

# Lecture 15: Register Transfer Language

COS 471a, COS 471b / ELE 375

Computer Architecture and Organization

Princeton University  
Fall 2004

Prof. David August

1

## Administrivia

- Reading Chapters 1-7,A-C
- Project 2
  - On course page this weekend
  - Due January 8th

2

## RTL (Register Transfer Language)

- Designing processors at the gate level is difficult
- Use a higher-level language

RTL: a language for describing the behavior of computers  
in terms of step-wise register contents

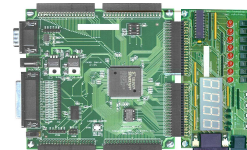
6

## RTL Tools

RTL, just like (most) programming languages:

- Precise and unambiguous
- Programmer must debug
- Code can be checked automatically for certain properties
  - Type checking
  - Check RTL model against more abstract state machine
- Tools can process the code
  - RTL à simulator
  - RTL à processor

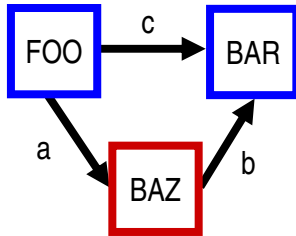
- RTL à



7

## Verilog

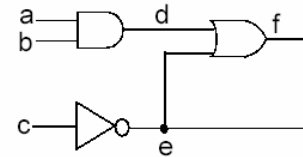
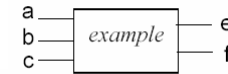
- Verilog and VHDL are both RTLs.
- We will look at Verilog - becoming the industry standard
- Designs consist of modules and their connections



8

## Example

Modules have initial, **continuous**, and always blocks

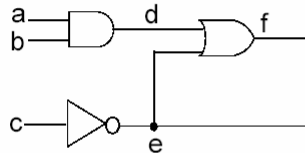


```
module example(a, b, c, f, e);  
  input a, b, c;  
  output f, e;  
  wire d;  
  and g1(d, a, b);  
  not g2(e, c);  
  or g3(f, d, e);  
endmodule
```

Youpyo Hong, Dongguk University

9

## Modified Example



```
module example(a, b, c, f, e);  
  input a, b, c;  
  output f, e;  
  assign d = a & b;  
  assign e = ~c;  
  assign f = d | e;  
endmodule
```

Youpyo Hong, Dongguk University

- Use assign statement for combinational circuit
- Note the bit-wise operators (implies wire width...)

10

## Wires

- Wires need not be declared unless it is multiple wires

```
module example(b, c);  
  input b;  
  output c;  
  wire a; // not necessary  
  
  not g1(a, b);  
  not g2(c, a);  
endmodule
```

11

## Wires

- Wires need not be declared unless it is multiple wires

```
module example(b, c, d);
  input [3:0]b, [3:0]c;
  output d;
  wire [3:0]a; // necessary

  a = b & c;
  d = a[2] | a[1] | a[0];
endmodule
```

12

## Register Transfer Language?

- Variables correspond to the hardware registers
- "always" blocks

```
Module D_FlipFlop(Q, D, CLK);
  output Q;
  reg Q;
  input D, CLK;
  always @(negedge CLK) // Sensitivity List
  begin // Not needed for 1 statement
    Q <= D; // Note: No assign
  end
endmodule
```

13

## Control

### Add a RESET

```
Module D_FlipFlop(Q, D, CLK, RESET);
  output Q;
  reg Q;
  input D, CLK, RESET;
  always @(negedge CLK or posedge RESET)
  if(RESET)
    Q <= 1'b0; // 1 bit binary value 0
  else // why should all ifs have else?
    Q <= D;
endmodule
```

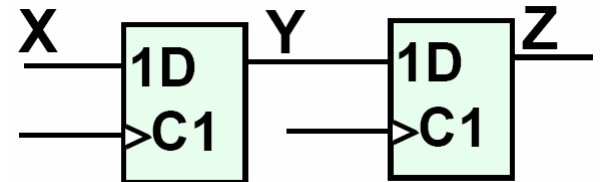
14

## Assignment

a = b            Do Sequentially (blocking, sequentially)  
A <= b           Do RHS first (nonblocking, grab at t=0)

Which 3 are Shift Registers?

Z<=Y; Y<=X;  
Y<=X; Z<=Y;  
Z=Y; Y=X;  
Y=X; Z=Y;



15

## Misc.

- Register file for MIPS:

```
reg [31:0] register_file[0:31];
```

- We are just scratching the surface
- Looks like C
  - Operators
  - switch-, for-, while-statements...
- Careful: some cases not synthesizable

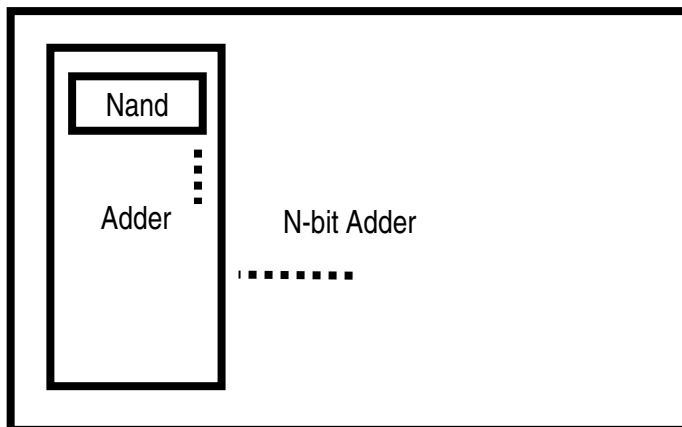
16

## Example: 4 to 1 Mux

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
    output out;
    input i0, i1, i2, i3;
    input s1, s0;
    reg out;
    always @(s1 or s0 or i0 or i1 or i2 or i3)
    begin
        case ({s1, s0})
            2'b00: out = i0;
            2'b01: out = i1;
            2'b10: out = i2;
            2'b11: out = i3;
            default: out = 1'bx; // x is don't care
        endcase
    end
endmodule
```

17

## Hierarchical Design



18

## Summary

- RTL programs
  - Declarations: modules, wires, registers, inputs, outputs
  - Module parts: initial, continuous, always blocks
  - Combinational assignments (assign)
  - Register transfer statements (blocking: =)
  - Control transfer statements (if-else)
- Verilog looks like C
- Build things hierarchically
- Use tools to create simulator, hardware

19