Design of an FSM: an up/down counter

Madhav P. Desai

March 1, 2017

Suppose we are given the following building blocks:

- Inverters, NAND2, NOR2 gates each with a delay of 2 units.
- D-flipflops with $Clock \rightarrow Q$ delay of 4 units, set-up time of 2 units and hold-time of 2 units.

We illustrate the design process of obtaining a logic circuit implementation of a Mealy FSM starting with an abstract specification.

1 The FSM Specification

- The set of input symbols is $\Sigma = \{reset,\ up,\ down\}$.
- The set of output symbols is $\Sigma = \{Y, N\}$.
- The set of states is $\Sigma = \{A, B, C\}$.
- The initial state is A.

The next state function and output functions are as follows

x(k)	q(k)	q(k+1)	y(k)
reset	_	A	N
up	A	B	N
down	A	C	N
up	B	C	N
down	B	A	Y
up	C	A	Y
down	C	B	N

Note that the input symbol *reset* is used to put the machine in the initial state A.

The specification can be visualized by the state transition graph (STG) shown in Figure 1.

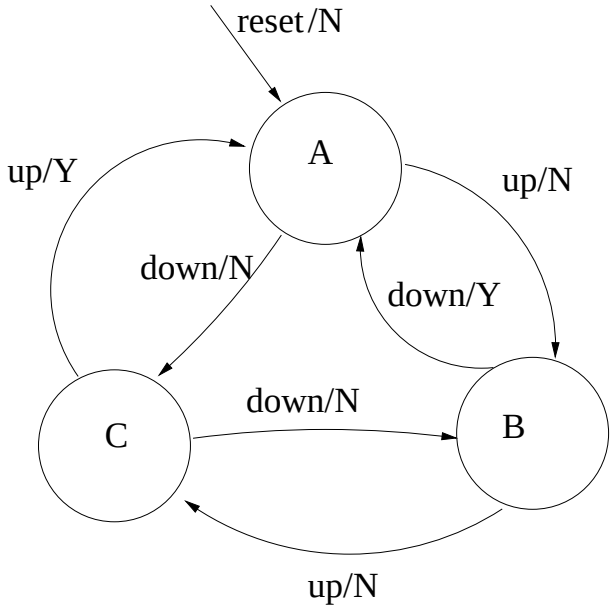

Figure 1: State transition graph of the up/down counter

2 Input encoding and combinational function implementation

First, we need to encode the input, output and state symbols. Let us use the binary encoding:

```
State    q1 q0
A        0  0
B        0  1
C        1  0

Input    r  x
reset    1  _
up       0  1
down     0  0

Output   w
N        0
Y        1
```

Thus, the output bit w has the truth-table

```
          00  01  11  10     q1q0
 00            1   d
```

```
01              d   1
11              d
10              d
rx
```

which has the simplest formula $q1.\overline{r}.x + q_0.\overline{r}.\overline{x}$.

The next state variable $nq1$ has the truth-table

```
nq1
     00  01  11  10  q1q0
 00   1       d
 01       1   d
 11           d
 10           d
 rx
```

so that $nq1 = \overline{r}.\overline{x}.\overline{q1}.\overline{q0} \ + \ q0.\overline{r}.x$.

The next state variable $nq0$ has the truth-table

```
nq0
     00  01  11  10  q1q0
 00           d    1
 01   1       d
 11           d
 10           d
 rx
```

so that $nq0 = \overline{r}.x.\overline{q1}.\overline{q0} \ + \ q1.\overline{r}.\overline{x}$.

To implement these three equations, we can obtain the following simplified set of formulas (we have introduced intermediates u, v, w):

$$\begin{aligned} u &= \overline{q1}.\overline{q0} \\ v &= \overline{r}.x \\ p &= \overline{r}.\overline{x} \\ w &= v.q1 + p.q_0 \\ nq1 &= p.u + v.q0 \\ nq0 &= v.u + w.q1 \end{aligned}$$

Implement these using our library of gates to get the logic network in Figure 2. In this logic network, even if we keep all gates at minimum size, we will get a reasonable solution (because the total effort of each path is quite small).

3 The final timing analysis

Assuming that all gates are sized to be 1-unit, the hold-time is not a problem (because the flip-flop delay is itself equal to the hold time).

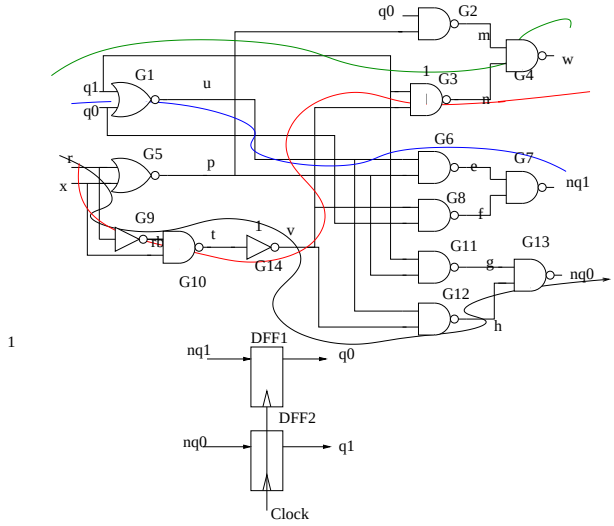

Figure 2: Logic Network

- The maximum delay from a circuit input to a flip-flop input (for the critical path shown in black) is 10 units, the blue path.
- The maximum delay from launch flip-flop to capture flip-flop (from *clock* to *nq*) is 10 units (including the flip-flop delay) shown by the blue path.
- The maximum delay from clock to the output is 8 units (including the flip-flop delay) shown by the green path..
- The maximum delay from circuit input to circuit output is 10 units shown by the red path.

4 VHDL Descriptions

The implemented FSM can be easily described as VHDL (using the library of gates):

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.EE224_Components.all;

entity UpDownCounter is
  port(r,x: in std_ulogic; w: out std_ulogic; clk: in std_ulogic);
end entity;
architecture LogicNetwork of UpDownCounter is
  signal q1, q0, nq1, nq0: std_ulogic;
```

```
  signal u,v,m,n,p,t,rb,e,f,g,h: std_ulogic;
begin

  -- note: positional port map
  --       which is not a good idea but is very useful
  --       if you are sure of what you are doing and
  --       feeling a bit lazy.
  G1: nor2 port map (q1,q0,u);
  G2: nand2 port map (q0,p,m);
  G3: nand2 port map (q1, v, n);
  G4: nand2 port map (m,n,w);

  G5: nor2 port map (r,x,p);
  G6: nand2 port map (u,p,e);
  G7: nand2 port map (e,f,nq1);
  G8: nand2 port map (q0,v,f);

  G9: inverter port map (r,rb);
  G10: nand2 port map (rb,x,t);
  G11: nand2 port map (q1,p,g);
  G12: nand2 port map (u,v,h);

  G13: nand2 port map (g,h,nq0);
  G14: inverter port map (t,v);

  -- the DFF's
  d1: DFF port map (d => nq1, clk => clk, q => q1);
  d0: DFF port map (d => nq0, clk => clk, q => q0);
end LogicNetwork;
```