

L5.1 Negative True and VHDL Fri. February 15, 2002

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

L5.2 More on Negative True Fri. February 15, 2002

/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.

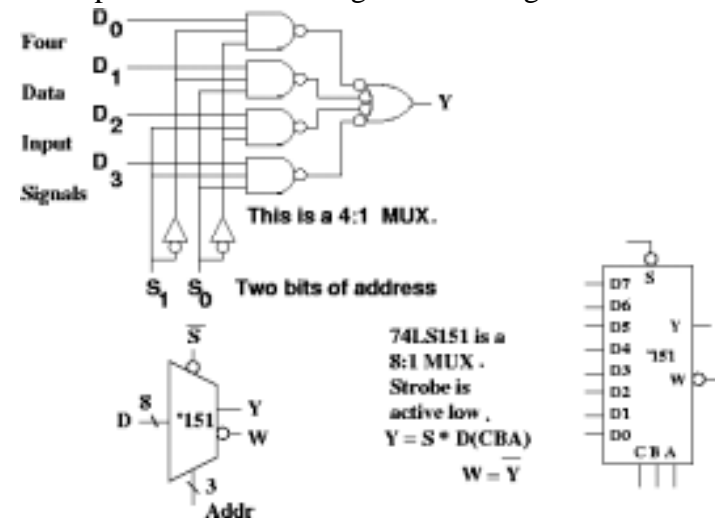
By convention, we will prepend n\_ to all signal names when the signal is negative true, that is, when a signal is low (a zero) when it "happens".

L5.3 Negative or Positive True? Fri. February 15, 2002

```
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;
```

L5.4 Digital Building Blocks – MUX Fri. February 15, 2002

A multiplexer selects one signal according to an address.

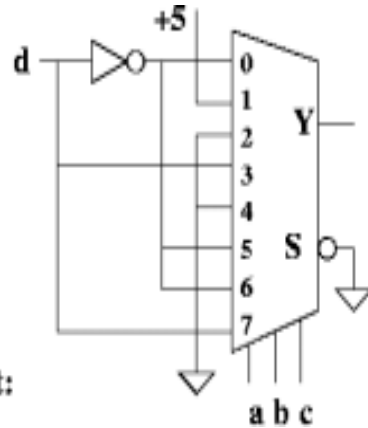


L5.5 Using a MUX as Logic Fri. February 15, 2002

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

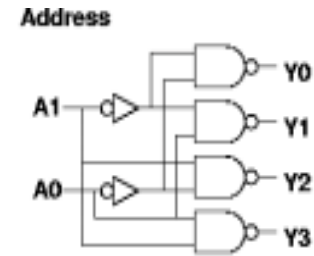
cd \ ab	00	01	11	10
00	1	0	1	0
01	0	0	0	0
11	1	1	1	0
10	1	0	0	1

'151 as a logic element:



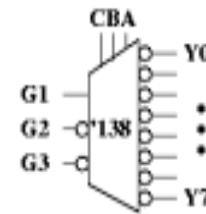
L5.6 Demultiplexer 2002 Fri. February 15,

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

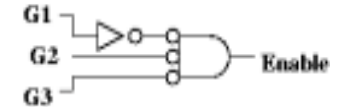


One of these lines is selected (pulled low in this case).

74LS138 3:8 Decoder

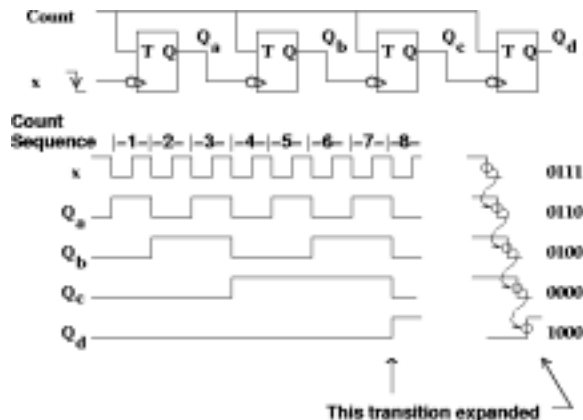


The '138 has a complex enable mechanism.



L5.7 Ripple Counters Fri. February 15, 2002

It is easy to make a counter using T flip-flops.  
 T flip-flops are usually negative edge triggered. An example is the 74LS393.  
 The toggle rate is fastest for the least significant bit.  
 The time for the count to settle depends both on the counter length and the particular count.  
 The apparent count is always less than or equal to the final count.



L5.8 Synchronous Counters Fri. February 15, 2002

Synchronous counters use more logic to set or clear all the bits on a single clock edge, eliminating the ripple.

$$I = P \cdot T$$

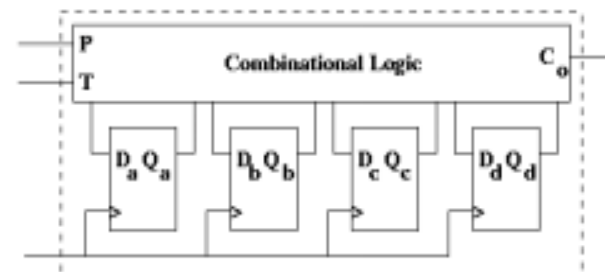
$$D_a = \bar{I} \cdot Q_a + I \cdot \bar{Q}_a$$

$$D_b = \bar{I} \cdot Q_b + I \cdot \bar{Q}_a \cdot \bar{Q}_b + \bar{Q}_a \cdot Q_b$$

$$D_c = \bar{I} \cdot Q_c + I \cdot \bar{Q}_a \cdot \bar{Q}_b \cdot \bar{Q}_c + \bar{Q}_a \cdot Q_c + \bar{Q}_b \cdot Q_c$$

$$D_d = \bar{I} \cdot Q_d + I \cdot \bar{Q}_a \cdot \bar{Q}_b \cdot \bar{Q}_c \cdot \bar{Q}_d + \bar{Q}_d \cdot \bar{Q}_a + \bar{Q}_d \cdot \bar{Q}_b + \bar{Q}_d \cdot \bar{Q}_c$$

$$R_{co} = T \cdot \bar{Q}_a \cdot \bar{Q}_b \cdot \bar{Q}_c \cdot \bar{Q}_d$$

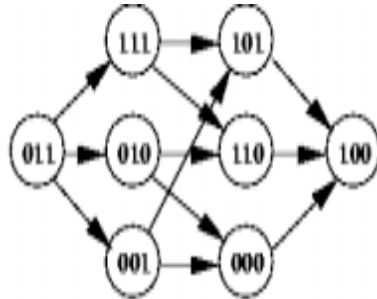


L5.9 Glitch on RCO Fri. February 15, 2002

Note that, while all bits of a synchronous counter are set on the same clock edge, they may not be set at EXACTLY the same time.

This means that there is a rapidly changing state of the counter. If it passes through all ones it will cause a 'glitch' on the ripple carry out. You are asked to look for this in Lab 1, but you may not see it!

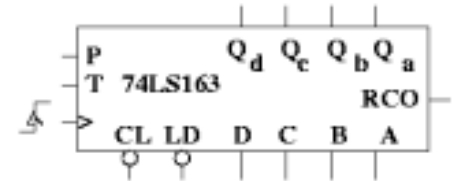
Care is required of the Ripple Carry Output. It can have glitches. Any of these transition paths are possible!



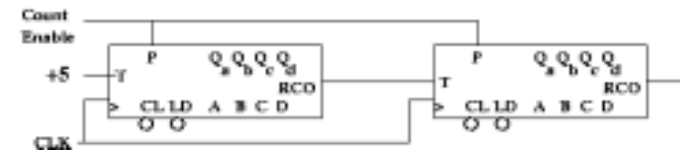
L5.10 Cascading Fri. February 15, 2002

The '163 will count ONLY if the P and T inputs are high. Note that the RCO is the AND of all four bits and T.

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

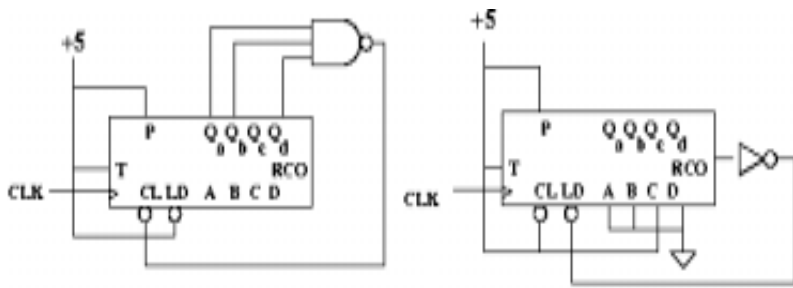


To cascade synchronous counters (to count more bits):  
 P is "count Enable".  
 RCO and T are daisy chained.



L5.11 Specific Counts Fri. February 15, 2002

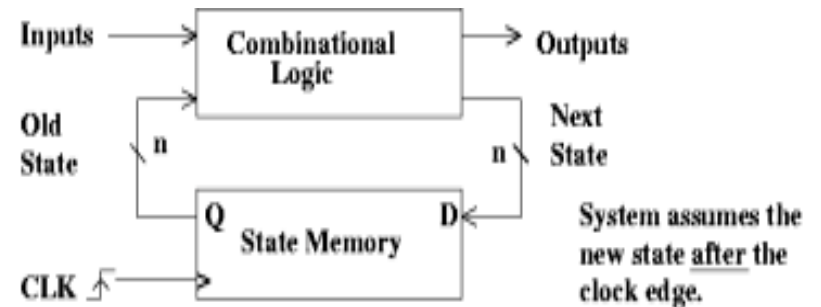
With a little ingenuity, you can achieve all kinds of count sequences. These are both divide by twelve circuits.



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

L5.12 Finite State Machines Fri. February 15, 2002

Finite state machines are clocked sequential systems.

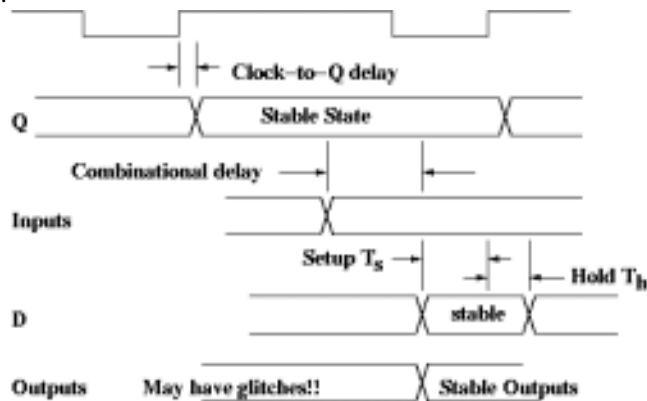


We have already seen simple FSMs in flip-flops and counters. But, you can do much more complex things with them. After a clock edge, the FSM assumes a state that depends on the state it WAS in and the inputs just before the clock edge.

L5.13 Timing of an FSM Fri. February 15, 2002

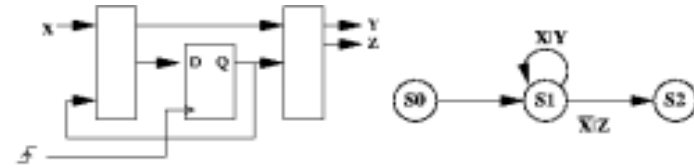
Clock Speed is limited by the flip-flop delay, combinational delay, and setup time.

The new state is determined by the inputs and the old state just BEFORE the clock edge.

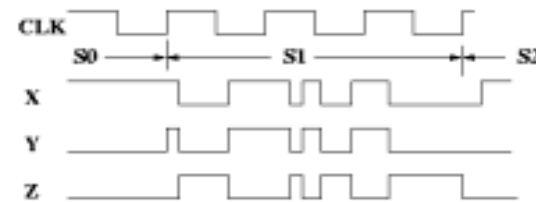


L5.14 Mealy Model Fri. February 15, 2002

With this type of FSM, the output can change asynchronously in response to changes in the input.



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

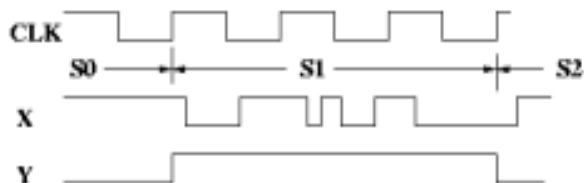


L5.15 Moore Model Fri. February 15, 2002

With this type of FSM, the output is fixed during each clock cycle and only changes after the clock edge.



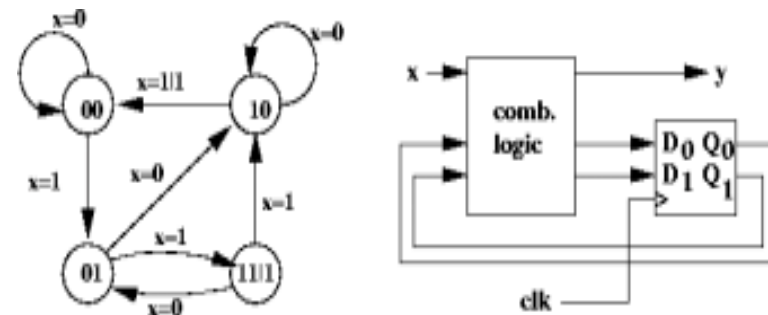
"Moore Model": Output = F(State) Arcs note transitions only. State names describe output.



L5.16 Simple FSM Fri. February 15, 2002

We have automated procedures to build the logic for finite state machines, but here is an example of a very simple machine.

This is one way of describing an FSM, namely, in terms of transitions to be made on each clock edge. The state names are numbers in the form Q1 Q0. Four possible states require two bits to encode them. There could be as much as four bits. This is a Mealy machine.

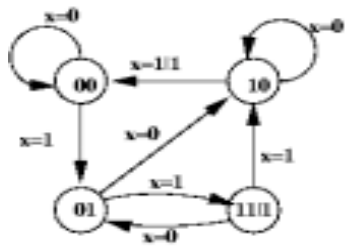


L5.17 FSM Logic Fri. February 15, 2002

It is straightforward to build a truth table for the next state based on the present state and the input.

The output is also derived from the same variables.

The equations can easily be derived, either directly from the truth table or from Kmaps.



Q <sub>1</sub>	Q <sub>0</sub>	x	D <sub>1</sub>	D <sub>0</sub>	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

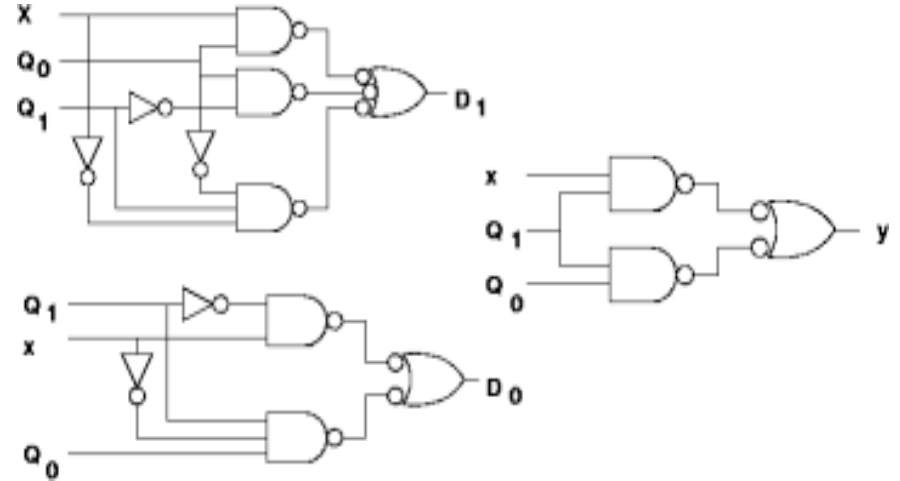
$$D_0 = x \cdot Q_1 + \bar{x} \cdot Q_0 \cdot Q_1$$

$$D_1 = x \cdot Q_0 + Q_1 \cdot Q_0 + \bar{x} \cdot Q_1 \cdot Q_0$$

$$y = x \cdot Q_1 + Q_1 \cdot Q_0$$

L5.18 FSM Combinational Logic Fri. February 15, 2002

Here is the logic that would be required to implement the FSM, if it were made out of discrete gates.



L5.19 Another FSM: Divide by Five Fri. February 15, 2002

This FSM has a single input which represents a number.

The LSB comes first, another with each clock pulse.

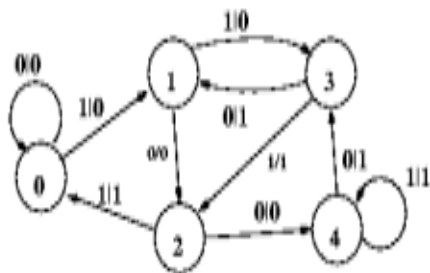
The output, also bit serial, is similar.

The state of the FSM is the remainder of the division operation.

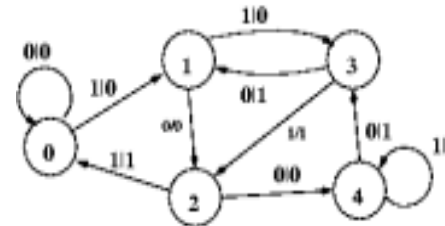
$$101 \overline{) 1011111} \quad 5 \overline{) 95}$$

$$NS = (2 * PS + input) \bmod 5$$

$$Output = 1 \text{ if } (2 * PS + input) \geq 5$$



L5.20 Divide by Five Implementation Fri. February 15, 2002



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

Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	x	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	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

$$D_2 = Q_2 \cdot x + Q_2 \cdot Q_0 + Q_1 \cdot x$$

$$D_1 = Q_0 \cdot x + Q_1 \cdot Q_0 + Q_2 \cdot x$$

$$D_0 = Q_2 \cdot x + Q_1 \cdot Q_0 \cdot x + Q_2 \cdot Q_1 \cdot x$$

$$y = Q_2 + Q_1 \cdot Q_0 + Q_1 \cdot x$$

L5.21 Extract of Lab 1 Fri. February 15, 2002

```
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;
```

L5.22 Simulation of the Extract Fri. February 15, 2002

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?

