Synthesizing your Processor

Pooria M. Yaghini

Performance Metrics

- Execution time
- MIPS (Million Instruction per Second)
- Operational Clock Frequency

Performance Metrics

- **Execution time**
- MIPS (Million Instruction per Second)
- Operational Clock Frequency

Performance Metrics

Execution time = #Inst. * CPI * (1/Freq.)

Notes:

- **In a Single-cycle processor, CPI equals to 1.**
  - **So frequency is the only factor to make it faster**

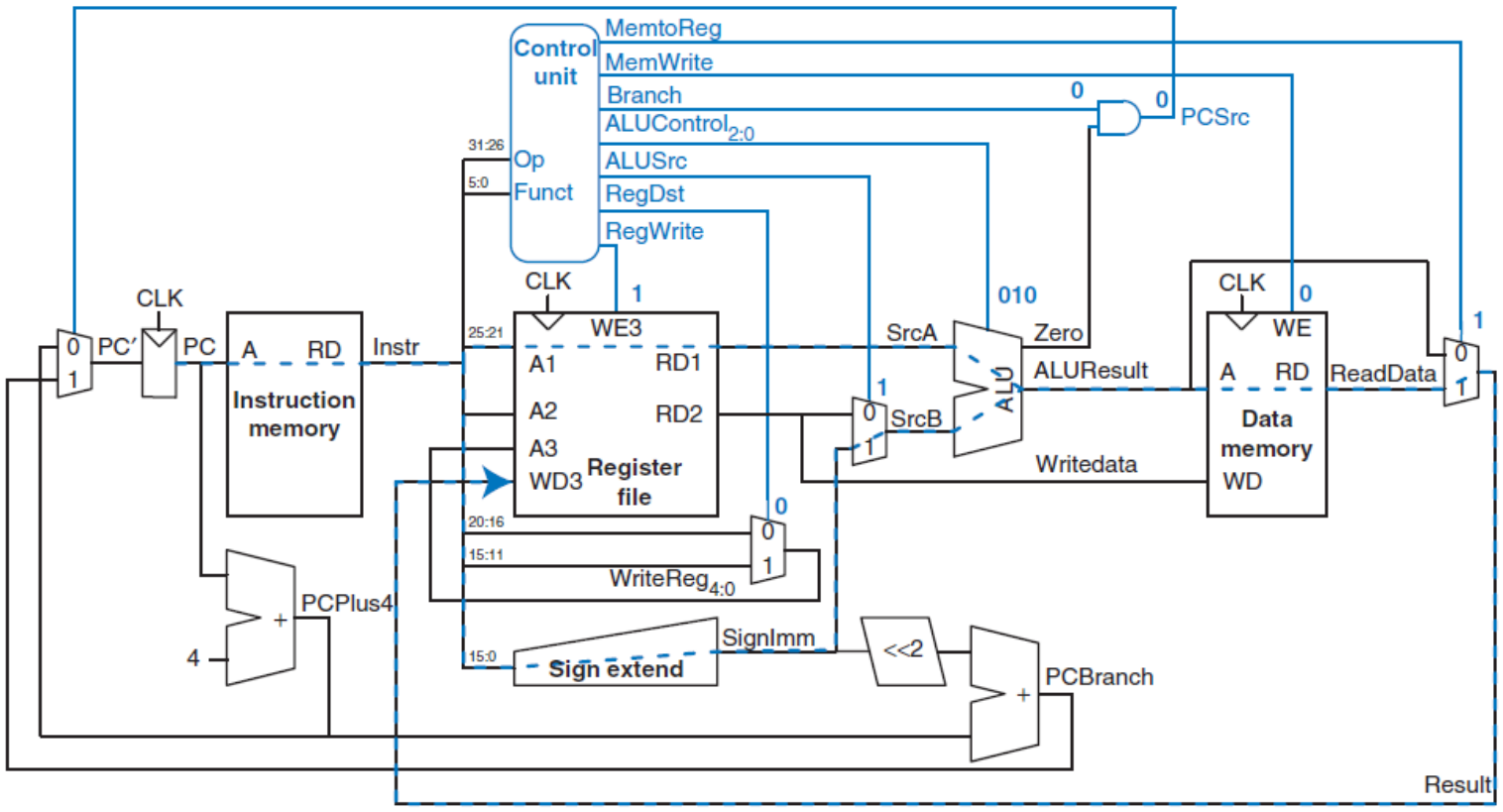
Performance Metrics

Execution time = #Inst. * CPI * (1/Freq.)

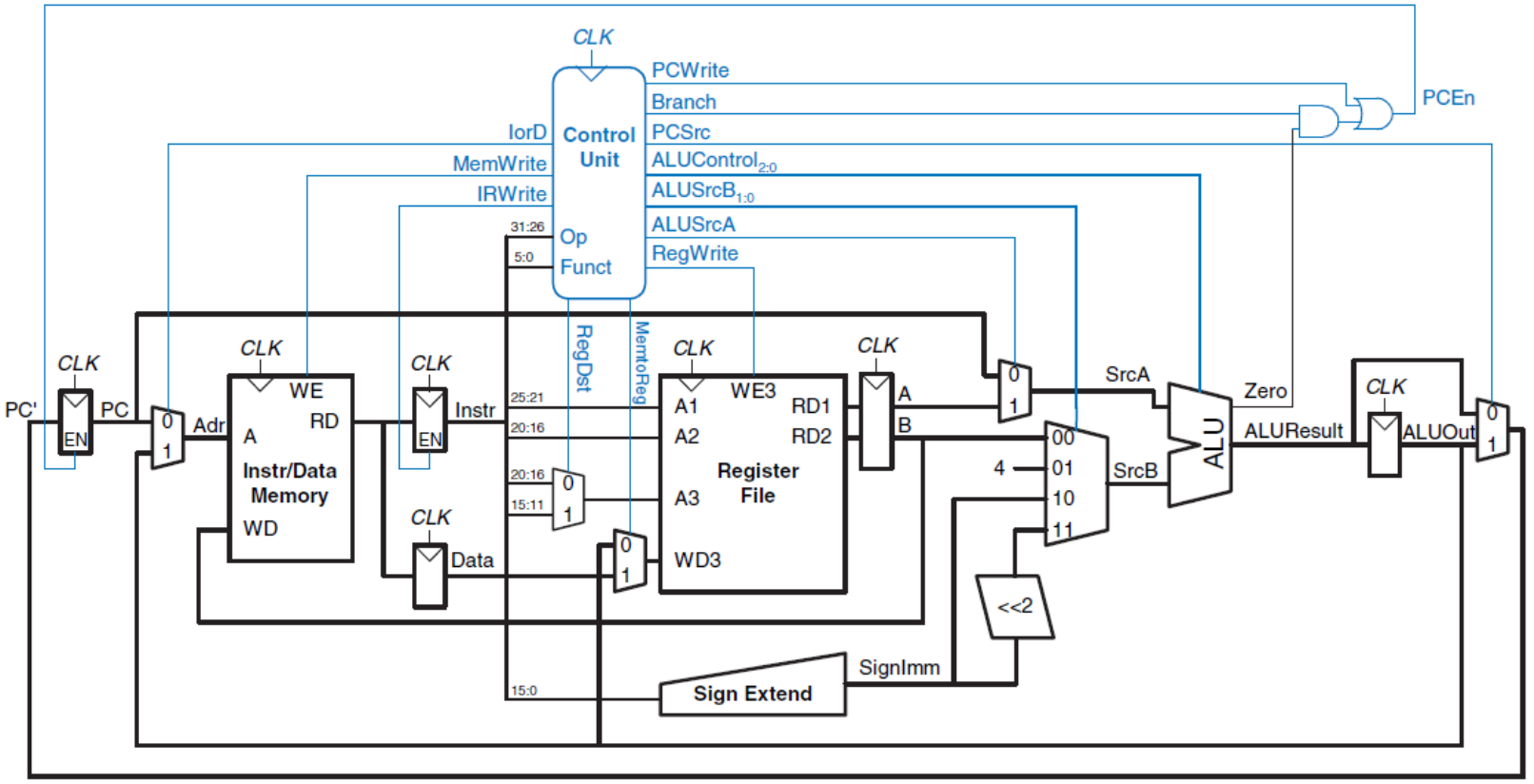
Notes:

- In a Single-cycle processor, CPI equals to 1.
  - So frequency is the only factor to make it faster
- **What is your suggestion to improve the frequency?**

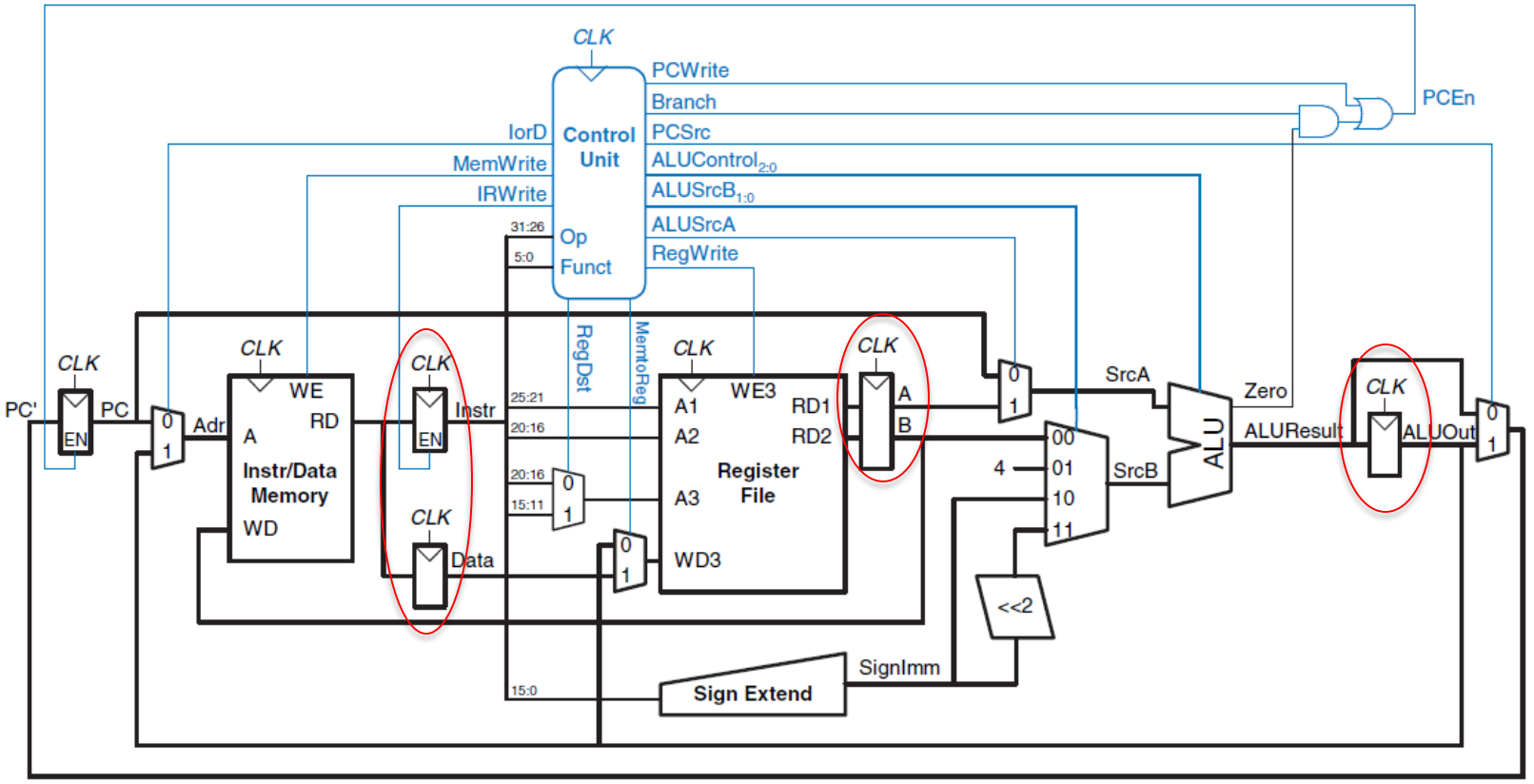
Single-cycle Architecture



Multicycle Architecture



Multicycle Architecture



Performance Evaluation

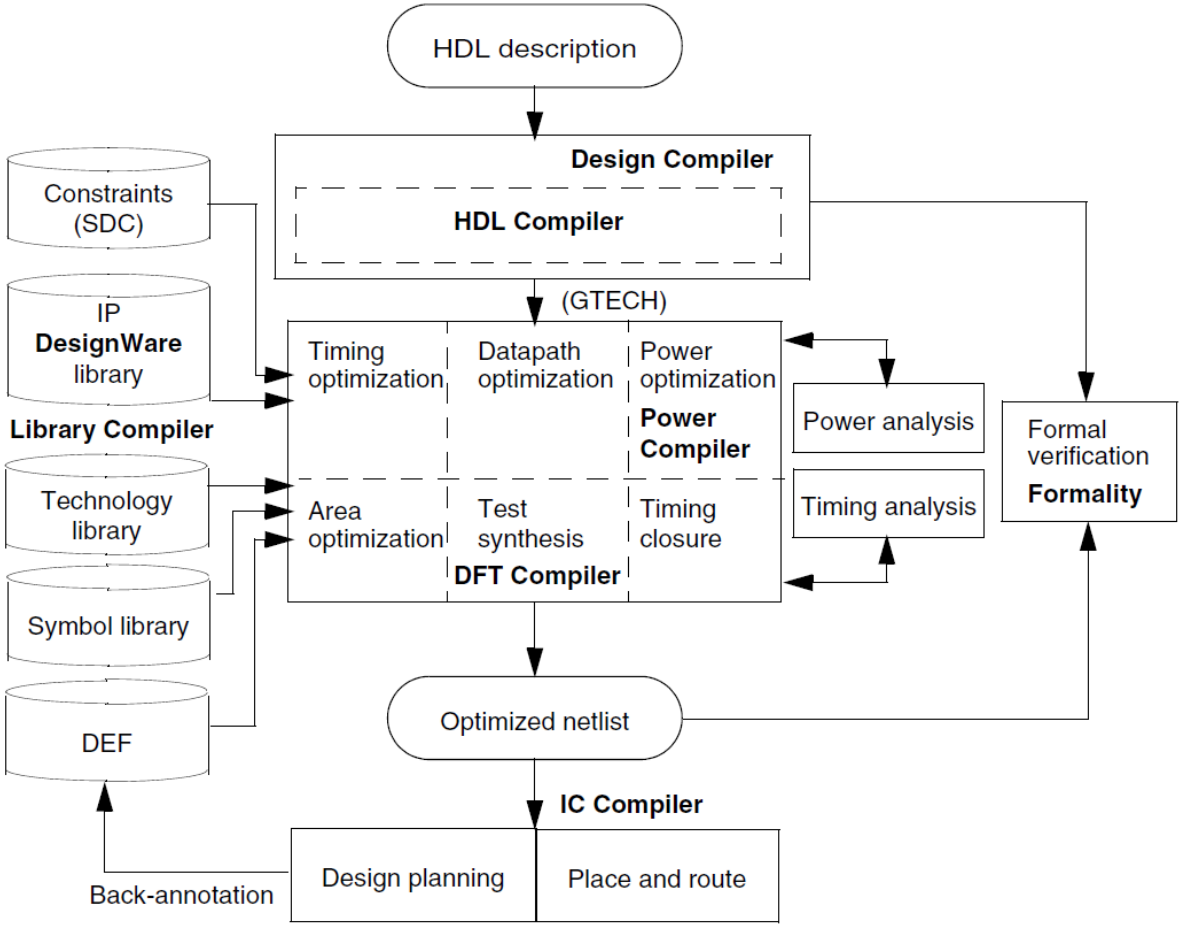
The SPECINT2000 benchmark consists of approximately 25% loads, 10% stores, 11% branches, 2% jumps, and 52% Data Processing instructions. Determine the average CPI for this benchmark.

Performance Evaluation

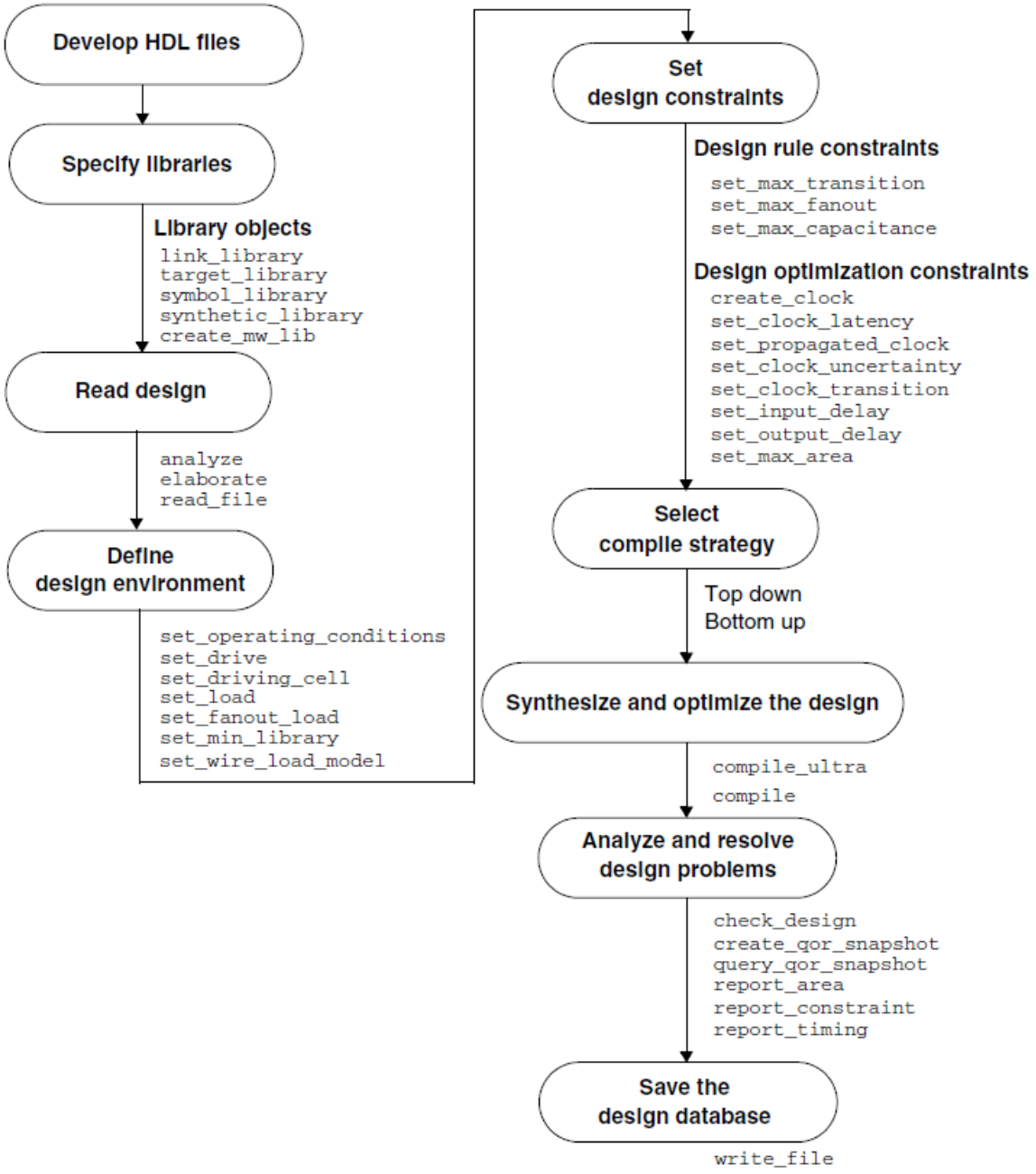
Now you have achieved building a single-cycle ARM processor in a 28nm CMOS manufacturing process.

- First, determine the delay for each major module of your design. Then compute the execution time for a program with 100 billion instructions.
- Second, do the same for the multicycle processor and compare.
  - If you have not implemented the multicycle processor, consider the following equation to determine the clock period:
  - $Tc = t_{pcq} + t_{mux} + \text{maximum}(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$

Synthesis Flow



Reference: **Design Compiler® User Guide**



Reference: **Design Compiler® User Guide**

Synthesis Commands

- analyze
- elaborate
- compile[_ultra]
- report_
  - area [-hierarchy]
  - power
  - timing

Synthesis Commands

- **set_max_area**
  - Sets the max_area attribute to a specified value on the current design. This command is only supported in DC Expert with the **compile** command. If you specify a max_area value and run the **compile_ultra** command, the max_area value is ignored and the default (zero) is used.

```
dc_shell> set_max_area 0.0
```

Synthesis Commands

- **set_max_delay**
  - Specifies a maximum delay target for paths in the current design.

The following example shows how to optimize the design so that any delay path to a port named Y is less than 10 units.

```
dc_shell> set_max_delay 10.0 -to {Y}
```

This example shows how to specify that all paths from cell ff1a or ff1b that pass through cell u1 and end at cell ff2e must be less than 15.0 units.

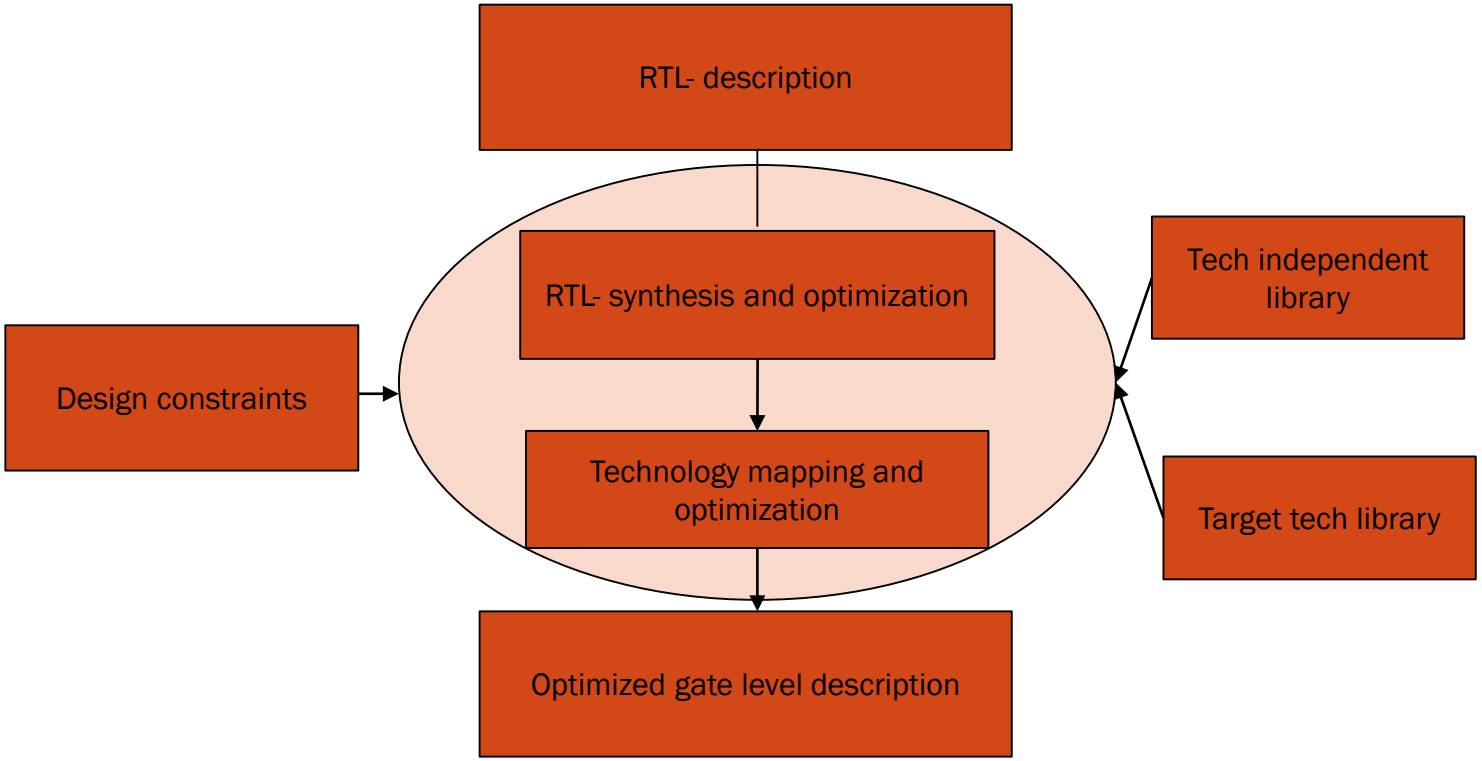
```
dc_shell> set_max_delay 15.0 -from {ff1a ff1b} -through {u1} -to {ff2e}
```

Synthesis Commands

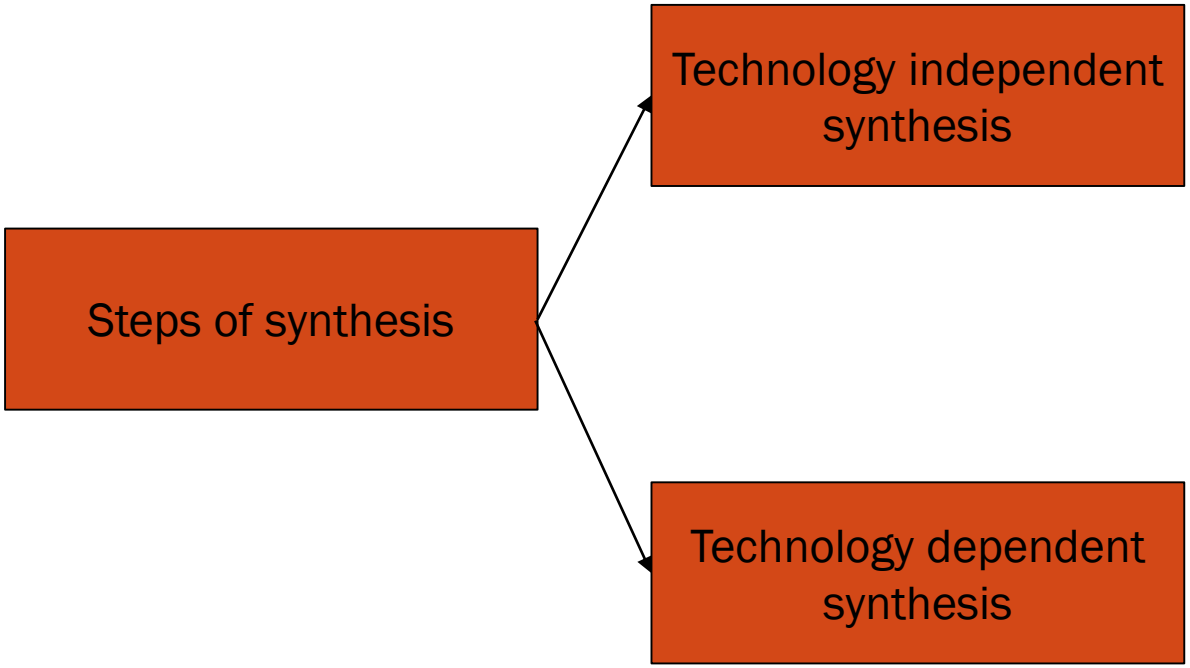
- **More commands to help optimization**

```
dc_shell> set_max_leakage_power 0
```

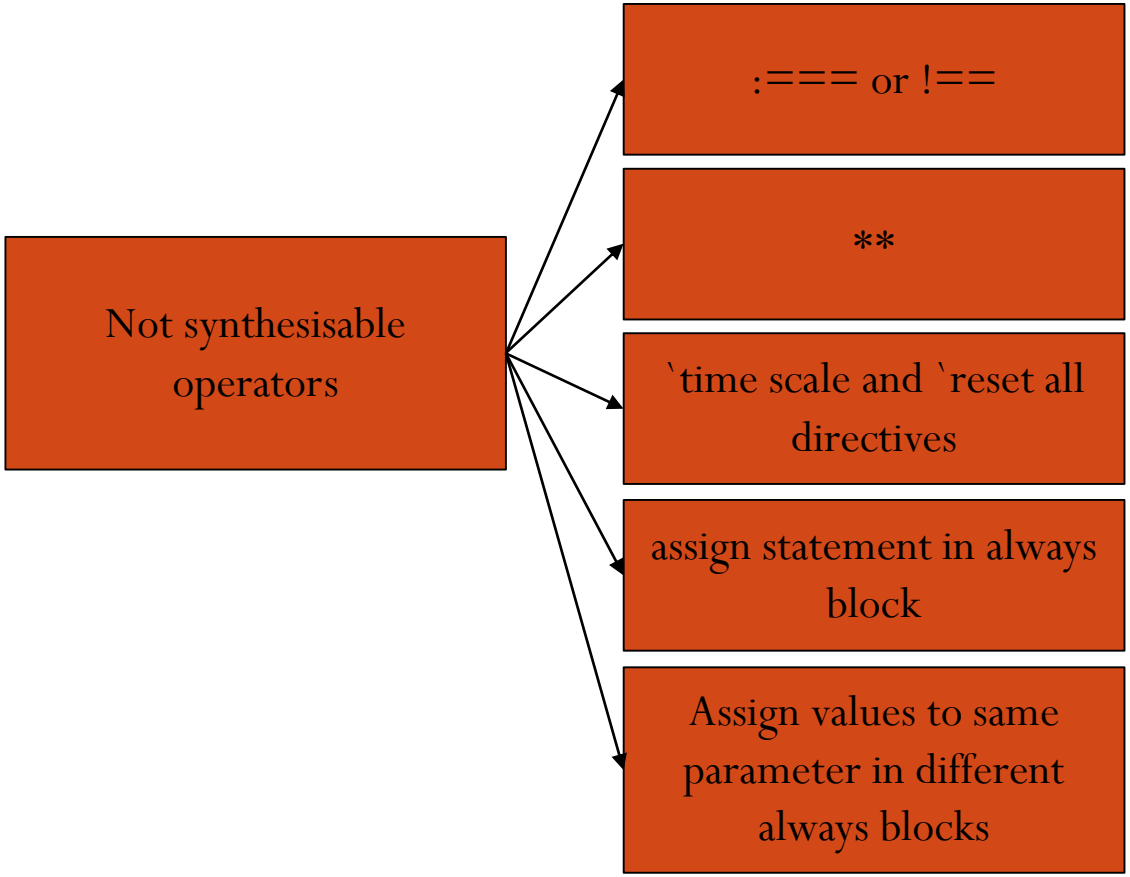
Process of Synthesis



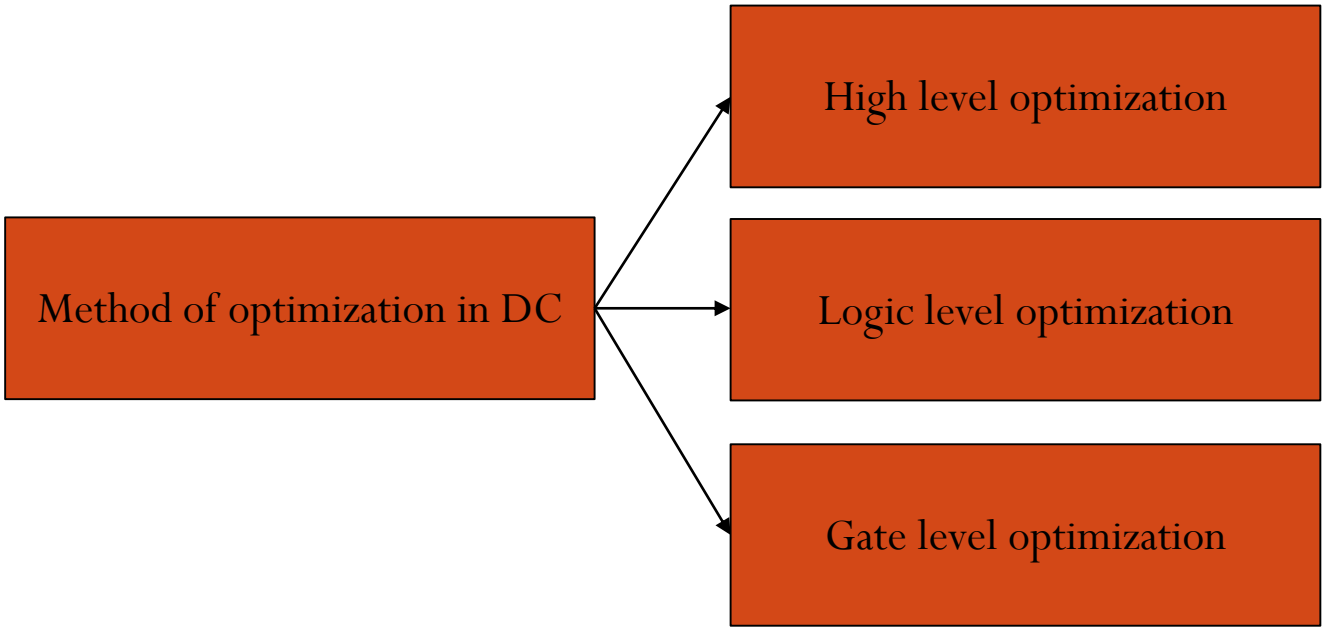
Steps of Synthesis



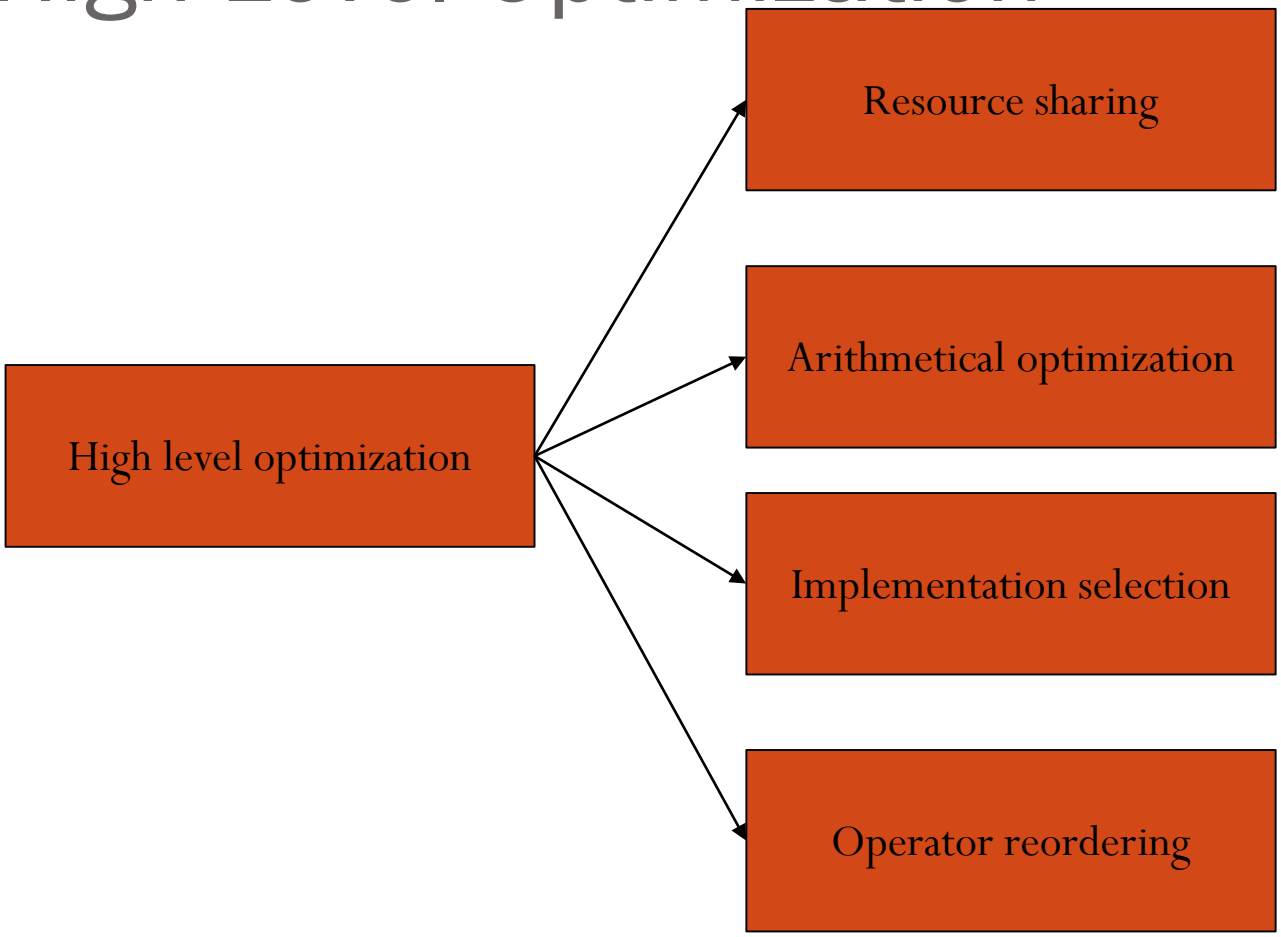
Not Synthesisable Operators



Optimization Methods in Design Compiler

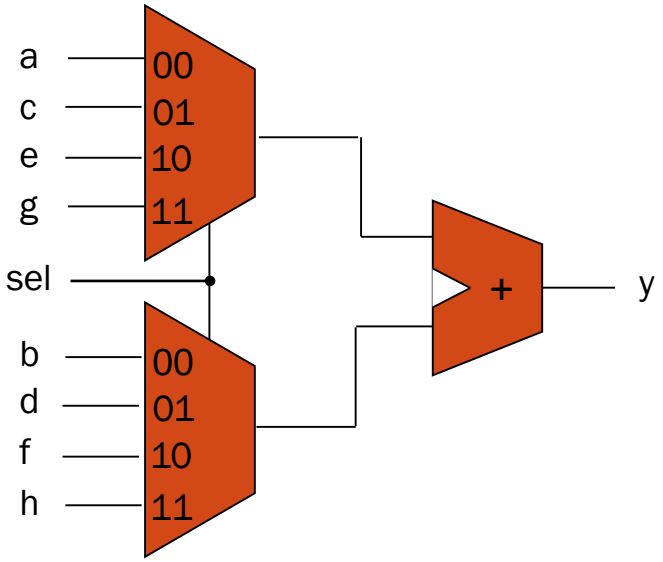


High Level Optimization



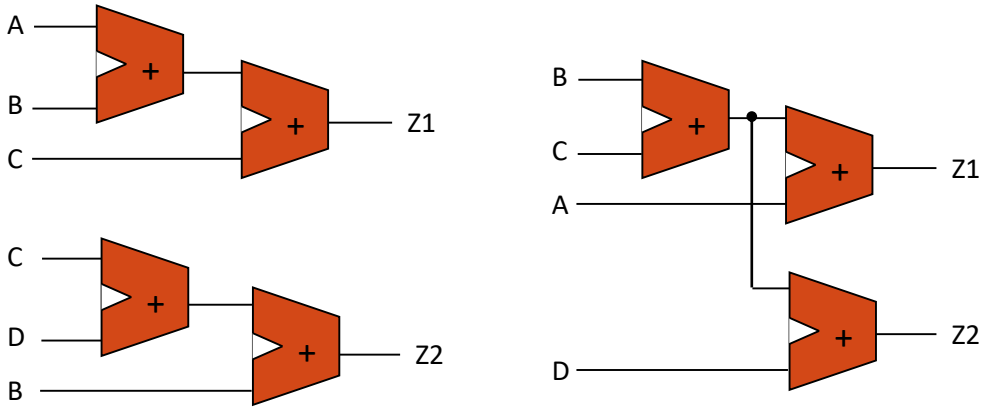
Resource Sharing

```
if(sel==2'b00)
    y=a+b;
else if(sel==2'b01)
  y=c+d;
else if(sel==2'b10)
  y=e+f;
else y=g+h;
....
```

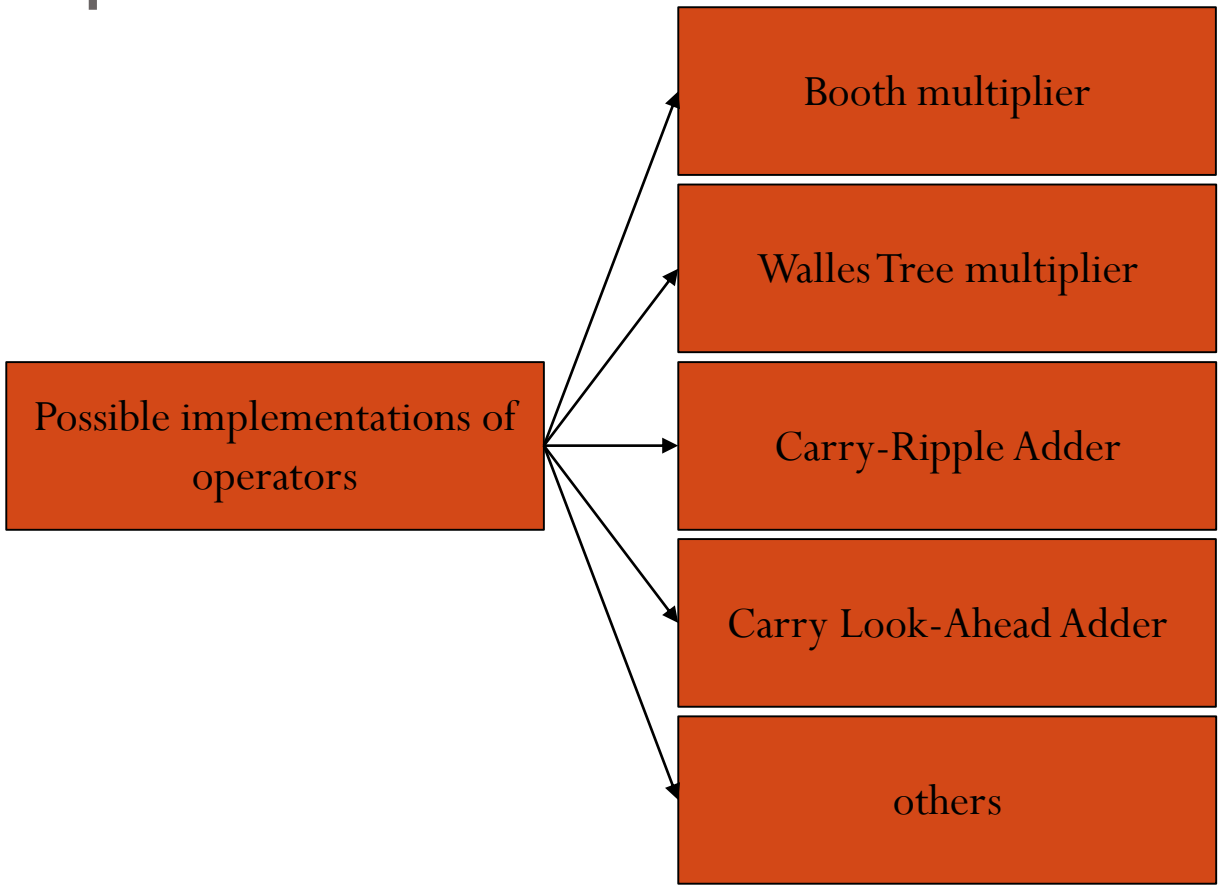


Arithmetical Optimization

- Z1= A+B+C
- Z2= C+D+B

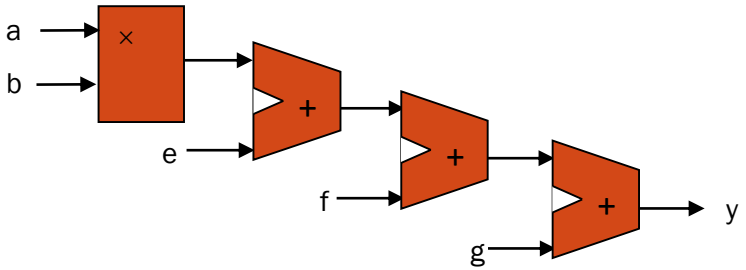


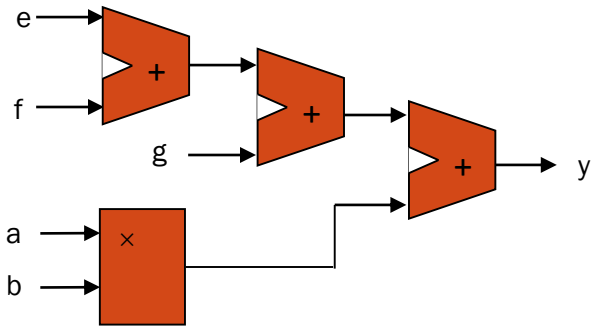
Implementation Selection



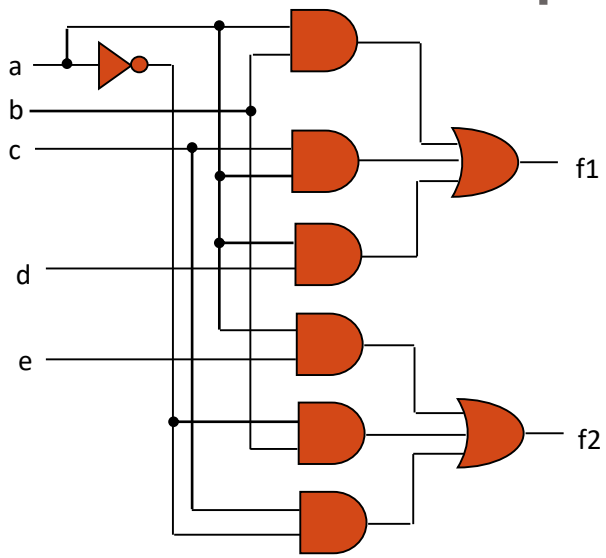
Operator Reordering

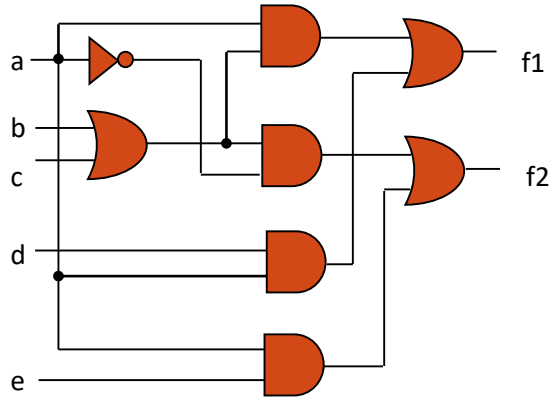
Y=a*b+e+f+g

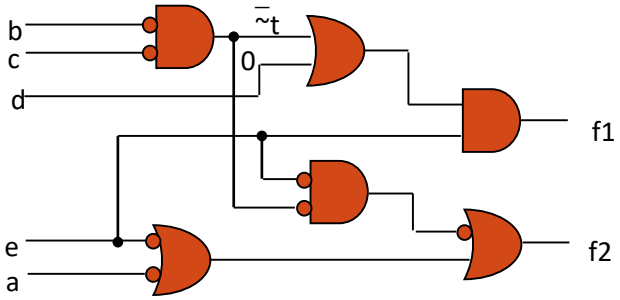




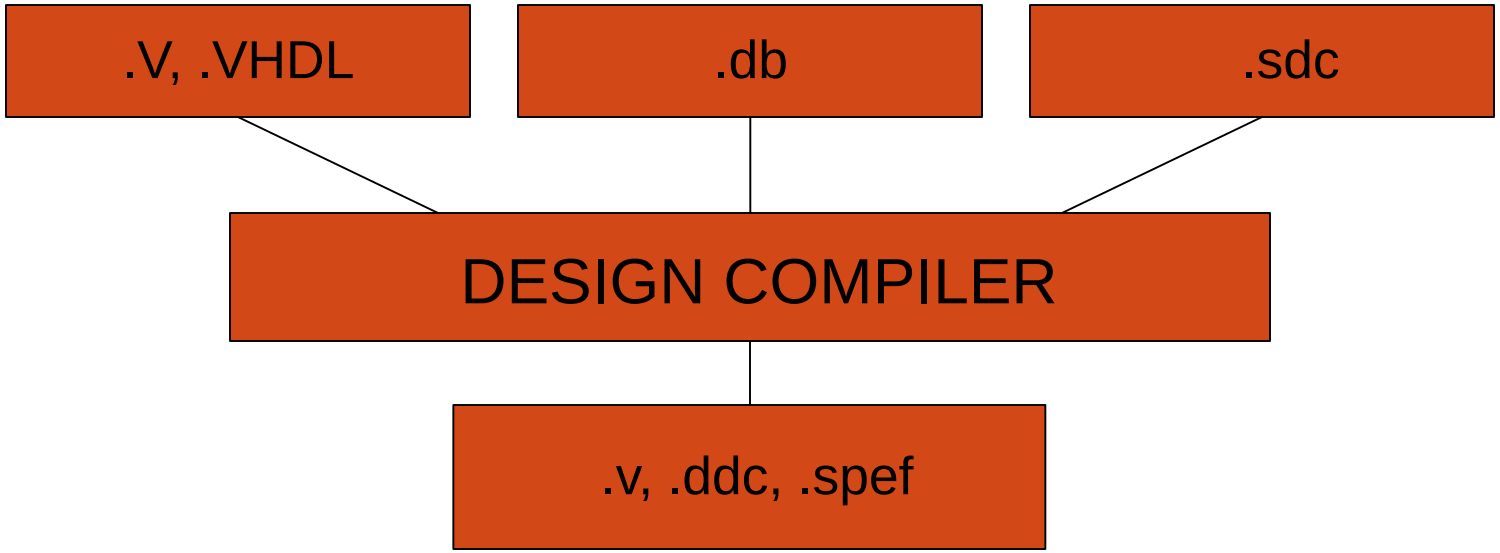
Gate Level Optimization







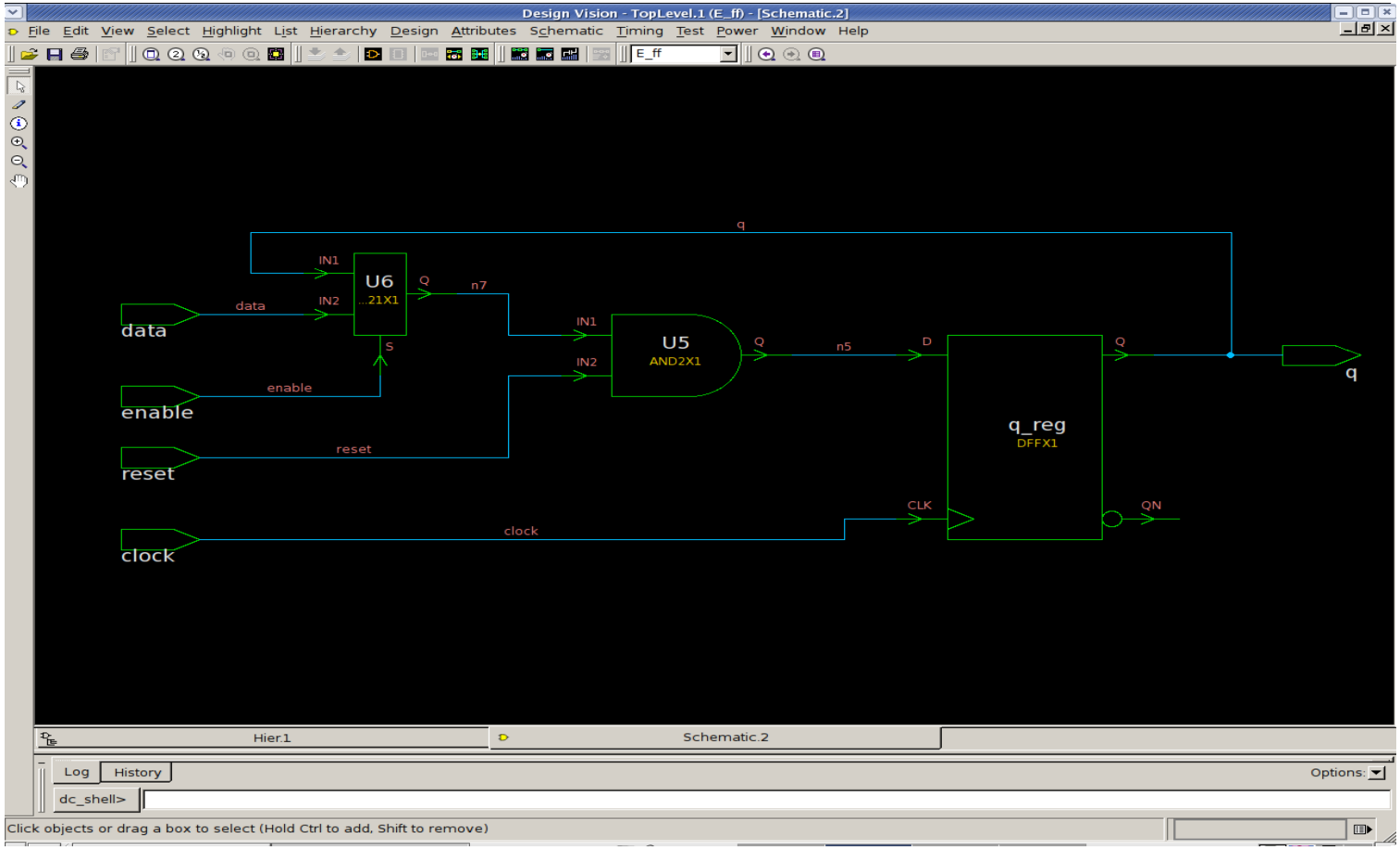
Inputs and Output of Logic Synthesis Tool (DC)



Behavioral Verilog Code of EFF

```
module E_ff(q, data, enable,
reset, clock);
output q;
input data, enable, reset,
clock;
reg q;
always @(posedge clock) //
whenever the clock makes a
transition to 1
if (reset == 0)
q = 1'b0;
else if (enable==1)
q = data;
else q = q
endmodule
```

Synthesized Design View of EFF from DC



Gate Level Verilog from DC

```
module E_ff ( q, data, enable, reset, clock );
  input data, enable, reset, clock;
  output q;
  wire   n5, n6;

  DFFX1 q_reg ( .D(n5), .CLK(clock), .Q(q) );
  AND2X1 U5 ( .IN1(n6), .IN2(reset), .Q(n5) );
  MUX21X1 U6 ( .IN1(q), .IN2(data),
.S(enable), .Q(n6) );
endmodule
```

SRAM

```
module dmem( input  logic        clk, we,
             input  logic [31:0] a,   wd,
             output logic [31:0] rd);
`ifdef sim
  logic [31:0] RAM[63:0];

  assign rd = RAM[a[31:2]]; // word aligned

  always @( posedge clk )
    if (we)
      RAM[ a[31:2] ] <= wd;
`else

   SRAM1RW512x32 RAM (
                              .A   ( a[8:0] ),
                              .CE  ( 1'b1   ),
                              .WEB ( ~we    ),
                              .OEB ( we     ),
                              .CSB ( 1'b0   ),
                              .I   ( wd     ),
                              .O   ( rd     )
                      );

`endif
endmodule
```