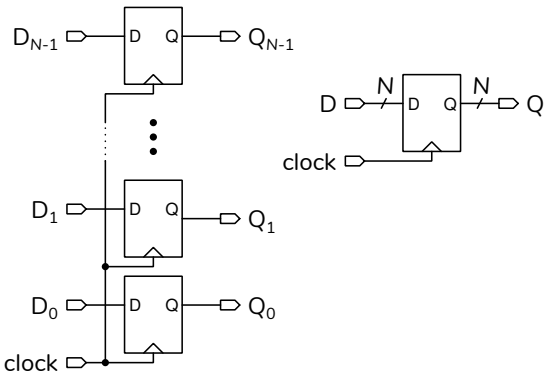
State Machines

This lecture defines state machines and describes how to document them and how to implement them using Verilog. After this lecture you should be able to: design a state machine based on an informal description of its operation, document it using state transition diagrams and tables, write a synthesizable Verilog description of it and convert between these three descriptions.

Introduction

Registers

We can connect N flip-flops to the same clock to form an N-bit *register*. The common clock loads every flip-flop at the same time:

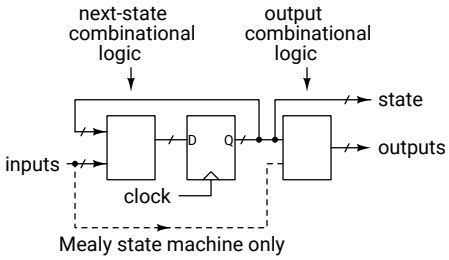


The Verilog `always_ff` statement creates a register.

State

The *state* of a register is its value. A *state machine* is a description of how the state changes.

The schematic of a state machine is a register whose next value is selected by a combination of the current state and, optionally, input signals:



State transitions, which are changes in the register value, only happen at the rising edge of the clock. The state register is always loaded on each rising edge of the clock, but there is no change of state if it's loaded with its current value.

The output of a state machine can be its state. Alternatively, a logic function such as lookup table can generate the desired output for each state.

The above describes a *Moore* state machine. A *Mealy* state machine is one where the output is a function of the current state and the inputs.

State Machine Descriptions

For example, consider a state machine with two bits of state that sequences through the values 00, 01, 10, 11 and back to 00. The output should be 0 in states 00 and 01 and 1 in states 10 and 11.

The following truth tables define this state machine's next-state and output combinational logic blocks:

state	next state
00	01
01	10
10	11
11	00

state	output
00	0
01	0
10	1
11	1

State Transition Table

A state transition table is a truth-table description of the next-state logic. It has columns for the current state, the input value(s), and the corresponding next state. Some useful conventions are: (1) x can be used for "don't care" state or input values; (2) the first matching state/input row is chosen; (3) if there is no match then there is no change of state; (4) expressions can be used to define the next state as a function of the current state and the input.

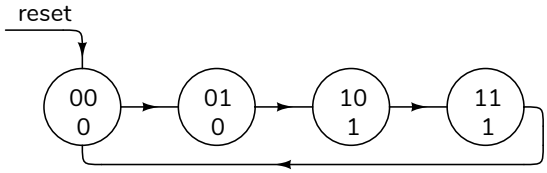
For example, if we added a reset input that set the state to 00 when it was asserted (1), we could write the state transition table for this resetable counter as:

state	reset	next state
xx	1	00
11	0	00
n	0	$n+1$

State Transition Diagram

A state machine can also be described by a state transition diagram drawn using the following conventions: (1) each state is represented by a circle labelled with the state value (or name); (2) each state shows the output for that state (unless the state is the output); (3) arrows show possible transitions between states; (4) transitions happen when the expression is non-zero (true); (5) unlabelled transitions happen unconditionally; (6) transitions with no origin come from all states; (7) conditions that don't cause a state change are not shown[1]; (8) only one transition out of a state may be true (the conditions must be mutually exclusive).

The following is a state transition diagram for the state machine above:



Exercise 1: Modify the diagram so the state machine counts to 11 and stops. Add a down input that cause the values to count down.

State Machines in Verilog

A state machine can be written in Verilog using one conditional operator for each transition in the diagram or table. The condition is true if the state and input value(s) match the values for that transition. The true value is the next state for that transition. The false value is the next conditional operator or, if no transition matches, the current state.

The Verilog for the example above would be:

[1]Some authors do not allow this.

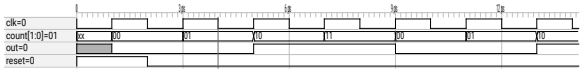
```
// 2-bit clock divider with reset

module ex67
   ( output logic [1:0] count,
     output logic out,
     input logic reset, clk ) ;

   always_ff @(posedge clk) count
      <= reset ? 2'b00 :
         count == 2'b11 ? 2'b00 :
         count + 1'b1 ;

   assign out = count == 2'b10 || count == 2'b11 ;

endmodule
```



Counters

A counter is a common state machine. Typical inputs include those to restart the sequence (typically called reset), to pause or continue the sequence (hold or enable), or change the order of the values generated (e.g. up/down).

Exercise 2: Show the state transition diagram and table for a 2-bit counter with reset, enable, and down inputs. Reset should have priority. Write the Verilog.

Timers and Clock Dividers

It takes N clock periods for a counter to count down[2] from $N-1$ to 0. If the clock period is T seconds then the time taken is NT seconds. A circuit can thus create a delay of NT seconds by counting N clock cycles[3]

Exercise 3: What value of N would result in a 20 ms delay if the clock frequency is 50 MHz? How many bits are needed for this timer's register?

If the counter is reset to $N-1$ when it reaches 0 then the count values will be periodic with a period NT. If some event happens each time the count reaches a specific value (e.g. 0) then this event happens with period NT (and thus a frequency $1/NT$).

Exercise 4: Assume the timer above is reset to $N-1$ each time it reaches 0. For how long is the register value 0? What are the period and frequency of a signal that is inverted each time the count reaches 0?

[2]Timers traditionally count down from $N-1$ to 0 rather than up from 0 to $N-1$ because it's simple to determine when the count reaches 0: adding -1 does not cause a carry.

[3]The time includes clock cycles during which the counter has values $N-1$ through 0 inclusive.

State Encodings

There are three ways to represent ("encode") states:

Binary States are encoded as binary numbers. This requires the fewest number of flip-flops. It is used for state machines with many states such as counters. In the above example the binary encoding of the states would be `00`, `01`, `10` and `11`.

One-Hot Each bit of the state register represents a different state. This is often used when there are only a few states as it can simplify the logic required to compute the output for a given state or to compute the next state. In the above example a one-hot encoding might be `1000`, `0100`, `0010` and `0001`.

Output Each state is encoded as the output for that state. This eliminates the need for any logic to determine the output as a function of the state. However, this is only possible if the output is different for each state. The above example could not use an output state encoding because the output is `0` for states `00` and `01` and the output is `1` for states `10` and `11`.

Exercise 5: How many bits need to be considered to detect a specific state when a binary encoding is used? How many need to be considered if a one-hot encoding is used?

Exercise 6: If we used 8-bits of state information, how many states could be represented? What if we used 8 bits of state but used a "one-hot" encoding?