Implementing finite state machines in Verilog

Opriţoiu Flavius
flavius.opritoiu@cs.upt.ro

October 31, 2024

Introduction

Objectives:

- Describing the finite state machines in Verilog

For reading:

- Chris Fletcher: "EECS150: Finite State Machines in Verilog", [Flet08]

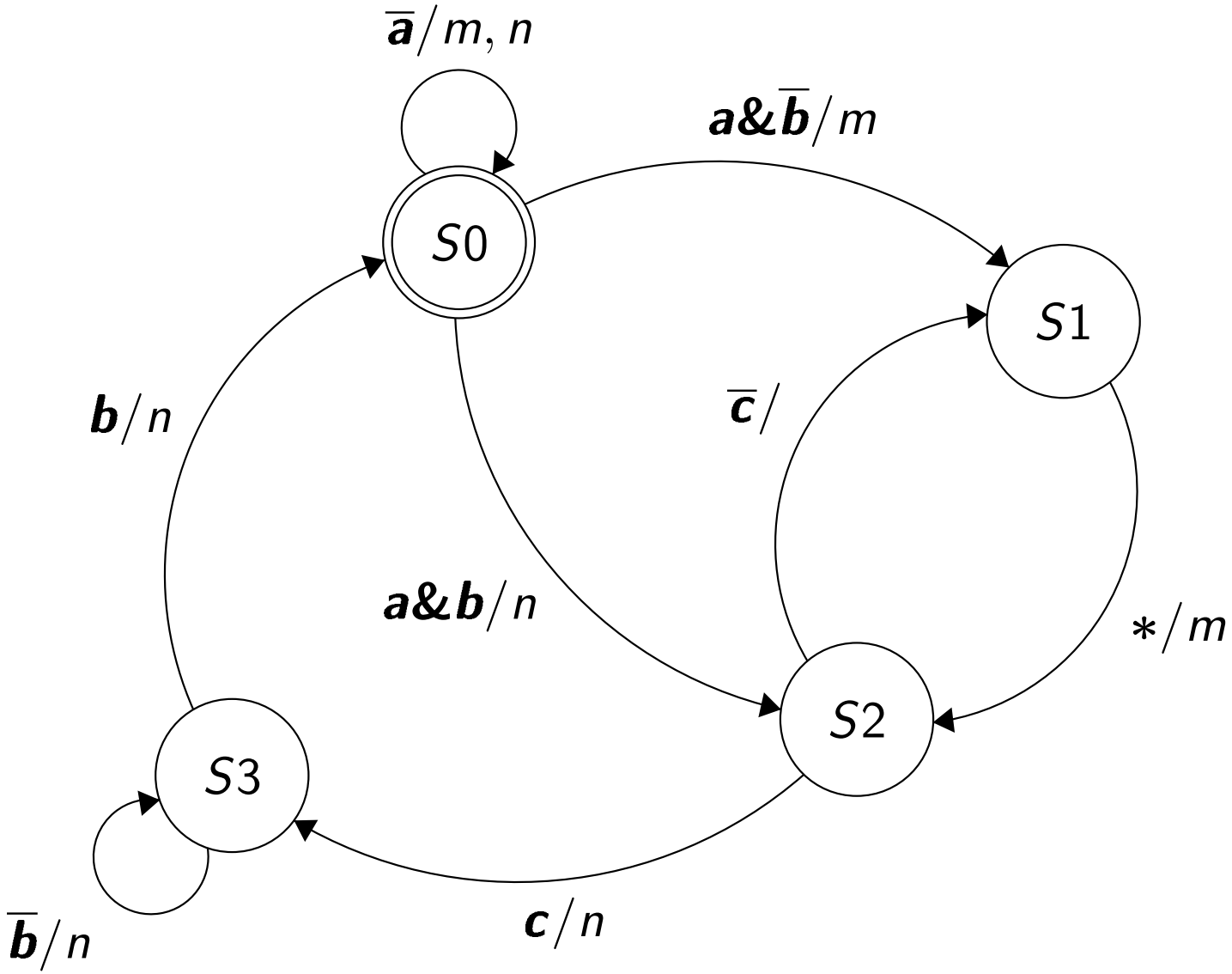
This material will present a method for behavioural description of Mealy type *finite state machines* using Verilog. The method can be readily adapted for Moore type FSM.

The Verilog behavioural description aim to use the synthesizable set of the HDL, facilitating direct implementation using the synthesis tools used for digital design.

Case study

Implementing a Mealy machine described by transition diagram

Exercise: Implement the following FSM:



Case study (contd.)

Implementing a Mealy machine described by transition diagram

By *analyzing* the transition diagram, the following observations can be made:

- the machine has 3, 1-bit inputs: *a*, *b* si *c*
- at the output, the automaton can activatre any of its 2, 1-bit outputs: *m* si *n*
- the machine can be in any of its 4 states: *S0*, *S1*, *S2* sau *S3*
- state *S0* is the initial state (activated after device's initialization)

A transition between two states of the machine is symbolized by an arc, labeled with the logic condition taht triggers the respective transition, together with the activated outputs.

The format of the arc's label is :
logic_condition/activated_output, ...

The five steps of implementing finite state machines

Step 1

Define named constants for each state of the machine. The constants are declared using `localparam` keyword. Each state constant has a distinct value, represented on the necessary number of bits (for a machine with 13 states, the state constants' values are represented on 4 bits)

For the proposed exercise, the state constants are defined as in the following code fragment:

```
1 localparam S0_ST = 2'd0;
2 localparam S1_ST = 2'd1;
3 localparam S2_ST = 2'd2;
4 localparam S3_ST = 2'd3;
```

Recommendation: The name of the four state constants should include the `_ST` suffix.

The five steps of implementing finite state machines

Step 2

Define the current state and the next state signals. The current state signal, *st*, is assigned values in an `always` block, thus being defined with the `reg` specifier, on the same number of bits as the values of the state constants.

The next state signal, *st_nxt*, has the same number of bits as *st*, and is generated by a combinational logic, dependent on the current state, *st*, and the finite state machine's inputs. However, if *st_nxt* is assigned values in an `always` block, it will be declared with `reg` specifier.

For the current exercise, the two signals are defined as follows:

```
1 reg [1:0] st;
2 reg [1:0] st_nxt;
```

The five steps of implementing finite state machines

Step 3

The next state is generated based on the transition diagram, in an combinational `always` block. One can use the `case(st)` instruction for handling the transitions associated with each state:

```
1  always @ (*)
2          case(st)
3                  S0_ST: if (!a) st_nxt = S0_ST;
4                                 else if (b) st_nxt = S2_ST;
5                                 else st_nxt = S1_ST;
6                  S1_ST: st_nxt = S2_ST;
7                  S2_ST: if (c) st_nxt = S3_ST;
8                                 else st_nxt = S1_ST;
9                  S3_ST: if (b) st_nxt = S0_ST;
10                                else st_nxt = S3_ST;
11         endcase
```

Important: The implementation must assure that all possible input configurations, for each state, are covered by the `if ... else` instructions (no input configuration is left without being handled by the Verilog code).

The five steps of implementing finite state machines

Step 4

The output of the finite state machine are constructed from the transition diagram, using a combinational `always` block. using the `case(st)` instruction, the outputs associated with each state transition are activated:

```
1  always @ (*) begin
2           m = 1'd0;
3           n = 1'd0;
4           case(st)
5                   S0_ST: if (!a) {m, n} = 2'b11;
6                                  else if (b) n = 1'd1;
7                                  else m = 1'd1;
8                   S1_ST: m = 1'd1;
9                   S2_ST: if (c) n = 1'd1;
10                  S3_ST: n = 1'd1;
11          endcase end
```

Important: For avoiding setting outputs to their default values on branches (such as on the *else* branch of instruction in line 9), all outputs are first initialized to their default value (lines 2 and 3).

The five steps of implementing finite state machines

Step 5

The current state is updated in a sequential `always` block. At each triggering edge of the clock, the current state takes the value of the next state signal. If the finite state machine has an asynchronous reset line, it is to be checked for activation.

With small adaptations, the code fragment bellow can be used for updating the current state for any finite state machine:

```
1  always @ (posedge clk, negedge rst_b)
2    if (!rst_b)    st <= S0_ST;
3    else           st <= st_nxt;
```

Complete Verilog implementation

```
1   module fsm (
2           input clk, rst_b,
3           input a, b, c,
4           output reg m, n
5   );
6           localparam S0_ST = 2'd0;
7           localparam S1_ST = 2'd1;
8           localparam S2_ST = 2'd2;
9           localparam S3_ST = 2'd3;
10          reg [1:0] st;
11          reg [1:0] st_nxt;
12          always @ (*)
13                  case(st)
14                          S0_ST:  if (!a) st_nxt = S0_ST;
15                                  else if (b) st_nxt = S2_ST;
16                                  else st_nxt = S1_ST;
17                          S1_ST:  st_nxt = S2_ST;
18                          S2_ST:  if (c) st_nxt = S3_ST;
19                                  else st_nxt = S1_ST;
20                          S3_ST:  if (b) st_nxt = S0_ST;
21                                  else st_nxt = S3_ST;
22                  endcase
23          always @ (*) begin
24                  m = 1'd0;
25                  n = 1'd0;
26                  case(st)
27                          S0_ST:  if (!a) {m, n} = 2'b11;
28                                  else if (b) n = 1'd1;
29                                  else m = 1'd1;
30                          S1_ST:  m = 1'd1;
31                          S2_ST:  if (c) n = 1'd1;
32                          S3_ST:  n = 1'd1;
33                  endcase end
34          always @ (posedge clk, negedge rst_b)
35                  if (! rst_b) st <= S0_ST;
36                  else st <= st_nxt;
37  endmodule
```

References

[Flet08] C. Fletcher. EECS150: Finite State Machines in Verilog. [Online]. Available: http://inst.eecs.berkeley.edu/~cs150/fa08/Documents/FSM.pdf (Last accessed 25/04/2016).