A. JAMES CLARK SCHOOL OF ENGINEERING

Department of Electrical and Computer Engineering
Advanced Laboratory of Digital Systems Using Systems Verilog
Professor Jerry Wu

A Dual Core MIPS CPU

By,

Alec Felix and Mitchell Fream

Table of Contents

I Introduction

As the final project culminating all that was learned in ENEE 459D a simplified dual core Microprocessor without Interlocked Pipelined Stages (MIPS) was developed. Each individual core of the processor is identical and was created with the traditional controller (Controller + ALU Control), datapath, and computation (ALU) separation in mind. In order to allow the identical cores to work together a simplified arbiter, memory management unit (MMU), and Read-Only-Memory (ROM) with firmware was needed. In the first stage of the FPGA design, the simplified MIPS instruction set was utilized to create a state diagram for the processor core. Then, the design for the controller, ALU control, datapath and ALU were created in Verilog using the state diagram in addition to the professors' diagrams and code. After the individual core was proven working the multicore design was created which included the arbiter, MMU, and ROM. In the next stage of the FPGA design, the functionality of the processor was verified with an external RAM module which encoded a Fibonacci program in addition to a randomized testbench for the processor's controller. In the Synthesis phase of the design, a simplified Input/Output (IO) module was created to allow the connection of the CPUs output with a seven segment display and the top file synthesized. Finally, the bitstream could be generated and the CPU tested on the BASYS 3 FPGA board.

II Architecture

Our processor implements 11 instructions of the R, I, and J type shown in *Table 1* below.

Instruction	Function	Type	Opcode	Func
add a,b,c	a = b+ c	R	000000	100000
sub a,b,c	a = b - c	R	000000	100010
and a,b,c	a = b && c	R	000000	100100
or a,b,c	a = b || c	R	000000	100101
slt a,b,c	a = 1 if b < c o.w: a= 0	R	000000	101010
addi a,b,#	a = b + #[15:0]	I	001000	n/a
beq a,b,addr	If a = b PC = PC + addr[15:0]	I	000100	n/a
j addr	PC = addr[25:0]	J	000010	n/a
lb a, offset(b)	a = mem[b+offset]	I	100000	n/a
sb a, offset(b)	Mem[b+offset] = a	I	110000	n/a
WRSPEC a, #	special_reg[#] = a	I	000001	n/a

Table 1: Processor Assembly Instructions

Each instruction is 32 bits long (4 bytes). For all instructions, the first 6 bits represent the opcode. All R-type instructions have an opcode of 0. This is followed by 5 bits for the source register (rs), 5 bits for the second register (rt), 5 bits for the destination register (rd), 5 bits for the shift amount (shamt) if applicable (0 otherwise), and 6 bits for the function code which specifies the specific R-format instruction. The R-type instructions we are implementing have no shift amount, just two operands and the destination register. I-type instructions begin with the opcode followed by 5 bits specifying a source register (rs), 5 bits for the destination register (rd), and 16 bits for an immediate value. Lasty, J instruction types have just the opcode followed by 26 bits for an address. This is summarized in *Table 2*.

Type	Format					
R	Opcode (6)	Rs (5)	Rt (5)	Rd (5)	Shamt (5)	Funct (6)
I	Opcode (6)	Rs (5)	Rt (5)	Immediate (16)		
J	Opcode (6)	Address (26)				

Table 2: MIPS Instruction Types

Our CPU was restricted to use 8-bit wide random-access-memory (RAM) and data paths meaning 4 fetch instructions (4 CPU cycles) are required to obtain the full 32 bit instruction. Then, depending on the instruction it is executed in 1-3 clock cycles. This is all handled by the Controller, a 16 state Finite State Machine (FSM). The state diagram detailing the operation of this FSM is shown below in *Figure 1*.

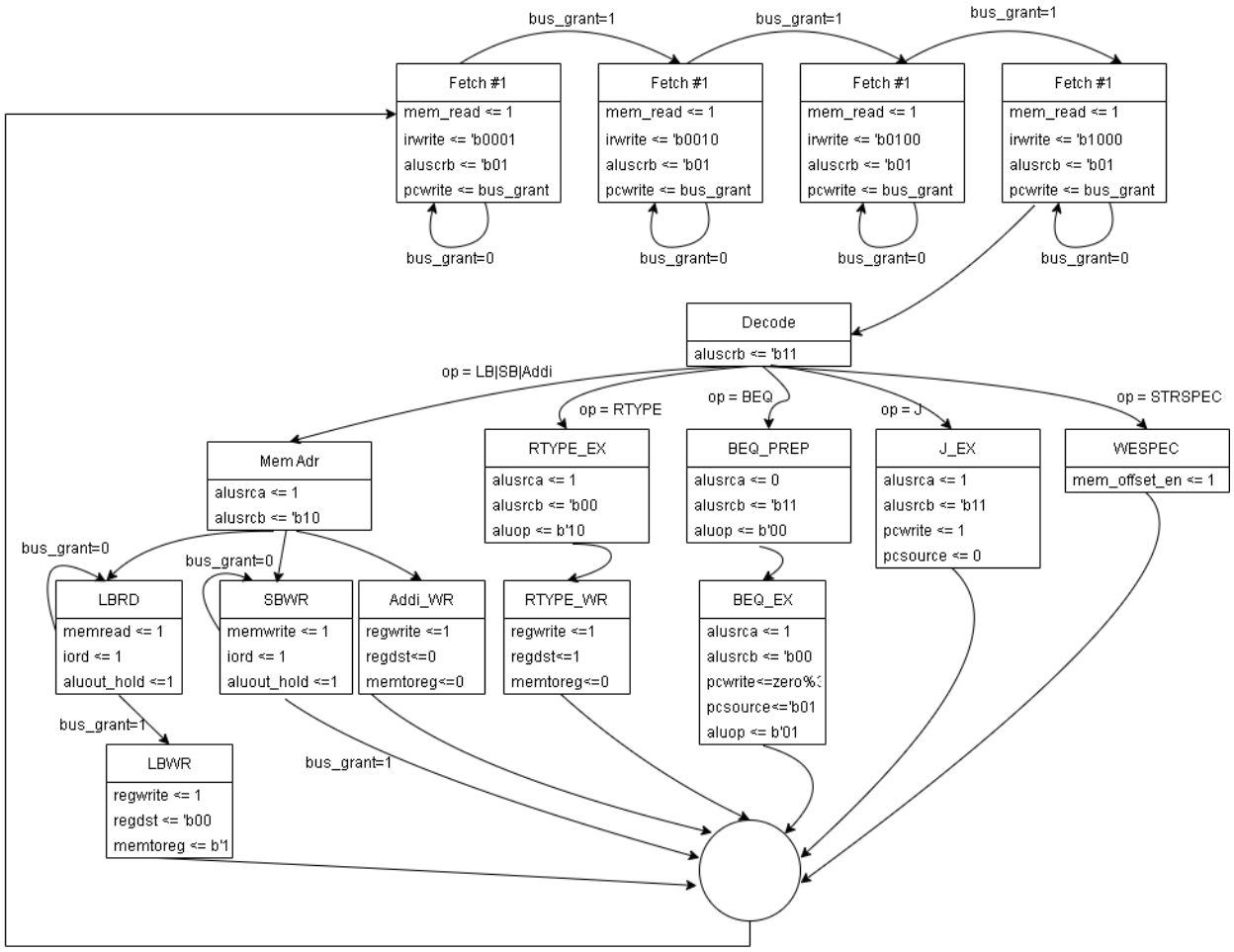


Figure 1: State Diagram for Controller FSM

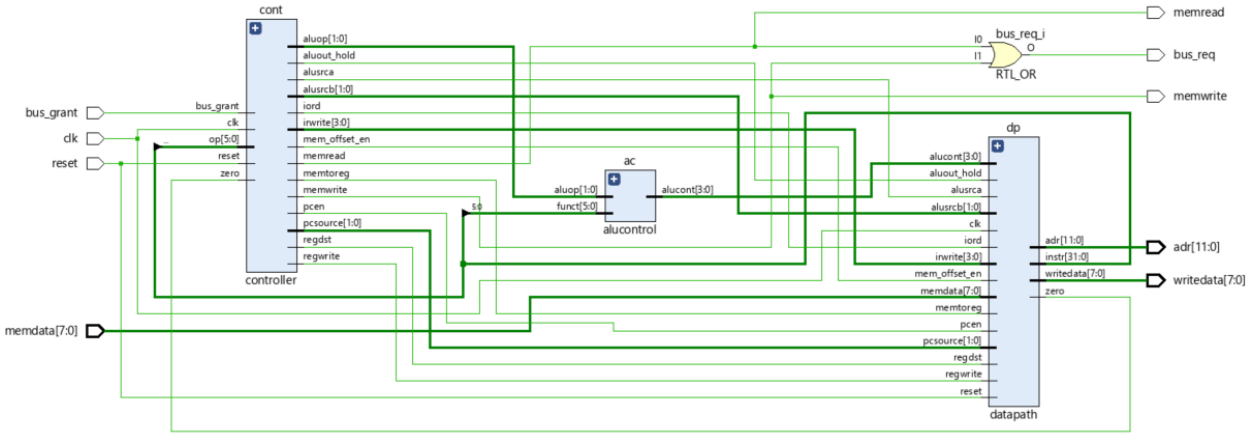
As shown in the state diagram of *Figure 1*, the first 4 states are fetch instructions. Additionally, a bus_grant flag is used to determine whether the Controller should move on to the next state or stall operation. This is needed so that only one core is granted access to the bus at a time. Once in the Decode state, the controller determines which state to move to based on the instructions opcode (op). For instance if the opcode is lb, sb, or addi, the next state is MemAdr which computes addresses used for lb read (LBRD), lb write (LBWR), and addi write (Addi_WR) states. If in the decode step the opcode is instead an R-type, the next state is RTYPE_EX which sets the a/b registers for the alu and sets the alu opcode (aluop) to go into its default state which will set the ALU using the specific funct of the instruction (last 6 bits of instruction). If instead the decoded instruction is a branch if equal (BEQ), the next stage is the BEQPREP which computes the end of the branch and stores it to the ALU output register. From here, the next state is BEQEX which performs the jump if the ALU zero flag is true. If the decoded instruction is a jump (J) then the next state is the jump execute (JEX) which sets the program counter to the lower 8 bits of the instruction. The last path decode can take is to the WESPEC state which happens if the opcode is decoded to WRSPEC. This is an additional instruction separate from the given MIPS instructions we were tasked to implement. The purpose of this is to store data from register A into a special register which in our case is the virtual memory register.

The controller sets the aluop and funct signals for the alu controller which then tells the ALU which operation to perform through the alucont signal. Since the ALU is just a multiplexer the alucont signal will select which operation (add, sub, and, or, set on less than) to perform. *Table 3* summarizes the operations the ALU can perform and the value the alu control signal will send to the ALU.

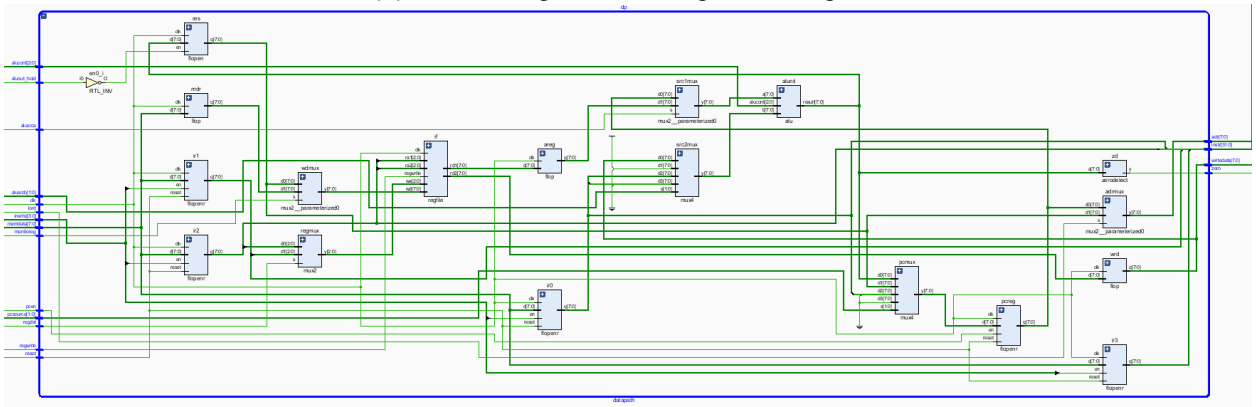
Opcode	Operation	ALU Control Output (ALU unit input)
Load / Store	Addition (memory address)	010
Branch	Subtraction (comparison)	110
Arithmetic (R-type)	Depends on <u>funct</u> field: 100000 add 100010 subtract 100100 and 100101 or 101010 set on less than	 010 110 000 001 111

Table 3: ALU Control Operation

These controller and alu control output signals are then sent to the data path which connects the ALU with the various registers (IR registers, mdr, mar, program counter (pc), register file, ect..). The overall design of a single core consisting of the controller, alu control, and datapath is summarized in the block diagram of *Figure 2*.



(a) Overall High-Level design of a Single Core



(b) Data Path Design

Figure 2: Block Diagram of a Single Core

Two copies of the individual core described above were used to create the architecture for the dual core cpu. In order to have two cores share access to the same memory, a simplified bus arbiter was designed. Each core has a bus_request signal which they set high whenever that core wants to read or write memory. Based on these two signals the bus arbiter sets one of the bus grant signals for the two cores high. The logic of the arbiter prioritizes allowing a core to finish multiple sequential memory accesses before transferring the bus so that cores aren't stuck part way through reading an instruction. If a core was not granted bus access and its controller is in a state that needs memory access then it will stall at that state as shown in the finite state machine of *Figure 1*. In addition to the arbiter, a simple memory management unit was built which splits the 1KB of RAM into 4 pages and allows each core to read or write to a particular space in RAM if that page is available. Then a memory map module was created to route the reads/writes to different devices (ROM, RAM, IO, or MMU) based on the address. The architecture of the complete dual-core MIPS CPU is summarized within the diagram of *Figure 3*.

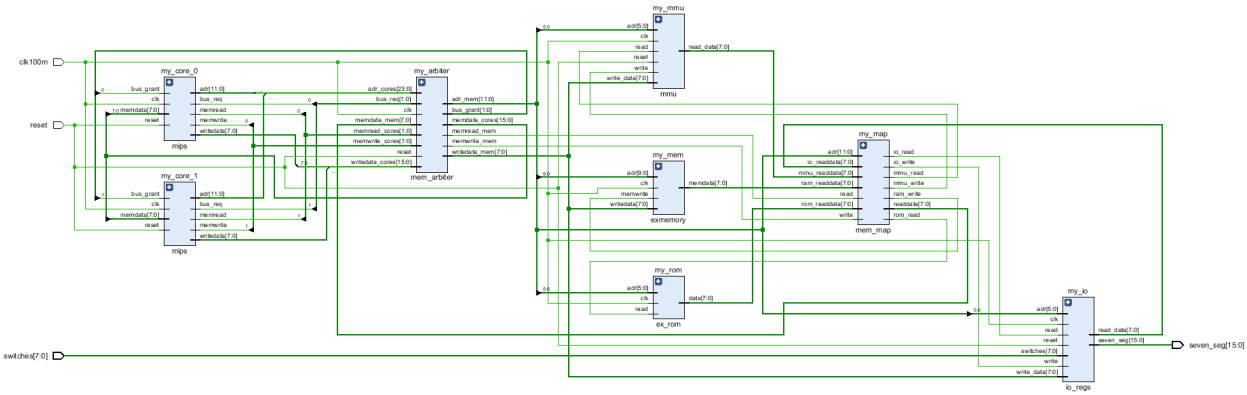


Figure 3: Dual-Core MIPS Cpu High-level Diagram

III Code

On the subsequent pages all code related to the CPU and its various sub modules and Testbench code will be supplied..

File name: Instructions.sv

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// ENEE 459D - Final Project
// Instruction.sv
// Contain classes to represent R, I , and J type instructions for Randomization
////////////////////////////////////////////////////////////////////////////////
package automatic Instructions;
    /*
    * Represents RType randomized instruction
    * And verifys that the CPU correctly performs the instruction
    */
    class RType;
        // RAND variables for RType instruction
        bit [5:0] opcode = 6'b000000; // RType opcode always = 0
        rand bit [4:0] sourceReg1; // 5 bit Rs register
        rand bit [4:0] sourceReg2; // 5 bit Rt register
        rand bit [4:0] destReg; // 5 bit Rd register
        bit [4:0] shamt = 5'b00000; // shamt always 0
        rand bit [5:0] funct; // 5 bits for funct
        bit [8*16-1:0] old_regs;

        // constrain RAND vars for registers based on the registers CPU has available
        constraint sourceReg1_constraint {sourceReg1 inside{[0:7]};}
        constraint sourceReg2_constraint {sourceReg2 inside{[0:7]};}
        constraint destReg_constraint {destReg inside{[1:7]};} //Don't store to the constant 0 register
        constraint funct_constraint {(funct == 6'b100000) || (funct == 6'b100010) || (funct == 6'b100100) ||
(funct == 6'b100101) || (funct == 6'b101010) || (funct == 6'b101011);}

        covergroup my_cg;

        // coverpoint for each possible funct of our instruction set
        funct_point: coverpoint funct {
            bins ADD_OP = {6'b100000};
            bins SUB_OP = {6'b100010};
            bins AND_OP = {6'b100100};
            bins OR_OP = {6'b100101};
            bins LSHIFT_OP = {6'b101010};
            bins RSHIFT_OP = {6'b101011};
        }
        endgroup
```

```
    // constructor
    function new();
        my_cg = new();
    endfunction

    // read the initial state of the registers
    function void read_initial_state(input [8*16-1:0] regs, input [11:0] pc);
        old_regs = regs;
    endfunction

    // based on funct check that dest register value is consistent with the instruction that was used
    // returns boolean (0 or 1)
    function integer check_final_state(input [8*16-1:0] regs, input [11:0] pc);
        // set  sourceReg1 & sourceReg2 to any value
        bit [14:0] destData = regs[destReg*16+:15]; //Ignore MSB because of overflow errors
        bit [14:0] srcData1 = old_regs[sourceReg1*16+:15];
        bit [14:0] srcData2 = old_regs[sourceReg2*16+:15];
        //$display("%b, %b, %b, %b", destData, srcData1, srcData2, srcData1 & srcData2);
        case(funct)
        6'b100000: return destData == (srcData1 + srcData2);
        6'b100010: return destData == (srcData1 - srcData2);
        6'b100100: return destData == (srcData1 & srcData2);
        6'b100101: return destData == (srcData1 | srcData2);
        6'b101010: return destData == (srcData1 < srcData2 ? 1 : 0);
        default: return 1;
        endcase
    endfunction;

    function void post_randomize();
        this.my_cg.sample();
    endfunction
endclass

/*
* Represents IType randomized instruction
* And verifys that the CPU correctly performs the instruction
*/
class IType;
    // random bits for each part of an IType instruction
    rand bit [5:0]opcode;
```

```
        rand bit [4:0] regA;
        rand bit [4:0] regB;
        rand bit [15:0]immediate;

        // constraint for the opcode so that it is indeed an IType
        constraint opcode_constraint {(opcode == 6'b001000) || (opcode == 6'b000100) || (opcode == 6'b100000)
|| (opcode == 6'b110000);}
        // contraint the 15th bit of immediate value to always be 0
        constraint immediate_constraint {(immediate[15] == 0);}
       // constraint registers to what is available to the CPU
        constraint regA_constraint {regA inside{[0:7]};}
        constraint regB_constraint {regB inside{[1:7]};} //Don't store to constant 0 register

        bit [8*16-1:0] old_regs; // used to store the inital state of registers
        bit [11:0] pc_old; // store the old program counter (needed to verify Jump instr)
        integer tmp;

        // covergroup for the opcodes and immediate
        covergroup my_cg;
        op: coverpoint opcode {
            bins BEQ = {6'b000100};
            bins ADDI = {6'b001000};
            bins LOAD = {6'b100000};
            bins STORE = {6'b110000};
        }
        imm: coverpoint immediate[7:0];
        endgroup

        function new();
            my_cg = new();
        endfunction // constructor

        // called after randomization
        function void post_randomize();
            this.my_cg.sample();
        endfunction

        // store initial values of registers and program counter
        function void read_initial_state(input [8*16-1:0] regs, input [11:0] pc);
            old_regs = regs;
            pc_old = pc;
```

```
    endfunction

    // return boolean True or false (1 or 0) if the final of regs/pc state matches up to
    // expected final state
    function integer check_final_state(input [8*16-1:0] regs, input [11:0] pc);
    bit [14:0] dataB = regs[16*regB+:15];
    bit [14:0] dataA = old_regs[16*regA+:15];

    tmp = pc_old + immediate[9:0]*4 + 12'd4;
    case(opcode)
        6'b001000: return dataB == (dataA + immediate[14:0]);
        6'b000100: return (dataA == dataB) ? (pc == tmp[11:0]) : (pc == pc_old + 12'd4);
        default: return 1;
    endcase
    endfunction;
endclass

/*
* Represents a randomized JType instruction
*/
class JType;

    bit [5:0]opcode = 6'b000010; // only opcode for jump instruction
    rand bit [25:0]address;
    //constraint address_constraint {}
    constraint address_constraint {(address[25:20] == 6'b0);}

    covergroup my_cg;
    addr: coverpoint address[7:0];
    endgroup

    function new();
        my_cg = new();
    endfunction

    function void post_randomize();
        this.my_cg.sample();
    endfunction

    // for jump initial states aren't necessary
    function void read_initial_state(input [8*16-1:0] regs, input [11:0] pc);
```

```
    endfunction

    // verify CPU pc is equal to the given address of the jump instruction
    function integer check_final_state(input [8*16-1:0] regs, input [11:0] pc);
        return pc == {address[9:0], 2'b00};
    endfunction;
endclass

/*
* Used to represent a rand instruction type
* 0-> RType, 1-> IType 2 -> JType
*/
class InstructionType;
    rand integer instructionType;
    constraint instructionType_constraint {instructionType inside {[0:2]};}
endclass;

/*
* Represents Randomized RAM
* In our case RAM only large enough to contain 1 instruction
*/
class RandomizedRAM;
    //rand integer numInstructions;
    rand bit [31:0] RAM; // can I make this dynamic and set the size = numInstructions?
    //constraint numInstructions_constraint {numInstructions inside {[10:1]};}

    //integer currentInstruction = 0;
    RType r_instr;
    IType i_instr;
    JType j_instr;
    InstructionType instructionType;

    function new();
        r_instr = new();
        i_instr = new();
        j_instr = new();
        instructionType = new();
    endfunction // constructor

    function void post_randomize();
        instructionType.randomize();
```

```
            case(instructionType.instructionType)
                0: begin
                    // generate R Type instruction
                    r_instr.randomize(); // this will call the post_randomize of R Type
                    RAM ={r_instr.opcode, r_instr.sourceReg1,r_instr.sourceReg2, r_instr.destReg,
r_instr.shamt, r_instr.funct};
                end
                1: begin
                    // generate I Type instruction

                    i_instr.randomize(); // this will call the post_randomize of I Type
                    RAM ={i_instr.opcode,i_instr.regA,i_instr.regB, i_instr.immediate};
                end
                2: begin
                    // generate J Type instruction
                    j_instr.randomize(); // this will call the post_randomize of R Type
                    RAM ={j_instr.opcode,j_instr.address};
                end
            endcase
        endfunction

        // store the initial states of reg/pc depending on the instr type
        function void read_initial_state(input [8*16-1:0] regs, input [11:0] pc);
            case(instructionType.instructionType)
                0: r_instr.read_initial_state(regs, pc);
                1: i_instr.read_initial_state(regs, pc);
                2: j_instr.read_initial_state(regs, pc);
            endcase
        endfunction

        // check the final state and return whether instr was successfully run
        function integer check_final_state(input [8*16-1:0] regs, input [11:0] pc);
            case(instructionType.instructionType)
                0: return r_instr.check_final_state(regs, pc);
                1: return i_instr.check_final_state(regs, pc);
                2: return j_instr.check_final_state(regs, pc);
            endcase
        endfunction
    endclass
endpackage
```

Filename: coverage_testbench.sv
Module: controller_testbench

```
////////////////////////////////////////////////////////////////////////////////
// ENEE 459D - Final Project
// Instruction.sv
// Contain classes to represent R, I , and J type instructions for Randomization
////////////////////////////////////////////////////////////////////////////////
timescale 1ns / 1ps
import Instructions::*;

module controller_testbench();

    reg clk; // clk for CPU
    reg reset; // reset for CPU
    wire [7:0] writedata;
    wire [11:0] adr;
    wire memread, memwrite;
    reg [31:0]data_reg;
    reg [7:0] memdata;
    wire [16*8-1:0] internal_regs; // wire to access registers for

    // MIPS CPU core
    mips #(.WIDTH(16), .REGBITS(3)) my_core(
        .clk(clk),
        .reset(reset),
        .memdata(memdata),
        .memread(memread),
        .memwrite(memwrite),
        .adr(adr),
        .writedata(writedata),
        .bus_grant(1'b1)
    );

    // assign each register from CPU register file to internal_regs
    generate
        for(genvar j=0; j<8; j=j+1) begin
            assign internal_regs[16*j+:16] = my_core.dp.rf.RAM[j];
        end
    endgenerate

    // task that runs the test
    task test();
        RandomizedRAM randRAM;
```

```
    randRAM = new();

    // run 1000 random instructions
    for(integer i=0; i<10000; i=i+1) begin
        randRAM.randomize();
        randRAM.read_initial_state(internal_regs, my_core.dp.pcreg.q); // save initial state

        data_reg = randRAM.RAM;
        memdata = data_reg[7:0];

        // fetch the instruction
        #1;
        clk <= 1; #1;
        clk <= 0; #1;
        memdata <= data_reg[15:8];
        clk <= 1; #1;
        clk <= 0; #1;
        memdata <= data_reg[23:16];
        clk <= 1; #1;
        clk <= 0; #1;
        memdata <= data_reg[31:24];
        clk <= 1; #1;
        clk <= 0; #1;

        // continue running control until back in FETCH#1 state
        while(mips.cont.state != 5'b00001) begin
            clk <= 1; #1;
            clk <= 0; #1;
        end

        // display message if the final state of the instruction is not as expected
        if(randRAM.check_final_state(internal_regs, my_core.dp.pcreg.q) != 1) begin
            $display("%b", data_reg);
            $finish();
        end
    end // loop through 1000 instructions

    // display coverage statistics
    $display("R instruction funct Coverage: %d",randRAM.r_instr.my_cg.funct_point.get_inst_coverage());
    $display("I instruction operand Coverage: %d",randRAM.i_instr.my_cg.op.get_inst_coverage());
    $display("I instruction Immediate Coverage: %d",randRAM.i_instr.my_cg.imm.get_inst_coverage());
    $display("J instruction Address Coverage: %d",randRAM.j_instr.my_cg.addr.get_inst_coverage());
endtask
```

```
    // 'main' program
    initial begin
    //reset clock and cpu and tick clock a few times
        clk <= 0;
        reset <= 1;
        #1;
        clk <= 1;
        #1;
        clk <= 0;
        #1;
        clk <= 1;
        #1;
        clk <= 0;
        reset <= 0;
        #1;
        for(integer i=0; i<8; i=i+1) my_core.dp.rf.RAM[i] <= 16'b0;//clear registers
        test(); // runs the test
    end
endmodule
```

Filename: mmu.sv
Module: mmu

```
`timescale 1ns / 1ps

module mmu(
    input clk,
    input reset,
    input read,
    input write,
    input [5:0] adr, //We never actually use the address, so any address in the range will have the same
effect
    output reg [7:0] read_data,
    input [7:0] write_data
    );

    reg [3:0] avail;

    always @(*) begin
        if(avail[0]) begin
            read_data <= 8'h4; //The read data will always be the next open page
        end else if(avail[1]) begin
            read_data <= 8'h5;
        end else if(avail[2]) begin
            read_data <= 8'h6;
        end else if(avail[3]) begin
            read_data <= 8'h7;
        end else begin
            read_data <= 8'h0;
        end
    end

    always @(posedge clk) begin
        if(reset) avail<= 4'b1111;
        else begin
            if(read) begin
                if(avail[0]) begin
                    avail[0] <= 0; //mark the page as unavailable after it is returned
                end else if(avail[1]) begin
                    avail[1] <= 0;
                end else if(avail[2]) begin
                    avail[2] <= 0;
                end else if(avail[3]) begin
                    avail[3] <= 0;
```

```
                end
            end
            if(write) begin
                avail[write_data] <= 1; //Pages can be freed by writing
            end
        end
    end

endmodule
```

Filename: ex_rom.sv
Module: ex_rom

```
`timescale 1ns / 1ps

module ex_rom(
    input clk,
    input [5:0] adr,
    input read,
    output reg [7:0] data
    );

    reg [31:0] ROM [15:0];
    wire [31:0] word;
    assign word = ROM[adr >> 2];

    always @(*) begin //This is just the RAM but without the ability to write
        case(adr[1:0])
            2'b00: data <= word[7:0];
            2'b01: data <= word[15:8];
            2'b10: data <= word[23:16];
            2'b11: data <= word[31:24];
        endcase
    end

    initial begin
        //ROM[0] <= 32'b001000_00000_00001_0000000000000100;
        ROM[0] <= 32'b100000_00000_00001_0000000010000000;
        ROM[1] <= 32'b000001_00001_00000_00000_00000000000;
        ROM[2] <= 32'b000010_00000_00000_00000010_00000000;
    end
endmodule
```

Filename: exmemory.sv
Module: exmemory

```
`timescale 1ns / 1ps

module exmemory #(parameter WIDTH=8, parameter ADDR_WIDTH=10) (
    input clk,
    input memwrite,
    input [ADDR_WIDTH-1 : 0] adr,
    input [WIDTH-1 : 0] writedata,
    output reg [WIDTH-1 : 0] memdata
    );

    reg [31:0] RAM [(1<<(ADDR_WIDTH-2))-1 : 0];
    wire [31:0] word;
    assign word = RAM[adr >> 2];

    initial begin
        //$readmemh("memfile.dat", RAM);
        /*RAM[0] = 32'b001000_00000_00001_0000000000000001;
        RAM[1] = 32'b001000_00000_00010_0000000000000001;
        RAM[2] = 32'b000000_00001_00010_00011_00000_100000;
        RAM[3] = 32'b000000_00011_00010_00010_00000_100000;
        RAM[4] = 32'b000000_00011_00010_00011_00000_100000;
        RAM[5] = 32'b000000_00010_00000_00010_00000_110000;
        RAM[6] = 32'b110000_00000_00011_0000000010000000;
        RAM[7] = 32'b000010_00000_0000000000010_00000011;*/

        RAM[0] <= 32'b100000_00000_00001_0000000001000000;
        RAM[1] <= 32'b001000_00000_00010_0000000000000001;
        RAM[2] <= 32'b001000_00000_00011_0000000000000001;
        RAM[3] <= 32'b001000_00000_00100_0000000000000001;
        RAM[4] <= 32'b001000_00000_00101_0000000000000010;
        RAM[5] <= 32'b000100_00001_00101_0000000000000101;
        RAM[6] <= 32'b000100_00001_00100_0000000000000101;
        RAM[7] <= 32'b000000_00001_00101_00001_00000_100010;
        RAM[8] <= 32'b000000_00010_00011_00010_00000_100000;
        RAM[9] <= 32'b000000_00010_00011_00011_00000_100000;
        RAM[10] <= 32'b000010_00000_0000000000010_00000101;
        RAM[11] <= 32'b000000_00010_00011_00011_00000_100000;
        RAM[12] <= 32'b110000_00000_00011_0000000001000000;
        RAM[13] <= 32'b000000_00011_00000_00011_00000_110001;
        RAM[14] <= 32'b110000_00000_00011_0000000001000001;
        RAM[15] <= 32'b000010_00000_0000000000010_00001111;
```

```
        RAM[64] <= 32'b001000_00000_00001_0000000000000001;
        RAM[65] <= 32'b001000_00000_00010_0000000000000001;
        RAM[66] <= 32'b001000_00000_00100_0000000000001000;
        RAM[67] <= 32'b000000_00001_00010_00011_00000_100000;
        RAM[68] <= 32'b000000_00011_00010_00010_00000_100000;
        RAM[69] <= 32'b000000_00011_00010_00011_00000_100000;
        RAM[70] <= 32'b000000_00011_00100_00101_00000_100010;
        RAM[71] <= 32'b000100_00010_00100_0000000000000001;
        RAM[72] <= 32'b000010_00000_0000000000010_00000100;
        RAM[73] <= 32'b001000_00000_00111_00000010_00000001;
        RAM[74] <= 32'b000010_00000_0000000000010_00001010;
    end
    always @(posedge clk) begin
        if(memwrite) begin
            case(adr[1:0])
                2'b00: RAM[adr >> 2][7:0] <= writedata;
                2'b01: RAM[adr >> 2][15:8] <= writedata;
                2'b10: RAM[adr >> 2][23:16] <= writedata;
                2'b11: RAM[adr >> 2][31:24] <= writedata;
            endcase
        end
    end
    always @(*) begin
        case(adr[1:0])
            2'b00: memdata <= word[7:0];
            2'b01: memdata <= word[15:8];
            2'b10: memdata <= word[23:16];
            2'b11: memdata <= word[31:24];
        endcase
    end
endmodule
```

Filename: io_regs.sv
Module: io_regs

```
`timescale 1ns / 1ps

module io_regs(
    input clk,
    input reset,
    input [5:0] adr,
    input read,
    input write,
    output [7:0] read_data,
    input [7:0] write_data,

    input [7:0] switches,
    output reg [15:0] seven_seg,
    output reg [7:0] leds
    );
    assign read_data = switches; //no matter what address you read from, you always get the switches
    always @(posedge clk) begin
        if(reset) begin
            seven_seg <= 0; //be sure to reset the output to 0
        end else if(write) begin
            case(adr[1:0]) //check the address and use that to determine which output to write
                2'b00: seven_seg[7:0] <= write_data;
                2'b01: seven_seg[15:8] <= write_data;
                2'b10: leds <= write_data;
                2'b11: leds <= write_data;
            endcase
        end
    end
endmodule
```

Filename: controller.v
Module: controller

```
`timescale 1ns / 1ps


module controller(
    input clk,
    input reset,
    input [5:0] op,
    input zero,
    output reg memread,
    output reg memwrite,
    output reg alusrca,
    output reg memtoreg,
    output reg iord,
    output pcen,
    output reg regwrite,
    output reg regdst,
    output reg [1:0] pcsource,
    output reg [1:0] alusrcb,
    output reg [1:0] aluop,
    output reg [3:0] irwrite,

    output reg aluout_hold,
    output reg mem_offset_en,

    input bus_grant
    );

    localparam FETCH1 = 5'b00001;
    localparam FETCH2 = 5'b00010;
    localparam FETCH3 = 5'b00011;
    localparam FETCH4 = 5'b00100;
    localparam DECODE = 5'b00101;
    localparam MEMADR = 5'b00110;
    localparam LBRD = 5'b00111;
    localparam LBWR = 5'b01000;
    localparam SBWR = 5'b01001;
    localparam RTYPEEX = 5'b01010;
    localparam RTYPEWR = 5'b01011;
    localparam BEQPREP = 5'b01100;
    localparam BEQEX = 5'b01101;
    localparam JEX = 5'b01110;
```

```
    localparam ADDIWR = 5'b01111;
    localparam WRSPEC = 5'b10000; //New state to write to special register

    localparam LB = 6'b100000;
    localparam SB = 6'b110000;
    localparam RTYPE = 6'b000000;
    localparam BEQ = 6'b000100;
    localparam J = 6'b000010;
    localparam ADDI = 6'b001000;
    localparam STRSPEC = 6'b000001; // New instruction op-code

    reg [4:0] state, nextstate;
    reg pcwrite, pcwritecond;
    assign pcen = pcwrite | (pcwritecond & zero);

    always @(posedge clk) begin
        if(reset) state <= FETCH1;
        else state <= nextstate;
    end
    always @(*) begin
        case(state)
            FETCH1: nextstate <= bus_grant ? FETCH2 : FETCH1; //If we don't have the bus, then the data from
the memory isn't actually our data, so don't move to the next state
            FETCH2: nextstate <= bus_grant ? FETCH3 : FETCH2;
            FETCH3: nextstate <= bus_grant ? FETCH4 : FETCH3;
            FETCH4: nextstate <= bus_grant ? DECODE : FETCH4;
            DECODE: case(op)
                LB: nextstate <= MEMADR;
                SB: nextstate <= MEMADR;
                ADDI: nextstate <= MEMADR;
                RTYPE: nextstate <= RTYPEEX;
                BEQ: nextstate <= BEQPREP;
                J: nextstate <= JEX;
                STRSPEC: nextstate <= WRSPEC;
                default: nextstate <= FETCH1;
            endcase
            MEMADR: case(op)
                LB: nextstate <= LBRD;
                SB: nextstate <= SBWR;
                ADDI: nextstate <= ADDIWR;
                default: nextstate <= FETCH1;
            endcase
            LBRD: nextstate <= bus_grant ? LBWR : LBRD;
```

```
    LBWR: nextstate <= FETCH1;
    SBWR: nextstate <= bus_grant ? FETCH1 : SBWR;
    RTYPEEX: nextstate <= RTYPEWR;
    RTYPEWR: nextstate <= FETCH1;
    BEQPREP: nextstate <= BEQEX; // I had to add an extra state for BEQ
    BEQEX: nextstate <= FETCH1;
    JEX: nextstate <= FETCH1; // Unconditional jump state
    ADDIWR: nextstate <= FETCH1;
    WRSPEC: nextstate <= FETCH1;
    default: nextstate <= FETCH1;
endcase
irwrite <= 4'b0000;
pcwrite <= 0;
pcwritecond <= 0;
regwrite <= 0;
regdst <= 0;
memread <= 0;
memwrite <= 0;
alusrca <= 0;
alusrcb <= 2'b00;
aluop <= 2'b00;
pcsource <= 2'b00;
iord <= 0;
memtoreg <= 0;
aluout_hold <= 0;
mem_offset_en <= 0;
case(state)
    FETCH1: begin
        memread <= 1;
        irwrite <= 4'b0001;
        alusrcb <= 2'b01;
        pcwrite <= bus_grant; //Don't increment the PC unless we are actually leaving this state
    end
    FETCH2: begin
        memread <= 1;
        irwrite <= 4'b0010;
        alusrcb <= 2'b01;
        pcwrite <= bus_grant;
    end
    FETCH3: begin
        memread <= 1;
        irwrite <= 4'b0100;
        alusrcb <= 2'b01;
```

```
    pcwrite <= bus_grant;
end
FETCH4: begin
    memread <= 1;
    irwrite <= 4'b1000;
    alusrcb <= 2'b01;
    pcwrite <= bus_grant;
end
DECODE: begin
    alusrcb <= 2'b11;
end
MEMADR: begin
    alusrca <= 1;
    alusrcb <= 2'b10;
end
LBRD: begin
    memread <= 1;
    iord <= 1;
    aluout_hold <= 1;
end
LBWR: begin
    regwrite <= 1;
    regdst <= 0;
    memtoreg <= 1;
end
SBWR: begin
    memwrite <= 1;
    iord <= 1;
    aluout_hold <= 1;
end
RTYPEEX: begin
    alusrca <= 1;
    alusrcb <= 2'b00;
    aluop <= 2'b10; //go to default in alu controller
end
RTYPEWR: begin
    regwrite <= 1;
    regdst <= 1;
    memtoreg <= 0;
end
BEQPREP: begin
    alusrca <= 0;
    alusrcb <= 2'b11;
```

```
                aluop <= 2'b00;
            end
            BEQEX: begin
                alusrca <= 1;
                alusrcb <= 2'b00;
                pcwrite <= zero;
                pcsource <= 2'b01;
                aluop <= 2'b01; //Subtraction mode
            end
            JEX: begin
                alusrca <= 1; //bits 25-21 MUST be zeros
                alusrcb <= 2'b11;
                pcwrite <= 1;
                pcsource <= 0;
            end
            ADDIWR: begin
                regwrite <= 1;
                regdst <= 0;
                memtoreg <= 0;
            end
            WRSPEC: begin
                mem_offset_en <= 1;
            end
            default: begin
            end
        endcase
    end
endmodule
```

Filename: datapath.v
Module: datapath

```
`timescale 1ns / 1ps

module datapath #(parameter WIDTH = 8, REGBITS = 3) (
    input clk,
    input reset,
    input [8-1 : 0] memdata,
    input alusrca,
    input memtoreg,
    input iord,
    input pcen,
    input regwrite,
    input regdst,
    input [1:0] pcsource,
    input [1:0] alusrcb,
    input [3:0] irwrite,
    input [3:0] alucont,
    input aluout_hold,
    output zero,
    output [31:0] instr,
    output [11 : 0] adr,
    output [8-1 : 0] writedata,

    input mem_offset_en
    );

    localparam CONST_ZERO = {WIDTH{1'b0}};
    localparam CONST_ONE = CONST_ZERO + 1'b1;

    assign writedata = writedata_int[7:0];
    wire [11:0] adr_int;

    wire [REGBITS-1 : 0] ra1, ra2, wa;
    wire [WIDTH-1 : 0] rd1, rd2, wd, a, src1, src2, aluresult, aluout, constx4, writedata_int;
    wire [11:0] pc, nextpc;
    wire [7:0] md;

    wire [3:0] mem_offset;

    assign constx4 = {instr[WIDTH-3:0], 2'b00};
    assign ra1 = instr[REGBITS + 20:21];
    assign ra2 = instr[REGBITS + 15:16];
```

```
mux2 #(REGBITS) regmux(
    .d0(instr[REGBITS + 15:16]),
    .d1(instr[REGBITS + 10:11]),
    .s(regdst),
    .y(wa)
);
flopenr #(8) ir0(
    .clk(clk),
    .reset(reset),
    .en(irwrite[0]),
    .d(memdata[7:0]),
    .q(instr[7:0])
);
flopenr #(8) ir1(
    .clk(clk),
    .reset(reset),
    .en(irwrite[1]),
    .d(memdata[7:0]),
    .q(instr[15:8])
);
flopenr #(8) ir2(
    .clk(clk),
    .reset(reset),
    .en(irwrite[2]),
    .d(memdata[7:0]),
    .q(instr[23:16])
);
flopenr #(8) ir3(
    .clk(clk),
    .reset(reset),
    .en(irwrite[3]),
    .d(memdata[7:0]),
    .q(instr[31:24])
);
flopenr #(12) pcreg(
    .clk(clk),
    .reset(reset),
    .en(pcen),
    .d(nextpc),
    .q(pc)
);
flopenr #(4) mem_offset_reg( //Special register for memory offset
    .clk(clk),
```

```
        .reset(reset),
        .en(mem_offset_en), //new controller output signal from WRSPEC state
        .d(a),
        .q(mem_offset)
    );
    flop #(8) mdr(
        .clk(clk),
        .d(memdata),
        .q(md)
    );
    flop #(WIDTH) areg(
        .clk(clk),
        .d(rd1),
        .q(a)
    );
    flop #(WIDTH) wrd(
        .clk(clk),
        .d(rd2),
        .q(writedata_int)
    );
    flopen #(WIDTH) res(
        .clk(clk),
        .en(~aluout_hold), //Had to change this from flop to flopen so that this signal could be added because
we need to be able to stall while waiting to read or write RAM
        .d(aluresult),
        .q(aluout)
    );
    mux2 #(WIDTH) adrmux(
        .d0(pc),
        .d1(aluout),
        .s(iord),
        .y(adr_int)
    );
    mux2 #(WIDTH) src1mux(
        .d0({4'b0000, pc}),
        .d1(a),
        .s(alusrca),
        .y(src1)
    );
    mux4 #(WIDTH) src2mux(
        .d0(writedata_int),
        .d1(CONST_ONE),
        .d2(instr[WIDTH-1:0]),
```

```
        .d3(constx4),
        .s(alusrcb),
        .y(src2)
    );
    mux4 #(WIDTH) pcmux(
        .d0(aluresult),
        .d1(aluout),
        .d2(constx4),
        .d3(CONST_ZERO),
        .s(pcsource),
        .y(nextpc)
    );
    mux2 #(WIDTH) wdmux(
        .d0(aluout),
        .d1({8'b0, md}),
        .s(memtoreg),
        .y(wd)
    );
    regfile #(WIDTH,REGBITS) rf(
        .clk(clk),
        .regwrite(regwrite),
        .ra1(ra1),
        .ra2(ra2),
        .wa(wa),
        .rd1(rd1),
        .rd2(rd2),
        .wd(wd)
    );
    alu #(WIDTH) alunit(
        .a(src1),
        .b(src2),
        .alucont(alucont),
        .result(aluresult)
    );
    zerodetect #(WIDTH) zd(
        .a(aluresult),
        .y(zero)
    );

    assign adr = adr_int[11] ? {mem_offset, adr_int[7:0]} : {4'b0000, adr_int[7:0]}; // Virtual memory
remapping if the address is 0x800 or greater, otherwise access the lowest 256 bytes where the physical devices
are
endmodule
```

Filename: datapath.v
Module: datapath

```
`timescale 1ns / 1ps

module alucontrol(
    input [1:0] aluop,
    input [5:0] funct,
    output reg [3:0] alucont //Had to make alucont wider to accomodate extra arithmetic options
    );

    always @(*) begin
        case(aluop)
            2'b00: alucont <= 4'b0010;
            2'b01: alucont <= 4'b1010;
            default: case(funct)
                6'b100000: alucont <= 4'b0010;
                6'b100010: alucont <= 4'b1010;
                6'b100100: alucont <= 4'b0000;
                6'b100101: alucont <= 4'b0001;
                6'b101010: alucont <= 4'b1011;
                6'b110000: alucont <= 4'b0100;
                6'b110001: alucont <= 4'b0101;
                default: alucont <= 4'b1001;
            endcase
        endcase
    end
endmodule
```

Filename: mux2.v
Module: mux2

```
`timescale 1ns / 1ps

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

Filename: mux4.v
Module: mux4

```
`timescale 1ns / 1ps

module mux4 #(parameter WIDTH = 8) (
    input [WIDTH-1 : 0] d0,
    input [WIDTH-1 : 0] d1,
    input [WIDTH-1 : 0] d2,
    input [WIDTH-1 : 0] d3,
    input [1:0] s,
    output reg [WIDTH-1 : 0] y
    );
    always @(*) begin
        case(s)
            2'b00: y <= d0;
            2'b01: y <= d1;
            2'b10: y <= d2;
            2'b11: y <= d3;
        endcase
    end
endmodule
```

Filename: regfile.v
Module: regfile

```
`timescale 1ns / 1ps

module regfile #(parameter WIDTH = 8, REGBITS = 3) (
    input clk,
    input regwrite,
    input [REGBITS-1 : 0] ra1,
    input [REGBITS-1 : 0] ra2,
    input [REGBITS-1 : 0] wa,
    input [WIDTH-1 : 0] wd,
    output [WIDTH-1 : 0] rd1,
    output [WIDTH-1 : 0] rd2
    );
    reg [WIDTH-1:1] RAM [(1<<REGBITS)-1:0]; // There is no register 0
    always @(posedge clk) begin
        if (regwrite && wa) RAM[wa] <= wd; // Don't try to write to register 0
    end
    assign rd1 = ra1 ? RAM[ra1] : 0;
    assign rd2 = ra2 ? RAM[ra2] : 0;

endmodule
```

Filename: alu.v
Module: alu

```
`timescale 1ns / 1ps

module alu #(parameter WIDTH = 8) (
    input [WIDTH-1 : 0] a,
    input [WIDTH-1 : 0] b,
    input [3:0] alucont,
    output reg [WIDTH-1 : 0] result
    );
    wire [WIDTH-1 : 0] b2, sum, slt;
    assign b2 = alucont[3] ? ~b : b;
    assign sum = a + b2 + alucont[3];
    assign slt = sum[WIDTH-1];

    always @(*) begin
        case(alucont[2:0])
            3'b000: result <= a&b;
            3'b001: result <= a|b;
            3'b010: result <= sum;
            3'b011: result <= slt;
            3'b100: result <= a << 8; //new arithmetic option to allow for accessing high bits in registers,
needed to store more than 1 byte numbers to the output registers
            3'b101: result <= a >> 8;
            default: result <= 0;
        endcase
    end
endmodule
```

Filename: flop.v
Module: flop

```
`timescale 1ns / 1ps

module flop #(parameter WIDTH = 8) (
    input clk,
    input [WIDTH-1 : 0] d,
    output reg [WIDTH-1 : 0] q
    );
    always @(posedge clk) begin
        q <= d;
    end
endmodule
```

Filename: flopen.v
Module: flopen

```
`timescale 1ns / 1ps

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

Filename: flopenr.v
Module: flopenr

```
`timescale 1ns / 1ps

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

Filename: zerodetect.v
Module: zerodetect

```
`timescale 1ns / 1ps

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

Filename: mips.v
Module: mips

```
`timescale 1ns / 1ps

module mips #(parameter WIDTH = 8, REGBITS = 3) ( //Single core design
    input clk,
    input reset,
    input [8-1 : 0] memdata,
    output memread,
    output memwrite,
    output [11 : 0] adr,
    output [8-1 : 0] writedata,
    output bus_req,
    input bus_grant
    );
    assign bus_req = memread | memwrite;

    wire [31:0] instr;
    wire zero, alusrca, memtoreg, iord, pcen, regwrite, regdst, mem_offset_en;
    wire [1:0] aluop, pcsource, alusrcb;
    wire [3:0] alucont;
    wire [3:0] irwrite;
    wire aluout_hold;

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

```
        .mem_offset_en(mem_offset_en)
    );
    alucontrol ac(
        .aluop(aluop),
        .funct(instr[5:0]),
        .alucont(alucont)
    );
    datapath #(.WIDTH(WIDTH), .REGBITS(REGBITS)) dp(
        .clk(clk),
        .reset(reset),
        .memtoreg(memtoreg),
        .iord(iord),
        .pcen(pcen),
        .regwrite(regwrite),
        .regdst(regdst),
        .pcsource(pcsource),
        .alusrca(alusrca),
        .alusrcb(alusrcb),
        .irwrite(irwrite),
        .memdata(memdata),
        .zero(zero),
        .alucont(alucont),
        .instr(instr),
        .writedata(writedata),
        .adr(adr),
        .aluout_hold(aluout_hold),
        .mem_offset_en(mem_offset_en)
    );
endmodule
```

Filename: mem_bus.v
Module: mem_arbiter, mem_map

```
`timescale 1ns / 1ps

module mem_arbiter(
    input clk,
    input reset,
    output [15:0] memdata_cores,
    input [7:0] memdata_mem,
    input [23:0] adr_cores,
    output [11:0] adr_mem,
    input [15:0] writedata_cores,
    output [7:0] writedata_mem,
    input [1:0] memread_cores,
    output memread_mem,
    input [1:0] memwrite_cores,
    output memwrite_mem,
    input [1:0] bus_req,
    output [1:0] bus_grant
    );
    localparam NO_GRANT = 2'b00;
    localparam CORE_0_GRANT = 2'b01;
    localparam CORE_1_GRANT = 2'b10;

    reg [1:0] state, next_state;
    always @(posedge clk) begin
        if (reset) state <= 0;
        else state <= next_state;
    end

    assign bus_grant[0] = state == CORE_0_GRANT;
    assign bus_grant[1] = state == CORE_1_GRANT;

    always @(*) begin
        case(state)
            NO_GRANT: case(bus_req)
                2'b00: next_state <= NO_GRANT;
                2'b01: next_state <= CORE_0_GRANT;
                2'b10: next_state <= CORE_1_GRANT;
                2'b11: next_state <= CORE_0_GRANT; //Core 0 always wins initially
            endcase
            CORE_0_GRANT: case(bus_req)
                2'b00: next_state <= NO_GRANT;
```

```
            2'b01: next_state <= CORE_0_GRANT;
            2'b10: next_state <= CORE_1_GRANT;
            2'b11: next_state <= CORE_0_GRANT; //Let core 0 keep control if it has control
        endcase
        CORE_1_GRANT: case(bus_req)
            2'b00: next_state <= NO_GRANT;
            2'b01: next_state <= CORE_0_GRANT;
            2'b10: next_state <= CORE_1_GRANT;
            2'b11: next_state <= CORE_1_GRANT; //Let core 1 keep control if it has control
        endcase
        default: next_state <= NO_GRANT;
    endcase
end

assign memdata_cores = {2{memdata_mem}};
mux4 #(12) adr_mux(
    .d0(12'b0),
    .d1({1'b0, adr_cores[11:0]}),
    .d2({1'b1, adr_cores[23:12]}),
    .d3(12'b0),
    .s(state),
    .y(adr_mem)
);
mux4 #(8) writedata_mux(
    .d0(8'b0),
    .d1(writedata_cores[7:0]),
    .d2(writedata_cores[15:8]),
    .d3(8'b0),
    .s(state),
    .y(writedata_mem)
);
mux4 #(1) memread_mux(
    .d0(1'b0),
    .d1(memread_cores[0]),
    .d2(memread_cores[1]),
    .d3(1'b0),
    .s(state),
    .y(memread_mem)
);
mux4 #(1) memwrite_mux(
    .d0(1'b0),
    .d1(memwrite_cores[0]),
    .d2(memwrite_cores[1]),
```

```
        .d3(1'b0),
        .s(state),
        .y(memwrite_mem)
    );
endmodule

module mem_map(
    input [11:0] adr,
    input read,
    input write,

    output reg rom_read,
    output reg ram_read,
    output reg ram_write,
    output reg io_read,
    output reg io_write,
    output reg mmu_read,
    output reg mmu_write,

    output reg [7:0] readdata,
    input [7:0] rom_readdata,
    input [7:0] ram_readdata,
    input [7:0] io_readdata,
    input [7:0] mmu_readdata
);
    always @(*) begin
        rom_read <= 0;
        ram_read <= 0;
        io_read <= 0;
        mmu_read <= 0;
        ram_write <= 0;
        mmu_write <= 0;
        io_write <= 0;
        if(read) begin
            if(adr >= 12'h400) begin
                ram_read <= 1;
                readdata <= ram_readdata;
            end
            if(adr < 12'h040) begin
                rom_read <= 1;
                readdata <= rom_readdata;
            end
            if(adr >= 12'h040 && adr < 12'h080) begin
```

```
                io_read <= 1;
                readdata <= io_readdata;
            end
            if(adr >= 12'h080 && adr < 12'h0C0) begin
                mmu_read <= 1;
                readdata <= mmu_readdata;
            end
        end
        if(write) begin
            if(adr >= 12'h400) ram_write <= 1;
            if(adr >= 12'h040 && adr < 12'h080) io_write <= 1;
            if(adr >= 12'h080 && adr < 12'h0C0) mmu_write <= 1;
        end
    end
endmodule
```

IV Synthesis

The CPU synthesis was relatively straightforward as all design files were written in Verilog and only synthesizable features of Verilog were used in the design. Additionally, no scripts were needed when synthesizing our design. A detailed view of the Synthesis reports is shown in Appendix B. The total on-chip power consumption is 0.086W. Of this power consumption 83% is static power consumed when there is no circuit activity.

V Testing

To verify the functionality of our design two test benches were used. The first testbench was simply an external memory module (RAM) with machine code representing the calculation of the first 15 fibonacci numbers and employed the use of all instructions except the and, or, and slt instruction. The memory for this fibonacci program is included in *Figure 5* below.

```
RAM[0] <= 32'b100000_00000_00001_0000000001000000;
RAM[1] <= 32'b001000_00000_00010_0000000000000001;
RAM[2] <= 32'b001000_00000_00011_0000000000000001;
RAM[3] <= 32'b001000_00000_00100_0000000000000001;
RAM[4] <= 32'b001000_00000_00101_0000000000000010;
RAM[5] <= 32'b000100_00001_00101_0000000000000101;
RAM[6] <= 32'b000100_00001_00100_0000000000000101;
RAM[7] <= 32'b000000_00001_00101_00001_00000_100010;
RAM[8] <= 32'b000000_00010_00011_00010_00000_100000;
RAM[9] <= 32'b000000_00010_00011_00011_00000_100000;
RAM[10] <= 32'b000010_00000_0000000000010_00000101;
RAM[11] <= 32'b000000_00010_00011_00011_00000_100000
RAM[12] <= 32'b110000_00000_00011_0000000001000000;
RAM[13] <= 32'b000000_00011_00000_00011_00000_110001
RAM[14] <= 32'b110000_00000_00011_0000000001000001;
RAM[15] <= 32'b000010_00000_0000000000010_00001111;

RAM[64] <= 32'b001000_00000_00001_0000000000000001;
RAM[65] <= 32'b001000_00000_00010_0000000000000001;
RAM[66] <= 32'b001000_00000_00100_0000000000001000;
RAM[67] <= 32'b000000_00001_00010_00011_00000_100000
RAM[68] <= 32'b000000_00011_00010_00010_00000_100000
RAM[69] <= 32'b000000_00011_00010_00011_00000_100000
RAM[70] <= 32'b000000_00011_00100_00101_00000_100010
RAM[71] <= 32'b000100_00010_00100_0000000000000001;
RAM[72] <= 32'b000010_00000_0000000000010_00000100;
RAM[73] <= 32'b001000_00000_00111_00000010_00000001;
RAM[74] <= 32'b000010_00000_0000000000010_00001010;
```

Figure 5: RAM for Testbench #1

The machine code of *Figure 5.* This code loads input from the IO device to determine which Fibonacci number the user wants to calculate, it then initializes some registers and enters a loop which calculates two fibonacci numbers each iteration. At the end of each loop, there are two conditional branch instructions to check if there are 1 or 0 remaining steps. If there are zero iterations left, the code jumps to an end point that writes the result to the output registers. If there is one iteration left, the code jumps to a single extra addition instruction and then falls through to the end code. After writing both output bytes, the code “halts” with an instruction which jumps to itself.

To verify the operation of the CPU the test bench was simulated in order to view the waveforms of the CPU pins. This output is shown in *Figure 6*.

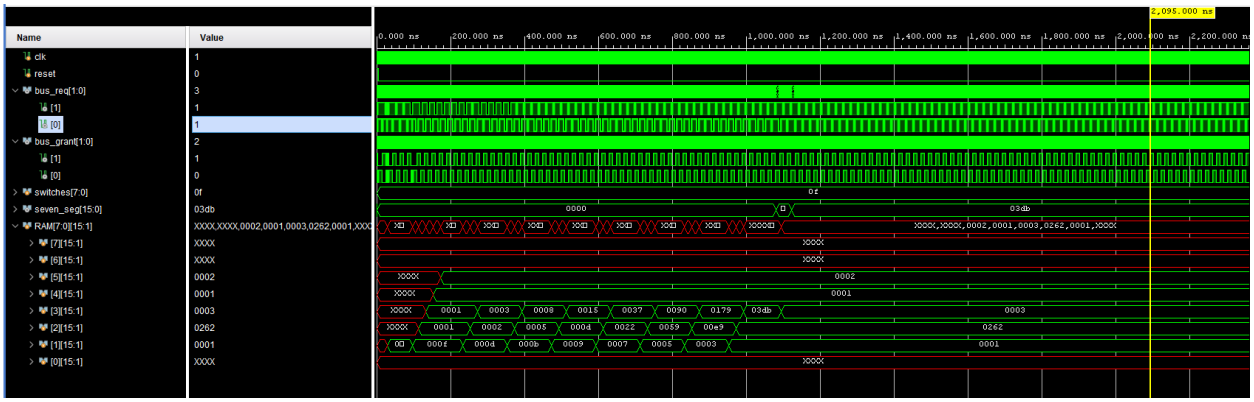


Figure 6: Results of Simulation Testbench #1

As can be seen in *Figure 6*, the results from running the program with an input of 15 is 0x03db or 987 (the 15th number in the fibonacci sequence). Additionally, it can be seen that both cpus are requesting but only one is given bus access at a time.

After this initial testing was completed a Randomized testbench was written to further validate the CPUs functionality. This testbench consisted of random classes for RType, IType, and JType instructions. Additionally a class to represent a random instruction and a class to represent a randomized RAM module that represents a single 32 bit instruction was created. The testbench performs 1000 iterations each time randomizing the instruction Type and specific values for that instruction stored in the ram module. Then the values of the registers are checked before allowing the controller to perform the fetch, decode, execute process. Afterwards, the CPU is allowed to run until execution of the instruction is completed. Then the state of the registers are again checked and if the final state is not what is expected by running the command then a message is raised and the test stopped. The details of how this testbench works can be seen in section III under the coverage_testbench.sv and instructions.sv files. The randomized testbench also utilized covergroups and coverpoints to determine the code coverage of the test. In our case with 1000 tests we achieved 100% code coverage. The output of the randomized test and coverage is shown below in *Figure 7*. A more detailed coverage report is included in Appendix C.

```
All tests passed
RTYPE funct coverage:          100 percent
ITYPE opcode coverage:          100 percent
ITYPE immediate coverage:          100 percent
JTYPE Address coverage:          100 percent
xsim: Time (s): cpu = 00:00:05 ; elapsed = 00:00:07 . Memory (MB): peak = 1522.727 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'controller_testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000000ns
launch_simulation: Time (s): cpu = 00:00:06 ; elapsed = 00:00:17 . Memory (MB): peak = 1522.727 ; gain = 0.000
```

(a) Output from Randomized Testbench

Cover Point Table for Inst : i_instr.my_cg	Variable :	imm		
Table tag :	my_cg_i_instr.my_cg_imm			
Summary				
Category	Expected	Uncovered	Covered	Percent
Automatic	64	0	64	100
Covered bins				
Name	Hit Count	AtLeast		
auto[0:3]	48	1		
auto[4:7]	48	1		
auto[8:11]	50	1		
auto[12:15]	33	1		
auto[16:19]	57	1		
auto[20:23]	61	1		
auto[24:27]	57	1		
auto[28:31]	49	1		
auto[32:35]	71	1		
auto[36:39]	66	1		
auto[40:43]	55	1		
auto[44:47]	41	1		
auto[48:51]	46	1		

(b) Code Coverage results

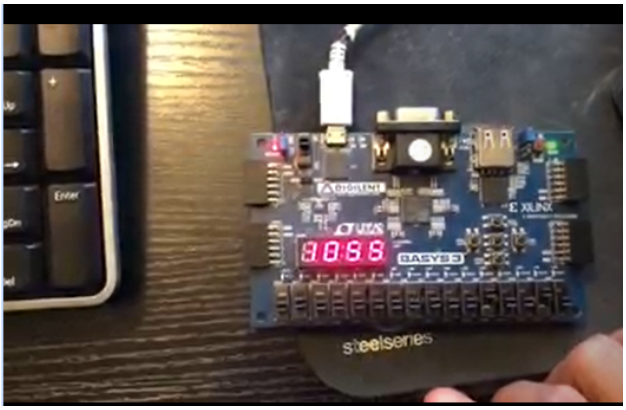
Figure 7: Output of Randomized Testbench and Code Coverage

VI FPGA board

The processor was synthesized to the BASYS 3 FPGA board. On the board the 8 right most switches were used as inputs into the processor and specified the nth number of the fibonacci sequence to compute in binary. The middle red button of the BASYS 3 was used to reset the CPU and the internal 100Mhz clock of the BASYS 3 was used as the CPUs clock. These physical inputs of the FPGA were mapped to the input/output ports of the multicore_top.sv file which housed the dual core MIPS CPU and logic connecting the switches and seven segment display to the CPU. The required constraint file is shown below in *Appendix A*. The input of the switches was routed into the CPUs IO module so it could be accessed by the CPU. The program in RAM reads from the IO to get the input from the FPGA switches and then computes the given fibonacci number. Results from three inputs into the CPU using the FPGA are shown below in *Figure 8*.

(a) Input = 4, Output = 0x0005 (5)

(b) input = 10, Output = 0x0059 (89)



(c) input = 18, Output = 0x1055 (4181)

VII Enhancements

Several enhancements were made to this CPU beyond the project requirements. In order to allow for the two cores to be truly identical and to have identical memory access, basic virtual memory functionality was added to the cores and several memory mapped peripherals were added to the system.

The virtual memory system consists of a new instruction named WRSPEC which writes from a general purpose register to a special purpose register. There is a single special purpose register which acts as the base offset for a virtual memory page. The low nibble of this register is used as the upper 4 bits of the physical address output when the CPU internally accesses any address above 0x800. In order to preserve access to memory mapped peripherals at all times, memory addresses below 0x800 are not remapped, meaning that the core always has direct access to the ROM, IO, and MMU.

A memory map module was created which routes memory access from the bus arbiter to the RAM, ROM, IO block, or MMU depending on the address. RAM backs all physical addresses above 0x400, IO and the MMU use addresses 0x040 and 0x080 respectively, and the ROM module is mapped to address 0x000, as it contains the first code to be executed after a reset. The memory map is a stateless, purely combinational module which passes the address, read, and write signals through to each device and masks the read and write signals to only the device corresponding to the input address.

The read only memory device functions exactly like that RAM except that it has fixed values stored in it and no write signal. The ROM allows up to 64 bytes of code to be executed when the CPU is reset. These 16 instructions should hold a small firmware which at minimum gets a page from the MMU, stores it into the virtual memory offset special register, and then jumps to the main program code. Our firmware does just this, ending with a jump to 0x800, which is the first address of the virtual memory space.

The memory management unit is a very simple device which tracks 4 available sections of the RAM to be handed out as pages of memory. Reading from the MMU returns a non-zero page offset if there is a free page and marks that page as unavailable. If there are no available pages, the MMU read will return 0, indicating that the CPU should wait and try again. Writing to the MMU register can be used to free a page; specifically, writing a page offset to the MMU will mark that page as free. The first two pages returned by the MMU are always 0x400 and 0x500, meaning that after a reset, core 0 will always be assigned page 0x400 and core 1 will always be assigned page 0x500. After this, the cores are free to request more pages or free their original page as needed.

The IO block allows for a single byte or input and 4 bytes of output to be accessed as simple memory mapped peripherals. The 1 byte of input is connected to 8 switches on the FPGA, and the 4 bytes of output are connected to 16 LEDs and the 4 digit seven segment display.

Combined, these enhancements allow for each core to boot and start running code from RAM independently which still being completely identical cores with identical access to memory. This also allows for code to be written which is relocatable, that is, all absolute jumps in user code can jump to 0x800 plus some offset, regardless of if the code is physically located at 0x400, 0x500, or anywhere else in RAM.

VIII Problems and Debugging

There were several problems encountered while creating this CPU. A single core design was easy to create after fixing a few typos in the provided code and creating a top level testbench, but moving to a dual core design led to some unexpected problems. It was known that the controller design would need to change to accommodate a shared memory bus design because the CPU cannot advance when it isn't able to read instructions; problems arose when changes needed to be made to the datapath. The most salient example is in the memory write instruction. In a single core with full memory access, address calculation always happens one clock cycle before the RAM write, but in a shared system the controller may need to stall before it gets access to the RAM to perform the write. In this case, the register which holds the address to be written to must be disabled or it will be overwritten with garbage. This required changing the flip flop to a flip flop with enable signal as well as creating a new controller output.

Debugging for this project was done exclusively with the Vivado waveform tool. Test benches were created with different instructions to execute and then the internal signals of the CPU were probed. The register file was a particularly helpful area to probe, the expected values in registers are fairly obvious from a set of instructions.

IX Conclusions

The final pincount of our CPU (multicore_top) was 22 pins. Additionally, 548 lookup tables, 285 slice registers, 70 F7 Muxes, 32 F8 Muxes were used. Overall, our CPU worked as intended and mimicked the architecture of a true dual-core CPU on a much simpler scale. However, if given more time there are many improvements that can be made to the CPU's design. First we could have added caching to allow for frequently used memory to be accessed faster. If the cache could be accessed 32 bits at a time, then the CPU could have executed cached instructions in a single cycle. If this were implemented, then it would also be reasonable to implement instruction pipelining to allow multiple instructions to be executed simultaneously. These two improvements would increase performance by as much as a factor of 7 depending on the instructions being executed. Other possible improvements would include support for more arithmetic operations, support for more instructions, or a more advanced boot sequence to simulate loading data from an external source such as an SD card. Despite the lackluster performance compared to optimized systems, we were still able to demonstrate the principles of a multicore system using basic memory management and memory mapped peripherals. We successfully designed a system to allow identical cores to boot simultaneously and transparently start working on different tasks.

Appendix A: constr.xdc

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk100m]
      set_property IOSTANDARD LVCMOS33 [get_ports clk100m]
      create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk100m]
# Switches
set_property PACKAGE_PIN V17 [get_ports {switches[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[0]}]
set_property PACKAGE_PIN V16 [get_ports {switches[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[1]}]
set_property PACKAGE_PIN W16 [get_ports {switches[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[2]}]
set_property PACKAGE_PIN W17 [get_ports {switches[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[3]}]
set_property PACKAGE_PIN W15 [get_ports {switches[4]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[4]}]
set_property PACKAGE_PIN V15 [get_ports {switches[5]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[5]}]
set_property PACKAGE_PIN W14 [get_ports {switches[6]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[6]}]
set_property PACKAGE_PIN W13 [get_ports {switches[7]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {switches[7]}]
#seven-segment LED display
set_property PACKAGE_PIN W7 [get_ports {cathode_pattern[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cathode_pattern[6]}]
set_property PACKAGE_PIN W6 [get_ports {cathode_pattern[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cathode_pattern[5]}]
set_property PACKAGE_PIN U8 [get_ports {cathode_pattern[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cathode_pattern[4]}]
set_property PACKAGE_PIN V8 [get_ports {cathode_pattern[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cathode_pattern[3]}]
set_property PACKAGE_PIN U5 [get_ports {cathode_pattern[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cathode_pattern[2]}]
set_property PACKAGE_PIN V5 [get_ports {cathode_pattern[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cathode_pattern[1]}]
set_property PACKAGE_PIN U7 [get_ports {cathode_pattern[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cathode_pattern[0]}]
set_property PACKAGE_PIN U2 [get_ports {anode_activate[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[0]}]
set_property PACKAGE_PIN U4 [get_ports {anode_activate[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[1]}]
set_property PACKAGE_PIN V4 [get_ports {anode_activate[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[2]}]
set_property PACKAGE_PIN W4 [get_ports {anode_activate[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[3]}]
##Buttons
set_property PACKAGE_PIN U18 [get_ports reset]
      set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN T18 [get_ports clk_reset]
      set_property IOSTANDARD LVCMOS33 [get_ports clk_reset]
```

Appendix B: Synthesis Reports

Power

Settings
Summary (0.086 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.014 W)
Clocks (0.001 W)
Signals (0.003 W)
Data (0.003 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (0.003 W)
I/O (0.008 W)

Device

Part:	xc7a35tcpg236-1
Temp grade:	commercial
Process:	typical
Characterization:	Production

Environment

Output Load:	0 pF
Ambient temperature:	25.0 °C
Airflow:	250 LFM
Heat sink:	medium (Medium Profile)
θSA:	4.6 °C/W
Board selection:	medium (10"x10")
Number of board layers:	12to15 (12 to 15 Layers)
θJB:	7.5 °C/W
Board temperature:	25.0 °C

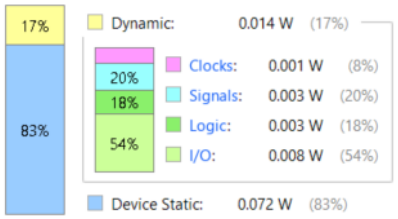
Settings
Summary (0.086 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.014 W)
Clocks (0.001 W)
Signals (0.003 W)
Data (0.003 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (0.003 W)
I/O (0.008 W)

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.086 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (11.8 W)
Effective θJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



Settings
Summary (0.086 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.014 W)
Clocks (0.001 W)
Signals (0.003 W)
Data (0.003 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (0.003 W)
I/O (0.008 W)

Supply Source	Voltage (V)	Total (A)	Dynamic (A)	Static (A)	Budget (A)	Margin (A)
Vccint	1.000	0.016	0.007	0.010	Unspecified	NA
Vccaux	1.800	0.013	0.000	0.013	Unspecified	NA
Vcco33	3.300	0.003	0.002	0.001	Unspecified	NA
Vcco25	2.500	0.000	0.000	0.000	Unspecified	NA
Vcco18	1.800	0.000	0.000	0.000	Unspecified	NA
Vcco15	1.500	0.000	0.000	0.000	Unspecified	NA
Vcco135	1.350	0.000	0.000	0.000	Unspecified	NA
Vcco12	1.200	0.000	0.000	0.000	Unspecified	NA
Vccaux_io	1.800	0.000	0.000	0.000	Unspecified	NA
Vccbram	1.000	0.000	0.000	0.000	Unspecified	NA
MGTAVcc	1.000	0.000	0.000	0.000	Unspecified	NA
MGTAVtt	1.200	0.000	0.000	0.000	Unspecified	NA
Vccadc	1.800	0.020	0.000	0.020	Unspecified	NA

Settings
Summary (0.086 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.014 W)
Clocks (0.001 W)
Signals (0.003 W)
Data (0.003 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (0.003 W)
I/O (0.008 W)

Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	I/O (W)
0.014 W (17% of total)	multicore_top							
0.008 W (10% of total)	Leaf Cells (47)							
0.002 W (3% of total)	my_core_0 (mips)	<0.001	0.001	0.001	<0.001	<0.001	0.001	<0.001
0.002 W (2% of total)	my_core_1 (mips_0)	<0.001	0.001	0.001	<0.001	<0.001	0.001	<0.001
0.001 W (1% of total)	nolabel_line45 (seven_segment_controller)	0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (1% of total)	my_mem (exmemory)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	my_arbiter (mem_arbiter)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	my_map (mem_map)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	my_io (io_regs)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	my_mmu (mmu)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001

Appendix B: Synthesis Reports - Utilization

```
Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2021.1 (win64) Build 3247384 Thu Jun 10 19:36:33 MDT 2021
| Date         : Thu Dec 16 21:33:18 2021
| Host         : LAPTOP-B3ESP72V running 64-bit major release  (build 9200)
| Command      : report_utilization -file multicore_top_utilization_synth.rpt -pb
multicore_top_utilization_synth.pb
| Design       : multicore_top
| Device       : 7a35tcpg236-1
| Design State : Synthesized
------------------------------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+----------------------------+------+-------+------------+-----------+-------+
|          Site Type         | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*                |  568 |     0 |          0 |     20800 |  2.73 |
|   LUT as Logic             |  392 |     0 |          0 |     20800 |  1.88 |
|   LUT as Memory            |  176 |     0 |          0 |      9600 |  1.83 |
|     LUT as Distributed RAM |  176 |     0 |            |           |       |
|     LUT as Shift Register  |    0 |     0 |            |           |       |
```

Slice Registers	285	0	0	41600	0.69
Register as Flip Flop	277	0	0	41600	0.67
Register as Latch	8	0	0	41600	0.02
F7 Muxes	64	0	0	16300	0.39
F8 Muxes	32	0	0	8150	0.39

* Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt_design after synthesis, if not already completed, for a more realistic count.

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
28	Yes	-	Reset
7	Yes	Set	-
250	Yes	Reset	-

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	50	0.00
RAMB36/FIFO*	0	0	0	50	0.00
RAMB18	0	0	0	100	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	0	0	0	90	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	22	0	0	106	20.75
Bonded IPADs	0	0	0	10	0.00
Bonded OPADs	0	0	0	4	0.00
PHY_CONTROL	0	0	0	5	0.00
PHASER_REF	0	0	0	5	0.00
OUT_FIFO	0	0	0	20	0.00
IN_FIFO	0	0	0	20	0.00
IDELAYCTRL	0	0	0	5	0.00
IBUFDS	0	0	0	104	0.00
GTPE2_CHANNEL	0	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	250	0.00
IBUFDS_GTE2	0	0	0	2	0.00
ILOGIC	0	0	0	106	0.00
OLOGIC	0	0	0	106	0.00

5. Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	2	0	0	32	6.25
BUFIO	0	0	0	20	0.00
MMCME2_ADV	0	0	0	5	0.00
PLLE2_ADV	0	0	0	5	0.00
BUFMRCE	0	0	0	10	0.00
BUFHCE	0	0	0	72	0.00
BUFR	0	0	0	20	0.00

6. Specific Feature

Site Type	Used	Fixed	Prohibited	Available	Util%
BSCANE2	0	0	0	4	0.00
CAPTUREE2	0	0	0	1	0.00
DNA_PORT	0	0	0	1	0.00
EFUSE_USR	0	0	0	1	0.00
FRAME_ECCE2	0	0	0	1	0.00
ICAPE2	0	0	0	2	0.00
PCIE_2_1	0	0	0	1	0.00
STARTUPE2	0	0	0	1	0.00
XADC	0	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	250	Flop & Latch
LUT6	204	LUT
RAMS64E	128	Distributed Memory
RAMD32	72	Distributed Memory
LUT3	66	LUT

MUXF7	64	MuxFx
LUT4	62	LUT
LUT5	60	LUT
LUT2	44	LUT
MUXF8	32	MuxFx
RAMS32	24	Distributed Memory
FDCE	20	Flop & Latch
CARRY4	17	CarryLogic
OBUF	11	IO
IBUF	11	IO
LDCE	8	Flop & Latch
FDSE	7	Flop & Latch
LUT1	4	LUT
BUFG	2	Clock

8. Black Boxes

| Ref Name | Used |

9. Instantiated Netlists

| Ref Name | Used |

Appendix C: Code Coverage

```
 Xcrg Coverage Main Report

   Date             :, Wed Dec  8 10:37:00 2021 EST
   User             :, m1fream
   Version          :, Vivado Simulator Coverage Report 2020.1
   Command Line     :, /tools/Xilinx/Vivado/2020.1/bin/unwrapped/lnx64.o/xcrg -report_format text
   Number of Tests  :, 1

   Coverage Score            :, 100
   Total Insts Score         :, 100
   Total no of Cover Groups  :, 3
   Total no of Instances     :, 3


     CgName,                                          Score,      NumInsts,   AvgInstScore,   Weight,
Goal,       Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

       Instructions::RType::my_cg                        ,100         ,1           ,100            ,1
,100         ,0               ,0              ,0           ,64              ,            ,
       Instructions::IType::my_cg                        ,100         ,1           ,100            ,1
,100         ,0               ,0              ,0           ,64              ,            ,
       Instructions::JType::my_cg                        ,100         ,1           ,100            ,1
,100         ,0               ,0              ,0           ,64              ,            ,




Cover Group Details : Instructions::RType::my_cg

CgName,                                       Score,      NumInsts,   AvgInstScore,   Weight,     Goal,
Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

Instructions::RType::my_cg                        ,100         ,1           ,100            ,1           ,100
,0               ,0              ,0           ,64              ,            ,

      Instances of CoverGroup - Instructions::RType::my_cg
```

```
            InstName,                                           Score,      Weight,     Goal,       AtLeast,
AutoBinMax, PrintMissing,   IsInst,     Comment,
            r_instr.my_cg                                       ,100        ,1          ,100        ,1
,64         ,0              ,1          ,           ,

      Instance r_instr.my_cg Cover Point Details

            Name,                                               TableTag,     Expected,   Uncovered,  Covered,
Percent,    Goal,       Weight,     AtLeast,    AutoBinMax, Comment,
            funct_point                                         ,my_cg_r_instr.my_cg_funct_point,6          ,0
,6          ,100        ,100        ,1          ,1            ,64          ,           ,



Cover Group Details : Instructions::IType::my_cg

CgName,                                         Score,      NumInsts,   AvgInstScore,   Weight,     Goal,
Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

Instructions::IType::my_cg                      ,100        ,1          ,100            ,1          ,100
,0              ,0              ,0          ,64             ,           ,

      Instances of CoverGroup - Instructions::IType::my_cg

            InstName,                                           Score,      Weight,     Goal,       AtLeast,
AutoBinMax, PrintMissing,   IsInst,     Comment,
            i_instr.my_cg                                       ,100        ,1          ,100        ,1
,64         ,0              ,1          ,           ,

      Instance i_instr.my_cg Cover Point Details

            Name,                                               TableTag,     Expected,   Uncovered,  Covered,
Percent,    Goal,       Weight,     AtLeast,    AutoBinMax, Comment,
```

```
            op                                                  ,my_cg_i_instr.my_cg_op,4           ,0
,4          ,100         ,100         ,1           ,1           ,64          ,            ,
            imm                                                 ,my_cg_i_instr.my_cg_imm,64         ,0
,64         ,100         ,100         ,1           ,1           ,64          ,            ,
```

```
Cover Group Details : Instructions::JType::my_cg

CgName,                                        Score,      NumInsts,   AvgInstScore,   Weight,     Goal,
Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

Instructions::JType::my_cg                      ,100         ,1           ,100            ,1           ,100
,0              ,0              ,0           ,64             ,            ,
```

```
      Instances of CoverGroup - Instructions::JType::my_cg

            InstName,                                       Score,      Weight,     Goal,       AtLeast,
AutoBinMax, PrintMissing,   IsInst,     Comment,
            j_instr.my_cg                                   ,100         ,1           ,100         ,1
,64         ,0              ,1           ,            ,
```

```
      Instance j_instr.my_cg Cover Point Details

            Name,                                           TableTag,     Expected,   Uncovered,  Covered,
Percent,    Goal,       Weight,     AtLeast,    AutoBinMax, Comment,
            addr                                            ,my_cg_j_instr.my_cg_addr,64         ,0
,64         ,100         ,100         ,1           ,1           ,64          ,            ,
```

```
Cover Point Table for Inst : r_instr.my_cg, Variable :, funct_point
Table tag :, my_cg_r_instr.my_cg_funct_point

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
UserDefined,   6           ,0          ,6           ,100        ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     ADD_OP      ,597         ,1          ,
     SUB_OP      ,285         ,1          ,
     AND_OP      ,351         ,1          ,
     OR_OP       ,476         ,1          ,
     LSHIFT_OP   ,606         ,1          ,
     RSHIFT_OP   ,1032        ,1          ,



Cover Point Table for Inst : i_instr.my_cg, Variable :, op
Table tag :, my_cg_i_instr.my_cg_op

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
UserDefined,   4           ,0          ,4           ,100        ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     BEQ         ,467         ,1          ,
     ADDI        ,1400        ,1          ,
     LOAD        ,623         ,1          ,
     STORE       ,855         ,1          ,
```

```
Cover Point Table for Inst : i_instr.my_cg, Variable :, imm
Table tag :, my_cg_i_instr.my_cg_imm

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
Automatic,     64          ,0          ,64          ,100         ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     auto[0:3]   ,48          ,1          ,
     auto[4:7]   ,48          ,1          ,
     auto[8:11]  ,50          ,1          ,
     auto[12:15] ,33          ,1          ,
     auto[16:19] ,57          ,1          ,
     auto[20:23] ,61          ,1          ,
     auto[24:27] ,57          ,1          ,
     auto[28:31] ,49          ,1          ,
     auto[32:35] ,71          ,1          ,
     auto[36:39] ,66          ,1          ,
     auto[40:43] ,55          ,1          ,
     auto[44:47] ,41          ,1          ,
     auto[48:51] ,46          ,1          ,
     auto[52:55] ,57          ,1          ,
     auto[56:59] ,57          ,1          ,
     auto[60:63] ,56          ,1          ,
     auto[64:67] ,53          ,1          ,
     auto[68:71] ,44          ,1          ,
     auto[72:75] ,59          ,1          ,
     auto[76:79] ,41          ,1          ,
     auto[80:83] ,54          ,1          ,
     auto[84:87] ,45          ,1          ,
     auto[88:91] ,60          ,1          ,
     auto[92:95] ,49          ,1          ,
     auto[96:99] ,45          ,1          ,
     auto[100:103],47         ,1          ,
```

```
auto[104:107],54          ,1          ,
auto[108:111],44          ,1          ,
auto[112:115],63          ,1          ,
auto[116:119],57          ,1          ,
auto[120:123],50          ,1          ,
auto[124:127],49          ,1          ,
auto[128:131],50          ,1          ,
auto[132:135],38          ,1          ,
auto[136:139],83          ,1          ,
auto[140:143],48          ,1          ,
auto[144:147],53          ,1          ,
auto[148:151],45          ,1          ,
auto[152:155],43          ,1          ,
auto[156:159],56          ,1          ,
auto[160:163],68          ,1          ,
auto[164:167],58          ,1          ,
auto[168:171],54          ,1          ,
auto[172:175],45          ,1          ,
auto[176:179],46          ,1          ,
auto[180:183],52          ,1          ,
auto[184:187],53          ,1          ,
auto[188:191],54          ,1          ,
auto[192:195],66          ,1          ,
auto[196:199],46          ,1          ,
auto[200:203],54          ,1          ,
auto[204:207],46          ,1          ,
auto[208:211],56          ,1          ,
auto[212:215],42          ,1          ,
auto[216:219],55          ,1          ,
auto[220:223],68          ,1          ,
auto[224:227],46          ,1          ,
auto[228:231],44          ,1          ,
auto[232:235],67          ,1          ,
auto[236:239],46          ,1          ,
auto[240:243],51          ,1          ,
auto[244:247],43          ,1          ,
auto[248:251],50          ,1          ,
auto[252:255],53          ,1          ,
```

```
Cover Point Table for Inst : j_instr.my_cg, Variable :, addr
Table tag :, my_cg_j_instr.my_cg_addr

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
Automatic,     64          ,0          ,64          ,100         ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     auto[0:3]   ,48          ,1           ,
     auto[4:7]   ,48          ,1           ,
     auto[8:11]  ,60          ,1           ,
     auto[12:15] ,55          ,1           ,
     auto[16:19] ,35          ,1           ,
     auto[20:23] ,45          ,1           ,
     auto[24:27] ,48          ,1           ,
     auto[28:31] ,49          ,1           ,
     auto[32:35] ,58          ,1           ,
     auto[36:39] ,60          ,1           ,
     auto[40:43] ,53          ,1           ,
     auto[44:47] ,63          ,1           ,
     auto[48:51] ,63          ,1           ,
     auto[52:55] ,50          ,1           ,
     auto[56:59] ,59          ,1           ,
     auto[60:63] ,51          ,1           ,
     auto[64:67] ,62          ,1           ,
     auto[68:71] ,50          ,1           ,
     auto[72:75] ,43          ,1           ,
     auto[76:79] ,47          ,1           ,
     auto[80:83] ,63          ,1           ,
     auto[84:87] ,53          ,1           ,
     auto[88:91] ,41          ,1           ,
     auto[92:95] ,56          ,1           ,
     auto[96:99] ,58          ,1           ,
     auto[100:103],44         ,1           ,
     auto[104:107],41         ,1           ,
```

```
auto[108:111],50          ,1          ,
auto[112:115],48          ,1          ,
auto[116:119],58          ,1          ,
auto[120:123],46          ,1          ,
auto[124:127],55          ,1          ,
auto[128:131],38          ,1          ,
auto[132:135],52          ,1          ,
auto[136:139],50          ,1          ,
auto[140:143],50          ,1          ,
auto[144:147],51          ,1          ,
auto[148:151],67          ,1          ,
auto[152:155],44          ,1          ,
auto[156:159],52          ,1          ,
auto[160:163],41          ,1          ,
auto[164:167],64          ,1          ,
auto[168:171],56          ,1          ,
auto[172:175],54          ,1          ,
auto[176:179],44          ,1          ,
auto[180:183],53          ,1          ,
auto[184:187],46          ,1          ,
auto[188:191],61          ,1          ,
auto[192:195],42          ,1          ,
auto[196:199],62          ,1          ,
auto[200:203],29          ,1          ,
auto[204:207],56          ,1          ,
auto[208:211],61          ,1          ,
auto[212:215],63          ,1          ,
auto[216:219],56          ,1          ,
auto[220:223],65          ,1          ,
auto[224:227],49          ,1          ,
auto[228:231],45          ,1          ,
auto[232:235],60          ,1          ,
auto[236:239],38          ,1          ,
auto[240:243],50          ,1          ,
auto[244:247],51          ,1          ,
auto[248:251],46          ,1          ,
auto[252:255],52          ,1          ,
```

```
 Xcrg Coverage Main Report

   Date             :, Wed Dec  8 10:37:19 2021 EST
   User             :, m1fream
   Version          :, Vivado Simulator Coverage Report 2020.1
   Command Line     :, /tools/Xilinx/Vivado/2020.1/bin/unwrapped/lnx64.o/xcrg -report_format all
   Number of Tests  :, 1

   Coverage Score            :, 100
   Total Insts Score         :, 100
   Total no of Cover Groups  :, 3
   Total no of Instances     :, 3


     CgName,                                          Score,      NumInsts,   AvgInstScore,   Weight,
Goal,       Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

       Instructions::RType::my_cg                      ,100        ,1          ,100            ,1
,100         ,0              ,0              ,0          ,64             ,           ,
       Instructions::IType::my_cg                      ,100        ,1          ,100            ,1
,100         ,0              ,0              ,0          ,64             ,           ,
       Instructions::JType::my_cg                      ,100        ,1          ,100            ,1
,100         ,0              ,0              ,0          ,64             ,           ,




Cover Group Details : Instructions::RType::my_cg

CgName,                                       Score,      NumInsts,   AvgInstScore,   Weight,     Goal,
Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

Instructions::RType::my_cg                      ,100        ,1          ,100            ,1          ,100
,0              ,0              ,0          ,64             ,           ,
```

```
      Instances of CoverGroup - Instructions::RType::my_cg

              InstName,                                          Score,      Weight,     Goal,       AtLeast,
AutoBinMax, PrintMissing,   IsInst,     Comment,
              r_instr.my_cg                                      ,100         ,1           ,100         ,1
,64           ,0               ,1           ,             ,


      Instance r_instr.my_cg Cover Point Details

              Name,                                              TableTag,     Expected,   Uncovered,  Covered,
Percent,     Goal,       Weight,     AtLeast,    AutoBinMax, Comment,
              funct_point                                        ,my_cg_r_instr.my_cg_funct_point,6           ,0
,6            ,100         ,100         ,1           ,1           ,64           ,             ,



Cover Group Details : Instructions::IType::my_cg

CgName,                                          Score,      NumInsts,   AvgInstScore,   Weight,     Goal,
Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

Instructions::IType::my_cg                        ,100         ,1           ,100             ,1           ,100
,0               ,0               ,0           ,64              ,             ,

      Instances of CoverGroup - Instructions::IType::my_cg

              InstName,                                          Score,      Weight,     Goal,       AtLeast,
AutoBinMax, PrintMissing,   IsInst,     Comment,
              i_instr.my_cg                                      ,100         ,1           ,100         ,1
,64           ,0               ,1           ,             ,


      Instance i_instr.my_cg Cover Point Details
```

```
             Name,                                                TableTag,     Expected,   Uncovered,  Covered,
Percent,     Goal,       Weight,     AtLeast,    AutoBinMax, Comment,
             op                                                  ,my_cg_i_instr.my_cg_op,4           ,0
,4           ,100         ,100         ,1           ,1           ,64          ,             ,
             imm                                                 ,my_cg_i_instr.my_cg_imm,64         ,0
,64          ,100         ,100         ,1           ,1           ,64          ,             ,
```

```
Cover Group Details : Instructions::JType::my_cg

CgName,                                        Score,      NumInsts,   AvgInstScore,   Weight,     Goal,
Merge Insts,    Get Inst Cov,   Per Inst,   AutoBinMax,     Comment,

Instructions::JType::my_cg                      ,100         ,1           ,100             ,1           ,100
,0               ,0              ,0           ,64              ,             ,
```

```
      Instances of CoverGroup - Instructions::JType::my_cg

             InstName,                                           Score,      Weight,     Goal,       AtLeast,
AutoBinMax, PrintMissing,   IsInst,     Comment,
             j_instr.my_cg                                       ,100         ,1           ,100         ,1
,64          ,0               ,1           ,            ,
```

```
      Instance j_instr.my_cg Cover Point Details

             Name,                                                TableTag,     Expected,   Uncovered,  Covered,
Percent,     Goal,       Weight,     AtLeast,    AutoBinMax, Comment,
             addr                                                ,my_cg_j_instr.my_cg_addr,64          ,0
,64          ,100         ,100         ,1           ,1           ,64          ,             ,
```

```
Cover Point Table for Inst : r_instr.my_cg, Variable :, funct_point
Table tag :, my_cg_r_instr.my_cg_funct_point

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
UserDefined,   6           ,0          ,6           ,100         ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     ADD_OP      ,1194        ,1          ,
     SUB_OP      ,570         ,1          ,
     AND_OP      ,702         ,1          ,
     OR_OP       ,952         ,1          ,
     LSHIFT_OP   ,1212        ,1          ,
     RSHIFT_OP   ,2064        ,1          ,




Cover Point Table for Inst : i_instr.my_cg, Variable :, op
Table tag :, my_cg_i_instr.my_cg_op

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
UserDefined,   4           ,0          ,4           ,100         ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     BEQ         ,934         ,1          ,
     ADDI        ,2800        ,1          ,
     LOAD        ,1246        ,1          ,
     STORE       ,1710        ,1          ,
```

```
Cover Point Table for Inst : i_instr.my_cg, Variable :, imm
Table tag :, my_cg_i_instr.my_cg_imm

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
Automatic,     64          ,0          ,64          ,100         ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     auto[0:3]   ,96          ,1          ,
     auto[4:7]   ,96          ,1          ,
     auto[8:11]  ,100         ,1          ,
     auto[12:15] ,66          ,1          ,
     auto[16:19] ,114         ,1          ,
     auto[20:23] ,122         ,1          ,
     auto[24:27] ,114         ,1          ,
     auto[28:31] ,98          ,1          ,
     auto[32:35] ,142         ,1          ,
     auto[36:39] ,132         ,1          ,
     auto[40:43] ,110         ,1          ,
     auto[44:47] ,82          ,1          ,
     auto[48:51] ,92          ,1          ,
     auto[52:55] ,114         ,1          ,
     auto[56:59] ,114         ,1          ,
     auto[60:63] ,112         ,1          ,
     auto[64:67] ,106         ,1          ,
     auto[68:71] ,88          ,1          ,
     auto[72:75] ,118         ,1          ,
     auto[76:79] ,82          ,1          ,
     auto[80:83] ,108         ,1          ,
     auto[84:87] ,90          ,1          ,
     auto[88:91] ,120         ,1          ,
     auto[92:95] ,98          ,1          ,
```

```
auto[96:99] ,90          ,1           ,
auto[100:103],94          ,1           ,
auto[104:107],108         ,1           ,
auto[108:111],88          ,1           ,
auto[112:115],126         ,1           ,
auto[116:119],114         ,1           ,
auto[120:123],100         ,1           ,
auto[124:127],98          ,1           ,
auto[128:131],100         ,1           ,
auto[132:135],76          ,1           ,
auto[136:139],166         ,1           ,
auto[140:143],96          ,1           ,
auto[144:147],106         ,1           ,
auto[148:151],90          ,1           ,
auto[152:155],86          ,1           ,
auto[156:159],112         ,1           ,
auto[160:163],136         ,1           ,
auto[164:167],116         ,1           ,
auto[168:171],108         ,1           ,
auto[172:175],90          ,1           ,
auto[176:179],92          ,1           ,
auto[180:183],104         ,1           ,
auto[184:187],106         ,1           ,
auto[188:191],108         ,1           ,
auto[192:195],132         ,1           ,
auto[196:199],92          ,1           ,
auto[200:203],108         ,1           ,
auto[204:207],92          ,1           ,
auto[208:211],112         ,1           ,
auto[212:215],84          ,1           ,
auto[216:219],110         ,1           ,
auto[220:223],136         ,1           ,
auto[224:227],92          ,1           ,
auto[228:231],88          ,1           ,
auto[232:235],134         ,1           ,
auto[236:239],92          ,1           ,
auto[240:243],102         ,1           ,
auto[244:247],86          ,1           ,
auto[248:251],100         ,1           ,
auto[252:255],106         ,1           ,
```

```
Cover Point Table for Inst : j_instr.my_cg, Variable :, addr
Table tag :, my_cg_j_instr.my_cg_addr

                    Summary

Category,       Expected,   Uncovered,  Covered,    Percent,
Automatic,     64          ,0          ,64          ,100         ,

                   Covered bins

     Name,       Hit Count,  AtLeast,
     auto[0:3]   ,96          ,1          ,
     auto[4:7]   ,96          ,1          ,
     auto[8:11]  ,120         ,1          ,
     auto[12:15] ,110         ,1          ,
     auto[16:19] ,70          ,1          ,
     auto[20:23] ,90          ,1          ,
     auto[24:27] ,96          ,1          ,
     auto[28:31] ,98          ,1          ,
     auto[32:35] ,116         ,1          ,
     auto[36:39] ,120         ,1          ,
     auto[40:43] ,106         ,1          ,
     auto[44:47] ,126         ,1          ,
     auto[48:51] ,126         ,1          ,
     auto[52:55] ,100         ,1          ,
     auto[56:59] ,118         ,1          ,
     auto[60:63] ,102         ,1          ,
     auto[64:67] ,124         ,1          ,
     auto[68:71] ,100         ,1          ,
     auto[72:75] ,86          ,1          ,
     auto[76:79] ,94          ,1          ,
     auto[80:83] ,126         ,1          ,
     auto[84:87] ,106         ,1          ,
     auto[88:91] ,82          ,1          ,
     auto[92:95] ,112         ,1          ,
     auto[96:99] ,116         ,1          ,
```

```
auto[100:103],88          ,1          ,
auto[104:107],82          ,1          ,
auto[108:111],100         ,1          ,
auto[112:115],96          ,1          ,
auto[116:119],116         ,1          ,
auto[120:123],92          ,1          ,
auto[124:127],110         ,1          ,
auto[128:131],76          ,1          ,
auto[132:135],104         ,1          ,
auto[136:139],100         ,1          ,
auto[140:143],100         ,1          ,
auto[144:147],102         ,1          ,
auto[148:151],134         ,1          ,
auto[152:155],88          ,1          ,
auto[156:159],104         ,1          ,
auto[160:163],82          ,1          ,
auto[164:167],128         ,1          ,
auto[168:171],112         ,1          ,
auto[172:175],108         ,1          ,
auto[176:179],88          ,1          ,
auto[180:183],106         ,1          ,
auto[184:187],92          ,1          ,
auto[188:191],122         ,1          ,
auto[192:195],84          ,1          ,
auto[196:199],124         ,1          ,
auto[200:203],58          ,1          ,
auto[204:207],112         ,1          ,
auto[208:211],122         ,1          ,
auto[212:215],126         ,1          ,
auto[216:219],112         ,1          ,
auto[220:223],130         ,1          ,
auto[224:227],98          ,1          ,
auto[228:231],90          ,1          ,
auto[232:235],120         ,1          ,
auto[236:239],76          ,1          ,
auto[240:243],100         ,1          ,
auto[244:247],102         ,1          ,
auto[248:251],92          ,1          ,
auto[252:255],104         ,1
```