One Hot encoding for FSMs

Opriţoiu Flavius
flavius.opritoiu@cs.upt.ro

December 3, 2023

Introduction

Objectives:

- Construct Finite State Machines (FSMs) using the One Hot encoding

For reading:

- Mircea Vlăduțiu: "Computer Arithmetic : Algorithms and Hardware Implementations", Appendix B [Vlad12]

The *One Hot encoding* for a FSM with s states uses s storage elements. Each storage element is associated with one state. In consequence, at any given moment, one and only one of the s storage elements is active (has active output).

The One Hot encoding implementation, although using more storage elements than the State Table method, has the advantage of a straightforward design and debug.

Frequency divisor

A frequency divisor having a division factor of n receives at input a clock signal with frequency f_{in} and generates at output a signal with frequency $\frac{f_{in}}{n}$. The generate signal ought not have a 50% duty cycle (the signal is active half the period and inactive the other half).

If n is of form 2^k, a k-bit binary counter is used, the divided clock signal being the Most Significant Bit (MSB) of the counter's output. If n is not a power of 2, a modulo-n counter is used.

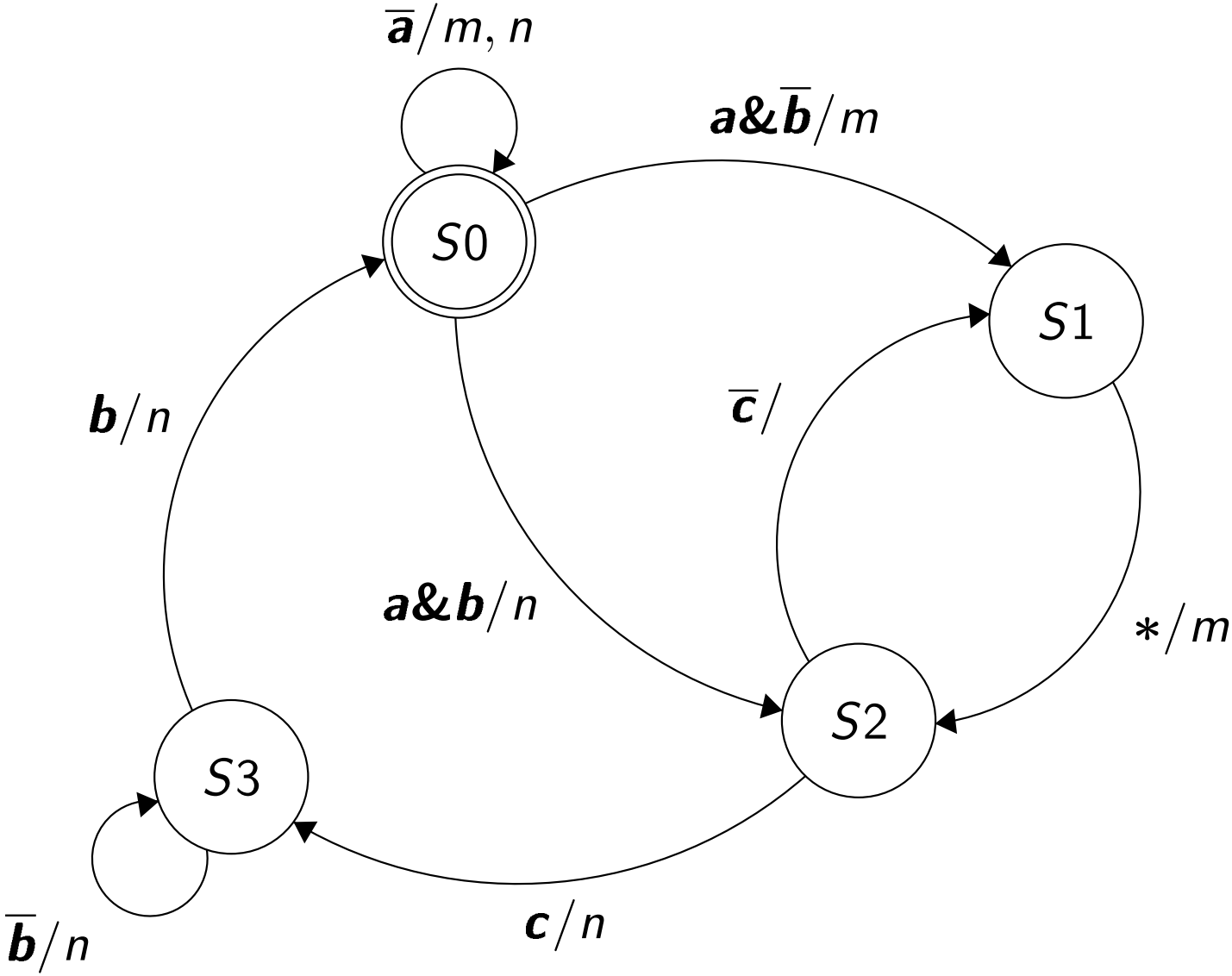
The interface of the frequency divisor:

- input *clk*: the signal having frequency f_{in}
- input *rst_b*: initialization signal, optional
- output *dclk* (divided clock): the signal with frequency $\frac{f_{in}}{n}$

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

The design uses 4 D type flip-flops: FF_0, FF_1, FF_2 and FF_3, with inputs D_i and outputs Q_i, associated to the 4 states S_0, S_1, S_2 and S_3.

At every moment one and only one of the 4 flip-flops is active, having output Q_i set to 1. The current state is indicated by the flip-flop with the active output. If Q_1 is active, the current state is S_1, if Q_4 is 1 then S_4 is the current state, etc.

At input D_i is connected the boolean equation activating state S_i. FSM's design reduces to writing these equations. The next state will be S_1 in the following cases:

- the current state is S_0, a is 1 and b is 0, **or**
- the current state is S_2 and c is 0

Thus $D_1 = Q_0 \cdot a \cdot \overline{b} \ + \ Q_2 cdot \overline{c}$

FSMs' implementation using the One Hot encoding

Step 1

For each state of the FSM, define a state constant, S_i, using `localparam`. Each state constant will have a distinct value, between 0 and $s - 1$ (s being the total number of states).

For the proposed exercise, these constants can be defined like bellow:

```
1 localparam S0 = 2;
2 localparam S1 = 0;
3 localparam S2 = 3;
4 localparam S3 = 1;
```

FSMs' implementation using the One Hot encoding

Step 2

Define the current state, *st*, and the next state *st_nxt* as 2 binary vectors on *s* bits (*s* being the FSM's number of states). Signal *st* will be declared of `reg` type whereas *st_nxt* as `wire` type.

For the proposed exercise, the two signals are defined as bellow:

```
1 reg [3:0] st;
2 wire [3:0] st_nxt;
```

FSMs' implementation using the One Hot encoding

Step 3

Assign to bit *st_nxt[S_i]* (S_i being one of the state constants defined in step 1) the boolean expression activating the state S_i. It is worth noting that these boolean equations are constructed by the model presented in slide 5, with the specification that signals D_i are replaced with *st_nxt[S_i]* and signals Q_i are replaced with *st[S_i]*.

For the proposed exercise, the next state generation becomes:

```
1  assign st_nxt[S0] = ( st[S0] & (~a) ) |
2                      ( st[S3] & b );
3  assign st_nxt[S1] = ( st[S0] & a & (~b) ) |
4                      ( st[S2] & (~c) );
5  assign st_nxt[S2] = st[S1] |
6                      ( st[S0] & a & b );
7  assign st_nxt[S3] = ( st[S2] & c ) |
8                      ( st[S3] & (~b) );
```

FSMs' implementation using the One Hot encoding

Step 4

Assign to each FSM output the boolean expression constructed directly from the state transition diagram, by enumerating using the OR logic operator all conditions on which the respective output is active. Example: output *m* is active:

- in state S_0, if *a* is 0, **or**
- in state S_0, if *a* is 1 and *b* is 0, **or**
- in state S_1

For the proposed exercise, the outputs are generated like bellow:

```
1  assign m = ( st[S0] & (~a) ) |
2             ( st[S0] & a & (~b) ) |
3             st[S1];
4  assign n = ( st[S0] & (~a) ) |
5             ( st[S0] & a & b ) |
6             ( st[S2] & c ) |
7             ( st[S3] & (~b) ) |
8             ( st[S3] & b );
```

FSMs' implementation using the One Hot encoding

Step 5

Update the current state in a sequential `always` block. At each triggering edge of the clock, the next state signal becomes the current state. Activation of the reset input brings the FSM in the initial state, which, for the proposed exercise is state S_0.

For the proposed exercise, the current state update is performed as in the following code:

```
1  always @ (posedge clk, negedge rst_b)
2    if (rst_b == 0) begin
3      st <= 0;
4      st[S0] <= 1;
5    end else
6      st <= st_nxt;
```

References

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