 U78 ( .A(n66), .B(n67), .C(n68), .Y(n59) );
  OR2X1 U79 ( .A(n69), .B(n70), .Y(N45) );
  OAI21X1 U80 ( .A(state[1]), .B(state[0]), .C(n5), .Y(n70) );
  OAI22X1 U81 ( .A(n10), .B(n67), .C(n38), .D(n9), .Y(n71) );
  NOR2X1 U82 ( .A(n11), .B(op[5]), .Y(n67) );
  NOR2X1 U83 ( .A(n64), .B(n38), .Y(n68) );
  NAND3X1 U84 ( .A(state[1]), .B(n24), .C(n43), .Y(n38) );
  NOR2X1 U85 ( .A(n26), .B(state[0]), .Y(n43) );
  NAND3X1 U86 ( .A(n13), .B(n12), .C(n49), .Y(n64) );
  NAND3X1 U87 ( .A(n72), .B(n21), .C(n73), .Y(n69) );
  NOR2X1 U88 ( .A(state[3]), .B(reset), .Y(n73) );
  NOR2X1 U89 ( .A(state[2]), .B(state[0]), .Y(n65) );
  OAI21X1 U90 ( .A(n8), .B(n74), .C(n23), .Y(n72) );
  NAND3X1 U91 ( .A(state[2]), .B(state[0]), .C(n75), .Y(n37) );
  NOR2X1 U92 ( .A(state[3]), .B(state[1]), .Y(n75) );
  OAI21X1 U93 ( .A(n57), .B(n12), .C(n13), .Y(n74) );
  NOR2X1 U94 ( .A(op[5]), .B(op[3]), .Y(n57) );
  NOR2X1 U95 ( .A(op[4]), .B(op[0]), .Y(n49) );
  INVX2 U3 ( .A(n1), .Y(pcsource[0]) );
  INVX2 U6 ( .A(aluop[0]), .Y(n1) );
  INVX2 U7 ( .A(n61), .Y(n3) );
  INVX2 U10 ( .A(n52), .Y(n4) );
  INVX2 U11 ( .A(n71), .Y(n5) );
  INVX2 U12 ( .A(n44), .Y(n6) );
  INVX2 U13 ( .A(n57), .Y(n7) );
  INVX2 U14 ( .A(n49), .Y(n8) );
  INVX2 U15 ( .A(n64), .Y(n9) );
  INVX2 U16 ( .A(n68), .Y(n10) );
  INVX2 U17 ( .A(op[3]), .Y(n11) );
  INVX2 U18 ( .A(op[2]), .Y(n12) );
  INVX2 U19 ( .A(op[1]), .Y(n13) );
  INVX2 U20 ( .A(n33), .Y(n14) );
  INVX2 U21 ( .A(n31), .Y(pcsource[1]) );
  INVX2 U22 ( .A(n39), .Y(n16) );
  INVX2 U23 ( .A(n42), .Y(irwrite[0]) );
  INVX2 U24 ( .A(n43), .Y(n18) );
  INVX2 U25 ( .A(aluop[1]), .Y(n19) );
  INVX2 U26 ( .A(memtoreg), .Y(n20) );
  INVX2 U27 ( .A(n65), .Y(n21) );
  INVX2 U28 ( .A(n41), .Y(irwrite[1]) );
  INVX2 U29 ( .A(n37), .Y(n23) );
  INVX2 U30 ( .A(state[3]), .Y(n24) );
  INVX2 U31 ( .A(n32), .Y(n25) );
  INVX2 U32 ( .A(state[2]), .Y(n26) );
  INVX2 U33 ( .A(state[1]), .Y(n27) );
  INVX2 U34 ( .A(reset), .Y(n76) );
endmodule


module alucontrol ( alucont, aluop, funct );
  output [2:0] alucont;
  input [1:0] aluop;
  input [5:0] funct;
  wire   n8, n9, n10, n11, n12, n13, n14, n15, n16, n1, n2, n3, n4, n5, n6;

  INVX2 U3 ( .A(n14), .Y(alucont[0]) );
  OAI21X1 U10 ( .A(aluop[1]), .B(n6), .C(n8), .Y(alucont[2]) );
  OAI21X1 U11 ( .A(n9), .B(n10), .C(aluop[1]), .Y(n8) );
  OAI21X1 U12 ( .A(funct[2]), .B(n5), .C(funct[5]), .Y(n10) );
  NAND3X1 U13 ( .A(n4), .B(n1), .C(n11), .Y(n9) );
  OAI21X1 U14 ( .A(n12), .B(n13), .C(aluop[1]), .Y(alucont[1]) );
  NAND2X1 U15 ( .A(funct[5]), .B(n11), .Y(n13) );
  NAND2X1 U16 ( .A(funct[3]), .B(n4), .Y(n11) );
  NAND3X1 U17 ( .A(n3), .B(n1), .C(n5), .Y(n12) );
  OAI21X1 U18 ( .A(n15), .B(n16), .C(aluop[1]), .Y(n14) );
  OAI21X1 U19 ( .A(n4), .B(n3), .C(funct[5]), .Y(n16) );
  NAND3X1 U20 ( .A(n2), .B(n1), .C(n5), .Y(n15) );
  INVX2 U4 ( .A(funct[4]), .Y(n1) );
  INVX2 U5 ( .A(funct[3]), .Y(n2) );
  INVX2 U6 ( .A(funct[2]), .Y(n3) );
  INVX2 U7 ( .A(funct[1]), .Y(n4) );
  INVX2 U8 ( .A(funct[0]), .Y(n5) );
  INVX2 U9 ( .A(aluop[0]), .Y(n6) );
endmodule



module mux2_WIDTH3 ( d0, d1, s, y );
  input [2:0] d0;
  input [2:0] d1;
  output [2:0] y;
  input s;
  wire   n5, n6, n7, n1;
  INVX2 U1 ( .A(n5), .Y(y[2]) );
  INVX2 U2 ( .A(n6), .Y(y[1]) );
  INVX2 U3 ( .A(n7), .Y(y[0]) );
  AOI22X1 U5 ( .A(d0[2]), .B(n1), .C(s), .D(d1[2]), .Y(n5) );
  AOI22X1 U6 ( .A(d0[1]), .B(n1), .C(d1[1]), .D(s), .Y(n6) );
  AOI22X1 U7 ( .A(d0[0]), .B(n1), .C(d1[0]), .D(s), .Y(n7) );
  INVX2 U4 ( .A(s), .Y(n1) );
endmodule


module dffen_WIDTH8_3 ( clk, en, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk, en;
  wire   n1, n3, n4, n5, n6, n7, n8, n9, n2, n10, n11, n12, n13, n14, n15, n16,
         n17;

  DFFPOSX1 q_reg_7_ ( .D(n2), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(n10), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(n11), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(n12), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(n13), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(n14), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(n15), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(n16), .CLK(clk), .Q(q[0]) );
  AOI22X1 U3 ( .A(en), .B(d[0]), .C(q[0]), .D(n17), .Y(n1) );
  AOI22X1 U5 ( .A(d[1]), .B(en), .C(q[1]), .D(n17), .Y(n3) );
  AOI22X1 U7 ( .A(d[2]), .B(en), .C(q[2]), .D(n17), .Y(n4) );
  AOI22X1 U9 ( .A(d[3]), .B(en), .C(q[3]), .D(n17), .Y(n5) );
  AOI22X1 U11 ( .A(d[4]), .B(en), .C(q[4]), .D(n17), .Y(n6) );
  AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n17), .Y(n7) );
  AOI22X1 U15 ( .A(d[6]), .B(en), .C(q[6]), .D(n17), .Y(n8) );
  AOI22X1 U17 ( .A(d[7]), .B(en), .C(q[7]), .D(n17), .Y(n9) );
  INVX2 U2 ( .A(n9), .Y(n2) );
  INVX2 U4 ( .A(n8), .Y(n10) );
  INVX2 U6 ( .A(n7), .Y(n11) );
  INVX2 U8 ( .A(n6), .Y(n12) );
  INVX2 U10 ( .A(n5), .Y(n13) );
  INVX2 U12 ( .A(n4), .Y(n14) );
  INVX2 U14 ( .A(n3), .Y(n15) );
  INVX2 U16 ( .A(n1), .Y(n16) );
  INVX2 U18 ( .A(en), .Y(n17) );
endmodule


module dffen_WIDTH8_2 ( clk, en, d, q );
```

```
input [7:0] d;
output [7:0] q;
input clk, en;
wire   n2, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20, n21, n22,
       n23, n24, n25;

DFFPOSX1 q_reg_7_ ( .D(n2), .CLK(clk), .Q(q[7]) );
DFFPOSX1 q_reg_6_ ( .D(n10), .CLK(clk), .Q(q[6]) );
DFFPOSX1 q_reg_5_ ( .D(n11), .CLK(clk), .Q(q[5]) );
DFFPOSX1 q_reg_4_ ( .D(n12), .CLK(clk), .Q(q[4]) );
DFFPOSX1 q_reg_3_ ( .D(n13), .CLK(clk), .Q(q[3]) );
DFFPOSX1 q_reg_2_ ( .D(n14), .CLK(clk), .Q(q[2]) );
DFFPOSX1 q_reg_1_ ( .D(n15), .CLK(clk), .Q(q[1]) );
DFFPOSX1 q_reg_0_ ( .D(n16), .CLK(clk), .Q(q[0]) );
AOI22X1 U3 ( .A(en), .B(d[0]), .C(q[0]), .D(n17), .Y(n25) );
AOI22X1 U5 ( .A(d[1]), .B(en), .C(q[1]), .D(n17), .Y(n24) );
AOI22X1 U7 ( .A(d[2]), .B(en), .C(q[2]), .D(n17), .Y(n23) );
AOI22X1 U9 ( .A(d[3]), .B(en), .C(q[3]), .D(n17), .Y(n22) );
AOI22X1 U11 ( .A(d[4]), .B(en), .C(q[4]), .D(n17), .Y(n21) );
AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n17), .Y(n20) );
AOI22X1 U15 ( .A(d[6]), .B(en), .C(q[6]), .D(n17), .Y(n19) );
AOI22X1 U17 ( .A(d[7]), .B(en), .C(q[7]), .D(n17), .Y(n18) );
INVX2 U2 ( .A(n18), .Y(n2) );
```

```
  INVX2 U4 ( .A(n19), .Y(n10) );
  INVX2 U6 ( .A(n20), .Y(n11) );
  INVX2 U8 ( .A(n21), .Y(n12) );
  INVX2 U10 ( .A(n22), .Y(n13) );
  INVX2 U12 ( .A(n23), .Y(n14) );
  INVX2 U14 ( .A(n24), .Y(n15) );
  INVX2 U16 ( .A(n25), .Y(n16) );
  INVX2 U18 ( .A(en), .Y(n17) );
endmodule


module dffen_WIDTH8_1 ( clk, en, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk, en;
  wire   n2, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20, n21, n22,
         n23, n24, n25;

  DFFPOSX1 q_reg_7_ ( .D(n2), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(n10), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(n11), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(n12), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(n13), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(n14), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(n15), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(n16), .CLK(clk), .Q(q[0]) );
  AOI22X1 U3 ( .A(en), .B(d[0]), .C(q[0]), .D(n17), .Y(n25) );
  AOI22X1 U5 ( .A(d[1]), .B(en), .C(q[1]), .D(n17), .Y(n24) );
  AOI22X1 U7 ( .A(d[2]), .B(en), .C(q[2]), .D(n17), .Y(n23) );
  AOI22X1 U9 ( .A(d[3]), .B(en), .C(q[3]), .D(n17), .Y(n22) );
  AOI22X1 U11 ( .A(d[4]), .B(en), .C(q[4]), .D(n17), .Y(n21) );
  AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n17), .Y(n20) );
  AOI22X1 U15 ( .A(d[6]), .B(en), .C(q[6]), .D(n17), .Y(n19) );
  AOI22X1 U17 ( .A(d[7]), .B(en), .C(q[7]), .D(n17), .Y(n18) );
  INVX2 U2 ( .A(n18), .Y(n2) );
  INVX2 U4 ( .A(n19), .Y(n10) );
  INVX2 U6 ( .A(n20), .Y(n11) );
  INVX2 U8 ( .A(n21), .Y(n12) );
  INVX2 U10 ( .A(n22), .Y(n13) );
  INVX2 U12 ( .A(n23), .Y(n14) );
  INVX2 U14 ( .A(n24), .Y(n15) );
  INVX2 U16 ( .A(n25), .Y(n16) );
  INVX2 U18 ( .A(en), .Y(n17) );
endmodule
```

```
module dffen_WIDTH8_0 ( clk, en, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk, en;
  wire   n2, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20, n21, n22,
         n23, n24, n25;

  DFFPOSX1 q_reg_7_ ( .D(n2), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(n10), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(n11), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(n12), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(n13), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(n14), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(n15), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(n16), .CLK(clk), .Q(q[0]) );
  AOI22X1 U3 ( .A(en), .B(d[0]), .C(q[0]), .D(n17), .Y(n25) );
  AOI22X1 U5 ( .A(d[1]), .B(en), .C(q[1]), .D(n17), .Y(n24) );
  AOI22X1 U7 ( .A(d[2]), .B(en), .C(q[2]), .D(n17), .Y(n23) );
  AOI22X1 U9 ( .A(d[3]), .B(en), .C(q[3]), .D(n17), .Y(n22) );
  AOI22X1 U11 ( .A(d[4]), .B(en), .C(q[4]), .D(n17), .Y(n21) );
  AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n17), .Y(n20) );
  AOI22X1 U15 ( .A(d[6]), .B(en), .C(q[6]), .D(n17), .Y(n19) );
  AOI22X1 U17 ( .A(d[7]), .B(en), .C(q[7]), .D(n17), .Y(n18) );
  INVX2 U2 ( .A(n18), .Y(n2) );
  INVX2 U4 ( .A(n19), .Y(n10) );
  INVX2 U6 ( .A(n20), .Y(n11) );
  INVX2 U8 ( .A(n21), .Y(n12) );
  INVX2 U10 ( .A(n22), .Y(n13) );
  INVX2 U12 ( .A(n23), .Y(n14) );
  INVX2 U14 ( .A(n24), .Y(n15) );
  INVX2 U16 ( .A(n25), .Y(n16) );
  INVX2 U18 ( .A(en), .Y(n17) );
endmodule


module dffenr_WIDTH8 ( clk, reset, en, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk, reset, en;
  wire   n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n1, n2, n3, n4, n5,
         n6, n7, n8, n9;

  DFFPOSX1 q_reg_7_ ( .D(n2), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(n3), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(n4), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(n5), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(n6), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(n7), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(n8), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(n1), .CLK(clk), .Q(q[0]) );
  AOI22X1 U12 ( .A(d[0]), .B(n11), .C(q[0]), .D(n12), .Y(n10) );
  AOI22X1 U13 ( .A(d[1]), .B(n11), .C(q[1]), .D(n12), .Y(n13) );
  AOI22X1 U14 ( .A(d[2]), .B(n11), .C(q[2]), .D(n12), .Y(n14) );
  AOI22X1 U15 ( .A(d[3]), .B(n11), .C(q[3]), .D(n12), .Y(n15) );
  AOI22X1 U16 ( .A(d[4]), .B(n11), .C(q[4]), .D(n12), .Y(n16) );
  AOI22X1 U17 ( .A(d[5]), .B(n11), .C(q[5]), .D(n12), .Y(n17) );
  AOI22X1 U18 ( .A(d[6]), .B(n11), .C(q[6]), .D(n12), .Y(n18) );
  AOI22X1 U19 ( .A(d[7]), .B(n11), .C(q[7]), .D(n12), .Y(n19) );
  NOR2X1 U20 ( .A(reset), .B(n11), .Y(n12) );
  NOR2X1 U21 ( .A(n9), .B(reset), .Y(n11) );
  INVX2 U3 ( .A(n10), .Y(n1) );
  INVX2 U4 ( .A(n19), .Y(n2) );
  INVX2 U5 ( .A(n18), .Y(n3) );
  INVX2 U6 ( .A(n17), .Y(n4) );
  INVX2 U7 ( .A(n16), .Y(n5) );
  INVX2 U8 ( .A(n15), .Y(n6) );
  INVX2 U9 ( .A(n14), .Y(n7) );
  INVX2 U10 ( .A(n13), .Y(n8) );
  INVX2 U11 ( .A(en), .Y(n9) );
```

```
endmodule


module dff_WIDTH8_3 ( clk, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk;


  DFFPOSX1 q_reg_7_ ( .D(d[7]), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(d[6]), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(d[5]), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(d[4]), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(d[3]), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(d[2]), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(d[1]), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(d[0]), .CLK(clk), .Q(q[0]) );
endmodule


module dff_WIDTH8_2 ( clk, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk;


  DFFPOSX1 q_reg_7_ ( .D(d[7]), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(d[6]), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(d[5]), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(d[4]), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(d[3]), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(d[2]), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(d[1]), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(d[0]), .CLK(clk), .Q(q[0]) );
endmodule


module dff_WIDTH8_1 ( clk, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk;


  DFFPOSX1 q_reg_7_ ( .D(d[7]), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(d[6]), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(d[5]), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(d[4]), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(d[3]), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(d[2]), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(d[1]), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(d[0]), .CLK(clk), .Q(q[0]) );
endmodule


module dff_WIDTH8_0 ( clk, d, q );
  input [7:0] d;
  output [7:0] q;
  input clk;


  DFFPOSX1 q_reg_7_ ( .D(d[7]), .CLK(clk), .Q(q[7]) );
  DFFPOSX1 q_reg_6_ ( .D(d[6]), .CLK(clk), .Q(q[6]) );
  DFFPOSX1 q_reg_5_ ( .D(d[5]), .CLK(clk), .Q(q[5]) );
  DFFPOSX1 q_reg_4_ ( .D(d[4]), .CLK(clk), .Q(q[4]) );
  DFFPOSX1 q_reg_3_ ( .D(d[3]), .CLK(clk), .Q(q[3]) );
  DFFPOSX1 q_reg_2_ ( .D(d[2]), .CLK(clk), .Q(q[2]) );
  DFFPOSX1 q_reg_1_ ( .D(d[1]), .CLK(clk), .Q(q[1]) );
  DFFPOSX1 q_reg_0_ ( .D(d[0]), .CLK(clk), .Q(q[0]) );
endmodule
```

```
module mux2_WIDTH8_2 ( d0, d1, s, y );
  input [7:0] d0;
  input [7:0] d1;
  output [7:0] y;
  input s;
  wire   n10, n11, n12, n13, n14, n15, n16, n17, n1;

  INVX2 U1 ( .A(n10), .Y(y[7]) );
  INVX2 U2 ( .A(n11), .Y(y[6]) );
  INVX2 U3 ( .A(n12), .Y(y[5]) );
  INVX2 U4 ( .A(n13), .Y(y[4]) );
  INVX2 U5 ( .A(n14), .Y(y[3]) );
  INVX2 U6 ( .A(n15), .Y(y[2]) );
  INVX2 U7 ( .A(n16), .Y(y[1]) );
  INVX2 U8 ( .A(n17), .Y(y[0]) );
  AOI22X1 U10 ( .A(d0[7]), .B(n1), .C(s), .D(d1[7]), .Y(n10) );
  AOI22X1 U11 ( .A(d0[6]), .B(n1), .C(d1[6]), .D(s), .Y(n11) );
  AOI22X1 U12 ( .A(d0[5]), .B(n1), .C(d1[5]), .D(s), .Y(n12) );
  AOI22X1 U13 ( .A(d0[4]), .B(n1), .C(d1[4]), .D(s), .Y(n13) );
  AOI22X1 U14 ( .A(d0[3]), .B(n1), .C(d1[3]), .D(s), .Y(n14) );
  AOI22X1 U15 ( .A(d0[2]), .B(n1), .C(d1[2]), .D(s), .Y(n15) );
  AOI22X1 U16 ( .A(d0[1]), .B(n1), .C(d1[1]), .D(s), .Y(n16) );
  AOI22X1 U17 ( .A(d0[0]), .B(n1), .C(d1[0]), .D(s), .Y(n17) );
  INVX2 U9 ( .A(s), .Y(n1) );
endmodule


module mux2_WIDTH8_1 ( d0, d1, s, y );
  input [7:0] d0;
  input [7:0] d1;
  output [7:0] y;
  input s;
  wire   n1, n2, n3, n4, n5, n6, n7, n8, n9;

  INVX2 U1 ( .A(n9), .Y(y[7]) );
  INVX2 U2 ( .A(n8), .Y(y[6]) );
  INVX2 U3 ( .A(n7), .Y(y[5]) );
  INVX2 U4 ( .A(n6), .Y(y[4]) );
  INVX2 U5 ( .A(n5), .Y(y[3]) );
  INVX2 U6 ( .A(n4), .Y(y[2]) );
  INVX2 U7 ( .A(n3), .Y(y[1]) );
  INVX2 U8 ( .A(n2), .Y(y[0]) );
  AOI22X1 U10 ( .A(d0[7]), .B(n1), .C(s), .D(d1[7]), .Y(n9) );
  AOI22X1 U11 ( .A(d0[6]), .B(n1), .C(d1[6]), .D(s), .Y(n8) );
  AOI22X1 U12 ( .A(d0[5]), .B(n1), .C(d1[5]), .D(s), .Y(n7) );
  AOI22X1 U13 ( .A(d0[4]), .B(n1), .C(d1[4]), .D(s), .Y(n6) );
  AOI22X1 U14 ( .A(d0[3]), .B(n1), .C(d1[3]), .D(s), .Y(n5) );
  AOI22X1 U15 ( .A(d0[2]), .B(n1), .C(d1[2]), .D(s), .Y(n4) );
  AOI22X1 U16 ( .A(d0[1]), .B(n1), .C(d1[1]), .D(s), .Y(n3) );
  AOI22X1 U17 ( .A(d0[0]), .B(n1), .C(d1[0]), .D(s), .Y(n2) );
  INVX2 U9 ( .A(s), .Y(n1) );
endmodule


module mux4_WIDTH8_1 ( d0, d1, d2, d3, s, y );
  input [7:0] d0;
  input [7:0] d1;
  input [7:0] d2;
  input [7:0] d3;
  input [1:0] s;
  output [7:0] y;
  wire   n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16,
         n17, n18, n19, n20, n21, n1;

  AND2X2 U1 ( .A(s[1]), .B(n1), .Y(n5) );
  AND2X2 U2 ( .A(s[1]), .B(s[0]), .Y(n4) );
  NAND2X1 U4 ( .A(n2), .B(n3), .Y(y[7]) );
  AOI22X1 U5 ( .A(d3[7]), .B(n4), .C(d2[7]), .D(n5), .Y(n3) );
  AOI22X1 U6 ( .A(d1[7]), .B(n6), .C(d0[7]), .D(n7), .Y(n2) );
  NAND2X1 U7 ( .A(n8), .B(n9), .Y(y[6]) );
  AOI22X1 U8 ( .A(d3[6]), .B(n4), .C(d2[6]), .D(n5), .Y(n9) );
  AOI22X1 U9 ( .A(d1[6]), .B(n6), .C(d0[6]), .D(n7), .Y(n8) );
  NAND2X1 U10 ( .A(n10), .B(n11), .Y(y[5]) );
  AOI22X1 U11 ( .A(d3[5]), .B(n4), .C(d2[5]), .D(n5), .Y(n11) );
  AOI22X1 U12 ( .A(d1[5]), .B(n6), .C(d0[5]), .D(n7), .Y(n10) );
  NAND2X1 U13 ( .A(n12), .B(n13), .Y(y[4]) );
  AOI22X1 U14 ( .A(d3[4]), .B(n4), .C(d2[4]), .D(n5), .Y(n13) );
  AOI22X1 U15 ( .A(d1[4]), .B(n6), .C(d0[4]), .D(n7), .Y(n12) );
  NAND2X1 U16 ( .A(n14), .B(n15), .Y(y[3]) );
  AOI22X1 U17 ( .A(d3[3]), .B(n4), .C(d2[3]), .D(n5), .Y(n15) );
  AOI22X1 U18 ( .A(d1[3]), .B(n6), .C(d0[3]), .D(n7), .Y(n14) );
  NAND2X1 U19 ( .A(n16), .B(n17), .Y(y[2]) );
  AOI22X1 U20 ( .A(d3[2]), .B(n4), .C(d2[2]), .D(n5), .Y(n17) );
  AOI22X1 U21 ( .A(d1[2]), .B(n6), .C(d0[2]), .D(n7), .Y(n16) );
  NAND2X1 U22 ( .A(n18), .B(n19), .Y(y[1]) );
  AOI22X1 U23 ( .A(d3[1]), .B(n4), .C(d2[1]), .D(n5), .Y(n19) );
  AOI22X1 U24 ( .A(d1[1]), .B(n6), .C(d0[1]), .D(n7), .Y(n18) );
  NAND2X1 U25 ( .A(n20), .B(n21), .Y(y[0]) );
  AOI22X1 U26 ( .A(d3[0]), .B(n4), .C(d2[0]), .D(n5), .Y(n21) );
  AOI22X1 U27 ( .A(d1[0]), .B(n6), .C(d0[0]), .D(n7), .Y(n20) );
  NOR2X1 U28 ( .A(s[0]), .B(s[1]), .Y(n7) );
  NOR2X1 U29 ( .A(n1), .B(s[1]), .Y(n6) );
  INVX2 U3 ( .A(s[0]), .Y(n1) );
endmodule


module mux4_WIDTH8_0 ( d0, d1, d2, d3, s, y );
  input [7:0] d0;
  input [7:0] d1;
  input [7:0] d2;
  input [7:0] d3;
  input [1:0] s;
  output [7:0] y;
  wire   n1, n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32, n33, n34,
         n35, n36, n37, n38, n39, n40, n41;

  AND2X2 U1 ( .A(s[1]), .B(n1), .Y(n38) );
  AND2X2 U2 ( .A(s[1]), .B(s[0]), .Y(n39) );
  NAND2X1 U4 ( .A(n41), .B(n40), .Y(y[7]) );
  AOI22X1 U5 ( .A(d3[7]), .B(n39), .C(d2[7]), .D(n38), .Y(n40) );
  AOI22X1 U6 ( .A(d1[7]), .B(n37), .C(d0[7]), .D(n36), .Y(n41) );
  NAND2X1 U7 ( .A(n35), .B(n34), .Y(y[6]) );
  AOI22X1 U8 ( .A(d3[6]), .B(n39), .C(d2[6]), .D(n38), .Y(n34) );
  AOI22X1 U9 ( .A(d1[6]), .B(n37), .C(d0[6]), .D(n36), .Y(n35) );
  NAND2X1 U10 ( .A(n33), .B(n32), .Y(y[5]) );
  AOI22X1 U11 ( .A(d3[5]), .B(n39), .C(d2[5]), .D(n38), .Y(n32) );
  AOI22X1 U12 ( .A(d1[5]), .B(n37), .C(d0[5]), .D(n36), .Y(n33) );
  NAND2X1 U13 ( .A(n31), .B(n30), .Y(y[4]) );
  AOI22X1 U14 ( .A(d3[4]), .B(n39), .C(d2[4]), .D(n38), .Y(n30) );
  AOI22X1 U15 ( .A(d1[4]), .B(n37), .C(d0[4]), .D(n36), .Y(n31) );
  NAND2X1 U16 ( .A(n29), .B(n28), .Y(y[3]) );
  AOI22X1 U17 ( .A(d3[3]), .B(n39), .C(d2[3]), .D(n38), .Y(n28) );
  AOI22X1 U18 ( .A(d1[3]), .B(n37), .C(d0[3]), .D(n36), .Y(n29) );
  NAND2X1 U19 ( .A(n27), .B(n26), .Y(y[2]) );
  AOI22X1 U20 ( .A(d3[2]), .B(n39), .C(d2[2]), .D(n38), .Y(n26) );
  AOI22X1 U21 ( .A(d1[2]), .B(n37), .C(d0[2]), .D(n36), .Y(n27) );
  NAND2X1 U22 ( .A(n25), .B(n24), .Y(y[1]) );
  AOI22X1 U23 ( .A(d3[1]), .B(n39), .C(d2[1]), .D(n38), .Y(n24) );
  AOI22X1 U24 ( .A(d1[1]), .B(n37), .C(d0[1]), .D(n36), .Y(n25) );
  NAND2X1 U25 ( .A(n23), .B(n22), .Y(y[0]) );
  AOI22X1 U26 ( .A(d3[0]), .B(n39), .C(d2[0]), .D(n38), .Y(n22) );
  AOI22X1 U27 ( .A(d1[0]), .B(n37), .C(d0[0]), .D(n36), .Y(n23) );
  NOR2X1 U28 ( .A(s[0]), .B(s[1]), .Y(n36) );
  NOR2X1 U29 ( .A(n1), .B(s[1]), .Y(n37) );
  INVX2 U3 ( .A(s[0]), .Y(n1) );
endmodule
```

```
module mux2_WIDTH8_0 ( d0, d1, s, y );
  input [7:0] d0;
  input [7:0] d1;
  output [7:0] y;
  input s;
  wire   n1, n2, n3, n4, n5, n6, n7, n8, n9;

  INVX2 U1 ( .A(n9), .Y(y[7]) );
  INVX2 U2 ( .A(n8), .Y(y[6]) );
  INVX2 U3 ( .A(n7), .Y(y[5]) );
  INVX2 U4 ( .A(n6), .Y(y[4]) );
  INVX2 U5 ( .A(n5), .Y(y[3]) );
  INVX2 U6 ( .A(n4), .Y(y[2]) );
  INVX2 U7 ( .A(n3), .Y(y[1]) );
  INVX2 U8 ( .A(n2), .Y(y[0]) );
  AOI22X1 U10 ( .A(d0[7]), .B(n1), .C(s), .D(d1[7]), .Y(n9) );
  AOI22X1 U11 ( .A(d0[6]), .B(n1), .C(d1[6]), .D(s), .Y(n8) );
  AOI22X1 U12 ( .A(d0[5]), .B(n1), .C(d1[5]), .D(s), .Y(n7) );
  AOI22X1 U13 ( .A(d0[4]), .B(n1), .C(d1[4]), .D(s), .Y(n6) );
  AOI22X1 U14 ( .A(d0[3]), .B(n1), .C(d1[3]), .D(s), .Y(n5) );
  AOI22X1 U15 ( .A(d0[2]), .B(n1), .C(d1[2]), .D(s), .Y(n4) );
  AOI22X1 U16 ( .A(d0[1]), .B(n1), .C(d1[1]), .D(s), .Y(n3) );
  AOI22X1 U17 ( .A(d0[0]), .B(n1), .C(d1[0]), .D(s), .Y(n2) );
  INVX2 U9 ( .A(s), .Y(n1) );
endmodule


module regfile_WIDTH8_REGBITS3 ( clk, regwrite, ra1, ra2, wa, wd, rd1, rd2 );
  input [2:0] ra1;
  input [2:0] ra2;
  input [2:0] wa;
  input [7:0] wd;
  output [7:0] rd1;
  output [7:0] rd2;
  input clk, regwrite;
  wire   N37, N38, N39, N40, N41, N42, N43, N44, N47, N48, N49, N50, N51, N52,
         N53, N54, n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25,
         n26, n27, n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39,
         n40, n41, n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53,
         n54, n55, n56, n57, n58, n59, n60, n61, n62, n63, n64, n65, n66, n67,
         n68, n69, n70, n71, n72, n73, n74, n75, n76, n77, n78, n79, n80, n81,
         n82, n83, n84, n85, n86, n87, n88, n89, n90, n91, n92, n93, n94, n95,
         n96, n97, n98, n99, n100, n101, n102, n103, n104, n105, n106, n107,
         n108, n109, n110, n111, n112, n113, n114, n115, n116, n117, n118,
         n119, n120, n121, n122, n123, n124, n125, n126, n127, n128, n129,
         n130, n131, n132, n133, n134, n135, n136, n137, n138, n139, n140,
         n141, n142, n143, n144, n145, n146, n147, n148, n149, n150, n151,
         n152, n153, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13,
         n154, n155, n156, n157, n158, n159, n160, n161, n162, n163, n164,
         n165, n166, n167, n168, n169, n170, n171, n172, n173, n174, n175,
         n176, n177, n178, n179, n180, n181, n182, n183, n184, n185, n186,
         n187, n188, n189, n190, n191, n192, n193, n194, n195, n196, n197,
         n198, n199, n200, n201, n202, n203, n204, n205, n206, n207, n208,
         n209, n210, n211, n212, n213, n214, n215, n216, n217, n218, n219,
         n220, n221, n222, n223, n224, n225, n226, n227, n228, n229, n230,
         n231, n232, n233, n234, n235, n236, n237, n238, n239, n240, n241,
         n242, n243, n244, n245, n246, n247, n248, n249, n250, n251, n252,
         n253, n254, n255, n256, n257, n258, n259, n260, n261, n262, n263,
         n264, n265, n266, n267, n268, n269, n270, n271, n272, n273, n274,
         n275, n276, n277, n278, n279, n280, n281, n282, n283, n284, n285,
         n286, n287;
  wire   [63:0] RAM;

  DFFPOSX1 RAM_reg_7__7_ ( .D(n153), .CLK(clk), .Q(RAM[63]) );
  DFFPOSX1 RAM_reg_7__6_ ( .D(n152), .CLK(clk), .Q(RAM[62]) );
  DFFPOSX1 RAM_reg_7__5_ ( .D(n151), .CLK(clk), .Q(RAM[61]) );
  DFFPOSX1 RAM_reg_7__4_ ( .D(n150), .CLK(clk), .Q(RAM[60]) );
  DFFPOSX1 RAM_reg_7__3_ ( .D(n149), .CLK(clk), .Q(RAM[59]) );
  DFFPOSX1 RAM_reg_7__2_ ( .D(n148), .CLK(clk), .Q(RAM[58]) );
  DFFPOSX1 RAM_reg_7__1_ ( .D(n147), .CLK(clk), .Q(RAM[57]) );
  DFFPOSX1 RAM_reg_7__0_ ( .D(n146), .CLK(clk), .Q(RAM[56]) );
  DFFPOSX1 RAM_reg_6__7_ ( .D(n145), .CLK(clk), .Q(RAM[55]) );
  DFFPOSX1 RAM_reg_6__6_ ( .D(n144), .CLK(clk), .Q(RAM[54]) );
  DFFPOSX1 RAM_reg_6__5_ ( .D(n143), .CLK(clk), .Q(RAM[53]) );
  DFFPOSX1 RAM_reg_6__4_ ( .D(n142), .CLK(clk), .Q(RAM[52]) );
  DFFPOSX1 RAM_reg_6__3_ ( .D(n141), .CLK(clk), .Q(RAM[51]) );
  DFFPOSX1 RAM_reg_6__2_ ( .D(n140), .CLK(clk), .Q(RAM[50]) );
  DFFPOSX1 RAM_reg_6__1_ ( .D(n139), .CLK(clk), .Q(RAM[49]) );
  DFFPOSX1 RAM_reg_6__0_ ( .D(n138), .CLK(clk), .Q(RAM[48]) );
  DFFPOSX1 RAM_reg_5__7_ ( .D(n137), .CLK(clk), .Q(RAM[47]) );
  DFFPOSX1 RAM_reg_5__6_ ( .D(n136), .CLK(clk), .Q(RAM[46]) );
  DFFPOSX1 RAM_reg_5__5_ ( .D(n135), .CLK(clk), .Q(RAM[45]) );
  DFFPOSX1 RAM_reg_5__4_ ( .D(n134), .CLK(clk), .Q(RAM[44]) );
  DFFPOSX1 RAM_reg_5__3_ ( .D(n133), .CLK(clk), .Q(RAM[43]) );
  DFFPOSX1 RAM_reg_5__2_ ( .D(n132), .CLK(clk), .Q(RAM[42]) );
  DFFPOSX1 RAM_reg_5__1_ ( .D(n131), .CLK(clk), .Q(RAM[41]) );
  DFFPOSX1 RAM_reg_5__0_ ( .D(n130), .CLK(clk), .Q(RAM[40]) );
  DFFPOSX1 RAM_reg_4__7_ ( .D(n129), .CLK(clk), .Q(RAM[39]) );
  DFFPOSX1 RAM_reg_4__6_ ( .D(n128), .CLK(clk), .Q(RAM[38]) );
  DFFPOSX1 RAM_reg_4__5_ ( .D(n127), .CLK(clk), .Q(RAM[37]) );
  DFFPOSX1 RAM_reg_4__4_ ( .D(n126), .CLK(clk), .Q(RAM[36]) );
  DFFPOSX1 RAM_reg_4__3_ ( .D(n125), .CLK(clk), .Q(RAM[35]) );
  DFFPOSX1 RAM_reg_4__2_ ( .D(n124), .CLK(clk), .Q(RAM[34]) );
  DFFPOSX1 RAM_reg_4__1_ ( .D(n123), .CLK(clk), .Q(RAM[33]) );
  DFFPOSX1 RAM_reg_4__0_ ( .D(n122), .CLK(clk), .Q(RAM[32]) );
  DFFPOSX1 RAM_reg_3__7_ ( .D(n121), .CLK(clk), .Q(RAM[31]) );
  DFFPOSX1 RAM_reg_3__6_ ( .D(n120), .CLK(clk), .Q(RAM[30]) );
  DFFPOSX1 RAM_reg_3__5_ ( .D(n119), .CLK(clk), .Q(RAM[29]) );
  DFFPOSX1 RAM_reg_3__4_ ( .D(n118), .CLK(clk), .Q(RAM[28]) );
  DFFPOSX1 RAM_reg_3__3_ ( .D(n117), .CLK(clk), .Q(RAM[27]) );
  DFFPOSX1 RAM_reg_3__2_ ( .D(n116), .CLK(clk), .Q(RAM[26]) );
  DFFPOSX1 RAM_reg_3__1_ ( .D(n115), .CLK(clk), .Q(RAM[25]) );
  DFFPOSX1 RAM_reg_3__0_ ( .D(n114), .CLK(clk), .Q(RAM[24]) );
  DFFPOSX1 RAM_reg_2__7_ ( .D(n113), .CLK(clk), .Q(RAM[23]) );
  DFFPOSX1 RAM_reg_2__6_ ( .D(n112), .CLK(clk), .Q(RAM[22]) );
  DFFPOSX1 RAM_reg_2__5_ ( .D(n111), .CLK(clk), .Q(RAM[21]) );
  DFFPOSX1 RAM_reg_2__4_ ( .D(n110), .CLK(clk), .Q(RAM[20]) );
  DFFPOSX1 RAM_reg_2__3_ ( .D(n109), .CLK(clk), .Q(RAM[19]) );
  DFFPOSX1 RAM_reg_2__2_ ( .D(n108), .CLK(clk), .Q(RAM[18]) );
  DFFPOSX1 RAM_reg_2__1_ ( .D(n107), .CLK(clk), .Q(RAM[17]) );
  DFFPOSX1 RAM_reg_2__0_ ( .D(n106), .CLK(clk), .Q(RAM[16]) );
  DFFPOSX1 RAM_reg_1__7_ ( .D(n105), .CLK(clk), .Q(RAM[15]) );
  DFFPOSX1 RAM_reg_1__6_ ( .D(n104), .CLK(clk), .Q(RAM[14]) );
  DFFPOSX1 RAM_reg_1__5_ ( .D(n103), .CLK(clk), .Q(RAM[13]) );
  DFFPOSX1 RAM_reg_1__4_ ( .D(n102), .CLK(clk), .Q(RAM[12]) );
  DFFPOSX1 RAM_reg_1__3_ ( .D(n101), .CLK(clk), .Q(RAM[11]) );
  DFFPOSX1 RAM_reg_1__2_ ( .D(n100), .CLK(clk), .Q(RAM[10]) );
  DFFPOSX1 RAM_reg_1__1_ ( .D(n99), .CLK(clk), .Q(RAM[9]) );
  DFFPOSX1 RAM_reg_1__0_ ( .D(n98), .CLK(clk), .Q(RAM[8]) );
  DFFPOSX1 RAM_reg_0__7_ ( .D(n97), .CLK(clk), .Q(RAM[7]) );
  DFFPOSX1 RAM_reg_0__6_ ( .D(n96), .CLK(clk), .Q(RAM[6]) );
  DFFPOSX1 RAM_reg_0__5_ ( .D(n95), .CLK(clk), .Q(RAM[5]) );
  DFFPOSX1 RAM_reg_0__4_ ( .D(n94), .CLK(clk), .Q(RAM[4]) );
  DFFPOSX1 RAM_reg_0__3_ ( .D(n93), .CLK(clk), .Q(RAM[3]) );
  DFFPOSX1 RAM_reg_0__2_ ( .D(n92), .CLK(clk), .Q(RAM[2]) );
  DFFPOSX1 RAM_reg_0__1_ ( .D(n91), .CLK(clk), .Q(RAM[1]) );
  DFFPOSX1 RAM_reg_0__0_ ( .D(n90), .CLK(clk), .Q(RAM[0]) );
  AND2X2 U2 ( .A(N47), .B(n286), .Y(rd2[7]) );
  AND2X2 U3 ( .A(N48), .B(n286), .Y(rd2[6]) );
  AND2X2 U4 ( .A(N49), .B(n286), .Y(rd2[5]) );
  AND2X2 U5 ( .A(N50), .B(n286), .Y(rd2[4]) );
  AND2X2 U6 ( .A(N51), .B(n286), .Y(rd2[3]) );
  AND2X2 U7 ( .A(N52), .B(n286), .Y(rd2[2]) );
  AND2X2 U8 ( .A(N53), .B(n286), .Y(rd2[1]) );
  AND2X2 U9 ( .A(N54), .B(n286), .Y(rd2[0]) );
  AND2X2 U10 ( .A(N37), .B(n285), .Y(rd1[7]) );
  AND2X2 U11 ( .A(N38), .B(n285), .Y(rd1[6]) );
  AND2X2 U12 ( .A(N39), .B(n285), .Y(rd1[5]) );
  AND2X2 U13 ( .A(N40), .B(n285), .Y(rd1[4]) );
  AND2X2 U14 ( .A(N41), .B(n285), .Y(rd1[3]) );
  AND2X2 U15 ( .A(N42), .B(n285), .Y(rd1[2]) );
  AND2X2 U16 ( .A(N43), .B(n285), .Y(rd1[1]) );
  AND2X2 U17 ( .A(N44), .B(n285), .Y(rd1[0]) );
  AND2X2 U18 ( .A(wa[2]), .B(regwrite), .Y(n31) );
  NOR3X1 U32 ( .A(ra2[1]), .B(ra2[2]), .C(ra2[0]), .Y(n14) );
  NOR3X1 U33 ( .A(ra1[1]), .B(ra1[2]), .C(ra1[0]), .Y(n15) );
  OAI21X1 U34 ( .A(n260), .B(n279), .C(n17), .Y(n141) );
  NAND2X1 U35 ( .A(RAM[51]), .B(n260), .Y(n17) );
  OAI21X1 U36 ( .A(n260), .B(n278), .C(n18), .Y(n142) );
  NAND2X1 U37 ( .A(RAM[52]), .B(n260), .Y(n18) );
  OAI21X1 U38 ( .A(n260), .B(n277), .C(n19), .Y(n143) );
  NAND2X1 U39 ( .A(RAM[53]), .B(n260), .Y(n19) );
  OAI21X1 U40 ( .A(n260), .B(n276), .C(n20), .Y(n144) );
  NAND2X1 U41 ( .A(RAM[54]), .B(n260), .Y(n20) );
  OAI21X1 U42 ( .A(n260), .B(n275), .C(n21), .Y(n145) );
  NAND2X1 U43 ( .A(RAM[55]), .B(n260), .Y(n21) );
  OAI21X1 U44 ( .A(n274), .B(n282), .C(n23), .Y(n146) );
  NAND2X1 U45 ( .A(RAM[56]), .B(n274), .Y(n23) );
  OAI21X1 U46 ( .A(n274), .B(n281), .C(n24), .Y(n147) );
  NAND2X1 U47 ( .A(RAM[57]), .B(n274), .Y(n24) );
  OAI21X1 U48 ( .A(n274), .B(n280), .C(n25), .Y(n148) );
  NAND2X1 U49 ( .A(RAM[58]), .B(n274), .Y(n25) );
  OAI21X1 U50 ( .A(n279), .B(n274), .C(n26), .Y(n149) );
  NAND2X1 U51 ( .A(RAM[59]), .B(n274), .Y(n26) );
  OAI21X1 U52 ( .A(n278), .B(n274), .C(n27), .Y(n150) );
  NAND2X1 U53 ( .A(RAM[60]), .B(n274), .Y(n27) );
  OAI21X1 U54 ( .A(n277), .B(n274), .C(n28), .Y(n151) );
  NAND2X1 U55 ( .A(RAM[61]), .B(n274), .Y(n28) );
  OAI21X1 U56 ( .A(n276), .B(n274), .C(n29), .Y(n152) );
  NAND2X1 U57 ( .A(RAM[62]), .B(n274), .Y(n29) );
  OAI21X1 U58 ( .A(n275), .B(n274), .C(n30), .Y(n153) );
  NAND2X1 U59 ( .A(RAM[63]), .B(n274), .Y(n30) );
  NAND3X1 U60 ( .A(wa[1]), .B(n31), .C(wa[0]), .Y(n22) );
  OAI21X1 U61 ( .A(n282), .B(n272), .C(n33), .Y(n90) );
  NAND2X1 U62 ( .A(RAM[0]), .B(n272), .Y(n33) );
  OAI21X1 U63 ( .A(n281), .B(n272), .C(n34), .Y(n91) );
  NAND2X1 U64 ( .A(RAM[1]), .B(n272), .Y(n34) );
  OAI21X1 U65 ( .A(n280), .B(n272), .C(n35), .Y(n92) );
  NAND2X1 U66 ( .A(RAM[2]), .B(n272), .Y(n35) );
  OAI21X1 U67 ( .A(n279), .B(n272), .C(n36), .Y(n93) );
  NAND2X1 U68 ( .A(RAM[3]), .B(n272), .Y(n36) );
  OAI21X1 U69 ( .A(n278), .B(n272), .C(n37), .Y(n94) );
  NAND2X1 U70 ( .A(RAM[4]), .B(n272), .Y(n37) );
  OAI21X1 U71 ( .A(n277), .B(n272), .C(n38), .Y(n95) );
  NAND2X1 U72 ( .A(RAM[5]), .B(n272), .Y(n38) );
  OAI21X1 U73 ( .A(n276), .B(n272), .C(n39), .Y(n96) );
  NAND2X1 U74 ( .A(RAM[6]), .B(n272), .Y(n39) );
  OAI21X1 U75 ( .A(n275), .B(n272), .C(n40), .Y(n97) );
  NAND2X1 U76 ( .A(RAM[7]), .B(n272), .Y(n40) );
  NAND3X1 U77 ( .A(n284), .B(n283), .C(n41), .Y(n32) );
  OAI21X1 U78 ( .A(n282), .B(n270), .C(n43), .Y(n98) );
  NAND2X1 U79 ( .A(RAM[8]), .B(n270), .Y(n43) );
  OAI21X1 U80 ( .A(n281), .B(n270), .C(n44), .Y(n99) );
  NAND2X1 U81 ( .A(RAM[9]), .B(n270), .Y(n44) );
  OAI21X1 U82 ( .A(n280), .B(n270), .C(n45), .Y(n100) );
  NAND2X1 U83 ( .A(RAM[10]), .B(n270), .Y(n45) );
  OAI21X1 U84 ( .A(n279), .B(n270), .C(n46), .Y(n101) );
  NAND2X1 U85 ( .A(RAM[11]), .B(n270), .Y(n46) );
  OAI21X1 U86 ( .A(n278), .B(n270), .C(n47), .Y(n102) );
  NAND2X1 U87 ( .A(RAM[12]), .B(n270), .Y(n47) );
  OAI21X1 U88 ( .A(n277), .B(n270), .C(n48), .Y(n103) );
  NAND2X1 U89 ( .A(RAM[13]), .B(n270), .Y(n48) );
  OAI21X1 U90 ( .A(n276), .B(n270), .C(n49), .Y(n104) );
  NAND2X1 U91 ( .A(RAM[14]), .B(n270), .Y(n49) );
  OAI21X1 U92 ( .A(n275), .B(n270), .C(n50), .Y(n105) );
```

```
NAND2X1 U93 ( .A(RAM[15]), .B(n270), .Y(n50) );
NAND3X1 U94 ( .A(wa[0]), .B(n283), .C(n41), .Y(n42) );
OAI21X1 U95 ( .A(n282), .B(n268), .C(n52), .Y(n106) );
NAND2X1 U96 ( .A(RAM[16]), .B(n268), .Y(n52) );
OAI21X1 U97 ( .A(n281), .B(n268), .C(n53), .Y(n107) );
NAND2X1 U98 ( .A(RAM[17]), .B(n268), .Y(n53) );
OAI21X1 U99 ( .A(n280), .B(n268), .C(n54), .Y(n108) );
NAND2X1 U100 ( .A(RAM[18]), .B(n268), .Y(n54) );
OAI21X1 U101 ( .A(n279), .B(n268), .C(n55), .Y(n109) );
NAND2X1 U102 ( .A(RAM[19]), .B(n268), .Y(n55) );
OAI21X1 U103 ( .A(n278), .B(n268), .C(n56), .Y(n110) );
NAND2X1 U104 ( .A(RAM[20]), .B(n268), .Y(n56) );
OAI21X1 U105 ( .A(n277), .B(n268), .C(n57), .Y(n111) );
NAND2X1 U106 ( .A(RAM[21]), .B(n268), .Y(n57) );
OAI21X1 U107 ( .A(n276), .B(n268), .C(n58), .Y(n112) );
NAND2X1 U108 ( .A(RAM[22]), .B(n268), .Y(n58) );
OAI21X1 U109 ( .A(n275), .B(n268), .C(n59), .Y(n113) );
NAND2X1 U110 ( .A(RAM[23]), .B(n268), .Y(n59) );
NAND3X1 U111 ( .A(wa[1]), .B(n284), .C(n41), .Y(n51) );
OAI21X1 U112 ( .A(n282), .B(n266), .C(n61), .Y(n114) );
NAND2X1 U113 ( .A(RAM[24]), .B(n266), .Y(n61) );
OAI21X1 U114 ( .A(n281), .B(n266), .C(n62), .Y(n115) );
NAND2X1 U115 ( .A(RAM[25]), .B(n266), .Y(n62) );
```

```
  OAI21X1 U116 ( .A(n280), .B(n266), .C(n63), .Y(n116) );
  NAND2X1 U117 ( .A(RAM[26]), .B(n266), .Y(n63) );
  OAI21X1 U118 ( .A(n279), .B(n266), .C(n64), .Y(n117) );
  NAND2X1 U119 ( .A(RAM[27]), .B(n266), .Y(n64) );
  OAI21X1 U120 ( .A(n278), .B(n266), .C(n65), .Y(n118) );
  NAND2X1 U121 ( .A(RAM[28]), .B(n266), .Y(n65) );
  OAI21X1 U122 ( .A(n277), .B(n266), .C(n66), .Y(n119) );
  NAND2X1 U123 ( .A(RAM[29]), .B(n266), .Y(n66) );
  OAI21X1 U124 ( .A(n276), .B(n266), .C(n67), .Y(n120) );
  NAND2X1 U125 ( .A(RAM[30]), .B(n266), .Y(n67) );
  OAI21X1 U126 ( .A(n275), .B(n266), .C(n68), .Y(n121) );
  NAND2X1 U127 ( .A(RAM[31]), .B(n266), .Y(n68) );
  NAND3X1 U128 ( .A(wa[0]), .B(wa[1]), .C(n41), .Y(n60) );
  NOR2X1 U129 ( .A(n287), .B(wa[2]), .Y(n41) );
  OAI21X1 U130 ( .A(n282), .B(n264), .C(n70), .Y(n122) );
  NAND2X1 U131 ( .A(RAM[32]), .B(n264), .Y(n70) );
  OAI21X1 U132 ( .A(n281), .B(n264), .C(n71), .Y(n123) );
  NAND2X1 U133 ( .A(RAM[33]), .B(n264), .Y(n71) );
  OAI21X1 U134 ( .A(n280), .B(n264), .C(n72), .Y(n124) );
  NAND2X1 U135 ( .A(RAM[34]), .B(n264), .Y(n72) );
  OAI21X1 U136 ( .A(n279), .B(n264), .C(n73), .Y(n125) );
  NAND2X1 U137 ( .A(RAM[35]), .B(n264), .Y(n73) );
  OAI21X1 U138 ( .A(n278), .B(n264), .C(n74), .Y(n126) );
  NAND2X1 U139 ( .A(RAM[36]), .B(n264), .Y(n74) );
  OAI21X1 U140 ( .A(n277), .B(n264), .C(n75), .Y(n127) );
  NAND2X1 U141 ( .A(RAM[37]), .B(n264), .Y(n75) );
  OAI21X1 U142 ( .A(n276), .B(n264), .C(n76), .Y(n128) );
  NAND2X1 U143 ( .A(RAM[38]), .B(n264), .Y(n76) );
  OAI21X1 U144 ( .A(n275), .B(n264), .C(n77), .Y(n129) );
  NAND2X1 U145 ( .A(RAM[39]), .B(n264), .Y(n77) );
  NAND3X1 U146 ( .A(n284), .B(n283), .C(n31), .Y(n69) );
  OAI21X1 U147 ( .A(n282), .B(n262), .C(n79), .Y(n130) );
  NAND2X1 U148 ( .A(RAM[40]), .B(n262), .Y(n79) );
  OAI21X1 U149 ( .A(n281), .B(n262), .C(n80), .Y(n131) );
  NAND2X1 U150 ( .A(RAM[41]), .B(n262), .Y(n80) );
  OAI21X1 U151 ( .A(n280), .B(n262), .C(n81), .Y(n132) );
  NAND2X1 U152 ( .A(RAM[42]), .B(n262), .Y(n81) );
  OAI21X1 U153 ( .A(n279), .B(n262), .C(n82), .Y(n133) );
  NAND2X1 U154 ( .A(RAM[43]), .B(n262), .Y(n82) );
  OAI21X1 U155 ( .A(n278), .B(n262), .C(n83), .Y(n134) );
  NAND2X1 U156 ( .A(RAM[44]), .B(n262), .Y(n83) );
  OAI21X1 U157 ( .A(n277), .B(n262), .C(n84), .Y(n135) );
  NAND2X1 U158 ( .A(RAM[45]), .B(n262), .Y(n84) );
  OAI21X1 U159 ( .A(n276), .B(n262), .C(n85), .Y(n136) );
  NAND2X1 U160 ( .A(RAM[46]), .B(n262), .Y(n85) );
  OAI21X1 U161 ( .A(n275), .B(n262), .C(n86), .Y(n137) );
  NAND2X1 U162 ( .A(RAM[47]), .B(n262), .Y(n86) );
  NAND3X1 U163 ( .A(n31), .B(n283), .C(wa[0]), .Y(n78) );
  OAI21X1 U164 ( .A(n260), .B(n282), .C(n87), .Y(n138) );
  NAND2X1 U165 ( .A(RAM[48]), .B(n260), .Y(n87) );
  OAI21X1 U166 ( .A(n260), .B(n281), .C(n88), .Y(n139) );
  NAND2X1 U167 ( .A(RAM[49]), .B(n260), .Y(n88) );
  OAI21X1 U168 ( .A(n260), .B(n280), .C(n89), .Y(n140) );
  NAND2X1 U169 ( .A(RAM[50]), .B(n260), .Y(n89) );
  NAND3X1 U170 ( .A(n31), .B(n284), .C(wa[1]), .Y(n16) );
  INVX2 U19 ( .A(n271), .Y(n272) );
  INVX2 U20 ( .A(n32), .Y(n271) );
  INVX2 U21 ( .A(n263), .Y(n264) );
  INVX2 U22 ( .A(n69), .Y(n263) );
  INVX2 U23 ( .A(n2), .Y(n201) );
  INVX2 U24 ( .A(n1), .Y(n256) );
  INVX2 U25 ( .A(n269), .Y(n270) );
  INVX2 U26 ( .A(n42), .Y(n269) );
  INVX2 U27 ( .A(n265), .Y(n266) );
  INVX2 U28 ( .A(n60), .Y(n265) );
  INVX2 U29 ( .A(n267), .Y(n268) );
  INVX2 U30 ( .A(n51), .Y(n267) );
  INVX2 U31 ( .A(n273), .Y(n274) );
  INVX2 U171 ( .A(n22), .Y(n273) );
  INVX2 U172 ( .A(n259), .Y(n260) );
  INVX2 U173 ( .A(n16), .Y(n259) );
  INVX2 U174 ( .A(n261), .Y(n262) );
  INVX2 U175 ( .A(n78), .Y(n261) );
  OR2X1 U176 ( .A(n254), .B(n253), .Y(n1) );
  OR2X1 U177 ( .A(n199), .B(n198), .Y(n2) );
  INVX2 U178 ( .A(n6), .Y(n203) );
  INVX2 U179 ( .A(n4), .Y(n258) );
  INVX2 U180 ( .A(n3), .Y(n257) );
  INVX2 U181 ( .A(n7), .Y(n200) );
  INVX2 U182 ( .A(n5), .Y(n255) );
  INVX2 U183 ( .A(n8), .Y(n202) );
  OR2X1 U184 ( .A(ra2[1]), .B(ra2[2]), .Y(n3) );
  OR2X1 U185 ( .A(n253), .B(ra2[2]), .Y(n4) );
  OR2X1 U186 ( .A(n254), .B(ra2[1]), .Y(n5) );
  OR2X1 U187 ( .A(n198), .B(ra1[2]), .Y(n6) );
  OR2X1 U188 ( .A(n199), .B(ra1[1]), .Y(n7) );
  OR2X1 U189 ( .A(ra1[1]), .B(ra1[2]), .Y(n8) );
  AOI22X1 U190 ( .A(RAM[32]), .B(n200), .C(RAM[48]), .D(n201), .Y(n10) );
  AOI22X1 U191 ( .A(RAM[0]), .B(n202), .C(RAM[16]), .D(n203), .Y(n9) );
  AOI21X1 U192 ( .A(n10), .B(n9), .C(ra1[0]), .Y(n154) );
  AOI22X1 U193 ( .A(RAM[40]), .B(n200), .C(RAM[56]), .D(n201), .Y(n12) );
  AOI22X1 U194 ( .A(RAM[8]), .B(n202), .C(RAM[24]), .D(n203), .Y(n11) );
```

```
AOI21X1 U195 ( .A(n12), .B(n11), .C(n197), .Y(n13) );
OR2X1 U196 ( .A(n154), .B(n13), .Y(N44) );
AOI22X1 U197 ( .A(RAM[33]), .B(n200), .C(RAM[49]), .D(n201), .Y(n156) );
AOI22X1 U198 ( .A(RAM[1]), .B(n202), .C(RAM[17]), .D(n203), .Y(n155) );
AOI21X1 U199 ( .A(n156), .B(n155), .C(ra1[0]), .Y(n160) );
AOI22X1 U200 ( .A(RAM[41]), .B(n200), .C(RAM[57]), .D(n201), .Y(n158) );
AOI22X1 U201 ( .A(RAM[9]), .B(n202), .C(RAM[25]), .D(n203), .Y(n157) );
AOI21X1 U202 ( .A(n158), .B(n157), .C(n197), .Y(n159) );
OR2X1 U203 ( .A(n160), .B(n159), .Y(N43) );
AOI22X1 U204 ( .A(RAM[34]), .B(n200), .C(RAM[50]), .D(n201), .Y(n162) );
AOI22X1 U205 ( .A(RAM[2]), .B(n202), .C(RAM[18]), .D(n203), .Y(n161) );
AOI21X1 U206 ( .A(n162), .B(n161), .C(ra1[0]), .Y(n166) );
AOI22X1 U207 ( .A(RAM[42]), .B(n200), .C(RAM[58]), .D(n201), .Y(n164) );
AOI22X1 U208 ( .A(RAM[10]), .B(n202), .C(RAM[26]), .D(n203), .Y(n163) );
AOI21X1 U209 ( .A(n164), .B(n163), .C(n197), .Y(n165) );
OR2X1 U210 ( .A(n166), .B(n165), .Y(N42) );
AOI22X1 U211 ( .A(RAM[35]), .B(n200), .C(RAM[51]), .D(n201), .Y(n168) );
AOI22X1 U212 ( .A(RAM[3]), .B(n202), .C(RAM[19]), .D(n203), .Y(n167) );
AOI21X1 U213 ( .A(n168), .B(n167), .C(ra1[0]), .Y(n172) );
AOI22X1 U214 ( .A(RAM[43]), .B(n200), .C(RAM[59]), .D(n201), .Y(n170) );
AOI22X1 U215 ( .A(RAM[11]), .B(n202), .C(RAM[27]), .D(n203), .Y(n169) );
AOI21X1 U216 ( .A(n170), .B(n169), .C(n197), .Y(n171) );
OR2X1 U217 ( .A(n172), .B(n171), .Y(N41) );
```

```
AOI22X1 U218 ( .A(RAM[36]), .B(n200), .C(RAM[52]), .D(n201), .Y(n174) );
AOI22X1 U219 ( .A(RAM[4]), .B(n202), .C(RAM[20]), .D(n203), .Y(n173) );
AOI21X1 U220 ( .A(n174), .B(n173), .C(ra1[0]), .Y(n178) );
AOI22X1 U221 ( .A(RAM[44]), .B(n200), .C(RAM[60]), .D(n201), .Y(n176) );
AOI22X1 U222 ( .A(RAM[12]), .B(n202), .C(RAM[28]), .D(n203), .Y(n175) );
AOI21X1 U223 ( .A(n176), .B(n175), .C(n197), .Y(n177) );
OR2X1 U224 ( .A(n178), .B(n177), .Y(N40) );
AOI22X1 U225 ( .A(RAM[37]), .B(n200), .C(RAM[53]), .D(n201), .Y(n180) );
AOI22X1 U226 ( .A(RAM[5]), .B(n202), .C(RAM[21]), .D(n203), .Y(n179) );
AOI21X1 U227 ( .A(n180), .B(n179), .C(ra1[0]), .Y(n184) );
AOI22X1 U228 ( .A(RAM[45]), .B(n200), .C(RAM[61]), .D(n201), .Y(n182) );
AOI22X1 U229 ( .A(RAM[13]), .B(n202), .C(RAM[29]), .D(n203), .Y(n181) );
AOI21X1 U230 ( .A(n182), .B(n181), .C(n197), .Y(n183) );
OR2X1 U231 ( .A(n184), .B(n183), .Y(N39) );
AOI22X1 U232 ( .A(RAM[38]), .B(n200), .C(RAM[54]), .D(n201), .Y(n186) );
AOI22X1 U233 ( .A(RAM[6]), .B(n202), .C(RAM[22]), .D(n203), .Y(n185) );
AOI21X1 U234 ( .A(n186), .B(n185), .C(ra1[0]), .Y(n190) );
AOI22X1 U235 ( .A(RAM[46]), .B(n200), .C(RAM[62]), .D(n201), .Y(n188) );
AOI22X1 U236 ( .A(RAM[14]), .B(n202), .C(RAM[30]), .D(n203), .Y(n187) );
AOI21X1 U237 ( .A(n188), .B(n187), .C(n197), .Y(n189) );
OR2X1 U238 ( .A(n190), .B(n189), .Y(N38) );
AOI22X1 U239 ( .A(RAM[39]), .B(n200), .C(RAM[55]), .D(n201), .Y(n192) );
AOI22X1 U240 ( .A(RAM[7]), .B(n202), .C(RAM[23]), .D(n203), .Y(n191) );
```

```
AOI21X1 U241 ( .A(n192), .B(n191), .C(ra1[0]), .Y(n196) );
AOI22X1 U242 ( .A(RAM[47]), .B(n200), .C(RAM[63]), .D(n201), .Y(n194) );
AOI22X1 U243 ( .A(RAM[15]), .B(n202), .C(RAM[31]), .D(n203), .Y(n193) );
AOI21X1 U244 ( .A(n194), .B(n193), .C(n197), .Y(n195) );
OR2X1 U245 ( .A(n196), .B(n195), .Y(N37) );
INVX2 U246 ( .A(ra1[0]), .Y(n197) );
INVX2 U247 ( .A(ra1[1]), .Y(n198) );
INVX2 U248 ( .A(ra1[2]), .Y(n199) );
AOI22X1 U249 ( .A(RAM[32]), .B(n255), .C(RAM[48]), .D(n256), .Y(n205) );
AOI22X1 U250 ( .A(RAM[0]), .B(n257), .C(RAM[16]), .D(n258), .Y(n204) );
AOI21X1 U251 ( .A(n205), .B(n204), .C(ra2[0]), .Y(n209) );
AOI22X1 U252 ( .A(RAM[40]), .B(n255), .C(RAM[56]), .D(n256), .Y(n207) );
AOI22X1 U253 ( .A(RAM[8]), .B(n257), .C(RAM[24]), .D(n258), .Y(n206) );
AOI21X1 U254 ( .A(n207), .B(n206), .C(n252), .Y(n208) );
OR2X1 U255 ( .A(n209), .B(n208), .Y(N54) );
AOI22X1 U256 ( .A(RAM[33]), .B(n255), .C(RAM[49]), .D(n256), .Y(n211) );
AOI22X1 U257 ( .A(RAM[1]), .B(n257), .C(RAM[17]), .D(n258), .Y(n210) );
AOI21X1 U258 ( .A(n211), .B(n210), .C(ra2[0]), .Y(n215) );
AOI22X1 U259 ( .A(RAM[41]), .B(n255), .C(RAM[57]), .D(n256), .Y(n213) );
AOI22X1 U260 ( .A(RAM[9]), .B(n257), .C(RAM[25]), .D(n258), .Y(n212) );
AOI21X1 U261 ( .A(n213), .B(n212), .C(n252), .Y(n214) );
OR2X1 U262 ( .A(n215), .B(n214), .Y(N53) );
AOI22X1 U263 ( .A(RAM[34]), .B(n255), .C(RAM[50]), .D(n256), .Y(n217) );
```

```
  AOI22X1 U264 ( .A(RAM[2]), .B(n257), .C(RAM[18]), .D(n258), .Y(n216) );
  AOI21X1 U265 ( .A(n217), .B(n216), .C(ra2[0]), .Y(n221) );
  AOI22X1 U266 ( .A(RAM[42]), .B(n255), .C(RAM[58]), .D(n256), .Y(n219) );
  AOI22X1 U267 ( .A(RAM[10]), .B(n257), .C(RAM[26]), .D(n258), .Y(n218) );
  AOI21X1 U268 ( .A(n219), .B(n218), .C(n252), .Y(n220) );
  OR2X1 U269 ( .A(n221), .B(n220), .Y(N52) );
  AOI22X1 U270 ( .A(RAM[35]), .B(n255), .C(RAM[51]), .D(n256), .Y(n223) );
  AOI22X1 U271 ( .A(RAM[3]), .B(n257), .C(RAM[19]), .D(n258), .Y(n222) );
  AOI21X1 U272 ( .A(n223), .B(n222), .C(ra2[0]), .Y(n227) );
  AOI22X1 U273 ( .A(RAM[43]), .B(n255), .C(RAM[59]), .D(n256), .Y(n225) );
  AOI22X1 U274 ( .A(RAM[11]), .B(n257), .C(RAM[27]), .D(n258), .Y(n224) );
  AOI21X1 U275 ( .A(n225), .B(n224), .C(n252), .Y(n226) );
  OR2X1 U276 ( .A(n227), .B(n226), .Y(N51) );
  AOI22X1 U277 ( .A(RAM[36]), .B(n255), .C(RAM[52]), .D(n256), .Y(n229) );
  AOI22X1 U278 ( .A(RAM[4]), .B(n257), .C(RAM[20]), .D(n258), .Y(n228) );
  AOI21X1 U279 ( .A(n229), .B(n228), .C(ra2[0]), .Y(n233) );
  AOI22X1 U280 ( .A(RAM[44]), .B(n255), .C(RAM[60]), .D(n256), .Y(n231) );
  AOI22X1 U281 ( .A(RAM[12]), .B(n257), .C(RAM[28]), .D(n258), .Y(n230) );
  AOI21X1 U282 ( .A(n231), .B(n230), .C(n252), .Y(n232) );
  OR2X1 U283 ( .A(n233), .B(n232), .Y(N50) );
  AOI22X1 U284 ( .A(RAM[37]), .B(n255), .C(RAM[53]), .D(n256), .Y(n235) );
  AOI22X1 U285 ( .A(RAM[5]), .B(n257), .C(RAM[21]), .D(n258), .Y(n234) );
  AOI21X1 U286 ( .A(n235), .B(n234), .C(ra2[0]), .Y(n239) );
  AOI22X1 U287 ( .A(RAM[45]), .B(n255), .C(RAM[61]), .D(n256), .Y(n237) );
  AOI22X1 U288 ( .A(RAM[13]), .B(n257), .C(RAM[29]), .D(n258), .Y(n236) );
  AOI21X1 U289 ( .A(n237), .B(n236), .C(n252), .Y(n238) );
  OR2X1 U290 ( .A(n239), .B(n238), .Y(N49) );
  AOI22X1 U291 ( .A(RAM[38]), .B(n255), .C(RAM[54]), .D(n256), .Y(n241) );
  AOI22X1 U292 ( .A(RAM[6]), .B(n257), .C(RAM[22]), .D(n258), .Y(n240) );
  AOI21X1 U293 ( .A(n241), .B(n240), .C(ra2[0]), .Y(n245) );
  AOI22X1 U294 ( .A(RAM[46]), .B(n255), .C(RAM[62]), .D(n256), .Y(n243) );
  AOI22X1 U295 ( .A(RAM[14]), .B(n257), .C(RAM[30]), .D(n258), .Y(n242) );
  AOI21X1 U296 ( .A(n243), .B(n242), .C(n252), .Y(n244) );
  OR2X1 U297 ( .A(n245), .B(n244), .Y(N48) );
  AOI22X1 U298 ( .A(RAM[39]), .B(n255), .C(RAM[55]), .D(n256), .Y(n247) );
  AOI22X1 U299 ( .A(RAM[7]), .B(n257), .C(RAM[23]), .D(n258), .Y(n246) );
  AOI21X1 U300 ( .A(n247), .B(n246), .C(ra2[0]), .Y(n251) );
  AOI22X1 U301 ( .A(RAM[47]), .B(n255), .C(RAM[63]), .D(n256), .Y(n249) );
  AOI22X1 U302 ( .A(RAM[15]), .B(n257), .C(RAM[31]), .D(n258), .Y(n248) );
  AOI21X1 U303 ( .A(n249), .B(n248), .C(n252), .Y(n250) );
  OR2X1 U304 ( .A(n251), .B(n250), .Y(N47) );
  INVX2 U305 ( .A(ra2[0]), .Y(n252) );
  INVX2 U306 ( .A(ra2[1]), .Y(n253) );
  INVX2 U307 ( .A(ra2[2]), .Y(n254) );
  INVX2 U308 ( .A(wd[7]), .Y(n275) );
  INVX2 U309 ( .A(wd[6]), .Y(n276) );
  INVX2 U310 ( .A(wd[5]), .Y(n277) );
  INVX2 U311 ( .A(wd[4]), .Y(n278) );
  INVX2 U312 ( .A(wd[3]), .Y(n279) );
  INVX2 U313 ( .A(wd[2]), .Y(n280) );
  INVX2 U314 ( .A(wd[1]), .Y(n281) );
  INVX2 U315 ( .A(wd[0]), .Y(n282) );
  INVX2 U316 ( .A(wa[1]), .Y(n283) );
  INVX2 U317 ( .A(wa[0]), .Y(n284) );
  INVX2 U318 ( .A(n15), .Y(n285) );
  INVX2 U319 ( .A(n14), .Y(n286) );
  INVX2 U320 ( .A(regwrite), .Y(n287) );
endmodule


module alu_WIDTH8_DW01_add_0 ( A, B, CI, SUM, CO );
  input [7:0] A;
  input [7:0] B;
  output [7:0] SUM;
  input CI;
  output CO;

  wire   [7:1] carry;
  FAX1 U1_7 ( .A(A[7]), .B(B[7]), .C(carry[7]), .YS(SUM[7]) );
  FAX1 U1_6 ( .A(A[6]), .B(B[6]), .C(carry[6]), .YC(carry[7]), .YS(SUM[6]) );
  FAX1 U1_5 ( .A(A[5]), .B(B[5]), .C(carry[5]), .YC(carry[6]), .YS(SUM[5]) );
  FAX1 U1_4 ( .A(A[4]), .B(B[4]), .C(carry[4]), .YC(carry[5]), .YS(SUM[4]) );
  FAX1 U1_3 ( .A(A[3]), .B(B[3]), .C(carry[3]), .YC(carry[4]), .YS(SUM[3]) );
  FAX1 U1_2 ( .A(A[2]), .B(B[2]), .C(carry[2]), .YC(carry[3]), .YS(SUM[2]) );
  FAX1 U1_1 ( .A(A[1]), .B(B[1]), .C(carry[1]), .YC(carry[2]), .YS(SUM[1]) );
  FAX1 U1_0 ( .A(A[0]), .B(B[0]), .C(CI), .YC(carry[1]), .YS(SUM[0]) );
endmodule


module alu_WIDTH8 ( a, b, alucont, result );
  input [7:0] a;
  input [7:0] b;
  input [2:0] alucont;
  output [7:0] result;
  wire   n13, n14, n15, n16, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27,
         n28, n29, n30, n31, n32, n33, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10,
         n11, n12, n17;
  wire   [7:0] b2;
  wire   [7:0] sum;

  AND2X2 U2 ( .A(alucont[1]), .B(sum[7]), .Y(n32) );
  OAI21X1 U13 ( .A(n12), .B(n13), .C(n14), .Y(result[7]) );
  AOI22X1 U14 ( .A(sum[7]), .B(n15), .C(n1), .D(n16), .Y(n14) );
  OR2X1 U15 ( .A(a[7]), .B(b[7]), .Y(n16) );
  NAND2X1 U16 ( .A(a[7]), .B(n17), .Y(n13) );
  OAI21X1 U17 ( .A(n4), .B(n5), .C(n18), .Y(result[6]) );
  AOI22X1 U18 ( .A(b[6]), .B(n19), .C(sum[6]), .D(n15), .Y(n18) );
  OAI21X1 U19 ( .A(alucont[1]), .B(n5), .C(n4), .Y(n19) );
  OAI21X1 U20 ( .A(n4), .B(n6), .C(n20), .Y(result[5]) );
  AOI22X1 U21 ( .A(b[5]), .B(n21), .C(sum[5]), .D(n15), .Y(n20) );
  OAI21X1 U22 ( .A(alucont[1]), .B(n6), .C(n4), .Y(n21) );
  OAI21X1 U23 ( .A(n4), .B(n7), .C(n22), .Y(result[4]) );
  AOI22X1 U24 ( .A(b[4]), .B(n23), .C(sum[4]), .D(n15), .Y(n22) );
  OAI21X1 U25 ( .A(alucont[1]), .B(n7), .C(n4), .Y(n23) );
  OAI21X1 U26 ( .A(n4), .B(n8), .C(n24), .Y(result[3]) );
  AOI22X1 U27 ( .A(b[3]), .B(n25), .C(sum[3]), .D(n15), .Y(n24) );
  OAI21X1 U28 ( .A(alucont[1]), .B(n8), .C(n4), .Y(n25) );
  OAI21X1 U29 ( .A(n4), .B(n9), .C(n26), .Y(result[2]) );
  AOI22X1 U30 ( .A(b[2]), .B(n27), .C(sum[2]), .D(n15), .Y(n26) );
  OAI21X1 U31 ( .A(alucont[1]), .B(n9), .C(n4), .Y(n27) );
  OAI21X1 U32 ( .A(n4), .B(n10), .C(n28), .Y(result[1]) );
  AOI22X1 U33 ( .A(b[1]), .B(n29), .C(sum[1]), .D(n15), .Y(n28) );
  OAI21X1 U34 ( .A(alucont[1]), .B(n10), .C(n4), .Y(n29) );
  NAND2X1 U35 ( .A(n30), .B(n31), .Y(result[0]) );
  AOI22X1 U36 ( .A(n32), .B(alucont[0]), .C(b[0]), .D(n33), .Y(n31) );
  OAI21X1 U37 ( .A(alucont[1]), .B(n11), .C(n4), .Y(n33) );
  AOI22X1 U38 ( .A(sum[0]), .B(n15), .C(a[0]), .D(n1), .Y(n30) );
  NOR2X1 U40 ( .A(n17), .B(alucont[0]), .Y(n15) );
  XNOR2X1 U41 ( .A(n12), .B(alucont[2]), .Y(b2[7]) );
  XOR2X1 U42 ( .A(b[6]), .B(alucont[2]), .Y(b2[6]) );
  XOR2X1 U43 ( .A(b[5]), .B(alucont[2]), .Y(b2[5]) );
  XOR2X1 U44 ( .A(b[4]), .B(n3), .Y(b2[4]) );
  XOR2X1 U45 ( .A(b[3]), .B(n3), .Y(b2[3]) );
  XOR2X1 U46 ( .A(b[2]), .B(n3), .Y(b2[2]) );
  XOR2X1 U47 ( .A(b[1]), .B(n3), .Y(b2[1]) );
  XOR2X1 U48 ( .A(b[0]), .B(n3), .Y(b2[0]) );
  alu_WIDTH8_DW01_add_0 add_1_root_add_61_2 ( .A(a), .B(b2), .CI(n3), .SUM(sum) );
  AND2X2 U3 ( .A(alucont[0]), .B(n17), .Y(n1) );
  INVX2 U4 ( .A(n2), .Y(n3) );
  INVX2 U5 ( .A(alucont[2]), .Y(n2) );
  INVX2 U6 ( .A(n1), .Y(n4) );
  INVX2 U7 ( .A(a[6]), .Y(n5) );
  INVX2 U8 ( .A(a[5]), .Y(n6) );
  INVX2 U9 ( .A(a[4]), .Y(n7) );
  INVX2 U10 ( .A(a[3]), .Y(n8) );
  INVX2 U11 ( .A(a[2]), .Y(n9) );
  INVX2 U12 ( .A(a[1]), .Y(n10) );
  INVX2 U39 ( .A(a[0]), .Y(n11) );
  INVX2 U49 ( .A(b[7]), .Y(n12) );
  INVX2 U50 ( .A(alucont[1]), .Y(n17) );
endmodule


module zerodetect_WIDTH8 ( a, y );
  input [7:0] a;
  output y;
  wire   n1, n2, n3, n4, n5, n6;

  NOR2X1 U1 ( .A(n1), .B(n2), .Y(y) );
  NAND2X1 U2 ( .A(n3), .B(n4), .Y(n2) );
  NOR2X1 U3 ( .A(a[3]), .B(a[2]), .Y(n4) );
  NOR2X1 U4 ( .A(a[1]), .B(a[0]), .Y(n3) );
  NAND2X1 U5 ( .A(n5), .B(n6), .Y(n1) );
  NOR2X1 U6 ( .A(a[7]), .B(a[6]), .Y(n6) );
  NOR2X1 U7 ( .A(a[5]), .B(a[4]), .Y(n5) );
endmodule


module datapath_WIDTH8_REGBITS3 ( clk, reset, const_gnd, memdata, alusrca,
        memtoreg, iord, pcen, regwrite, regdst, pcsource, alusrcb, irwrite,
        alucont, zero, instr, adr, writedata );
  input [7:0] memdata;
  input [1:0] pcsource;
  input [1:0] alusrcb;
  input [3:0] irwrite;
  input [2:0] alucont;
  output [31:0] instr;
  output [7:0] adr;
  output [7:0] writedata;
  input clk, reset, const_gnd, alusrca, memtoreg, iord, pcen, regwrite, regdst;
  output zero;
  wire   n1, n2;
  wire   [2:0] wa;
  wire   [7:0] nextpc;
  wire   [7:0] pc;
  wire   [7:0] md;
  wire   [7:0] rd1;
  wire   [7:0] a;
  wire   [7:0] rd2;
  wire   [7:0] aluresult;
  wire   [7:0] aluout;
  wire   [7:0] src1;
  wire   [7:0] src2;
```

```
  wire   [7:0] wd;


  mux2_WIDTH3 regmux ( .d0(instr[18:16]), .d1(instr[13:11]), .s(regdst), .y(wa) );

  dffen_WIDTH8_3 ir0 ( .clk(clk), .en(irwrite[3]), .d(memdata), .q(instr[7:0])

         );

  dffen_WIDTH8_2 ir1 ( .clk(clk), .en(irwrite[2]), .d(memdata), .q(instr[15:8]) );

  dffen_WIDTH8_1 ir2 ( .clk(clk), .en(irwrite[1]), .d(memdata), .q(

        instr[23:16]) );

  dffen_WIDTH8_0 ir3 ( .clk(clk), .en(irwrite[0]), .d(memdata), .q(

        instr[31:24]) );

  dffenr_WIDTH8 pcreg ( .clk(clk), .reset(reset), .en(pcen), .d(nextpc), .q(pc) );

  dff_WIDTH8_3 mdr ( .clk(clk), .d(memdata), .q(md) );

  dff_WIDTH8_2 areg ( .clk(clk), .d(rd1), .q(a) );

  dff_WIDTH8_1 wrd ( .clk(clk), .d(rd2), .q(writedata) );

  dff_WIDTH8_0 res ( .clk(clk), .d(aluresult), .q(aluout) );

  mux2_WIDTH8_2 adrmux ( .d0(pc), .d1(aluout), .s(iord), .y(adr) );

  mux2_WIDTH8_1 src1mux ( .d0(pc), .d1(a), .s(alusrca), .y(src1) );

  mux4_WIDTH8_1 src2mux ( .d0(writedata), .d1({n2, n2, n2, n2, n2, n2, n2, n1}), .d2(instr[7:0]),
.d3({instr[5:0], n2, n2}), .s(alusrcb), .y(src2) );

  mux4_WIDTH8_0 pcmux ( .d0(aluresult), .d1(aluout), .d2({instr[5:0], n2, n2}),

        .d3({n2, n2, n2, n2, n2, n2, n2, n2}), .s(pcsource), .y(nextpc) );

  mux2_WIDTH8_0 wdmux ( .d0(aluout), .d1(md), .s(memtoreg), .y(wd) );

  regfile_WIDTH8_REGBITS3 rf ( .clk(clk), .regwrite(regwrite), .ra1(
        instr[23:21]), .ra2(instr[18:16]), .wa(wa), .wd(wd), .rd1(rd1), .rd2(
        rd2) );
  alu_WIDTH8 alunit ( .a(src1), .b(src2), .alucont(alucont), .result(aluresult) );
  zerodetect_WIDTH8 zd ( .a(aluresult), .y(zero) );
  INVX2 U1 ( .A(n1), .Y(n2) );
  INVX2 U2 ( .A(const_gnd), .Y(n1) );
endmodule


module mips ( clk, reset, const_gnd, memdata, memread, memwrite, adr,
        writedata );
  input [7:0] memdata;
  output [7:0] adr;
  output [7:0] writedata;
  input clk, reset, const_gnd;
  output memread, memwrite;
  wire   zero, alusrca, memtoreg, iord, pcen, regwrite, regdst, n1, n2, n3, n4,
         n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18,
         SYNOPSYS_UNCONNECTED_1, SYNOPSYS_UNCONNECTED_2,
         SYNOPSYS_UNCONNECTED_3, SYNOPSYS_UNCONNECTED_4,
         SYNOPSYS_UNCONNECTED_5, SYNOPSYS_UNCONNECTED_6,
         SYNOPSYS_UNCONNECTED_7, SYNOPSYS_UNCONNECTED_8,
         SYNOPSYS_UNCONNECTED_9, SYNOPSYS_UNCONNECTED_10,
        SYNOPSYS_UNCONNECTED_11, SYNOPSYS_UNCONNECTED_12,
        SYNOPSYS_UNCONNECTED_13, SYNOPSYS_UNCONNECTED_14,
        SYNOPSYS_UNCONNECTED_15, SYNOPSYS_UNCONNECTED_16,
        SYNOPSYS_UNCONNECTED_17, SYNOPSYS_UNCONNECTED_18,
        SYNOPSYS_UNCONNECTED_19, SYNOPSYS_UNCONNECTED_20;
  wire   [31:0] instr;
  wire   [1:0] pcsource;
  wire   [1:0] alusrcb;
  wire   [1:0] aluop;
  wire   [3:0] irwrite;
  wire   [2:0] alucont;

  controller cont ( .alusrca(alusrca), .alusrcb(alusrcb), .aluop(aluop),
        .pcen(pcen), .iord(iord), .irwrite(irwrite), .memread(memread),
        .memwrite(memwrite), .memtoreg(memtoreg), .pcsource(pcsource),
        .regwrite(regwrite), .regdst(regdst), .op(instr[31:26]), .clk(clk),
        .reset(n2), .zero(zero) );
  alucontrol ac ( .alucont(alucont), .aluop(aluop), .funct(instr[5:0]) );
  datapath_WIDTH8_REGBITS3 dp ( .clk(clk), .reset(n2), .const_gnd(const_gnd),
        .memdata({n18, n16, n14, n12, n10, n8, n6, n4}), .alusrca(alusrca),
        .memtoreg(memtoreg), .iord(iord), .pcen(pcen), .regwrite(regwrite),
        .regdst(regdst), .pcsource(pcsource), .alusrcb(alusrcb), .irwrite(
        irwrite), .alucont(alucont), .zero(zero), .instr({instr[31:26],
        SYNOPSYS_UNCONNECTED_1, SYNOPSYS_UNCONNECTED_2, SYNOPSYS_UNCONNECTED_3,
        SYNOPSYS_UNCONNECTED_4, SYNOPSYS_UNCONNECTED_5, SYNOPSYS_UNCONNECTED_6,
        SYNOPSYS_UNCONNECTED_7, SYNOPSYS_UNCONNECTED_8, SYNOPSYS_UNCONNECTED_9,
        SYNOPSYS_UNCONNECTED_10, SYNOPSYS_UNCONNECTED_11,
        SYNOPSYS_UNCONNECTED_12, SYNOPSYS_UNCONNECTED_13,
        SYNOPSYS_UNCONNECTED_14, SYNOPSYS_UNCONNECTED_15,
        SYNOPSYS_UNCONNECTED_16, SYNOPSYS_UNCONNECTED_17,
        SYNOPSYS_UNCONNECTED_18, SYNOPSYS_UNCONNECTED_19,
        SYNOPSYS_UNCONNECTED_20, instr[5:0]}), .adr(adr), .writedata(writedata) );
  INVX2 U1 ( .A(reset), .Y(n1) );
  INVX2 U2 ( .A(n1), .Y(n2) );
  INVX2 U3 ( .A(memdata[0]), .Y(n3) );
  INVX2 U4 ( .A(n3), .Y(n4) );
  INVX2 U5 ( .A(memdata[1]), .Y(n5) );
  INVX2 U6 ( .A(n5), .Y(n6) );
  INVX2 U7 ( .A(memdata[2]), .Y(n7) );
  INVX2 U8 ( .A(n7), .Y(n8) );
  INVX2 U9 ( .A(memdata[3]), .Y(n9) );
  INVX2 U10 ( .A(n9), .Y(n10) );
  INVX2 U11 ( .A(memdata[4]), .Y(n11) );
  INVX2 U12 ( .A(n11), .Y(n12) );
  INVX2 U13 ( .A(memdata[5]), .Y(n13) );
  INVX2 U14 ( .A(n13), .Y(n14) );
  INVX2 U15 ( .A(memdata[6]), .Y(n15) );
  INVX2 U16 ( .A(n15), .Y(n16) );
  INVX2 U17 ( .A(memdata[7]), .Y(n17) );
  INVX2 U18 ( .A(n17), .Y(n18) );
endmodule
```

Mips_scan.v

```
/////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : O-2018.06-SP1
// Date      : Sat Apr 30 16:39:43 2022
/////////////////////////////////////////////////////////////


module alucontrol ( alucont, aluop, funct );
  output [2:0] alucont;
  input [1:0] aluop;
  input [5:0] funct;
  wire   n8, n9, n10, n11, n12, n13, n14, n7, n15, n16, n17, n18, n19;

  INVX2 U3 ( .A(n13), .Y(alucont[0]) );
  OAI21X1 U10 ( .A(aluop[1]), .B(n19), .C(n8), .Y(alucont[2]) );
  OAI21X1 U11 ( .A(n9), .B(n10), .C(aluop[1]), .Y(n8) );
  OAI21X1 U12 ( .A(funct[2]), .B(n18), .C(n17), .Y(n10) );
  OAI21X1 U13 ( .A(n11), .B(n12), .C(aluop[1]), .Y(alucont[1]) );
  OAI21X1 U14 ( .A(funct[1]), .B(n15), .C(funct[5]), .Y(n12) );
  NAND3X1 U15 ( .A(n16), .B(n7), .C(n18), .Y(n11) );
  OAI21X1 U16 ( .A(n9), .B(n14), .C(aluop[1]), .Y(n13) );
  OAI21X1 U17 ( .A(n17), .B(n16), .C(n18), .Y(n14) );
  NAND3X1 U18 ( .A(n15), .B(n7), .C(funct[5]), .Y(n9) );
  INVX2 U4 ( .A(funct[4]), .Y(n7) );
  INVX2 U5 ( .A(funct[3]), .Y(n15) );
  INVX2 U6 ( .A(funct[2]), .Y(n16) );
  INVX2 U7 ( .A(funct[1]), .Y(n17) );
  INVX2 U8 ( .A(funct[0]), .Y(n18) );
  INVX2 U9 ( .A(aluop[0]), .Y(n19) );
endmodule


module mux2_WIDTH3 ( d0, d1, s, y );
  input [2:0] d0;
  input [2:0] d1;
  output [2:0] y;
  input s;
  wire   n5, n6, n7, n2;
  INVX2 U1 ( .A(n5), .Y(y[2]) );
  INVX2 U2 ( .A(n6), .Y(y[1]) );
  INVX2 U3 ( .A(n7), .Y(y[0]) );
  AOI22X1 U5 ( .A(d0[2]), .B(n2), .C(s), .D(d1[2]), .Y(n5) );
  AOI22X1 U6 ( .A(d0[1]), .B(n2), .C(d1[1]), .D(s), .Y(n6) );
  AOI22X1 U7 ( .A(d0[0]), .B(n2), .C(d1[0]), .D(s), .Y(n7) );
  INVX2 U4 ( .A(s), .Y(n2) );
endmodule


module mux2_WIDTH8_2 ( d0, d1, s, y );
  input [7:0] d0;
  input [7:0] d1;
  output [7:0] y;
  input s;
  wire   n10, n11, n12, n13, n14, n15, n16, n17, n2;

  INVX2 U1 ( .A(n10), .Y(y[7]) );
  INVX2 U2 ( .A(n11), .Y(y[6]) );
  INVX2 U3 ( .A(n12), .Y(y[5]) );
  INVX2 U4 ( .A(n13), .Y(y[4]) );
  INVX2 U5 ( .A(n14), .Y(y[3]) );
  INVX2 U6 ( .A(n15), .Y(y[2]) );

  INVX2 U7 ( .A(n16), .Y(y[1]) );

  INVX2 U8 ( .A(n17), .Y(y[0]) );

  AOI22X1 U10 ( .A(d0[7]), .B(n2), .C(s), .D(d1[7]), .Y(n10) );

  AOI22X1 U11 ( .A(d0[6]), .B(n2), .C(d1[6]), .D(s), .Y(n11) );

  AOI22X1 U12 ( .A(d0[5]), .B(n2), .C(d1[5]), .D(s), .Y(n12) );

  AOI22X1 U13 ( .A(d0[4]), .B(n2), .C(d1[4]), .D(s), .Y(n13) );

  AOI22X1 U14 ( .A(d0[3]), .B(n2), .C(d1[3]), .D(s), .Y(n14) );

  AOI22X1 U15 ( .A(d0[2]), .B(n2), .C(d1[2]), .D(s), .Y(n15) );

  AOI22X1 U16 ( .A(d0[1]), .B(n2), .C(d1[1]), .D(s), .Y(n16) );

  AOI22X1 U17 ( .A(d0[0]), .B(n2), .C(d1[0]), .D(s), .Y(n17) );

  INVX2 U9 ( .A(s), .Y(n2) );

endmodule



module mux2_WIDTH8_1 ( d0, d1, s, y );

  input [7:0] d0;

  input [7:0] d1;

  output [7:0] y;

  input s;

  wire   n10, n11, n12, n13, n14, n15, n16, n17, n18;


  INVX2 U1 ( .A(n10), .Y(y[7]) );
  INVX2 U2 ( .A(n11), .Y(y[6]) );
  INVX2 U3 ( .A(n12), .Y(y[5]) );
  INVX2 U4 ( .A(n13), .Y(y[4]) );
  INVX2 U5 ( .A(n14), .Y(y[3]) );
  INVX2 U6 ( .A(n15), .Y(y[2]) );
  INVX2 U7 ( .A(n16), .Y(y[1]) );
  INVX2 U8 ( .A(n17), .Y(y[0]) );
  AOI22X1 U10 ( .A(d0[7]), .B(n18), .C(s), .D(d1[7]), .Y(n10) );
  AOI22X1 U11 ( .A(d0[6]), .B(n18), .C(d1[6]), .D(s), .Y(n11) );
  AOI22X1 U12 ( .A(d0[5]), .B(n18), .C(d1[5]), .D(s), .Y(n12) );
  AOI22X1 U13 ( .A(d0[4]), .B(n18), .C(d1[4]), .D(s), .Y(n13) );
  AOI22X1 U14 ( .A(d0[3]), .B(n18), .C(d1[3]), .D(s), .Y(n14) );
  AOI22X1 U15 ( .A(d0[2]), .B(n18), .C(d1[2]), .D(s), .Y(n15) );
  AOI22X1 U16 ( .A(d0[1]), .B(n18), .C(d1[1]), .D(s), .Y(n16) );
  AOI22X1 U17 ( .A(d0[0]), .B(n18), .C(d1[0]), .D(s), .Y(n17) );
  INVX2 U9 ( .A(s), .Y(n18) );
endmodule


module mux4_WIDTH8_1 ( d0, d1, d2, d3, s, y );
  input [7:0] d0;
  input [7:0] d1;
  input [7:0] d2;
  input [7:0] d3;
  input [1:0] s;
  output [7:0] y;
  wire   n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16,
         n17, n18, n19, n20, n21, n22;

  AND2X2 U1 ( .A(s[0]), .B(s[1]), .Y(n5) );
  AND2X2 U2 ( .A(s[1]), .B(n22), .Y(n4) );
  NAND2X1 U4 ( .A(n2), .B(n3), .Y(y[7]) );
  AOI22X1 U5 ( .A(d2[7]), .B(n4), .C(d3[7]), .D(n5), .Y(n3) );
  AOI22X1 U6 ( .A(d0[7]), .B(n6), .C(d1[7]), .D(n7), .Y(n2) );
  NAND2X1 U7 ( .A(n8), .B(n9), .Y(y[6]) );
  AOI22X1 U8 ( .A(d2[6]), .B(n4), .C(d3[6]), .D(n5), .Y(n9) );
  AOI22X1 U9 ( .A(d0[6]), .B(n6), .C(d1[6]), .D(n7), .Y(n8) );
  NAND2X1 U10 ( .A(n10), .B(n11), .Y(y[5]) );
  AOI22X1 U11 ( .A(d2[5]), .B(n4), .C(d3[5]), .D(n5), .Y(n11) );
  AOI22X1 U12 ( .A(d0[5]), .B(n6), .C(d1[5]), .D(n7), .Y(n10) );
  NAND2X1 U13 ( .A(n12), .B(n13), .Y(y[4]) );
  AOI22X1 U14 ( .A(d2[4]), .B(n4), .C(d3[4]), .D(n5), .Y(n13) );
  AOI22X1 U15 ( .A(d0[4]), .B(n6), .C(d1[4]), .D(n7), .Y(n12) );
  NAND2X1 U16 ( .A(n14), .B(n15), .Y(y[3]) );
  AOI22X1 U17 ( .A(d2[3]), .B(n4), .C(d3[3]), .D(n5), .Y(n15) );
  AOI22X1 U18 ( .A(d0[3]), .B(n6), .C(d1[3]), .D(n7), .Y(n14) );
  NAND2X1 U19 ( .A(n16), .B(n17), .Y(y[2]) );
  AOI22X1 U20 ( .A(d2[2]), .B(n4), .C(d3[2]), .D(n5), .Y(n17) );
  AOI22X1 U21 ( .A(d0[2]), .B(n6), .C(d1[2]), .D(n7), .Y(n16) );
  NAND2X1 U22 ( .A(n18), .B(n19), .Y(y[1]) );
  AOI22X1 U23 ( .A(d2[1]), .B(n4), .C(d3[1]), .D(n5), .Y(n19) );
  AOI22X1 U24 ( .A(d0[1]), .B(n6), .C(d1[1]), .D(n7), .Y(n18) );
  NAND2X1 U25 ( .A(n20), .B(n21), .Y(y[0]) );
  AOI22X1 U26 ( .A(d2[0]), .B(n4), .C(d3[0]), .D(n5), .Y(n21) );
  AOI22X1 U27 ( .A(d0[0]), .B(n6), .C(d1[0]), .D(n7), .Y(n20) );
  NOR2X1 U28 ( .A(n22), .B(s[1]), .Y(n7) );
  NOR2X1 U29 ( .A(s[0]), .B(s[1]), .Y(n6) );
  INVX2 U3 ( .A(s[0]), .Y(n22) );
endmodule


module mux4_WIDTH8_0 ( d0, d1, d2, d3, s, y );
  input [7:0] d0;
  input [7:0] d1;
  input [7:0] d2;
  input [7:0] d3;
  input [1:0] s;
  output [7:0] y;
  wire   n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16,
       n17, n18, n19, n20, n21, n42;

  AND2X2 U1 ( .A(s[0]), .B(s[1]), .Y(n5) );
  AND2X2 U2 ( .A(s[1]), .B(n42), .Y(n4) );
  NAND2X1 U4 ( .A(n2), .B(n3), .Y(y[7]) );
  AOI22X1 U5 ( .A(d2[7]), .B(n4), .C(d3[7]), .D(n5), .Y(n3) );
  AOI22X1 U6 ( .A(d0[7]), .B(n6), .C(d1[7]), .D(n7), .Y(n2) );
  NAND2X1 U7 ( .A(n8), .B(n9), .Y(y[6]) );
  AOI22X1 U8 ( .A(d2[6]), .B(n4), .C(d3[6]), .D(n5), .Y(n9) );
  AOI22X1 U9 ( .A(d0[6]), .B(n6), .C(d1[6]), .D(n7), .Y(n8) );
  NAND2X1 U10 ( .A(n10), .B(n11), .Y(y[5]) );
  AOI22X1 U11 ( .A(d2[5]), .B(n4), .C(d3[5]), .D(n5), .Y(n11) );
  AOI22X1 U12 ( .A(d0[5]), .B(n6), .C(d1[5]), .D(n7), .Y(n10) );
  NAND2X1 U13 ( .A(n12), .B(n13), .Y(y[4]) );
  AOI22X1 U14 ( .A(d2[4]), .B(n4), .C(d3[4]), .D(n5), .Y(n13) );
  AOI22X1 U15 ( .A(d0[4]), .B(n6), .C(d1[4]), .D(n7), .Y(n12) );
  NAND2X1 U16 ( .A(n14), .B(n15), .Y(y[3]) );
  AOI22X1 U17 ( .A(d2[3]), .B(n4), .C(d3[3]), .D(n5), .Y(n15) );
  AOI22X1 U18 ( .A(d0[3]), .B(n6), .C(d1[3]), .D(n7), .Y(n14) );
  NAND2X1 U19 ( .A(n16), .B(n17), .Y(y[2]) );
  AOI22X1 U20 ( .A(d2[2]), .B(n4), .C(d3[2]), .D(n5), .Y(n17) );
  AOI22X1 U21 ( .A(d0[2]), .B(n6), .C(d1[2]), .D(n7), .Y(n16) );
  NAND2X1 U22 ( .A(n18), .B(n19), .Y(y[1]) );
  AOI22X1 U23 ( .A(d2[1]), .B(n4), .C(d3[1]), .D(n5), .Y(n19) );
  AOI22X1 U24 ( .A(d0[1]), .B(n6), .C(d1[1]), .D(n7), .Y(n18) );
  NAND2X1 U25 ( .A(n20), .B(n21), .Y(y[0]) );
  AOI22X1 U26 ( .A(d2[0]), .B(n4), .C(d3[0]), .D(n5), .Y(n21) );
  AOI22X1 U27 ( .A(d0[0]), .B(n6), .C(d1[0]), .D(n7), .Y(n20) );
  NOR2X1 U28 ( .A(n42), .B(s[1]), .Y(n7) );
  NOR2X1 U29 ( .A(s[0]), .B(s[1]), .Y(n6) );
  INVX2 U3 ( .A(s[0]), .Y(n42) );
endmodule


module mux2_WIDTH8_0 ( d0, d1, s, y );
  input [7:0] d0;
  input [7:0] d1;
  output [7:0] y;
  input s;
  wire   n18, n19, n20, n21, n22, n23, n24, n25, n26;

  INVX2 U1 ( .A(n26), .Y(y[7]) );
  INVX2 U2 ( .A(n25), .Y(y[6]) );
  INVX2 U3 ( .A(n24), .Y(y[5]) );
  INVX2 U4 ( .A(n23), .Y(y[4]) );
  INVX2 U5 ( .A(n22), .Y(y[3]) );
  INVX2 U6 ( .A(n21), .Y(y[2]) );
  INVX2 U7 ( .A(n20), .Y(y[1]) );
  INVX2 U8 ( .A(n19), .Y(y[0]) );
  AOI22X1 U10 ( .A(d0[7]), .B(n18), .C(s), .D(d1[7]), .Y(n26) );
  AOI22X1 U11 ( .A(d0[6]), .B(n18), .C(d1[6]), .D(s), .Y(n25) );
  AOI22X1 U12 ( .A(d0[5]), .B(n18), .C(d1[5]), .D(s), .Y(n24) );
  AOI22X1 U13 ( .A(d0[4]), .B(n18), .C(d1[4]), .D(s), .Y(n23) );
  AOI22X1 U14 ( .A(d0[3]), .B(n18), .C(d1[3]), .D(s), .Y(n22) );
  AOI22X1 U15 ( .A(d0[2]), .B(n18), .C(d1[2]), .D(s), .Y(n21) );
  AOI22X1 U16 ( .A(d0[1]), .B(n18), .C(d1[1]), .D(s), .Y(n20) );
  AOI22X1 U17 ( .A(d0[0]), .B(n18), .C(d1[0]), .D(s), .Y(n19) );
  INVX2 U9 ( .A(s), .Y(n18) );
endmodule


module alu_WIDTH8_DW01_add_0 ( A, B, CI, SUM, CO );
  input [7:0] A;
  input [7:0] B;
  output [7:0] SUM;
  input CI;
  output CO;

  wire   [7:1] carry;
  FAX1 U1_7 ( .A(A[7]), .B(B[7]), .C(carry[7]), .YS(SUM[7]) );
  FAX1 U1_6 ( .A(A[6]), .B(B[6]), .C(carry[6]), .YC(carry[7]), .YS(SUM[6]) );
  FAX1 U1_5 ( .A(A[5]), .B(B[5]), .C(carry[5]), .YC(carry[6]), .YS(SUM[5]) );
  FAX1 U1_4 ( .A(A[4]), .B(B[4]), .C(carry[4]), .YC(carry[5]), .YS(SUM[4]) );
  FAX1 U1_3 ( .A(A[3]), .B(B[3]), .C(carry[3]), .YC(carry[4]), .YS(SUM[3]) );
  FAX1 U1_2 ( .A(A[2]), .B(B[2]), .C(carry[2]), .YC(carry[3]), .YS(SUM[2]) );
  FAX1 U1_1 ( .A(A[1]), .B(B[1]), .C(carry[1]), .YC(carry[2]), .YS(SUM[1]) );
  FAX1 U1_0 ( .A(A[0]), .B(B[0]), .C(CI), .YC(carry[1]), .YS(SUM[0]) );
endmodule


module alu_WIDTH8 ( a, b, alucont, result );
  input [7:0] a;
  input [7:0] b;
  input [2:0] alucont;
  output [7:0] result;
  wire   n20, n21, n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32, n33,
         n34, n35, n36, n37, n38, n39, n18, n19, n40, n41, n42, n43, n44, n45,
         n46, n47, n48, n49, n50, n51, n52, n53, n54, n55, n56, n57, n58;
  wire   [7:0] b2;
  wire   [7:0] sum;
  OAI21X1 U21 ( .A(n20), .B(n41), .C(n21), .Y(result[7]) );
  AOI22X1 U22 ( .A(b[7]), .B(n22), .C(n56), .D(a[7]), .Y(n21) );
  OAI21X1 U23 ( .A(alucont[1]), .B(n42), .C(n23), .Y(n22) );
  OAI21X1 U24 ( .A(n20), .B(n43), .C(n24), .Y(result[6]) );
  AOI22X1 U25 ( .A(b[6]), .B(n25), .C(a[6]), .D(n56), .Y(n24) );
  OAI21X1 U26 ( .A(alucont[1]), .B(n44), .C(n23), .Y(n25) );
  OAI21X1 U27 ( .A(n20), .B(n45), .C(n26), .Y(result[5]) );
  AOI22X1 U28 ( .A(b[5]), .B(n27), .C(a[5]), .D(n56), .Y(n26) );
  OAI21X1 U29 ( .A(alucont[1]), .B(n46), .C(n23), .Y(n27) );
  OAI21X1 U30 ( .A(n20), .B(n47), .C(n28), .Y(result[4]) );
  AOI22X1 U31 ( .A(b[4]), .B(n29), .C(a[4]), .D(n56), .Y(n28) );
  OAI21X1 U32 ( .A(alucont[1]), .B(n48), .C(n23), .Y(n29) );
  OAI21X1 U33 ( .A(n20), .B(n49), .C(n30), .Y(result[3]) );
  AOI22X1 U34 ( .A(b[3]), .B(n31), .C(a[3]), .D(n56), .Y(n30) );
  OAI21X1 U35 ( .A(alucont[1]), .B(n50), .C(n23), .Y(n31) );
  OAI21X1 U36 ( .A(n20), .B(n51), .C(n32), .Y(result[2]) );
  AOI22X1 U37 ( .A(b[2]), .B(n33), .C(a[2]), .D(n56), .Y(n32) );
  OAI21X1 U38 ( .A(alucont[1]), .B(n52), .C(n23), .Y(n33) );
  OAI21X1 U39 ( .A(n20), .B(n53), .C(n34), .Y(result[1]) );
  AOI22X1 U40 ( .A(b[1]), .B(n35), .C(a[1]), .D(n56), .Y(n34) );
  OAI21X1 U41 ( .A(alucont[1]), .B(n54), .C(n23), .Y(n35) );
  NAND2X1 U42 ( .A(n36), .B(n37), .Y(result[0]) );
  AOI22X1 U43 ( .A(n38), .B(sum[7]), .C(b[0]), .D(n39), .Y(n37) );
  OAI21X1 U44 ( .A(alucont[1]), .B(n55), .C(n23), .Y(n39) );
  NOR2X1 U45 ( .A(n58), .B(n18), .Y(n38) );
  AOI22X1 U46 ( .A(a[0]), .B(n56), .C(sum[0]), .D(n57), .Y(n36) );
  NAND2X1 U47 ( .A(alucont[1]), .B(n58), .Y(n20) );
  NAND2X1 U48 ( .A(alucont[0]), .B(n18), .Y(n23) );
  XOR2X1 U49 ( .A(b[7]), .B(alucont[2]), .Y(b2[7]) );
  XOR2X1 U50 ( .A(b[6]), .B(alucont[2]), .Y(b2[6]) );
  XOR2X1 U51 ( .A(b[5]), .B(alucont[2]), .Y(b2[5]) );
  XOR2X1 U52 ( .A(b[4]), .B(n40), .Y(b2[4]) );
  XOR2X1 U53 ( .A(b[3]), .B(n40), .Y(b2[3]) );
  XOR2X1 U54 ( .A(b[2]), .B(n40), .Y(b2[2]) );
  XOR2X1 U55 ( .A(b[1]), .B(n40), .Y(b2[1]) );
  XOR2X1 U56 ( .A(b[0]), .B(n40), .Y(b2[0]) );
  alu_WIDTH8_DW01_add_0 add_1_root_add_61_2 ( .A(a), .B(b2), .CI(n40), .SUM(
        sum) );
  INVX2 U2 ( .A(n19), .Y(n40) );
  INVX2 U3 ( .A(alucont[2]), .Y(n19) );
  INVX2 U4 ( .A(alucont[1]), .Y(n18) );
  INVX2 U5 ( .A(sum[7]), .Y(n41) );
  INVX2 U6 ( .A(a[7]), .Y(n42) );
  INVX2 U7 ( .A(sum[6]), .Y(n43) );
  INVX2 U8 ( .A(a[6]), .Y(n44) );
  INVX2 U9 ( .A(sum[5]), .Y(n45) );
  INVX2 U10 ( .A(a[5]), .Y(n46) );
  INVX2 U11 ( .A(sum[4]), .Y(n47) );
  INVX2 U12 ( .A(a[4]), .Y(n48) );
  INVX2 U13 ( .A(sum[3]), .Y(n49) );
  INVX2 U14 ( .A(a[3]), .Y(n50) );
  INVX2 U15 ( .A(sum[2]), .Y(n51) );
  INVX2 U16 ( .A(a[2]), .Y(n52) );
  INVX2 U17 ( .A(sum[1]), .Y(n53) );
  INVX2 U18 ( .A(a[1]), .Y(n54) );
  INVX2 U19 ( .A(a[0]), .Y(n55) );
  INVX2 U20 ( .A(n23), .Y(n56) );
  INVX2 U57 ( .A(n20), .Y(n57) );
  INVX2 U58 ( .A(alucont[0]), .Y(n58) );
endmodule


module zerodetect_WIDTH8 ( a, y );
  input [7:0] a;
  output y;
  wire   n1, n2, n3, n4, n5, n6;

  NOR2X1 U1 ( .A(n1), .B(n2), .Y(y) );
  NAND2X1 U2 ( .A(n3), .B(n4), .Y(n2) );
  NOR2X1 U3 ( .A(a[3]), .B(a[2]), .Y(n4) );

  NOR2X1 U4 ( .A(a[1]), .B(a[0]), .Y(n3) );

  NAND2X1 U5 ( .A(n5), .B(n6), .Y(n1) );

  NOR2X1 U6 ( .A(a[7]), .B(a[6]), .Y(n6) );

  NOR2X1 U7 ( .A(a[5]), .B(a[4]), .Y(n5) );

endmodule



module dff_WIDTH8_test_0 ( clk, d, q, test_si, test_se );

  input [7:0] d;

  output [7:0] q;

  input clk, test_si, test_se;



  DFFPOSX1_SCAN q_reg_7_ ( .D(d[7]), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );

  DFFPOSX1_SCAN q_reg_6_ ( .D(d[6]), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );

  DFFPOSX1_SCAN q_reg_5_ ( .D(d[5]), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );

  DFFPOSX1_SCAN q_reg_4_ ( .D(d[4]), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );

  DFFPOSX1_SCAN q_reg_3_ ( .D(d[3]), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(d[2]), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(d[1]), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(d[0]), .TI(test_si), .TE(test_se), .CLK(clk),
        .Q(q[0]) );
endmodule


module dffen_WIDTH8_test_0 ( clk, en, d, q, test_si, test_se );
  input [7:0] d;
  output [7:0] q;
  input clk, en, test_si, test_se;
  wire   n10, n11, n12, n13, n14, n15, n16, n17, n34, n35, n36, n37, n38, n39,
         n40, n41, n42;

  AOI22X1 U11 ( .A(en), .B(d[7]), .C(q[7]), .D(n42), .Y(n10) );
  AOI22X1 U12 ( .A(d[6]), .B(en), .C(q[6]), .D(n42), .Y(n11) );
  AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n42), .Y(n12) );
  AOI22X1 U14 ( .A(d[4]), .B(en), .C(q[4]), .D(n42), .Y(n13) );
  AOI22X1 U15 ( .A(d[3]), .B(en), .C(q[3]), .D(n42), .Y(n14) );
  AOI22X1 U16 ( .A(d[2]), .B(en), .C(q[2]), .D(n42), .Y(n15) );
  AOI22X1 U17 ( .A(d[1]), .B(en), .C(q[1]), .D(n42), .Y(n16) );
  AOI22X1 U18 ( .A(d[0]), .B(en), .C(q[0]), .D(n42), .Y(n17) );
  DFFPOSX1_SCAN q_reg_7_ ( .D(n34), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(n35), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(n36), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(n37), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(n38), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(n39), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(n40), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(n41), .TI(test_si), .TE(test_se), .CLK(clk), .Q(
        q[0]) );
  INVX2 U26 ( .A(n10), .Y(n34) );
  INVX2 U27 ( .A(n11), .Y(n35) );
  INVX2 U28 ( .A(n12), .Y(n36) );
  INVX2 U29 ( .A(n13), .Y(n37) );
  INVX2 U30 ( .A(n14), .Y(n38) );
  INVX2 U31 ( .A(n15), .Y(n39) );
  INVX2 U32 ( .A(n16), .Y(n40) );
  INVX2 U33 ( .A(n17), .Y(n41) );
  INVX2 U34 ( .A(en), .Y(n42) );
endmodule


module dffen_WIDTH8_test_1 ( clk, en, d, q, test_si, test_se );
  input [7:0] d;
  output [7:0] q;
  input clk, en, test_si, test_se;
  wire   n10, n11, n12, n13, n14, n15, n16, n17, n42, n43, n44, n45, n46, n47,
         n48, n49, n50;

  AOI22X1 U11 ( .A(en), .B(d[7]), .C(q[7]), .D(n50), .Y(n10) );
  AOI22X1 U12 ( .A(d[6]), .B(en), .C(q[6]), .D(n50), .Y(n11) );
  AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n50), .Y(n12) );
  AOI22X1 U14 ( .A(d[4]), .B(en), .C(q[4]), .D(n50), .Y(n13) );
  AOI22X1 U15 ( .A(d[3]), .B(en), .C(q[3]), .D(n50), .Y(n14) );
  AOI22X1 U16 ( .A(d[2]), .B(en), .C(q[2]), .D(n50), .Y(n15) );
  AOI22X1 U17 ( .A(d[1]), .B(en), .C(q[1]), .D(n50), .Y(n16) );
  AOI22X1 U18 ( .A(d[0]), .B(en), .C(q[0]), .D(n50), .Y(n17) );
  DFFPOSX1_SCAN q_reg_7_ ( .D(n42), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(n43), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(n44), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(n45), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(n46), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(n47), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(n48), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(n49), .TI(test_si), .TE(test_se), .CLK(clk), .Q(
        q[0]) );
  INVX2 U26 ( .A(n10), .Y(n42) );
  INVX2 U27 ( .A(n11), .Y(n43) );
  INVX2 U28 ( .A(n12), .Y(n44) );
  INVX2 U29 ( .A(n13), .Y(n45) );
  INVX2 U30 ( .A(n14), .Y(n46) );
  INVX2 U31 ( .A(n15), .Y(n47) );
  INVX2 U32 ( .A(n16), .Y(n48) );
  INVX2 U33 ( .A(n17), .Y(n49) );
  INVX2 U34 ( .A(en), .Y(n50) );
endmodule


module dffen_WIDTH8_test_2 ( clk, en, d, q, test_si, test_se );
  input [7:0] d;
  output [7:0] q;
  input clk, en, test_si, test_se;
  wire   n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53, n54, n55,
         n56, n57, n58;

  AOI22X1 U11 ( .A(en), .B(d[7]), .C(q[7]), .D(n50), .Y(n58) );
  AOI22X1 U12 ( .A(d[6]), .B(en), .C(q[6]), .D(n50), .Y(n57) );
  AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n50), .Y(n56) );
  AOI22X1 U14 ( .A(d[4]), .B(en), .C(q[4]), .D(n50), .Y(n55) );
  AOI22X1 U15 ( .A(d[3]), .B(en), .C(q[3]), .D(n50), .Y(n54) );
  AOI22X1 U16 ( .A(d[2]), .B(en), .C(q[2]), .D(n50), .Y(n53) );
  AOI22X1 U17 ( .A(d[1]), .B(en), .C(q[1]), .D(n50), .Y(n52) );
  AOI22X1 U18 ( .A(d[0]), .B(en), .C(q[0]), .D(n50), .Y(n51) );
  DFFPOSX1_SCAN q_reg_7_ ( .D(n42), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(n43), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(n44), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(n45), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(n46), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(n47), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(n48), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(n49), .TI(test_si), .TE(test_se), .CLK(clk), .Q(
        q[0]) );
  INVX2 U26 ( .A(n58), .Y(n42) );
  INVX2 U27 ( .A(n57), .Y(n43) );
  INVX2 U28 ( .A(n56), .Y(n44) );
  INVX2 U29 ( .A(n55), .Y(n45) );
  INVX2 U30 ( .A(n54), .Y(n46) );
  INVX2 U31 ( .A(n53), .Y(n47) );
  INVX2 U32 ( .A(n52), .Y(n48) );
  INVX2 U33 ( .A(n51), .Y(n49) );
  INVX2 U34 ( .A(en), .Y(n50) );
endmodule
```

```
module dffen_WIDTH8_test_3 ( clk, en, d, q, test_si, test_se );
  input [7:0] d;
  output [7:0] q;
  input clk, en, test_si, test_se;
  wire   n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53, n54, n55,
         n56, n57, n58;

  AOI22X1 U11 ( .A(en), .B(d[7]), .C(q[7]), .D(n50), .Y(n58) );
  AOI22X1 U12 ( .A(d[6]), .B(en), .C(q[6]), .D(n50), .Y(n57) );
  AOI22X1 U13 ( .A(d[5]), .B(en), .C(q[5]), .D(n50), .Y(n56) );
  AOI22X1 U14 ( .A(d[4]), .B(en), .C(q[4]), .D(n50), .Y(n55) );
  AOI22X1 U15 ( .A(d[3]), .B(en), .C(q[3]), .D(n50), .Y(n54) );
  AOI22X1 U16 ( .A(d[2]), .B(en), .C(q[2]), .D(n50), .Y(n53) );
  AOI22X1 U17 ( .A(d[1]), .B(en), .C(q[1]), .D(n50), .Y(n52) );
  AOI22X1 U18 ( .A(d[0]), .B(en), .C(q[0]), .D(n50), .Y(n51) );
  DFFPOSX1_SCAN q_reg_7_ ( .D(n42), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(n43), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(n44), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(n45), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(n46), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(n47), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(n48), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(n49), .TI(test_si), .TE(test_se), .CLK(clk), .Q(
        q[0]) );
  INVX2 U26 ( .A(n58), .Y(n42) );
  INVX2 U27 ( .A(n57), .Y(n43) );
  INVX2 U28 ( .A(n56), .Y(n44) );
  INVX2 U29 ( .A(n55), .Y(n45) );
  INVX2 U30 ( .A(n54), .Y(n46) );
  INVX2 U31 ( .A(n53), .Y(n47) );
  INVX2 U32 ( .A(n52), .Y(n48) );
  INVX2 U33 ( .A(n51), .Y(n49) );
  INVX2 U34 ( .A(en), .Y(n50) );
endmodule



module dff_WIDTH8_test_1 ( clk, d, q, test_si, test_se );
  input [7:0] d;
  output [7:0] q;
  input clk, test_si, test_se;


  DFFPOSX1_SCAN q_reg_7_ ( .D(d[7]), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(d[6]), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(d[5]), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(d[4]), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(d[3]), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(d[2]), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(d[1]), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(d[0]), .TI(test_si), .TE(test_se), .CLK(clk),
        .Q(q[0]) );
endmodule
```

```
module dffenr_WIDTH8_test_1 ( clk, reset, en, d, q, test_si, test_se );
  input [7:0] d;
  output [7:0] q;
  input clk, reset, en, test_si, test_se;
  wire   n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n36, n37, n38, n39,
         n40, n41, n42, n43, n44;

  AOI22X1 U12 ( .A(q[7]), .B(n11), .C(d[7]), .D(n12), .Y(n10) );
  AOI22X1 U13 ( .A(q[6]), .B(n11), .C(d[6]), .D(n12), .Y(n13) );
  AOI22X1 U14 ( .A(q[5]), .B(n11), .C(d[5]), .D(n12), .Y(n14) );
  AOI22X1 U15 ( .A(q[4]), .B(n11), .C(d[4]), .D(n12), .Y(n15) );
  AOI22X1 U16 ( .A(q[3]), .B(n11), .C(d[3]), .D(n12), .Y(n16) );
  AOI22X1 U17 ( .A(q[2]), .B(n11), .C(d[2]), .D(n12), .Y(n17) );
  AOI22X1 U18 ( .A(q[1]), .B(n11), .C(d[1]), .D(n12), .Y(n18) );
  AOI22X1 U19 ( .A(q[0]), .B(n11), .C(d[0]), .D(n12), .Y(n19) );
  NOR2X1 U20 ( .A(n12), .B(reset), .Y(n11) );
  NOR2X1 U21 ( .A(n44), .B(reset), .Y(n12) );
  DFFPOSX1_SCAN q_reg_7_ ( .D(n43), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(n42), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(n41), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(n40), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(n39), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(n38), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(n37), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(n36), .TI(test_si), .TE(test_se), .CLK(clk), .Q(
        q[0]) );
  INVX2 U29 ( .A(n19), .Y(n36) );
  INVX2 U30 ( .A(n18), .Y(n37) );
  INVX2 U31 ( .A(n17), .Y(n38) );
  INVX2 U32 ( .A(n16), .Y(n39) );
  INVX2 U33 ( .A(n15), .Y(n40) );
  INVX2 U34 ( .A(n14), .Y(n41) );
  INVX2 U35 ( .A(n13), .Y(n42) );
  INVX2 U36 ( .A(n10), .Y(n43) );
  INVX2 U37 ( .A(en), .Y(n44) );
endmodule


module dff_WIDTH8_test_2 ( clk, d, q, test_si, test_se );
```

```
  input [7:0] d;
  output [7:0] q;
  input clk, test_si, test_se;


  DFFPOSX1_SCAN q_reg_7_ ( .D(d[7]), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(d[6]), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(d[5]), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(d[4]), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(d[3]), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(d[2]), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(d[1]), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(d[0]), .TI(test_si), .TE(test_se), .CLK(clk),
        .Q(q[0]) );
endmodule
```

```
module regfile_WIDTH8_REGBITS3_test_1 ( clk, regwrite, ra1, ra2, wa, wd, rd1,
        rd2, test_si, test_so, test_se );
  input [2:0] ra1;
  input [2:0] ra2;
  input [2:0] wa;
  input [7:0] wd;
  output [7:0] rd1;
  output [7:0] rd2;
  input clk, regwrite, test_si, test_se;
  output test_so;
  wire   RAM_62_, RAM_61_, RAM_60_, RAM_59_, RAM_58_, RAM_57_, RAM_56_,
         RAM_55_, RAM_54_, RAM_53_, RAM_52_, RAM_51_, RAM_50_, RAM_49_,
         RAM_48_, RAM_47_, RAM_46_, RAM_45_, RAM_44_, RAM_43_, RAM_42_,
         RAM_41_, RAM_40_, RAM_39_, RAM_38_, RAM_37_, RAM_36_, RAM_35_,
         RAM_34_, RAM_33_, RAM_32_, RAM_31_, RAM_30_, RAM_29_, RAM_28_,
         RAM_27_, RAM_26_, RAM_25_, RAM_24_, RAM_23_, RAM_22_, RAM_21_,
         RAM_20_, RAM_19_, RAM_18_, RAM_17_, RAM_16_, RAM_15_, RAM_14_,
         RAM_13_, RAM_12_, RAM_11_, RAM_10_, RAM_9_, RAM_8_, n98, n99, n100,
         n101, n102, n103, n104, n105, n106, n107, n108, n109, n110, n111,
         n112, n113, n114, n115, n116, n117, n118, n119, n120, n121, n130,
         n131, n132, n133, n134, n135, n136, n137, n138, n139, n140, n141,
         n142, n143, n144, n145, n146, n147, n148, n149, n150, n151, n152,
         n153, n79, n80, n81, n82, n84, n86, n87, n88, n89, n90, n91, n92, n93,
         n94, n95, n96, n97, n122, n123, n124, n125, n126, n127, n128, n129,
         n154, n155, n156, n157, n158, n159, n160, n161, n162, n163, n164,
         n165, n166, n167, n168, n169, n170, n171, n172, n173, n174, n176,
         n178, n179, n180, n181, n182, n183, n184, n185, n186, n187, n188,
         n189, n190, n191, n192, n193, n194, n195, n196, n197, n198, n199,
         n200, n201, n202, n203, n204, n205, n206, n207, n208, n209, n210,
         n211, n212, n213, n214, n215, n216, n217, n218, n219, n220, n221,
         n222, n223, n224, n225, n226, n227, n228, n229, n230, n231, n288,
         n289, n290, n291, n292, n293, n294, n295, n296, n297, n298, n299,
         n300, n301, n302, n303, n304, n305, n418, n419, n420, n421, n422,
         n423, n424, n425, n426, n427, n428, n429, n430, n431, n432, n433,
         n434, n435, n436, n437, n438, n439, n440, n441, n442, n443, n444,
         n445, n446, n447, n448, n449, n450, n451, n452, n453, n454, n455,
         n456, n457, n458, n459, n460, n461, n462, n463, n464, n465, n466,
         n467, n468, n469, n470, n471, n472, n473, n474, n475, n476, n477,
         n478, n479, n480, n481, n482, n483, n484, n485, n486, n487, n488,
         n489;

  AND2X2 U2 ( .A(wa[2]), .B(regwrite), .Y(n218) );
  OAI21X1 U81 ( .A(ra2[0]), .B(n79), .C(n80), .Y(rd2[7]) );
  OAI21X1 U82 ( .A(n81), .B(n82), .C(n293), .Y(n80) );
  OAI22X1 U83 ( .A(n436), .B(n457), .C(n304), .D(n441), .Y(n82) );
  OAI22X1 U84 ( .A(n84), .B(n481), .C(n305), .D(n465), .Y(n81) );
  AOI21X1 U85 ( .A(RAM_39_), .B(n86), .C(n87), .Y(n79) );
  OAI22X1 U86 ( .A(n304), .B(n449), .C(n305), .D(n473), .Y(n87) );
  OAI21X1 U87 ( .A(n293), .B(n88), .C(n89), .Y(rd2[6]) );
  OAI21X1 U88 ( .A(n90), .B(n91), .C(n293), .Y(n89) );
  OAI22X1 U89 ( .A(n436), .B(n458), .C(n304), .D(n442), .Y(n91) );
  OAI22X1 U90 ( .A(n84), .B(n482), .C(n305), .D(n466), .Y(n90) );
  AOI21X1 U91 ( .A(RAM_38_), .B(n86), .C(n92), .Y(n88) );
  OAI22X1 U92 ( .A(n304), .B(n450), .C(n305), .D(n474), .Y(n92) );
  OAI21X1 U93 ( .A(ra2[0]), .B(n93), .C(n94), .Y(rd2[5]) );
  OAI21X1 U94 ( .A(n95), .B(n96), .C(n293), .Y(n94) );
  OAI22X1 U95 ( .A(n436), .B(n459), .C(n304), .D(n443), .Y(n96) );
  OAI22X1 U96 ( .A(n84), .B(n483), .C(n305), .D(n467), .Y(n95) );
  AOI21X1 U97 ( .A(RAM_37_), .B(n86), .C(n97), .Y(n93) );
  OAI22X1 U98 ( .A(n304), .B(n451), .C(n305), .D(n475), .Y(n97) );
  OAI21X1 U99 ( .A(n293), .B(n122), .C(n123), .Y(rd2[4]) );
  OAI21X1 U100 ( .A(n124), .B(n125), .C(n293), .Y(n123) );
  OAI22X1 U101 ( .A(n436), .B(n460), .C(n304), .D(n444), .Y(n125) );
  OAI22X1 U102 ( .A(n84), .B(n484), .C(n305), .D(n468), .Y(n124) );
  AOI21X1 U103 ( .A(RAM_36_), .B(n86), .C(n126), .Y(n122) );
  OAI22X1 U104 ( .A(n304), .B(n452), .C(n305), .D(n476), .Y(n126) );
  OAI21X1 U105 ( .A(ra2[0]), .B(n127), .C(n128), .Y(rd2[3]) );
  OAI21X1 U106 ( .A(n129), .B(n154), .C(ra2[0]), .Y(n128) );
  OAI22X1 U107 ( .A(n436), .B(n461), .C(n304), .D(n445), .Y(n154) );
  OAI22X1 U108 ( .A(n84), .B(n485), .C(n305), .D(n469), .Y(n129) );
  AOI21X1 U109 ( .A(RAM_35_), .B(n86), .C(n155), .Y(n127) );
  OAI22X1 U110 ( .A(n304), .B(n453), .C(n305), .D(n477), .Y(n155) );
  OAI21X1 U111 ( .A(n293), .B(n156), .C(n157), .Y(rd2[2]) );
  OAI21X1 U112 ( .A(n158), .B(n159), .C(n293), .Y(n157) );
  OAI22X1 U113 ( .A(n436), .B(n462), .C(n304), .D(n446), .Y(n159) );
  OAI22X1 U114 ( .A(n84), .B(n486), .C(n305), .D(n470), .Y(n158) );
  AOI21X1 U115 ( .A(RAM_34_), .B(n86), .C(n160), .Y(n156) );
  OAI22X1 U116 ( .A(n304), .B(n454), .C(n305), .D(n478), .Y(n160) );
  OAI21X1 U117 ( .A(ra2[0]), .B(n161), .C(n162), .Y(rd2[1]) );
  OAI21X1 U118 ( .A(n163), .B(n164), .C(ra2[0]), .Y(n162) );
  OAI22X1 U119 ( .A(n436), .B(n463), .C(n304), .D(n447), .Y(n164) );
  OAI22X1 U120 ( .A(n84), .B(n487), .C(n305), .D(n471), .Y(n163) );
  AOI21X1 U121 ( .A(RAM_33_), .B(n86), .C(n165), .Y(n161) );
  OAI22X1 U122 ( .A(n304), .B(n455), .C(n305), .D(n479), .Y(n165) );
  OAI21X1 U123 ( .A(n293), .B(n166), .C(n167), .Y(rd2[0]) );
  OAI21X1 U124 ( .A(n168), .B(n169), .C(n293), .Y(n167) );
  OAI22X1 U125 ( .A(n436), .B(n464), .C(n304), .D(n448), .Y(n169) );
  OAI22X1 U126 ( .A(n84), .B(n488), .C(n305), .D(n472), .Y(n168) );
  OR2X1 U127 ( .A(ra2[2]), .B(ra2[1]), .Y(n84) );
  AOI21X1 U128 ( .A(RAM_32_), .B(n86), .C(n170), .Y(n166) );
  OAI22X1 U129 ( .A(n304), .B(n456), .C(n305), .D(n480), .Y(n170) );
  NOR2X1 U132 ( .A(n437), .B(ra2[1]), .Y(n86) );
  OAI21X1 U133 ( .A(ra1[0]), .B(n171), .C(n172), .Y(rd1[7]) );
  OAI21X1 U134 ( .A(n173), .B(n174), .C(n295), .Y(n172) );
  OAI22X1 U135 ( .A(n457), .B(n434), .C(n441), .D(n302), .Y(n174) );
  OAI22X1 U136 ( .A(n481), .B(n176), .C(n465), .D(n303), .Y(n173) );
  AOI21X1 U137 ( .A(n178), .B(RAM_39_), .C(n179), .Y(n171) );
  OAI22X1 U138 ( .A(n449), .B(n302), .C(n473), .D(n303), .Y(n179) );
  OAI21X1 U139 ( .A(n295), .B(n180), .C(n181), .Y(rd1[6]) );
  OAI21X1 U140 ( .A(n182), .B(n183), .C(n295), .Y(n181) );
  OAI22X1 U141 ( .A(n458), .B(n434), .C(n442), .D(n302), .Y(n183) );
  OAI22X1 U142 ( .A(n482), .B(n176), .C(n466), .D(n303), .Y(n182) );
  AOI21X1 U143 ( .A(n178), .B(RAM_38_), .C(n184), .Y(n180) );
  OAI22X1 U144 ( .A(n450), .B(n302), .C(n474), .D(n303), .Y(n184) );
  OAI21X1 U145 ( .A(ra1[0]), .B(n185), .C(n186), .Y(rd1[5]) );
  OAI21X1 U146 ( .A(n187), .B(n188), .C(n295), .Y(n186) );
  OAI22X1 U147 ( .A(n459), .B(n434), .C(n443), .D(n302), .Y(n188) );
  OAI22X1 U148 ( .A(n483), .B(n176), .C(n467), .D(n303), .Y(n187) );
  AOI21X1 U149 ( .A(n178), .B(RAM_37_), .C(n189), .Y(n185) );
  OAI22X1 U150 ( .A(n451), .B(n302), .C(n475), .D(n303), .Y(n189) );
  OAI21X1 U151 ( .A(n295), .B(n190), .C(n191), .Y(rd1[4]) );
  OAI21X1 U152 ( .A(n192), .B(n193), .C(n295), .Y(n191) );
  OAI22X1 U153 ( .A(n460), .B(n434), .C(n444), .D(n302), .Y(n193) );
  OAI22X1 U154 ( .A(n484), .B(n176), .C(n468), .D(n303), .Y(n192) );
  AOI21X1 U155 ( .A(n178), .B(RAM_36_), .C(n194), .Y(n190) );
  OAI22X1 U156 ( .A(n452), .B(n302), .C(n476), .D(n303), .Y(n194) );
  OAI21X1 U157 ( .A(ra1[0]), .B(n195), .C(n196), .Y(rd1[3]) );
  OAI21X1 U158 ( .A(n197), .B(n198), .C(ra1[0]), .Y(n196) );
  OAI22X1 U159 ( .A(n461), .B(n434), .C(n445), .D(n302), .Y(n198) );
  OAI22X1 U160 ( .A(n485), .B(n176), .C(n469), .D(n303), .Y(n197) );
  AOI21X1 U161 ( .A(n178), .B(RAM_35_), .C(n199), .Y(n195) );
  OAI22X1 U162 ( .A(n453), .B(n302), .C(n477), .D(n303), .Y(n199) );
  OAI21X1 U163 ( .A(n295), .B(n200), .C(n201), .Y(rd1[2]) );
  OAI21X1 U164 ( .A(n202), .B(n203), .C(n295), .Y(n201) );
  OAI22X1 U165 ( .A(n462), .B(n434), .C(n446), .D(n302), .Y(n203) );
  OAI22X1 U166 ( .A(n486), .B(n176), .C(n470), .D(n303), .Y(n202) );
  AOI21X1 U167 ( .A(n178), .B(RAM_34_), .C(n204), .Y(n200) );
  OAI22X1 U168 ( .A(n454), .B(n302), .C(n478), .D(n303), .Y(n204) );
  OAI21X1 U169 ( .A(ra1[0]), .B(n205), .C(n206), .Y(rd1[1]) );
  OAI21X1 U170 ( .A(n207), .B(n208), .C(ra1[0]), .Y(n206) );
  OAI22X1 U171 ( .A(n463), .B(n434), .C(n447), .D(n302), .Y(n208) );
  OAI22X1 U172 ( .A(n487), .B(n176), .C(n471), .D(n303), .Y(n207) );
  AOI21X1 U173 ( .A(n178), .B(RAM_33_), .C(n209), .Y(n205) );
  OAI22X1 U174 ( .A(n455), .B(n302), .C(n479), .D(n303), .Y(n209) );
  OAI21X1 U175 ( .A(n295), .B(n210), .C(n211), .Y(rd1[0]) );
  OAI21X1 U176 ( .A(n212), .B(n213), .C(n295), .Y(n211) );
  OAI22X1 U177 ( .A(n464), .B(n434), .C(n448), .D(n302), .Y(n213) );
  OAI22X1 U178 ( .A(n488), .B(n176), .C(n472), .D(n303), .Y(n212) );
  OR2X1 U179 ( .A(ra1[2]), .B(ra1[1]), .Y(n176) );
  AOI21X1 U180 ( .A(n178), .B(RAM_32_), .C(n214), .Y(n210) );
  OAI22X1 U181 ( .A(n456), .B(n302), .C(n480), .D(n303), .Y(n214) );
  NOR2X1 U184 ( .A(n435), .B(ra1[1]), .Y(n178) );
  OAI22X1 U185 ( .A(n296), .B(n487), .C(n215), .D(n430), .Y(n99) );
  OAI22X1 U186 ( .A(n296), .B(n488), .C(n215), .D(n432), .Y(n98) );
  OAI22X1 U187 ( .A(n301), .B(n441), .C(n418), .D(n217), .Y(n153) );
  OAI22X1 U188 ( .A(n301), .B(n442), .C(n420), .D(n217), .Y(n152) );
  OAI22X1 U189 ( .A(n301), .B(n443), .C(n422), .D(n217), .Y(n151) );
  OAI22X1 U190 ( .A(n301), .B(n444), .C(n424), .D(n217), .Y(n150) );
  OAI22X1 U191 ( .A(n301), .B(n445), .C(n426), .D(n217), .Y(n149) );
  OAI22X1 U192 ( .A(n301), .B(n446), .C(n428), .D(n217), .Y(n148) );
  OAI22X1 U193 ( .A(n301), .B(n447), .C(n430), .D(n217), .Y(n147) );
  OAI22X1 U194 ( .A(n301), .B(n448), .C(n432), .D(n217), .Y(n146) );
  NAND3X1 U195 ( .A(n218), .B(wa[0]), .C(wa[1]), .Y(n217) );
  OAI22X1 U196 ( .A(n300), .B(n449), .C(n418), .D(n219), .Y(n145) );
  OAI22X1 U197 ( .A(n300), .B(n450), .C(n420), .D(n219), .Y(n144) );
  OAI22X1 U198 ( .A(n300), .B(n451), .C(n422), .D(n219), .Y(n143) );
  OAI22X1 U199 ( .A(n300), .B(n452), .C(n424), .D(n219), .Y(n142) );
  OAI22X1 U200 ( .A(n300), .B(n453), .C(n426), .D(n219), .Y(n141) );
  OAI22X1 U201 ( .A(n300), .B(n454), .C(n428), .D(n219), .Y(n140) );
  OAI22X1 U202 ( .A(n300), .B(n455), .C(n430), .D(n219), .Y(n139) );
  OAI22X1 U203 ( .A(n300), .B(n456), .C(n432), .D(n219), .Y(n138) );
  NAND3X1 U204 ( .A(n218), .B(n440), .C(wa[1]), .Y(n219) );
  OAI22X1 U205 ( .A(n299), .B(n457), .C(n418), .D(n220), .Y(n137) );
  OAI22X1 U206 ( .A(n299), .B(n458), .C(n420), .D(n220), .Y(n136) );
  OAI22X1 U207 ( .A(n299), .B(n459), .C(n422), .D(n220), .Y(n135) );
  OAI22X1 U208 ( .A(n299), .B(n460), .C(n424), .D(n220), .Y(n134) );
  OAI22X1 U209 ( .A(n299), .B(n461), .C(n426), .D(n220), .Y(n133) );
  OAI22X1 U210 ( .A(n299), .B(n462), .C(n428), .D(n220), .Y(n132) );
  OAI22X1 U211 ( .A(n299), .B(n463), .C(n430), .D(n220), .Y(n131) );
  OAI22X1 U212 ( .A(n299), .B(n464), .C(n432), .D(n220), .Y(n130) );
  NAND3X1 U213 ( .A(wa[0]), .B(n439), .C(n218), .Y(n220) );
  AOI22X1 U214 ( .A(n222), .B(RAM_39_), .C(wd[7]), .D(n298), .Y(n221) );
  AOI22X1 U215 ( .A(n222), .B(RAM_38_), .C(wd[6]), .D(n298), .Y(n223) );
  AOI22X1 U216 ( .A(n222), .B(RAM_37_), .C(wd[5]), .D(n298), .Y(n224) );
  AOI22X1 U217 ( .A(n222), .B(RAM_36_), .C(wd[4]), .D(n298), .Y(n225) );
  AOI22X1 U218 ( .A(n222), .B(RAM_35_), .C(wd[3]), .D(n298), .Y(n226) );
  AOI22X1 U219 ( .A(n222), .B(RAM_34_), .C(wd[2]), .D(n298), .Y(n227) );
  AOI22X1 U220 ( .A(n222), .B(RAM_33_), .C(wd[1]), .D(n298), .Y(n228) );
  AOI22X1 U221 ( .A(n222), .B(RAM_32_), .C(wd[0]), .D(n298), .Y(n229) );
  NAND3X1 U222 ( .A(n440), .B(n439), .C(n218), .Y(n222) );
  OAI22X1 U223 ( .A(n297), .B(n465), .C(n418), .D(n230), .Y(n121) );
  OAI22X1 U224 ( .A(n297), .B(n466), .C(n420), .D(n230), .Y(n120) );
  OAI22X1 U225 ( .A(n297), .B(n467), .C(n422), .D(n230), .Y(n119) );
  OAI22X1 U226 ( .A(n297), .B(n468), .C(n424), .D(n230), .Y(n118) );
  OAI22X1 U227 ( .A(n297), .B(n469), .C(n426), .D(n230), .Y(n117) );
  OAI22X1 U228 ( .A(n297), .B(n470), .C(n428), .D(n230), .Y(n116) );
  OAI22X1 U229 ( .A(n297), .B(n471), .C(n430), .D(n230), .Y(n115) );
  OAI22X1 U230 ( .A(n297), .B(n472), .C(n432), .D(n230), .Y(n114) );
  NAND3X1 U231 ( .A(wa[0]), .B(n216), .C(wa[1]), .Y(n230) );
  OAI22X1 U232 ( .A(n438), .B(n473), .C(n418), .D(n231), .Y(n113) );
  OAI22X1 U233 ( .A(n438), .B(n474), .C(n420), .D(n231), .Y(n112) );
  OAI22X1 U234 ( .A(n438), .B(n475), .C(n422), .D(n231), .Y(n111) );
  OAI22X1 U235 ( .A(n438), .B(n476), .C(n424), .D(n231), .Y(n110) );
  OAI22X1 U236 ( .A(n438), .B(n477), .C(n426), .D(n231), .Y(n109) );
  OAI22X1 U237 ( .A(n438), .B(n478), .C(n428), .D(n231), .Y(n108) );
  OAI22X1 U238 ( .A(n438), .B(n479), .C(n430), .D(n231), .Y(n107) );
  OAI22X1 U239 ( .A(n438), .B(n480), .C(n432), .D(n231), .Y(n106) );
  NAND3X1 U240 ( .A(n216), .B(n440), .C(wa[1]), .Y(n231) );
  OAI22X1 U241 ( .A(n296), .B(n481), .C(n215), .D(n418), .Y(n105) );
  OAI22X1 U242 ( .A(n296), .B(n482), .C(n215), .D(n420), .Y(n104) );
  OAI22X1 U243 ( .A(n296), .B(n483), .C(n215), .D(n422), .Y(n103) );
  OAI22X1 U244 ( .A(n296), .B(n484), .C(n215), .D(n424), .Y(n102) );
  OAI22X1 U245 ( .A(n296), .B(n485), .C(n215), .D(n426), .Y(n101) );
  OAI22X1 U246 ( .A(n296), .B(n486), .C(n215), .D(n428), .Y(n100) );
  NAND3X1 U247 ( .A(n216), .B(n439), .C(wa[0]), .Y(n215) );
  NOR2X1 U248 ( .A(n489), .B(wa[2]), .Y(n216) );
  DFFPOSX1_SCAN RAM_reg_7__7_ ( .D(n153), .TI(RAM_62_), .TE(test_se), .CLK(clk), .Q(test_so) );
  DFFPOSX1_SCAN RAM_reg_7__6_ ( .D(n152), .TI(RAM_61_), .TE(test_se), .CLK(clk), .Q(RAM_62_) );
  DFFPOSX1_SCAN RAM_reg_7__5_ ( .D(n151), .TI(RAM_60_), .TE(test_se), .CLK(clk), .Q(RAM_61_) );
  DFFPOSX1_SCAN RAM_reg_7__4_ ( .D(n150), .TI(RAM_59_), .TE(test_se), .CLK(clk), .Q(RAM_60_) );
  DFFPOSX1_SCAN RAM_reg_7__3_ ( .D(n149), .TI(RAM_58_), .TE(test_se), .CLK(clk), .Q(RAM_59_) );
  DFFPOSX1_SCAN RAM_reg_7__2_ ( .D(n148), .TI(RAM_57_), .TE(test_se), .CLK(clk), .Q(RAM_58_) );
  DFFPOSX1_SCAN RAM_reg_7__1_ ( .D(n147), .TI(RAM_56_), .TE(test_se), .CLK(clk), .Q(RAM_57_) );
  DFFPOSX1_SCAN RAM_reg_7__0_ ( .D(n146), .TI(RAM_55_), .TE(test_se), .CLK(clk), .Q(RAM_56_) );
  DFFPOSX1_SCAN RAM_reg_6__7_ ( .D(n145), .TI(RAM_54_), .TE(test_se), .CLK(clk), .Q(RAM_55_) );
  DFFPOSX1_SCAN RAM_reg_6__6_ ( .D(n144), .TI(RAM_53_), .TE(test_se), .CLK(clk), .Q(RAM_54_) );
  DFFPOSX1_SCAN RAM_reg_6__5_ ( .D(n143), .TI(RAM_52_), .TE(test_se), .CLK(clk), .Q(RAM_53_) );
  DFFPOSX1_SCAN RAM_reg_6__4_ ( .D(n142), .TI(RAM_51_), .TE(test_se), .CLK(clk), .Q(RAM_52_) );
  DFFPOSX1_SCAN RAM_reg_6__3_ ( .D(n141), .TI(RAM_50_), .TE(test_se), .CLK(clk), .Q(RAM_51_) );
  DFFPOSX1_SCAN RAM_reg_6__2_ ( .D(n140), .TI(RAM_49_), .TE(test_se), .CLK(clk), .Q(RAM_50_) );
  DFFPOSX1_SCAN RAM_reg_6__1_ ( .D(n139), .TI(RAM_48_), .TE(test_se), .CLK(clk), .Q(RAM_49_) );
  DFFPOSX1_SCAN RAM_reg_6__0_ ( .D(n138), .TI(RAM_47_), .TE(test_se), .CLK(clk), .Q(RAM_48_) );
  DFFPOSX1_SCAN RAM_reg_5__7_ ( .D(n137), .TI(RAM_46_), .TE(test_se), .CLK(clk), .Q(RAM_47_) );
  DFFPOSX1_SCAN RAM_reg_5__6_ ( .D(n136), .TI(RAM_45_), .TE(test_se), .CLK(clk), .Q(RAM_46_) );
  DFFPOSX1_SCAN RAM_reg_5__5_ ( .D(n135), .TI(RAM_44_), .TE(test_se), .CLK(clk), .Q(RAM_45_) );
  DFFPOSX1_SCAN RAM_reg_5__4_ ( .D(n134), .TI(RAM_43_), .TE(test_se), .CLK(clk), .Q(RAM_44_) );
  DFFPOSX1_SCAN RAM_reg_5__3_ ( .D(n133), .TI(RAM_42_), .TE(test_se), .CLK(clk), .Q(RAM_43_) );
  DFFPOSX1_SCAN RAM_reg_5__2_ ( .D(n132), .TI(RAM_41_), .TE(test_se), .CLK(clk), .Q(RAM_42_) );
  DFFPOSX1_SCAN RAM_reg_5__1_ ( .D(n131), .TI(RAM_40_), .TE(test_se), .CLK(clk), .Q(RAM_41_) );
  DFFPOSX1_SCAN RAM_reg_5__0_ ( .D(n130), .TI(RAM_39_), .TE(test_se), .CLK(clk), .Q(RAM_40_) );
  DFFPOSX1_SCAN RAM_reg_4__7_ ( .D(n419), .TI(RAM_38_), .TE(test_se), .CLK(clk), .Q(RAM_39_) );
  DFFPOSX1_SCAN RAM_reg_4__6_ ( .D(n421), .TI(RAM_37_), .TE(test_se), .CLK(clk), .Q(RAM_38_) );
  DFFPOSX1_SCAN RAM_reg_4__5_ ( .D(n423), .TI(RAM_36_), .TE(test_se), .CLK(clk), .Q(RAM_37_) );
  DFFPOSX1_SCAN RAM_reg_4__4_ ( .D(n425), .TI(RAM_35_), .TE(test_se), .CLK(clk), .Q(RAM_36_) );
  DFFPOSX1_SCAN RAM_reg_4__3_ ( .D(n427), .TI(RAM_34_), .TE(test_se), .CLK(clk), .Q(RAM_35_) );
  DFFPOSX1_SCAN RAM_reg_4__2_ ( .D(n429), .TI(RAM_33_), .TE(test_se), .CLK(clk), .Q(RAM_34_) );
  DFFPOSX1_SCAN RAM_reg_4__1_ ( .D(n431), .TI(RAM_32_), .TE(test_se), .CLK(clk), .Q(RAM_33_) );
  DFFPOSX1_SCAN RAM_reg_4__0_ ( .D(n433), .TI(RAM_31_), .TE(test_se), .CLK(clk), .Q(RAM_32_) );
  DFFPOSX1_SCAN RAM_reg_3__7_ ( .D(n121), .TI(RAM_30_), .TE(test_se), .CLK(clk), .Q(RAM_31_) );
  DFFPOSX1_SCAN RAM_reg_3__6_ ( .D(n120), .TI(RAM_29_), .TE(test_se), .CLK(clk), .Q(RAM_30_) );
  DFFPOSX1_SCAN RAM_reg_3__5_ ( .D(n119), .TI(RAM_28_), .TE(test_se), .CLK(clk), .Q(RAM_29_) );
  DFFPOSX1_SCAN RAM_reg_3__4_ ( .D(n118), .TI(RAM_27_), .TE(test_se), .CLK(clk), .Q(RAM_28_) );
  DFFPOSX1_SCAN RAM_reg_3__3_ ( .D(n117), .TI(RAM_26_), .TE(test_se), .CLK(clk), .Q(RAM_27_) );
  DFFPOSX1_SCAN RAM_reg_3__2_ ( .D(n116), .TI(RAM_25_), .TE(test_se), .CLK(clk), .Q(RAM_26_) );
  DFFPOSX1_SCAN RAM_reg_3__1_ ( .D(n115), .TI(RAM_24_), .TE(test_se), .CLK(clk), .Q(RAM_25_) );
  DFFPOSX1_SCAN RAM_reg_3__0_ ( .D(n114), .TI(RAM_23_), .TE(test_se), .CLK(clk), .Q(RAM_24_) );
  DFFPOSX1_SCAN RAM_reg_2__7_ ( .D(n113), .TI(RAM_22_), .TE(test_se), .CLK(clk), .Q(RAM_23_) );
  DFFPOSX1_SCAN RAM_reg_2__6_ ( .D(n112), .TI(RAM_21_), .TE(test_se), .CLK(clk), .Q(RAM_22_) );
  DFFPOSX1_SCAN RAM_reg_2__5_ ( .D(n111), .TI(RAM_20_), .TE(test_se), .CLK(clk), .Q(RAM_21_) );
  DFFPOSX1_SCAN RAM_reg_2__4_ ( .D(n110), .TI(RAM_19_), .TE(test_se), .CLK(clk), .Q(RAM_20_) );
  DFFPOSX1_SCAN RAM_reg_2__3_ ( .D(n109), .TI(RAM_18_), .TE(test_se), .CLK(clk), .Q(RAM_19_) );
  DFFPOSX1_SCAN RAM_reg_2__2_ ( .D(n108), .TI(RAM_17_), .TE(test_se), .CLK(clk), .Q(RAM_18_) );
  DFFPOSX1_SCAN RAM_reg_2__1_ ( .D(n107), .TI(RAM_16_), .TE(test_se), .CLK(clk), .Q(RAM_17_) );
  DFFPOSX1_SCAN RAM_reg_2__0_ ( .D(n106), .TI(RAM_15_), .TE(test_se), .CLK(clk), .Q(RAM_16_) );
  DFFPOSX1_SCAN RAM_reg_1__7_ ( .D(n105), .TI(RAM_14_), .TE(test_se), .CLK(clk), .Q(RAM_15_) );
  DFFPOSX1_SCAN RAM_reg_1__6_ ( .D(n104), .TI(RAM_13_), .TE(test_se), .CLK(clk), .Q(RAM_14_) );
  DFFPOSX1_SCAN RAM_reg_1__5_ ( .D(n103), .TI(RAM_12_), .TE(test_se), .CLK(clk), .Q(RAM_13_) );
  DFFPOSX1_SCAN RAM_reg_1__4_ ( .D(n102), .TI(RAM_11_), .TE(test_se), .CLK(clk), .Q(RAM_12_) );
  DFFPOSX1_SCAN RAM_reg_1__3_ ( .D(n101), .TI(RAM_10_), .TE(test_se), .CLK(clk), .Q(RAM_11_) );
  DFFPOSX1_SCAN RAM_reg_1__2_ ( .D(n100), .TI(RAM_9_), .TE(test_se), .CLK(clk),
        .Q(RAM_10_) );
  DFFPOSX1_SCAN RAM_reg_1__1_ ( .D(n99), .TI(RAM_8_), .TE(test_se), .CLK(clk),
        .Q(RAM_9_) );
  DFFPOSX1_SCAN RAM_reg_1__0_ ( .D(n98), .TI(test_si), .TE(test_se), .CLK(clk),
        .Q(RAM_8_) );
  INVX2 U3 ( .A(n222), .Y(n298) );
  INVX2 U4 ( .A(n219), .Y(n300) );
  INVX2 U5 ( .A(n230), .Y(n297) );
  INVX2 U6 ( .A(n215), .Y(n296) );
  INVX2 U7 ( .A(n220), .Y(n299) );
  INVX2 U8 ( .A(n217), .Y(n301) );
  INVX2 U9 ( .A(n290), .Y(n303) );
  INVX2 U10 ( .A(n291), .Y(n302) );
  INVX2 U11 ( .A(n288), .Y(n305) );
  INVX2 U12 ( .A(n289), .Y(n304) );
  INVX2 U13 ( .A(n292), .Y(n293) );
  INVX2 U14 ( .A(n294), .Y(n295) );
  AND2X2 U15 ( .A(ra2[1]), .B(n437), .Y(n288) );
  INVX2 U16 ( .A(ra2[0]), .Y(n292) );
  AND2X2 U17 ( .A(ra2[2]), .B(ra2[1]), .Y(n289) );
  AND2X2 U18 ( .A(ra1[1]), .B(n435), .Y(n290) );
  AND2X2 U19 ( .A(ra1[2]), .B(ra1[1]), .Y(n291) );
  INVX2 U20 ( .A(ra1[0]), .Y(n294) );
  INVX2 U297 ( .A(wd[7]), .Y(n418) );
  INVX2 U298 ( .A(n221), .Y(n419) );
  INVX2 U299 ( .A(wd[6]), .Y(n420) );
  INVX2 U300 ( .A(n223), .Y(n421) );
  INVX2 U301 ( .A(wd[5]), .Y(n422) );
  INVX2 U302 ( .A(n224), .Y(n423) );
  INVX2 U303 ( .A(wd[4]), .Y(n424) );
  INVX2 U304 ( .A(n225), .Y(n425) );
  INVX2 U305 ( .A(wd[3]), .Y(n426) );
  INVX2 U306 ( .A(n226), .Y(n427) );
  INVX2 U307 ( .A(wd[2]), .Y(n428) );
  INVX2 U308 ( .A(n227), .Y(n429) );
  INVX2 U309 ( .A(wd[1]), .Y(n430) );
  INVX2 U310 ( .A(n228), .Y(n431) );
  INVX2 U311 ( .A(wd[0]), .Y(n432) );
  INVX2 U312 ( .A(n229), .Y(n433) );
  INVX2 U313 ( .A(n178), .Y(n434) );
  INVX2 U314 ( .A(ra1[2]), .Y(n435) );
  INVX2 U315 ( .A(n86), .Y(n436) );
  INVX2 U316 ( .A(ra2[2]), .Y(n437) );
  INVX2 U317 ( .A(n231), .Y(n438) );
  INVX2 U318 ( .A(wa[1]), .Y(n439) );
  INVX2 U319 ( .A(wa[0]), .Y(n440) );
  INVX2 U320 ( .A(test_so), .Y(n441) );
  INVX2 U321 ( .A(RAM_62_), .Y(n442) );
  INVX2 U322 ( .A(RAM_61_), .Y(n443) );
  INVX2 U323 ( .A(RAM_60_), .Y(n444) );
  INVX2 U324 ( .A(RAM_59_), .Y(n445) );
  INVX2 U325 ( .A(RAM_58_), .Y(n446) );
  INVX2 U326 ( .A(RAM_57_), .Y(n447) );
  INVX2 U327 ( .A(RAM_56_), .Y(n448) );
  INVX2 U328 ( .A(RAM_55_), .Y(n449) );
  INVX2 U329 ( .A(RAM_54_), .Y(n450) );
  INVX2 U330 ( .A(RAM_53_), .Y(n451) );
  INVX2 U331 ( .A(RAM_52_), .Y(n452) );
  INVX2 U332 ( .A(RAM_51_), .Y(n453) );
  INVX2 U333 ( .A(RAM_50_), .Y(n454) );
  INVX2 U334 ( .A(RAM_49_), .Y(n455) );
  INVX2 U335 ( .A(RAM_48_), .Y(n456) );
  INVX2 U336 ( .A(RAM_47_), .Y(n457) );
  INVX2 U337 ( .A(RAM_46_), .Y(n458) );
  INVX2 U338 ( .A(RAM_45_), .Y(n459) );
  INVX2 U339 ( .A(RAM_44_), .Y(n460) );
  INVX2 U340 ( .A(RAM_43_), .Y(n461) );
  INVX2 U341 ( .A(RAM_42_), .Y(n462) );
  INVX2 U342 ( .A(RAM_41_), .Y(n463) );
  INVX2 U343 ( .A(RAM_40_), .Y(n464) );
  INVX2 U344 ( .A(RAM_31_), .Y(n465) );
  INVX2 U345 ( .A(RAM_30_), .Y(n466) );
  INVX2 U346 ( .A(RAM_29_), .Y(n467) );
  INVX2 U347 ( .A(RAM_28_), .Y(n468) );
  INVX2 U348 ( .A(RAM_27_), .Y(n469) );
  INVX2 U349 ( .A(RAM_26_), .Y(n470) );
  INVX2 U350 ( .A(RAM_25_), .Y(n471) );
  INVX2 U351 ( .A(RAM_24_), .Y(n472) );
  INVX2 U352 ( .A(RAM_23_), .Y(n473) );
  INVX2 U353 ( .A(RAM_22_), .Y(n474) );
  INVX2 U354 ( .A(RAM_21_), .Y(n475) );
  INVX2 U355 ( .A(RAM_20_), .Y(n476) );
  INVX2 U356 ( .A(RAM_19_), .Y(n477) );
  INVX2 U357 ( .A(RAM_18_), .Y(n478) );
  INVX2 U358 ( .A(RAM_17_), .Y(n479) );
  INVX2 U359 ( .A(RAM_16_), .Y(n480) );
  INVX2 U360 ( .A(RAM_15_), .Y(n481) );
  INVX2 U361 ( .A(RAM_14_), .Y(n482) );
  INVX2 U362 ( .A(RAM_13_), .Y(n483) );
  INVX2 U363 ( .A(RAM_12_), .Y(n484) );
  INVX2 U364 ( .A(RAM_11_), .Y(n485) );
  INVX2 U365 ( .A(RAM_10_), .Y(n486) );
  INVX2 U366 ( .A(RAM_9_), .Y(n487) );
  INVX2 U367 ( .A(RAM_8_), .Y(n488) );
  INVX2 U368 ( .A(regwrite), .Y(n489) );
endmodule


module dff_WIDTH8_test_3 ( clk, d, q, test_si, test_se );
  input [7:0] d;
  output [7:0] q;
  input clk, test_si, test_se;


  DFFPOSX1_SCAN q_reg_7_ ( .D(d[7]), .TI(q[6]), .TE(test_se), .CLK(clk), .Q(
        q[7]) );
  DFFPOSX1_SCAN q_reg_6_ ( .D(d[6]), .TI(q[5]), .TE(test_se), .CLK(clk), .Q(
        q[6]) );
  DFFPOSX1_SCAN q_reg_5_ ( .D(d[5]), .TI(q[4]), .TE(test_se), .CLK(clk), .Q(
        q[5]) );
  DFFPOSX1_SCAN q_reg_4_ ( .D(d[4]), .TI(q[3]), .TE(test_se), .CLK(clk), .Q(
        q[4]) );
  DFFPOSX1_SCAN q_reg_3_ ( .D(d[3]), .TI(q[2]), .TE(test_se), .CLK(clk), .Q(
        q[3]) );
  DFFPOSX1_SCAN q_reg_2_ ( .D(d[2]), .TI(q[1]), .TE(test_se), .CLK(clk), .Q(
        q[2]) );
  DFFPOSX1_SCAN q_reg_1_ ( .D(d[1]), .TI(q[0]), .TE(test_se), .CLK(clk), .Q(
        q[1]) );
  DFFPOSX1_SCAN q_reg_0_ ( .D(d[0]), .TI(test_si), .TE(test_se), .CLK(clk),
        .Q(q[0]) );
endmodule


module datapath_WIDTH8_REGBITS3_test_1 ( clk, reset, const_gnd, memdata,
        alusrca, memtoreg, iord, pcen, regwrite, regdst, pcsource, alusrcb,
        irwrite, alucont, zero, instr, adr, writedata, test_si, test_se );
  input [7:0] memdata;
  input [1:0] pcsource;
  input [1:0] alusrcb;
  input [3:0] irwrite;
  input [2:0] alucont;
  output [31:0] instr;
  output [7:0] adr;
  output [7:0] writedata;
  input clk, reset, const_gnd, alusrca, memtoreg, iord, pcen, regwrite, regdst,
         test_si, test_se;
  output zero;
  wire   n3, n4, n5, n7;
  wire   [2:0] wa;
  wire   [7:0] nextpc;
  wire   [7:0] pc;
  wire   [7:0] md;
  wire   [7:0] rd1;
  wire   [7:0] a;
  wire   [7:0] rd2;
  wire   [7:0] aluresult;
  wire   [7:0] aluout;
  wire   [7:0] src1;
  wire   [7:0] src2;
  wire   [7:0] wd;

  mux2_WIDTH3 regmux ( .d0(instr[18:16]), .d1(instr[13:11]), .s(regdst), .y(wa) );
  dffen_WIDTH8_test_0 ir0 ( .clk(clk), .en(irwrite[3]), .d(memdata), .q(
        instr[7:0]), .test_si(a[7]), .test_se(test_se) );
    dffen_WIDTH8_test_1 ir1 ( .clk(clk), .en(irwrite[2]), .d(memdata), .q(
        instr[15:8]), .test_si(instr[7]), .test_se(test_se) );
    dffen_WIDTH8_test_2 ir2 ( .clk(clk), .en(irwrite[1]), .d(memdata), .q(
        instr[23:16]), .test_si(instr[15]), .test_se(test_se) );
    dffen_WIDTH8_test_3 ir3 ( .clk(clk), .en(irwrite[0]), .d(memdata), .q(
        instr[31:24]), .test_si(instr[23]), .test_se(test_se) );
    dffenr_WIDTH8_test_1 pcreg ( .clk(clk), .reset(reset), .en(pcen), .d(nextpc),
        .q(pc), .test_si(md[7]), .test_se(test_se) );
    dff_WIDTH8_test_1 mdr ( .clk(clk), .d(memdata), .q(md), .test_si(instr[31]),
        .test_se(test_se) );
    dff_WIDTH8_test_0 areg ( .clk(clk), .d(rd1), .q(a), .test_si(test_si),
        .test_se(test_se) );
    dff_WIDTH8_test_3 wrd ( .clk(clk), .d(rd2), .q(writedata), .test_si(n7),
        .test_se(test_se) );
    dff_WIDTH8_test_2 res ( .clk(clk), .d(aluresult), .q(aluout), .test_si(pc[7]), .test_se(test_se) );
    mux2_WIDTH8_2 adrmux ( .d0(pc), .d1(aluout), .s(iord), .y(adr) );
    mux2_WIDTH8_1 src1mux ( .d0(pc), .d1(a), .s(alusrca), .y(src1) );
    mux4_WIDTH8_1 src2mux ( .d0(writedata), .d1({n5, n4, n5, n4, n5, n4, n5, n3}), .d2(instr[7:0]),
  .d3({instr[5:0], n4, n5}), .s(alusrcb), .y(src2) );
    mux4_WIDTH8_0 pcmux ( .d0(aluresult), .d1(aluout), .d2({instr[5:0], n5, n4}),
        .d3({n4, n5, n4, n5, n4, n5, n4, n5}), .s(pcsource), .y(nextpc) );
    mux2_WIDTH8_0 wdmux ( .d0(aluout), .d1(md), .s(memtoreg), .y(wd) );
    regfile_WIDTH8_REGBITS3_test_1 rf ( .clk(clk), .regwrite(regwrite), .ra1(
        instr[23:21]), .ra2(instr[18:16]), .wa(wa), .wd(wd), .rd1(rd1), .rd2(
        rd2), .test_si(aluout[7]), .test_so(n7), .test_se(test_se) );
    alu_WIDTH8 alunit ( .a(src1), .b(src2), .alucont(alucont), .result(aluresult) );
    zerodetect_WIDTH8 zd ( .a(aluresult), .y(zero) );
    INVX2 U1 ( .A(n3), .Y(n4) );
    INVX2 U2 ( .A(n3), .Y(n5) );
    INVX2 U3 ( .A(const_gnd), .Y(n3) );
endmodule


module controller_test_1 ( alusrca, alusrcb, aluop, pcen, iord, irwrite,
        memread, memwrite, memtoreg, pcsource, regwrite, regdst, op, clk,
        reset, zero, test_si, test_so, test_se );
    output [1:0] alusrcb;
    output [1:0] aluop;
    output [3:0] irwrite;
    output [1:0] pcsource;
    input [5:0] op;
    input clk, reset, zero, test_si, test_se;
    output alusrca, pcen, iord, memread, memwrite, memtoreg, regwrite, regdst,
           test_so;
    wire   state_2_, state_1_, state_0_, N45, n34, n35, n36, n37, n38, n39, n40,
           n41, n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53, n54,
           n55, n56, n57, n58, n59, n60, n61, n62, n63, n64, n65, n66, n67, n68,
           n69, n70, n71, n72, n73, n74, n75, n76, n77, n78, n79, n88, n89, n90,
           n91, n92, n93, n94, n95, n96, n97, n98, n99, n100, n103, n104, n105,
           n107, n108, n109, n110, n111, n112, n113;

    INVX2 U3 ( .A(n39), .Y(pcsource[1]) );
    INVX2 U4 ( .A(n36), .Y(memtoreg) );
    INVX2 U5 ( .A(n53), .Y(irwrite[1]) );
    AND2X2 U6 ( .A(n40), .B(n107), .Y(memwrite) );
    INVX2 U9 ( .A(n73), .Y(aluop[0]) );
    OAI21X1 U37 ( .A(n34), .B(n35), .C(n36), .Y(regwrite) );
    AOI21X1 U38 ( .A(n37), .B(n35), .C(n34), .Y(regdst) );
    NAND2X1 U39 ( .A(n38), .B(n97), .Y(pcen) );
    AOI21X1 U40 ( .A(zero), .B(pcsource[0]), .C(pcsource[1]), .Y(n38) );
    NAND3X1 U41 ( .A(n40), .B(state_0_), .C(state_2_), .Y(n39) );
    OAI21X1 U42 ( .A(n42), .B(n43), .C(n113), .Y(n41) );
    OAI21X1 U43 ( .A(n96), .B(n44), .C(n90), .Y(n43) );
    AOI21X1 U44 ( .A(op[1]), .B(op[2]), .C(n46), .Y(n45) );
    NAND2X1 U45 ( .A(n47), .B(n100), .Y(n44) );
    NAND2X1 U46 ( .A(n48), .B(n99), .Y(n42) );
    OAI21X1 U47 ( .A(n50), .B(n51), .C(n113), .Y(n49) );
    OAI21X1 U48 ( .A(n94), .B(n108), .C(n52), .Y(n51) );
    NAND3X1 U49 ( .A(n109), .B(n99), .C(n53), .Y(n50) );
    OAI21X1 U50 ( .A(n55), .B(n56), .C(n113), .Y(n54) );
    OAI21X1 U51 ( .A(n57), .B(n46), .C(n58), .Y(n56) );
    NAND3X1 U52 ( .A(n47), .B(n59), .C(n60), .Y(n58) );
    NAND3X1 U53 ( .A(n91), .B(n60), .C(n61), .Y(n46) );
    NOR2X1 U54 ( .A(op[4]), .B(op[0]), .Y(n61) );
    XNOR2X1 U55 ( .A(op[1]), .B(op[2]), .Y(n57) );
    NAND3X1 U56 ( .A(n62), .B(n63), .C(n52), .Y(n55) );
    NAND3X1 U57 ( .A(n100), .B(n64), .C(n47), .Y(n52) );
    OAI21X1 U58 ( .A(op[3]), .B(n93), .C(n65), .Y(n64) );
    NAND2X1 U59 ( .A(n40), .B(n98), .Y(n36) );
    NAND2X1 U60 ( .A(n97), .B(n48), .Y(memread) );
    OAI21X1 U61 ( .A(n34), .B(n37), .C(n66), .Y(iord) );
    NOR2X1 U62 ( .A(memwrite), .B(n105), .Y(n66) );
    NAND3X1 U63 ( .A(state_2_), .B(state_0_), .C(n67), .Y(n48) );
    NAND2X1 U64 ( .A(n108), .B(n68), .Y(alusrcb[1]) );
    NAND2X1 U65 ( .A(n97), .B(n108), .Y(alusrcb[0]) );
    NAND3X1 U66 ( .A(n53), .B(n109), .C(n70), .Y(n69) );
    NOR2X1 U67 ( .A(irwrite[3]), .B(irwrite[2]), .Y(n70) );
    NAND2X1 U68 ( .A(n67), .B(n107), .Y(n62) );
    NAND2X1 U69 ( .A(state_0_), .B(n111), .Y(n35) );
    NAND3X1 U70 ( .A(n112), .B(n110), .C(n103), .Y(n63) );
    NOR2X1 U71 ( .A(n71), .B(state_2_), .Y(irwrite[0]) );
```

```
NAND2X1 U72 ( .A(n67), .B(n98), .Y(n53) );
NAND3X1 U73 ( .A(n73), .B(n99), .C(n68), .Y(alusrca) );
NOR2X1 U74 ( .A(n72), .B(n34), .Y(aluop[1]) );
NAND2X1 U75 ( .A(test_so), .B(state_1_), .Y(n34) );
NAND2X1 U76 ( .A(n103), .B(n40), .Y(n73) );
NOR2X1 U77 ( .A(n110), .B(state_1_), .Y(n40) );
NAND3X1 U78 ( .A(n74), .B(n72), .C(n75), .Y(N45) );
AOI21X1 U79 ( .A(n104), .B(n112), .C(n76), .Y(n75) );
OAI21X1 U80 ( .A(n77), .B(n108), .C(n78), .Y(n76) );
OAI21X1 U81 ( .A(n94), .B(n65), .C(n100), .Y(n78) );
NAND2X1 U82 ( .A(n67), .B(n103), .Y(n68) );
NAND2X1 U83 ( .A(state_2_), .B(n104), .Y(n37) );
NOR2X1 U84 ( .A(n112), .B(test_so), .Y(n67) );
NAND2X1 U85 ( .A(op[3]), .B(n93), .Y(n65) );
NOR2X1 U86 ( .A(n95), .B(op[2]), .Y(n47) );
NOR2X1 U87 ( .A(n71), .B(n111), .Y(n60) );
NAND3X1 U88 ( .A(n112), .B(n110), .C(state_0_), .Y(n71) );
AOI21X1 U89 ( .A(op[2]), .B(n59), .C(n95), .Y(n77) );
NOR3X1 U90 ( .A(op[1]), .B(op[4]), .C(op[0]), .Y(n79) );
NAND2X1 U91 ( .A(n96), .B(n93), .Y(n59) );
NAND2X1 U92 ( .A(n104), .B(n111), .Y(n72) );
NOR2X1 U93 ( .A(test_so), .B(reset), .Y(n74) );
DFFPOSX1_SCAN state_reg_0_ ( .D(N45), .TI(test_si), .TE(test_se), .CLK(clk),
      .Q(state_0_) );
DFFPOSX1_SCAN state_reg_3_ ( .D(n89), .TI(state_2_), .TE(test_se), .CLK(clk),
      .Q(test_so) );
DFFPOSX1_SCAN state_reg_2_ ( .D(n88), .TI(state_1_), .TE(test_se), .CLK(clk),
      .Q(state_2_) );
DFFPOSX1_SCAN state_reg_1_ ( .D(n92), .TI(state_0_), .TE(test_se), .CLK(clk),
      .Q(state_1_) );
INVX2 U16 ( .A(n54), .Y(n88) );
INVX2 U17 ( .A(n41), .Y(n89) );
INVX2 U18 ( .A(n45), .Y(n90) );
INVX2 U19 ( .A(n59), .Y(n91) );
INVX2 U20 ( .A(n49), .Y(n92) );
INVX2 U21 ( .A(op[5]), .Y(n93) );
INVX2 U22 ( .A(n47), .Y(n94) );
INVX2 U23 ( .A(n79), .Y(n95) );
INVX2 U24 ( .A(op[3]), .Y(n96) );
INVX2 U25 ( .A(n69), .Y(n97) );
INVX2 U26 ( .A(n72), .Y(n98) );
INVX2 U27 ( .A(aluop[1]), .Y(n99) );
INVX2 U28 ( .A(n68), .Y(n100) );
INVX2 U29 ( .A(n73), .Y(pcsource[0]) );
INVX2 U30 ( .A(n63), .Y(irwrite[3]) );
INVX2 U31 ( .A(n37), .Y(n103) );
```

```
  INVX2 U32 ( .A(state_0_), .Y(n104) );
  INVX2 U33 ( .A(n48), .Y(n105) );
  INVX2 U34 ( .A(n62), .Y(irwrite[2]) );
  INVX2 U35 ( .A(n35), .Y(n107) );
  INVX2 U36 ( .A(n60), .Y(n108) );
  INVX2 U94 ( .A(irwrite[0]), .Y(n109) );
  INVX2 U95 ( .A(test_so), .Y(n110) );
  INVX2 U96 ( .A(state_2_), .Y(n111) );
  INVX2 U97 ( .A(state_1_), .Y(n112) );
  INVX2 U98 ( .A(reset), .Y(n113) );
endmodule


module mips ( clk, reset, const_gnd, memdata, memread, memwrite, adr,
        writedata, test_si, test_so, test_se );
  input [7:0] memdata;
  output [7:0] adr;
  output [7:0] writedata;
  input clk, reset, const_gnd, test_si, test_se;
  output memread, memwrite, test_so;
  wire   zero, alusrca, memtoreg, iord, pcen, regwrite, regdst, n19, n20, n21,
         n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32, n33, n34, n35,
         n36, n38, n40, SYNOPSYS_UNCONNECTED_1, SYNOPSYS_UNCONNECTED_2,
         SYNOPSYS_UNCONNECTED_3, SYNOPSYS_UNCONNECTED_4,
         SYNOPSYS_UNCONNECTED_5, SYNOPSYS_UNCONNECTED_6,
         SYNOPSYS_UNCONNECTED_7, SYNOPSYS_UNCONNECTED_8,
         SYNOPSYS_UNCONNECTED_9, SYNOPSYS_UNCONNECTED_10,
         SYNOPSYS_UNCONNECTED_11, SYNOPSYS_UNCONNECTED_12,
         SYNOPSYS_UNCONNECTED_13, SYNOPSYS_UNCONNECTED_14,
         SYNOPSYS_UNCONNECTED_15, SYNOPSYS_UNCONNECTED_16,
         SYNOPSYS_UNCONNECTED_17, SYNOPSYS_UNCONNECTED_18,
         SYNOPSYS_UNCONNECTED_19, SYNOPSYS_UNCONNECTED_20;
  wire   [31:0] instr;
  wire   [1:0] pcsource;
  wire   [1:0] alusrcb;
  wire   [1:0] aluop;
  wire   [3:0] irwrite;
  wire   [2:0] alucont;

  controller_test_1 cont ( .alusrca(alusrca), .alusrcb(alusrcb), .aluop(aluop),
        .pcen(pcen), .iord(iord), .irwrite(irwrite), .memread(memread),
        .memwrite(memwrite), .memtoreg(memtoreg), .pcsource(pcsource),
        .regwrite(regwrite), .regdst(regdst), .op(instr[31:26]), .clk(clk),
        .reset(n20), .zero(zero), .test_si(test_si), .test_so(n38), .test_se(
        test_se) );
  alucontrol ac ( .alucont(alucont), .aluop(aluop), .funct(instr[5:0]) );
  datapath_WIDTH8_REGBITS3_test_1 dp ( .clk(clk), .reset(n20), .const_gnd(
        const_gnd), .memdata({n36, n34, n32, n30, n28, n26, n24, n22}),
        .alusrca(alusrca), .memtoreg(memtoreg), .iord(iord), .pcen(pcen),
        .regwrite(regwrite), .regdst(regdst), .pcsource(pcsource), .alusrcb(
        alusrcb), .irwrite(irwrite), .alucont(alucont), .zero(zero), .instr({
        instr[31:26], SYNOPSYS_UNCONNECTED_1, SYNOPSYS_UNCONNECTED_2,
        SYNOPSYS_UNCONNECTED_3, SYNOPSYS_UNCONNECTED_4, SYNOPSYS_UNCONNECTED_5,
        SYNOPSYS_UNCONNECTED_6, SYNOPSYS_UNCONNECTED_7, SYNOPSYS_UNCONNECTED_8,
        SYNOPSYS_UNCONNECTED_9, SYNOPSYS_UNCONNECTED_10,
        SYNOPSYS_UNCONNECTED_11, SYNOPSYS_UNCONNECTED_12,
        SYNOPSYS_UNCONNECTED_13, SYNOPSYS_UNCONNECTED_14,
        SYNOPSYS_UNCONNECTED_15, SYNOPSYS_UNCONNECTED_16,
        SYNOPSYS_UNCONNECTED_17, SYNOPSYS_UNCONNECTED_18,
        SYNOPSYS_UNCONNECTED_19, SYNOPSYS_UNCONNECTED_20, instr[5:0]}), .adr(
        adr), .writedata(writedata), .test_si(n38), .test_se(test_se) );
  INVX2 U1 ( .A(reset), .Y(n19) );
  INVX2 U2 ( .A(n19), .Y(n20) );
  INVX2 U3 ( .A(memdata[0]), .Y(n21) );
  INVX2 U4 ( .A(n21), .Y(n22) );
  INVX2 U5 ( .A(memdata[1]), .Y(n23) );
  INVX2 U6 ( .A(n23), .Y(n24) );
  INVX2 U7 ( .A(memdata[2]), .Y(n25) );
  INVX2 U8 ( .A(n25), .Y(n26) );
  INVX2 U9 ( .A(memdata[3]), .Y(n27) );
  INVX2 U10 ( .A(n27), .Y(n28) );
  INVX2 U11 ( .A(memdata[4]), .Y(n29) );
  INVX2 U12 ( .A(n29), .Y(n30) );
  INVX2 U13 ( .A(memdata[5]), .Y(n31) );
  INVX2 U14 ( .A(n31), .Y(n32) );
  INVX2 U15 ( .A(memdata[6]), .Y(n33) );
  INVX2 U16 ( .A(n33), .Y(n34) );
  INVX2 U17 ( .A(memdata[7]), .Y(n35) );
  INVX2 U18 ( .A(n35), .Y(n36) );
  INVX8 U19 ( .A(writedata[7]), .Y(n40) );
  INVX8 U20 ( .A(n40), .Y(test_so) );
endmodule
```

Appendix D

tmax_atpg.tcl

```
##################################################################
#### TetraMax Script for ECE 128
#### Performs ATPG Pattern Generation for Synopsys Generic files
#### author: tjf
#### update: wgibb, spring 2010
#### note: this script will only run in TMAX TCL mode
#### start tmax like this:   tmax -tcl
##################################################################


############################################################
#### local variables, designer must change these values ####
############################################################

set top_module mips
set synthesized_files [list ./src/mips_scan.v]
set cell_lib ./src/osu05_stdcells.v
set scan_lib ./src/osu_scan.v
set stil_file [list ./src/mips_scan.spf]
```

```
################################################
#### read in standard cells and user's design ###
################################################

# remove any other designs from design compiler's memory
read_netlist -delete

# read in standard cell library
read_netlist $cell_lib -library

# read in scan cell library
read_netlist $scan_lib -library

# read in user's synthesized verilog code
read_netlist $synthesized_files


################################################
#### BUILD and DRC test model
################################################
```

```
run_build_model $top_module
# ignoring warnings like N20 or B10

# Set STIL file from DFT Compiler
set_drc $stil_file

# run check to see if synthesized code violates any testing rules
run_drc

#################################################
#### Generate ATPG (patterns)- full sequential
#################################################

# capture all faults, 9 capture cycles
set_atpg -capture_cycles 9 -full_seq_atpg
remove_faults -all
add_faults -all

# run atpg in full sequential mode
run_atpg full_sequential_only

# write out patterns (overwrite old files)
write_patterns ./src/${top_module}_tb_patterns.v -replace -internal -format verilog_single_file -parallel 0
```

```
################################################
#### Output reports
################################################

report_patterns -all >> ./reports/${top_module}.tmax.patterns
report_violations -all >> ./reports/${top_module}.tmax.violations
report_faults -summary -collapsed >> ./reports/${top_module}.tmax.coverage

################################################
#### Analyze Faults
################################################

# up to user to run these commands, they can inspect the faults and various reasons for them:
#analyze_faults -class an
#analyze_faults -class an -verbose -max 3
#analyze_faults in_a_reg_reg/p_dregscan0/q -stuck 1
```

Appendix E

maxtb.dat

```
//                      Copyright (c) 2007 - 2018 Synopsys, Inc.
//    This software and the associated documentation are proprietary to Synopsys,
//  Inc. This software may only be used in accordance with the terms and conditions
//  of a written license agreement with Synopsys, Inc. All other use, reproduction,
//              or distribution of this software is strictly prohibited.
//
// MAX TB Test Data File, generated by MAX TB Version O-2018.06-SP1
// Sat Apr 30 14:11:05 2022
// Module under test: mips
// Generated from original STIL file : mips.stil
// STIL file version: "1.0"

// TPC
100010000
// WFT _multiclock_capture_WFT_
000_0000000000000011
// Condition
000010_0000000000000100
0000000000000_0000000000000_100000
1111111111111111111_XXXXXXXXXXXXXXXXXX_100010
```

```
// Macro test_setup

000000_0000000000001110

//SetForceSI 0

00_0000000000001011

//Pattern #0

0000000000000000000000000000_0000000000000010

// Proc load_unload

000001_0000000000001111

1110010110001001011010100011110101011001001011101010001001101111010110011100011001110011100110101111001001010
10011001001101001111101010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100100100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000100111010_100000

//SetForceSI 1

01_0000000000001011

//Pattern #1

0000000000000001

// Proc load_unload

000010_0000000000001111

1011001000010010110101000111101010110010010111010100010011011110100011011000110111100100001101011110010010 1
0011011100100000100100101_011111
```

```
1011111011000011011111110000100111110000000101010110110101101011001001010101001001000111101010100100111000001
11001011100001100100111 01_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111010100011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101000011110_100000

//Pattern #2

0000000000000001

// Proc load_unload

000010_0000000000001111

00000000100001100010101000100111100000000101010110110101101011001000011110000000011000010010100100111000001 1
100101110000110101011000 1_011111

01101000000000100000000111101100101011010101010001001101111110001001001100110000110101010101110101001101100000
001111010111111111111011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001010010001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001010101010_100000

//Pattern #3

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1011010100001000000001111011001010110101010100010011011111000100010111101000011010101010111010100110110000
0011110101111101010001 0001_011111

1010001101111010101001010100101110110110001010010000010101111110000011101110111110101010100111111001101000
01010000000100011111000011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100101101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101101001100_100000

//Pattern #4

0000000000000001

// Proc load_unload

000010_0000000000001111

0110110011110101010010101001011101101100010100100000101011111100000000000000000001001011000111111001101000
0101000000010001111111000001_011111

0100000111000101111111010000011001010000111100110000111010110110110110100111110101001111110011101001000110
1010010110000001010110 0001_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0100101101111_100000

// Proc allclock_launch_capture
```

```
000001_0000000000010011

Z000000000000_1011011111110_100000

//Pattern #5

0000000000000001

// Proc load_unload

000010_0000000000001111

1011011011000101111111010000011001010000111100110000111010110110011111100000000011111011111110111001000110
10100101100000010100000001_011111

0100011110001000100001011000000000111011111010101100101011100000101100111001001001100110001100110111010011
1000010010011011110000101 1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001110111111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011001100110_100000

//Pattern #6

0000000000000001

// Proc load_unload

000010_0000000000001111

1100000100010001000010110000000000111011111010101100101011100000110110011000000000011001101100110111010011
10000100100110100010001000 1_011111

1110001011010011001000110101001101100110001001111100000110010001011111100100000010001001111111000010011110
00111010000010101000000101_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100000101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011010110110_100000

//Pattern #7

0000000000000001

// Proc load_unload

000010_0000000000001111

10100110101001100100011010100110110011001111110010000011001000100100011000000000011010111111100001001111000
1110100000101100000110001_011111

00000100101110011001010110111111001110001101010001100011101000110000111001010101100110010100001111101110011
1011111111000000000010101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101000000111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010001010_100000

//Pattern #8

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
01110001011100110010101100011100011100011010100011000111010001101011010010101011100010101000011111011100111
0111111100000001010110001_011111

00011100100101011110110101011110011101101100100001000111111111011100010111011100111011110000001001000111100
0111100000010011111100011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010011000101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110100100100_100000

//Pattern #9

0000000000000001

// Proc load_unload

000010_0000000000001111

00101011001010111101101010111100111011011001000010001111111110110000000000000000001001010000010010001111000
1111000000100111011010001_011111

11011011100111011001000011010010010010011010000011010110000110010101011100110001100010001111011101001011110
0101000110001011111110101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011111011001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000101010010_100000
```

```
//Pattern #10

0000000000000001

// Proc load_unload

000010_0000000000001111

0011101100111011001000011010010010010011010000011010110000110010010001110110001101010100111011101001011110
0101000110001010010011101_011111

0000010001111000000110111100000110110000110111101001100101010001111101001001011110001001011000111111110101
0100111111001100001111010_001011

// Proc multiclock_capture

000001_0000000000010010

Z000000000000_1110100111101_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1011100110100_100000

//Pattern #11

0000000000000001

// Proc load_unload

000010_0000000000001111

1000001111110000001101111000001101100001101111010011001010100011001100000000000001100111110001111111101010
1001101100111111100000001_011111

1000001001010110100001011000110101001111100011110000000010100100110100011110010000010011000111100110000001
00100100100010011001001101_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1010110100111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001011010000_100000

//Pattern #12

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001010110100001011000110101001111100011110101000110100100111001100110010000101101000111100110000000100
10010010001000000010110001_011111

111000000111010011110000101101100110011011000001110010100011101100011101101001011111101111111110001011111011
11101101001011000100010111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100110011000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011010001100_100000

//Pattern #13

0000000000000001

// Proc load_unload

000010_0000000000001111

011101001100110011110000101101100110011011000001110010100011101100011001000000000100010111111110001011111011
11101101001011100101000001_011111
```

```
11111101111110100101001000110001110101001001111011101101111010010011110101011111111010011110001111100100101
1001111001011000111011100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110101010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010100001110_100000

//Pattern #14

0000000000000001

// Proc load_unload

000010_0000000000001111

110100101111101001010010001100011101010010011110111011011110100101011101000000000100001011100011111001001011
0011110010110101001000001_011111

00011000010100011010011100001100110101101110000110100110101100000111101010001101100111000111001000011100010
0011110101101101010101010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011110111010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100101000010_100000

//Pattern #15

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

110101100101000110100111000011001101011011100001101001101011000001100011100011010001010001110010000011100010
00111101011010000000000001_011111

001111100001011001011111111010010010001100101100111110011010111110101000100000000111101001001000111101011100
10100000000001110010110111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101001111000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001001100010_100000

//Pattern #16

0000000000000001

// Proc load_unload

000010_0000000000001111

000101101111001001011111111010010010001100101100111110011010111110001011100000000100110010001000111101011100
10100000000001010111110001_011111

110101111101001100100111111010100110000000010110000000100101011110011001000100011111100110001011101111011010
1110001010010001011100101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111000010001_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1000010001010_100000

//Pattern #17

0000000000000001

// Proc load_unload

000010_0000000000001111

010011111010011001001111110101001100000000101100000001001010111100000110001000111100010000010111011110110101
1100010100100101001101011_011111

010010101110111111111111101011000010101001100010111110010001101100001110100001000110001001111100110111001101
1100111010111101111011011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110001011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100111011000_100000

//Pattern #18

0000000000000001

// Proc load_unload

000010_0000000000001111

100010111101111111111111010110000101010011000101111100100011011000010101000010001110111001111001101110011011
1001110101111100010110001_011111

01000110110110000110011011101011101011111000100100010011101100000100010010110000010001000100100101110010011
1101010000111110000111101_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011010110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010011010100_100000

//Pattern #19

0000000000000001

// Proc load_unload

000010_0000000000001111

01011111101100001000100111010111010111110001001000100111011000001110110100000000010110011001001011100100111
1010100001111101100000001_011111

101111100101011000000000111000010010000001010101101010000100100101001101011000101110000001000011000010001010
1011001111100011010010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110011111000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100001100110_100000

//Pattern #20

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
1001001001010110000000001110000100100000010101011010100001001001011000001000000000011000010000110111110010011000001111100010101100001_011111

0111101110110110110101110110100010010010111000000000000111010011001101110000010011001000010101101001110101111011000010011111011111101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110001111111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111110011000_100000

//Pattern #21

0000000000000001

// Proc load_unload

000010_0000000000001111

1101000101101101101011101101110000100101110000000000001110100110000001010000100111100111101011010011101011110110000100111110000000001_011111

000101101000110010111101111100010101110111010011111001101000110001010000011000101010011111011011100001101101001000111100101110110100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010111001100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000010011010_100000
```

```
//Pattern #22

0000000000000001

// Proc load_unload

000010_0000000000001111

1011110110001100101111011111000101011101111010011111001101000110000000001000000011100100011001000100001101101
1001010011101010111010010_011111

1111101000000110101000110111001100011100110001110001111001010101111011110101101000100101010101010011100110
1011011110110100101110110 1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011010101011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110001011110_100000

//Pattern #23

0000000000000001

// Proc load_unload

000010_0000000000001111

1000111000001101010001101110011000111001110111100011110010101011101010000000000001101000110101010011100110101
1011110110101000111000 01_011111

0001010010101100111000100100001001000111100111101100000110100111011110100100110100011111101100101000000011 0
0101010111000101000011001_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1011101101101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111111010100_100000

//Pattern #24

0000000000000001

// Proc load_unload

000010_0000000000001111

01001110010110011100010010000100100011110011110110000011010011101001100000000000010111110110010101011111100
1010101110001000000000001_011111

01000010001111010111001101001101101110100011001011011001010000100110010011111001101111010101111101100111010
0001000011000101100101110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110011100100_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0111100111011_100000

1111111111111111111_0000000010101111000_100010

//Pattern #25

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
0011110100111101011100110100110110111010001100101101100101000010001110110000000000111001010111110110011101O
0001000011000001100100001_011111

1101001101001100000110101100011001001110011001010110011101000100101001111101010000110011101011110011110010O
1011000000011011011101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110101000011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100101001010_100000

//Pattern #26

0000000000000001

// Proc load_unload

000010_0000000000001111

1000100110011000001101011000110010011100110010101100111010001001010011101010100010010100010111100111100100l
0110000000110110010101000_011111

0010110100100011000011111111111100100011111111111110100111100010100001001010001101110101010000001010010011O
0010101010010010011101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001001100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100111011010_100000
```

Jaret Williams
Nathan Pen

```
//Pattern #27

0000000000000001

// Proc load_unload

000010_0000000000001111

11000101010001100001111111111110010001111111111111010011110001010001100001000110110111001000000101001001100
01010101001001101001110 11_011111

01001001000000011100001110110110101010101010001011000110011010001101110000100101000110111010101000101111 0001
1000111001110001001000101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011110101101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110010100110_100000

//Pattern #28

0000000000000001

// Proc load_unload

000010_0000000000001111

10101010000000111000011101101101010101010100010110001100110100011110110100000000000010100110101000101111100011
0001110011100110110100001_011111

00000111010010000011000100110001011001010101110001101010111100111000000101111101001110101001011011001101010
0110000111111010000100 1000_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1010110100011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011011001000_100000

//Pattern #29

0000000000000001

// Proc load_unload

000010_0000000000001111

11010101100100000110001001100010110010101011100011010101111001111111101111111011100110111001101110011010100
1100001111101110010100010_011111

10001111111100101001011010100000001000000010101011010101110011100000110110000001100111000111010101101000010
1111010011010111011000010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110111000110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011110100000_100000

//Pattern #30

0000000000000001

// Proc load_unload

000010_0000000000001111

1010000001111001010010110101000000001000000010101011010101110011100000000100000001001011110010111100011101010
1111010011010000000000010_011111
```

```
00110010010110001000011001011101000100111111010101111110011011111010111010000001010010011010111101100010011
1001011111011011111110000_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110101110101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000001101010_100000

//Pattern #31

0000000000000001

// Proc load_unload

000010_0000000000001111

00100111101100010000110010111010001001111110101011111100110111110110011000000010101100000101111011000100111
0010111110110000011000001_011111

01100110110001000011100011100000110100011000010010111101110000010110011000001111000101011101100111010000100
1000001110011101011101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100010111111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111000001100_100000

//Pattern #32

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

100000101000100001110001110000011100110000001001011110111000001011101011000000000100000111011001110100001001
0000011100111110000010001_011111

10101110111101100000111100110010000010110111010011100110010111000010101110100010101000110110011110110011000
1011010001010001100010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001001000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010110011110_100000

//Pattern #33

0000000000000001

// Proc load_unload

000010_0000000000001111

00000000111101100000111100110010000010110111010011100110010111001010010000000000110011010110011110010000110
0110110001010000010110001_011111

10101010010100101111101101101100010011010111110000000011011110101010011000010001001110010001011110101011110
0101000100011111111111101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010111111011_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1000101011000_100000

//Pattern #34

0000000000000001

// Proc load_unload

000010_0000000000001111

1010010110100101111101101101100010011010111110000100110011110101011101100010001011010100001011110101011100
1010001000111000001100001_011111

1000000011010111110111110011101000011111111111100110101000001011010010010111110011110011110101011110100000
0
1110011001110110010001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100000010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010111110010_100000

//Pattern #35

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000010010010101111100111010000111111111111001101010000010110111110101111100101111101101010111101000000
1
1100110011101101111100001_011111

0100011001000011110000100110100000101000000000100110110111011101010110100000000001011100010000100111000100
011
1010001110001010110010101_001011
```

290

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000011111000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000111001000_100000

//Pattern #36

0000000000000001

// Proc load_unload

000010_0000000000001111

00101000010000111100001001101000001010000000010011011011101110100010100111000100100111000000010011100010011
1010001110001110000100001_011111

10001101100101101000111010010001000110011001101100000101100001010001001100011001000101000001001110011010000
1110001111110000111000100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110000001111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100000000000_100000

//Pattern #37

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
0011001100101101000111010010001000110011001101100000101100001010010011010011001000000000000100111001101000011100011111100000010100001_011111

0001111011110011001000110110100000010011010101101101111100010101001000110000001011011110000100100111000111111111010010000000110100101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100000011010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110010101000_100000

//Pattern #38

0000000000000001

// Proc load_unload

000010_0000000000001111

0001010111110011001000110110100000010011010101101101111100010101010000010000001010101001000100100111000111111111010010000101011100001_011111

0001100111101001000110101101001111110010011010111111001110100011010101011000100111001001100111010010011010111111100110001011000101 10_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001100100111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011001000010_100000
```

```
//Pattern #39

0000000000000001

// Proc load_unload

000010_0000000000001111

10100111110100100011010110100111111001001101011111100111010001100100011011000100000100110011101001001101011
11111001100011110011100001_011111

10100101100010110110001010010001000101000001000101001100100010001110101101111100100100001010001011000010111
10010010010001100001000110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101010111010_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0100010001000_100000

1111111111111111111_0101000001001100100_100010

//Pattern #40

0000000000000001

// Proc load_unload

000010_0000000000001111

01001100100010110110001010010001000101000001000101001100100010000000101001111100111010101010001011000010111
10010010010000011000101001_011111

01000011000000011111111101101010110010100110110000011001011110001001110010100100110010011010010010001011100
01111010001010001011010001_001011

// Proc allclock_launch
```

```
000001_0000000000010000

Z000000000000_1111011001010_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0010100000001_100000

1111111111111111111_0010010110001101100_100010

//Pattern #41

0000000000000001

// Proc load_unload

000010_0000000000001111

011011000000000111111111011010101100101001101100000110010111100011100111101001001001101110100100100010111000
0111101000101110010100001_011111

101001010110000011110100000010010011010010100000010110010011110110111010011111010111110111110001100010110111
1110010100000110001101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000111100101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001101011100_100000

//Pattern #42

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
1110100011000001111010000000100100110100101000000101100100111101100010000000000001101011011100011000101101111
11100101000000000000000001_011111

1000111110101000110101100110110001101110110011000100110011110101011000011001100011000101111000011100111101110
1010011100100000110011010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001001110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000110001100_100000

//Pattern #43

0000000000000001

// Proc load_unload

000010_0000000000001111

1010100010101000110101100110110001101110110011000100110011110101000000001000000001000110010001100110011110111
0100111001001101011000001_011111

0110101110101001100100011110000111101011000111010111111001101110000000111110100000111011111011111111010111000
001101111110011101011110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001100100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100011111000_100000
```

```
//Pattern #44

0000000000000001

// Proc load_unload

000010_0000000000001111

0010001101010011001000111100001111010110001110101111110011011100000000000000011111111100010111111101011100000
01101111110011100001100001_011111

11101111111011101001111011111011100110010001110011111001111101110001111011100000111000000110000101000111100
0000000111001111101111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010100111001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100101110010_100000

//Pattern #45

0000000000000001

// Proc load_unload

000010_0000000000001111

11011101110111010011110111110111001100100011100111110011111011100001000011000001011101001100001010001111000
0000001110011001100100001_011111

01011111110000111111000100101110111010011101111100011000001010101011000101000000110011101011011000000000010
0001001111100011011111110_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1001011000001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010011000000_100000

//Pattern #46

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000010000111111000100101110111010011101111100011000001010101001000001000000100011001011011000000000000100
0010011111000000000000001_011111

1011111111111000110000100010100010100100111000010110101000110100010001000110000000111101010010010111001011111
0010101011011001111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101100000101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111001011000_100000

//Pattern #47

0000000000000001

// Proc load_unload

000010_0000000000001111

1000010111100011000010001010001010010011100001011010100011010001000000000001000111010011001001011100101111 0
0101010110110000010000001_011111
```

```
10111111101110100000010010111101011001010110110011011110110010111100101000000000100111111001010010100010011
1100010000100010111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110100101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001001100000_100000

//Pattern #48

0000000000000001

// Proc load_unload

000010_0000000000001111

110010100111010000001001011110101100101011011001101111011001011101000000000000001001100100010100101000100111
1000100001000101111010001_011111

01111111110101010011110100111000111110000110100110000010101100101101110110000000010110100101011000111101100
0000110100001011111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101101101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100100010010_100000

//Pattern #49

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

11010011101010100111101001110001111100001101001100000101011001010000000010111011010001001010110001111011000
0001101000010110100110001_011111

0111111110111011111111111000011001000000000000101001001000011101101101001000000000100101000010011001101011000
1110000011110101111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000100101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101110000000_100000

//Pattern #50

0000000000000001

// Proc load_unload

000010_0000000000001111

11111110011101111111111100001100100000000000010100100100001110110110000000000000001000011100100110011010110001
11000001111011111111100001_011111

0001111110000000110011110010000010001001100001100111111011110000000100001000010000110101010100100011000001101
1000100000110111111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110101101001_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1101011100000_100000

//Pattern #51

0000000000000001

// Proc load_unload

000010_0000000000001111

00011001000000110011110010000010001001100001100111111011110000001100000000100001001110101001000110000011011
0001000011011001001100001_011111

1111111110010101001111000001100111100000101001011110110101111101001100011110001101010011101011010101010110111
0000001000000000000000110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000011110101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110101101000_100000

//Pattern #52

0000000000000001

// Proc load_unload

000010_0000000000001111

0011001100101010011110000011001111000001010010111101101011111010000000011100011010110101010110101010101101110
0000010000000001100110001_011111

111111011111001011111111100110000010000000011000110101000111101011001001011100100100010100001111000101111001
0101001011010111111101110_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001100010101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010111110010_100000

//Pattern #53

0000000000000001

// Proc load_unload

000010_0000000000001111

11111111111100101111111111001100000100000000110001101010001111010110000001011001001011111010011110001011110010
10100101101010101000010001_011111

11111011110110100100110000000100001111101000101100000111010010000010010101001000110110011001101011000000101
01110000101101111111011110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011011100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010111001010_100000

//Pattern #54

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
10010000101101001001100000000100001111101000101100000111010010000000001001001000110011101001101011000000101011100001011010111110100 01_011111

1111011110101111101000111100001110011000000110001010110001011101101111011111000000111110101101101000100110010000010111010111101111 10_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001101111101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001000001010_100000

//Pattern #55

0000000000000001

// Proc load_unload

000010_0000000000001111

0011000101011111010001111000011100110000001100010101100010111011000010001110000010000010011011010001001100100000101110100000000000 01_011111

0001101110111100100110010000000111111011110010110101010111111100110001001111100001010101100000000010001011000000011100011010110011 1110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110011100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011111001010_100000
```

```
//Pattern #56

0000000000000001

// Proc load_unload

000010_0000000000001111

00110010011110010011001000000111111011110010110101010111111110011001100001110000110011111000000001000101100000
0001110001101000000000001_011111

11000000010100101110110110110110011011101110000001100100110111111010011001000000010001001101100110011111001
1001111010100010111111100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110100110101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111011011010_100000

//Pattern #57

0000000000000001

// Proc load_unload

000010_0000000000001111

11011011101001011000100101101100110111011100000011001001101111110000000010000000110110111011001100111110011
0011110101000101111110001_011111

10111000110101100011111110011100110111111001000100001000001010001101110010111111101011101100010001000101001 1
0101000000000000000000110_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1100101100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011111000000_100000

//Pattern #58

0000000000000001

// Proc load_unload

000010_0000000000001111

1011111010101100011111110011100110111110010001000010000010100011100011111111110000111110001000100010100110
1010000000000000000000001_011111

0111000101110000011010110011010111001111101000110010000011010000011101010111001100100010100100001000001 0011
1001110011110000110110110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111000011111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011111100010_100000

//Pattern #59

0000000000000001

// Proc load_unload

000010_0000000000001111

1001111111110000011010110011010111001111101000110010000011010000011001000111100000011111100100001000001 00111
0011100111100000000000001_011111
```

```
10101010001010001010010011000110101111011011100111101111001011111000010001011000010100111010001110111010000
1101110101111101111100101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000000101010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001110101010_100000

//Pattern #60

0000000000000001

// Proc load_unload

000010_0000000000001111

11101111001010001010010011000110101111011011100111101111001011110111010101011000101011101010001110111010000
1101110101111110001101001_011111

01011011111100101111001101011111111011011101110000100010000000100110101101010000111111011010100001011000011 1
0110001001001001000110011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001000100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010000010010_100000

//Pattern #61

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

000010011110010111100110101111111101101110111000010001000000100111011000010000110100000110100001011000011101100010010010101110001001_011111

110011100010110010001101111111101011010010001001101101111111011000001010100010110001111110000001101001110001101101100111001100100010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101101111011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010110011000_100000

//Pattern #62

0000000000000001

// Proc load_unload

000010_0000000000001111

000110110101100100011011111111010110100100010011011011111110110001001110000101101100110100000011010011100011011011001110011011111010_011111

110101111000011101101110000111110001010100001011011110100100100100000111111101001100001110001101110001111000110010010010001000111010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011000100111_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1110101101010_100000

//Pattern #63

0000000000000001

// Proc load_unload

000010_0000000000001111

0000111000001110110111000011111000101010000101101111010010010010011111001110100110110101000110111000111100
0
1100100100100001010100001_011111

1011100001111101001101011100000000000011001110011100010111010001101011100000011111101110010001001000011101
10
1101111001010110000010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000110010011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111001110010_100000

//Pattern #64

0000000000000001

// Proc load_unload

000010_0000000000001111

1000000011111010011010111000000000000110011100111000101110100011001110110000111110111001100010010000111011
01
1011110010101000000001100_011111

1001100110100110111111111010110101110010010001001001101001011101011110110100000110001110101000011011001110
0
1101111011101000001010110_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100101101110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010001100010_100000

//Pattern #65

0000000000000001

// Proc load_unload

000010_0000000000001111

010001001010011011111111110101101011100100100010010011010010111010000000100000001001100010011000110110011100
11011110111011010110100l0_011111

000110011010011000001010110001110101011111101100010110101011100110110110000101010010010010011101101010000000
0110100101010001101000111_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0010011111001_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1011010010100_100000

//Pattern #66

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
0000000010100110000010101100011101010111111011000101101010111001001011100000000000100101110011101101010000000
01101001010101100011100001_011111

00110010011111110010100101000011011001001011011001100111011000010111110111000101110101111011101110101110001 0
1111100100001110111000111_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0011110100111_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1110010001000_100000

//Pattern #67

0000000000000001

// Proc load_unload

000010_0000000000001111

11001001011111110010100101000011011001001011011001100111011000010101111011000101110001001011101110101110001 0
1111100100001110011101000_011111

01100101000000000100101001100111100001011111010110111100111110111101110001101110010011011100001101000101011 0
1101101001110001010001001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101100000101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101100010000_100000
```

```
//Pattern #68

0000000000000001

// Proc load_unload

000010_0000000000001111

1001111000000000100101001100111100001011111010110111100111110111110110111101101110100011000001101010001101011011010011100000000000011_011111

0110110101101001100100100110001100100101110100000001010001010000111111011001000100111101011101010100101011111010110100000010001100001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101100101011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100000010000_100000

//Pattern #69

0000000000000001

// Proc load_unload

000010_0000000000001111

1100011011010011001001001100011001001011101000000010100010100001001000000001000000100000011101010100101010100000010100000001001011010_011111

0011100010000000100011101001010011000011110001110011110000011000111001100011101101011001101000001000110110101111111000011101000110011_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1011010000001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011111001000_100000

//Pattern #70

0000000000000001

// Proc load_unload

000010_0000000000001111

0011101000000010001110100101001100001111000111001111000001100011110011011110110110011111100000100011011010 1
11111100001110110001101111_011111

1011100001011101111001111111011101101110011111010110100101101010111111001011101011101111100000110101001001 0
1101011111011011001100110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001101101101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111100110000_100000

//Pattern #71

0000000000000001

// Proc load_unload

000010_0000000000001111

1101110010111011110011111110111011011100111110101101001011010101010111000111010101100111000001101010010010 1
1010111110110111011100001_011111
```

```
010011000000100111011010100001010011110011010000010100001101010111100001011010000111001011000001000101111110
1111111011001000010010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000011000111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010011100000_100000

//Pattern #72

0000000000000001

// Proc load_unload

000010_0000000000001111

000100110001001110110101000010100111100110100000101000011010101110011000110100000011100110000010001011111101
1111110110010101010110110_011111

000111111010100100111101010010001110010001101000010100101011011010110111011010100001101100100010010001101110
0111100110100001100011010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111011001011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000011000000_100000

//Pattern #73

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1001000101010010011110101001000111001000110100001010010101101101011101001101010000011000010001001000110111O
111100110100011001000000 1_011111
```

```
000001_0000000000010001

Z000000000000_1101011101010_100000

//Pattern #75

0000000000000001

// Proc load_unload

000010_0000000000001111

1110111011111000111011101100001101000100100001010010111000011100011110111000000010111010000010010010011001010
100011011111000011100110 1_011111

0111011111011111001010100101101011011011000110111001111110010100100000000110110111000111001010000100001101101
01111100010010000011 01010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100110110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010101010_100000

//Pattern #76

0000000000000001

// Proc load_unload

000010_0000000000001111

1011010110111111001010100101101011011011000110111001111110010100111111111101101111010100110100001000011011010
1111000100100000000001 10_011111

1101110111111011111101101001010011111110000111010010011101001011001101011001000001101111000010101111101100001
000000100001101110100010_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011010011111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011000010000_100000

//Pattern #77

0000000000000001

// Proc load_unload

000010_0000000000001111

11101101111101111110110100101001111111000011101001001110100101100010110000100000010000110001010111110110000
1000001000011111101110001_011111

01110010010101110000111001101011000011110011100001100100011101100101001110011011001000111010000011100000001
1101111110110100111010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010110000011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010111010_100000

//Pattern #78

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
0000000010101110000111001101011000011110011100001100100011101100111011010011011011101010010000011100000000011
10111111011010001110000001_011111

1110011011001011001010100010011001000000000001100001000111010000001001001100011001111101010001000010100101011
1000111111110001100101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000100110101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011111011000_100000

//Pattern #79

0000000000000001

// Proc load_unload

000010_0000000000001111

0100011110010110010101000100110010000000000011000010001110100000101100001001100111101111100100001010010101111
0001111111100010001111110_011111

1101110000001000100010000101101100000110010010011100010010001011000011000110010010111010010000000010001101010
0101101100001010010001011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010000000001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011101010000_100000
```

```
//Pattern #80

0000000000000001

// Proc load_unload

000010_0000000000001111

0110110000100010001000010110110000011001001001110001001000101100010100111001001001010111000000000100011010101
01101100001000011001000 1_011111

0000011010010100100001111100001011000000101111010111111000100011100100100111101111000010101010110001011011000
0111110000001100110000011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101001111011_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0000001101111_100000

111111111111111111_100100101010110000 0_100010

//Pattern #81

0000000000000001

// Proc load_unload

000010_0000000000001111

0000110100101001000011111000010110000001011110101111110001000111001001001111011110000101010101100010110110000
011110000000110011000001 11_011111

1101101110010001010111001110100000000111010110000100010100000000110011101100010110000110010010001101010011001
0011011001000101110000111_001011

// Proc allclock_launch
```

```
000001_0000000000010000

Z000000000000_1011010001000_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0001111011010_100000

1111111111111111111_0110100000000011011_100010

//Pattern #82

0000000000000001

// Proc load_unload

000010_0000000000001111

1011000010010001010111001110100000000111010110000100010100000000111111000100010110100010110010001101010011001
0011011001000100010101000_011111

101010111110010000011001001101110110100001100101011111100110011111111100010000011011101100011010010100100101000
1000001110110011001110010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001011110111_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0011101001011_100000

1111111111111111111_0110000000111010100_100010

//Pattern #83

0000000000000001

// Proc load_unload
```

```
000010_0000000000001111

01010111100100000110010011011101101000011001010111111001100111111111000100000110111011000110100101001010001
0000011101100110011100101_011111

10100010000000111001101110001100011111100000000000100001010111010111001001100101001110111011111101100011100 11
0111111001010001110101010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010000011001_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0110110000001_100000

1111111111111111111_1001010010001000100_100010

//Pattern #84

0000000000000001

// Proc load_unload

000010_0000000000001111

01000100000001110011011100011000111111000000000001000010101110101110010011001010011101110111111011000111001 10
1111110010100011101010100_011111

00100010011010001010010101111010101111010100100101110011001000111101101110101111000101010101010110101010010
0011001010100011001011111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000010111001_100000

// Proc allclock_capture
```

Jaret Williams
Nathan Pen

```
000010_0000000000010001

0000000000000_0111110110011_100000

1111111111111111111_1110110100001000100_100010

//Pattern #85

0000000000000001

// Proc load_unload

000010_0000000000001111

0100010011010001010010101111010101111010100100101110011001000111101101110101111000101010101010110101010100100
0110010101000110010111110_011111

1000010001100101110110000101010010010001001010110010111111000010110001110000001111011011011011011011111111100
0101101000001111011100111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110111001101_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0001010010001_100000

1111111111111111111_0111000100000100000_100010

//Pattern #86

0000000000000001

// Proc load_unload

000010_0000000000001111

0000100011001011101100001010100100100010010101100101111110000101100011100000011110110110110110110111111111000
1011010000011110111001110_011111
```

```
0011000000100100010000011010000001110100110100001101111011110000100010010001111010000000100000110001111110
0
1000001010111111001110110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001011111000_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0111000111101_100000

1111111111111111111_0111110010001001000_100010

//Pattern #87

0000000000000001

// Proc load_unload

000010_0000000000001111

0010010000100100010000011010000001110100110100001101111011110000001111100001111011111010100000110001111110
0
1000001010111000000000111_011111

1010110010101000010000011101110111001000011001010110101011010010101001110011011101101001010110010010111110
1
0111001101111100011111010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010000000011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010101100010_100000

//Pattern #88
```

```
0000000000000001

// Proc load_unload

000010_0000000000001111

01010000010100001000001110111011100100001100101011010101101001011110101001101110001101011011001001011111010
11100110111111010101010001_011111

11110011011110100101110110011100101111000010011011101001000000010100100000110001101110001100100111101001010
01101011001000101111110010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111010110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101011101000_100000

//Pattern #89

0000000000000001

// Proc load_unload

000010_0000000000001111

110100101111010010111011001110010111100001001101110100100000000101000011001100011101110101001001111010010100
1101011001000101110110001_011111

00110100101101111110010101001011101011011011100010001101110101111110101011100110011111011100101001010010111
0100111001110001000110101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000110000110_100000
```

```
// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110000100010_100000

//Pattern #90

0000000000000001

// Proc load_unload

000010_0000000000001111

1000110110110111111001010100101110101101101110001000110111010111111111101111111000100001001000010101001011
1010011100111010001101001 0_011111

0001110111110100010101011000100101000001001010001000110101000010110100011001001110100011001000011100101010
1000001000101101110000010 1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010110111010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011100000010_100000

//Pattern #91

0000000000000001

// Proc load_unload

000010_0000000000001111

1000110111110100010101011000100101000001001010001000110101000010111000000100100110000011100100001110010101
01000001000101101010101111 0_011111
```

```
100010100111000010010111111111101011101011010101111110001010110001101011001111111011101001110101000110101111
0100110101100010111000011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010011100000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101101100010_100000

//Pattern #92

0000000000000001

// Proc load_unload

000010_0000000000001111

111111010111000010010111111111101011101011010101111110001010110000111111001111110001101101110101000110101001
1100100110110010110000101_011111

100000011000101100000110010111010001100010010110000101010000000100000010011101011111100111000000110101000l111
0101001101101100000110110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101111101010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000111111010_100000

//Pattern #93

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

0000001010001011000001100101110100011000100101100001010100000001011101101111010111111110010000011010100011110
101001101101000101011000_011111

1001111101001000010111000110010011010011100011000000011111001110111100010001100001100101100100001011001010100
1001000000010000111000110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110001100010_100000

//Pattern #94

0000000000000001

// Proc load_unload

000010_0000000000001111

1100100110010000101110001100100110100111000110000000111110011101000111100011000000110001001000010110010101001
0010000000100001100011110_011111

1011010110000100011000000100011110111111110110101101111101011010001010010001011101010010010000000000000001101
1000000001010100011000010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111011000111_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1011000001000_100000

//Pattern #95

0000000000000001

// Proc load_unload

000010_0000000000001111

1011010000001000110000001000111101111111101101011011111010110100010101100010111010000011100000000000000001101
1000000001010000000000110_011111

10000001101000011101101101100111101110010111110110110111110011111011111110001111010111001001000100101110011
1111100011011111110111010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110000110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001010110000_100000

//Pattern #96

0000000000000001

// Proc load_unload

000010_0000000000001111

01110010010000111011011011001111011100101111101101101111100111111111101000011110011010100010001001011100111
1111000110111011011110110_011111

10101101100100111001111000010101100100111110010001111011001100100001100111101110001001101000001010001010111
1100010010010100100100101_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000100100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010001100_100000

//Pattern #97

0000000000000001

// Proc load_unload

000010_0000000000001111

0010101100100111001111000010101100100111110010001111011001100100011111110000000010001010000000101000101011111
10001001001010000000000001_011111

1100001110010110110100000001010100010000110100000101110111101110011101100100100101010010011101101011111100101
01010001100110100000001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110110011001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010100010_100000

//Pattern #98

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
101110010010110110100000010101000100001101000001011101111011100110000111001001010010101011011010111110010101
01000110011000101101101_011111

0100010101000000100101100100110101101111010001001111101110001101110100011111110110000110110100101011111000010
1111111001000010011011 01_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101110011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101111111100_100000

//Pattern #99

0000000000000001

// Proc load_unload

000010_0000000000001111

101111011000001001011001001101011011110100010011111011100011011111001111000000001111111010100101011111000010
111111001000000100110001_011111

000000110111001001011101000000001001001101011000011100000011111000000000101001110111111110111000101010100001011
0001010100001000100101 01_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000101001001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111111010110_100000
```

```
//Pattern #100

0000000000000001

// Proc load_unload

000010_0000000000001111

111110001110010010111010000000100100110101100001110000001111100000010110000000000101111110001010101000010110
0010101000010000000100001_011111

010000111100000101110111011101011100010001111100011010110100011100001001110110101010111100001101110101101110
1011110101010111100011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111111100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010001000100_100000

//Pattern #101

0000000000000001

// Proc load_unload

000010_0000000000001111

111010111000001011101110111010111000100011111000110101101000111000000000000000000000100010001101110101101110
1011110101010111010110001_011111

001011010010101011010111100101111001010100111101011011100101000000110010111100001010000111111000010101100000
0001010100101011010010101_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1100100011001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111000000000_100000

//Pattern #102

0000000000000001

// Proc load_unload

000010_0000000000001111

00101010010101011010111100101111001010100111101011011100101000001101101011100001000000111111000010101100000
0010101001010001011111011_011111

01111111100100101111000001110001001001010001110100010100100110111111001101101011100011010001110111001101001
0100110010100000101000101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111000111001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010001101100_100000

//Pattern #103

0000000000000001

// Proc load_unload

000010_0000000000001111

00101001001001011111000001110001001001010001110100010100100110111111111110000000010110001001110111001101001O
1001100101000010010100001_011111
```

```
11100010000100011110000011001101110000100100111000110010110011011001111010100011101000101101100100101101001
1000100100010001001111101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000011101101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110101011010_100000

//Pattern #104

0000000000000001

// Proc load_unload

000010_0000000000001111

01100101001000111100000110011011100001001001110001100101100110111100111101000111110101011011001001011010011
0001001000100011001011011_011111

101000010110110110011100000000010110010011101011101111111100011100111100000101011001000111101000000001101101
0010110011010100010101100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001101101101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011011001000_100000

//Pattern #105

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1010111011011011001110000000010110010011010001111111111100011100100110000101011010011011101000000000110110100
1011001101010000000000001_011111

0001111011100010010101110110001011110001010010001111100001010111000010110001111110101111001011110111101110010
111111010101100101101100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011100011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110110111000_100000

//Pattern #106

0000000000000001

// Proc load_unload

000010_0000000000001111

1100010001011110101011101100010111100010100100011111000010101110011110000111111110110101011110111101110010
1111101010111100010000001_011111

0100001001010101100010010111010011101111101001111100010110000100000010011101111001010011001001111011101101010
1110101111110111101110101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100100110111_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1110010111010_100000

//Pattern #107

0000000000000001

// Proc load_unload

000010_0000000000001111

0001001010101011000100100001001111011111010011111000101100001000010000001011110011101001010011110111011010101
0111010111110010011110001_011111

1111000101001010000101110101100101100011000001011010000000100111001011111111110000011001110111100110111010101
0110110101011101000101100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000111001001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100011101010_100000

//Pattern #108

0000000000000001

// Proc load_unload

000010_0000000000001111

1011001010010100001011100011001111000110000010110100000000100111011011010111110001011100010111100110111010101
1101101010111001011100001_011111

1011010001110010011100011001110011011110101000110100010100001101101001000111000011001110111001110001011110110
1000110101100110100110100_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110010010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010111000_100000

//Pattern #109

0000000000000001

// Proc load_unload

000010_0000000000001111

111001001110010011100011001110011011110101000110100010100001101101001001111000011110101011001110001011110110
00110101100100011011000 1_011111

00001011000100001000000110001110110010010011001100010110111110110111110111011000010000101010011000100000111
001001011101001000000101 0_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001101110001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111111100110_100000

//Pattern #110

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
1111011000100001000000110001110110010010011001100010110111110110101011100000000000011111101001100010000001110
0100100111111001011010001_011111

0010010111101101110111001010010111100011011110100111011011000011101100110101110010000100100101010010011001 0
1000000111100011111011010_001011

// Proc multiclock_capture

000001_0000000000010010

Z000000000000_1000011001001_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1101010111010_100000

//Pattern #111

0000000000000001

// Proc load_unload

000010_0000000000001111

1100011011011011101110010100101111000110111101001110110110000111101101111011011111101010001010100100110010 1
0000011101010111011010101_011111

0101100110011101100011000011111001100010101101011100110011000011101111010101110110100000101001111111110111
1000011110011010010011010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101111000101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111001001010_100000
```

```
//Pattern #112

0000000000000001

// Proc load_unload

000010_0000000000001111

0111110000111011000110000111110011000101011010111001100110000111101110011011100110010011010011111111101111
0000110010011001110110101_011111

0010011111000111010100010001110011001110101000010101010011011001101010101001101101101010110111010110111110
0000001010010100111110010_001011

// Proc multiclock_capture

000001_0000000000010010

Z000000000000_1110110111101_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1110110001100_100000

//Pattern #113

0000000000000001

// Proc load_unload

000010_0000000000001111

1000111010001110101000100011100110011101010000101010100110110011001101000000000001000110110111010110111111100
0000010001101101000100001_011111

0101101000010000000000111011101010011001010101111000000001000101001101000000011011001011001110111000100000100
1110000110000000010111010_001011

// Proc multiclock_capture

000001_0000000000010010
```

```
Z000000000000_1000000100101_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1010011001100_100000

//Pattern #114

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000001000000000011101110101001100101010111100000000100010100000111000000000100110010111011100010000101
1100010011001000000000001_011111

0111111111001010101101001111111110001101100101110010100101000100011001100001010100001111101010110001110110 0
1111001011010100101000010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001000000000_100000

//Pattern #115

0000000000000001

// Proc load_unload

000010_0000000000001111

0010111010010101011010011111111100011011001011100101001010001000001010110010101100000001001010110001110110 01
1110000000010100010000101_011111
```

```
1010000100111011111011011111111000001010110010011001100101001110110101000000000001110000001000011001110000000 1
0110000100000010000000100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011001011011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000111110010_100000

//Pattern #116

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000001110111101101111111100000101011001001100110010100111011001001001000001110111110000011001110000000 10
110000100000010110111 0001_011111

1001010111111000000111001111011101000000100011110000111110001001110011011011111000111100010000010010001001000
10000101010011100101 01010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101001000001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100011111100_100000

//Pattern #117

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

0111110011110000011100111101110100000100011110000111110001001110111011110000000011111000000001001000100100001
0000111111000000000000001_011111

0001000111101000101110100001111001110011111000111111110000101110101111100100011101100111101111010001001001 1
1011011000001110011010100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101101101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011010110010_100000

//Pattern #118

0000000000000001

// Proc load_unload

000010_0000000000001111

1110011111010001011101000011110011001111110001111111100001011101101100011000111001101011011110100010010011 1
0110110000011010111010001_011111

0110101010100001011010001110100001001011001000100000101010011101000110101111101011001001111000110000100101 1
0000001011001000111100100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101110111101_100000

// Proc allclock_capture
```

Jaret Williams
Nathan Pen

```
000001_0000000000010001

Z000000000000_1011111010110_100000

//Pattern #119

0000000000000001

// Proc load_unload

000010_0000000000001111

0001010101000010110100011101000010010110010001001001001100111010110010100000000000101111111000110000100101100
00000101100100000000000001_011111

0111110011100001101001110100101001011001111001101111111101101100011000001100000000110111111110111110011111000
1000000110110101110000011_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0110111100111_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1011000111010_100000

//Pattern #120

0000000000000001

// Proc load_unload

000010_0000000000001111

1110000111100001101001110100101001011001111001101111111101101100110000011100000111100011111110111110011111110
00110001101111110000010100_011111

0101000110011100100101001010010011001100001100110111010111000000110100001111100101100110111010000111001100001
11110011111111110111001 1_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011001000100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111001111100_100000

//Pattern #121

0000000000000001

// Proc load_unload

000010_0000000000001111

0011001110011100100101001010010011001100001100110111010111000000111111000000000111100111111001101110011000
10011001111111001100110001_011111

01111101001000101001111000110000100011100101010110010000010101111000111000011011111000001111111110111000110
11110100111111011111000011_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0100101010000_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1001011101000_100000

//Pattern #122

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
00000000001000101001111000110000100011100101010110010000010101110001110000011100101110101111111110111000101
1101010011111001100000100_011111

0010101000011011000011100110100010111000110000011111000000001111100000111110110100110011001000111101000000001
0110001110111110110010011_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0011100010001_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1010001111010_100000

//Pattern #123

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000000011011000011100110100010111000110000011111000000001111111011011110110111111000101000111101000000111
1000101110111011010000100_011111

1011110100101001010111000000001100000000111101011101010010011100011110110001101111110011000111000010111110111
0101111001110100110110010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011011010000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010101001100_100000
```

```
//Pattern #124

0000000000000001

// Proc load_unload

000010_0000000000001111

1110101100101001010111000000001100000000111101011101010010011100000111001000000000100101010111000001011011100
1010111001110101010010001_011111

10010111100001010111010100000011001010110001000010010001011101101000001001111101011001100000101010011101100
1110001011011101001010011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010100101110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111010110110_100000

//Pattern #125

0000000000000001

// Proc load_unload

000010_0000000000001111

0000001110000101011101010000001100101011000100001001000101110110111111000000000011010110110101110011101101
0010101011011001010110001_011111

0010111010010111110001101100100101101001101110001010001100010100000000110001100010100100000110011001111001011
1111011100001011101110001_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1111010110111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111110001110_100000

//Pattern #126

0000000000000001

// Proc load_unload

000010_0000000000001111

10001100001011110001101100100101101001101110001010001100010100001100001100000000100011111100110011110010100
0111111000010001011110001_011111

00100111111110110111000110101100101101111011101001111011010011010100010101011110011111101101000001100101100
01101000000100011110011001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011010111101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110110011100_100000

//Pattern #127

0000000000000001

// Proc load_unload

000010_0000000000001111

01110100111101101110001101011001011011110111010011110110100110101011101000000000110011011010000011001101000
11010000001000111000110001_011111
```

```
01100011011010101010010110100111001110011101010100001010100100011010100111110101000110000000001111111001111
00011101001111111000000001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101000100000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110101100100_100000

//Pattern #128

0000000000000001

// Proc load_unload

000010_0000000000001111

10010001011010101010010110100111001110011101010100001010100100011111000100000000001101010010001000110101111
00011101001110110101000001_011111

10110001111101100000101111001011011001001110100011101101000110011010011001100010000010001010101011101011011
10100100100111000110100010_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0001000110000_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1110110011110_100000

//Pattern #129

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1110100011110110000010111100101101100100111010001110110100011001011000000000000001100110110101010110011010111010010010011111101100001_011111

0011111111000100100111001001110101001101111111011001000101101111011000100001111010001010100111101000011110101000111010100010010100010_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0011001011010_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1110011100010_100000

//Pattern #130

0000000000000001

// Proc load_unload

000010_0000000000001111

1100010011000100100111001001110101001101111111011001000101101111000111000001110000111001100111100011100110101000111010100100110100 11_011111

1010100000011111100010111000010001011000000000111110110101111000001010100110101100101111000101111011010110011011111101101110100100 0001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001000000010_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1101111010100_100000

//Pattern #131

0000000000000001

// Proc load_unload

000010_0000000000001111

0000011100011111100010111000010001011000000001111101101011110000110101010000000001011110000000100101111000 1
1011111011011000001110001_011111

11101000010001100000000101100110111111111001000100101010010011111110010010101100010110011100010011001101111 0
1010101100100001100100010_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0100111011000_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1010110100110_100000

//Pattern #132

0000000000000001

// Proc load_unload

000010_0000000000001111

0010001001000110000000101100110111111111001000100101010010011111010110010000000000101101100010010010110111 0
1010101100100111111110001_011111

1000011010111000011011101111110000110010011010110010011110011000101011110101001101000000101001111001111111 01
1011110100110010110100010_001011
```

```
// Proc multiclock_capture

000001_0000000000010010

0000000000000_0110111011001_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1110101011000_100000

//Pattern #133

0000000000000001

// Proc load_unload

000010_0000000000001111

101110001011100001101110111111000011001001101011001001111001100011010010010100100110101010100111111010101101
10111110100110001100010011_011111

010010010010110001100100011000100010000010101110101100110001110100111100000000010010101011110010000111110100
0111010101001010000100001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110111101010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111011100100_100000

//Pattern #134

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
01100100001011000110010001100010001000001010111010110011000111011111110100000000001110111011110100111011100
01110101010010000000000001_011111

10000110101011010010100000101000010111101000010101111110101110101011010001100110001101100111111110100001110
1111100011100001011010010_001011

// Proc multiclock_capture

000001_0000000000010010

0000000000000_0010100001010_100000

// Proc allclock_launch_capture

000001_0000000000010011

Z000000000000_1001100011000_100000

//Pattern #135

0000000000000001

// Proc load_unload

000010_0000000000001111

10111010101011010010100000101000010111101000010101111110101110100110011101100111110001100111111111000110110
1111100011100001010000011_011111

11011011000110001000111001011011110011100111001000010000001010111010100011010010111011001011101010010010111
00100000001110001001110001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001001110001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000001011000_100000
```

```
//Pattern #136

0000000000000001

// Proc load_unload

000010_0000000000001111

1011011100110001000111001011011110011100111001000010000001010111101001101010011011010000011101011101000011O
0100000011100010101110011_011111

0100100010101101101001001010100011101011011000101011111111001101011110101011100000111000001010001000011010O
0111000000111000101000001_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010111011111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100110110100_100000

//Pattern #137

0000000000000001

// Proc load_unload

000010_0000000000001111

0101000101011011010010010101000111010110110001010111111110011010011011100000000001101100010100010000110101 1
0110000000111000000000001_011111

1111111011000101010111000000111011101100100010010100001001110011001110000000101001110100110001000100110100110
1100100011111100101111100_001011

// Proc allclock_launch

000001_0000000000010000
```

Jaret Williams
Nathan Pen

```
Z000000000000_1110001110100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001000001000_100000

//Pattern #138

0000000000000001

// Proc load_unload

000010_0000000000001111

1101100111000101010111000001110111011001000100101000010011100110000000010000000110000010100000100011010011O
110010001111111001100010_011111

0111111110001101110110110000010111011110010110010110110101000011010100010100001110010100110100101000100101 1
011011010001111111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010101001001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011001100000_100000

//Pattern #139

0000000000000001

// Proc load_unload

000010_0000000000001111

1101101000011011101101100000101110111100101100101101101010000110000000001010001000110011101001010001001011 0
110110100011100000000001_011111
```

```
10000011110011111010001101000001100101100000000011100000110111111101101010100011011001110101010010010101001O
0110010000010010111000110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011000000011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111011001000_100000

//Pattern #140

0000000000000001

// Proc load_unload

000010_0000000000001111

00101100100111110100011010000011001011000000000111000001101111111100101000001001110011011010100100101010010O
11001000001001000001100O1_011111

00101001100010011110011010000101101010001010110101010000110101111111000010010101010011101110010110100001001
0001000001111011111010110O_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010011101010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011111100000_100000

//Pattern #141

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1101011110001001111001101000010110101000101011010101000011010111100101101001011000111111001111110100001001
0001000011110100001010010_011111

010011001010111100000101001110101011101100110011100100111000000110001000011000000001111010000110001100000011
1000100110000000010011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010010010100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011011110000_100000

//Pattern #142

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001010111100000101001110101011101100110011100100111000000100000001000000010111101101111011011000000011
1000100110000001100110010_011111

00101001000101101101100000011010110101011001010101101100101111010111101110000010100011011101000010011101011
1101001101101110011011110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010001001111_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1111010000010_100000

//Pattern #143

0000000000000001

// Proc load_unload

000010_0000000000001111

11011001001011011011000000110101101010110010101011011001011110100101011101101111000010111010000100111010111
1010011011011011110100001_011111

01111111110000000010110100010110100000011101100100111001001000100011101011100110010101100010100011110101111
1000100110110011111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000110010101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110000100010_100000

//Pattern #144

0000000000000001

// Proc load_unload

000010_0000000000001111

101100101000000001011010001011010000001110110010011100100100010000000000001110101001000010101000111101011111
0001001101100100000000001_011111

10000110011110000100001010010111000010010101001110001110110010101011110011100100000000111110101101111010101
1100110110011110110000110_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101010110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011001010000_100000

//Pattern #145

0000000000000001

// Proc load_unload

000010_0000000000001111

0010111011111000010000101001011100001001010100111000111011001010111010100100111000101001111010110111101010 11
1001101100111111100000001_011111

1111011110011010101010100110000111110110100000100011000111111010000011111100101010100101110010101101111 0011
1010010011110000010001100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110010000000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000000000000_100000

//Pattern #146

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
1010101010011010101010100110000111110110100000100011000111111010110010111100101100000000000000000011011110011
1010010011110101010100010_011111

1111111100000101111101110010101001100001110001011101001110011011010110110000010101010010101011101111100100 0
00001010111101110100011 00_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001110100100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001001111000_100000

//Pattern #147

0000000000000001

// Proc load_unload

000010_0000000000001111

1001101100000101111101110010101001100001110001011101001110011011000000010000000111110010111100101111100100 0
0000101011110000001010010_011111

0111111110001110001001111011101110110110010100110110111010001110001000100010001111010011000001101000011011 0
1100001100011011111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000010101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000100111000_100000
```

Jaret Williams
Nathan Pen

```
//Pattern #148

0000000000000001

// Proc load_unload

000010_0000000000001111

0111011100011100010011110111011101101100101001101101110100011100000000000010001001110010000001101000011011011
1000011000110000000000001_011111

0110111101111000000010110010010001110110011100011100110011000110110000001011000101100101001001001101001011011
1011000100100110101001100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010001100010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000011001000_100000

//Pattern #149

0000000000000001

// Proc load_unload

000010_0000000000001111

0010010001111000000010110010010001110110011100011100110011000110011000110110001110011000100110001010010110110
1011000100100001001000010_011111

1010101101110110100101100001011100110101110001101111010000010011001111110111100110010010000010101110011011000
0101000011110100000001101_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1110101011110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010110000_100000

//Pattern #150

0000000000000001

// Proc load_unload

000010_0000000000001111

10010110011101101001011000010111001101011100011011110100000100110000000100000001011010010110100111100110110
0101000011110011101100010_011111

01000001001010001000100110000101001010000000000111001100111011101000110110001100111111110001011000010111100
1111110000101000111111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101010101011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100000011000_100000

//Pattern #151

0000000000000001

// Proc load_unload

000010_0000000000001111

01010001010100010001001100001010010100000000001110011001110111011001101100101011110000000010110000101111001
1111100001010110111010001_011111
```

```
10100110010101111000001010111010000010001101000110011011110001101111101001010010110000011101111000110111110
10111010110110101000001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010100100100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010000000_100000

//Pattern #152

0000000000000001

// Proc load_unload

000010_0000000000001111

01010111010101111000001010111010000010001101000110011011110001100000000100000001000010010000100100110111110
1011101011011110001100010_011111

1111001111001001001000011100101010110011110101000101010100000000100011001000100011000101110100001010111100
0011101100110100111101100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100010110100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010100010_100000

//Pattern #153

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1100100111001001001000011100101010110011110101000101010101000000000000001000000010010100100101001010101111100
0011101100110010101010010_011111

1100011100101000001011110000111001011101011100000110001001111111101111000111000011001001110110000100000001
1110010010100011100101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110001001000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010100000_100000

//Pattern #154

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000000101000001011110000111001011101011100000110001001111111010100010101000100101010001010100010000000l
1110010010100011111110010_011111

1110000101110100100010001010110110111011101011100011111010111001000101111011001010001100001100011010001110l
1100011011100100101001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101110010110_100000

// Proc allclock_capture
```

Jaret Williams
Nathan Pen

```
000001_0000000000010001

Z000000000000_1110100111010_100000

//Pattern #155

0000000000000001

// Proc load_unload

000010_0000000000001111

1010111001110100100010001010110110111011101011100011111010111001111111101111111011100101111001011010001110 1
110001101110010101101001 0_011111

101001011101000001111101101010000110101001001010001110010111111110100011001000000111001101100011111011001 0
010111010110101010000001 0_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000101110000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110011010100_100000

//Pattern #156

0000000000000001

// Proc load_unload

000010_0000000000001111

011010101101000001111101101010000110101001001010001110010111111100011111000000000101100101100011011101000 10
110011010110101001010000 1_011111

101010011111110110101000111100010001100111110001100100011010110010001011001011001001111000100010111000110 11
110101001011101011001101 0_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001100000111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101001000110_100000

//Pattern #157

0000000000000001

// Proc load_unload

000010_0000000000001111

0101000111111011010100011110001000110011111000110010001101011001000101000000000000001001001000101110001101111
10101001011100101000100001_011111

0111101111110110111001100111000010100111101001100101010000101010011101010010011100110101101010101111111111000
1100001010011110101010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110001110110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101000111100_100000

//Pattern #158

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
0010101011111011011100110011100001010011110100110010101000010101011111110000000001110001011100010011100010001
10000101001110100110000 1_011111

1001110100100111011100010101001110111010010000000101111000110100001101110100010001011000011010100111110001 0
1111001000011000001011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111111100101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000101001000_100000

//Pattern #159

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000001001110111000101010011101111010010000000101111000110100000111011100010001001010011010100111110001 01
111001000011001001110101 1_011111

1011100100000110010111111010100011010000011010011110101000101110000110011110111100000110101101100101000000 1
101010000011111010000110 0_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011010100111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011000000100_100000
```

Jaret Williams
Nathan Pen

```
//Pattern #160

0000000000000001

// Proc load_unload

000010_0000000000001111

000000000000110010111111010100011010000011010011110101000101110001010000000000000000000110110110010100000011
01010000011110101000100001_011111

101011111011100110011011010000100110101101001100000011100011011100001011010000100001001111111100010100101111
10001000111000000101000000_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100000010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000100101100_100000

//Pattern #161

0000000000000001

// Proc load_unload

000010_0000000000001111

100001000111001100110110100001001101011010011000000111000110111010000101000000000101001001010010010100101111
00010001110001000010000001_011111

010001000010110000111001100011110001010101001010110110011001000000111011011000010101000011001010111010010110
00100000101001110000100010_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1011101110110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010011101010_100000

//Pattern #162

0000000000000001

// Proc load_unload

000010_0000000000001111

0010110000101100001110011000111100010101010010101101100110010000000000001000000011011100110111001011101111110
0010000101001010010100010_011111

00111110001011000000100110111101010000001001100110001010001000111011001111101101111010001000010010110000100
1010011011001000010010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010010110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100110010110_100000

//Pattern #163

0000000000000001

// Proc load_unload

000010_0000000000001111

01000111010110000000100110111101010000001001100110001010001000111101001100000000000100110000000100101100001001
0100110110010001100110001_011111
```

```
10101011111011001001110100010111001010000101111000000101011001100101011011011100001010000111001100101000011
11100011110000011011001 00_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011110010001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110000110010_100000

//Pattern #164

0000000000000001

// Proc load_unload

000010_0000000000001111

00000000110110010011101000101110010100001011110000000101011001100000011111011100001100001111001100101000011 1
110001111000000000101000 01_011111

1100001000000100101110101110110010101001111100110011101110001110110100011000000110111101000101001011001110 0
01000010000101010111100 01_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110110100101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000111001000_100000

//Pattern #165

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

0000100100001001011101011101100101010011111001100111011100011101000001000000001001001110000101001100111000000
10000100001011111001100011_011111

11111101011001010100011100101001001101100000111001111101010100011110100111010101110110010111110111011011001
01000110111000100100000010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100101010100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101101000110_100000

//Pattern #166

0000000000000001

// Proc load_unload

000010_0000000000001111

00110110011001010100011100101001001101100000111001111101010100011111110000000000000101100001011001010100001
01000110111000111111010001_011111

01010011110110001001110101001001010100110010101010110000011010011110100101110010011011110001010000111101101
1010011000100101100100110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000010001111_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1110001110110_100000

//Pattern #167

0000000000000001

// Proc load_unload

000010_0000000000001111

0101010110110001001110101001001010100110010101010110000011010011100010110000000000111000100101000011110110110
10011000100101010101010001_011111

0001000101100010110111101110100100110111110010011011001110110001100111110001100001100100011011011111011011
00001110101011110101100 10_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000000110000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011001011010_100000

//Pattern #168

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000001100010110111101110100100110111110010011011001110110001000110100001101011010011011011010110000011010
01111010101110010010100_011111

1110101000110011000000011000101101000111001000110001011000101110001111110101111101011010011000111100000100111
100100000011010110100 10_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101001000000_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000011110000_100000

//Pattern #169

0000000000000001

// Proc load_unload

000010_0000000000001111

000101100011001100000000110001011010001110010001100010110001011100101111001011110011110000110001100010010011
110001000000100000000100_011111

11000011100100110010010110101100110111001000110001110111000111100000010001110111011000101011010011011111001
11100111111011110110100010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010100100100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111000011010_100000

//Pattern #170

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
10101100100100110010010110101100110111001000110001110111000111101111111011111110110000111100001100100101001
111001111110100011110001O_011111

1011011101011101001011111100101001111100101111001000111010010101001010001000110101011001011100011010000101 1
0011111100110111100010000_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111011000111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110000010110_100000

//Pattern #171

0000000000000001

// Proc load_unload

000010_0000000000001111

00011101101110100101111110010100111110010111100100011101001010100001100000000000001000001010000010100001011 0
0111111001101000111010001_011111

11100010110001001101001101000011010000010000111011100001000001001000111110001100101101110100110011000111010
1010011001001010010011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111010111001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001100000100_100000
```

```
//Pattern #172

0000000000000001

// Proc load_unload

000010_0000000000001111

10100110100010011010011010000110100000100001110111000010000010011101011100000000000001101001100110001110101
0100110010010100000100001_011111

11100111100111110010010001010110100100001010110111000010110000110111011001111011011010001010000011000111111
0111111011110111111110010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100000110110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101101101000_100000

//Pattern #173

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001001111110010010001010110100100001010110111000010110000111111111011111110101101101011011001100000111
0111111011110101011010010_011111

01101010101000010111011110001011110100010011001001101000000100010010111000110101000001111001100101001100111
0010001000101110011101111_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1101010110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100110010100_100000

//Pattern #174

0000000000000001

// Proc load_unload

000010_0000000000001111

0010001001000010111011110001011110100010011001001101000000100010001111110000000001001100001100101001100111 0
0100010001011101000100001_011111

1000101110111101111000000010110110000110000011010001100101011111110110110000011110101100111100100001100110 1
0011110100010001100100010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010101110110_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000101100010_100000

//Pattern #175

0000000000000001

// Proc load_unload

000010_0000000000001111

0010110110111101111000000010110110000110000011010001100101011111000000001000000010011010000110100011101011 01
0011110100010000011010010_011111
```

Jaret Williams
Nathan Pen

```
10100011001111001111010001010100100011101000100100010101111000011000110111001100011001011000100000101100000
1101000010101011110011010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111011110001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010111101010_100000

//Pattern #176

0000000000000001

// Proc load_unload

000010_0000000000001111

00000000011110011110100010101001000111010001001000101011110000111001100110011001101111010001000001011000001
1010010111101001010110101_011111

00101100000111011010000100010111011010011100110001101101111010000111000001111111011111000111111111101100011
1101000001011100001010010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000110001010_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0010011100101_100000

111111111111111111_0000000110100101100_100010

//Pattern #177
```

```
0000000000000001

// Proc load_unload

000010_0000000000001111

011010010001110110100001000101110110100111001100011011011110100010000000100000001000110001111111100011000111101000001011000111010011_011111

101100001000110010110101101011010111111001100010010101000101000101000011110001101111101100001100000010111110001001001000111110000111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110001111101_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0100100001101_100000

1111111111111111111_1110000100100001100_100010

//Pattern #178

0000000000000001

// Proc load_unload

000010_0000000000001111

011000010001100101101011010110101111110011000100101010001010001010000111100011011111011000011000000010111110001001001000111100001110_011111

1011001001011111101011010010100111010010110000110011110011100100100000101011111111000001111101011110110001100111010000100000110100010_001011

// Proc allclock_launch

000001_0000000000010000
```

Jaret Williams
Nathan Pen

```
Z000000000000_1001011101010_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0001111000001_100000

111111111111111111_0000001110000000000_100010

//Pattern #179

0000000000000001

// Proc load_unload

000010_0000000000001111

000000000101111110101101001010011101001011000011001111001110010011000000110000001011101011101011101111010110
0111010000100101011010011_011111

0010010101101101011011001100010101111110000101001001001111100001110010011010001100000001001011111101100110111
1010100111110100101111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001101001000_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0011111101001_100000

111111111111111111_0110001010001101100_100010

//Pattern #180

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
0110110001101101011011001100010101111110000101001001001111110000110110101101000110100101100101111101100110111
1010100111110001010010001_011111

1011101110001110010100010100000100001001100110000000100110111111010101101100110100111000110010111001111l1001
0100010010110000000001111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001101101_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0001010001110_100000

111111111111111111_101101010011101110O_100010

//Pattern #181

0000000000000001

// Proc load_unload

000010_0000000000001111

0111011100011100101000101000001000010011001100000000100110111111010101101100110100111000110010111001111110010
1000100101100000000011110_011111

1111000000010100001011001100001110011000000001100010100001110000011010011101011010100001111000111010011101OO
0011100011110011101010000_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110101110101_100000

// Proc allclock_capture

000010_0000000000010001
```

Jaret Williams
Nathan Pen

```
0000000000000_0010100100110_100000

1111111111111111111_0101101000000001111_100010

//Pattern #182

0000000000000001

// Proc load_unload

000010_0000000000001111

11100000001010000101100110000111001100000001100010100001110000011010011101011010100001111000111010011101000
0111000111100111010100000_011111

010010100110100100011100011101100011010011010001111101011001010100100110101011101110001000101010000001000011
0000001011111011010011110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000100101000_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0001001010000_100000

1111111111111111111_0111010110000000000_100010

//Pattern #183

0000000000000001

// Proc load_unload

000010_0000000000001111

000000000110100100011100011101100011010011010001111101011001010111111000101011101010010000101010000001000011
0000001011111000000000001_011111
```

```
01010000001111111100010000010001011001111100111111010101110101101111001001110101111101001100011001100010101
01001001100000011100111110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001001110_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0110100010101_100000

1111111111111111111_0000000010101010111_100010

//Pattern #184

0000000000000001

// Proc load_unload

000010_0000000000001111

11010101001111111100010000010001011001111100111111010101110101101100010100000000010010000110001100110001010101
01001001100001100111110001_011111

10100001111100101110111101101110110111111011001101011000110101011110101100100101110010110010001101100101000
11010100000000011011100110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001011010_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0100111011100_100000

1111111111111111111_0111011100011101100_100010
```

```
//Pattern #185

0000000000000001

// Proc load_unload

000010_0000000000001111

0110111011110010111011110110111011011111101100110101100011010101111011100010010111010000001000110110010100
0110101000000010110011110_011111

0010010011111000111100110000101110100000111111100001011100001010100010000101101100010011010000111101100111
1011000010101111001001110_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101001110010_100000

// Proc allclock_capture

000010_0000000000010001

0000000000000_0100111100100_100000

1111111111111111111_1101101010010100000_100010

//Pattern #186

0000000000000001

// Proc load_unload

000010_0000000000001111

0000101011111000111100110000101110100000111111100001011100001010011111110101101101110010010000111101100111
1011000010101111100110001_011111

0011000100111101110100000000100010110010010111101100011111001111111111010111011011111010111101011100111101
1011100000000111111111011_001011

// Proc allclock_launch
```

```
000001_0000000000010000

Z000000000000_1100000011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110101011100_100000

//Pattern #187

0000000000000001

// Proc load_unload

000010_0000000000001111

0010001001111011110100000000100010110010010111101100011111001111110000000000000000110101011101011100111101111
0111000000001001111110001_011111

1101101010100011110101000111100001010000011101100100100111011100100110111000001101110111000100111101101 0001
1111101101111100010010101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001111101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010001101100_100000

//Pattern #188

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
1010000000110111101010001111000010100000111011001001001110111001101110110000000010110001001001111011010001 1
1111011011111111100000001_011111

0010111110001100011100000001101110010101011001110111010011100011000110010010111011001100011110010000111110 0
0100010101110111110000100_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001110101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110110011100_100000

//Pattern #189

0000000000000001

// Proc load_unload

000010_0000000000001111

0001100010011000111000000011011100101010110011101110100111000110101111000000000011001101111100100000111110 00
1000101011101000000000001_011111

0001001100010101110110001101011100110101111011111100000101000011001110110101100010101010000111001000000110 1
1111101110101010110100010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011111101100_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001010001100_100000
```

```
//Pattern #190

0000000000000001

// Proc load_unload

000010_0000000000001111

0001010100010101110110001101011100110101111011111100000101000011000000010000000010001010100010101011111111101
111110111010111010111000 1_011111

0110100010111011111010101001100110110010100111101000101100111101110011100000110100111101100111100110011100 10
001001011111101111111 0101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111011100011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010001100110_100000

//Pattern #191

0000000000000001

// Proc load_unload

000010_0000000000001111

1100101001110111101010100110011011001010011110100010110000111000000000101000000000011000101111001100111001 00
0100101111110110010100001_011111

1010011101111000101001101011001011111110010111010011101001000011001011010001000000100101001111111111001 0010
0010000100001010011010101_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1101100111011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100000010000_100000

//Pattern #192

0000000000000001

// Proc load_unload

000010_0000000000001111

1111110011110001010011010110010111111100101110100111010001011010011011100010000001000000011111111100100100
010000100001011110001000l_011111

0101100100001100101011110101110011000101011111001110001001010000000101100100110010001011101000000111110111l
0010000101101101110001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001001111111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010100110000_100000

//Pattern #193

0000000000000001

// Proc load_unload

000010_0000000000001111

1111100100011001010111101011100110001010111111001110001000010110001001011100110010110010101000000111110111110
0100001011011000110010001_011111
```

```
011111010100001111001100011100001011110110011011101101110111101001000101101011100001101110010000110110110100111000001100111001101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101000011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110101001010_100000

//Pattern #194

0000000000000001

// Proc load_unload

000010_0000000000001111

100110001000011110011000111000010111101100110111011011101111010011100110010111001001010100100001101101101001110000011001111000011110_011111

0111101010000011100011001100000000000110111111100101101000101001111100101111110010110110000110100100011101000111111101011100110011011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001111101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001110011000_100000

//Pattern #195

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

100000000000011100011001100000000000110111111100101101000101001111110011111110010110011100110100100011101000
1111111010111000000001000_011111

01111001000011111101100110011001010101011010001001010110000010100110010010100001100000100111111000011001000
0010111001010011100110101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001000011011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011101011000_100000

//Pattern #196

0000000000000001

// Proc load_unload

000010_0000000000001111

10101100000111111011001100110010101010110100010010101100110010010011010101000011110101111111110000110010000
0101110010100000101000001_011111

10010100001111000101100101110100111101100011001011000110111011000110101111010111101000011011011000110001111
1111100001101101000001011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111011011001_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1101110001110_100000

//Pattern #197

0000000000000001

// Proc load_unload

000010_0000000000001111

011001010111100010110010111010011110110001100101100011011101100001011100000000001000111001101100011000111111
1111000011011011001010001_011111

011001101100101001010010011111010000111001001001011010001000111101110010000100000111001111100011000001001111
1110010110010001100011011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001110010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100011100100_100000

//Pattern #198

0000000000000001

// Proc load_unload

000010_0000000000001111

000111101001010010100100111110100001110010010010110100010001111011101101000000000011100011000110000010011111
1100101100100000000000001_011111

000100001000010100111010101011100000101110100001101000010110100010101110001010010100111011101001111001101101
1101101110011110010101011_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010111101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000010101100_100000

//Pattern #199

0000000000000001

// Proc load_unload

000010_0000000000001111

0101110001011100011101010101110000010111010000110100001011010001011100110000000010101000110100111100110110111011011100110111010100001_011111

1001100101110111001100010110011010101001101100011000001011001100100110101001000011001110110101010100100001111110011000101101011010111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110110010101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010110100_100000

//Pattern #200

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
00000000111011100110001011001101010100110110001100000101100110010111001000000000011010101010101010010000111
1110011000101010100110001_011111

11111110110100100111101000101101000011110101111100000110111001010001111000100001101001001001010011111000001
1110101100110001100111011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110000000011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111101011010_100000

//Pattern #201

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001010010011110100010110100001111010111110000011011100101001000000001000011110101110010100111110000011
1101011001100101001001000_011111

10001110000000100000100011001111010111001000011100101011000110111101011010001110001000010001100111111011000
1111011111001010011011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110000110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111000011010_100000
```

```
//Pattern #202

0000000000000001

// Proc load_unload

000010_0000000000001111

0001000110101101000100011001111010111001000011100101011000110111001110000001110011000011001100111111011000l
11101111100100000010000001_011111

1111000101001111011110011010111000100110110111011101001010001101001101001110100101111111101011001110101000O
11100100101100100011011O1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001001101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000111101010_100000

//Pattern #203

0000000000000001

// Proc load_unload

000010_0000000000001111

0100110101101001111100110101110001001101101110111010010100011010101101001101001010111100010110011101010000l
110010010110011110011000l_011111

0101000011001001100110111100000101000000000100010010001010100111011010101111110011111111000010110111100100l
1011001100100111100011011_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1111001101101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101011111010_100000

//Pattern #204

0000000000000001

// Proc load_unload

000010_0000000000001111

0100010110010011001101111000001010000000001000100100010101001110101011001111100111110100001011011110010011
0110011001001100100110001_011111

00001000000101100101111110111101000010111101100000000000101010100000110000000100101011000111011100111111101101
1100010001111100010101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101110010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010110010_100000

//Pattern #205

0000000000000001

// Proc load_unload

000010_0000000000001111

1100000001011001110000001110100001011110110000000000010101010000000101101001001010011010010111001111111011011
1000100011111010110010001_011111
```

```
11000110101001000100110110000101010010100110010111011101110110101010001011111111001101001110000110101011001
11000111010000111111110101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010110011011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100110000110_100000

//Pattern #206

0000000000000001

// Proc load_unload

000010_0000000000001111

100110110100100001000101000010101001010011001011101110111011010110001011000000000000011001100001101010110011
1000111010000101110110001_011111

01001011011001001101100011001111010100010010000111000111011101100110100100110110101110100100100011111000110
1100111001000111110110101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100001010011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011111100110_100000

//Pattern #207

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

111011001100100111010010100111101010001001000011100011101110110000000011000000000011111110010001111110001101
10011100100011100100100001_011111

001001110101011000101000010011011011111000001010111011101110011001111110101111001001100110011100111000000001
10110010011110011111101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010100100101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101110101010_100000

//Pattern #208

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001010110001010000100110110111110000010101110111011100110000011011011110011010111000111001110000000011
01100100111100101000000001_011111

111001111100101100011101011111101000110100000101100001010010000110111011100011000110100100000100110000001001
10100000110100001011010111_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010010110001_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1110011010110_100000

//Pattern #209

0000000000000001

// Proc load_unload

000010_0000000000001111

00001010100101100011101011111101000110100000101100001010010000110110000100000000010110010000100110000010011
0100000110100000110100001_011111

11110101001000000011010010011110100001000111111110100111100111001100110011100011110010100101101010100010001
0100001011111101011001011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011001011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101110010100_100000

//Pattern #210

0000000000000001

// Proc load_unload

000010_0000000000001111

00001000010000000110100100111101000010001111111101001111001110010001100000000000010011101011010101000100010
1000010111111010011110001_011111

00100001001010011000101101111011101110111101011001011111001000111001110101101010001001000111101010101010001
1011011110011000100100101_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101110110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101011010010_100000

//Pattern #211

0000000000000001

// Proc load_unload

000010_0000000000001111

01110111010100110001011000111010011101111010110010111110010001110001011011010100010110101111010101010100011
0110111100110101111100001_011111

01110100001010010010001001111001010100110101011111001100001011010000001011001110010000010101100111010010001
0100110111000010000101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000001011011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000100111100_100000

//Pattern #212

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
1010011001010010010001001111001010100110101011111001100001011010100001000000000011100100101100111010010001010011011100001111001000001_011111

01100100010011011101000011011001000100110100000011111110100000000011101110011100001100011000000100100000011010111000011001011110101 1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110111101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000010000100_100000

//Pattern #213

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001001101111010000110110010001001101000000111111101000000000000000000000000000001000000000100100000011010111000011001111111010001_011111

11111010111101011001110101010000111101001001111000010100101110000011110111100110000101001011110101011111011101011011101100110100010 1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001100101011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010111111000_100000
```

```
//Pattern #214

0000000000000001

// Proc load_unload

000010_0000000000001111

001110101110101100111010011110111110100100111100001010010111000011000001110011001111110101111010101111110111
0101101110110101000010001_011111

010101000000110100010001110101100101000111111100100100011101000011001110111001111110111111001010100001110001
1010101111100011111101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000010110111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111110011110_100000

//Pattern #215

0000000000000001

// Proc load_unload

000010_0000000000001111

001000110001101000100011100111011010001111111001001000111010000101110111000000000110011111001010100001110001
0101011111000000000000001_011111

101111011110000110111111111001001111100100000111001011110010010101001110111001010110101000101000001101001001
0100001011010101100111011_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1011111011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010110011000_100000

//Pattern #216

0000000000000001

// Proc load_unload

000010_0000000000001111

1011110011000011011111111001001111100100000111001011110010010101000111001001010111001101101000001101001001O
10000101101010111111111001_011111

0100001011101100001011001010110000000011000101100011101011001001010010000101011010011001000110111100001000101O
10101000100011111110011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111100111011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001001001110_100000

//Pattern #217

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001101100001011001001000010000110001011000111010110010010111011111000000001001001001101111000010001O1
0101000100011000000000001_011111
```

```
0101101011010001000000011010001001000011000110100010001000011101000101011011010100010011001100011000011011O
10100010111111001011000101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111100100011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000100000000_100000

//Pattern #218

0000000000000001

// Proc load_unload

000010_0000000000001111

0100010010100010000000110010101110000110001101000100010000111010000111110110101000000010001100011000011011O1
01000101111110000000000001_011111

1011111100011000000000001001100011001000100011101010101001101101111010100110101000110001000100010001011110O
10110111000000000010000101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010100101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000010000110_100000

//Pattern #219

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

0011000000110000000000001110101001001000100011101010101001101101101010010000000000000100000100010001011110001
0110111000000110110110001_011111

0010010001100011011001111001000010001111100110111111101111101100010011111100000101000011001011100110110101 1
0101110011101110101111011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000011000011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110001111010_100000

//Pattern #220

0000000000000001

// Proc load_unload

000010_0000000000001111

1111011111000110110011110010000100011111001101111111011111011000110010101000001011110001010111001101101011 0
1011100111011110011111000_011111

0101100111011111100011001110111011001110010111100111110000111110100011011000100111010001101011111010011101 0
1001110010111100000010101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001100010111_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1101010110000_100000

//Pattern #221

0000000000000001

// Proc load_unload

000010_0000000000001111

0001100110111111100011001110111010001101110111100111110000111110111000110000100110110101001011111010011101010
01110010111111111100000001_011111

1111001011010100111101111100000011111001010000011101011110100011111001100010101001001111100010000011010111100
1111101010100100001101 1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101010000111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001111100010_100000

//Pattern #222

0000000000000001

// Proc load_unload

000010_0000000000001111

1000001110101001111011111000000111110010100000111010111101000111001110010101010000111110000100000110101111001
11111010101010000011 1000_011111

1110100110111000101111011011100010000100001011101100101101000000011010011100100011001110101100101110111010100
1000110110001110010101 1_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000011001001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110100101000_100000

//Pattern #223

0000000000000001

// Proc load_unload

000010_0000000000001111

01110001011100010111101101110001000010000101110110010110100000001011110110010001101001010110010111011101010
0100011011000011110110001_011111

01101010011100011110010100110101011111010110110010101110011000001100110101000111111101001101011100110110011
1000100110001010111001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111000100111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100000110010_100000

//Pattern #224

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
11111010111000111100101001101010100110101101100101011100110000010110001110001111011000001010111001101100111
0001001100010110110010001_011111

00000011101001110101001111100110000011110100001000110011100100100001100101100100010011001101000110111100101
0000111011011010101011011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011100000011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010101010_100000

//Pattern #225

0000000000000001

// Proc load_unload

000010_0000000000001111

00100100010011101010011111001100000111101000010001100111001001001100111111001000101010011010001101111001010
0001110110110100001001000_011111

00101111111011100110100111100110001001111110111110010110100000110111011110001000001101111011000100000001011
1001101001111110101011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001011010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111011101010_100000
```

```
//Pattern #226

0000000000000001

// Proc load_unload

000010_0000000000001111

0010110111011100110100111100110011101111110111110010110100000110011011110001000010111011011000100000001011l1
0011010011111000000000001_011111

00110110010100010111101011110111001011110100101001100000110000011101000010011010100111111010110100111110001
100000001110001000101010l_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100100111011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001001100000_100000

//Pattern #227

0000000000000001

// Proc load_unload

000010_0000000000001111

01011110101000101111010111101110101000011001010011000001100000111010000100110101001100100101101001111100011
0000000111000100101000001_011111

01101101100011101110111011010110101010110110001110011000100110100000100011110000100000010001000111101100011
1000011010101011000011101_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1100010000111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111000011010_100000

//Pattern #228

0000000000000001

// Proc load_unload

000010_0000000000001111

00000000000111011101110110101101000100011100011100110001001101001011110011100001110000110010001111011000111
000011010101010110111010001_011111

01000101101110110000101001010100101101100001000111001010110011100101001011111110010000111111010000110000010
100000110111001101000110 1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010110000111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001101010000_100000

//Pattern #229

0000000000000001

// Proc load_unload

000010_0000000000001111

00000000011101100001010010101001101001010010001110010101100111001000011111111100010101101110100001100000101
0000011011100001000110001_011111
```

Jaret Williams
Nathan Pen

```
1011001000111011001000001010111100100010001010110110100100100100110000110101100100001001110110010101110101011011011011011000101010 11_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000100100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110111110110_100000

//Pattern #230

0000000000000001

// Proc load_unload

000010_0000000000001111

1101001001110110010000010101111001000100010101101101001001001111100000000000000111110110110010101110101011011011011011010111100001_011111

1111001010010100101010000010101110101001101101001000010011111100101100001011001001110001010001011010001110001101111101010100010011 01_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111110101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010011110_100000

//Pattern #231

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

00101001001010010101000001010111010100110110000100001001111110010100100100000000110010011000101101000111000
1101111101010000010010001_011111

01011110100001101100000111000100101110010101001011101111010000111000001000100111010110000001101110100010000
1100100001001000101101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100100101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000011100100_100000

//Pattern #232

0000000000000001

// Proc load_unload

000010_0000000000001111

01110010000011011000000111000100101110010101001011101111010000111000010110000000000111000001101110100010000l
1001000010010110111100001_011111

01010000111100101000001010010001111010101111011010001000100101001101101110001001100111011111001000010000000
0110011000101111011011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011110000011_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1111110000110_100000

//Pattern #233

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001110010100000101001000111101010110110111000100010010100110110100000000000000011111110010000100000000
1100110001011001010010001_011111

110011100011001111111010101011001010001100001111011001011011100011111001100011001100111100011010011100110101
1100101010010000101001011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111000010011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111111100100_100000

//Pattern #234

0000000000000001

// Proc load_unload

000010_0000000000001111

111101010110011111110101010110010100011000011110110010110111000110110101000000000001111110011010011100110101
1001010100100011001110001_011111

110111100000000101101110101101110010110111011110011011101110010000000001100000101011100001010110111101011101
0111011010010101101010101_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111110011111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100110111100_100000

//Pattern #235

0000000000000001

// Proc load_unload

000010_0000000000001111

10111100000000101101110101101110010110110000001111011101110010001100000100000000111011001010110111101011010
1110110100101000000100001_011111

10001101010001101001001001101100111000101010001111001010101101011110110000100100001101100001111111001011010
1101010101110010101111011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110011110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100010101110_100000

//Pattern #236

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
00100100100011010010010011011001110001010100011110010101011010110110001000000000101010000011111110010110101
1010101011100110001010001_011111

011011101100100110111111111000010101111110001111011100011100101101001011101100001101001100100101111001100101
0110001010111110100001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111000101011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011100110100_100000

//Pattern #237

0000000000000001

// Proc load_unload

000010_0000000000001111

0010110110010011011111111100001010111111000101110110001110010110110100000000000000011001111001011110011001010
1100010101111011111100001_011111

011111101111101001000100100001110111111000101110010100111011001111100001011111000010000111011110010001000000
1101001011010010110111011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000100011001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010101010_100000
```

```
//Pattern #238

0000000000000001

// Proc load_unload

000010_0000000000001111

000000001111010010001001000011101111100010111001010011101100111101101011111100001010101001111001000100000001
10100101101000000000000001_011111

001101100101000110111100000011000011101000000100010100100101010001011101010010100010100100101001110100111111
01100110111111110101001101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110001110111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111011011100_100000

//Pattern #239

0000000000000001

// Proc load_unload

000010_0000000000001111

101000111010001101111000000110000111010010111010101001001010100000000000000000000110110110101001110100111110
11001101111111000110000001_011111

100010111001011000001110011101110000111110001110000100010010011001010011001111110010101100101001110101100000
11000000101111101000001011_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1110011010011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101101010110_100000

//Pattern #240

0000000000000001

// Proc load_unload

000010_0000000000001111

0001111100101100000111001110111000011111000111000010001001001100100101010000000000101011001010011101011000001
10000001011111111011100001_011111

1001011100011001001100000010110000000110001110111100100110101100010010100000111110010100001101101011001111110
00010000111111000100001011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111010101011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100110001110_100000

//Pattern #241

0000000000000001

// Proc load_unload

000010_0000000000001111

0011001000110010011000000101100000001100011101111001001101011000101010101000000000010001100101101011001111110000
00100000111111001100010001_011111
```

```
0111011011011100101100110011101111111001000101010010010110010110010011101100110010111001001010001110000001000
010110110110001111111101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110110010011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010011001010_100000

//Pattern #242

0000000000000001

// Proc load_unload

000010_0000000000001111

1001011010111001011001100111011111100100010101000011101101011001000111110011001010011001101000111000001000
0101101101100011100100000 1_011111

1011011010110110000000000001010111101111110000000000001011010101111101001110011100110011000100101111100011000
1000010000101011010101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100110011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110100100100_100000

//Pattern #243

0000000000000001
```

Jaret Williams
Nathan Pen

```
// Proc load_unload

000010_0000000000001111

0000000001101100000000000010101111011111000000000000101101010111111011111000000000001001011001011111000110001
000010000101000000000000001_011111

0111111101011111000100010100001001100111110110001111100100000000110010011001100111001100100011011001011000000
0101010100110000010000101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101100101011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101100011010_100000

//Pattern #244

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000010111111000100010100001001100111110110001001001100000000110110010101100111110001100011011001011000000
1010101001100111100100001_011111

1101010001110000000011000001100010101011110111110100101011110001000110101000111100001100000001001100111111110
1011011001000110011011011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001111011001_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1110100001000_100000

//Pattern #245

0000000000000001

// Proc load_unload

000010_0000000000001111

1110000011100000000110000011000101010111101111101001010111100010000101100001111001000010100100110011111111101
011011001000101111101000 1_011111

1110101010000010101100011100110101100101010100111101011001000010100010110000000001010010010001111010101011110
0101100101010100101110101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111000101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101111000110_100000

//Pattern #246

0000000000000001

// Proc load_unload

000010_0000000000001111

0000010100000101011000111001101011001010101001110001011010000101110101100000000000000111100001111101010111100
1011001010101101011000001_011111

0011110011000001011101011100000010010000100001011101001000000000111011101111000001111100111110111100110011 10
1001000001000100111011011_001011
```

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100001010111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111110001110_100000

//Pattern #247

0000000000000001

// Proc load_unload

000010_0000000000001111

000010111000001011101011100000001001000010000101110100100000000110011101000000000100011111110111100110011101
0010000001000100000001110001_011111

001011010110100101100001110000000110011010110111000111011010011110110011111001110000000100111110000010011100
1011010111000100001011011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101010100011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100010010110_100000

//Pattern #248

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
1101001011010010110000111000000011001101011011100011101101001111001010000000000000100100011111000000100111001
01101011100010100111110001_011111

01110101001111010010100001000001010000000011011000001000100110010111100000110101110110001101111010101100100
0100101100101010110111101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000111111011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001000100010_100000

//Pattern #249

0000000000000001

// Proc load_unload

000010_0000000000001111

00110010011110100101000010000010100000000110110011110000001100100101010101101011001000101011110101011001000
1001011001010000100010001_011111

001000111011011001011000010101100010101101110000111000010011000010011000000110000100101110011100101001110111
00001000000010001110100101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1110101110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011110110100_100000
```

```
//Pattern #250

0000000000000001

// Proc load_unload

000010_0000000000001111

1010110001101100101100001010110001010110111000011100001001100001101010000000000000110111101110010100111011100
0010000000100010101100001_011111

0110110110010100010010111100011100001110101110110011111110100111101001000111111111100011110100000110011110
0100101111100010111101011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111110110111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011010100000_100000

//Pattern #251

0000000000000001

// Proc load_unload

000010_0000000000001111

0010100000101000100101111000111000011101011101100111111101001111010110100111111100101011110100000110011110
01001011111000011101101000_011111

1110011110010000101000011101100110110011001000111011101011111100010011011010001110101001000111000001010111
10010000000111101111111101_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1001101110011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100101011000_100000

//Pattern #252

0000000000000001

// Proc load_unload

000010_0000000000001111

010001110010000101000011101100110110011001000111011101011001101100010110010001111101010000111000001010111100
10000000111101111110000001_011111

000100111100110001000101111001000000001110100011110000101101001111101111111000101111111000100010011000101000
010010010001000001110101l_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101111110001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010010101110_100000

//Pattern #253

0000000000000001

// Proc load_unload

000010_0000000000001111

000010111001100010001011110010000000011101000111100001011010011110110000100000000101010011000100110001010000
10010010001000000011100001_011111
```

```
0001111000000010110001100001010110101110111010010110010000110110101010010100110001110001011110010010011111100010001000010000101100101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010100010011_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001101101000_100000

//Pattern #254

0000000000000001

// Proc load_unload

000010_0000000000001111

0000101100001011000110000101011010111011101001011001000010100101011011010011000110110110111001001001111110001000100001001011101100001_011111

0011010110000100111011011001000111110011110011101110110010010001000110110001100011010011110111111000011000100111100111010001101001011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011011101111_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1001000001110_100000

//Pattern #255

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1110011100001001110110110010001111100111100111011101100100110110100111000000000001000001010111111000011000100
11110011101000000000000001_011111

0101010001010010000001001011010111010000101101110111110101110000101010111101110100001010111000011000100100001
1000110010100000111l011_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101111111001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011100101010_100000

//Pattern #256

0000000000000001

// Proc load_unload

000010_0000000000001111

1111101010100100000010010110101110100001011011101111101011100001110110011011101010100111110000110001001000011
00011001010000000000001_011111

0101010000011110010010111101011001010000011011110001100101001010100101111010010011001111101100001100000111000
0010100111101001110 0101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010101011111_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1110110000100_100000

//Pattern #257

0000000000000001

// Proc load_unload

000010_0000000000001111

1101111000111100100101111010110010100000110111100011001010010101111100010000000000001101011000011000001110000010100111101010000000001_011111

1111011000011110011111100001110011000101111111110010100011100100111010000101110000101101101011100001110000100000001110010010010011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001001110001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100101110100_100000

//Pattern #258

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000000111100111111000011100110001011111111100101000111001001100000000000000000111010001011100001110000100000011100100110010010001_011111

1110001110001100101010111010100101111010101111101111011001101001101110101101010100010000111000100001100101011000110100100010000011101_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000101010101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110001000110_100000

//Pattern #259

0000000000000001

// Proc load_unload

000010_0000000000001111

1110110000011001010101110101001011110101011111011110110011010011010000110000000000010001110001000011001010 1
1000110100100110100110001_011111

1000100100110011110011011000101111110011001000100111101011010001000000100001011101011001000101001111101111 0
0000000111111010010101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010011011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1101010111100_100000

//Pattern #260

0000000000000001

// Proc load_unload

000010_0000000000001111
```

```
0110011101100111100110110001011111100110010001001111010110100010100101110000000011101010001010011111011110
0000001111110011001110001_011111

0111110011110001111111000100110101011001100110111001000110110111000010010001111110110111011101011110101001
0110111001001000111010 0101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000010100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011000000110_100000

//Pattern #261

0000000000000001

// Proc load_unload

000010_0000000000001111

1011001111100011111110001001101010110011001101110010001101101110011100000000000000000001111101011110101001
01101110010010011111000001_011111

0100111000000101101101100110100101010100111111011010111010000011110100101110000010101100110000000010010011
10110100000001010111100010 10_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011100100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1110111111000_100000
```

```
//Pattern #262

0000000000000001

// Proc load_unload

000010_0000000000001111

1111101100001011011011001101001010101001111110110101110100000111101111111011111111111110110000000010010011101
10100111111011010100101_011111

0100101001000010011111010000101011000000111110001010101000110100100011111100111100100110110100100100011000l0
000001010011010101111111101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000110011101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111111100010_100000

//Pattern #263

0000000000000001

// Proc load_unload

000010_0000000000001111

0000000010000100111110100001010110000001111100010101010001101001011011111001111000011111101001001000110001 00
00001010011010000000001011_011111

0011110001111001101110010000011110111101101011110111010100110101000100010101001101111001000111010100000001 00
10111111100100011001011 01_001011

// Proc allclock_launch

000001_0000000000010000
```

```
Z000000000000_1011010100001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100110100110_100000

//Pattern #264

0000000000000001

// Proc load_unload

000010_0000000000001111

01101010111100110111001000001111011110110101111011101010011010100110000000000000001011000011101010000001001
0111111100100011110110001_011111

00001001010001100011000100011011011000111111100110100111100011110111110010010000101100000011111001001001000
0000001011100000010001010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1010000001101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111001000000_100000

//Pattern #265

0000000000000001

// Proc load_unload

000010_0000000000001111

01001111100011000110001000110110110001111111001101001111000111100001111100011111000100110111111001001001000
0
0000000010011110001110101_011111
```

```
11001000110000101101011111010111111011010101111001011110001010010000111100001110111100001100001100101001110
10011111011111001011011O1_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1011001101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1111010011010_100000

//Pattern #266

0000000000000001

// Proc load_unload

000010_0000000000001111

10111100100001011010111110101111110110101011110010111100010100101011110100011101110010111000011001010011101
0011111011111101111001011_011111

11100001111011011111011100001000000010111110111101011100010110111101111111001011110101100001111000001101000
0000010001110000100101101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100100010001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000110011010_100000

//Pattern #267

0000000000000001
```

```
// Proc load_unload

000010_0000000000001111

1011100011011011111011100001000000001011111011110101110001011011111100111100101111100110000111100000011010000
00001000111000000000001011_011111

0110000000110011110000001111111111010011111001101001011100000001000001000001100001000010110000111101101010100
00100100101010101100111101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1101011110101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1000101000010_100000

//Pattern #268

0000000000000001

// Proc load_unload

000010_0000000000001111

0010111001100111100000011111111110100111110011010010111000000010000000001011000010001010000000111101101010100
01001001010101100110110111_011111

1110011111010100100010010001000101100101001001110101111110000001011001010010011010001111101000010101011011111
10000000100100011011011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1000100001001_100000

// Proc allclock_capture
```

```
000001_0000000000010001

Z000000000000_1100111101000_100000

//Pattern #269

0000000000000001

// Proc load_unload

000010_0000000000001111

10101001101010010001001000100010110010100100111010111110000010110100110100110100101111000000101010110111111
0000000100100001000101011_011111

00101000011101000001010011010001100011100000101100111111100010001100101110010001110101011101101100100001101
0100000110101010010011101_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1001010111101_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1011100101010_100000

//Pattern #270

0000000000000001

// Proc load_unload

000010_0000000000001111

000101101110100000101001101000110001110000010110011111110001000100000000000100011101001111011011001000011010
1000001101010011111111011_011111

11100001000100111111000111100001100100110111111011110000100010110001011100010011100000001100110000110001011
0001010010101010100000101_001011
```

Jaret Williams
Nathan Pen

```
// Proc allclock_launch

000001_0000000000010000

Z000000000000_1111110101001_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1010101010010_100000

//Pattern #271

0000000000000001

// Proc load_unload

000010_0000000000001111

1110000100100111111000111100001100100110111111011110000100010110000000010010011101010101100110000110001011O
0010100101010111111011011_011111

0101101111001001111001110010110100111100100101111011110001101110100101111010001011011100010010010010010001 1
1010100101010101100101010_001011

// Proc allclock_launch

000001_0000000000010000

Z000000000000_1100100111010_100000

// Proc allclock_capture

000001_0000000000010001

Z000000000000_1100010110110_100000

// Proc load_unload

000001_0000000000001111

0011110011001001000000000010110100111100100101111011110001101110110111100000000001101000010010010010010001 1
1010100101010011011100001_011111
```

```
//End of Patterns
0000000000000000
```

Appendix F

Do.do file

```
rule IC (inter_layer_clearance -1 (layer_pair cc via))

rule IC (inter_layer_clearance -1 (layer_pair poly via))

rule IC (inter_layer_clearance -1 (layer_pair active via))

rule IC (inter_layer_clearance -1 (layer_pair nactive via))

rule IC (inter_layer_clearance -1 (layer_pair pactive via))

setup_check (antenna_rule off) (conflict on) (corner_corner_check off) \

  (xtalk on) (end_cap off) (length on) (limit_way off) \

  (min_width_wire  on) (min_mask_edge_length off) (max_vias off) \

  (miter off) (order off) (polygon_wire on) (protected off) \

  (reentrant_path on) (same_net_check  on) (segment off) \

  (stub off) (use_layer off) (use_via off)
```

Default.view

```
# Version:1.0 MMMC View Definition File

# Do Not Remove Above Line

create_library_set -name CommonTiming -timing {innovus/osu05_stdcells_expanded.tlf
innovus/osu05_stdcells.tlf}

create_constraint_mode -name TimingConstraints -sdc_files {../ece4150/project_N/cpu/src/mips_scan.sdc}

create_delay_corner -name corner_min -library_set {CommonTiming}

create_analysis_view -name view_hold -constraint_mode {TimingConstraints} -delay_corner {corner_min}

set_analysis_view -setup {view_hold} -hold {view_hold}
```