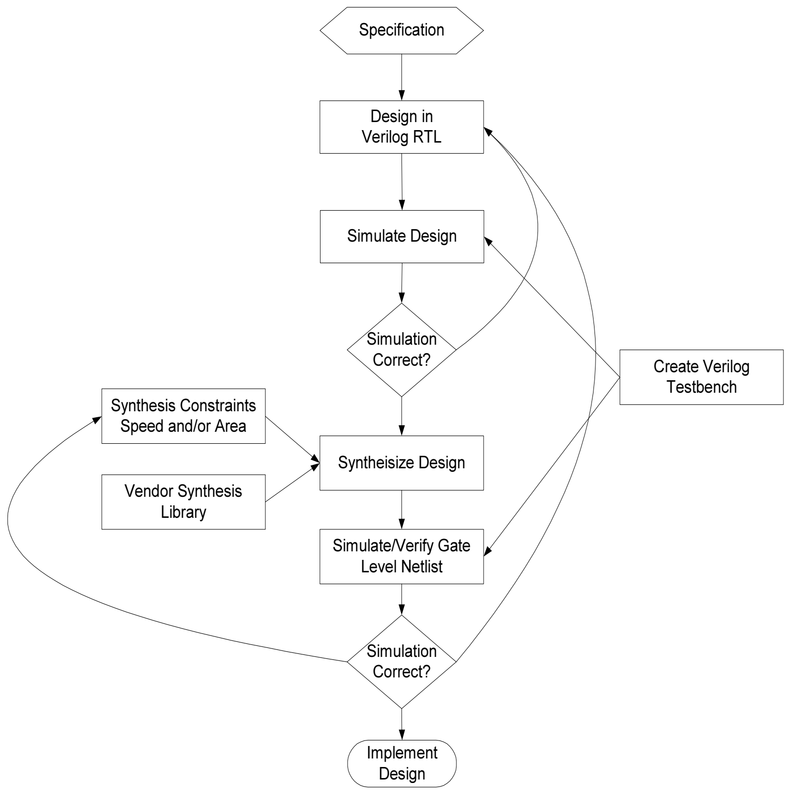
Section 15

Verilog RTL Digital Design Methodology

Design Flow



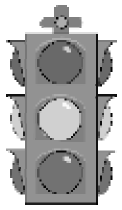
- Specification
  - Lab Exercise Manual
- Design in Verilog RTL and Simulate:
  - ModelSim, VerilogXL, etc.
- Synthesis
  - Xilinx XST, Ambit, Design Compiler, etc.
- Gate Level Verification
  - ModelSim, VerilogXL, etc.
  - Vendor Simulation Library Required.
  - Not done in class Lab exercises.

Design Specification

- Generally a design specification will be an overall circuit/system description, that may include:
  - Waveforms
  - Block diagrams
  - Logic Diagrams
  - Functional Descriptions

Specification

- Example:
  - Design a circuit to control a single traffic light.



  - The design will meet the following specifications:
    - At power-up, the North-South light will be green, the East-West light will be red. After power-up, the lights will cycle continuously under these parameters:
      - Light will change from green to yellow after 30 seconds.
      - Light will remain yellow for 5 seconds.
      - Both lights will be red for 2 seconds.
    - The design will have a 250 Hz system clock, and will be completely synchronous. Reset is active low.

Design Elements

- Control Block
  - Generally, a control block is implemented with a Finite State Machine (FSM).
- Timers
  - Create individual blocks for each counter (30 second, 5 second, and 2 second timers).

Design and Simulation

- Create the design blocks in RTL Code.
  - Test each block separately by creating Verilog testbenches.
- Bring the blocks together in a top-level module.
  - Test the top level block by creating a top-level testbench.

RTL Code - Timers

```
module thirtyseccounter
 (TIMER30, TIMER30START,
   CLK, RESET);

output TIMER30;
input TIMER30START, CLK, RESET;

integer COUNT;
reg TIMER30;

always @(posedge CLK) begin
  if (~RESET or ~TIMER30START) begin
   TIMER30 <= 0;
   COUNT <= 0;
  end
  else if (COUNT == 7499) begin
    TIMER30 <= 1;
    COUNT <= 0;
  end
  else begin
     TIMER30 <= 0;
    COUNT <= COUNT + 1;
  end
end
endmodule
```

```
module fiveseccounter
 (TIMER5, TIMER5START,
   CLK, RESET);

output TIMER5;
input TIMER5START, CLK, RESET;

integer COUNT;
reg TIMER5;

always @(posedge CLK) begin
  if (~RESET or ~TIMER5START) begin
   TIMER5 <= 0;
   COUNT <= 0;
  end
  else if (COUNT == 1199) begin
    TIMER5 <= 1;
    COUNT <= 0;
  end
  else begin
     TIMER5 <= 0;
    COUNT <= COUNT + 1;
  end
end
endmodule
```

```
module twoseccounter
 (TIMER2, TIMER2START,
   CLK, RESET);

output TIMER2;
input TIMER2START, CLK, RESET;

integer COUNT;
reg TIMER2;

always @(posedge CLK) begin
  if (~RESET or ~TIMER2START) begin
   TIMER2 <= 0;
   COUNT <= 0;
  end
  else if (COUNT == 499) begin
    TIMER2 <= 1;
    COUNT <= 0;
  end
  else begin
     TIMER2 <= 0;
    COUNT <= COUNT + 1;
  end
end
endmodule
```

RTL Code - FSM

```
module trafficlight (NSGREEN, NSYELLOW, NSRED, EWGREEN,
                     EWYELLOW, EWRED, TIMER30START,
                     TIMER5START, TIMER2START, CLK, RESET,
                     TIMER30, TIMER5, TIMER2);

  input  CLK, RESET, TIMER30, TIMER5, TIMER2;
  output NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW,
         EWRED, TIMER30START, TIMER5START, TIMER2START;

  reg NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW,
      EWRED,  TIMER30START, TIMER5START, TIMER2START;

  reg [2:0] STATE, NEXTSTATE;

  parameter NSG_EWR = 0;
  parameter NSY_EWR = 1;
  parameter NSR_EWR = 2;
  parameter EWG_NSR = 3;
  parameter EWY_NSR = 4;
  parameter EWR_NSR = 5;

  always @(posedge CLK) begin
    if (~RESET) STATE <= NSG_EWR;
    else STATE <= NEXTSTATE;
  end
```

```
always @(TIMER30, TIMER5, TIMER2, STATE) begin
   case (STATE)
     NSG_EWR : begin
       NSGREEN = 1;
       NSYELLOW = 0;
       NSRED = 0;
       EWGREEN = 0;
       EWYELLOW = 0;
       EWRED = 1;
       TIMER30START = 1;
       TIMER5START =0;
       TIMER2START = 0;
        if (TIMER30 == 1) NEXTSTATE = NSY_EWR;
        else NEXTSTATE = STATE;
     end
     NSY_EWR : begin
       NSGREEN = 0;
       NSYELLOW = 1;
       NSRED = 0;
       EWGREEN = 0;
       EWYELLOW = 0;
       EWRED = 1;
       TIMER30START = 0;
       TIMER5START = 1;
       TIMER2START = 0;
        if (TIMER5 == 1) NEXTSTATE = NSR_EWR;
        else NEXTSTATE = STATE;
      end
      NSR_EWR : begin …
```

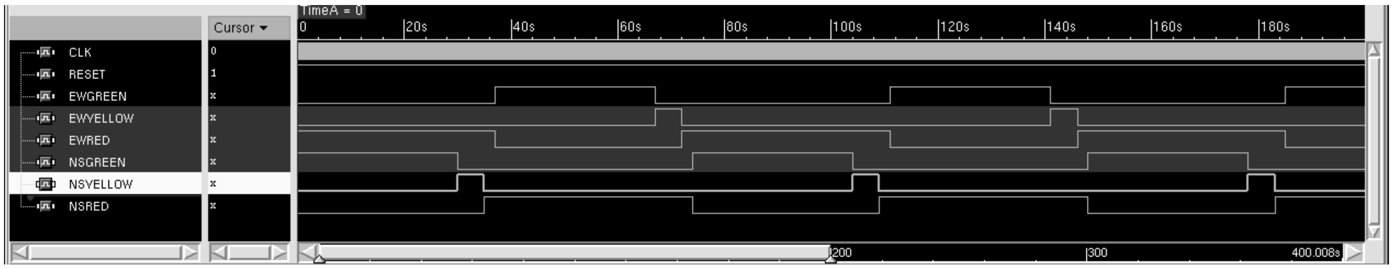
RTL Code – Top Level

```
module stoplight (NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW,
                  EWRED, CLK, RESET);

    input CLK, RESET;
    output NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW, EWRED;

    control u1 (NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW,
                EWRED,  TIMER30START, TIMER5START, TIMER2START, CLK,
                RESET, TIMER30, TIMER5, TIMER2);
    thirtyseccounter u2 (TIMER30, TIMER30START, CLK, RESET);
    fiveseccounter u3 (TIMER5, TIMER5START, CLK, RESET);
    twoseccounter u4 (TIMER2, TIMER2START, CLK, RESET);

endmodule
```

Testbench – Top Level

```
`timescale 1ms/10us
module tb;

  reg CLK, RESET;

  stoplight u1 (NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW, EWRED, CLK, RESET);

  initial begin
    $shm_open;
    $shm_probe("AC");
    CLK = 0;
    RESET = 1;
    @(negedge CLK) RESET = 0;
    @(negedge CLK) RESET = 1;
    repeat (100000) @(negedge CLK);
    $finish;
  end

  always #2 CLK = ~CLK;

endmodule // tb
```

Testbench – Results



Synthesize Design

```
set search_path .
set search_path [concat  $search_path /tools/ami/synopsys/lib/symbols]
set search_path [concat  $search_path /tools/synopsys/libraries/syn]
set search_path [concat  $search_path /tools/ami/synopsys/lib/ami350lxsc3]
set link_library [list {*} ami350lxsc3.db]
set target_library [list ami350lxsc3.db]
set symbol_library [list amilibs.sdb]
set generic_symbol_library {/tools/ami/synopsys/lib/symbols/generic.sdb}
set verilogout_show_unconnected_pins {true}
set verilog_no_tri {true}
set verilogout_no_tri {true}
set verilogout_equation {false}

read_file -f verilog stoplight.v
read_file -f verilog control.v
read_file -f verilog thirtyseccount.v
read_file -f verilog fiveseccount.v
read_file -f verilog twoseccount.v

current_design stoplight

link

compile
write -format verilog -output stoplight_netlist.v
```

```
Inferred memory devices in process in routine control line 21 in file
'/home/bolsen/Training/Verilog/ISU_Verilog_Course/Examples/Stoplight/control.v'.
===============================================================================
|   Register Name   |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|     STATE_reg     | Flip-flop |   3   |  N  | N  | N  | N  | N  | N  | N  |
===============================================================================
Presto compilation completed successfully.
Current design is now
'/home/bolsen/Training/Verilog/ISU_Verilog_Course/Examples/Stoplight/control.db:control'
control
dc_shell-t> Loading verilog file
'/home/bolsen/Training/Verilog/ISU_Verilog_Course/Examples/Stoplight/thirtyseccount.v'
Detecting input file type automatically (-rtl or -netlist).
Reading with Presto HDL Compiler (equivalent to -rtl option).
Running PRESTO HDLC
Compiling source file
/home/bolsen/Training/Verilog/ISU_Verilog_Course/Examples/Stoplight/thirtyseccount.v
Inferred memory devices in process
in routine thirtyseccounter line 9 in file
'/home/bolsen/Training/Verilog/ISU_Verilog_Course/Examples/Stoplight/thirtyseccount.v'.
===============================================================================
|   Register Name   |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|     COUNT_reg     | Flip-flop |  32   |  Y  | N  | N  | N  | N  | N  | N  |
|    TIMER30_reg    | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
===============================================================================
```

Synthesis Script

Synthesis Results Log

Synthesis Results – Verilog Gate-Level Netlist

```
module stoplight ( NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW, EWRED, CLK, RESET );

input  CLK, RESET;

output NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW, EWRED;

   wire TIMER30START, TIMER5START, TIMER2START, TIMER30, TIMER5, TIMER2,
      \u4/add_20/n1 , \u4/add_20/n2 , \u4/add_20/n3 , \u4/add_20/n4 ,
      \u4/add_20/n5 , \u4/add_20/n6 , \u4/add_20/n7 , \u4/add_20/n8 …

   inv1 \u4/add_20/U5  ( .Q(\u4/N17 ), .A(\u4/COUNT[0] ) );
   no21 \u4/add_20/U6  ( .Q(\u4/add_20/n1 ), .A(\u4/add_20/n2 ), .B(\u4/add_20/n3 ) );
   aa21 \u4/add_20/U7  ( .Q(\u4/add_20/n4 ), .A(\u4/add_20/n5 ), .B(\u4/COUNT[30] ) );
   no21 \u4/add_20/U8  ( .Q(\u4/add_20/n6 ), .A(\u4/add_20/n7 ), .B(\u4/add_20/n8 ) );
   no21 \u4/add_20/U9  ( .Q(\u4/add_20/n9 ), .A(\u4/add_20/n10 ), .B(\u4/add_20/n11 ) );
   na21 \u4/add_20/U10  ( .Q(\u4/add_20/n12 ), .A(\u4/COUNT[1] ), .B(\u4/COUNT[0] ) );
   inv1 \u4/add_20/U11  ( .Q(\u4/add_20/n13 ), .A(\u4/COUNT[2] ) );
   no21 \u4/add_20/U12  ( .Q(\u4/add_20/n14 ), .A(\u4/add_20/n13 ), .B(\u4/add_20/n12 ) );
   inv1 \u4/add_20/U13  ( .Q(\u4/add_20/n15 ), .A(\u4/COUNT[5] ) );
   na31 \u4/add_20/U14  ( .Q(\u4/add_20/n16 ), .A(\u4/COUNT[3] ), .B(\u4/add_20/n14 ), .C(\u4/COUNT[4] ) );
. . .
```

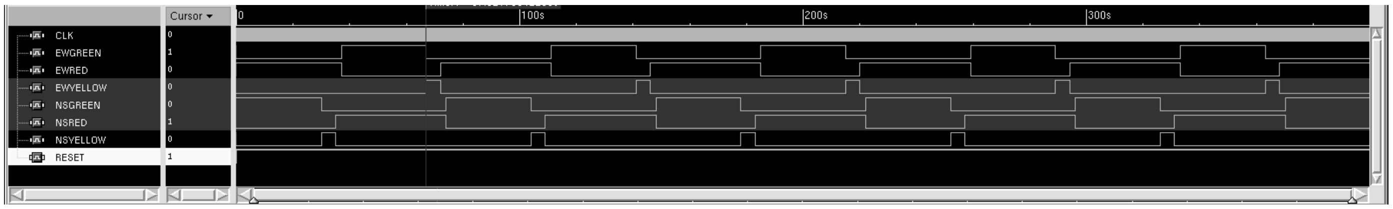
Simulate Netlist

- Use the same testbench as used to simulate RTL code.
- Simulate the Verilog gate-level netlist.
- Provide a gate-level Verilog simulation library

```
`timescale 1ms/10us
module aa21 (Q,A,B);
  output Q;
  input A,B;

  and (Q,A,B);

   parameter LVDD="VDD";
   parameter LVSS="VSS";

specify
    specparam TDR_A_Q  = 0.000;
    specparam TDF_A_Q  = 0.000;
    specparam TDR_B_Q  = 0.000;
    specparam TDF_B_Q  = 0.000;
    (A => Q)=(TDR_A_Q,TDF_A_Q);
    (B => Q)=(TDR_B_Q,TDF_B_Q);
  endspecify
endmodule
module aa22 (Q,A,B);
. . .
```

Sample portion of Verilog gate-level simulation library

Netlist Simulation



Summary

- Front end digital design methodology requires several steps before the design is implemented
  - RTL Code
  - Simulation
  - Revision (if necessary) to RTL Code.
  - Synthesis
  - Gate-Level Simulation
  - Revision (if necessary) to RTL Code, Synthesis Parameters, etc.