FreeRange Verilog Foundation Modeling Solutions Manual

V1.00

Chapter 1

1) List the two most commonly used HDLs.

ANS: Verilog and VHDL

2) List and briefly describe the two main purposes of HDLs.

ANS: HDLs are used primarily for modeling and verification of digital circuits.

3) List the three levels of computer programming.

ANS: Lowest level: machine code. Next lowest level: assembly code. Highest level: higher-level programming.

4) What is the final output of all programming languages?

ANS: Machine code, or a bunch of 1's and 0's, is the final output of all programming code.

5) In your own words, describe the purpose of the HDL synthesizer.

ANS: The synthesizer translates the text-based modeling code to another form, which is either to something that can be made into a circuit of something that can test a circuit.

6) In your own words, describe the meaning and purpose of "knobs" in HDL modeling.

ANS: Knobs are controls that the synthesizer has over the synthesis process. The HDL modeling can and should limit the number of knobs available to the synthesizer.

7) In your own words, describe the two approaches this text uses to getting you to write solid HDL models.

ANS: 1) you need to understand the basics of HDLs, so we introduce this theory. 2) You need to keep your circuits are simple as possible to help you get them up and running; once you gain more HDL modeling skills, you can expand your horizons.

8) List and briefly describe the two main tenets of digital design.

ANS: Modern digital design is modular and hierarchical.

9) Briefly describe what we mean by the term *flat design.*

ANS: A flat design is a design that is not hierarchical.

10) Briefly list the main difference between Verilog and SystemVerilog.

ANS: The main difference between Verilog and SystemVerilog is that SystemVerilog contains many more non-synthesizeable constructs that are used primarily for Verification.

11) Briefly list why this text primarily introduces Verilog and not SystemVerilog.

ANS: This text introduces Verilog because it is more basic and SystemVerilog and because most of the new features in SystemVerilog are associated with circuit verification while the associated course focuses on circuit design and less so on verification.

12) List and briefly describe the four Golden Rules of using HDL to model circuits.

ANS: **Golden Rule #1:** Keep models simple by leveraging modular and hierarchical design.

Golden Rule #2: Don't rely on the synthesizer to make your circuit work for you.

Golden Rule #3: Design knowing that you'll need to verify the design.

Golden Rule #4: Neatness counts in HDL models.

13) Briefly describe the purpose of a style file.

ANS: A style file gives humans a guide for writing good code by indicating how good code should look. Some items in a style file are proper indentation, a solid file banner, etc.

Chapter 2

1) Briefly describe whether it is possible to plagiarize yourself. If this is possible, what entity would be ethically and morally responsible for making such a determination?

ANS: It's my work, so I can reuse it all I want. The label of plagiarism is stupid… maybe I should be copyrighting ever though that goes through my brain.

2) Briefly describe what is meant by the term "digital design".

ANS: Digital design is the act of creating a digital circuit to solve some given problem.

3) List and briefly describe the two main tenets of digital design.

ANS: Modern digital designs are "efficient"; they are both hierarchical and modular.

4) List and briefly describe the two main types of digital circuits.

ANS: The two main type of digital circuits are combinatorial circuits (contain no memory) and sequential circuits (contain memory). Another way to look at this is that the outputs of a combinatorial circuit are a function of the circuits inputs, while the outputs of a sequential circuit are a function of the sequence of the circuit's inputs.

5) List and briefly describe the four major classifications of inputs and outputs of a digital circuit.

ANS: The four main classes of inputs and outputs are 1) data inputs, 2) data outputs, 3) control inputs, and 4) status outputs.

6) Briefly describe the main purpose of an FSM in digital design.

ANS: The finite state machine acts as a controller in digital circuits, that is, the FSM is a digital circuit that controls other parts of the circuit.

7) List and briefly describe the three main components of an FSM.

ANS: The three main components of an FSM are 1) the Next State decoder, which decodes inputs and current state information to determine the next state, 2) the output decoder, which determine the circuits outputs based on state (Moore machines) or state and external inputs (Mealy machine), and 3) the state registers, which store the current state the FSM is in.

8) List and briefly describe the two main classifications of FSM outputs.

ANS: The two classifications of outputs are 1) Mealy outputs, which are a function of state and external FSM inputs, and 2) Moore outputs, which are a function of state only.

9) List and briefly describe the three main types of approaches to digital design.

ANS: The three main approach to digital design are 1) Brute Force Design (BFD), which defines an output for every possible input combination, 2) Iterative Modular Design (IMD), which iteratively uses smaller identical circuits to form a larger circuit, and 3) Modular Design (MD), which primarily relies on using standard and/or previously designed modules as a basis for a larger circuit.

10) Briefly explain the concept of structured design as it relates to digital design.

ANS: The concept of structured design means that you can decompose any digital circuit in a set of standard digital modules (referred to as Foundation Modules). Stated differently, the digital circuit solution to any circuit problem can be formed by a set of standard digital modules.

11) List and briefly describe the four approaches to controlling a digital circuit.

ANS: The four approaches to controlling a circuit are 1) no control, 2) internal control, where a status signal from an internal circuit controls the circuit, 3) external control, where a signal external to the circuit (from the outside world) controls the circuit, and 4) when another circuit control the circuit, such as an FSM.

12) Digital design foundation modeling divides circuits into two categories; list and briefly describe those categories.

ANS: The two types of circuits are controlled circuits and controller circuits. Controlled circuits expected to be controlled by some other entity; controller circuits act to directly control other digital circuits.

13) Control signals are typically output from one circuit and input to another circuit. List what types of circuits they are output from and what type of circuits that are input to.

ANS: Control signals are typically the control outputs from FSMs. They can also be status inputs from other modules in the circuit, which makes them internal controls.

14) Status signals are typically output from one circuit and input to another circuit. List what types of circuits they are output from and what type of circuits that are input to.

ANS: Status signals are typically output from a circuit and used as inputs to FSMs. Status signals can also be connected directly to control inputs of other circuits.

Chapter 3

1) Briefly describe the original purpose of the Verilog HDL.

ANS: Verilog was originally a language designed for verification.

2) What is the main theme behind using an HDL to model digital circuits?

ANS: The main theme of HDLs is concurrency.

3) In your own words, briefly describe the notion of concurrency in a digital circuit.

ANS: The notion of concurrency in digital circuits is associated with how we must write the models as text. Digital circuits are inherently operate concurrently while a page of text reads sequentially. The result is to interpret certain aspects of the text file (the HDL code) to operate concurrently even though there is no way to "write them" as such in a text file.

4) Briefly describe the difference, if any, between the words concurrent and parallel in the context of a digital circuit.

ANS: In the context of digital circuits, concurrent and parallel mean the same thing. Elements of digital circuits are said to act concurrently, or in parallel.

5) Briefly describe whether computer programs operate in a parallel manner or not.

ANS: Computer programs typically act sequentially, not in parallel. This means that the computer hardware only processes one instruction at a time, then goes on to the next instruction.

6) Briefly and in your own words, describe what we mean in by the term "model" in engineering.

ANS: A model is a description of something.

7) Briefly explain whether the terms "Verilog code" and "Verilog models" are synonymous.

ANS: They are almost synonymous, but, Verilog code can also contain items such as comments that you could argue are not part of the "description", and thus model of the circuit.

8) Briefly explain why using dataflow models when you can give the synthesizer fewer options.

ANS: Dataflow models represent low-level descriptions of circuits, which means your model describes exactly what you want in the circuit rather than describing what the circuit is supposed to do. Because of this "more exact" description, the synthesizer has fewer options as how to "modify" the circuit during the synthesis process.

9) Briefly explain why using behavioral models is a more powerful modeling approach compared to using dataflow models.

ANS: Behavioral models are more powerful because they allow you to describe how circuits should operate rather than describing the exact logic required to make that circuit operate as is done in dataflow models. This is an extremely important concept as circuit become more complex.

10) Briefly explain whether you can use numerical values in identifiers.

ANS: You can use numerical values in identifiers, but the first character in your label can't be a numeric value.

11) List and briefly describe the two types of comments we use in Verilog.

ANS: There are single line comments: everything after the "//" on a line is a comment. There are also block comments, where everything between "/*" and "*/" are comments. Block comments can be multiline comments.

12) Briefly describe why the Verilog synthesizer does not care if your code is not readable by humans.

ANS: Verilog only cares if the file has followed proper Verilog syntax rules; this it does not care if humans can read the code or not.

13) Briefly describe why you should never use the tab key when writing Verilog code in an editor.

ANS: The tab key is interpreted differently by different printers and different text editors. It is thus best to not use the tab key unless you're the only person who will ever examine your code.

14) Briefly describe a major way it is possible to get by without knowing the operator precedence rules in Verilog.

ANS: The best was to get by without knowing precedence rules is to liberally use parenthesis in an intelligent manner.

15) Briefly describe the two types of comments used in Verilog and the Verilog syntax that supports them.

ANS: There are single line comments: everything after the "//" on a line is a comment. There are also block comments, where everything between "/*" and "*/" are comments. Block comments can be multiline comments.

16) Comments are information passed between two entities: which two entities are these?

ANS: Comments in Verilog are information passed between human readers of the code; the synthesizer ignores all comments.

17) Briefly describe how statements are terminated in Verilog.

ANS: Statements in Verilog are terminated with a semi-colon.

Chapter 4

1) Briefly describe the two levels you can use in Verilog to describe a digital circuit.

ANS: You can model circuits in Verilog at the Dataflow or Behavioral level.

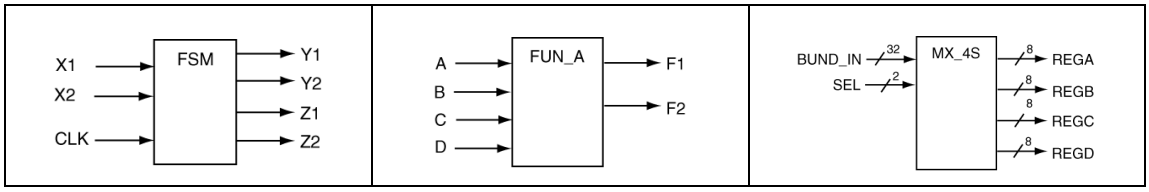
2) Briefly describe whether the so-called "dataflow model" is actually behavioral modeling or not.

ANS: The statements we've been using in dataflow models are actually considered a type of behavioral modeling. It's simply more clear to definitively separate the two type of modeling if even it is not exactly true.

3) Name and briefly describe the three parts of a Verilog model.

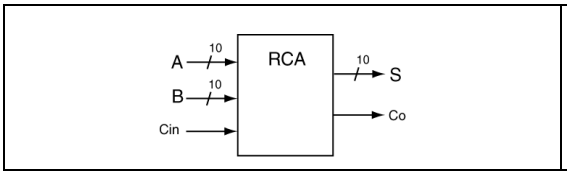
ANS: The three parts are 1) the External Interface, which is what the outside world sees, 2) the Internal Circuitry, which are the individual modules defined in or included with a given model, and, 3) the Internal Interface, which is the internal signals that connect the modules with the model.

4) Write the external interface portion of a Verilog model that describe the following block diagrams:

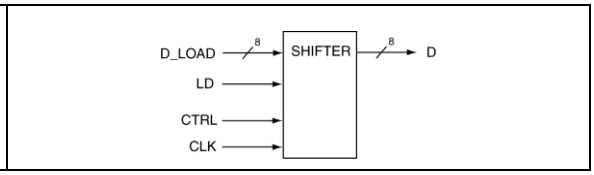


(a) **(b)** **(c)**

```
module a(
    input    X1,
    input    X2,
    input    CLK,
    output   Y1
    output   Y2
    output   Z1
    output   Z2);

module b(
    input    A,
    input    B,
    input    C,
    input    D,
    output   F1
    output   F2);

module c(
    input  [31:0]  BUND_IN,
    input   [1:0]  SEL,
    output  [7:0]  REGA,
    output  [7:0]  REGB,
    output  [7:0]  REGC,
    output  [7:0]  REGD     );
```



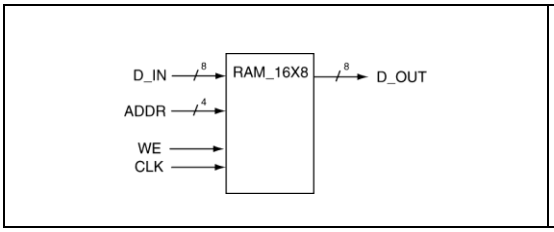
(d)



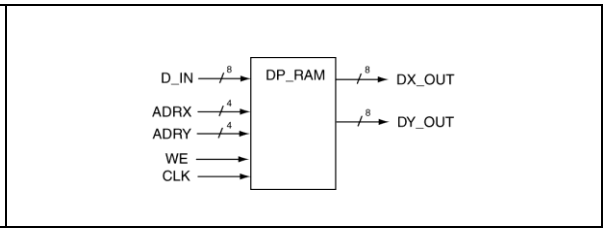
(e)

```
module d(
    input  [9:0] A,
    input  [9:0] B,
    input        Cin,
    output [9:0] S,
    output       Co     );
```

```
module e(
    input   [7:0]  D_LOAD,
    input          LD,
    input          CTRL,
    input          CLK,
    output  [7:0]  D     );
```



(f)



(g)

```
module f(
    input  [7:0] D_IN,
    input  [3:0] ADDR,
    input        WE,
    input        CLK,
    output [7:0] D_OUT    );
```

```
module g(
    input  [7:0] D_IN,
    input  [3:0] ADRX,
    input  [3:0] ADRY,
    input        WE,
    input        CLK,
    output [7:0] DX_OUT,
    output [7:0] DY_OUT    );
```

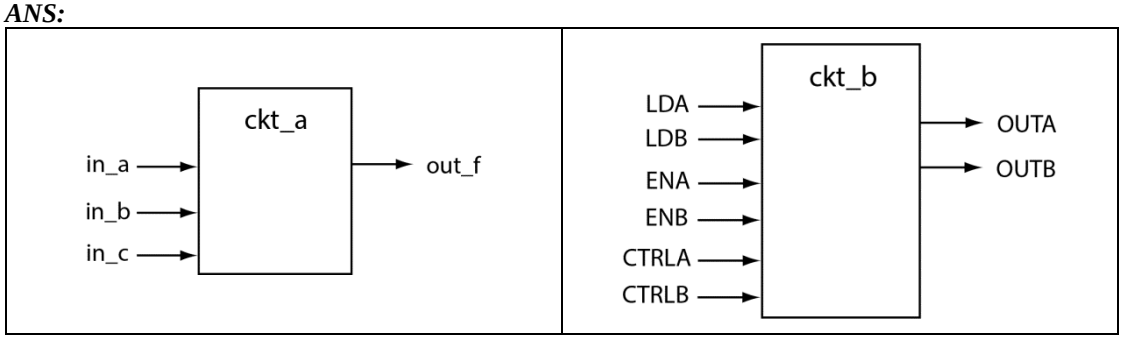
5) Provide black box diagrams that are defined by the following Verilog external interface specifications.

```
module ckt_a  (
     input   in_a,
     input   in_b,
     input   in_c,
     output  out_f );
```

```
module ckt_b  (
   input   LDA,
   input   LDB,
   input   ENA,
   input   ENB,
   input   CTRLA,
   input   CTRLB,
   output  OUTA,
   output  OUTB  );
```

(a) **(b)**

ANS:



(a) **(b)**

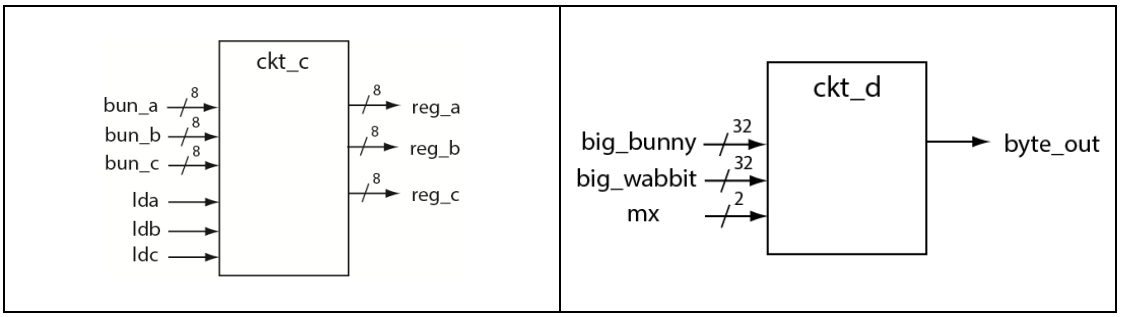
```
module ckt_c  (
   input   [7:0] bun_a,
   input   [7:0] bun_b,
   input   [7:0] bun_c,
   input   lda,
   input   ldb,
   input   ldc,
   output  [7:0] reg_a,
   output  [7:0] reg_b,
   output  [7:0] reg_c);
```

(c)

```
module ckt_d  (
   input   [31:0] big_bunny,
   input   [31:0] big_wabbit,
   input   [1:0]  mx,
   output   byte_out  );
```

(d)

ANS:



(c) **(d)**

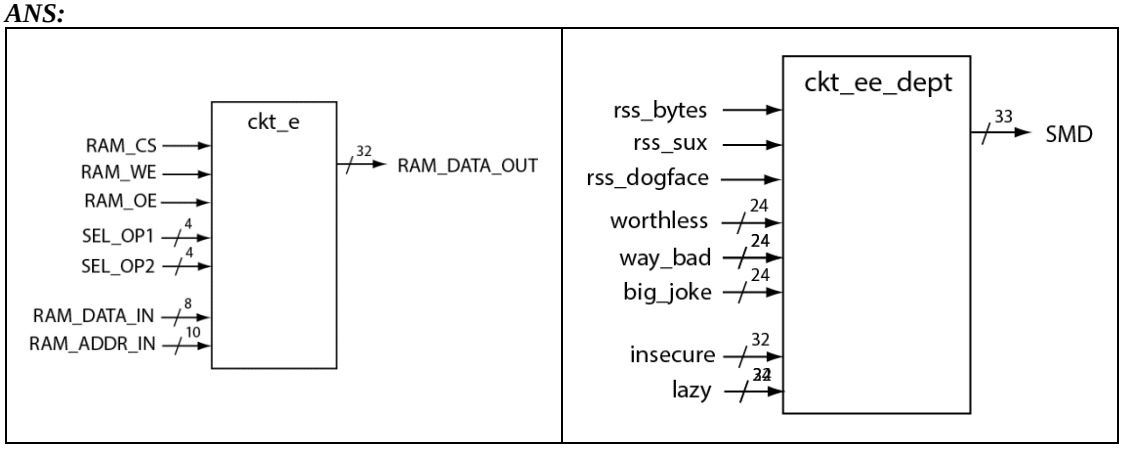
```
module ckt_e  (
   input   RAM_CS,
   input   RAM_WE,
   input   RAM_OE,
   input   [3:0]  SEL_OP1,
   input   [3:0] SEL_OP2,
   input   [7:0] RAM_DATA_IN,
   input   [9:0] RAM_ADDR_IN,
   output  [7:0] RAM_DATA_OUT  );
```

(e)

```
module ckt_ee_dept  (
   input  rss_bytes,
   input  rss_sux,
   input  rss_dogface,
   input  [23:0]  worthless,
   input  [23:0]  way_bad,
   input  [23:0]  go_away,
   input  [31:0]  big_joke,
   input  [31:0]  insecure,
   input  [31:0]  lazy,
   output [32:0]  SMD );
```

(f)

ANS:



(d) **(e)**

6) Provide Verilog models that implement the following Boolean expressions.

(a) $F(A,B,C) = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C}$

ANS:

```
module ckt_a  (
     input   A,
     input   B,
     input   C,
     output  F);

     assign F = (~A & ~B & ~C) | (~A & B & C) | (A & ~B & C) | (A & B & ~C);

endmodule;
```

(b) $F(A,B,C) = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C}$

ANS:

```
module ckt_b  (
     input   A,
     input   B,
     input   C,
     output  F);

     assign F = (~A & ~B & ~C) | (~A & B & C) | (A & ~B & C) | (A & B & ~C);

endmodule;
```

(c) $F(A,B,C) = (A+B+C)\cdot(A+\overline{B}+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+\overline{B}+C)$

ANS:

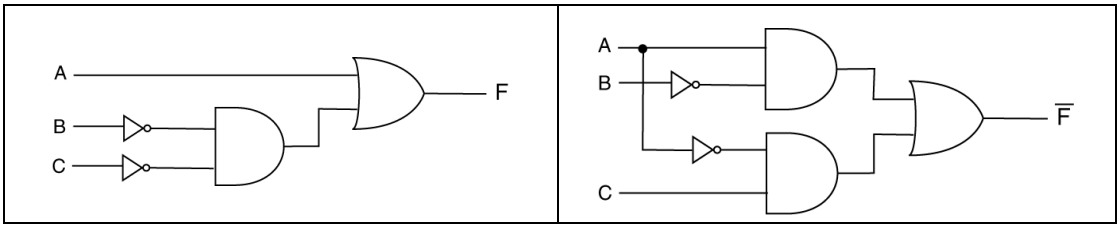
```
module ckt_c  (
     input   A,
     input   B,
     input   C,
     output  F);

     assign F = (A & B & C) | (A & ~B & C) | (A & ~B & ~C) | (~A & ~B & C);

endmodule
```

(d) $F(X,Y,Z) = (\overline{X}+\overline{Y}+Z)\cdot(\overline{X}+Y+\overline{Z})\cdot(X+\overline{Y}+Z)\cdot(\overline{X}+\overline{Y}+\overline{Z})$

ANS:

```
module ckt_d  (
     input   X,
     input   Y,
     input   Z,
     output  F);

     assign F = (~X & ~Y & Z) | (~X & Y & ~Z) | (X & ~Y & Z) | (~X & ~Y & ~Z);

endmodule
```

7) Provide Verilog models that implement the following circuit models



(a) **(b)**

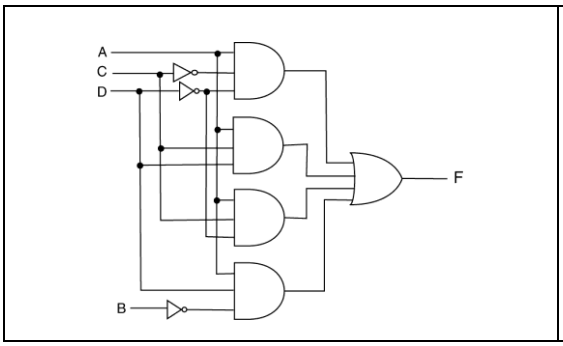
ANS:

```
module ckt_a  (
     input   A,
     input   B,
     input   C,
     output  F);

     assign F = (A) | (~B & ~C);

endmodule
```

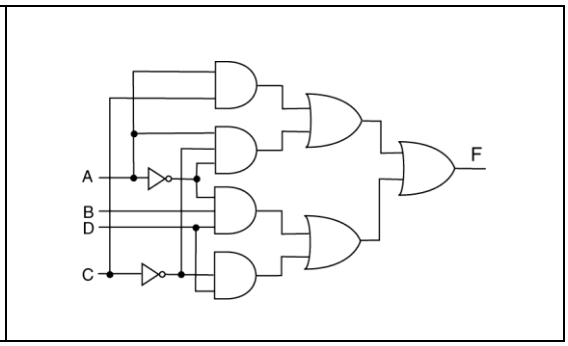
(a)

```
module ckt_b  (
     input   A,
     input   B,
     input   C,
     output  F);

     assign F = (A & ~B) | (~A & C);

endmodule
```

(b)



(c)



(d)

ANS:

```
module ckt_c  (
     input   A,
     input   B,
     input   C,
     input   D,
     output  F);

     assign F = (A & ~C & ~D) |
                (A & ~C & D)  |
                (A & C & ~D)  |
                (A & ~B & D);


endmodule
```

(c)

```
module ckt_d  (
     input   A,
     input   B,
     input   C,
     output  F);

     assign F = (A & C) |
                (A & ~A & ~C)  |
                (~A & B & D)  |
                (B & ~C & D);


endmodule
```

(d)

8) Provide a Verilog model that implements a half adder (HA).

ANS:

```
module ckt_ha  (
     input   a,
     input   b,
     output  sum,
     output  co);

     assign sum = (a ^ b);

     assign co  = (a & b);


endmodule;
```

9) Provide a Verilog model that implements a full adder (FA).

ANS:

```
module ckt_ha  (
     input   a,
     input   b,
     input   cin,
     output  sum,
     output  co);

     assign sum = (a & ~b & ~cin) | (~a & b & ~cin) | (a & ~b & ~cin);

     assign co  = (a ^ b ^ cin);


endmodule;
```

10) Write an equivalent equation for F in following Verilog model and draw the equivalent circuit.

```
module ex1_model  (
    input  A,
    input  B,
    input  C,
    output F   );


   assign F = ( A | ~B | C )    &
              ( A | ~B | ~C )   &
              ( ~A | B | ~C )   &
              ( ~A | ~B | ~C );

endmodule
```

ANS:

```
module ex1_model  (
    input  A,
    input  B,
    input  C,
    output F   );


   assign F = (~A & ~B & ~C )   |
              (~A & ~B &  C )   |
              ( A & ~B & ~C )   |
              ( A &  B & ~C );

endmodule
```

Chapter 5

1) Name the three types of models associated with the Verilog HDL.

ANS: 1) dataflow models, 2) behavioral models, and 3) structural models.

2) Can a Verilog model be both a structural model and a dataflow model? Briefly explain.

ANS: The definitions of models are convenient, but we don't think much about them after we gain experience modeling digital circuits with Verilog. As you gain skills, you just model circuits in the most efficient way you can, and don't think much about what type of modeling you're doing. So, in the strict sense of the definitions, if you instantiate modules in your design, then it is a structural model despite any other type of modeling your design contains. The same thing can be said for behavioral designs as they relate to strictly dataflow models.

3) Explain the concept that structural model is about using previously design boxes in your design rather than designing new boxes.

ANS: The notion of instantiating a module means that the module has been defined elsewhere and you don't need to provide a new and/or different definition of the module. Reusing previously designed modules provides an approach to making your designs "efficient".

4) Briefly describe what we mean by the term *flat design*.

ANS: A flat design is on that contains no module instantiations, thus structural models are not flat designs by definition.

5) Briefly explain why structural modeling exists. Be sure to mention the notion of flat designs in your explanation.

ANS: Structural modeling exists in order to make the modeling process efficient for designers. It is possible to design with purely flat designs, but such designs are hard for humans to understand.

6) Briefly explain whether structural modeled designs and flat designs are functionally equivalent.

ANS: Structural models and flat designs are functionally equivalent. Structural models are essentially a hack to support human understanding of the models.

7) Briefly describe what differentiates separate instantiations of the same module at any given level of a design.

ANS: The label associated with an instantiation separates the different instantiations of the same module.

8) Briefly explain how it is that all structural designs eventually become flat designs anyways.

ANS: Structural models are a way to increase human understanding of the circuit; the synthesizer at some point flattens all structural models as part of the synthesis process.

Chapter 6

1) Briefly describe how behavioral modeling differs from dataflow modeling.

ANS: Dataflow modeling is low-level, and generally describes circuits using gate-level logic. Behavioral model is higher level and models circuits by describing their intended behavior rather than the lower-level logic that implements that behavior.

2) List the two types of procedural blocks that Verilog uses.

ANS: The two types of procedural blocks in Verilog are initial blocks and always blocks.

3) List the three ways you can model a "box" using Verilog.

ANS: You can model boxes as 1) separate modules, as 2) instantiations of existing modules, and 3) in the body of code using constructs such as always blocks.

4) Briefly describe the relation, if any, between sequential circuits and sequential statements within an **`always`** block.

ANS: There is no relation between sequential circuits the sequential evaluation of statements within an always block.

5) List the two main types of statements we place in an **`always`** block.

ANS: We place basic assignment statements and procedural programming statements within an always block.

6) List the two types of procedural assignment statements used in Verilog.

ANS: Verilog's two types of procedural assignment statements are if-else and case statements.

7) Briefly describe the difference between blocking and non-blocking assignment statements.

ANS: The results from blocking assignment statements can be used later in the procedural block if they need to be; the results from non-blocking assignment statements are "scheduled" to be made, and the actual results are not made until the procedural block terminates, which means the results cannot be used in calculations within the current procedural block evaluation.

8) Briefly describe whether you can use procedural statements outside of procedural blocks.

ANS: Procedural programming statements must appear with a procedural block (always blocks and initial blocks).

9) Briefly describe one way that Verilog uses the notion of "completely specified circuits" and "incompletely specified circuits" to generate combinatorial or sequential circuits.

ANS: If the Verilog model specifies an output for every possible input combination, then the circuit is completely specified and the model synthesizes to a combinatorial circuit. If the Verilog model does not specify an output for every possible input combination but uses a catchall, the circuit is incompletely specified but synthesizes to a

combinatorial circuit because the of the catchall. If the circuit does not specify an output for every possible input combination and does not use a catchall statement, the model synthesizes to a sequential circuit.

10) Briefly describe a way you can ensure your procedural block models a combinatorial circuit.

ANS: Your procedural block will always generate a combinatorial circuit if the block specifies an output for every possible combination of inputs.

11) Briefly describe a way you can ensure your procedural block models a sequential circuit.

ANS: Your procedural block will always generate a sequential circuit if the block does not specify an output for every possible combination of inputs, or if the block contains a posedge or negedge in the block's sensitivity list.

12) List the three new procedural blocks available in System Verilog.

ANS: There are always_ff, always_latch, and always_comb

13) List a few reasons why using only **`always`** blocks (and not the new System Verilog procedural blocks) is a good idea.

ANS: Using only always blocks requires that modelers understand Verilog to the point of being able to model sequential or combinatorial blocks as they need to.

Chapter 7

1) Using Verilog vernacular, what is the type for **`wire`**?

ANS: The wire is a net type.

2) Using Verilog vernacular, what is the type for a **`reg`**?

ANS: The reg is a variable type.

3) Briefly explain why **wires** are only associated with combinatorial circuits.

ANS: wires are only associated with combinatorial circuits because they cannot retain values (and are thus not "memory" and therefore must be only combinatorial).

4) Briefly describe what determines whether a **`reg`** induces memory or not.

ANS: A variable declared as a reg can be memory; it all depends how it used in the given model.

5) Briefly describe why is it good practice to always use begin and end clauses when you use **`always`** blocks.

ANS: This is good practice because it provides information and consistency to your models. This is not a requirement though, particularly for always blocks that contain a minimum amount of code in the body of the statement.

6) Briefly describe what we typically mean by the term "incompletely specified" output for a circuit.

ANS: An incompletely specified circuit is one when not all input combinations are provided with a specific output value.

7) List the two ways you can use an **`if-else`** clause to model a combinatorial circuit.

ANS: The two ways you can use an if-else clause to model a combinatorial circuit are to 1) use an else clause as a catchall, or 2) use the if clauses to provide an output for every possible input combination.

8) List the two ways you can use a **`case`** statement to model a combinatorial circuit.

ANS: The two ways you can use a case statement to model a combinatorial circuit are to 1) use a catchall default clause, or 2) specify an output for ever possible input combination.

9) Briefly describe the general rule of using a **`case`** statement vs. an **`if-else`** statement.

ANS: You should strive to use a case statement if there are going to be more than two or three else if clauses in your if statement.

10) Briefly describe the good general rule as to use **`begin-end`** pairs where required in Verilog modeling.

ANS: You should use a begin-end pair when it is required syntax or when it makes your code more understandable for the human reader.

11) Provide a Verilog model for an 3:8 standard decoder with one-cold outputs.

ANS:

```
module stand_dcdr_3t8_1cold (
    input wire [2:0] SEL,
    output reg [7:0] D_OUT  );


    //- standard decoder for display multiplex
    always @ (*)
    begin
       case (SEL)
          0: D_OUT = 4'b1111_1110;
          1: D_OUT = 4'b1111_1101;
          2: D_OUT = 4'b1111_1011;
          3: D_OUT = 4'b1111_0111;
          4: D_OUT = 4'b1110_1111;
          5: D_OUT = 4'b1101_1111;
          6: D_OUT = 4'b1011_1111;
          7: D_OUT = 4'b0111_1111;
          //default D_OUT = 4'b0000_0000;
       endcase
    end

endmodule
```

Chapter 8

1) Briefly describe the connection between completely specified outputs and combinatorial circuit in procedural blocks.

ANS: For a procedural block to cause the synthesizer to generate a combinatorial circuit, the block must provide an output for all possible input combinations, which is what the term "completely specified outputs" is referring to.

2) Briefly describe what the term "catch-all" means for procedural programming statements.

ANS: The term catchall means that there is a "short-cut" for ensuring that all possible input combinations have specified outputs, which in turn guarantees the code will generate a combinatorial circuit.

3) Briefly describe whether you need to include a catchall statement in your procedural programming statements if your model specifies an output for every possible input combination.

ANS: If your model explicitly specifies an output for all possible input combinations, then you don't need a catchall to ensure the synthesizer will generate a combinatorial circuit. In this case, adding a catchall statement will not provide anything other than a message to humans that the circuit is in fact combinatorial. Additionally, adding a catchall in this case may cause the synthesizer to generate a warning.

4) Briefly describe why generic model approaches support the basic tenets of digital design.

ANS: Generic modules are easier to modify to create the model you require. The less desirable solution would be to generate a new module each time a circuit parameter (such as signal widths) change.

5) Briefly describe why it is important to model digital modules such as MUXes with generic modeling techniques in Verilog.

ANS: Modeling standard circuits in a generic manner allows you to easily reuse the modules for similar circuits that have different bit-widths.

6) Briefly describe why it is the best idea to not rely on catch-all statements for valid input combinations.

ANS: Relying on catch-alls to handle valid cases is confusing to human readers and thus makes modifications to the code potentially more problematic.

7) List the two *parameter* types used in Verilog.

ANS: There are module parameters and specify parameters.

8) Briefly describe the functionality using a "*" in the sensitivity list for a procedural block provides.

ANS: Using an (*) it the sensitivity list makes the block automatically sensitive to all the operands appearing on the right side of assignment operators used in the procedural block.

9) Provide a Verilog model for a 2:1 MUX that uses 1-bit data.

ANS:

```
module mux_2t1  (
       input wire SEL,
       input wire  D0,
       input wire  D1,
       output reg  D_OUT ) ;

       always @ (*)
       begin
          if      (SEL == 1'b0)  D_OUT = D0;
          else if (SEL == 1'b1)  D_OUT = D1;
          else                   D_OUT = 0;
       end

endmodule
```

10) Convert the model from the previous problem to use a parameter for the data width.

ANS:

```
module mux_2t1_nb  #(parameter n=8) (
       input wire SEL,
       input wire [n-1:0] D0,
       input wire [n-1:0] D1,
       output reg [n-1:0] D_OUT ) ;


       always @ (*)
       begin
          if      (SEL == 1'b0)  D_OUT = D0;
          else if (SEL == 1'b1)  D_OUT = D1;
          else                   D_OUT = 0;
       end

endmodule
```

11) Provide a Verilog model for an 8:1 MUX that uses 1-bit data.

ANS: The

ANS:

```
module mux_8t1   (
       input wire [2:0] SEL,
       input wire       D0,
       input wire       D1,
       input wire       D2,
       input wire       D3,
       input wire       D4,
       input wire       D5,
       input wire       D6,
       input wire       D7,
       output reg       D_OUT  );


       always @(*)
       begin
          case (SEL)
             0:  D_OUT = D0;
             1:  D_OUT = D1;
             2:  D_OUT = D2;
             3:  D_OUT = D3;
             4:  D_OUT = D4;
             5:  D_OUT = D5;
             6:  D_OUT = D6;
             7:  D_OUT = D7;
             //default: D_OUT = 0;
          endcase
       end

endmodule
```

12) Convert the model from the previous problem to use a parameter for the data width.

ANS:

```
module mux_8t1_nb #(parameter n=3) (
       input wire [2:0] SEL,
       input wire [n-1:0] D0,
       input wire [n-1:0] D1,
       input wire [n-1:0] D2,
       input wire [n-1:0] D3,
       input wire [n-1:0] D4,
       input wire [n-1:0] D5,
       input wire [n-1:0] D6,
       input wire [n-1:0] D7,
       output reg [n-1:0] D_OUT  );


       always @(*)
       begin
          case (SEL)
             0:  D_OUT = D0;
             1:  D_OUT = D1;
             2:  D_OUT = D2;
             3:  D_OUT = D3;
             4:  D_OUT = D4;
             5:  D_OUT = D5;
             6:  D_OUT = D6;
             7:  D_OUT = D7;
             //default: D_OUT = 0;
          endcase
       end

endmodule
```

13) Repeat the previous problem but include a control input: CS. When this input is asserted (active high), the MUX acts like a standard MUX; when not asserted, the MUX output is zero.

ANS:

```
module mux_8t1_nb #(parameter n=3) (
       input wire         CS
       input wire [2:0]   SEL,
       input wire [n-1:0] D0,
       input wire [n-1:0] D1,
       input wire [n-1:0] D2,
       input wire [n-1:0] D3,
       input wire [n-1:0] D4,
       input wire [n-1:0] D5,
       input wire [n-1:0] D6,
       input wire [n-1:0] D7,
       output reg [n-1:0] D_OUT  );


       always @(*)
       begin
          if (CS == 1'b1)   // chip select
             case (SEL)
                0:  D_OUT = D0;
                1:  D_OUT = D1;
                2:  D_OUT = D2;
                3:  D_OUT = D3;
                4:  D_OUT = D4;
                5:  D_OUT = D5;
                6:  D_OUT = D6;
                7:  D_OUT = D7;
                //default: D_OUT = 0;
             endcase
          else
                D_OUT = 0;

       end   // always block

endmodule
```

14) Briefly describe why we never remove unused inputs and outputs from modules when we instantiate them into our design.

ANS: If you remove an input, the synthesizer must assign a value for that input, which is problematic. If you remove an output, human readers of the model are not sure is you removed the output for a legitimate reason or not; humans may think you simply forgot to assign that output.

15) Briefly describe what happens when you do not provide a signal name for an input in an instantiated module.

ANS: If you don't assign an input value, the synthesizer assigns an input value for you, which is problematic because you don't know what value the synthesizer will assign.

16) Briefly describe the correct approach to mapping unused inputs in module instantiations.

ANS: All inputs need to be assigned to some value, which means either a one or a zero.

17) Briefly describe the correct approach to mapping unused outputs in module instantiations.

ANS: Unused outputs should remain in the instantiation listing; the associated parenthesis should remain empty.

18) Briefly describe why it is good practice to never allow procedural assignment statement catchalls to include expected cases.

ANS: Including expected cases in the catchall confuses human reader of the models; additionally such models are problematic to modify and maintain.

Chapter 9

1) Briefly explain why we often list all the outputs of an **`always`** block at the beginning of the **`always`** block (after the **`begin`** clause).

ANS: We typically declare all the outputs of an always block after the block begins to prevent the synthesizer from generating a latch on any given output signal.

2) Briefly explain what happens when an **`always`** block does not assign all the outputs once an input change causes the **`always`** block to be evaluated.

ANS: Any value that is not assigned during the evaluation of an always block becomes a latch.

3) Briefly describe what happens if you don't override the parameters when you instantiate a parameterized module.

ANS: If you don't override default parameters in the model, the instantiation takes the default value.

4) Briefly describe why explicitly listing all parameterized when instantiating a parametrized model is a better idea than relying on a parameterized model's default values.

ANS: Explicitly stating all parameter default overrides in the instantiation makes the model easier to understand for humans and also makes the model easier to modify.

5) Provide a model for a 22-bit comparator using at least two instantiated comparator modules. For this problem, use the model in **Error! Reference source not found.** as the module to instantiate.

```
module comp_nb  #(parameter n=8) (a, b, eq, lt, gt);
    input  [n-1:0] a, b;    //- on one line to save space
    output reg eq, lt, gt;

    always @ (a,b)
    begin
       if (a == b)
       begin
          eq = 1; lt = 0;  gt = 0;
       end
       else if (a > b)
       begin
          eq = 0; lt = 0;  gt = 1;
       end
       else if (a < b)
       begin
          eq = 0; lt = 1;  gt = 0;
       end
       else
       begin
          eq = 0; lt = 0;  gt = 0;
       end
    end


endmodule
```

ANS:

```
module comp_22b(a, b, eq);
   input  [21:0] a,b;
   output eq;

   //- intermediate signal declarations
   wire eq18,eq4;

   assign eq = eq4 & eq18; // continuous assignment

   //- instantiate 4-bit comparator
   comp_nb  #(.n(4)) MY_COMP4 (
      .a  (a[21:18]), //- most significant 4 bits
      .b  (b[21:18]),
      .eq (eq4),
      .gt () ,
      .lt ()    );

   //- instantiate 18-bit comparator
   comp_nb  MY_COMP18 (
      .a  (a[17:0]), //- least significant 18 bits
      .b  (b[17:0]),
      .eq (eq18),
      .gt (),
      .lt ()   );


endmodule
```

6) Briefly describe whether a given set of bits in a digital circuit know whether they represent a signed or unsigned number.

ANS: The hardware just deals with bits; the hardware has no notion of whether the given set of bits will be interpreted by the outside world as either signed or unsigned.

7) Briefly describe what we mean by the "signedness" of a number.

ANS: The signedness of a number refers to whether some entity (such as the synthesizer or a human) interprets a value as either signed or unsigned.

8) Briefly describe what happens if you don't state the sign of a value in a declaration.

ANS: If you don't state the sign of a value when you declare it, the synthesizer assumes the value is unsigned.

9) Briefly describe if the values in a given design are signed or unsigned.

ANS: The values in a given design are unsigned unless explicitly declared as signed.

Chapter 10

1) Briefly describe why using a full adder in the LSB position of an RCA provides more flexibility.

ANS: Placing a full adder in the LSB position allows the RCA to be cascaded to effectively produce an RCA of greater bit-width.

2) Briefly explain the relationship between using a Verilog mathematical operator in a model and the resultant circuit generated by the synthesizer.

ANS: If you use a Verilog mathematical operator, the synthesizer must then provide (synthesize) the hardware that can implement that mathematical operation.

3) Briefly describe the difference between a *correct* result and a *valid* result for a given arithmetic operation.

ANS: The hardware will always generate a result for whatever operation the hardware is asked to do; this result is correct. Beyond that, in many operations, the hardware must verify the result is also valid. For example, RCAs always generate the correct result, but not always a valid result dependent upon the operands that were being added.

4) Which format does Verilog use to represent signed numbers?

ANS: Verilog uses two's complement (radix complement) notation to represent negative number.

5) Briefly describe if it possible to have a 1-bit signed number using 2's complement notation.

ANS: No; there is no such thing as a 1-bit number in 2's complement format. Yeah, what is the sound of one hand clapping? Ask an academic administrator to get a BS reply; they're good at that.

6) List the two situations where we use typecasting in Verilog models.

ANS: We use typecasting to 1) direct Verilog as to how to expand the bit-width of values in expressions, and 2) to direct Verilog as to how interpret values in comparison operations.

7) Briefly describe what determines when Verilog automatically expands the bit-width of values.

ANS: Verilog automatically extends the bit-width of values in an expression when there is a value in an expression with a larger bit-width.

8) Briefly describe what determines how Verilog expands the bit-width of numbers when it needs to.

ANS: Verilog expands the bit-width of value according to the sign of the given value; the sign can be modified with a type-cast.

9) What conditions needs to be in place in order for an expression to be evaluated as a signed value.

ANS: For an expression to be treated as signed, every operand on the right side of the assignment operator and the result must be signed.

10) Briefly describe if you the Verilog modeler know any specifics about how the hardware implements mathematical operators used in a given model.

ANS: Hardware modelers generally don't know what hardware the synthesizer will generate when they use mathematical operators. To know this information, the modeler must understand both the rules of Verilog and the basic features of the hardware the circuit will be implemented on (such as silicone vs. FPGA).

11) Briefly explain why the hardware generated from a good RCA model will always generate the correct result, but that result may not be valid.

ANS: The hardware is dumb; it just adds the input values and generates a result. The RCA does not know or care whether the inputs are signed or unsigned. The sum is thus always correct, but designers must add more hardware to determine whether the sum is valid or not.

12) Briefly explain how modelers determine the validity of the sum output of signed operations when using an RCA.

ANS: If the RCA is performing a signed operation, there are several methods to determine whether the sum is valid or not, which typically involve the sign of the input operands and sum. For example, if the sign of the two addends is the same, but different from the sign of the sum, the sum is not valid.

13) Briefly explain how modelers determine the validity of the sum output of unsigned operations when using an RCA.

ANS: If the RCA is performing an unsigned operation, the carry-out determines the validity of the n-bit sum (if the carry-out is asserted, the sum output is not valid).

14) Briefly explain whether RCA can have both signed and unsigned inputs and still generate the correct and valid sum.

ANS: An RCA can have both unsigned and signed values in an expression and *sometimes* generate the correct answer, but doing so does not guarantee it will always generate a correct and valid answer.

Chapter 11

1) List the two main ways modelers can create sequential circuits using Verilog.

ANS: Modelers can create sequential circuits by not specifying an output for every possible input condition (generates a latch), or, by using a "posedge" or "negedge" in the sensitivity list (generates a register).

2) Briefly describe what the term "level sensitive" refers to in the context of a simple latch.

ANS: Level sensitive refers to the notion the latch can change state any time; such changes do not need to be synchronized with a given signal edge.

3) List the two contextual definitions of the word "latch".

ANS: The noun "latch" refers to a 1-bit level-sensitive memory element. The verb "to latch" refers to placing a value into a synchronous sequential circuit (such as a register).

4) Describe the similarities, if any, between the sequential nature of statements within an **`always`** block and sequential circuit.

ANS: There are no similarities between these two terms other than the word sequential.

5) Provide the Verilog model for a D flip-flop that has active low asynchronous preset and clear. The preset takes precedence over the clear.

ANS:

```
module d_ff_preset_clr(D, CLK, Q, nCLR, nPSET);
    input D, CLK, nCLR, nPSET;
    output reg Q;

    always @ (negedge nCLR, negedge nPSET, posedge CLK)
    begin
       if (nPSET == 1'b0)
          Q <= 1'b1;
       else if (nCLR == 1'b0)
          Q <= 1'b0;
       else
          Q <= D;
    end

endmodule
```

6) Repeat the previous problem but make the clear has precedence over the preset.

ANS:

```
module d_ff_preset_clr(D, CLK, Q, nCLR, nPSET);
    input D, CLK, nCLR, nPSET;
    output reg Q;

    always @ (negedge nCLR, negedge nPSET, posedge CLK)
    begin
       if (nCLRT == 1'b0)
          Q <= 1'b1;
       else if (nPSET == 1'b0)
          Q <= 1'b0;
       else
          Q <= D;
    end

endmodule
```

Chapter 12

1) List the various types of control inputs a typical register can have.

ANS: The typical register controls are load, clear, and preset (all 1's). We typically consider the clock as a control input as well; most operations with the counter are synchronous but can be modeled to be asynchronous as well.

2) Briefly explain whether you can know if register control inputs such as clr, ld, preset, etc. are synchronous or asynchronous by looking at the schematic diagram.

ANS: The schematic diagram does not indicate whether control inputs are synchronous or asynchronous; typically annotations included with the schematic diagram will provide that information.

3) Briefly explain whether you can know if which typical register control inputs such as clr, ld, preset, etc. have a higher precedence.

ANS: Once again, this information is not apparent from the schematic, and yet once again, the schematic diagram should state the precedence and the synchronicity of the inputs.

4) Briefly explain whether an **`always`** block be active to both the positive and negative edge of a signal.

ANS: An always block can't be sensitive to the both positive and negative edges of the same signal. If your circuit needs that sort of functionality, you'll have to model it in some other way.

5) Briefly explain whether the order in which **`if`** clauses appear in the body of an **`always`** block affect the operation of the device.

ANS: The order of the clauses in an if statement affect the operation of the device by essentially providing precedence. The clauses in an if statement are evaluated in the order they appear; once a given if clause evaluates as true, the if statement essentially terminates.

Chapter 13

1) List the three different parts in a finite state machine.

ANS: The three part of a FSM are 1) the next state decoder, 2) the output decoder, and 3) the state registers.

2) Briefly describe the differences between the two types of outputs in a FSM.

ANS: The two types of outputs are Mealy and Moore-type outputs. The Moore-type outputs are a function of state only while Mealy-type outputs are a function of both state and external inputs.

3) Briefly describe the most efficient way for humans to describe and understand the operation of FSMs.

ANS: State diagrams provide the most efficient way for humans to understand FSMs.

4) Briefly explain the function of an **`if`** statement within a state clause for behaviorally modeled FSMs.

ANS: The if statement provides the construct for conditional outputs (Mealy-type outputs) and state transitions (the conditional state transitions).

5) Briefly explain how we make FSM self-correcting when modeling them with Verilog.

ANS: We make FSM self-correction by providing a catchall statement control the transitions from undefined states in the states. In this context, undefined states are the states that "exist" based on unused bit combinations associated with the state registers.

6) Briefly explain it is good practice to assign all outputs the state of the combinatorial process in an FSM.

ANS: Assigning all the outputs at the start of the combinatorial process ensures that the process will not inadvertently generate a latch.

7) Briefly describe what happens in a single state we don't assign all the outputs in a behaviorally modeled FSM.

ANS: If the output was assigned a "default" value, the FSM will take that default value; otherwise, the FSM will need to "remember" the previous value of the output, and thus generate a latch (memory).

8) We can classify the outputs of an FSM as either Mealy or Moore outputs. How is it then that we can say that in a given state, a known Mealy output acts like a Moore output?

ANS: In some states, a given output that had conditional values in another state (Mealy) can have unconditional values (Moore) in a given state. In this case, the output is considered a Mealy-type output regardless.

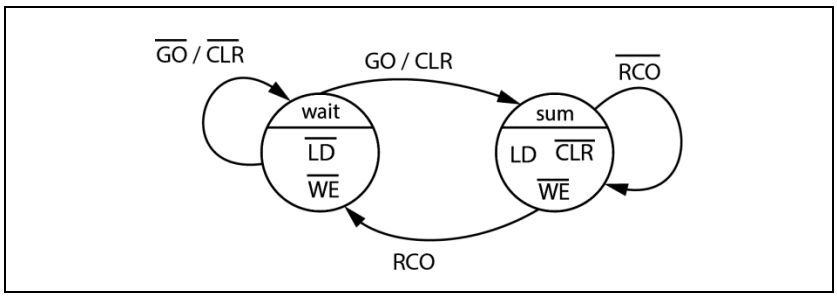
9) Briefly explain under what conditions we can model a Mealy output as a Moore output.

ANS: We can model a Mealy-type output as a Moore-type output in a given state is that output has the same value of all conditions in that state (thus, the output is unconditional).

10) Briefly explain under what conditions we can model a Moore output a Mealy output.

ANS: Under no conditions can you model a Moore-type output as a Mealy-type output; such an attempt would violate the basic definition of the terms.

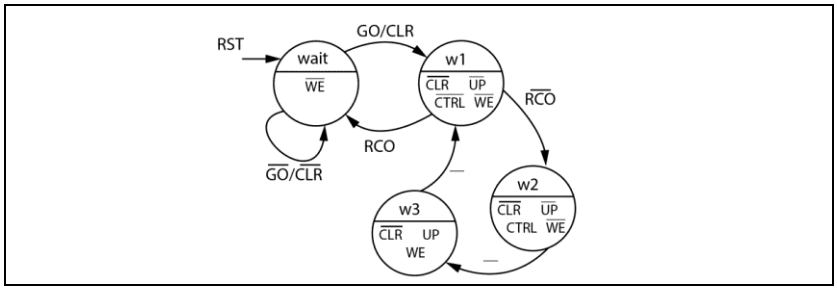
11) Provide Verilog behavioral model for the following state diagram.



ANS:

```
module fsm_a(GO, CLR, clk, LD, WE, RCO);
    input  GO, clk, RCO;
    output reg CLR, LD, WE;

    //- next state & present state variables
    reg NS, PS;
    //- bit-level state representations (defined as constants)
    parameter st_wait=1'b0, st_sum=1'b1;

    //- model the state registers
    always @ (posedge clk)
          PS <= NS;

    //- model the next-state and output decoders
    always @ (GO, RCO, PS)
    begin
       CLR = 1'b0; LD = 1'b0; WE = 1'b0; // assign all outputs
       case(PS)
          st_wait:  //--------------------------------
          begin
             LD = 1'b0;  WE = 1'b0;
             if (GO == 1'b0)
             begin
                CLR = 1'b0;
                NS = st_wait;
             end
             else
             begin
                CLR = 1'b1;
                NS = st_sum;
             end
          end

          st_sum:  //--------------------------------
          begin
             LD = 1'b1;  WE = 1'b0; CLR = 1'b0;
             if (RCO == 1'b0)
                NS = st_sum;
             else
                NS = st_wait;
          end

          default: NS = st_wait;
          endcase
      end

endmodule
```

12) Provide Verilog behavioral models for the following state diagram.

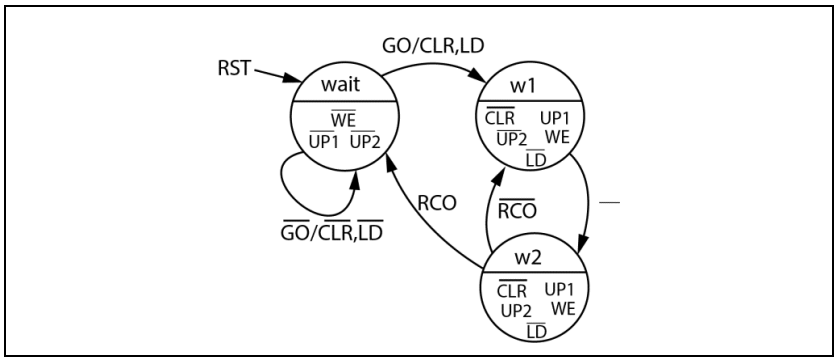


ANS:

```
module fsm_b(GO, CLR, clk, WE, RCO, CTRL, RST, UP);
    input  GO, clk, RCO, RST;
    output reg CLR, WE, CTRL, UP;

    //- next state & present state variables
    reg [1:0] NS, PS;
    //- bit-level state representations (defined as constants)
    parameter st_wait=2'b00, st_w1=2'b01, st_w2=2'b10, st_w3=2'b11;

    //- model the state registers
    always @ (posedge clk, posedge RST)
       if (RST == 1'b1)
          PS <= st_wait;
       else
          PS <= NS;

    //- model the next-state and output decoders
    always @ (GO, RCO, PS)
    begin
       CLR = 1'b0; WE = 1'b0; CTRL = 1'b0; UP = 1'b0; // assign all outputs
       case(PS)
          st_wait:  //--------------------------------
          begin
             WE = 1'b0;
             if (GO == 1'b0)
             begin
                CLR = 1'b0;
                NS = st_wait;
             end
             else
             begin
                CLR = 1'b1;
                NS = st_w1;
             end
          end

          st_w1:  //--------------------------------
          begin
             CLR = 1'b0; WE = 1'b0; CTRL = 1'b0; UP = 1'b0;
             if (RCO == 1'b1)
                NS = st_wait;
             else
                NS = st_w2;
          end

          st_w2:  //--------------------------------
          begin
             CLR = 1'b0; WE = 1'b0; CTRL = 1'b1; UP = 1'b0;
             NS = st_w3;
          end

          st_w3:  //--------------------------------
          begin
             CLR = 1'b0; WE = 1'b1; UP = 1'b1;
             NS = st_w1;
          end
```

```
          default: NS = st_wait;
          endcase
      end

endmodule
```

13) Provide Verilog behavioral models for the following state diagram.



ANS:

```
module fsm_b(GO, CLR, clk, WE, RCO, RST, UP1, UP2, LD);
    input  GO, clk, RCO, RST;
    output reg CLR, WE, LD, UP1, UP2;

    //- next state & present state variables
    reg [1:0] NS, PS;
    //- bit-level state representations (defined as constants)
    parameter st_wait=2'b00, st_w1=2'b01, st_w2=2'b10;

    //- model the state registers
    always @ (posedge clk, posedge RST)
       if (RST == 1'b1)
          PS <= st_wait;
       else
          PS <= NS;

    //- model the next-state and output decoders
    always @ (GO, RCO, PS)
    begin
       CLR = 1'b0; WE = 1'b0; LD = 1'b0; UP1 = 1'b0; UP2 = 1'b0; // assign all outputs
       case(PS)
          st_wait:  //--------------------------------
          begin
             WE = 1'b0; UP1 = 1'b0;  UP2 = 1'b0;
             if (GO == 1'b0)
             begin
                CLR = 1'b0;  LD = 1'b0;
                NS = st_wait;
             end
             else
             begin
                CLR = 1'b1; LD = 1'b1;
                NS = st_w1;
             end
          end

          st_w1:  //--------------------------------
          begin
             CLR = 1'b0; WE = 1'b1; LD = 1'b0; UP1 = 1'b1;  UP2 = 1'b0;
             NS = st_w2;
          end


          st_w2:  //--------------------------------
          begin
```

```
            CLR = 1'b0; WE = 1'b1; LD = 1'b0; UP1 = 1'b1; UP2 = 1'b1;
            if (RCO == 1'b1)
               NS = st_wait;
            else
               NS = st_w1;
         end

         default: NS = st_wait;
         endcase
      end

endmodule
```

Chapter 14

1) Briefly explain why we consider counters to be a special type of register.

ANS: We consider counter to be special types of registers because the counter must "remember" the counter between clock edges; registers are our basic memory element.

2) Briefly describe the accepted relationship between counting up and down and incrementing/decrementing.

ANS: Incrementing and decrementing are generally accepted to mean counting up by one or counting down by one, respectively. If you need to increment by 3, such an operation is better described as counting up by three rather than incrementing.

3) Briefly explain how Verilog determines the precedence of counter control inputs.

ANS: Counters are best modeling using behavioral models. As such, the procedural programming statements such as if statements interpret the conditions associated with those states in the order they appear in the statement; once a given statement is found to be true, the if statement terminates.

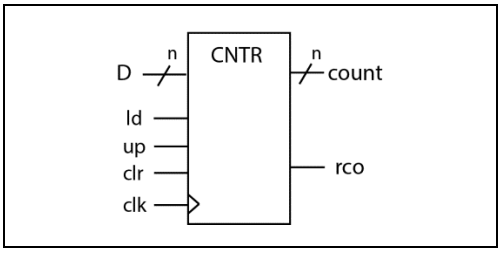
4) Briefly describe how Verilog determines which signal edges the counter's actions are synchronized to.

ANS: The signal edge the counter is synchronized to is listed in the procedural programming block's sensitivity list.

5) Briefly explain why the ripple carry out status signal from an up/down counter is dependent upon current count direction.

ANS: The ripple carry out (RCO) signal is generally considered to be the smallest counter value when counting down (all zeros) or the largest value when counting up (all ones); because of this, the RCO signal is a function of both the count and the count direction.

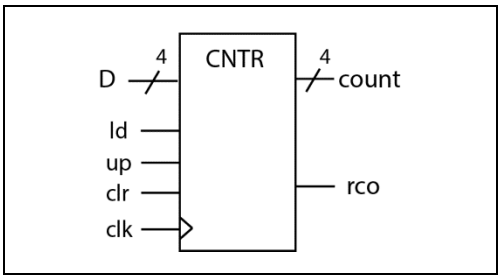
6) Provide a Verilog model that describes the operation of the following up/down counter according to the included diagram. All control signals are active high. All control signals are synchronous except for the **ld** signal, which is asynchronous. The **clr** signal has precedence over the **up** signal. The **ld** signal has precedence over the **clr** and **up** signals. When **up** signal is asserted the counter increments (according to precedence rules); otherwise the counter decrements.



ANS:

```
module cntr_udclr_nb #(parameter n=8) (clk, clr, up, ld, D, count,rco);
    input   clk;
    input   clr;
    input   up;
    input   ld;
    input   [n-1:0] D;
    output  reg [n-1:0] count;
    output  reg rco;

    always @(posedge ld, posedge clk)
    begin
        if (ld == 1)        // asynch reset
           count <= D;
        else if (clr == 1)   // load new value
           count <= 0;
        else if (up == 1)   // count up (increment)
           count <= count + 1;
        else if (up == 0)   // count down (decrement)
           count <= count - 1;
    end


    //- handles the RCO, which is direction dependent
    always @(count, up)
    begin
       if ( up == 1 && &count == 1'b1)
          rco = 1'b1;
       else if (up == 0 && |count == 1'b1)
          rco = 1'b1;
       else
          rco = 1'b0;
    end

endmodule
```

7) Provide a Verilog model that describes a 4-bit decade up counter. This counter has an asynchronous clear and an RCO that indicates when the output has reached its maximum value. The **up** input serves as a hold when not asserted. The **clr** has precedence over the **ld** and up signals; the **ld** signal has precedence over the **up** signal.



ANS:

```
module cntr_dec_4b (clk, clr, up, ld, D, count, rco);
    input   clk;
    input   clr;
    input   up;
    input   ld;
    input   [3:0] D;
    output  reg [3:0] count;
    output  reg rco;

    always @(posedge ld, posedge clk)
    begin
        if (clr == 1)       // asynch reset
           count <= D;
        else if (ld == 1)   // load new value
           count <= 0;
        else if (up == 1)   // count up (increment)
           if (count == 4'b1001)
              count <= 4'b0000;
           else
              count <= count + 1;
    end

    //- handles the RCO
    always @(count)
    begin
       if ( count == 4'b1001)
          rco = 1'b1;
       else
          rco = 1'b0;
    end


endmodule
```

Chapter 15

1) Briefly explain why we often use shift registers in designs that rely on “integer math”.

ANS: We often use shift registers to support integer math because the can perform multiplications and divisions (by integral powers of two) fast compared to other mathematical algorithms.

2) Briefly explain why using Verilog shift operators is less flexible than coding the shift operations directly using concatenation operators.

ANS: The Verilog shift operators automatically insert zeros into unfilled bit locations, which means you need to not use the shift operators if you require ones to be shifted into the register.

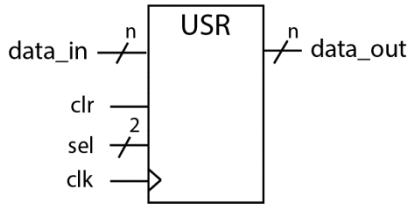
3) Briefly describe whether the basic Verilog shift operators need to know the declared signedness of numbers in order for the operator to work correctly.

ANS: The basic shift operators (non-arithmetic) shift in zeros for both signed and unsigned values. What gets shifted into signed number for arithmetic shift operators is a function of the signedness (how the values are declared) of the values.

4) Briefly describe whether the underlying hardware ever knows about the signedness of the numbers that it will shift.

ANS: The hardware knows nothing about signedness of values; the hardware just does what it’s told to do.

5) Write a Verilog model for a generic n-bit shift register that does the following operations: do not use arithmetic shift operators in your solution. The clr signal is an asynchronous active high reset signal.

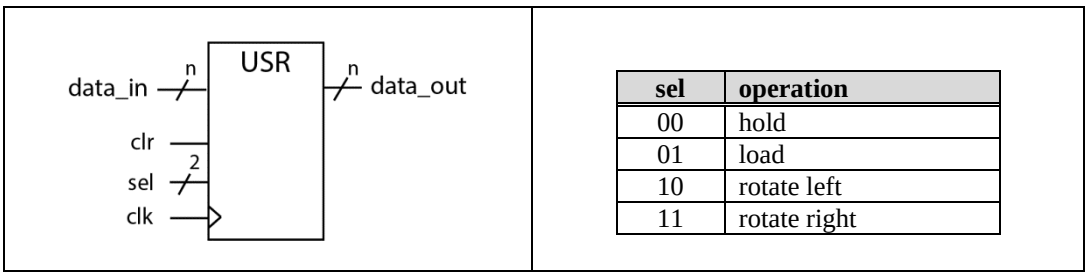


sel	operation
00	hold
01	load
10	arithmetic shift left (fill in zeros on right)
11	arithmetic shift right

ANS:

```
module usr_nb #(parameter n=8) (
    input  wire [n-1:0] data_in,
    input  wire dbit,
    input  wire clk,
    input  wire clr,
    input  wire [1:0] sel,
    output reg [n-1:0] data_out  );


    always @(posedge clr, posedge clk)
    begin
        if (clr == 1'b1)     // asynch +logic reset
           data_out <= 0;
        else
           case (sel)
              0: data_out <= data_out;                // hold value
              1: data_out <= data_in;                 // load
              2: data_out <= {data_out[n-2:0],dbit};  // shift left
              3: data_out <= {dbit,data_out[n-1:1]};  // shift right
              //default data_out <= 0;
           endcase
    end

endmodule
```

6) Write a Verilog model for a shifter that does the following operations: use arithmetic shift operators in your solution when possible. The clr signal is an asynchronous active high reset signal.



sel	operation
00	hold
01	load
10	rotate left
11	rotate right

ANS:

```
module usr_nb #(parameter n=8) (
    input  wire [n-1:0] data_in,
    input  wire clk,
    input  wire clr,
    input  wire [1:0] sel,
    output reg [n-1:0] data_out  );


    always @(posedge clr, posedge clk)
    begin
        if (clr == 1'b1)     // asynch +logic reset
           data_out <= 0;
        else
           case (sel)
              0: data_out <= data_out;                // hold value
              1: data_out <= data_in;                 // load
              2: data_out <= data_out << 1;           // shift left
              3: data_out <= data_out >> 1;           // shift right
              //default data_out <= 0;
           endcase
    end

endmodule
```

Chapter 16

1) What are the two main sources of errors in our HDL models?

ANS: The two main sources of errors in HDL models are 1) from humans making syntax and general design errors, and 2) the synthesizer interpreting models in ways that the modeler did not expect (which are essentially the same error if you think about it).

2) Briefly explain why typical academic courses focus on design and attenuate the teaching of verification?

ANS: Academia focuses on design because that forms the main half of digital design. Verification is equally as important (probably more important) but digital design courses often don't leave time for learning about the power of circuit verification.

3) List and describe the two main types of verification.

ANS: There is functional verification and behavioral verification. Behavior verification is mostly associated with ensuring the logic in your circuit works, but ignores other physical characteristics of circuits such as propagation delays. Functional verification is much closer to the actual hardware associated with the circuit in that it attempts to include propagation delays and true physical timing issues associated with a given circuit in the testing of that circuit.

4) Briefly explain why verification is more of an art form than a science.

ANS: Its more of an art form because it requires such a wide knowledge base to do and a lot of cleverness to successfully test a meaningful portion of circuit and then be able to state with confidence that the circuit actually works.

5) Briefly explain why a 100% complete testing of a large and/or complex circuit is virtually impossible.

ANS: It is impossible to test a large circuit at a 100% level because there are so many gates in the circuit that testing them all would require highly specialized tests and would take many human lifetimes to complete.

6) List the two main blocks in a testbench.

ANS: The two main blocks in a testbench are the device under test (DUT) and the stimulus driver.

7) Briefly explain why it is that testbenches have no external interface.

ANS: Testbenches have no external interface because they do not need to interact with the outside world; in other words, testbench models are self-contained.

8) Briefly explain why behavioral verification is only the first step in verifying your circuit is operating correctly.

ANS: It is only the first step because although you circuit may perform the way you designed it, the design may not adequately solve the problem at hand.

9) List the two types Verilog's procedural blocks.

ANS: Verilog has both initial and always blocks, which are both procedural blocks.

10) Briefly explain the main differences between the two types of procedural blocks in Verilog.

ANS: The two types of block are always blocks and initial block. The main difference is that initial blocks are not synthesizeable and are thus only used in verification. Additionally, initial blocks have no sensitivity list and are evaluated only one time (when the simulation starts, or at time zero), while always blocks are evaluated any time a signal the block is active to changes.

11) When using initial blocks in your testbench, briefly explain at what time the initial block executes.

ANS: The initial block executes or evaluates only one time: at time zero, or the start of the simulation.

12) If you used the following initial block to generate a clock signal, what would be the period of the clock in time units?

```
initial
begin
   clk = 0;    //- init signal
   forever  #20 clk = ~clk;
end;
```

ANS: The clock period would be 40 time units because the clock signal changes state every 20 time units.

13) What would the maximum number of initial blocks you could use in a testbench model?

ANS: There is not stated maximum number of initial blocks in a testbench; it there is a limit, it would be provided by the associated tools.

14) Briefly describe one advantage to using multiple initial blocks for different signals in a single testbench model.

ANS: Using multiple initial blocks to divide drivers could potentially make the testbench more readable and understandable to humans; it would also be thus easier to modify.