Outline

- Yosys
- ABC
- FreePDK
- Homework

Outline

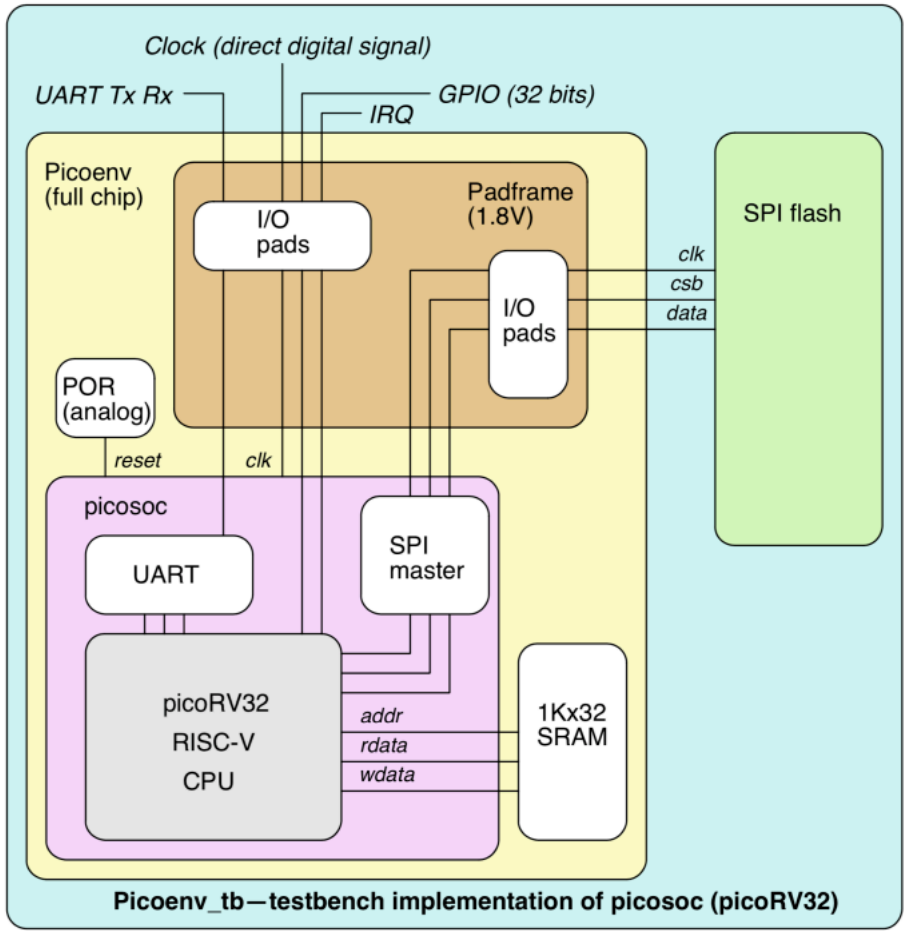
- Yosys
- ABC
- FreePDK
- Homework

Yosys

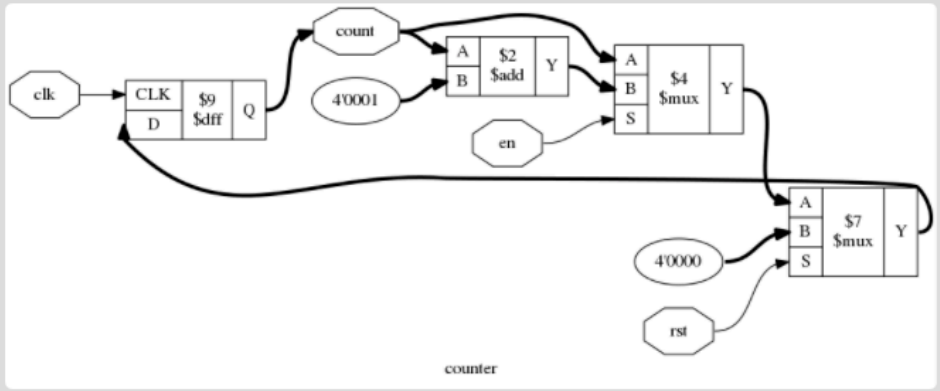
- Open source synthesis suite, framework for Verilog RTL synthesis developed by Clifford Wolf
  - Process almost any synthesizable Verilog-2005 design
  - Mapping to ASIC standard cell libraries
  - Mapping to Xilinx 7-Series and Lattice iCE FPGAs
  - Foundation and front-end for custom flows
- Since it’s open source, it has limited power compared to those closed-source EDA software

https://riscv.org/wp-content/uploads/2017/12/Wed-1142-RISCV-Tim-Edwards.pdf

Yosys



Simple RTL Netlist




```
# read design
read_verilog counter.v
hierarchy -check

# high-level synthesis
proc; opt; fsm; opt; memory; opt
```

Download: show_rtl.ys

```
module counter (clk, rst, en, count);

   input clk, rst, en;
   output reg [3:0] count;

   always @(posedge clk)
      if (rst)
         count <= 4'd0;
      else if (en)
         count <= count + 4'd1;

endmodule
```

Download: counter.v

Example in homework

- cpu.ys

```
### Read verilog files
read_verilog ./codes/cpu.v
read_verilog ./codes/moduleA.v
read_verilog ./codes/moduleB.v
# include all your *.v files here except data_memory.v,
# instruction_memory.v and testbench.v

### Constraints
write_file cpu.constr <<EOT
set_driving_cell BUF_X2
set_load 0.01
EOT

### Map to gate level
synth -top cpu; flatten;
write_verilog -noattr cpu_syn.v

### Map to tech library
dfflibmap -liberty stdcells.lib
abc -constr cpu.constr -D 1000 -liberty stdcells.lib
```

Output

- cpu_syn.v

```
/* Generated by Yosys 0.9+3679 (git sha1 58e8901f, gcc 9.3.0-17ubuntu1~20.04 -fPIC -Os) */

module cpu(i_clk, i_rst_n, i_i_valid_inst, i_i_inst, i_d_valid_data, i_d_data, o_i_valid_ad
data, o_d_w_addr, o_d_r_addr, o_d_MemRead, o_d_MemWrite, o_finish);
  wire _07354_;
  wire _07355_;
  wire _07356_;
  wire _07357_;
  wire _07358_;
  wire _07359_;
  wire _07360_;
  wire _07361_;
  wire _07362_;
  wire _07363_;
  wire _07364_;
  wire _07365_;
  wire _07366_;
  wire _07367_;
  wire _07368_;
  wire _07369_;
  wire _07370_;
  wire _07371_;
  wire _07372_;
```

Outline

- Yosys
- ABC
- FreePDK
- Homework

ABC

- A system for sequential synthesis and verification developed by Berkeley logic synthesis and verification group
- ABC combines scalable logic optimization based on And-Inverter Graphs (AIGs), optimal-delay DAG-based technology mapping for look-up tables and standard cells, and innovative algorithms for sequential synthesis and verification.

ABC result

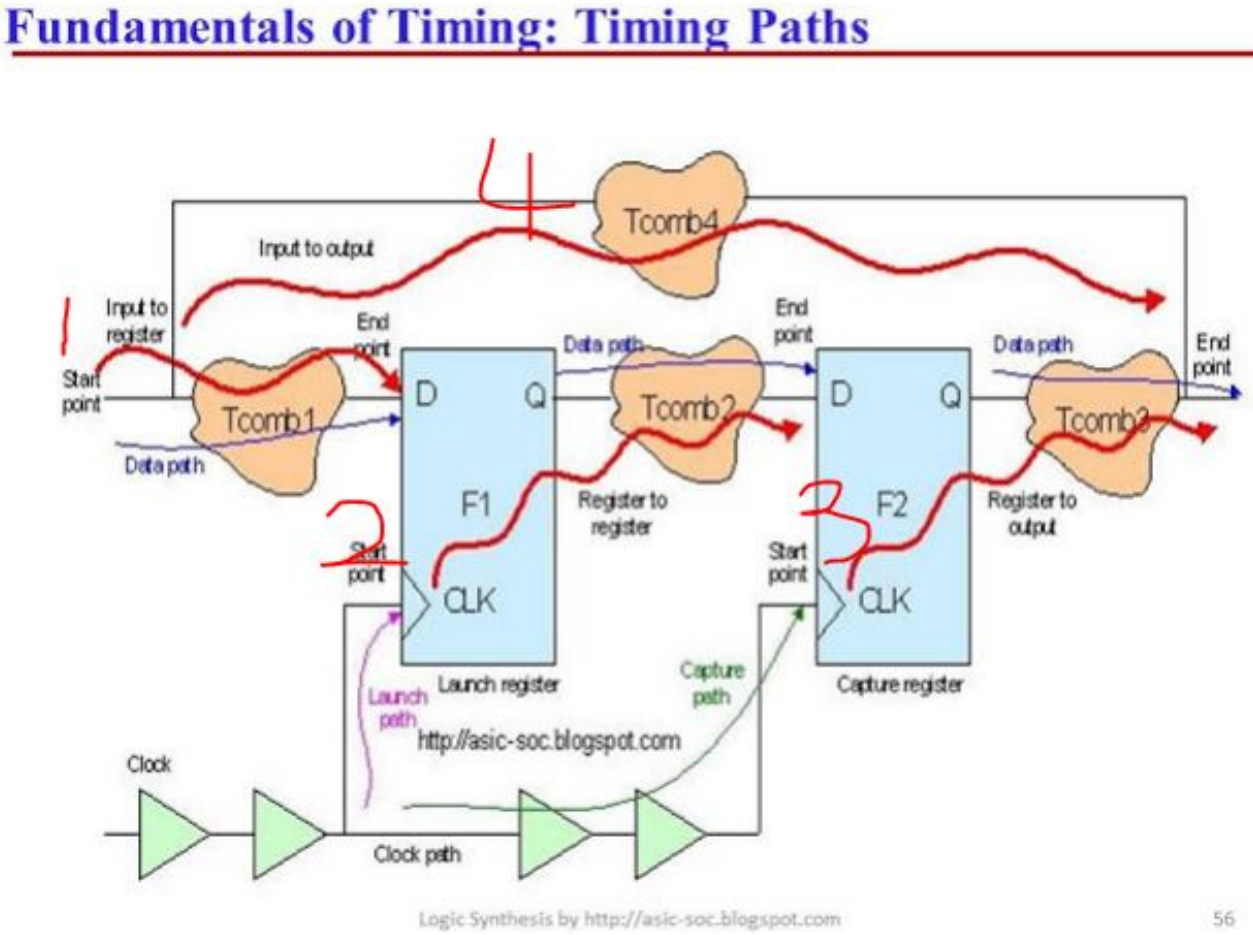
- cpu.yslog

```
2.25. Printing statistics.

=== cpu ===

   Number of wires:                 16492
   Number of wire bits:             36507
   Number of public wires:           1483
   Number of public wire bits:      21498
   Number of memories:                  0
   Number of memory bits:               0
   Number of processes:                 0
   Number of cells:                 20117
     $_ANDNOT_                       5932
     $_AND_                           197
     $_DFFE_PN0N_                       6
     $_DFFE_PN0P_                    2054
     $_DFF_PN0_                      1881
```

```
ABC: WireLoad = "none"  Gates =  13123 ( 14.8 %)   Cap =  3.2 ff (  1.9 %)   Area =    17519.56 ( 87.9 %)  Del
ay =  1091.13 ps  (  5.1 %)
ABC: Path  0 --    2060 : 0    5 pi        A =   0.00  Df =  12.4   -4.2 ps  S =  13.9 ps  Cin =  0.0 ff  Cout
=   9.5 ff  Cmax =   0.0 ff  G =    0
ABC: Path  1 --   10532 : 2    1 XOR2_X1   A =   1.60  Df =  69.3   -8.7 ps  S =  30.0 ps  Cin =  2.3 ff  Cout
=   3.5 ff  Cmax =  25.3 ff  G =  144
ABC: Path  2 --   10535 : 5    2 AOI221_X2 A =   2.93  Df = 132.9  -45.4 ps  S =  41.5 ps  Cin =  3.1 ff  Cout
=   3.3 ff  Cmax =  27.6 ff  G =  101
ABC: Path  3 --   10817 : 4    4 AND4_X2   A =   1.86  Df = 197.9  -68.4 ps  S =  15.7 ps  Cin =  1.6 ff  Cout
=   6.7 ff  Cmax = 120.4 ff  G =  388
ABC: Path  4 --   10867 : 3    1 AOI21_X1  A =   1.06  Df = 213.9  -54.0 ps  S =  21.5 ps  Cin =  1.6 ff  Cout
=   1.7 ff  Cmax =  25.3 ff  G =  102
ABC: Path  5 --   10869 : 3    2 OR3_X2    A =   1.60  Df = 290.1   -5.3 ps  S =  13.2 ps  Cin =  1.6 ff  Cout
=   5.2 ff  Cmax = 121.2 ff  G =  296
ABC: Path  6 --   10870 : 3    2 AOI21_X2  A =   1.86  Df = 318.5  -21.1 ps  S =  17.5 ps  Cin =  3.1 ff  Cout
=   1.7 ff  Cmax =  50.7 ff  G =   52
ABC: Path  7 --   11649 : 3    4 MUX2_X2   A =   2.39  Df = 360.1   -2.3 ps  S =  13.1 ps  Cin =  1.9 ff  Cout
=   7.1 ff  Cmax = 120.8 ff  G =  345
```

```
ABC: Start-point = pi2059 (\EX1_store_addr [9]).  End-point = po2985 (\EX1_result_c [16]).
ABC: + write_blif <abc-temp-dir>/output.blif
```

Path



https://electronics.stackexchange.com/questions/351360/how-to-get-the-timing-report-register-to-register-and-input-to-output-in-sta

Outline

- Yosys
- ABC
- FreePDK
- Homework

FreePDK

- The FreePDK45 kit is an open-source generic process design kit (PDK) (i.e., does not correspond to any real process and cannot be fabricated) that allows researchers and students to experiment with designing in a modern technology node without signing restrictive non-disclosure agreements or paying for licenses.
- The PDK allows you to use commercial full-custom layout tools (e.g., Cadence Virtuoso) to design both analog and digital circuits.

https://www.eda.ncsu.edu/wiki/FreePDK45:Contents

https://ieeexplore.ieee.org/document/4231502

https://github.com/cornell-brg/freepdk-45nm

stdcells.lib

```
/*****************************************************************************************
 Module           : AND2_X2
 Cell Description : Combinational cell (AND2_X2) with drive strength X2
*****************************************************************************************/

cell (AND2_X2) {

  drive_strength      : 2;

  area                : 1.330000;
  pg_pin(VDD) {
      voltage_name : VDD;
      pg_type      : primary_power;
  }
  pg_pin(VSS) {
      voltage_name : VSS;
      pg_type      : primary_ground;
  }


  cell_leakage_power  : 50.353160;

  leakage_power () {
      when           : "!A1 & !A2";
      value          : 40.690980;
  }
```

Outline

- Yosys
- ABC
- FreePDK
- Homework

Similar to HW3

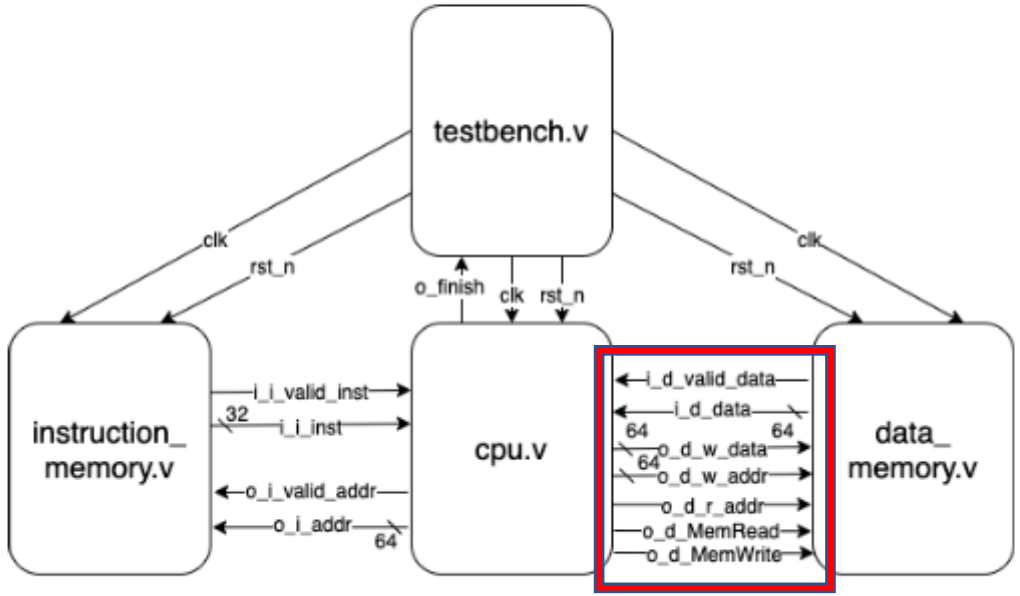
- Except forcing you to use pipeline implementation
- Without pipeline implementation, the latency will be long.
- Data memory
  - Read write simultaneously, pipeline access
- Instruction memory
  - Three cycles latency, pipeline access

Update

And the provided data memory is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_data	Input	64	64-bits data that will be stored
i_w_addr	Input	64	Write to or read from target 64-bits address
i_r_addr	Input	64	Read from target 64-bits address
i_MemRead	Input	1	One cycle signal and set current mode to reading
i_MemWrite	Input	1	One cycle signal and set current mode to writing
o_valid	Output	1	One cycle signal telling data is ready (used when `ld` happens)
o_data	Output	64	64-bits data from data memory (used when `ld` happens)

The test environment is as follows:



The naming of the wire
is in the perspective of cpu

Data memory

The data is stored in first cycle for write.
Read takes two cycles.

```
// cycle 1
always @(*) begin
    temp1_r_addr_w   = i_r_addr;
    temp1_MemRead_w  = i_MemRead;
end

// cycle 1
always @(*) begin
    for (i=0; i < 1024; i++) begin
        mem_w[i] = mem[i];
    end
    if (i_MemWrite) begin
        mem_w[i_w_addr+0] = i_data[ 7:0] ;
        mem_w[i_w_addr+1] = i_data[15:8] ;
        mem_w[i_w_addr+2] = i_data[23:16];
        mem_w[i_w_addr+3] = i_data[31:24];
        mem_w[i_w_addr+4] = i_data[39:32];
        mem_w[i_w_addr+5] = i_data[47:40];
        mem_w[i_w_addr+6] = i_data[55:48];
        mem_w[i_w_addr+7] = i_data[63:56];
    end else begin
        mem_w[i_w_addr+0] = mem[i_w_addr+0];
        mem_w[i_w_addr+1] = mem[i_w_addr+1];
        mem_w[i_w_addr+2] = mem[i_w_addr+2];
        mem_w[i_w_addr+3] = mem[i_w_addr+3];
        mem_w[i_w_addr+4] = mem[i_w_addr+4];
        mem_w[i_w_addr+5] = mem[i_w_addr+5];
        mem_w[i_w_addr+6] = mem[i_w_addr+6];
        mem_w[i_w_addr+7] = mem[i_w_addr+7];
    end
end

// cycle 2
always @(*) begin
    o_valid_w = (temp1_MemRead_r) ? 1 : 0;
    o_data_w  = (temp1_MemRead_r) ? {mem[temp1_r_addr_r+7], mem[temp1_r_addr_r+6],
                                     mem[temp1_r_addr_r+5], mem[temp1_r_addr_r+4],
                                     mem[temp1_r_addr_r+3], mem[temp1_r_addr_r+2],
                                     mem[temp1_r_addr_r+1], mem[temp1_r_addr_r+0]} : 0;
end
```

Instruction memory

3 cycles delay

```
// cycle 1
always @(*) begin
    temp1_valid_w = (i_valid) ? 1 : 0;
    temp1_w       = (i_valid) ? mem[i_addr/4] : 0;
end

// cycle 2
always @(*) begin
    temp2_valid_w = temp1_valid_r;
    temp2_w       = temp1_r;
end

// cycle 3
always @(*) begin
    o_valid_w     = temp2_valid_r;
    o_inst_w      = temp2_r;
end
```

Adder

- Demo yosys and adder

Workload

100 iterations

```
.global workload1
workload1:
    xor a5, a6, a6     # a5 = 0
    xor a6, a5, a5     # a6 = 0
    addi a4, a5, 1     # a4 = 1
    addi a1, a0, 4     # a1 = array address + 4
    addi a2, a5, 134
    addi a3, a5, 177
    addi t3, a5, 200   # t3 = 200
    add t0, a5, a6     # t0 = 0
L1:
    add t1, t0, a5     # t1 = 0
    addi a2, a2, 999
    add t2, t1, a5     # t2 = 0
    addi a3, a3, 888
    ld t4, 0(a0)
    ld t5, 16(a0)
    ld t6, 8(a0)
    addi t4, t4, 77
    addi t5, t4, 77
    addi t6, t4, 77
    add t4, t5, t6
    or t4, t4, t6
    and t5, a2, t5
    xori t5, t6, 1
    andi t6, t4, 111
    slli t6, t5, 1
    slli t4, t6, 4
    slli t5, t3, 5
    or t4, t5, t3
    xor t4, a3, t5
    add t5, t6, t4
    add t6, t2, a2
    xor t6, t4, t5
    xor t5, t5, a3
    and t4, a2, t4
    addi a6, a0, 5
    addi a0, a6, 2     # a0 += 7
    addi a6, t2, 1
    addi t2, a6, 1     # t2 += 2
    addi a1, a0, 1
    addi a0, a1, 1     # a0 += 2
    addi a6, t1, 1
    addi t1, a6, 1     # t1 += 2
    addi a6, t0, 1
    addi t0, a6, 1     # t0 += 2
    sd t4, 0(a0)
    sd t5, 16(a0)
    sd t6, 8(a0)
    bne t1, t3, L1     # t0 vs t3
```

```
.global workload2
workload2:
    xor a2, a1, a1     # a2 = 0
    addi a3, a2, 147   # a3 = 147
    slli a4, a3, 1     # a4 = 147 << 1
    or a2, a3, a4      # a2 = (147 << 1 + 147)
    addi a1, a0, 1000  # a1 = addr + 1000
    xor a3, a2, a2     # a3 = 0
J1:
    ld t0, 0(a0)
    addi t1, t0, 7
    bne a0, a1, J3
    beq a0, a0, J5
J2:
    addi t1, a0, 1
    addi a0, t1, 1
    beq a0, a0, J4
J3:
    add t0, a2, t1
    sd t0, 0(a0)
    beq a0, a0, J2
J4:
    addi a3, a2, 123
    addi a2, a3, 456
    beq a0, a0, J1
J5:
    ret              # eof
```

500 iterations

```
.global workload3
workload3:
    xor  a1, a0, a0
    xor  t0, a1, a1
    addi t1, a1, 1
    addi t4, a0, 1016
R1:
    add t2, t1, t0
    sd t2, 0(a0)
    xor a2, a0, a0
    addi a1, a0, 1
    add a0, a1, a2
    xor a1, a0, a0
    add t0, t1, a1
    add t1, t2, a1
    bne t4, a0, R1
    ret                # eof
```

1000 iterations

Some information

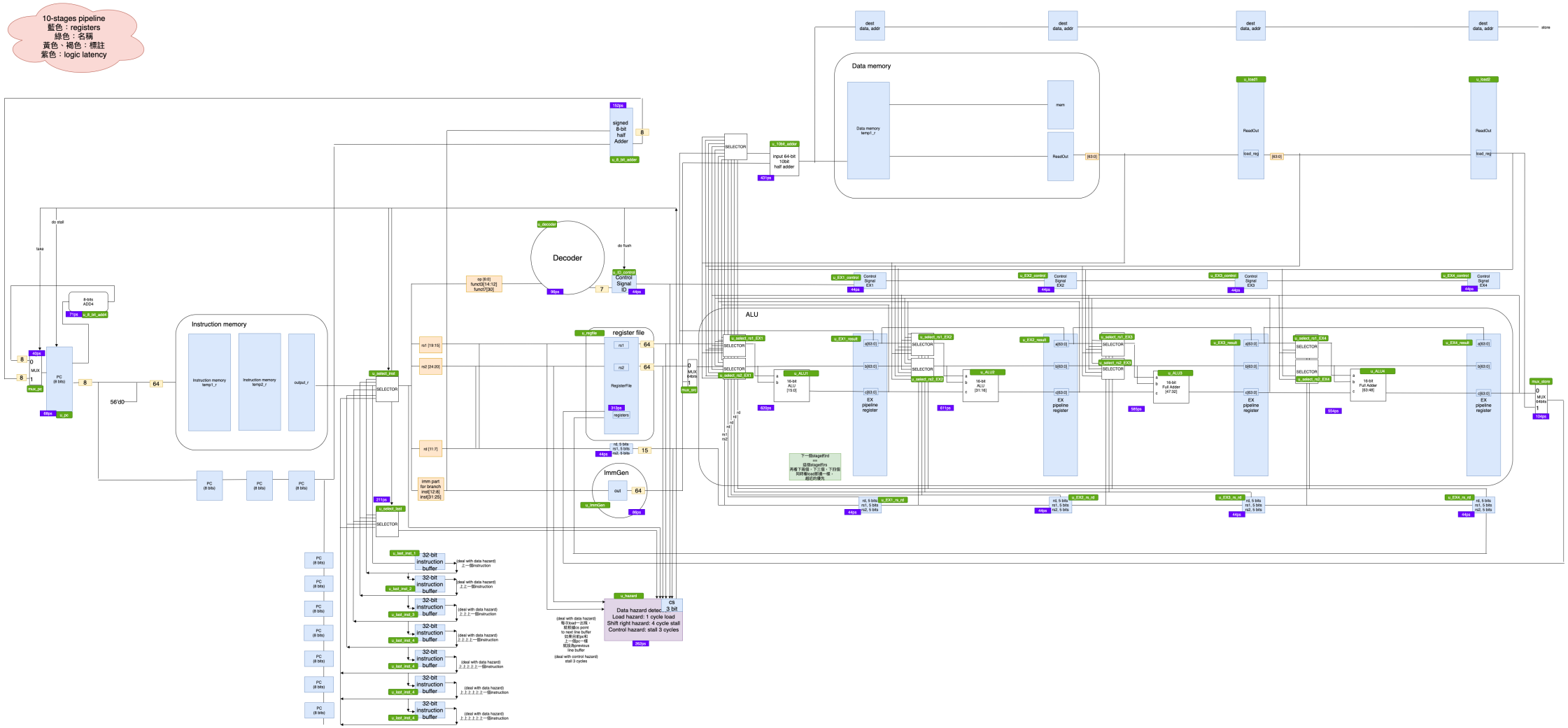
	測試時間i5	Cycle counts (pipeline)	Cycle counts (HW3)	Spike simulator rdcycle
T8	3分鐘內	4518	58630	108623
T9	3分鐘內	18540	90160	157891
T10	3分鐘內	15254	137230	178064
T8 small	10s內	468	5980	10550
T9 small	10s內	780	3760	6331
T10 small	10s內	254	2230	2814

	Area (μm^2)	Latency (ps)
HW3	12274.84	2259.8
HW4	17519.56	1091.13

Some notice

- You can optimize your design for these three workloads
  - 8-bit instruction range, 10 bit data range, patterns
- Implement forwarding to make your implementation faster
- Implement hazard detection to make your pipeline work correctly
- Branch predictor may further boost your cpu performance

Example design



Reference

- Yosys: http://www.clifford.at/yosys/
- ABC: https://people.eecs.berkeley.edu/~alanmi/abc/
- FreePDK: https://github.com/cornell-brg/freepdk-45nm