INTRODUCTION TO VERILOG

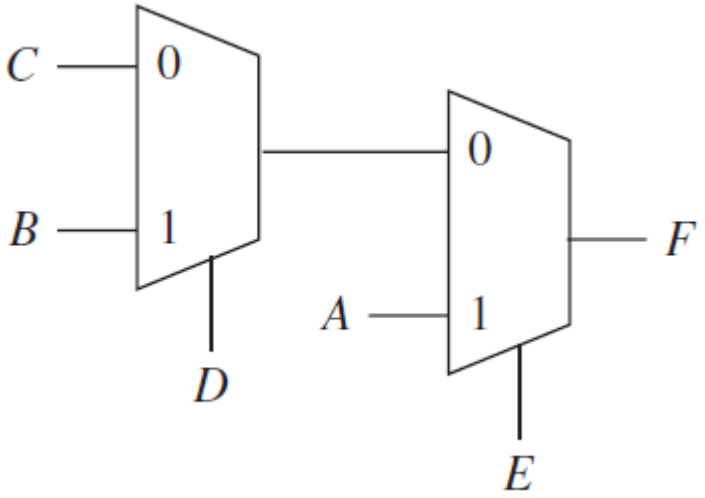
Synthesis examples

```
assign signal_name = condition ? expression_T : expression_F;
```

```
assign F = E ? A : ( D ? B : C );
// nested conditional assignment
```

Synthesis examples

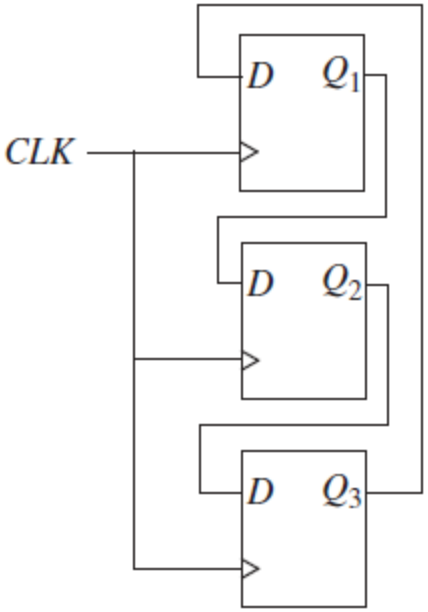
Cascaded 2 to 1 MUXes using conditional assignment




```
assign F = E ? A : ( D ? B : C );
// nested conditional assignment
```

Synthesis examples

Cyclic shift register




```
always @ (posedge CLK)
begin
   Q1 <= #5 Q3;
   Q2 <= #5 Q1;
   Q3 <= #5 Q2;
end
```

Synthesis examples

At the rising edge of the clock – all of the D inputs loaded into the Flip-flops – state change does not occur until after a propagation delay.

Non-blocking assignment – order of the statements not important.

Synthesis examples

```
module reg3 (Q1,Q2,Q3,A,CLK);
input   A;
input   CLK;
output  Q1,Q2,Q3;

reg     Q1,Q2,Q3;

always @(posedge CLK)
begin
  Q3 = Q2;  // statement 1
  Q2 = Q1;  // statement 2
  Q1 = A;  // statement 3

end

endmodule
```

Synthesis examples

```
module reg3 (Q1,Q2,Q3,A,CLK);
input   A;
input   CLK;
output  Q1,Q2,Q3;

reg     Q1,Q2,Q3;

always @(posedge CLK)
begin
  Q3 = Q2;  // statement 1
  Q2 = Q1;  // statement 2
  Q1 = A;  // statement 3

end

endmodule
```

3 bit shift register.

Blocking operator – execution of statements from top to bottom in order.

Synthesis examples

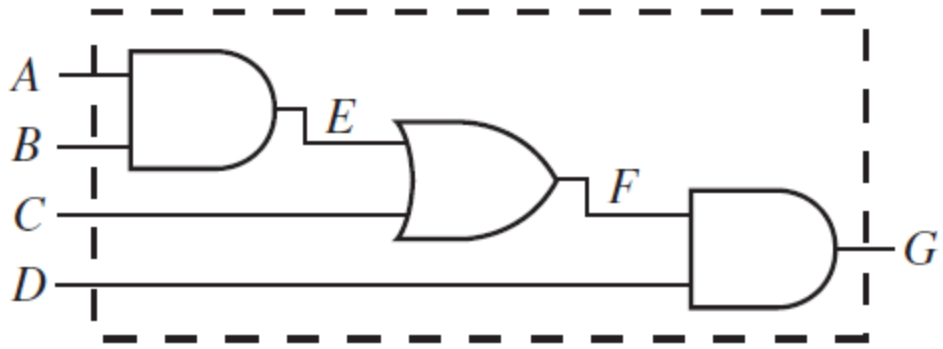
```
module reg31 (Q1,Q2,Q3,A,CLK);
input   A;
input   CLK;
output  Q1,Q2,Q3;

reg     Q1,Q2,Q3;

always @(posedge CLK)
begin

  Q1 = A;  // statement 1
  Q2 = Q1;  // statement 2
  Q3 = Q2;  // statement 3

end

endmodule
```

Synthesis examples

```
module reg31 (Q1,Q2,Q3,A,CLK);
input   A;
input   CLK;
output  Q1,Q2,Q3;

reg     Q1,Q2,Q3;

always @(posedge CLK)
begin

   Q1 = A;  // statement 1
   Q2 = Q1;  // statement 2
   Q3 = Q2;  // statement 3

end

endmodule
```

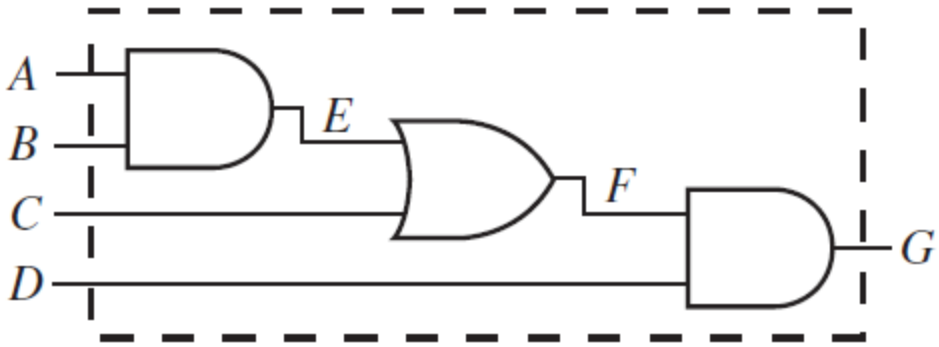
Single flip-flop

Synthesis examples



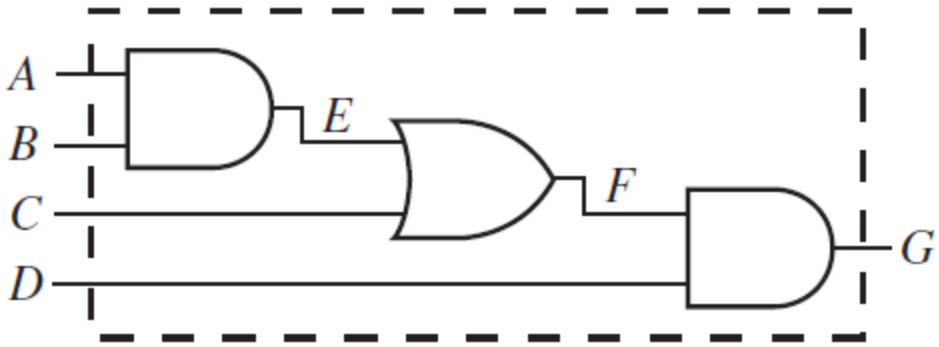
1. Verilog code using concurrent statements.
2. Using an always block.

Synthesis examples




```
module circuit(A, B, C, D, G);

input A, B, C, D;
output G;
wire E, F;

assign E = A && B;
assign F = E || C;
assign G = D && F;

endmodule
```

Synthesis examples




```
module circuit(A, B, C, D, G);

input A, B, C, D;
output reg G;
reg E, F;

initial begin
E <= 0;
F <= 0;
G <= 0;
end

always @(*)
begin
E <= A & & B;
F <= E | | C;
G <= F & & D;
end

endmodule
```

Writing test bench

- Two types of verification – functional and timing verification.
- Functional verification – study the circuit logical operation independent of timing considerations.
- Timing verification – study the circuit operation, including the effect of delays through the gates.

Writing test bench

Test bench for describing and applying stimulus to an HDL model of the circuit – test and observe its response during simulation.

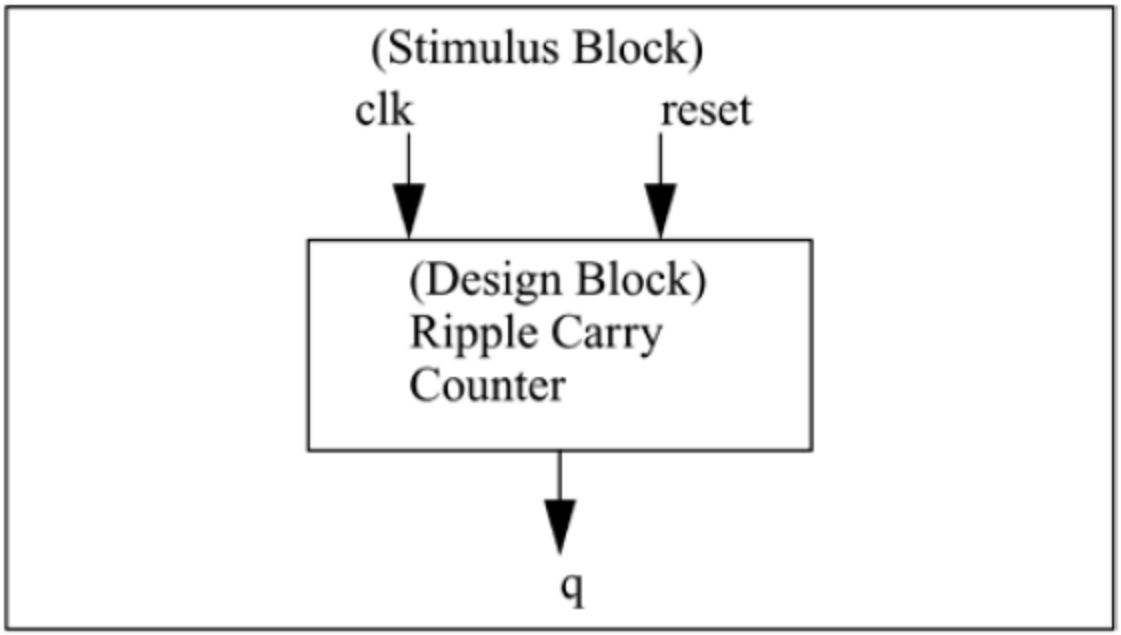
Test benches – van be quiet complex, lengthy.

Care must be taken to –write stimuli that will test a circuit thoroughly

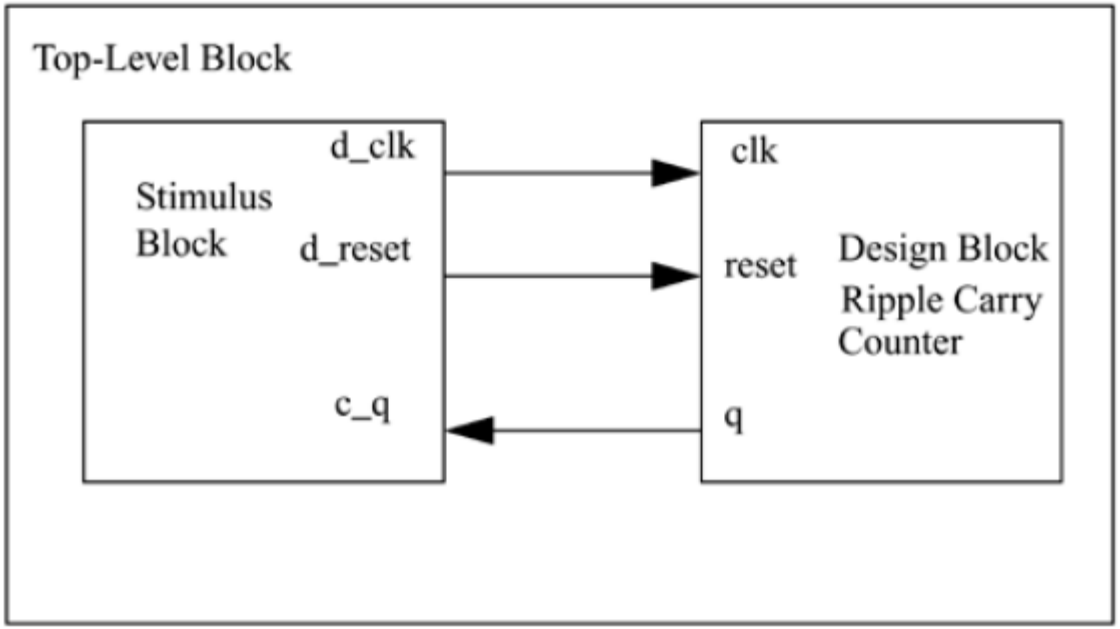
Exercise all the operating features that are specified.

Hierarchical modeling concepts

Stimulus block instantiating design block.



Hierarchical modeling concepts



Hierarchical modeling concepts

- Instantiating both the stimulus and design blocks in a top-level dummy module.
- Stimulus block interacts with the design block through interface.
- Top level block simply instantiates the design and stimulus block.

Writing test bench

```
initial
begin
A  = 0; B = 0;
#10 A = 1;
#20 A  = 0; B  = 1;
end

initial
begin
D = 3'b000;
repeat (7)
#10 D =  D + 3'b001;
end
```

Stimulus test module

module test_module_name;

// Declare local **reg** and **wire** identifiers.
// Instantiate the design module under test.
// Specify a stopwatch, using $finish to terminate the simulation.
// Generate stimulus, using **initial** and **always** statements.
// Display the output response (text or graphics (or both)).

endmodule

Stimulus test module

- Test module like any other module – but typically has no inputs or outputs.
- Local reg data types, local reg wire types.
- Instantiate the module under test – using local identifiers in the port list.
- Simulator associates the local identifiers within the test bench with the formal identifiers of the module.
- Response to the stimulus – appear in text format as standard output and as waveforms – timing diagrams.

Stimulus test module

- Numerical outputs displayed by system tasks

- Built in system functions – recognized by keywords that begin with $ symbol.

Some system tasks useful for display:
$display —display a one-time value of variables or strings with an end-of-line return,
$write —same as **$display ,** but without going to next line,
$monitor —display variables whenever a value changes during a simulation run,
$time —display the simulation time,
$finish —terminate the simulation.

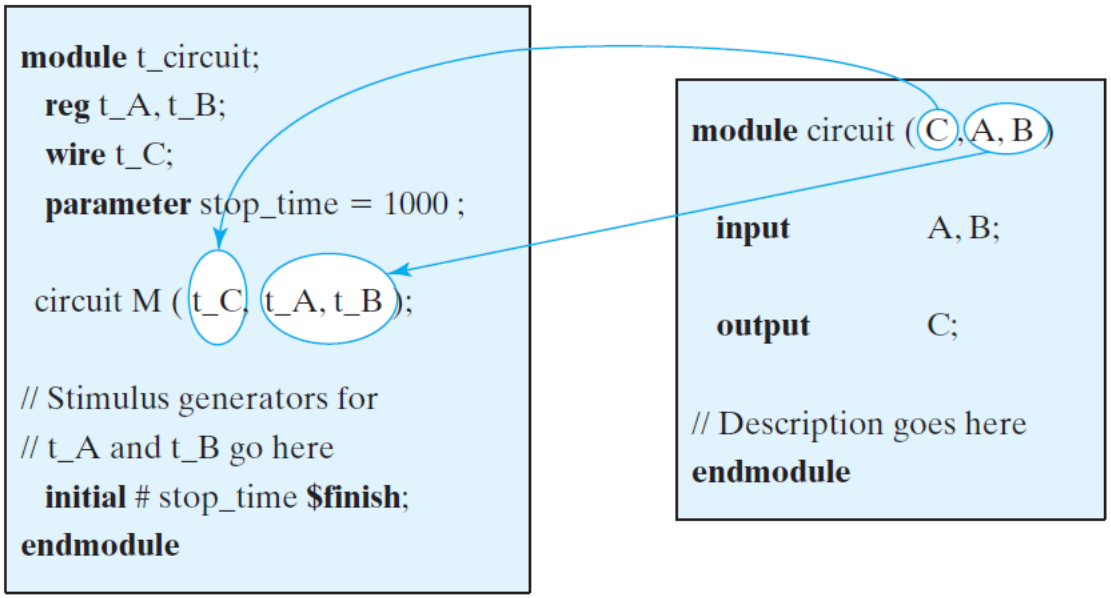
Stimulus test module

$display ("%d %b %b", C, A, B);

- Base may be binary, decimal, hexadecimal, or octal - identified with the symbols %b, %d, %h, and %o. (%B, %D, %H, and %O are valid too).

- No commas in the format specification – argument list separated by commas.

Stimulus test module



Interaction between stimulus and design modules

Stimulus test module

```
// Dataflow description of two-to-one-line multiplexer

module mux_2x1_df(m_out, A, B, select);

output m_out;
input A, B;
input select;

assign m_out  (select)? A : B;

endmodule
```

Stimulus test module

```
// Test bench with stimulus for mux_2x1_df

module t_mux_2x1_df;

wire t_mux_out;
reg t_A, t_B;
reg t_select;
parameter stop_time  = 50;

mux_2x1_df M1 (t_mux_out, t_A, t_B, t_select);
                        // Instantiation of circuit to be tested

initial # stop_time $finish;
```

Stimulus test module

```
initial begin                                   // Stimulus generator

t_select = 1; t_A = 0; t_B = 1;
#10 t_A  = 1; t_B  = 0;
#10 t_select = 0;
#10 t_A = 0; t_B = 1;

end

initial begin                                   // Response monitor
// $display (“ time Select A B m_out “);
// $monitor ( $time ,,  “%b %b %b %b” , t_select, t_A, t_B,
t_m_out);
```

Stimulus test module

$monitor (“time = “ , **$time** ,, “select = %b A = %b B = %b OUT = %b” , t_select, t_A, t_B, t_mux_out);
end
endmodule