IC Design
HW4 Tutorial

Yu-Tung Liu
Advisor: Tzi-Dar Chiueh
2020/12/18

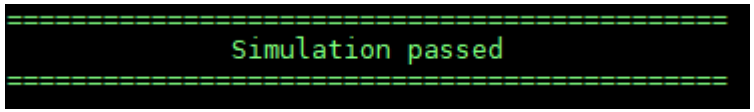
Outline

- Workflow & Notification of HW4
- Algorithm
- Standard Cell Library
- Tips
- Verification
- Reminder

Workflow (1/2)

- Grading
  - Correctness (40%)
  - Ranking (30%)
  - Report (30%)

- First, design a circuit that can pass the simulation.



- Second, decrease the CYCLE in testbench.v until it failed (CYCLE < critical path).

Workflow (2/2)

- Third, find the number of transistors in your design.

```
module AN3(Z,A,B,C,number);
    output Z;
    input A,B,C;
    parameter size = 10'd50;
    output [size:0] number;
    wire  [size:0] number;
    assign number=11'd8;
```

```
==========================================
                Summary
==========================================
        Clock cycle:           5.6 ns
        Number of transistors: 2358
        Total excution cycle:  1004
        Correctness Score:     40.0
        Performance Score:     13257619.2
==========================================
```

- Finally, modify you design to get better performance.
  - Trade-off between area & speed. Ex. Try different adders, carry-skip, carry-lookahead, etc.
  - Try different algorithm.

Notification (1/2)

- In this HW, all the logic operation MUST consist of standard cell (defined in lib.v). You can NOT use logic operators.

```
wire a, b, c;
assign a = b & c;
```

→ Behavioral Modeling

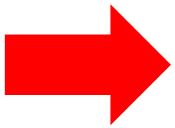
```
wire a, b, c;
AN2 an(a, b, c);
```

→ Structural Modeling

Notification (2/2)

- Do NOT change any module name and port name in sqrt.v, just modifiy the module description, otherwise you can't pass the simulation.

```
module sqrt (
    input clk,
    input rst_n,
    input [9:0] i_radicand,
    output o_finish,
    output [4:0] o_root,
    output [50:0] number
);
```



Don't Change

- Use FD2 (positive edge) module for flip flop.

臺灣大學

Algorithm

Possible algorithm

- Trial & error
  - Long latency / easy to implement
- Binary search
- Newton's method
  - $root_{n+1} = \frac{1}{2}(root_n + \frac{Radicand}{root_n})$
- 長除法
- And more…
- ***No look-up-table***

臺灣大學

Standard Cell Library

Standard Cell Library (lib.v)

- Choose what you need
- Compose your circuit according to I/O connections

- IV // not
- AN3
- AN4
- AN2
- EN // xnor
- EN3
- EO // xor
- EO3
- FA1 // full adder
- FD1 // negative edge DFF
- FD2 // positive edge DFF
- ND2 // nand
- ND3
- ND4
- NR2 // nor
- NR3
- OR2 // or
- OR3
- OR4
- HA1 // half adder
- MUX21H // 2-to-1 MUX

Number of Transistors

```
module Reg3(Q, DD, CLK, RESET, Reg3_num);

output [2:0] Q;
input [2:0] DD;
input CLK,RESET;
output [50:0] Reg3_num;


wire [50:0] FD_num0, FD_num1, FD_num2;
assign Reg3_num = FD_num0 + FD_num1 + FD_num2;

FD2 fd0(Q[0], DD[0], CLK,RESET, FD_num0);
FD2 fd1(Q[1], DD[1], CLK,RESET, FD_num1);
FD2 fd2(Q[2], DD[2], CLK,RESET, FD_num2);

endmodule
```

```
module DUT(clk, rst, A, B, C, D);

input clk;
input rst;
input [3:0] A;
input [6:0] B;
input [5:0] C;
output [14:0] D;
output [50:0] DUT_num;

wire [50:0] num0, num1;
assign DUT_num = num0 + num1;

wire [2:0] A_reg, B_reg;
Reg3 rr0(A_reg, A[2:0], clk, rst, num0);
Reg3 rr1(B_reg, B[2:0], clk, rst, num1);

endmodule
```

臺灣大學

Tips

Parameterized module

- Verilog “generate” statement
  - Copy the codes.
  - Instantiate multi modules easily.

- Use generate and parameter to define parameterized module.
  - High flexibility.
  - High readability.
  - Shorten your codes.
  - Prevent wire misconnection.

Example – n-bit DFF

```
//BW-bit FD2
module REGP #(
    parameter BW = 2
)(
    input clk,
    input rst_n,
    output [BW-1:0] Q,
    input [BW-1:0] D,
    output [50:0] number
);

wire [50:0] numbers [0:BW-1];

genvar i;
generate
    for (i=0; i<BW; i=i+1) begin
        FD2 f0(Q[i], D[i], clk, rst_n, numbers[i]);
    end
endgenerate

//sum number of transistors
reg [50:0] sum;
integer j;
always @(*) begin
    sum = 0;
    for (j=0; j<BW; j=j+1) begin
        sum = sum + numbers[j];
    end
end

assign number = sum;

endmodule
```

→ **Generate FD2 according to BW.**

→ **Use for loop to sum transistor counts.**

```
REGP#(1) r0_0(clk, rst_n, root_s1_r, root_s1_w, numbers[3]);
```

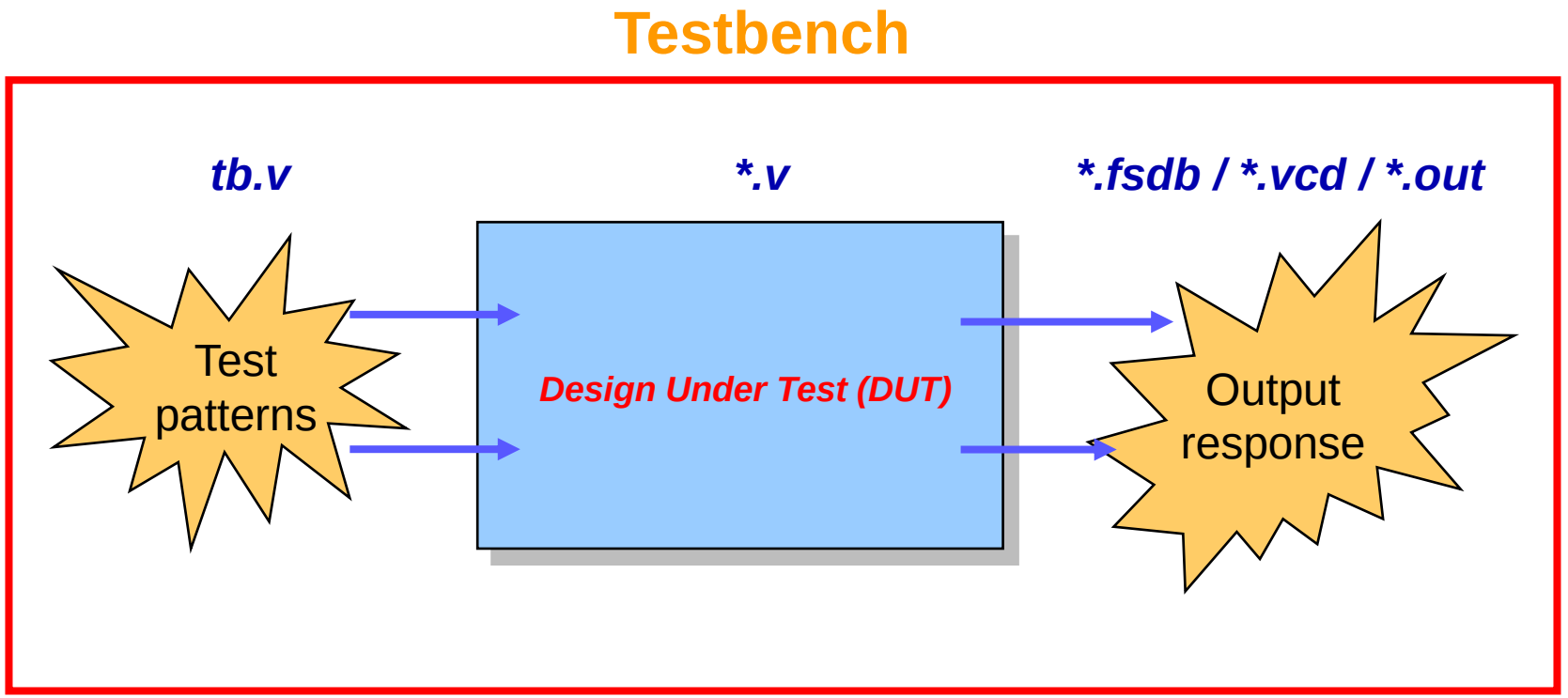
→ **Instantiate in top module.**

臺灣大學

Verification

Test and Verify Your Circuit

- By applying input patterns and observing output responses



Compile and debug

- Source
  - source /usr/cad/cadence/cshrc
  - source /usr/spring_soft/CIC/verdi.cshrc

- Include the tb.v & lib.v files to run simulation
  - ncverilog tb.v sqrt.v lib.v +access+r
  - ncverilog tb.v sqrt.v lib.v +access+r +define+DEBUG
  - ncverilog tb.v sqrt.v lib.v +access+r +define+PIPELINE
  - ncverilog tb.v sqrt.v lib.v +access+r +define+DEBUG+PIPELINE

Reminder (1/2)

- Loosen the clock cycle when you're checking your circuit logic.
- Once the logic is correct, start to shorten the clock period to find the critical path.
- Use basic gates provided in lib.v to design your circuit. No behavior level code will be accepted.

Reminder (2/2)

- Due on 2021/01/08 13:20
- For any further questions , contact TAs !
  - 劉雨東 r08943018@ntu.edu.tw
  - 李嘉恆 r08943006@ntu.edu.tw
- To know more about Verilog, refer to
  - http://www.ece.umd.edu/courses/enee359a.S2008/verilog_tutorial.pdf