W6. Mealy State Machines

Designing Finite State Machines in Verilog

1. Laboratory Objective

Implementing FSMs in five steps using Verilog language

2. Theoretical Background

2.1 The state transition diagram of the FSM

In computational theory, the Mealy machine is an FSM (Finite State Machine) whose output values are determined by both the current state and the current inputs (in contrast to the Moore machine whose output values are determined by the current state). The Mealy machine is thus a deterministic FSM.

Figure 1 shows a Mealy type FSM that has the following internal states:

- S0 - default state
- S1 - unidirectional transition state
- S2 - bidirectional transition state

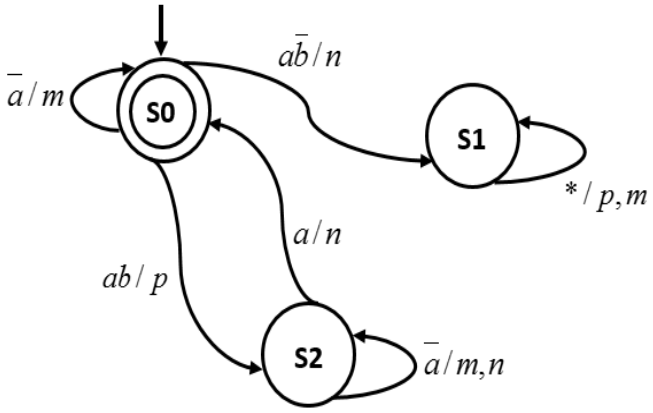

Fig.1 – State Transition Diagram Example

The interpretation of the expressions on the transition arcs can be summarized as follows:

logic condition / ***output activated***.

2.2 Verilog design of Finite State Machines in 5 Steps

2.2.1 Construction of named constants for each state of the machine

The first stage in the implementation of FSM consists of declaring the constants using localparam and each is assigned a distinct value, on the corresponding number of bits. After a brief analysis of the FSM presented (see Fig. 1) we can easily notice that the internal states of the machine, marked from 0 to 2 can be encoded on a 2-bit number. Accordingly, we will write:

```
localparam S0 = 2'd0; // will be stored as bits 00

localparam S1 = 2'd1; // will be stored as bits 01

localparam S2 = 2'd2; // will be stored as bits 10
```

2.2.2 Defining register values

Defining 2 **reg** signals that will keep the current state (***st***) and the next state (***st_next***). The two signals are declared on the same number of bits as the state constants.

```
reg [1:0] st;

reg [1:0] st_next;
```

2.2.3 Construction of the next state in an always combinational block

From the state transition diagram, construct the next state, st_next, in an ***always combinational block*** (please read more in the Verilog modeling using always and initial blocks – Flavius Opriţoiu).

Using the case (***st***) instruction, branches are provided for each state, in order to evaluate the logical conditions of transition to another state.

Important: Code branches without ***st*** assignments are removed (eg unused ***else*** branches)

```
always @ (*) begin

  st_next = S0; // the initial state is the default state

  case (st)
```

```
    S0: if (!a)
        st_next = S0;
        else
        if (!b)
        st_next = S1;
    S1: // will be completed with code lines
    S2: // will be completed with code lines
    endcase
end
```

2.2.4 Construction of machine outputs in an always combinational block

From the diagram of state transitions, the machine's outputs are built in an always combinational block. Similar to generating the next state, using the case (***st***) instruction, branches are provided for each state, in which the outputs corresponding to each transition are activated.

Important: To avoid unallocated code branches of some or all machine outputs, all outputs will be initialized to their default values prior to the case (***st***) instruction.

```
always @ (*) begin
    m = 1'd0 ; // active on the logical value 1
    n = 1'd1 ; // active on the logical value 0
    p = 1'd0 ; // active on the logical value 1
    case (st)
```

```
S0: if (!a)
     m = 1'd1 ; // will be activated on the value 1
    else
    if (!b)
     n = 1'd0 ; // will be activated on the value 0
S1: begin
     m = 1'd1 ;
     p = 1'd1 ;
    end
S2: // will be completed with code lines
```

2.2.5 Updating the current state in a sequential always block

In the last step of designing the FSMs, the current state will be updated in an always sequential block. With minor modifications, the code below can be used to implement any machine with finite states, provided that the previous steps are followed.

```
always @ (posedge clk, negedge rst_b)  begin
  if (!rst_b)
    st <= INIT_ST ; // in our case the initial state is SO
  else
    st <= st_next ;
end
```

3. References

[Vlad12] M. Vlăduțiu, *Computer Arithmetic: Algorithms and Hardware Implementations*, 2012th ed. Springer, 2012.

[Opri17] F. Oprițoiu, *Interface of digital components for data transfer*, Lab. 2017.