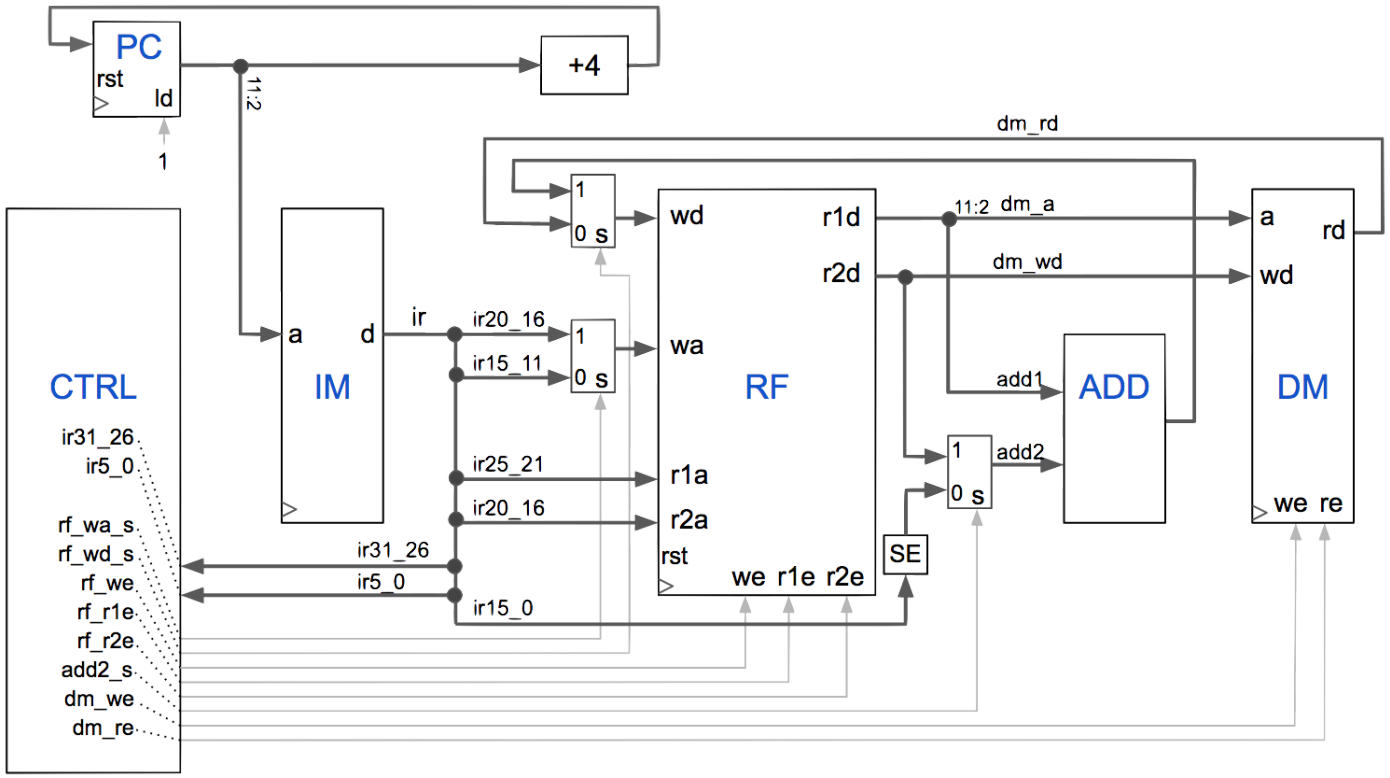
7.15 Base MIPSzy : Verilog

Top-level structure

An earlier section introduced the base MIPSzy's processor design. The base MIPSzy only supports the lw, sw, addi, and adc
The base MIPSzy is kept simple so the design's main concepts aren't lost in details of supporting other instructions. The de
again below. The design connects several components like CTRL, IM, RF, ADD, DM, PC, SE, +4, and some muxes.

Figure 7.15.1: Base MIPSzy processor design.



A ***hardware description language*** (***HDL***) is a language for describing digital circuit designs. ***Verilog*** is a popular hardware de language. An HDL can describe a design's structure, meaning the connection of components. Below is Verilog describing M structure. The design is a "module" with inputs clk and rst (clock and reset). The module lists numerous wires, each with a r rf_wd_s (one bit) and ir (32 bits). The module introduces several components (known as "instantiating" components) like IM RF_wd_mux, each an instance of a particular component type like mem_1024x32, register_32, and mux2x1_32 (respectivel component instantiation also lists the wires that connect to that component's inputs and outputs, as in PC (clk, rst, 1, pc_in Each such component will be defined later in the Verilog; this module merely instantiates and connects those "top-level" co

Figure 7.15.2: Verilog for MIPSzy's top-level structure.

```
`timescale 1ns / 1ps
module MIPSzy(clk, rst);
   input clk, rst;

   // Ctrl wires
   wire rf_wd_s, rf_wa_s, rf_we, rf_r1e, rf_r2e;
   wire add2_s;
   wire dm_we, dm_re;

   // PC wires
   wire [31:0] pc_out, pc_inc4;

   // IM/IR wires
   wire [31:0] ir;
   wire [5:0] ir31_26, ir5_0;
   wire [4:0] ir20_16, ir15_11, ir25_21;
   wire [15:0] ir15_0;

   // SW wires
   wire [31:0] ir15_0_se;

   // RF wires
   wire [4:0] rf_wa;
   wire [31:0] rf_wd;
   wire [31:0] rf_r1d, rf_r2d;

   // Adder wires
   wire [31:0] add2, add1, add_sum;

   // DM wires
   wire [31:0] dm_a;
   wire [31:0] dm_wd;
   wire [31:0] dm_rd;
```

```
    // internal connections
    assign dm_a = rf_r1d;
    assign dm_wd = rf_r2d;
    assign add1 = rf_r1d;

    assign ir31_26 = ir[31:26];
    assign ir5_0 = ir[5:0];
    assign ir20_16 = ir[20:16];
    assign ir15_11 = ir[15:11];
    assign ir25_21 = ir[25:21];
    assign ir15_0 = ir[15:0];

    // component instantiations
    mipszy_ctrl    CTRL(ir31_26, ir5_0,
                        rf_wd_s, rf_wa_s, rf_we, rf_r1e, rf_r2e,
                        add2_s, dm_we, dm_re);
    register_32    PC(clk, rst, 1, pc_inc4, pc_out);
    inc4_32        PC_inc(pc_out, pc_inc4);
    mux2x1_32      RF_wd_mux(add_sum, dm_rd, rf_wd_s, rf_wd);
    mux2x1_32      RF_wa_mux(ir20_16, ir15_11, rf_wa_s, rf_wa);
    regfile_32x32  RF(clk, rst,
                      ir25_21, rf_r1d, rf_r1e,
                      ir20_16, rf_r2d, rf_r2e,
                      rf_wa, rf_wd, rf_we);
    signext_16_32  SE(ir15_0, ir15_0_se);
    mux2x1_32      ADD_mux(rf_r2d, ir15_0_se, add2_s, add2);
    adder_32       ADD(add1, add2, add_sum);
    mem_1024x32    IM(clk, pc_out[11:2], ir, 1, 0, 0, 0);
    mem_1024x32    DM(clk, dm_a[11:2], dm_rd, dm_re,
                      dm_a[11:2], dm_wd, dm_we);
endmodule
```

PARTICIPATION ACTIVITY 7.15.1: Base MIPSzy's top-level structure in Verilog.

1) How many bits is the dm_we wire?

- ○ 1
- ○ 32

2) How many bits is the ir wire?

- ○ 1
- ○ 32

3) ir31_26 is defined using an _____ statement.
   - assign
   - component instantiation

4) The PC_inc component instance connects to wire pc_out and wire ____.
   - pc_inc4
   - clk

5) Wire pc_inc4 connects component instance PC_inc to component instance ____.
   - pc_out
   - PC

6) How many 32-bit 2x1 muxes are instantiated?
   - 1
   - 3

CTRL: Control logic

MIPSzy's CTRL component carries out the control logic actions for a given instruction. In the Verilog above, CTRL was insta component of type mipszy_ctrl, which can be described in Verilog as a module having combinational logic behavior, shown inputs are ir31_26 and ir5_0, which are the opcode and function fields of an instruction. CTRL's outputs are numerous singl control the other components in the design, like dm_we which enables writing to the DM component, or add2_s which cont front of the ADD component's second input. The "always" represents a task, which executes if any input changes (indicated task determines the instruction (lw, sw, addi, or add) from the input values, and assigns the output control signals with valu that instruction.

Figure 7.15.3: Verilog for MIPSzy's control logic component CTRL.

```
`timescale 1ns / 1ps
module mipszy_ctrl(ir31_26, ir5_0,
                   rf_wd_s, rf_wa_s, rf_we, rf_r1e, rf_r2e,
                   add2_s,
                   dm_we, dm_re);

   input [5:0] ir31_26, ir5_0;
   output reg rf_wd_s, rf_wa_s, rf_we, rf_r1e, rf_r2e;
   output reg add2_s;
   output reg dm_we, dm_re;

   always @* begin
      if (ir31_26 == 'b100011) begin // lw
         rf_wd_s = 0;
         rf_wa_s = 1;
         rf_we = 1;
         rf_r1e = 1;
         rf_r2e = 0;
         add2_s = 0;
         dm_we = 0;
         dm_re = 1;
      end
      else if (ir31_26 == 'b101011) begin // sw
         rf_wd_s = 0;
         rf_wa_s = 0;
         rf_we = 0;
         rf_r1e = 1;
         rf_r2e = 1;
         add2_s = 0;
         dm_we = 1;
         dm_re = 0;
      end
      else if (ir31_26 == 'b001000) begin // addi
         rf_wd_s = 1;
         rf_wa_s = 1;
         rf_we = 1;
         rf_r1e = 1;
         rf_r2e = 0;
         add2_s = 0;
         dm_we = 0;
         dm_re = 0;
      end
      else if (ir31_26 == 'b000000 &amp;&amp;
               ir5_0   == 'b100000) begin // add
         rf_wd_s = 1;
         rf_wa_s = 0;
```

```
            rf_we = 1;
            rf_r1e = 1;
            rf_r2e = 1;
            add2_s = 1;
            dm_we = 0;
            dm_re = 0;
        end
        else begin
            rf_wd_s = 0;
            rf_wa_s = 0;
            rf_we = 0;
            rf_r1e = 0;
            rf_r2e = 0;
            add2_s = 0;
            dm_we = 0;
            dm_re = 0;
        end
    end
endmodule
```

PARTICIPATION ACTIVITY 7.15.2: Base MIPSzy's control logic.

1) Is dm_we an input or output of the control logic?
   - Input
   - Output

2) How does the description detect an addi instruction?
   - (ir31_26 == 'b100011)
   - (ir31_26 == 'b001000)
   - (ir31_26 == 'b000000 && ir5_0 == 'b100000)

3) What does the description do if the opcode isn't recognized?
   -

Nothing

- Assigns all outputs with 0's
- Exit

Register, memory, and register file

Below are Verilog descriptions for a 32-bit register (used for PC), a 1024x32 memory (used for IM and DM), and a 32x32 reg for RF) component.

The register is only updating on rising clock edges (i.e., is "synchronous"). If rst is 1, the register is reset to 0, and otherwise input D.

The memory is described using an array of 1024 registers ([0:1023]). The memory has a task to handle writes synchronous on a rising clock edge), and another task to handle reads asynchronously (executes if any input changes, via "@*").

The register file is described using an array of 32 registers ([0:31]). The register file has a task to handle writes synchronous only on a rising clock edge), and another task to handle reads asynchronously (executes if any input changes, via "@*"). The handles both read ports.

Figure 7.15.4: Verilog for 32-bit register used for PC.

```
`timescale 1ns / 1ps
module register_32(clk, rst, ld, D, Q);
   input clk, rst;
   input ld;
   input [31:0] D;
   output reg [31:0] Q;

   always @(posedge clk) begin
      if (rst) begin
         Q = 0;
      end
      else if (ld) begin
         Q = D;
      end
   end
endmodule
```

Figure 7.15.5: Verilog for 1024x32 memory used for instruction and data memories.

```
`timescale 1ns / 1ps
module mem_1024x32(clk, ra, rd, re, wa, wd, we);
   input clk;
   input [9:0] ra, wa;
   input re, we;
   output reg [31:0] rd;
   input [31:0] wd;

   reg [31:0] memory [0:1023];

   always @(posedge clk) begin
     if (we) begin
         memory[wa] = wd;
      end
   end

   always @* begin
      if (re) begin
         rd = memory[ra];
      end
      else begin
         rd = 0;
      end
   end
endmodule
```

Figure 7.15.6: Verilog for 32x32 register file with 2 read ports and 1 write port.

```
`timescale 1ns / 1ps
module regfile_32x32(clk, rst,
                     r1a, r1d, r1e,
                     r2a, r2d, r2e,
                     wa, wd, we);
   input clk, rst;
   input [4:0] r1a, r2a, wa;
   input r1e, r2e, we;
   output reg [31:0] r1d, r2d;
   input [31:0] wd;

   integer i;
   reg [31:0] registers [0:31];

   // Write procedure
   always @(posedge clk) begin
      if (rst) begin
         for (i = 0; i < 32; i = i+1) begin
            registers[i] = 0;
         end
      end
      else if (we) begin
         registers[wa] = wd;
      end
   end

   // Read ports procedure
   always @* begin
      // Read port 1
      if (r1e) begin
         r1d = registers[r1a];
      end
      else begin
         r1d = 0;
      end

      // Read port 2
      if (r2e) begin
         r2d = registers[r2a];
      end
      else begin
         r2d = 0;
      end
   end
endmodule
```

PARTICIPATION ACTIVITY 7.15.3: Base MIPSzy's storage components: PC, memory, and RF.

1) For register_32 , what happens to the register's output value Q if rst changes from 0 to 1 and clk stays at 0?
   - ○ Q = 0
   - ○ Nothing

2) How many component instances in the base MIPSzy's top-level structure are a 32-bit register?
   - ○ 1
   - ○ 3

3) For mem_1024x32, on a positive clock edge, what value of input we causes the memory's array to be updated?
   - ○ 0
   - ○ 1

4) For mem_1024x32, how many bits is input wa?
   - ○ 10
   - ○ 32

5) For mem_1024x32, is the read synchronous?
   - ○ Yes
   - ○

No

6) For regfile_32x32, does the task handle the case of both r1e and r2e being 1's?
   - Yes
   - No

7) For regfile_32x32, what is output on read port 1 if r1e is 0?
   - 0
   - Nothing

Other combinational components

Below are Verilog descriptions for the other combinational components in the base MIPSzy's design.

Figure 7.15.7: Verilog for 32-bit adder.

```
`timescale 1ns / 1ps
module adder_32(A, B, S);
   input [31:0] A, B;
   output reg [31:0] S;

   always @(A, B) begin
      S = A + B;
   end
endmodule
```

Figure 7.15.8: Verilog for 32-bit incrementer that increments by 4.

```
`timescale 1ns / 1ps
module inc4_32(A, S);
   input [31:0] A;
   output reg [31:0] S;

   always @(A) begin
      S = A + 4;
   end
endmodule
```

Figure 7.15.9: Verilog for 32-bit 2x1 mux.

```
`timescale 1ns / 1ps
module mux2x1_32(I1, I0, s, D);
   input [31:0] I1, I0;
   input s;
   output reg [31:0] D;

   always @(I1, I0, s)
   begin
      if (!s) begin
         D = I0;
      end
      else begin
         D = I1;
      end
   end
endmodule
```

Figure 7.15.10: Verilog for 16-bit to 32-bit sign extension.

```
`timescale 1ns / 1ps
module signext_16_32(I, O);
   input signed [15:0] I;
   output reg [31:0] O;

   always @(I) begin
      O = I;
   end
endmodule
```

PARTICIPATION ACTIVITY 7.15.4: Base MIPSzy's other combinational components.

Match the combinational component with the Verilog behavior.

mux2x1_32 signext_16_32 inc4_32 adder_32

S = A + B;

S = A + 4;

if (!s) begin D = I0; end else begin D = I1;

O = I;

Reset

Testbench

A designer can test the base MIPSzy's Verilog using a testbench. The testbench instantiates the base MIPSzy's top-level m naming the instance MIPSzy_0. The testbench initializes that module's IM component's first four words with four machine one DM word (at address 5000) with the value 10 (an "initial" task runs only once, at the start of the Verilog's execution). Th another "initial" task to assign the rst input with 1 for one clock cycle. The testbench has an "always" task that simply pulse repeatedly. That testbench will cause the four instructions in IM to execute on the MIPSzy's design. A designer might set up simulator to view DM's values, to see if DM gets updated as expected (the 10 should be doubled to 20).

Figure 7.15.11: Simple testbench for MIPSzy processor.

```
`timescale 1ns / 1ps
module MIPSzy_TB();
   reg clk_tb, rst_tb;

   localparam CLK_PERIOD = 20;

   MIPSzy MIPSzy_0(clk_tb, rst_tb);

   initial begin
      // initialize instruction memory
      MIPSzy_0.IM.memory[0] = 'b00100000000011100001001110001000; // addi $t6, $zero, 5000
      MIPSzy_0.IM.memory[1] = 'b10001101110010000000000000000000; // lw $t0, 0($t6)  # Load from
DM[5000]
      MIPSzy_0.IM.memory[2] = 'b00000001000010000100100000100000; // add $t1, $t0, $t0 # Double the
values
      MIPSzy_0.IM.memory[3] = 'b10101101110010010000000000000000; // sw $t1, 0($t6)  # Store to
DM[5000]

      MIPSzy_0.DM.memory[(5000-4096)/4] = 10;
   end


   // Generate clock
   always begin
      clk_tb = 0;
      #(CLK_PERIOD / 2);
      clk_tb = 1;
      #(CLK_PERIOD / 2);
   end

   initial begin
      rst_tb = 1;
      #CLK_PERIOD rst_tb = 0;
   end

endmodule
```

PARTICIPATION ACTIVITY 7.15.5: Base MIPSzy's testbench.

1) The number of clock cycles that the testbench should execute is ____.

- 10
- unlimited

Provide feedback on this section