Introduction to Computer Architecture

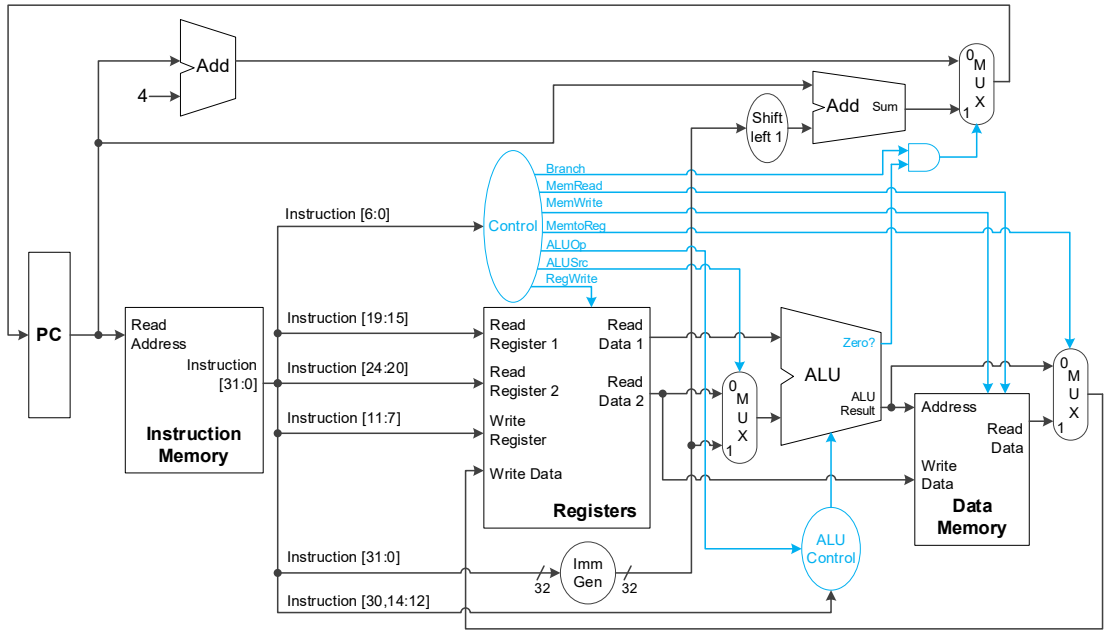
How Real Processors Are Made

Hyungmin Cho

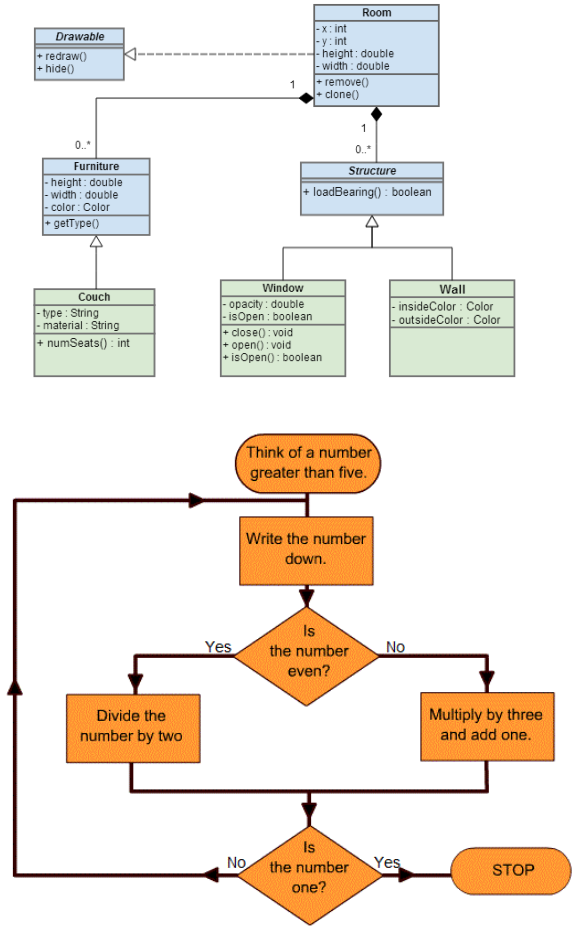
Department of Computer Science and Engineering
Sungkyunkwan University

Designing a Processor

- The processor datapath diagram is just a design blueprint



≈



- How to make a real CPU ?

Language for Hardware Designs

- Hardware **Description** Language (HDL)
  - Verilog
  - VHDL

Verilog Example:

```
module test ( clk , input_1, input_2, out);
  input clk;
  input [31:0] input_1;
  input [31:0] input_2;
  output reg [31:0] counter;

  wire [31:0] counter_new;

  always @(*) begin
    counter_new = counter;
    if( input_1 == input_2 )
     counter_new = counter + 1;
  end

  always @(posedge clk) begin
    counter <= counter_new;
  endz
endmodule
```

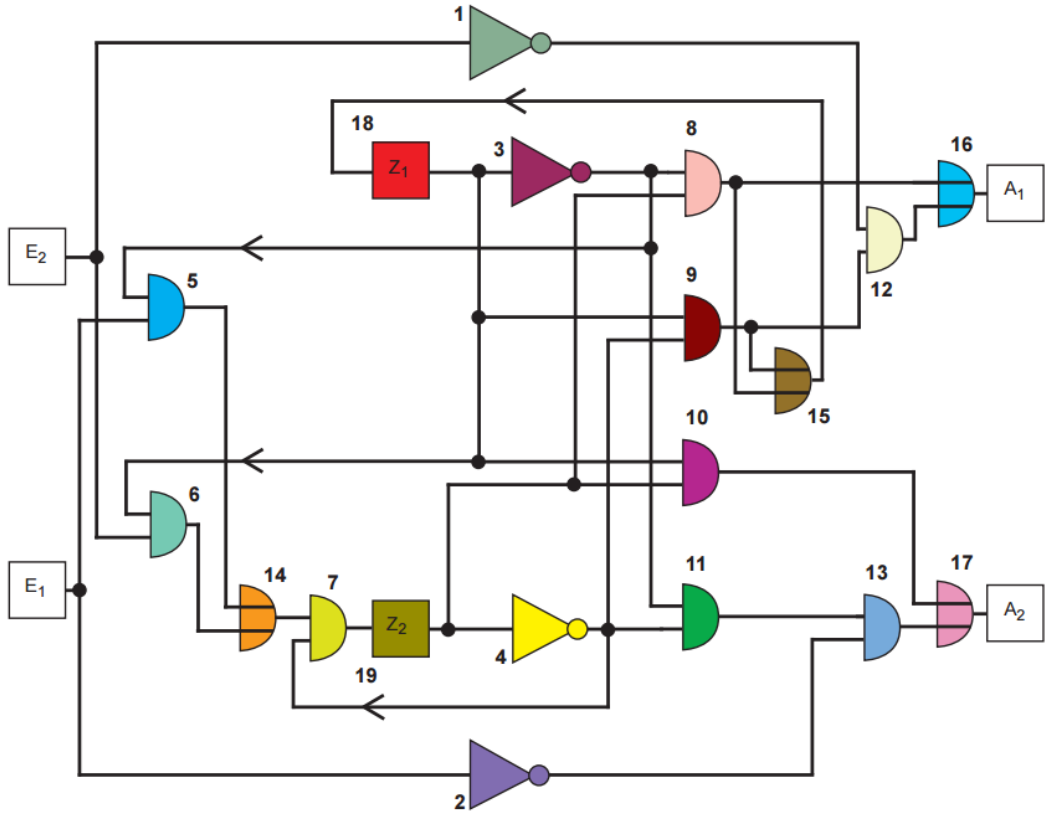
Combinational logic →

Sequential logic →

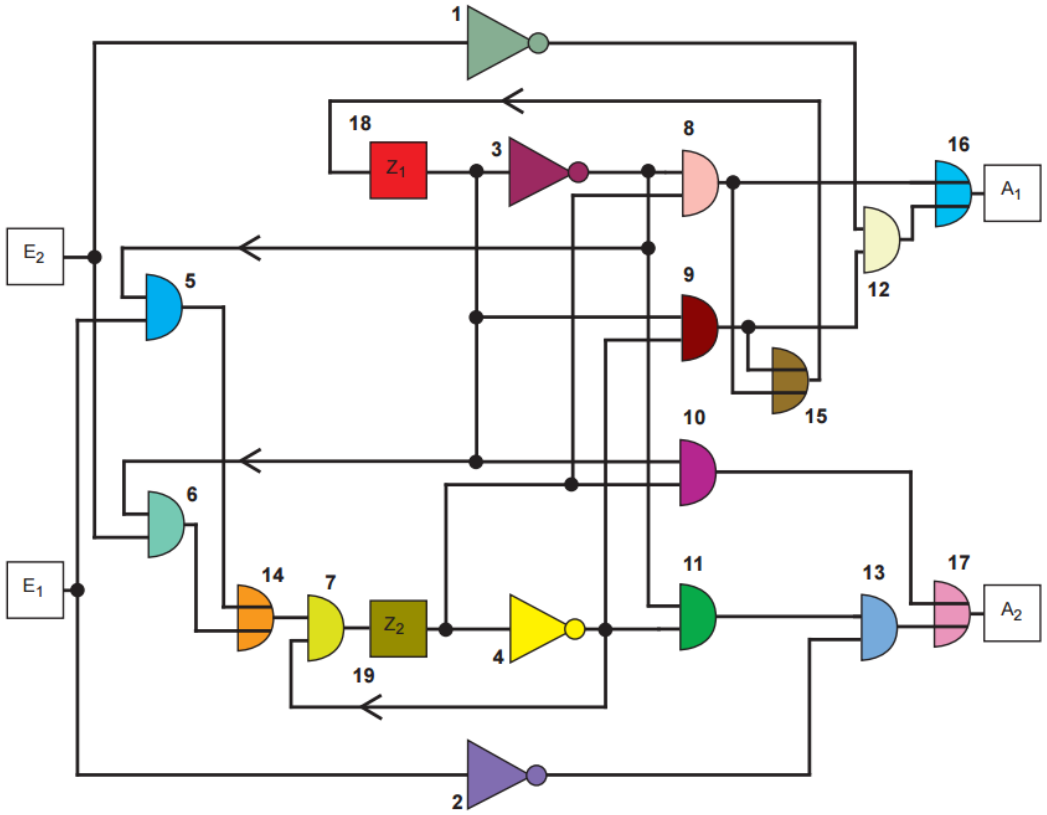
Synthesis

- High level language → Logic gates

```
module test ( clk , input_1, input_2, out);
  input clk;
  input [31:0] input_1;
  input [31:0] input_2;
  output reg [31:0] counter;

  wire [31:0] counter_new;

  always @(*) begin
    counter_new = counter;
    if( input_1 == input_2 )
     counter_new = counter + 1;
  end

  always @(posedge clk) begin
    counter <= counter_new;
  end
endmodule
```



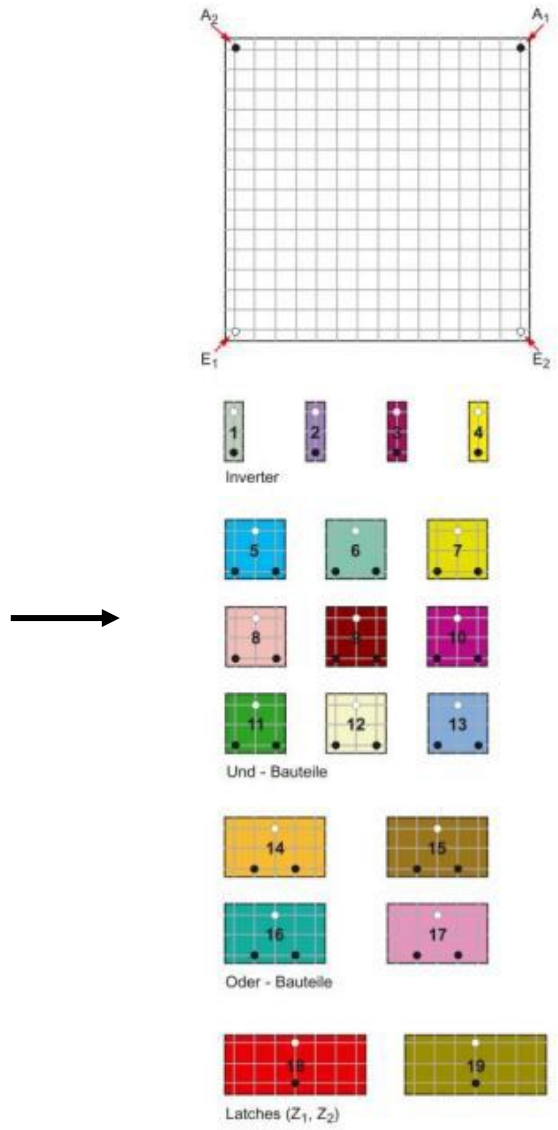
Synthesis Result: Netlist

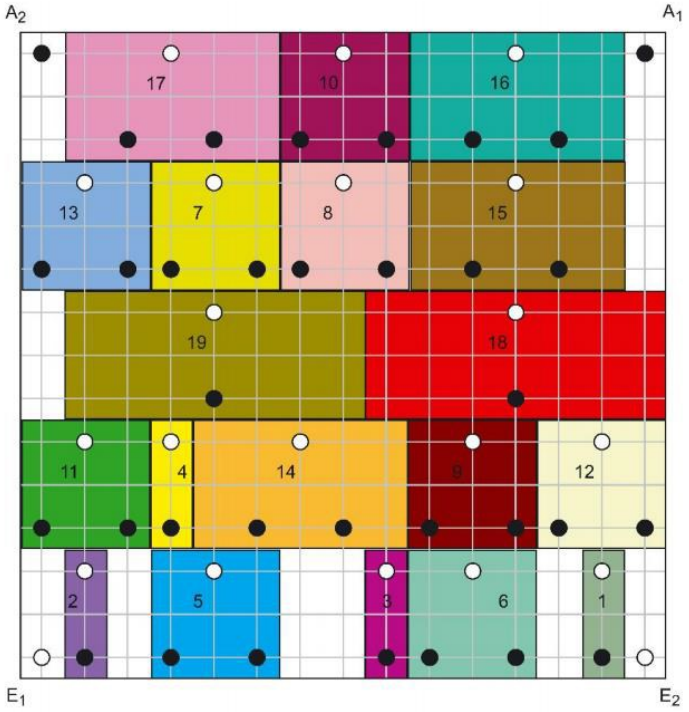



```
input clk, reset;
output [7:0] acc;
wire [7:0] in;
wire clk, reset;
wire [7:0] acc;
wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
DFFPOSX1 \acc_reg[0] (.CLK (clk), .D (n_5), .Q (acc[0]));
DFFPOSX1 \acc_reg[1] (.CLK (clk), .D (n_9), .Q (acc[1]));
DFFPOSX1 \acc_reg[2] (.CLK (clk), .D (n_13), .Q (acc[2]));
DFFPOSX1 \acc_reg[3] (.CLK (clk), .D (n_17), .Q (acc[3]));
DFFPOSX1 \acc_reg[4] (.CLK (clk), .D (n_21), .Q (acc[4]));
DFFPOSX1 \acc_reg[5] (.CLK (clk), .D (n_25), .Q (acc[5]));
DFFPOSX1 \acc_reg[6] (.CLK (clk), .D (n_29), .Q (acc[6]));
DFFPOSX1 \acc_reg[7] (.CLK (clk), .D (n_31), .Q (acc[7]));
NOR2X1 g146(.A (n_30), .B (reset), .Y (n_31));
XOR2X1 g148(.A (n_26), .B (n_3), .Y (n_30));
NOR2X1 g149(.A (n_28), .B (reset), .Y (n_29));
INVX1 g150(.A (n_27), .Y (n_28));
FAX1 g151(.A (acc[6]), .B (in[6]), .C (n_22), .YC (n_26), .YS (n_27));
NOR2X1 g153(.A (n_24), .B (reset), .Y (n_25));
INVX1 g154(.A (n_23), .Y (n_24));
FAX1 g155(.A (acc[5]), .B (in[5]), .C (n_18), .YC (n_22), .YS (n_23));
NOR2X1 g157(.A (n_20), .B (reset), .Y (n_21));
INVX1 g158(.A (n_19), .Y (n_20));
FAX1 g159(.A (acc[4]), .B (in[4]), .C (n_14), .YC (n_18), .YS (n_19));
NOR2X1 g161(.A (n_16), .B (reset), .Y (n_17));
INVX1 g162(.A (n_15), .Y (n_16));
FAX1 g163(.A (acc[3]), .B (in[3]), .C (n_10), .YC (n_14), .YS (n_15));
NOR2X1 g165(.A (n_12), .B (reset), .Y (n_13));
INVX1 g166(.A (n_11), .Y (n_12));
FAX1 g167(.A (acc[2]), .B (in[2]), .C (n_6), .YC (n_10), .YS (n_11));
NOR2X1 g169(.A (n_8), .B (reset), .Y (n_9));
INVX1 g170(.A (n_7), .Y (n_8));
FAX1 g171(.A (acc[1]), .B (in[1]), .C (n_1), .YC (n_6), .YS (n_7));
NOR2X1 g173(.A (n_4), .B (reset), .Y (n_5));
INVX1 g174(.A (n_2), .Y (n_4));
HAX1 g175(.A (in[0]), .B (acc[0]), .YC (n_1), .YS (n_2));
OAI21X1 g176(.A (in[7]), .B (acc[7]), .C (n_0), .Y (n_3));
NAND2X1 g177(.A (acc[7]), .B (in[7]), .Y (n_0));
```

Physical Design - Placement

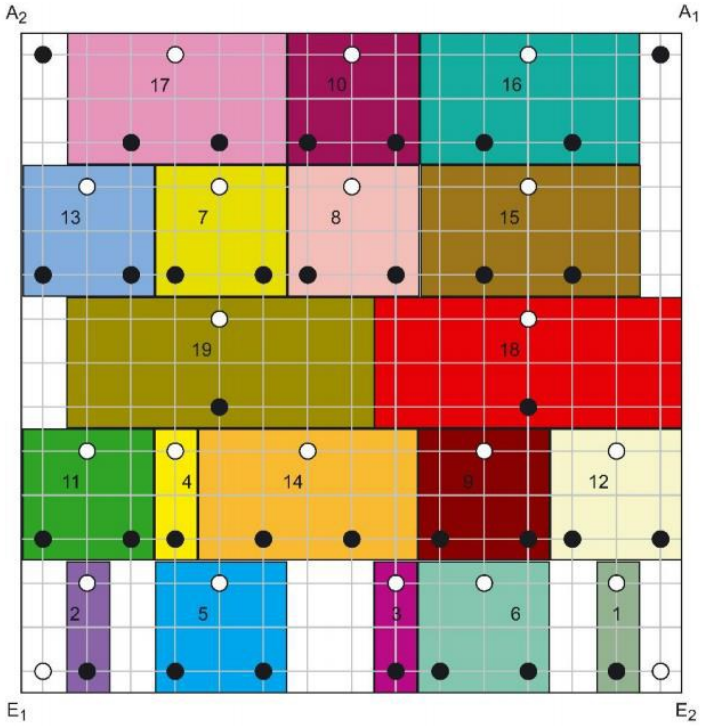
```
input clk, reset;
output [7:0] acc;
wire [7:0] in;
wire clk, reset;
wire [7:0] acc;
wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
DFFPOSX1 \acc_reg[0] (.CLK (clk), .D (n_5), .Q (acc[0]));
DFFPOSX1 \acc_reg[1] (.CLK (clk), .D (n_9), .Q (acc[1]));
DFFPOSX1 \acc_reg[2] (.CLK (clk), .D (n_13), .Q (acc[2]));
DFFPOSX1 \acc_reg[3] (.CLK (clk), .D (n_17), .Q (acc[3]));
DFFPOSX1 \acc_reg[4] (.CLK (clk), .D (n_21), .Q (acc[4]));
DFFPOSX1 \acc_reg[5] (.CLK (clk), .D (n_25), .Q (acc[5]));
DFFPOSX1 \acc_reg[6] (.CLK (clk), .D (n_29), .Q (acc[6]));
DFFPOSX1 \acc_reg[7] (.CLK (clk), .D (n_31), .Q (acc[7]));
NOR2X1 g146(.A (n_30), .B (reset), .Y (n_31));
XOR2X1 g148(.A (n_26), .B (n_3), .Y (n_30));
NOR2X1 g149(.A (n_28), .B (reset), .Y (n_29));
INVX1 g150(.A (n_27), .Y (n_28));
FAX1 g151(.A (acc[6]), .B (in[6]), .C (n_22), .YC (n_26), .YS (n_27));
NOR2X1 g153(.A (n_24), .B (reset), .Y (n_25));
INVX1 g154(.A (n_23), .Y (n_24));
FAX1 g155(.A (acc[5]), .B (in[5]), .C (n_18), .YC (n_22), .YS (n_23));
NOR2X1 g157(.A (n_20), .B (reset), .Y (n_21));
INVX1 g158(.A (n_19), .Y (n_20));
FAX1 g159(.A (acc[4]), .B (in[4]), .C (n_14), .YC (n_18), .YS (n_19));
NOR2X1 g161(.A (n_16), .B (reset), .Y (n_17));
INVX1 g162(.A (n_15), .Y (n_16));
FAX1 g163(.A (acc[3]), .B (in[3]), .C (n_10), .YC (n_14), .YS (n_15));
NOR2X1 g165(.A (n_12), .B (reset), .Y (n_13));
INVX1 g166(.A (n_11), .Y (n_12));
FAX1 g167(.A (acc[2]), .B (in[2]), .C (n_6), .YC (n_10), .YS (n_11));
NOR2X1 g169(.A (n_8), .B (reset), .Y (n_9));
INVX1 g170(.A (n_7), .Y (n_8));
FAX1 g171(.A (acc[1]), .B (in[1]), .C (n_1), .YC (n_6), .YS (n_7));
NOR2X1 g173(.A (n_4), .B (reset), .Y (n_5));
INVX1 g174(.A (n_2), .Y (n_4));
HAX1 g175(.A (in[0]), .B (acc[0]), .YC (n_1), .YS (n_2));
OAI21X1 g176(.A (in[7]), .B (acc[7]), .C (n_0), .Y (n_3));
NAND2X1 g177(.A (acc[7]), .B (in[7]), .Y (n_0));
```

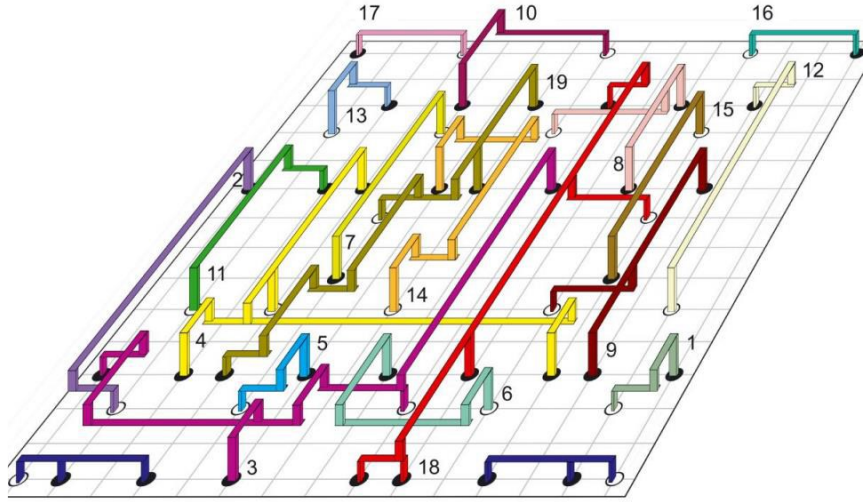


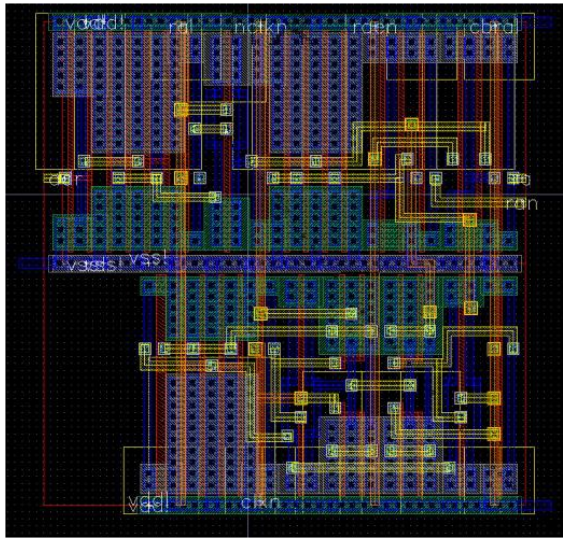


[Image credit: Jens Vygen(U of Bonn)]

Physical Design - Routing

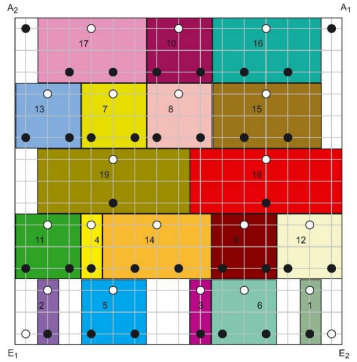


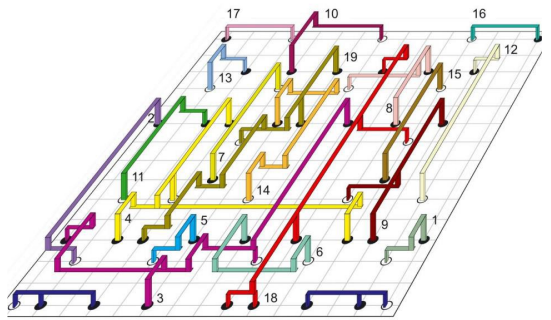


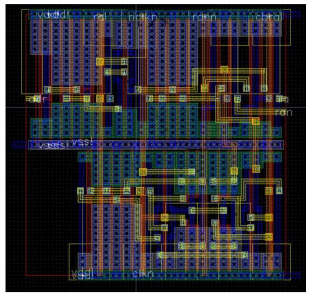


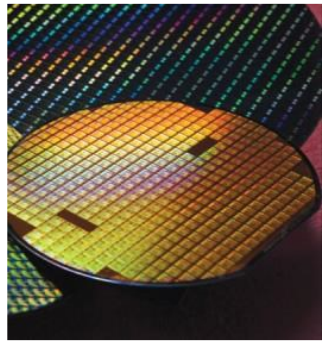
Semiconductor Design and Manufacturing

```
module test ( clk , input_1, input_2, out);
  input clk;
  input [31:0] input_1;
  input [31:0] input_2;
  output reg [31:0] counter;

  wire [31:0] counter_new;

  always @(*) begin
    counter_new = counter;
    if( input_1 == input_2 )
     counter_new = counter + 1;
  end

  always @(posedge clk) begin
    counter <= counter_new;
  endz
endmodule
```

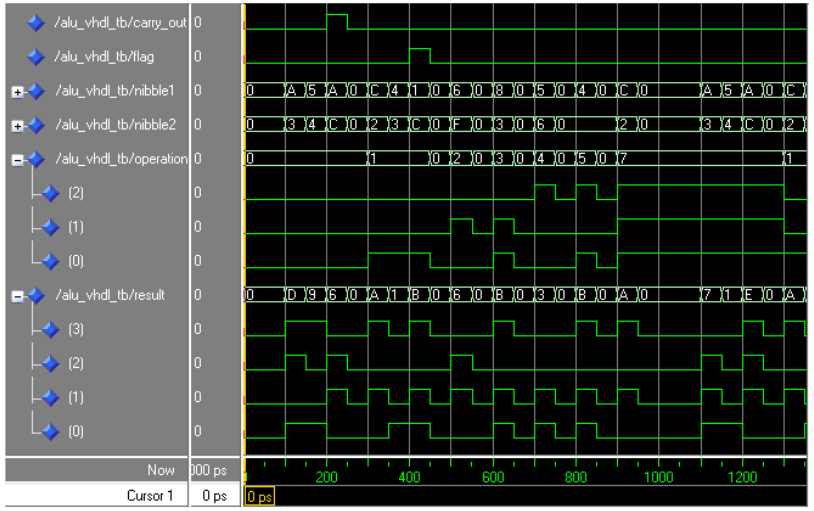




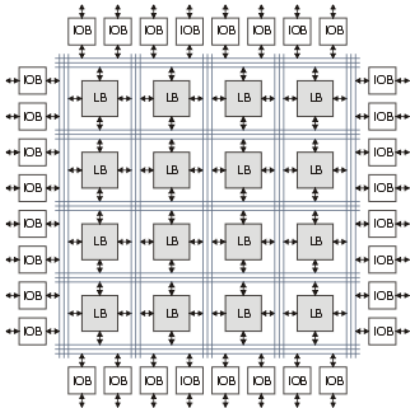




Design Verification / Validation is Important



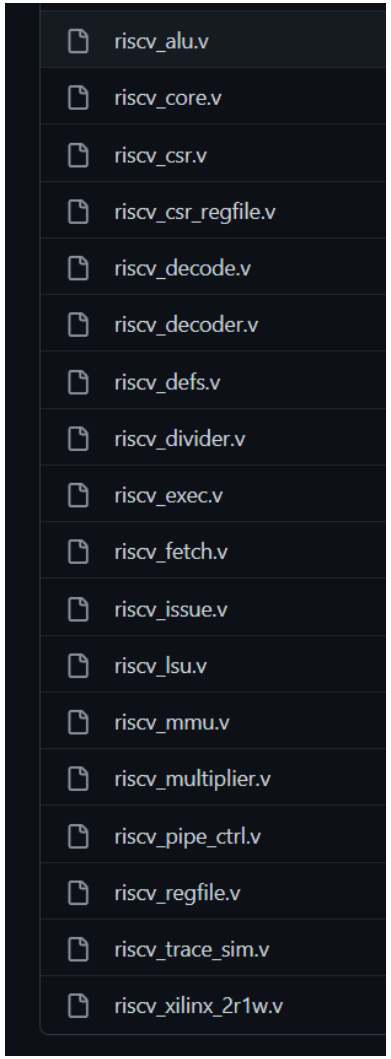
Logic Simulation



FPGA Emulation

RISC-V Processor Implementation Example

- https://github.com/ultraembedded/riscv



ALU in Verilog

```
module riscv_alu (
     input  [  3:0]  alu_op_i
    ,input  [ 31:0]  alu_a_i
    ,input  [ 31:0]  alu_b_i
    ,output [ 31:0]  alu_p_o );
...
    case (alu_op_i)
       `ALU_ADD :  result_r      = (alu_a_i + alu_b_i);
       `ALU_AND :  result_r      = (alu_a_i & alu_b_i);
       `ALU_OR  :  result_r      = (alu_a_i | alu_b_i);
       `ALU_XOR :  result_r      = (alu_a_i ^ alu_b_i);
       `ALU_LESS_THAN : result_r      = (alu_a_i < alu_b_i) ? 32'h1 : 32'h0;
       `ALU_LESS_THAN_SIGNED :
       begin
            if (alu_a_i[31] != alu_b_i[31])
                result_r  = alu_a_i[31] ? 32'h1 : 32'h0;
            else
                result_r  = sub_res_w[31] ? 32'h1 : 32'h0;
       end
...
```