Lecture 10
Logic Synthesis, Part 1

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese461/

Outline

Write Synthesizable Code

Write Synthesis Script

Synthesizable Verilog

- Top-down design
  - top-level module
  - example: module MyChip_ASIC();
  - lower-level modules
  - initially empty placeholders, or stubs

```
module MyChip_ASIC()
// behavioral "always", etc. ...
SecondLevelStub1 port mapping
SecondLevelStub2 port mapping
... endmodule
module SecondLevelStub1() ... assign Output1 = ~Input1; endmodule
module SecondLevelStub2() ... assign Output2 = ~Input2;
endmodule
```

- Functionally identical or equivalent

Logic Circuit Synthesis

- Combinational circuit
  - continuous assignment

```
assign y = (a&b) | (c^d);
```

  - always statement
  - contains only signals with no edges
  - level-sensitive list

```
module And_Always(x, y, z); input x,y; output z; reg z;

always @(x or y) z <= x & y; // combinational logic method 1

endmodule
```

Logic Circuit Synthesis

- Combinational circuit
  - concatenation or bit reduction

```
module And_Always(x, y, z); input x,y; output z; reg z;
always @(x or y) z <= x & y; // combinational logic method 1
endmodule

module And_Or (a,b,c,z); input a,b,c; output z; reg [1:0]z;
always @(a or b or c) begin z[1]<= &{a,b,c}; z[2]<= |{a,b,c}; end
endmodule
module Parity (BusIn, outp); input [7:0] BusIn; output outp; reg outp;
always @(BusIn) if (^Busin == 0) outp = 1; else outp = 0;
endmodule
```

Logic Circuit Synthesis

- Example 3-8 decoder

```
module decoder (in,out);
input [2:0] in;
output [7:0] out;
wire [7:0] out;
assign out =       (in == 3'b000 ) ? 8'b0000_0001 :
(in == 3'b001 ) ? 8'b0000_0010 :
(in == 3'b010 ) ? 8'b0000_0100 :
(in == 3'b011 ) ? 8'b0000_1000 :
(in == 3'b100 ) ? 8'b0001_0000 :
(in == 3'b101 ) ? 8'b0010_0000 :
(in == 3'b110 ) ? 8'b0100_0000 :
(in == 3'b111 ) ? 8'b1000_0000 : 8'h00;

endmodule
```

Logic Circuit Synthesis

- Don’ts
  - miss signals in the sensitivity list
  - use variables before assignment

```
module And_Bad(a, b, c); input a, b; output c; reg c;
always @(a) c <= a & b; // b is missing from this sensitivity list
endmodule

module CL_good(a, b, c); input a, b; output c; reg c;
always @(a or b)
begin c = a + b; d = a & b; e = c + d; end // c, d: LHS before RHS
endmodule
module CL_bad(a, b, c); input a, b; output c; reg c;
always @(a or b)
begin e = c + d; c = a + b; d = a & b; end // c, d: RHS before LHS
endmodule
```

Logic Circuit Synthesis

- Multiplexers
  - imply a MUX using case

```
module Mux_21a(sel, a, b, z); input sel, a , b; output z; reg z;
always @(a or b or sel)
begin case (sel) 1'b0: z <= a; 1'b1: z <= b; end
endmodule
```

  - if not always assign to an input, might imply latch

```
module Mux_Latch(sel, a, b, z); input sel, a, b; output z; reg z;
always @(a or sel) begin if (sel) z <= a; end
endmodule
```

Logic Circuit Synthesis

- Case statement
  - include default to be exhaustive
  - don't care x, gives synthesizer flexibility

```
module case8_oneHot(oneHot, a, b, c, z);
input a, b, c; input [2:0] oneHot; output z; reg z;
always @(oneHot or a or b or c)
begin case (oneHot) //synopsys full_case
3'b001: z <= a; 3'b010: z <= b; 3'b100: z <= c;
default: z <= 1'bx; endcase
end
endmodule
```

Logic Circuit Synthesis

- Sequential logic
  - edge sensitive always block, with non-blocking
- Example: D Flip Flop
- Don't use "@ clk"
  - will have latch
  - we hate latch

```
module flif_flop (clk,reset, q, d);
input clk, reset, d;
output q;
reg q;

always @ (posedge clk )
begin
  if (reset == 1) begin
    q <= 0;
  end else begin
    q <= d;
  end
end

endmodule
```

Logic Circuit Synthesis

- Sequential logic
  - initialize the memory element by specifying reset
  - use a template

```
always @( posedge clk or negedge reset) begin // template for reset:
if (reset == 0) Q = 0; // initialize,
else Q = D; // normal clocking
end
```

Logic Circuit Synthesis

- Datapath synthesis
  - bus-wide arithmetic and other bus-wide operations
  - example: 32-bit multiplier
  - Synopsys DesignWare library
  - use directives

```
module DP_csum(A1,B1,Z1); input [3:0] A1,B1; output Z1; reg [3:0] Z1;
always @(A1 or B1) Z1 <= A1 + B1;//Compass adder_arch cond_sum_add
endmodule
module DP_ripp(A2,B2,Z2); input [3:0] A2,B2; output Z2; reg [3:0] Z2;
always @(A2 or B2) Z2 <= A2 + B2;//Compass adder_arch ripple_add
endmodule
```

Logic Circuit Synthesis

- Hand instantiation
  - generic logic netlist
  - to ensure a particular cell be included

```
module Count4(clk, reset, Q0, Q1, Q2, Q3);
input clk, reset; output Q0, Q1, Q2, Q3; wire Q0, Q1, Q2, Q3;
//          Q , D , clk, reset
asDff dff0( Q0, ~Q0, clk, reset); // The asDff is a
asDff dff1( Q1, ~Q1, Q0, reset); // standard component,
asDff dff2( Q2, ~Q2, Q1, reset); // unique to one set of tools.
asDff dff3( Q3, ~Q3, Q2, reset);
endmodule
```

Finite-State Machine Synthesis

- Encoding style

Binary encoding : each state is represented in binary code (i.e. 000, 001, 010....)
Gray encoding : each state is represented in gray code (i.e. 000, 001, 011,...)
One Hot : only one bit is high and the rest are low (i.e. 0001, 0010, 0100, 1000)
One Cold : only one bit is low, the rest are high (i.e. 1110,1101,1011,0111)

- Style template
  - using two always blocks
  - using single always blocks
  - using a function for combinational logic

- Example
  - http://www.asic-world.com/tidbits/verilog_fsm.html

Non-Synthesizable Constructs

- Initial, events: used only in test bench
- Real, time (delay with #)
  - data not supported by synthesizer
- Comparison to x and z

```
if ((b == 1'bz) || (b == 1'bx))
```

- Assign (to register)
- Fork – join
  - test bench only
- Primitives
  - only gate level primitives are supported
- Table

Write Synthesizable Code

- Use meaningful names for signals and variables
- Don't mix level and edge sensitive elements in the same always block
- Avoid mixing positive and negative edge-triggered flip-flops
- Use parentheses to optimize logic structure
- Use continuous assign statements for simple combo logic
- Use nonblocking for sequential and blocking for combo logic
- Don't mix blocking and nonblocking assignments in the same always block (even if Design compiler supports them!!)
- Be careful with multiple assignments to the same variable
- Define if-else or case statements explicitly

Synopsys Design Compiler

- Tutorial posted on class website
- Follow the step-by-step instruction before Wed lecture

Questions?

Comments?

Discussion?