Integrated Circuit Design
Verilog Tutorial – part 2

Speaker : Yu-Cheng Li (d01943008@ntu.edu.tw)

Advisor : Yi-Chang Lu

Date : 2019.03.27

Graduate Institute of Electronics Engineering
National Taiwan University

Outline

Part 1

- Introduction
- Verilog for RTL
  - Module Description and Declaration
  - Data Type and Operators
  - Combinational Behavior
  - Sequential Behavior

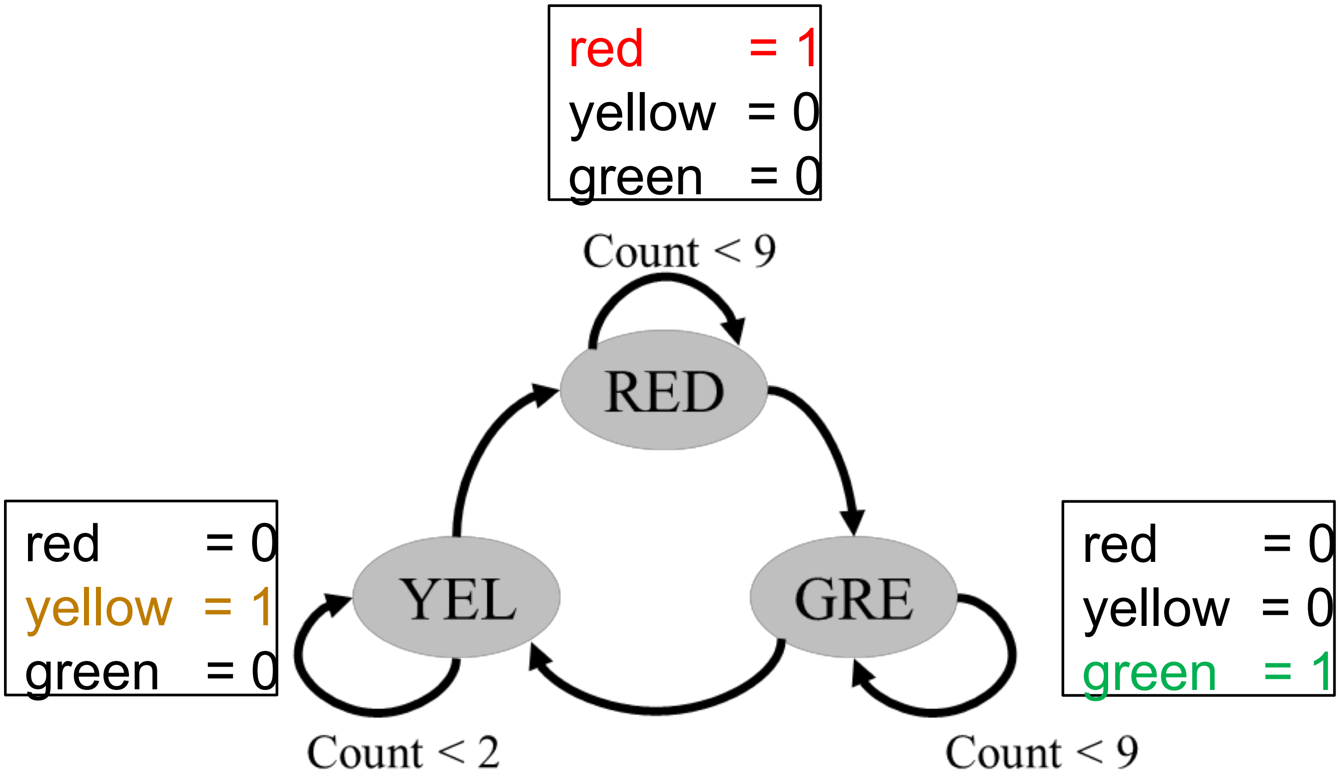
Part 2

  - Finite State Machine
  - Advanced Topics
- Design Example
- RTL Simulation Tool

Part 3

- Synthesis Tool

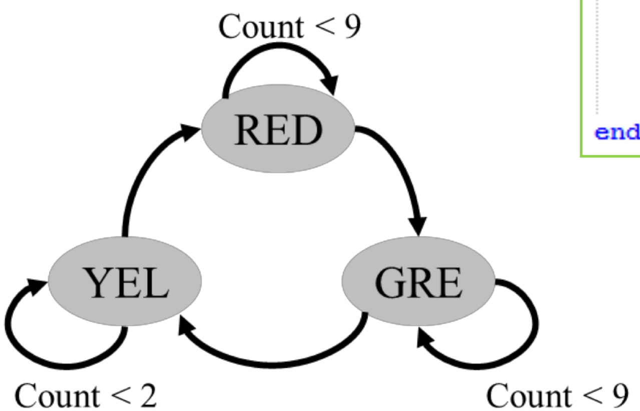
Finite State Machine (FSM) (1/3)



Finite State Machine (FSM) (2/3)

State Transition

```
always @ (posedge clock) begin
    if ( reset ) begin
        state <= RED;
        count <= 0;
    end
    else begin
        state <= next_state;
        count <= next_count;
    end
end
```



Next State Logic

```
always @ (*) begin
    case (state)
        RED:if ( count == 4'd9 )
                next_state = GRE;
            else
                next_state = RED;
        GRE:if ( count == 4'd9 )
                next_state = YEL;
            else
                next_state = GRE;
        YEL:if ( count == 4'd2 )
                next_state = RED;
            else
                next_state = YEL;
    default:next_state = RED;
    endcase
end
```

Output Logic

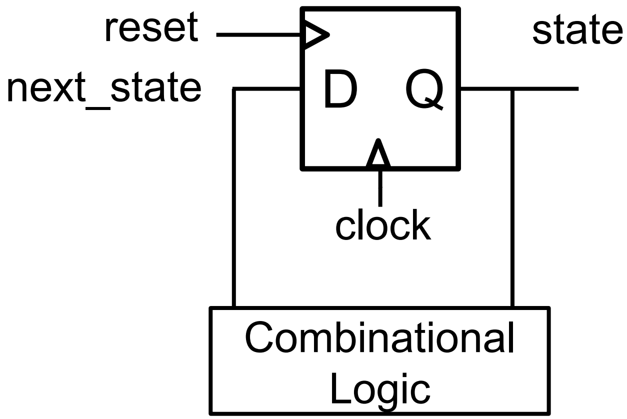
```
always @ (*) begin
    case (state)
        RED:begin
            red    = 1;
            green  = 0;
            yellow = 0;
            end
        GRE:begin
            red    = 0;
            green  = 1;
            yellow = 0;
            end
        YEL:begin
            red    = 0;
            green  = 0;
            yellow = 1;
            end
    default:begin
            red    = 1;
            green  = 0;
            yellow = 0;
            end
    endcase
```

Finite State Machine (FSM) (3/3)

- **Why we have x and next_x ?**

```
always @ (posedge clock) begin
    if ( reset ) begin
        state <= RED;
        count <= 0;
    end
    else begin
        state <= next_state;
        count <= next_count;
    end
end
```

```
always @ (*) begin
    case (state)
        RED:if ( count == 4'd9 )
                next_state = GRE;
            else
                next_state = RED;
        GRE:if ( count == 4'd9 )
                next_state = YEL;
            else
                next_state = GRE;
        YEL:if ( count == 4'd2 )
                next_state = RED;
            else
                next_state = YEL;
    default:next_state = RED;
    endcase
end
```



Advanced - For loop

- **Different with C/C++, just provide a convenient way of writing a series of statement**
- **Loop index variables must be integer**

```
integer k;
always@(posedge clk)
begin
   for (k=0;k<=2;k=k+1)
   begin
      out[k] <= n_out[k];
   end
end
always@(*) begin
   for (k=0;k<=2;k=k+1)
   begin
      n_out[k] = a[k]&b[k];
   end
end
```

→

```
integer k;
always@(posedge clk)
begin
   out[0] <= n_out[0];
   out[1] <= n_out[1];
   out[2] <= n_out[2];
end
always@(*) begin
   out[0] = a[0]&b[0];
   out[1] = a[1]&b[1];
   out[2] = a[2]&b[2];
end
```

Advanced - Completeness of condition statement

- **You should assign all conditions clearly.**

Wrong!

```
reg [1:0]n_c,c;
always@(posedge clk) begin
  c <= n_c;
end
always@(*) begin
  if (XXX) begin
    if(XXX)
      n_c = 2'b00;
    else
      n_c = 2'b11;
  end
end
```

Correct!

```
reg [1:0]n_c,c;
always@(posedge clk) begin
  c <= n_c;
end
always@(*) begin
  if (XXX) begin
    if(xxx)
      n_c = 2'b00;
    else
      n_c = 2'b11;
  end
  else
    n_c = c;
end
```

Advanced - Completeness of condition statement

- **Even for array!**

Wrong!	Correct!

Wrong!

```
reg [1:0] c[0:15];
reg [1:0] n_c[0:15];
always@(posedge clk) begin
  for (k=0;k<=15;k=k+1)
    c[k] <= n_c[k];
end
always@(*) begin
  if(XXX)
    n_c[0] = 2'b11;
  else
    n_c[0] = 2'b10;
end
```

Correct!

```
reg [1:0] c[0:15];
reg [1:0] n_c[0:15];
always@(posedge clk) begin
  for (k=0;k<=15;k=k+1)
    c[k] <= n_c[k];
end
always@(*) begin
  if(XXX)
    n_c[0] = 2'b11;
    for (k=1;k<=15;k=k+1)
      n_c[k] = c[k];
  else
    n_c[0] = 2'b10;
    ......
end
```

Advanced - Completeness of condition statement

- **Convenient way to save coding times.**
- **To assign value for register in the beginning.**

```
reg [1:0]n_c,c;
always@(posedge clk) begin
  c <= n_c;
end
always@(*) begin
  n_c = c;
  if (XXX)
  begin
    if(XXX)
      n_c = 2'b00;
    else
      n_c = 2'b11;
  end
end
```

```
reg [1:0] c[0:15];
reg [1:0] n_c[0:15];
always@(posedge clk) begin
  for (k=0;k<=15;k=k+1)
    c[k] <= n_c[k];
end
always@(*) begin
  for (k=0;k<=15;k=k+1)
      n_c[k] = c[k];
  if(XXX)
    n_c[0] = 2'b11;
  else
    n_c[0] = 2'b10;
end
```

Verilog Memories (Array)

- **A Verilog memory is an array of reg vectors.**

```
reg [MSB:LSB] memory[first_addr:last_addr];
```

- **You can use module parameters to configure the memory**

```
parameter wordsize = 16;
parameter memsize = 1024;
reg [wordsize-1:0] mem [0:memsize-1];
```

- **2D array**

```
reg [MSB:LSB] memory[xf_addr:xl_addr] [xf_addr:xl_addr];
```

```
parameter wordsize = 16;
parameter xsize = 7;
parameter ysize = 7;
reg [wordsize-1:0] mem [0:xsize-1][0:ysize-1];
```

Unsigned and signed number

	Unsigned	Signed
3’b000	0	0
3’b001	1	1
3’b010	2	2
3’b011	3	3
3’b100	4	-4
3’b101	5	-3
3’b110	6	-2
3’b111	7	-1

Unsigned and signed number

- In verilog, unsigned number is default.
- To use signed number

```
input signed[23:0]  xR1, xR2, xI1, xI2;
output signed[23:0]  Ry,  Iy;
reg signed[2:0]  a, c;
reg [2:0]  b;

a = 3'b101;  //-3
b = 3'b101;  //5
c = a + 3'sb111; //-3 + -1 = -4
```

Unsigned and signed number

- There's bid difference if you don't declare a signed number as a signed register.
- **If any of the input is unsigned, the operation is unsigned**
- Example

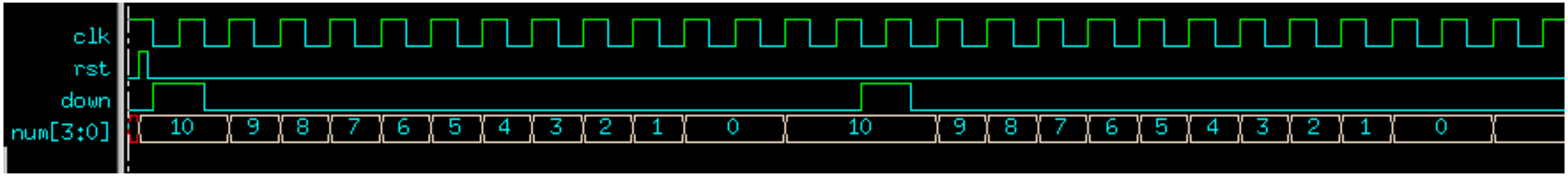
```
reg signed [2:0] a, b;
reg signed [3:0] c;
a = 3'b111; //-1
b = 3'b011; //3
c = a + b;   //1111+0011 = 0010 //2
reg [2:0] d, e;
reg [3:0] f;
d = 3'b111;  //7
e = 3'b011;  //3
f = d + e;   //0111+0011 = 1010 //10
```

Outline

- Introduction
- Verilog for RTL
  - Module Description and Declaration
  - Data Type and Operators
  - Combinational Behavior
  - Sequential Behavior
  - Finite State Machine
  - Advances Topics
- **Design Example**
- RTL Simulation Tool
- Synthesis

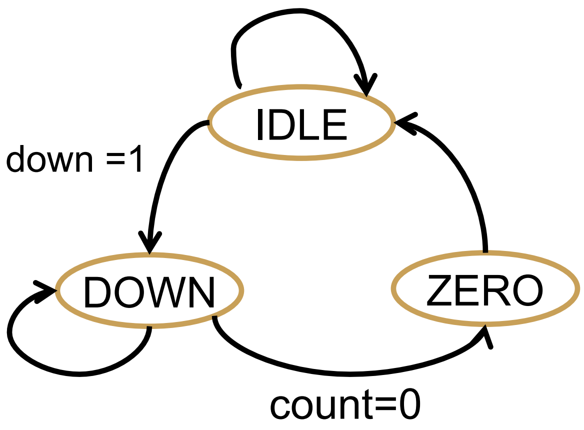
Design Example (1/6)

- **A count down counter.**
  - The circuit has three input port: clk, rst, and down.
  - The circuit has one output port: num.
  - The circuit is synchronous to posedge clk
  - The circuit has asynchronous active high reset
  - If down=1, the circuit begin count from 10 to 0.
  - If the counter count to 0, it will stay at zero 2 clock cycle and back to 10.



Design Example(2/6)

- Step 1: plot the state diagram to describe the circuit behavior.
- Step 2: Estimate the essential sequential element.
- Step 3: Construct the FSM for control signals based on state diagram.
- Step 4: Construct the combinational circuit based on control signals

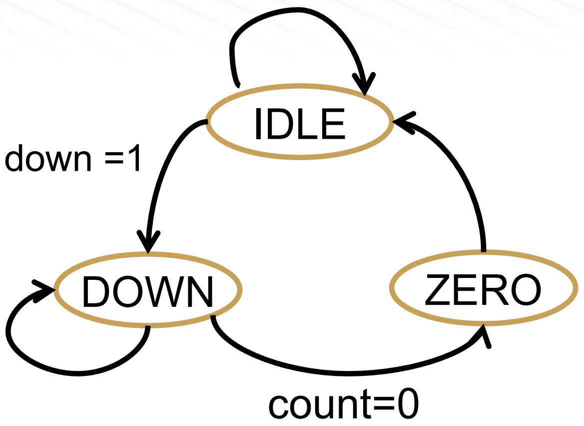


Design Example(3/6)

```
// -------------------------------------
//   Module/ Input/ Ourput/Parameter
// -------------------------------------
module counter( clk, rst, down, num);
// --- Input/Output declaration
  input         clk;
  input         rst;
  input         down;
  output [3:0]  num;

// --- MARCRO declaration for FSM
  parameter IDLE = 2'b00;
  parameter DOWN = 2'd1;
  parameter ZERO = 2'd2;

// --- Wire/ Reg declaration  ---
  reg  [1:0] state, next_state;
  reg  [3:0] count;
  reg  [3:0] next_count;
  reg  [3:0] num,   next_num;
```

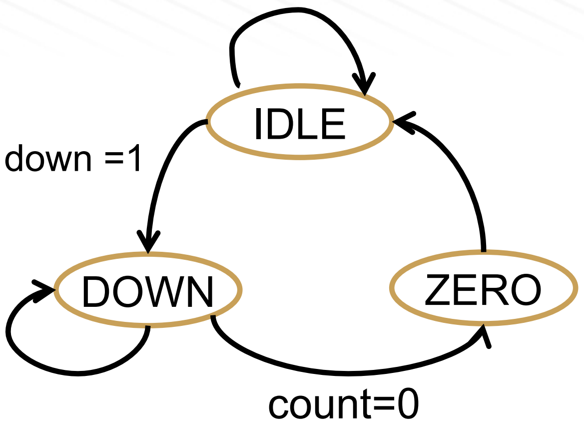


Design Example(4/6)

```
// -----------------------------------
//   Combinational Part
// -----------------------------------
  // continous assignment
  //assign wire_next_num = next_num; continous assignment exam

  // procedural assignment
  // Next State Logic, Output Logic, and Other Control Signal
  always@(*) begin
    case(state)
      IDLE: begin
        if (down==1'b1) next_state = DOWN;
        else next_state = IDLE;
        next_num = 4'd10;
        next_count = (down ==1'b1) ? count-1'b1 :4'd10;
      end
      DOWN: begin
        next_state = (count==4'd0) ? ZERO : DOWN;
        next_count = (count==4'd0) ? 4'd0 : count-1'b1;
        next_num   = count;
      end
      ZERO: begin
        next_state = IDLE;
        next_count = 4'd10;
        next_num   = count;
      end
      default: begin
        next_state = state;
        next_count = count;
        next_num   = num;
      end
    endcase
  end
```

Next state logic



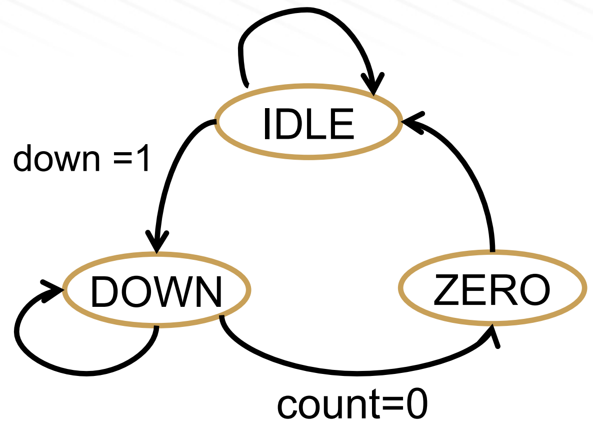

```
// -----------------------------------
//   Sequential Part
// -----------------------------------
  always@(posedge clk or posedge rst)
  begin
    if(rst) begin
      state <= IDLE;
      count <= 4'd10;
      num   <= 4'd10;
    end
    else begin
      state <= next_state;
      count <= next_count;
      num   <= next_num;
    end
  end
endmodule
```

Example of Design (5/6)

```
// ------------------------------------
//   Combinational Part
// ------------------------------------
  // continous assignment
  //assign wire_next_num = next_num; continous assignment exam

  // procedural assignment
  // Next State Logic, Output Logic, and Other Control Signal
  always@(*) begin
    case(state)
      IDLE: begin
        if (down==1'b1) next_state = DOWN;
        else next_state = IDLE;
        next_num = 4'd10;
        next_count = (down ==1'b1) ? count-1'b1 :4'd10;
      end
      DOWN: begin
        next_state = (count==4'd0) ? ZERO : DOWN;
        next_count = (count==4'd0) ? 4'd0 : count-1'b1;
        next_num   = count;
      end
      ZERO: begin
        next_state = IDLE;
        next_count = 4'd10;
        next_num   = count;
      end
      default: begin
        next_state = state;
        next_count = count;
        next_num   = num;
      end
    endcase
  end
```

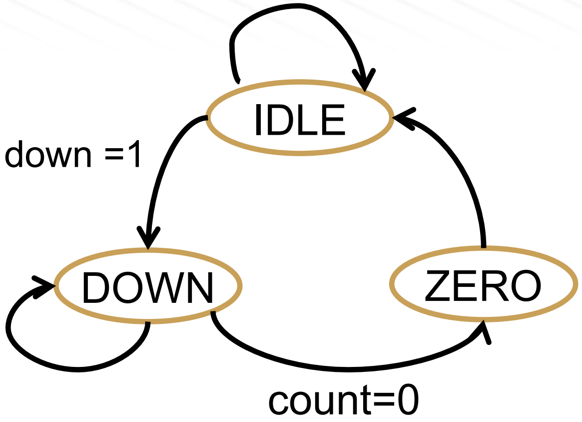
Output logic

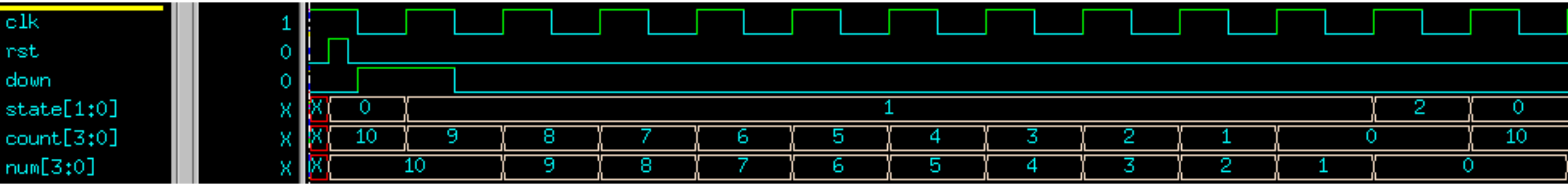



```
// ------------------------------------
//   Sequential Part
// ------------------------------------
  always@(posedge clk or posedge rst)
  begin
    if(rst) begin
      state <= IDLE;
      count <= 4'd10;
      num   <= 4'd10;
    end
    else begin
      state <= next_state;
      count <= next_count;
      num   <= next_num;
    end
  end
endmodule
```

Design Example(6/6)

- **The wave of state, count and num**





Outline

- Introduction
- Verilog for RTL
  - Module Description and Declaration
  - Data Type and Operators
  - Combinational Behavior
  - Sequential Behavior
  - Finite State Machine
  - Advances Topics
- Design Example
- **RTL Simulation Tool**
- Synthesis

Before Setup Environment

- **At lab 231 workstation, you should set environment .**
- **We will use 2 tools in RTL Simulation and Debugging**
  - Cadence **INCISIV (NC-Verilog)**, Verilog Simulator

```
% source /usr/cadence/CIC/incisiv.cshrc
```

  - SpringSoft **nWave**, Waveform viewer

```
% source /usr/spring_soft/CIC/verdi.cshrc
```

Setup Environment

- **Launch “NC-Verilog” to do RTL simulation:**

```
% ncverilog <test bench> <related design> +access+r
```

- **Launch “nWave” to observe simulated waveforms:**

```
% nWave &
```

RTL Simulation (1/7)

- **ncverilog [testbench.v] [design.v] +access+r**

If you need waveform for debugging

- **FSDB format**
  $fsdbDumpfile();
  $fsdbDumpvars();
- **VCD format**
  $dumpfile();
  $dumpvars();

```
initial begin
  // dumping waveform of FSDB format
  $fsdbDumpfile("counter.fsdb");
  $fsdbDumpvars();
  // dumping waveform of VCD format
  $dumpfile("counter.vcd");
  $dumpvars();
  //$sdf_annotate("light_syn.sdf",DUT);
  $display("\n === 2014 Srping ICD Verilog Example === \n");
end
```

In counter_tb.v

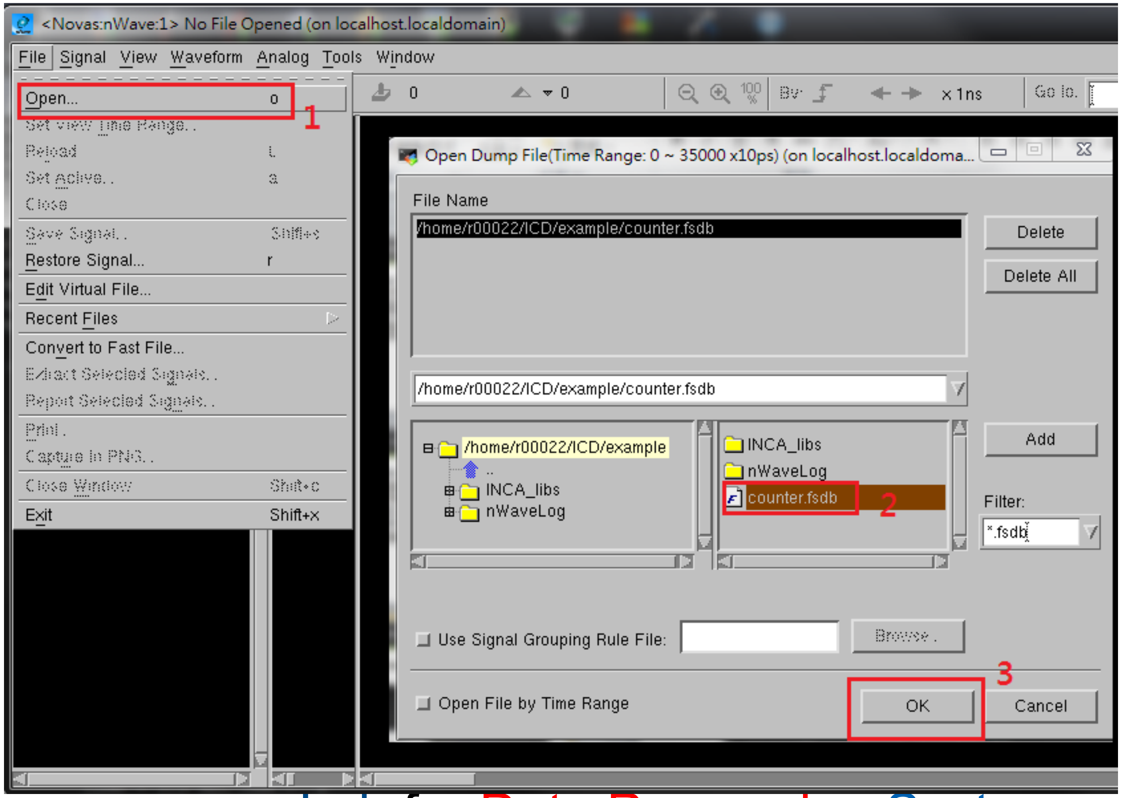
RTL Simulation (2/7)

- **Debug if there is error during simulation**
  – Check out the error message and modify the corresponding RTL code for the syntax errors.
  – Logical errors can be debugged by waveform viewer

```
[r00022@localhost example]$ ncverilog counter_tb.v counter.v +access+r
ncverilog: 10.20-s114: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
Recompiling... reason: file './counter.v' is newer than expected.
        expected: Mon May  6 00:59:19 2013
        actual:   Tue May  7 14:57:52 2013
file: counter_tb.v
        module worklib.light_tb:v
                errors: 0, warnings: 0
file: counter.v
          end
            |
ncvlog: *E,EXPSMC (counter.v,54|5): expecting a semicolon (';') [9.2.2(IEEE)].
        module worklib.counter:v
                errors: 1, warnings: 0
ncverilog: *E,VLGERR: An error occurred during parsing.  Review the log file for errors with the code *E and f
ix those identified problems to proceed.  Exiting with code (status 1).
[r00022@localhost example]$ 
```

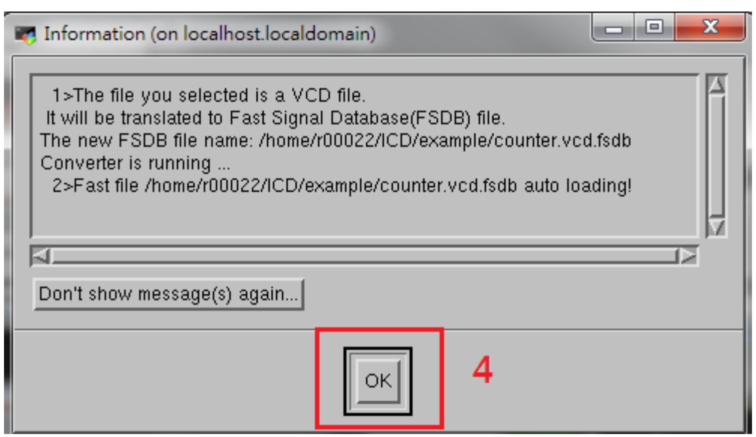
RTL Simulation (3/7)

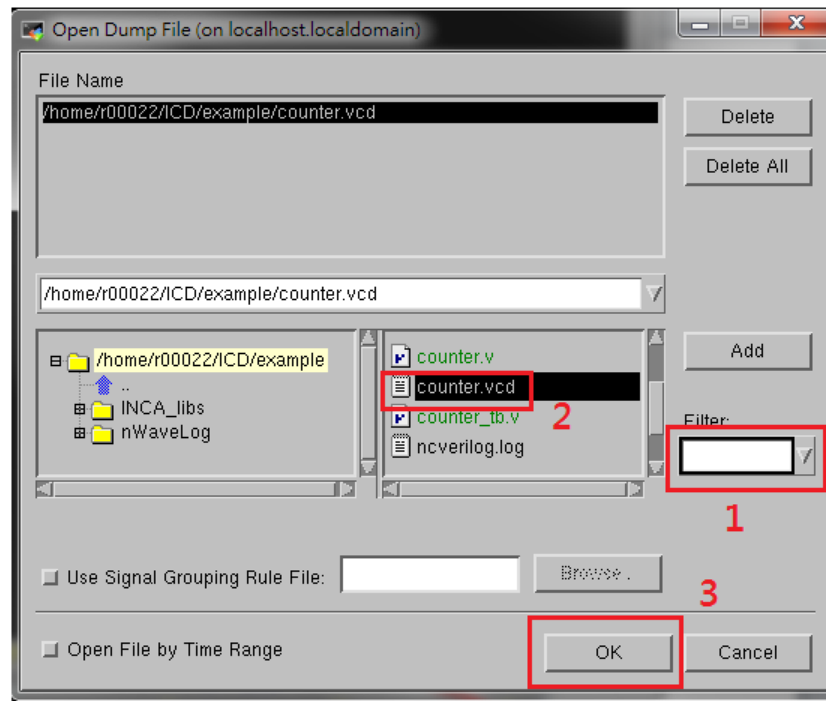
- **Launch the nWave**
  - nWave &
- **Open the generated .fsdb dumped waveform result**
  - File->Open
  - Choose the counter.fsdb
  - Press the OK button



RTL Simulation (4/7)

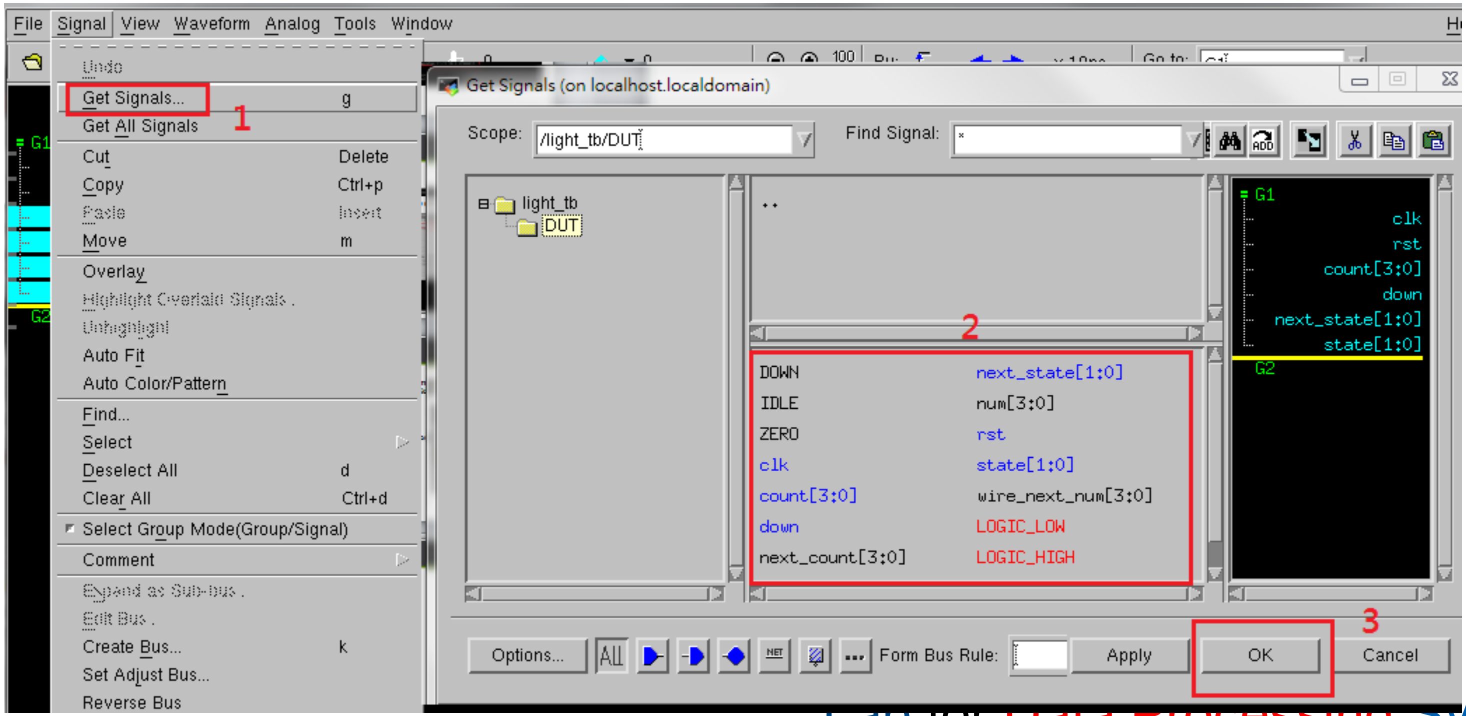
- **Sometimes, the $fsdbDumpfile() doesn't work at the 231 workstations. Please try $dumpfile() for .vcd waveform.**
  - Clear the filter option
  - Choose counter.vcd
  - Press the OK button
  - Press the OK of pop Information.



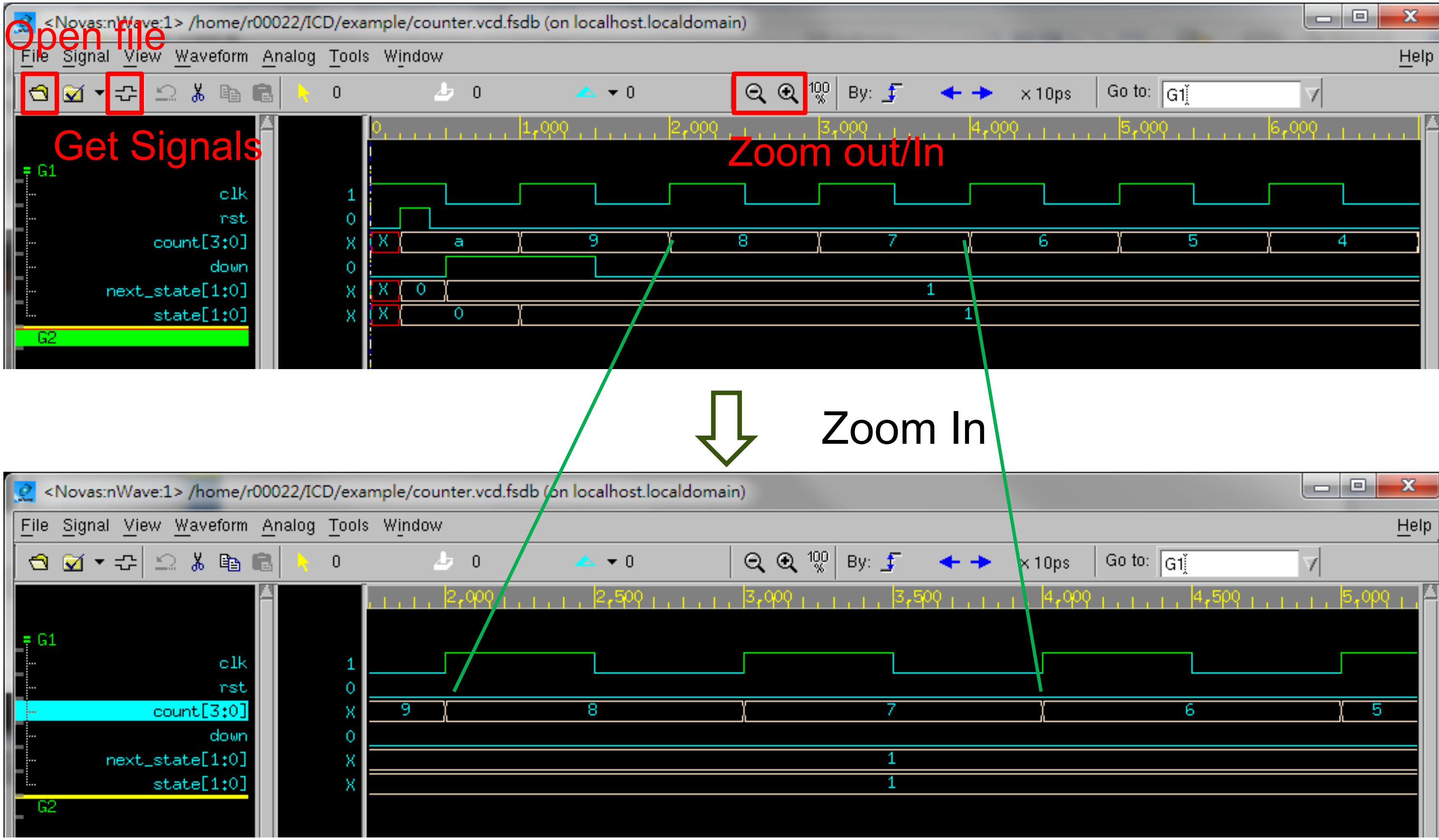


RTL Simulation (5/7)

- **Get the observed signals for debugging**
  - Signal -> Get Signals
  - Choose the signals which you want to observe.
  - Press the OK button.



RTL Simulation (6/7)



RTL Simulation (7/7)

- **Change the Signal's Radix**
  - Choose the signal
  - Waveform->Signal Value Radix->Binary/Octal/Hexadecimal/Decimal