Three sources of help
1. Chapter 4 of Harris and Harris
2. IEEE Standard for System verilog
3. Quartus II Handbook Version 13.1

.

SystemVerilog

1. Use `always_ff @(posedge clk)` and nonblocking assignments to model synchronous sequential logic.

```
always_ff @(posedge clk)
  begin
    n1 <= d; // nonblocking
    q <= n1; // nonblocking
  end
```

2. Use continuous assignments to model simple combinational logic.

```
assign y = s ? d1 : d0;
```

3. Use `always_comb` and blocking assignments to model more complicated combinational logic where the `always` statement is helpful.

```
always_comb
  begin
    p = a ^ b; // blocking
    g = a & b; // blocking
    s = p ^ cin;
    cout = g | (p & cin);
  end
```

4. Do not make assignments to the same signal in more than one `always` statement or continuous assignment statement.

4.7.1 SystemVerilog

Prior to SystemVerilog, Verilog primarily used two types: `reg` and `wire`. Despite its name, a `reg` signal might or might not be associated with a register. This was a great source of confusion for those learning the language. SystemVerilog introduced the `logic` type to eliminate the confusion; hence, this book emphasizes the `logic` type. This section explains the `reg` and `wire` types in more detail for those who need to read old Verilog code.

In Verilog, if a signal appears on the left hand side of <= or = in an `always` block, it must be declared as `reg`. Otherwise, it should be declared as `wire`. Hence, a `reg` signal might be the output of a flip-flop, a latch, or combinational logic, depending on the sensitivity list and statement of an `always` block.

Input and output ports default to the `wire` type unless their type is explicitly defined as `reg`. The following example shows how a flip-flop is described in conventional Verilog. Note that `clk` and `d` default to `wire`, while `q` is explicitly defined as `reg` because it appears on the left hand side of <= in the `always` block.

```
module flop(input            clk,
            input      [3:0] d,
            output reg [3:0] q);

  always @(posedge clk)
    q <= d;
endmodule
```

SystemVerilog introduces the `logic` type. `logic` is a synonym for `reg` and avoids misleading users about whether it is actually a flip-flop. Moreover, SystemVerilog relaxes the rules on `assign` statements and hierarchical port instantiations so `logic` can be used outside `always` blocks where a `wire` traditionally would have been required. Thus, nearly all SystemVerilog signals can be `logic`. The exception is that signals with multiple drivers (e.g., a tristate bus) must be declared as a net, as described in HDL Example 4.10. This rule allows SystemVerilog to generate an error message rather than an `x` value when a `logic` signal is accidentally connected to multiple drivers.

The most common type of net is called a `wire` or `tri`. These two types are synonymous, but `wire` is conventionally used when a single driver is present and `tri` is used when multiple drivers are present. Thus, `wire` is obsolete in SystemVerilog because `logic` is preferred for signals with a single driver.

Verilog HDL State Machines

To ensure proper recognition and inference of Verilog HDL state machines, observe the following additional Verilog HDL guidelines. Some of these guidelines may be specific to Quartus II integrated synthesis. Refer to your synthesis tool documentation for specific coding recommendations.

If the state machine is not recognized and inferred by the synthesis software (such as Quartus II integrated synthesis), the state machine is implemented as regular logic gates and registers, and the state machine is not listed as a state machine in the **Analysis & Synthesis** section of the Quartus II Compilation Report. In this case, the software does not perform any of the optimizations that are specific to state machines.

- If you are using the SystemVerilog standard, use enumerated types to describe state machines. For more information, refer too "SystemVerilog State Machine Coding Example" on page 13–65.
- Represent the states in a state machine with the parameter data types in Verilog-1995 and Verilog-2001, and use the parameters to make state assignments. For more information, refer too"Verilog-2001 State Machine Coding Example" on page 13–63. This parameter implementation makes the state machine easier to read and reduces the risk of errors during coding.

Altera recommends against the direct use of integer values for state variables, such as `next_state <= 0`. However, using an integer does not prevent inference in the Quartus II software.

- No state machine is inferred in the Quartus II software if the state transition logic uses arithmetic similar to that in the following example:

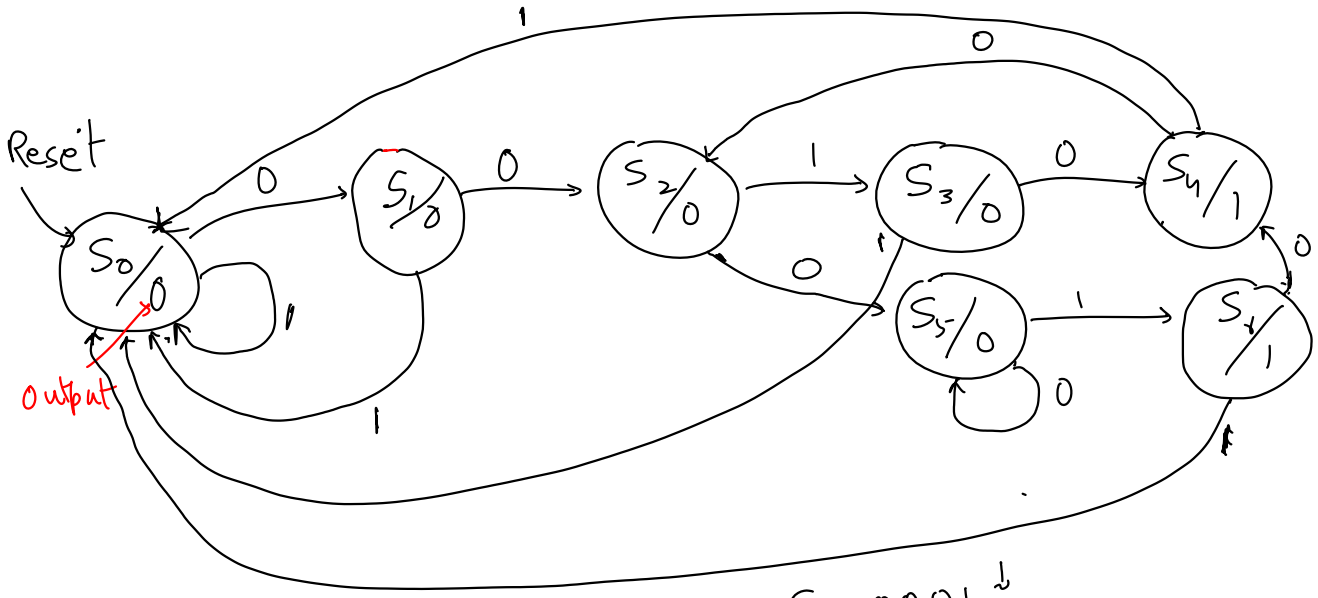
```
case (state)
   0: begin
      if (ena) next_state <= state + 2;
      else next_state <= state + 1;
   end
   1: begin
   ...
endcase
```

- No state machine is inferred in the Quartus II software if the state variable is an output.
- No state machine is inferred in the Quartus II software for signed variables.

Moore state graph

Sequence detector

0 or
0010
or
0001
] can overlap



$S_0 = [\,]$
$S_1 = [0]$
$S_2 = [00]$
$S_3 = [001]$
$S_4 = [0010]$
$S_5 = [000]$
$S_6 = [0001]$

$S_6 = 0001$ ↓ 1

S.