Alexandria University
Faculty of Engineering
Electrical Engineering Department

EE432: VLSI Modeling and Design
Sheet 4: From Algorithm to Architecture

1. Locate listings 4.11 and 4.13 (SystemVerilog). Note that each listing is accompanied in the text by a short assignment that consists in reverse engineering the code and in representing the findings graphically. If you haven't done so, catch up now.

```
// module header with external interface
module lfsr4
   ( output logic Oup_DO,
     input logic Clk_CI, Rst_RBI, Ena_SI ) ; // reset is active low

// module body with circuit structure
```

```
   // declare a variable for each inner node
   logic [1:4] State_DP;
   logic n11, n21, n31, n41, n42;

   // instantiate cells and connect them by listing port maps
   // (cells are from Synopsys' generic library in this example)
   GTECH_FD2 u10 ( .D(n11), .CP(Clk_CI), .CD(Rst_RBI), .Q(State_DP[1]) );
   GTECH_FD2 u20 ( .D(n21), .CP(Clk_CI), .CD(Rst_RBI), .Q(State_DP[2]) );
   GTECH_FD2 u30 ( .D(n31), .CP(Clk_CI), .CD(Rst_RBI), .Q(State_DP[3]) );
   GTECH_FD4 u40 ( .D(n41), .CP(Clk_CI), .SD(Rst_RBI), .Q(State_DP[4]) );
   GTECH_MUX2 u11 ( .A(State_DP[1]), .B(n42),          .S(Ena_SI), .Z(n11) );
   GTECH_MUX2 u21 ( .A(State_DP[2]), .B(State_DP[1]), .S(Ena_SI), .Z(n21) );
   GTECH_MUX2 u31 ( .A(State_DP[3]), .B(State_DP[2]), .S(Ena_SI), .Z(n31) );
   GTECH_MUX2 u41 ( .A(State_DP[4]), .B(State_DP[3]), .S(Ena_SI), .Z(n41) );
   GTECH_XOR2 u42 ( .A(State_DP[3]), .B(State_DP[4]),              .Z(n42) );

   // connect state bit of rightmost flip-flop to output port
   assign Oup_DO = State_DP[4];

endmodule
```

```
// external interface of module
module lfsr4 (
      input logic Clk_CI, Rst_RBI, Ena_SI,   // reset is active low
      output logic Oup_DO );

// behavioral model for module

   // declare internal variables
   logic [1:4] State_DP, State_DN; // for present and next state

   // computation of next state using concatenation of bits
   assign State_DN = {(State_DP[3]   State_DP[4]), State_DP[1:3]};

   // updating of state
   always_ff @(posedge Clk_CI, negedge Rst_RBI)
      // activities triggered by asynchronous reset
      if ( Rst_RBI)
         State_DP <= 4'b0001;
      // activities triggered by rising edge of clock
      else
         if (Ena_SI)
            State_DP <= State_DN; // admit next state into state register

   // updating of output
   assign Oup_DO = State_DP[4];

endmodule
```

2. Section 4.3.2 includes examples of conditional and non-conditional signal assignments. For each such example, state the conditions that cause the code to get re-evaluated during simulation.

```
always_comb
   begin
      Spring_D = 0;   // execution begins here
      if (ThisMonth_D==MARCH & ThisDay>=21) Spring_D = 1;
      if (ThisMonth_D==APRIL)               Spring_D = 1;
      if (ThisMonth_D==MAY)                 Spring_D = 1;
      if (ThisMonth_D==JUNE  & ThisDay<=20) Spring_D = 1;
   end   // process suspends here
```

```
always_ff @(posedge Clk_C, negedge Rst_RB)   // sensitivity list
   // activities triggered by asynchronous reset
   if (~Rst_RB)
      State_DP <= '0;   // shorthand for all bits zero
   // activities triggered by rising edge of clock
   else
      // when synchronous load is asserted
      if (Lod_S)
         State_DP <= '1;   // shorthand for all bits one
      // otherwise assume new value iff enable is asserted
      else if (Ena_S)
         State_DP <= State_DN;   // admit next state into state register
```

```
// external interface of module
module lfsr4 (
      input logic Clk_CI, Rst_RBI, Ena_SI,   // reset is active low
      output logic Oup_DO );

// behavioral model for module

   // declare internal variables
   logic [1:4] State_DP, State_DN; // for present and next state

   // computation of next state using concatenation of bits
   assign State_DN = {(State_DP[3]   State_DP[4]), State_DP[1:3]};

   // updating of state
   always_ff @(posedge Clk_CI, negedge Rst_RBI)
      // activities triggered by asynchronous reset
      if ( Rst_RBI)
         State_DP <= 4'b0001;
      // activities triggered by rising edge of clock
      else
         if (Ena_SI)
            State_DP <= State_DN; // admit next state into state register

   // updating of output
   assign Oup_DO = State_DP[4];

endmodule
```

```
// compute result in a series of sequential steps
module fulladd_procedural
   ( input logic Aa_DI, Bb_DI, Cc_DI,
     output logic Sum_DO, Carry_DO );

   always_comb
   begin
      logic loc1, loc3, loc4;
      loc1     =   Aa_DI ^ Bb_DI;
      Sum_DO   =   Cc_DI ^ loc1;
      loc3     = ~(Cc_DI & loc1);
      loc4     = ~(Aa_DI & Bb_DI);
      Carry_DO = ~(loc3 & loc4);
    end
endmodule
//-------------------------------------------------------------------------------
// spawn a continuous assignment for each logic operation
```

```
module fulladd_dataflow
   ( input logic Aa_DI, Bb_DI, Cc_DI,
     output logic Sum_DO, Carry_DO );

   logic Loc1_D, Loc3_D, Loc4_D;
   assign Loc1_D   =   Aa_DI  ^ Bb_DI;
   assign Sum_DO   =   Cc_DI  ^ Loc1_D;
   assign Loc3_D   = ~(Cc_DI & Loc1_D);
   assign Loc4_D   = ~(Aa_DI & Bb_DI);
   assign Carry_DO = ~(Loc3_D & Loc4_D);
endmodule
//------------------------------------------------------------------------------
// describe circuit network as a bunch of interconnected std cells
// note: cells from Synopsys' generic library are used here
module fulladd_structuralgtech
   ( input logic Aa_DI, Bb_DI, Cc_DI,
     output logic Sum_DO, Carry_DO );

   logic Loc1_D, Loc3_D, Loc4_D;
   GTECH_XOR2  u1 ( .A(Bb_DI),  .B(Aa_DI),  .Z(Loc1_D) );
   GTECH_XOR2  u2 ( .A(Cc_DI),  .B(Loc1_D), .Z(Sum_DO) );
   GTECH_NAND2 u3 ( .A(Cc_DI),  .B(Loc1_D), .Z(Loc3_D) );
   GTECH_NAND2 u4 ( .A(Aa_DI),  .B(Bb_DI),  .Z(Loc4_D) );
   GTECH_NAND2 u5 ( .A(Loc3_D), .B(Loc4_D), .Z(Carry_DO) );
endmodule
```

3. Consider process statement `memless1` from section 4.3.2 (Problem 2) and note that signal `Spring_D` is unconditionally set false before being assigned its actual value in a series of conditional statements. That value will thus evolve from true to false and back again when the process is invoked during springtime, e.g. at midnight of May 18. Do you think this trait will become visible as a brief transient during simulation? Would a circuit synthesized from this model exhibit a hazard? Explain your reasoning.

4. Listing 4.14 includes a procedural model, a dataflow model, and a structural model for a small sub-circuit in SystemVerilog.
    a. For each of the three models, find out in what way it is possible to reorder the statements without affecting the model's functionality.
    b. Although the three models describe the same functionality at the same level of abstraction, they greatly differ in the total count of signals, variables, design entities, instances, processes, and statements involved. Determine the respective numbers.
    c. Establish three schedules that list what is happening simulation cycle after simulation cycle in response of an event on any of the inputs. Think about the impact on computational efficiency when the entity gets simulated.

```
// compute result in a series of sequential steps
module fulladd_procedural
   ( input logic Aa_DI, Bb_DI, Cc_DI,
     output logic Sum_DO, Carry_DO );

   always_comb
   begin
      logic loc1, loc3, loc4;
      loc1     =   Aa_DI ^ Bb_DI;
      Sum_DO   =   Cc_DI ^ loc1;
      loc3     = ~(Cc_DI & loc1);
      loc4     = ~(Aa_DI & Bb_DI);
      Carry_DO = ~(loc3 & loc4);
    end
endmodule
//------------------------------------------------------------------------------
// spawn a continuous assignment for each logic operation

module fulladd_dataflow
   ( input logic Aa_DI, Bb_DI, Cc_DI,
     output logic Sum_DO, Carry_DO );

   logic Loc1_D, Loc3_D, Loc4_D;
   assign Loc1_D   =   Aa_DI  ^ Bb_DI;
   assign Sum_DO   =   Cc_DI  ^ Loc1_D;
   assign Loc3_D   = ~(Cc_DI & Loc1_D);
   assign Loc4_D   = ~(Aa_DI & Bb_DI);
   assign Carry_DO = ~(Loc3_D & Loc4_D);
endmodule
//------------------------------------------------------------------------------
// describe circuit network as a bunch of interconnected std cells
// note: cells from Synopsys' generic library are used here
module fulladd_structuralgtech
   ( input logic Aa_DI, Bb_DI, Cc_DI,
     output logic Sum_DO, Carry_DO );

   logic Loc1_D, Loc3_D, Loc4_D;
   GTECH_XOR2  u1 ( .A(Bb_DI),  .B(Aa_DI),  .Z(Loc1_D) );
   GTECH_XOR2  u2 ( .A(Cc_DI),  .B(Loc1_D), .Z(Sum_DO) );
   GTECH_NAND2 u3 ( .A(Cc_DI),  .B(Loc1_D), .Z(Loc3_D) );
   GTECH_NAND2 u4 ( .A(Aa_DI),  .B(Bb_DI),  .Z(Loc4_D) );
   GTECH_NAND2 u5 ( .A(Loc3_D), .B(Loc4_D), .Z(Carry_DO) );
endmodule
```

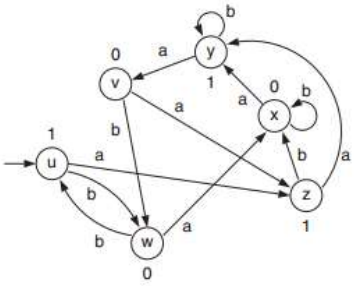
5. Explain the differences between these conditional constructs: `if...` and `generate if...endgenerate` (SystemVerilog). Are there any other SystemVerilog statements that are related to each other in the same way?

6. Write a SystemVerilog model for a binary-coded-decimal (BCD) counter that is amenable to simulation and synthesis. A control input of two bits is to decide between count-up (01), count-down (10), and hold (00) condition. Input-to-output latency shall not exceed one clock cycle. Under no circumstance are hazards tolerated on any of the output signals. Also, do not forget to address parasitic inputs and parasitic states.

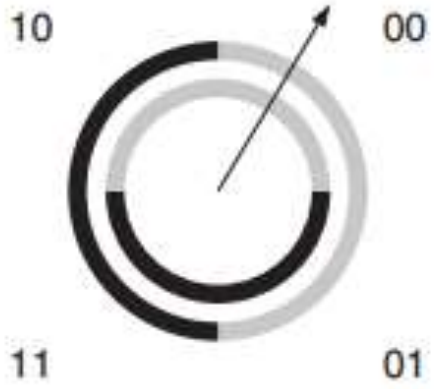
7. SystemVerilog knows of no built-in construct for computing the absolute value of integers. Write a function for that.

8. Write a SystemVerilog synthesis model for the finite state machine of Fig.B.4. A synchronous clear shall be provided for initializing the circuit. Explain how your code reflects the fact that this is a Moore automaton.

$i(k)$		a	b
$s(k)$	$o(k)$	$s(k+1)$	
u	1	z	w
v	0	z	w
w	0	x	u
x	0	y	x
y	1	v	y
z	1	y	x



9. Consider a shaft equipped with an angle encoder that indicates the shaft's current position using a two-bit unit-distance code, see Fig.4.25. Design a state machine that accepts this code and tells whether the shaft is currently rotating clockwise or counterclockwise. The former sense of rotation shall be indicated when the shaft is at standstill. Establish a synthesis model. You may want to extend the functionality such as to indicate the angular position of the shaft and the number of turns it has made since time zero. Write the SystemVerilog model for your design.



10. Design a synchronous first-in first-out(FIFO) queue with the features below.
    a. Separate read and write ports ("data" plus "read" or "write" respectively).
    b. A pair of outputs that flag "full" and "empty" conditions respectively.
    c. Two more outputs that indicate "almost full" and "almost empty" conditions.
    d. Parametrized queue depth, data width, and "almost full/empty" thresholds.
    e. Show-ahead capability, aka first-word-fall-through property. Use a register file for data storage and code your RTL synthesis model in either VHDL or SystemVerilog. Chose meaningful port and signal names and observe the naming conventions of section 6.7.