Computer Organization

Verilog Review

Presenter: Kai-Chieh Hsu

Outline

- Verilog Intrudution
  - What, Why, How
- Combinational logic & Sequential logic
- Examples
  - ALU
  - Program Counter
  - ALU vs Program Counter
  - Module instance CPU
  - Mux, Shift left, Pipeline Register
  - bit rearragement
- Other Resource

Verilog Intruduction

- What is verilog
  - Verilog is hardware description language (HDL), can be used to describe gate logic and digital system, which is the fundamental of modern CPU.
- Why using verilog
  - Implementing a CPU using Verilog can solidify your knowledge from lectures, as the process requires a thorough understanding of how each CPU component works and behaves.
- Our goal
  - Help you review how to write verilog via CPU components
  - The basic verilog syntax and HDL concept.

ALU - Arithmetic logic Unit

- Combinational Logic
  - outputs only depend on input
- Module Interface
- wire & register
- signed, [3:0], [31:0]
- assign
- always block
  - trigger condition
- begin, end
- case statement
- `=` blocking assignment

```
module ALU (
    input [3:0] ALUctl,
    input signed [31:0] A,B,
    output reg signed [31:0] ALUOut,
    output zero
);
    assign zero = (ALUOut == 0);
    always @(*) begin
        case (ALUctl)
            0: ALUOut = A & B;
            1: ALUOut = A | B;
            2: ALUOut = A + B;
            6: ALUOut = A - B;
            default: ALUOut = 0;
        endcase
    end
endmodule
```

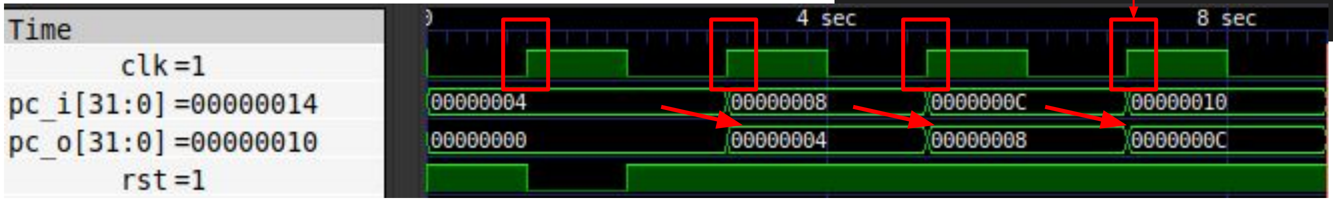
Program Counter

- Sequencial Logic
  - clock, reset
  - output might depends on internal state
- initial statement
- always block
  - posedge, negedge trigger
- <= non-blocking assignment

```
module PC (
    input clk,
    input rst,
    input [31:0] pc_i,
    output reg [31:0] pc_o
);

initial begin
    pc_o = 0;
end

always @(posedge clk, negedge rst) begin
    if (~rst)
        pc_o <= 0;
    else
        pc_o <= pc_i;
end
endmodule
```



ALU & Program Counter

Conbinational
Sequencial logic

```
module ALU (
    input [3:0] ALUctl,
    input signed [31:0] A,B,
    output reg signed [31:0] ALUOut,
    output zero
);
    assign zero = (ALUOut == 0);
    always @(*) begin
        case (ALUctl)
            0: ALUOut = A & B;
            1: ALUOut = A | B;
            2: ALUOut = A + B;
            6: ALUOut = A - B;
            default: ALUOut = 0;
        endcase
    end
endmodule
```

blocking
non-blocking
Assignment

```
module PC (
    input clk,
    input rst,
    input [31:0] pc_i,
    output reg [31:0] pc_o
);

initial begin
    pc_o = 0;
end

always @(posedge clk, negedge rst)
begin
    if (~rst)
        pc_o <= 0;
    else
        pc_o <= pc_i;
end

endmodule
```

Module Instance CPU

- Module Instance
    - <Module name> <Instance Name>
    - connect wire .clk(clk)

```
module SingleCycleCPU (
    input clk,
    input start,
    output signed [31:0] r [0:31]
);

wire [31:0] pc_i, pc_o, pc_a4, pc_aimm;
wire [31:0] inst;

PC m_PC(
    .clk(clk),
    .rst(start),
    .pc_i(pc_i),
    .pc_o(pc_o)
);
```

```
module PC (
    input clk,
    input rst,
    input [31:0] pc_i,
    output reg [31:0] pc_o
);
```

Mux, Shift left, Pipeline Register

```
module Mux2to1 #(
    parameter size = 32
)
(
    input sel,
    input signed [size-1:0] s0,
    input signed [size-1:0] s1,
    output signed [size-1:0] out
);
    assign out = (sel)? s1: s0;
endmodule
```

```
module ShiftLeftOne (
    input signed [31:0] i,
    output signed [31:0] o
);
    assign o = i << 1;
endmodule
```

```
module PipeReg #(
    size = 1
) (
    input clk,
    input rst,
    input [size-1:0] i,
    output reg [size-1:0] o
);
always @(posedge clk) begin
    if(~rst) o <= 0;
    else     o <= i;
end
endmodule
```

Bits Rearrangement

31-27	26-25	24-20	19-15	14-12	11-7		6-2	1-0
offset[12|10:5]		rs2	rs1	000	offset[4:1|11]		11000	11

Format beq rs1,rs2,offset

Description Take the branch if registers rs1 and rs2 are equal.

Implementation if (x[rs1] == x[rs2]) pc += sext(offset)

```
input [31:0] inst,
output reg signed [31:0] imm
// beq
imm <= {{20{inst[31]}}, inst[7], inst[30:25], inst[11:8], 0};
```

[31:12] [11] [10:5] [4:1] [0]

Other Verilog Resource

- Ask GPT: https://chatgpt.com/
- HDLBits: https://hdlbits.01xz.net/wiki/Main_Page
- Search on the internet by yourself.
- Textbook: Digital Design, mano
- NYCU OCW: https://ocw.nycu.edu.tw/?post_type=course_page&p=83423

Thanks!

Q&A