Synthesizable Coding of Verilog

REF:
- Logic Synthesis with Design Compiler, CIC, July, 2008

Speaker: Y. –X. Chen

Nov. 2012
11/21

- 課程主題: Synthesizable Verilog & Coding
- 學習目標
  - Synthesizable coding style in Verilog
  - Syntax check with nLint
- LAB1簡介-撰寫simple 8-bit microprocessor之Verilog code
  - 步驟一:RTL coding並使用nLint確定為可合成之code
  - 步驟二:使用修正好的RTL netlist跑simulation，並觀察波型
Outline

- Basic of Logic Synthesis Concept
- Basic Concept of Verilog HDL
- Synthesizable Verilog
- LAB 1-1: Design Rule Check with nLint
- Tips for Verilog Design
- LAB 1-2: RTL Simulation
Basic Concept of the Synthesis
Cell-Based Design Flow

System Level
- MATLAB/ C/ C++/ System C/
  ADS/ Covergen (MaxSim)

RTL Level
- Verilog/ VHDL
- NC-Verilog/ ModelSim
  Debussy (Verdi)/ VCS
- Design/ Power Compiler
- DFT Compiler/ TetraMAX
- NC-Verilog/ ModelSim
  Debussy (Verdi)/ VCS
- SOC Encounter/ Astro
- GDS II
- DRC/ LVS (Calibre)
- PVS: Calibre xRC/ NanoSim
  (Time/ Power Mill)
- Tape Out

Logic Synthesis
- Conformal/ Formality

Design for Test

Gate Level

Layout Level

Post-Layout Verification

Memory Generator

Syntest

Advanced Reliable Systems (ARES) Lab.
What is Synthesis

Synthesis = translation + optimization + mapping

if(high_bits == 2'b10)begin
    residue = state_table[i];
    end
else begin
    residue = 16'h0000;
    end

The synthesis is constraint driven and technology independent!!
Your RTL design
- Functional verification by some high-level language
  - Also, the code coverage of your test benches should be verified (i.e. VN)
- Coding style checking (i.e. n-Lint)
  - Good coding style will reduce most hazards while synthesis
  - Better optimization process results in better circuit performance
  - Easy debugging after synthesis

Constraints
- The area and timing of your circuit are mainly determined by your circuit architecture and coding style
- There is always a trade-off between the circuit timing and area
- In fact, a super tight timing constraint may be worked while synthesis, but failed in the Place & Route (P&R) procedure
Basic Concept of Verilog HDL
Verilog Model

- Key features of Verilog
  - Supports various level of abstraction
    - Switch level model or transistor level model
    - Gate level model
    - Data flow model or register transfer model
    - Behavioral model
Register Transfer Level (RTL)

```verilog
// BEH_AND2
module AND2(in1,in2,Out)
input in1,in2;
output Out;
reg Out;
always @(in1 or in2)
begin
    Out = in1 & in2;
end
endmodule
```
Gate Level Model

- Model consists of basic logic
  - Ex. AND, NAND, OR, NOR, XOR, NOT, etc.

```verilog
module AND2(in1,in2,Out);
  input in1,in2;
  output Out;
  and u1(Out,in1,in2);
endmodule
```
Verilog Module

module module_name(port_names);

• Port declaration
• Data type declaration
• Task & function declaration
• Module functionality or structure
• Timing Specification

endmodule

/* This is sample code. The function is ALU. */
module ALU(a,b,sel,out);
input [7:0] a,b; //Data in
output [7:0] out; //Data out
input [2:0] sel; //Control select

reg [7:0] out;
wire ... ...
always@(...) begin ...
end
endmodule
Verilog Syntax

- Verilog consists of a series token

  - Comment: //, /* * /

  - Operators: unary, binary, ternary
    - A=~B;
    - A=B&C;
    - C=SEL?A:B;

  - Numbers: size, unsized
    - Sized: 4'b0010, 8'ha

  - Identifiers: $, #, etc.

  - Keywords

  - ...
Verilog Syntax (Cont’d)

- always@ statement
  - Blocking
  - Non-blocking

```verilog
always @ (posedge clk) begin
  x_temp<=x;
end

always @ (a or x_temp) begin
  if (a) begin
    x = x_temp+1’b1;
  end
  else begin
    x = x_temp;
  end
end
```
Verilog Syntax (Cont’d)

- Case statement
- If-else statement

```
always @ (d) begin
  case (d)
    2'b00: z=1'b1;
    2'b01: z=1'b0;
    default : z=1'b0;
  endcase
end

always @ (a or x_temp)begin
  if (a) begin
    x= x_temp+1'b1;
  end
  else begin
    x= x_temp;
  end
end
```
Connection Manners
Synthesizable Verilog
Importance of Coding Style

- Make sure your code is readable, modifiable, and reusable
- Good coding style helps to achieve better results in synthesis and simulation
Concept of Clocks and Reset

Synchronous

Mixed Clock Edges

Combination Feedback

Gated Clocks

Avoid !!

Avoid !!

Not Allow !!
Asynchronous and Synchronous Reset

- **Synchronous reset**

  ```verilog
  always @(posedge clock) begin
    if (rst) begin
      ..............
    end
    ...
  end
  ```

- **Asynchronous reset**

  ```verilog
  always @(posedge clock or negedge reset) begin
    if (!rst) begin
      ..............
    end
    ...
  end
  ```
Synthesizable Verilog

- Not all kinds of Verilog constructs can be synthesized
- Only a subset of Verilog constructs can be synthesized and the code containing only this subset is synthesizable
Synthesizable Verilog (Cont’)

- **Verilog Basis**
  - parameter declarations
  - wire, wand, wor declarations
  - reg declarations
  - input, output, inout
  - continuous assignment
  - module instructions
  - gate instructions
  - always blocks
  - task statement
  - function definitions
  - for, while loop

- **Synthesizable Verilog primitives cells**
  - and, or, not, nand, nor, xor, xnor
  - bufif0, bufif1, notif0, notif1

- **Can not use for Synthesis**
  - ===
  - !==
  - delay
  - Initial
  - repeat
  - forever
  - wait
  - fork
  - join
  - event
Operators

- Concatenation ( { }, { { } } )
- Unary reduction ( !, ~, & , |, ^ )
- 2’s complement arithmetic ( +, -, * )
- Logic shift ( >>, << )
- Relational ( >, <, >=, <= )
- Equality ( ==, != )
- Binary bit-wise ( & , |, ^, ~^ )
- Logical ( && , || )
- Conditional ( ?: )

Precedence:

- Highest
- Lowest
Coding for Synthesis

- **Combinational Blocks**
  ```verilog
  always @ (d) begin
    case (d)
      2'b00: z=1'b1;
      2'b01: z=1'b0;
      default : z=1'b0;
    endcase
  end

  always @ (a or x_temp)begin
    if (a) begin
      x= x_temp+1'b1;
    end
    else begin
      x= x_temp;
    end
  end
  ```

- **Sequential Blocks**
  ```verilog
  always @ (posedge clk )begin
    if (a) begin
      z<=1'b1;
    end
    else begin
      z<=1'b0;
    end
  end
  ```
Avoid Combinational Feedback

```verilog
always @ (a or x)begin
    if (a) begin
        x = x + 1'b1;
    end
    else begin
        x = x;
    end
end
```

```verilog
always @ (posedge clk) begin
    x_temp <= x;
end
```

```verilog
always @ (a or x_temp)begin
    if (a) begin
        x = x_temp + 1'b1;
    end
    else begin
        x = x_temp;
    end
end
```
Coding for Synthesis (Cont’)

- **Blocking Assignment**

```verilog
code
always @ (posedge clk )begin
  b = a;
  c = b;
end
```

- **Non-Blocking Assignment**

```verilog
code
always @ (posedge clk )begin
  b <= a;
  c <= b;
end
```

Just like “a=c;”

Just like “shift register”
Coding for Synthesis (Cont’)

Avoid Latches

```verilog
always @ (d) begin
  case (d)
    2'b00: z=1'b1;
    2'b01: z=1'b0;
    default : z=1'b0;
  endcase
end

always @ (posedge clk )begin
  if (a) begin
    ...........
  end
  else begin
    ...........
  end
end
```

Advanced Reliable Systems (ARES) Lab.
Coding for Synthesis (Cont’)

Sensitivity List

always @ (d) begin
  case (d)
    2'b00: z=1'b1;
    2'b01: z=1'b0;
    default : z=1'b0;
  endcase
end

always @ (a or b or c or d)begin
  if (a) begin
    ............
    end
  else begin
    if (b)begin
      z=c;
      end
    else begin
      z=d;
      end
    end
  end
end
Coding for Synthesis (Cont’)

- **Case statements**

  ```
  always @ ( sel or a or b or c or d) begin
    case (sel)
      2'b00: out=a;
      2'b01: out=b;
      2'b10: out=c;
      2'b11: out=d;
    endcase
  end
  ```

- **if – else statements**

  ```
  always @ ( sel or a or b or c or d) begin
    if (sel==2'b00) out=a;
    else if (sel==2'b01) out=b;
    else if (sel==2'b10) out=c;
    else out=d;
  end
  ```

---

Advanced Reliable Systems (ARES) Lab.
Lab 1-1
Design Rule Check with nLint
Design Rule Check

- Use nLint tool (include by Debussy) and the Verilog Coding Guideline to check your design and modify parts of code to match the coding guidelines
Start nLint

Unix% nLint –gui &
Load Verilog Code (1/2)
Load Verilog Code (2/2)
Run nLint Check

Compile

Advanced Reliable Systems (ARES) Lab.
nLint Check Result (1/2)

Clock Domain Analysis
Total – 14 Error(s), 3 Warning(s)
- Simulation – 7 Error(s)
  - 22139 – Constant Event Expression – Simulation, Language Construct – 6 Error(s)
  - 23011 – Incomplete Sensitivity List – Simulation, Synthesis – 1 Error(s);
    mytraffic_light.v(68) : Error : signal "ST_out" should be included in the sensitivity list.
- Synthesis – 1 Error(s)
- Design Style – 3 Warning(s)
- Language Construct – 6 Error(s)
nLint Check Results (2/2)

```
always @(state or nextstate)
begin
    st_o = ~(state == nextstate);
end

always @(posedge clock or posedge st) // counter
begin
    if (st) 
        count = 0;
    else
        count = count + 5'd1;
end

always @(posedge clock or posedge reset) // for next state
begin
```

Lint processing done.
Loading report DB file ...
Totally 2 Error(s), 3 Warning(s) loaded
Loading report DB file, done!

Total 2 Error(s), 3 Warning(s)
Lab Time
11/28

- 課程主題: Synthesizable Verilog & Coding
- 學習目標
  - Tips for Verilog Design
  - RTL simulation
  - Waveform viewer – nWave / Debussy
- LAB1簡介-撰寫simple 8-bit microprocessor之Verilog code 並模擬結果
  - 步驟一: RTL coding並使用nLint確定為可合成之code
  - 步驟二: 使用修正好的RTL netlist跑simulation，並觀察波型
Outline

- Basic of Logic Synthesis Concept
- Basic Concept of Verilog HDL
- Synthesizable Verilog
- LAB 1-1: Design Rule Check with nLint
- Tips for Verilog Design
- LAB 1-2: RTL Simulation
Tips for Verilog Design
Pre-RTL Preparation Checklist

☐ Communicate design issues with your team
  ■ Naming conventions, revision control, directory tree and other design organizations

☐ Have a specification for your design
  ■ Everyone should have a specification before they start coding

☐ Design partition
  ■ Follow the specification’s recommendations for partition
  ■ Break the design into major function blocks
RTL Coding Style

- Create a block level drawing of your design before you begin coding
  - Draw a block diagram of the function and sub-function of your design
- Always think of the poor guy who has to read your RTL code
  - Correlate “top to bottom in the RTL description” with left to right in block diagram
  - Comments and headers
- Hierarchy design
Basic Coding Practices

- Naming Conventions
  - Use lowercase letters for all signal names, and port names, versus uppercase letters for names of constants and user-defined types
  - Use meaningful names
  - For active low signals, end the signal name with an underscore followed by a lowercase character (e.g., rst_ or rst_n)
  - Recommend using “bus[X:0]” for multi-bit signals
Basic Coding Practices (Cont’)

- Include Headers in Source Files and Comments

```vhdl
module bist(
    clk,
    rst,
    CSI,
    Do,
    hold,
    WEN_T,
    CS_T,
    OE_T,
    D1_T,
    ADDR_T,
    cmd_done,
    SYN,
    Fail,
    test_done);

    Parameter declarations:
    parameter INIT_ADDR_NUM = 8'b00000000;
    parameter FIN_ADDR_NUM = 8'b11111111;
    parameter WORD_LEN = 8;
    parameter ADDR_LEN = 8;
    parameter ROW_ADDR_LEN = 4;
    parameter COL_ADDR_LEN = 4;
    parameter BIT_ADDR_LEN = 3;
    parameter EXP_COUNT = 5'b10100;

    IO declarations:
    //BIST input control signals
    input clk;
    input rst;
    input CSI;

    //INITIAL ADDRESS OF THE ADDR COUNTER
    //FINAL ADDRESS OF THE ADDR COUNTER
    //WORD LENGTH
    //ADDRESS LENGTH (ROW_ADDR_LEN+COL_ADDR_LEN)
    //ROW ADDRESS LENGTH
    //COLUMN ADDRESS LENGTH
    //BIT ADDRESS LENGTH
    //EXPORTATION COUNT (WORD_LEN+ADDR_LEN+BIT_ADDR_LEN+1)

    //SYSTEM CLOCK
    //MBIST RESET
    //COMMAND SERIAL INPUT
```

Advanced Reliable Systems (ARES) Lab.
Basic Coding Practices (Cont’)

- Indentation

```verilog
//_______SERIAL READ COUNTER_____
always@(posedge clk or posedge rst)
begin
  if(rst)begin
    ser_read_count <= 4’b0000;
  end
  else begin
    if(hold|self_hold)begin
      ser_read_count <= 4’b0000;
    end
    else begin
      if (CS_T || ((CS_T)&&(addr_change))) begin
        ser_read_count <= ser_read_value;
      end
      else begin
        ser_read_count <= ser_read_count - 1’b1;
      end
    end
  end
end
```

- Port Maps and Generic Maps

```verilog
//___ ctr module: programmable_ctr ___
programmable_ctr programmable_ctr ( 
  .mar_or_xf(mar_or_xf),
  .rst_bist(rst_bist),
  .bsc(bsc),
  .bsi(bsi),
  .clk(clk),
  .test_done(test_done),
  .shift(shift),
  .fail(fail),
  .final_addr(final_addr),
  .final_data(final_data),
  .Comp(Comp),
  .data_type(data_type),
  .addr_type(addr_type),
  .a_count_shift(a_count_shift),
  .a_up_down(a_up_down),
  .a_left_right(a_left_right),
  .a_hold(a_hold),
  .d_hold(d_hold),
  .d_left_right(d_left_right),
  .CEN(CEN),
  .WEN(WEN),
  .OEN(OEN),
  .CEN_b(CEN_b),
  .WEN_b(WEN_b),
  .OEN_b(OEN_b)
);
```
Basic Coding Practices (Cont’)

- Use Functions or Tasks
  - Which Instead of repeating the same sections of code
Write Efficient HDL Code

- Use parentheses control complex structure of a design
- Resource Sharing
- Scalable design and propagate constant value
- Use operator bit-width efficiently
- Timescale
Use Parentheses Properly

- out = a + b + c + d + e
- out = ((a + (b + c)) + (d + e));
Resource Sharing

- Operations can be shared if they lie in the same always blocks

```plaintext
Always @(sel or a or b or c )
begin
    if (sel) z=a+b;
    else    z=a+c;
end
```
Scalable Design & Constant

```
parameter cb_size=8;
parameter data_size=64;
parameter address_size=13;

input clk;
input cen;
input wen;
input oen;
input [address_size-1:0]address;
input [data_size-1:0]data;
//
output [data_size-1:0]Q;
output ed;
output dec;
```

```
parameter size=8;
wire [3:0] a,b,c,d,e;

assign a=size+2;
assign b=a+1;
assign c=d+e;
```

Constant Increaser Adder
module fixed_multiplier(a, b, c);
input [8:0] a, b;
output [8:0] c;
reg [15:0] tmp;
reg [8:0] c;
assign tmp = a*b;
assign c = tmp[15:8];
endmodule
Timescale

- `timescale`: which declares the time unit and precision.
  - `timescale <time_unit> / <time_precision>`
  - e.g.: `timescale 1s/1ps`, to advance 1 sec, the timewheel scans its queues $10^{12}$ times versus a `timescale 1s/1ms`, where it only scans the queues $10^3$ times.

- The time_precision must be at least as precise as the time_unit.
- Keep precision as close in scale to the time units as is practical.
- If not specified, the simulator may assign a default timescale unit.
- The smallest precision of all the timescale directive determines the “simulation time unit ” of the simulation.
Omit for Synthesis

☐ Omit the Wait for XX ns Statement
   ■ Do not use “#XX;”

☐ Omit the ...After XX ns or Delay Statement
   ■ Do not use “assign #XX Q=0;”

☐ Omit initial values
   ■ Do not use “initial sum = 1’b0;”
Non-Synthesizable Style

- Either non-synthesizable or incorrect after synthesis
- **initial** block is forbidden (non-synthesizable)
- Multiple assignments (multiple driving sources)
- Mixed blocking and non-blocking assignment
Summary

- No initial in the RTL code
- Avoid unnecessary latches
- Avoid combinational feedback
- For sequential blocks, use non-blocking statement
- For combinational blocks, use blocking statements
Lab 1-2
RTL Simulation
Tools

- Simulators
  - Verilog-XL, NC-Verilog, Altera Quartus, ModelSim and etc.

- Synthesizers
  - Design vision, Ambit, and etc.

- Debugger and verification tools
  - Debussy, nWave, nLint, and etc.
  - nLint can check the correctness of your code’s syntax
Verilog Simulator

Circuit description
module cpu(clk,rst,...)
input clk, rst;
output sum, c;
....
....
endmodule

Testbench
module stimuli;
reg clk, rst;
wire sum, c;
@(clk)
....
....
endmodule

Verilog simulator

Waveform simulation result
Text simulation result

0ns sum = x; c = x;
1ns sum = 0; c = 1;
2ns sum = 0; c = 1;
......
Run Verilog Simulation (1/2)

- **Method 1:**
  - `unix% verilog alu.v t_alu.v`
  - `unix% ncverilog +access+r alu.v t_alu.v`
- **Method 2:**
  - Using additional file `alu.f`
    - `alu.v`
    - `t_alu.v`
  - `unix% verilog -f alu.f`
  - `unix% ncverilog +access+r -f alu.f`
- **Method 3:**
  - Using additional description `include “module_file”`
Run Verilog Simulation(2/2)

No syntax error

1 Syntax error!
Testbench

- Compare this with your design

module testfixture;
  • Declare signals
  • Instantiate modules
  • Applying stimulus
  • Monitor signals
endmodule
FSDB File

- Waveform file format
- Add commands in testbench

```v
// testbench.v
module ...();
...
initial begin
    $fsdbDumpfile("abcd.fsdb");
    $fsdbDumpvars;
End
...
endmodule
```
Example of Testbench

//alu.v
/* This is sample code. The function is ALU.
 */
module ALU(a, b, sel, out);
input [7:0] a, b; // Data in
output [7:0] out; // Data out
input [2:0] sel; // Control select

reg [7:0] out;
wire ...
...
always@(...)begin
...
end
...
endmodule

//t_alu.v
/* This is testbench of sample code. The function is ALU.
 */
module test_ALU;
reg [7:0] A, B;
reg[2:0] SEL;
wire[7:0] OUT;

ALU U0(.a(A),.b(B),.sel(SEL),.out(OUT));
always #5 B=~B;
initial
begin
A=0;B=0;SEL=0;
#10 A=0;SEL=1;
#10 SEL=0;
.....
#10 SEL=1;
  #10 $finish;
end
initial begin
$fsdbDumpfile("ALU.fsdb");
$fsdbDumpvars;
end
endmodule
Debussy – Getting Start

- Using nWave or Debussy
  - `unix% nWave&`
  - `unix% debussy&`

![Image of nWave and Debussy interface]

Advanced Reliable Systems (ARES) Lab.
Select “Signal” -> “Get Signal”
Observe Waveform
Change Radix
Save Waveform
LAB Time