

5.1 Negative True and VHDL

Friday, February 16, 2001

- Signals in VHDL are inherently positive true. A signal is high (a one) when it "happens".
- A negative true signal is low (a zero) when it "happens".
- It is nice to be able to recognize whether a signal is negative or positive true from a clue provided by the signal name.
- Good names to use for the negative true signal, foo, are:
 - /foo
 - nfoo
 - n_foo
 - not_foo
 - neg_true_foo

5.2 More on Negative True

- /foo is out as a VHDL identifier cannot begin with a slash.
- nfoo could be troublesome for signal names beginning with the letter n.
- not_foo and neg_true_foo are verbose.

So, we will choose n_foo to mean negative true.

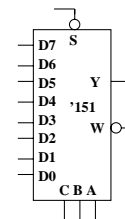
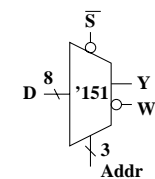
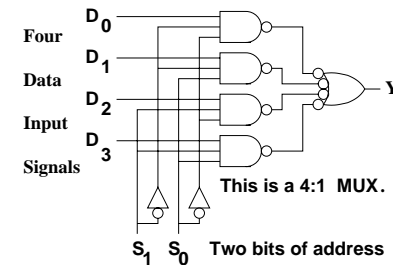
By convention, we will prepend n_ to all signal names when the signal is negative true. That is, when a signal is low (a zero) when it "happens".

5.3 Negative or Positive True?

```
library ieee;
use ieee.std_logic_1164.all;
entity neg is
  port (a1, b1, a2, b2, a3, b3 : in std_logic;
        x, n_y, n_z : out std_logic);
end neg;
architecture equations of neg is
  signal y : std_logic;
begin
  -- The only clue we have are the signal names.
  -- The next two are positive true.
  x <= a1 AND b1;
  y <= a2 AND b2;
  -- The next two are negative true.
  n_y <= not y;
  n_z <= not (a3 OR b3);
end equations;
```

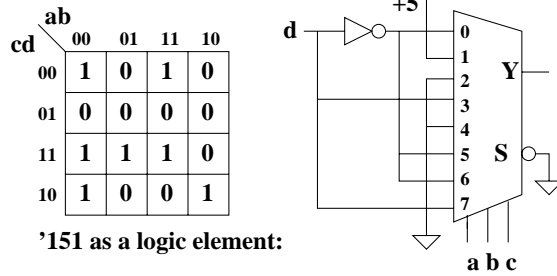
5.4 Digital Building Blocks - MUX

A Multiplexer selects one signal according to an address.



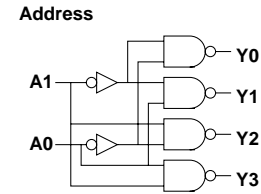
5.5 Using a MUX as Logic

- Muxes are usually used to select a source of signal.
- But can be used for making logical functions.



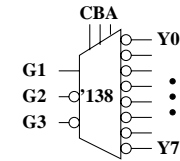
5.6 Demultiplexer

Demultiplexer or Selector is the inverse of the Multiplexer. It selects the addressed line.

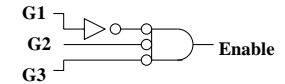


One of these lines is selected (pulled low in this case).

74LS138
3:8 Decoder

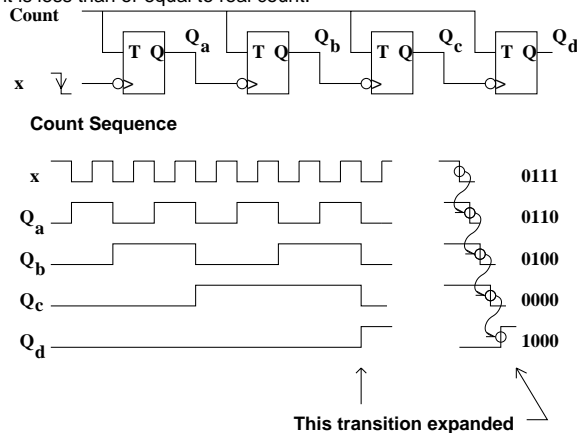


The '138 has a complex enable mechanism.



5.7 Ripple Counters

It is easy to make a counter using T flip-flops. They are usually negative edge triggered. An example is the 74LS393. The toggle rate is fastest for the least significant bit. The count can take time to settle. The apparent count is less than or equal to real count.



5.8 Synchronous Counters

Synchronous Counters: reduce ripple by setting all bits at once.

$$I = P * T$$

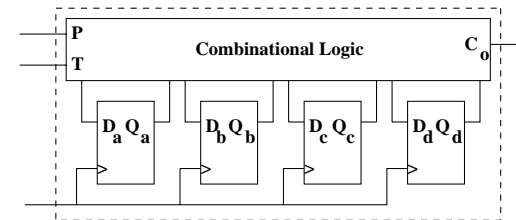
$$D_a = I * Q_a + I * /Q_a$$

$$D_b = I * Q_b + I * Q_a * /Q_b + /Q_a * Q_b$$

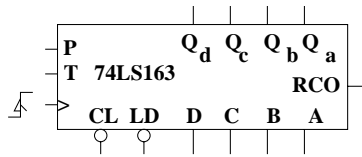
$$D_c = I * Q_c + I * Q_a * Q_b * /Q_c + /Q_a * Q_c + /Q_b * Q_c$$

$$D_d = I * Q_d + I * Q_a * Q_b * Q_c * /Q_d + Q_d * /Q_a + Q_d * /Q_b + Q_d * /Q_c$$

$$R_{co} = T * Q_a * Q_b * Q_c * Q_d$$

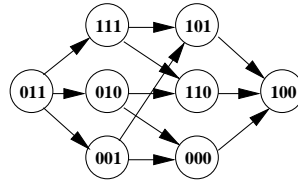


5.9 Glitch on RCO



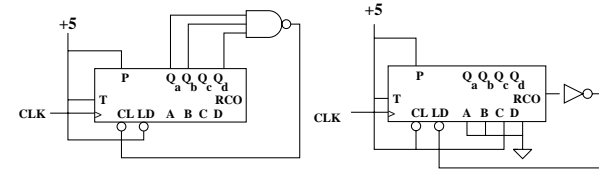
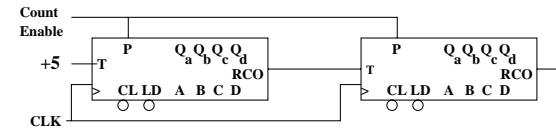
74LS163 has clear and load functions too.
 CL and LD are synchronous
 if CL then $Q := 0$
 else if LD then $Q := 1$
 else if $P * T = 1$ then $Q := Q + 1$
 else $Q := Q$

Care is required of the
 Ripple Carry Output:
 It can have glitches:
 Any of these transition
 paths are possible!



5.10 Cascading / Specific Counts

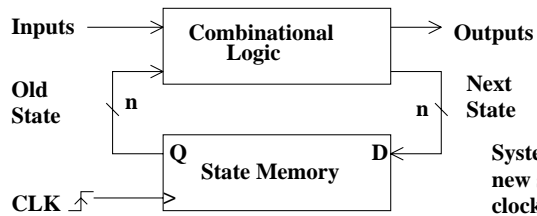
To cascade synchronous counters (to count more bits):
 P is "count Enable".
 RCO and T are daisy chained.



This one counts 0.1.2. 11. 0. 1 ... This one counts 4. 5. 15. 4. 5...

5.11 Finite State Machines

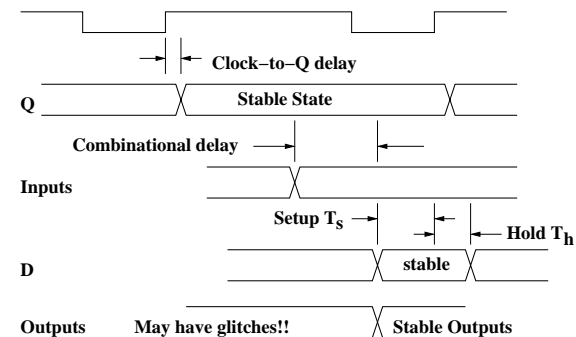
Finite State Machines are Clocked Sequential Systems.



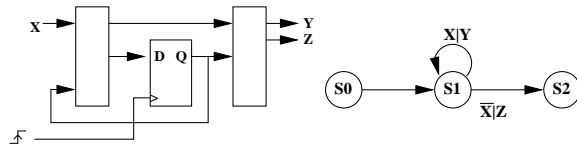
System assumes the
 new state after
 the clock edge.

5.12 Timing of an FSM

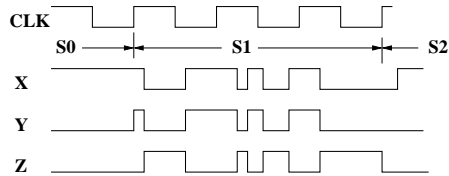
Clock speed is limited by FF delay, Combinational delay, and setup time.
 New state is determined by inputs and old state just BEFORE clock edge.



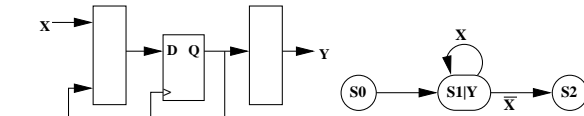
5.13 Mealy Model



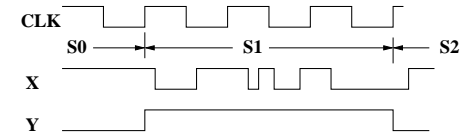
"Mealy Model": Output = F(State, Input) Arcs between states also note output.



5.14 Moore Model

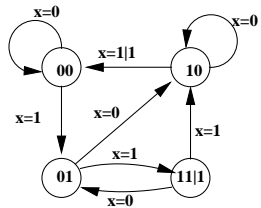


"Moore Model": Output = F(State) Arcs note transitions only. State names describe output.



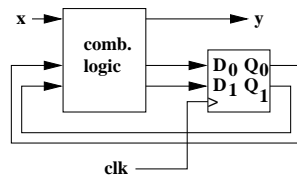
5.15 Simple FSM

- One input. One output.
- State names are numbers - Q_0 Q_1 .



Next State and Output

Q_1	Q_0	x	D_1	D_0	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	1	0	1



5.16 FSM Logic

Next State and Output Determination: Logic Specification

Q_1	Q_0	x	D_1	D_0	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	1	0	1

Q_1	Q_0	x	D_1	D_0
00	0	0	0	0
01	1	0	1	0
11	0	1	0	0
10	1	1	0	0

$$D_1 = x*Q_0 + Q_1*Q_0 + x*Q_1*/Q_0$$

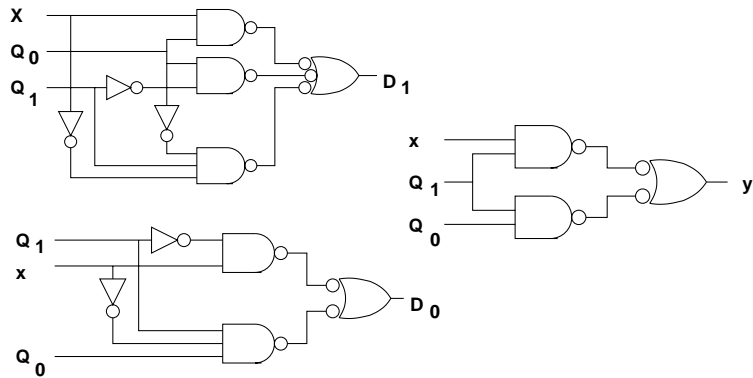
Q_1	Q_0	x	y
00	0	0	0
01	0	0	0
11	1	0	1
10	0	1	1

$$y = x*Q_1 + Q_1*Q_0$$

Q_1	Q_0	x	D_0
00	0	0	1
01	0	1	1
11	1	0	0
10	0	0	0

$$D_0 = x*/Q_1 + x*Q_0*Q_1$$

5.17 FSM Combinational Logic

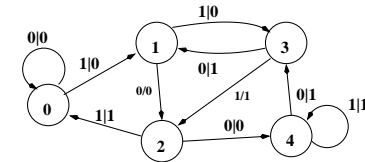


5.18 Another FSM: Divide by Five

Divide by 5 FSM

$$\begin{array}{r} 0010011 \\ 101 \overline{) 1011111} \\ \underline{101} \\ 000 \\ \underline{000} \\ 000 \\ \underline{000} \\ 000 \\ \underline{000} \\ 000 \end{array}$$

State of the FSM is the remainder of the division operation.
Binary division: shift input bit stream left
 $NS = (2 * PS + input) \bmod 5$
Output = 1 if $(2 * PS + input) \geq 5$



One can (if you wish) derive these equations from the truth table assuming the extra states result in "don't cares".

$$D_2 = /Q_2 * x + /Q_2 * Q_0 + /Q_1 * /x$$

$$D_1 = Q_0 * x + /Q_1 * Q_0 + Q_2 * /x$$

$$D_0 = Q_2 * /x + Q_1 * Q_0 * /x + /Q_2 * /Q_1 * x$$

$$y = Q_2 + Q_1 * Q_0 + Q_1 * x$$

Q_2	Q_1	Q_0	x	D_2	D_1	D_0	y
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0
0	0	1	1	0	1	1	0
0	1	0	0	1	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	1
1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	1

5.19 Extract of Lab 1

```
library ieee;
use ieee.std_logic_1164.all;
entity ff is
  port (ffclk, din, tin, jin, kin : in std_logic;
        dff, tff, jkff : out std_logic);
end ff;

architecture behavioral of ff is
  -- This declares two signals so we don't
  -- use tff and jkff as inputs.
  signal insidetff, insidejkff : std_logic;
begin
  process (ffclk) begin
    if rising_edge(ffclk) then
      dff <= din;
      insidetff <= tin xor insidetff;
      insidejkff <= (jin and (not insidejkff)) or
                    ((not kin) and insidejkff);

    end if;
  end process;
  tff <= insidetff;
  jkff <= insidejkff;
end behavioral;
```

5.20 Simulation of the Extract

Notice that the dff is a sample and hold.

Both the dff and tff are synchronous.

Does this test all possible transitions of the jkff?



5.21 A Simple Counter

```
library ieee; use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ctr is
  port(clk, n_clr, enb : in std_logic;
        cnt : out std_logic_vector(1 downto 0);
        rco : out std_logic);
end ctr;
architecture behavioral of ctr is
  signal intcnt : std_logic_vector(1 downto 0);
begin
  clocked: process (clk)
  begin
    if rising_edge(clk) then
      if n_clr = '0' then
        intcnt <= "00";
      elsif (enb = '1') then
        intcnt <= intcnt + 1;
      end if;
    end if;
  end process clocked;
  rco <= '1' when (intcnt = "11") else '0';
  cnt <= intcnt;
end behavioral;
```

5.23 Simulation of Simple Counter

- Notice that the counter doesn't count when not enabled.
- Also that n_clr is synchronous, that is the counter doesn't clear unless n_clr is low when the clock has a rising edge.
- What change to the VHDL would make n_clr asynchronous?

