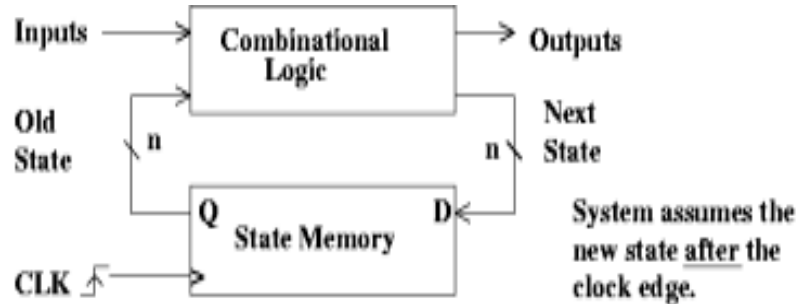


L7.1 (L5.12) Finite State Machines Wed. February 20, 2002

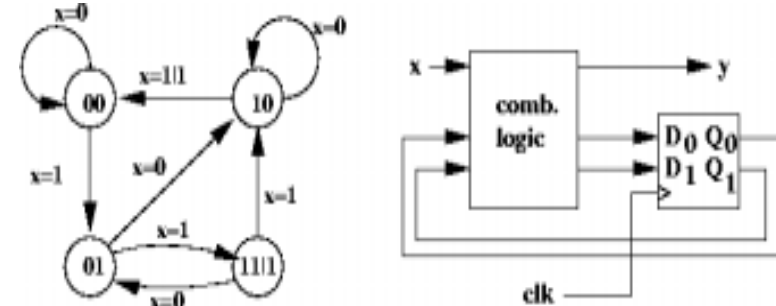
Finite state machines are clocked sequential systems.



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.

L7.2 (L5.16) Simple FSM Wed. February 20, 2002
 We have automated procedures to build the logic for finite state machines, but here is an example of a very simple machine.

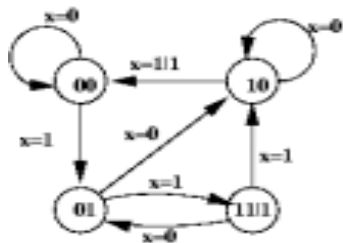
This is one way of describing an FSM, namely, in terms of transitions to be made on each clock edge. The state names are numbers in the form Q1 Q0. Four possible states require two bits to encode them. There could be as much as four bits. This is a Mealy machine.



L7.3 (L5.17) FSM Logic Wed. February 20, 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 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

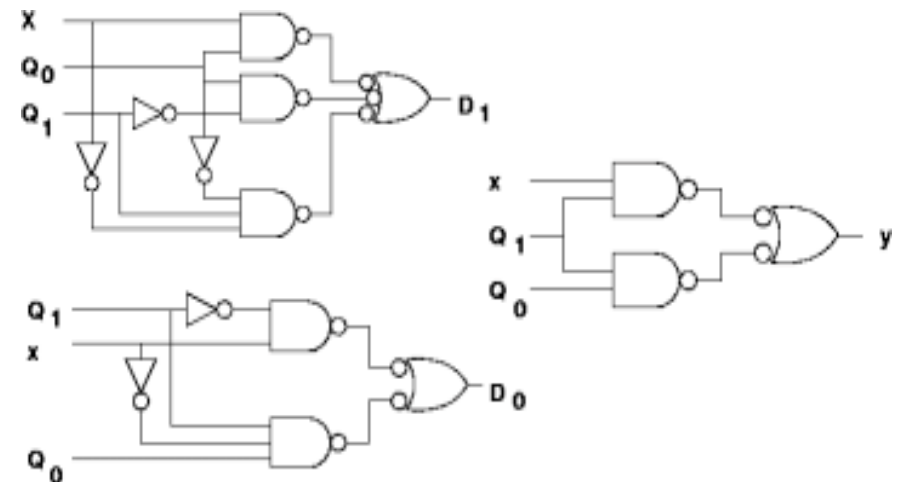
$$D_0 = x^*Q_1 + x^*Q_0^*Q_1$$

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

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

L7.4 (L5.18) FSM Combinational Logic Wed. February 20, 2002

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



L7.5 (L5.19) Another FSM: Divide by Five Wed. February 20, 2002

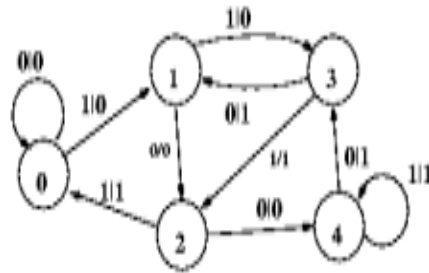
This FSM has a single input which represents a number.
 The LSB comes first, another with each clock pulse.
 The output, also bit serial, is similar.
 The state of the FSM is the remainder of the division operation.

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

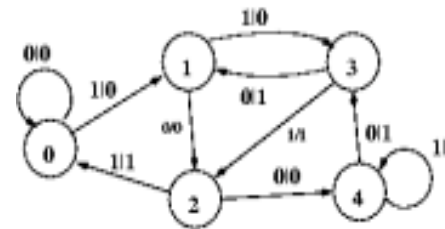
$$\begin{array}{r} 19 \\ 5 \overline{) 95} \\ \underline{5} \\ 40 \\ \underline{40} \\ 00 \end{array}$$

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

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



L7.6 (L5.20) Divide by Five Implementation Wed. February 20, 2002



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

$$D2 = /Q2 * x + /Q2 * Q0 + /Q1 * /x$$

$$D1 = Q0 * x + /Q1 * Q0 + Q2 * /x$$

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

$$D0 = Q2 * /x + Q1 * Q0 * /x + /Q2 * /Q1 * x$$

$$y = Q2 + Q1 * Q0 + Q1 * x$$

L7.7 Implementing FSMs in VHDL Wed. February 20, 2002

The hard part is to figure out what you want to do.
 Implementation in VHDL is straightforward provided you copy from something that works, i.e., get the syntax right!

The entity is straightforward and easy.

Don't forget that the semicolon goes AFTER the last parenthesis.

The architecture has three goals.

1. Implement the state register.
2. Implement the combinational logic for the next state.
3. Implement the combinational logic for the outputs.

All three goals can be accomplished with one process or goal 1 can be implemented with a process and process(es) or (other) concurrent statements used to implement the combinational logic.

L7.8 Two Processes i1by5two.vhd Wed. February 20, 2002

```
library ieee;
use ieee.std_logic_1164.all;
entity by5 is port (
    x, clk : in std_logic;
    y      : out std_logic);
end by5;
architecture state_machine of by5 is
    type StateType is (state0, state1,
        state2, state3, state4);
    signal p_s, n_s : StateType;
begin
    state_clocked:process(clk) - - register
    begin
        if rising_edge(clk) then p_s <= n_s;
        end if;
    end process state_clocked;
    fsm:process(p_s, x) - - combinational
    begin -- case
        y <= '0';
        case p_s is
            when state0 => if x = '1'
                then n_s <= state1; end if;
            when state1 => if (x = '1')
                then n_s <= state2;
                else n_s <= state1; end if;
            when state2 => if (x = '1')
                then n_s <= state0;
                else n_s <= state4; end if;
            when state3 => y <= '1';
                if (x = '1')
                then n_s <= state2;
                else n_s <= state1; end if;
            when state4 => y <= '1';
                if (x = '0')
                then n_s <= state3;
                else n_s <= state0; end if;
            when others => n_s <= state0;
        end case;
    end process fsm;
end architecture state_machine;
```

L7.9 One Process `ilby5one.vhd` Wed. February 20, 2002

```
library ieee;
use ieee.std_logic_1164.all;
entity by5 is port (
  x, clk : in std_logic;
  y      : out std_logic);
end by5;
architecture state_machine of by5 is
  type StateType is (state0, state1,
                    state2, state3, state4);
  signal p_s : StateType;
begin
  -- Note that the output is registered.
  -- Is this what you wanted?
  fsm:process(clk)
  begin -- case
    if rising_edge(clk) then
      y <= '0';
      case p_s is
        when state0 => if x = '1'
          then p_s <= state1;
        else p_s <= state0; end if;
      end case;
    end if;
  end process fsm;
end architecture state_machine;
```

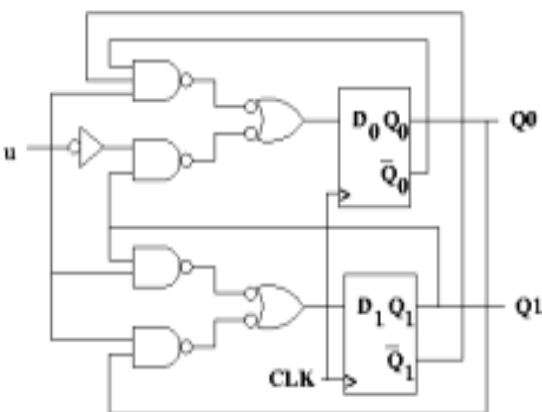
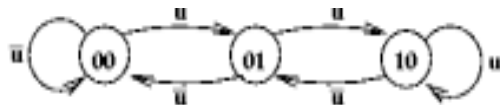
```
When state1 => if x = '1'
  then p_s <= state3;
  else p_s <= state2; end if;
when state2 => if x = '1'
  then p_s <= state0;
  else p_s <= state4; end if;
when state3 => y <= '1';
  if x = '1'
  then p_s <= state2;
  else p_s <= state1;
  end if;
when state4 => y <= '1';
  if x = '1'
  then p_s <= state4;
  else p_s <= state3;
  end if;
  when others => p_s <= state0;
end case;
end if;
end process fsm;
end architecture state_machine;
```

L7.10 Three Processes `ilby5sv.vhd` Wed. February 20, 2002

```
library ieee;
use ieee.std_logic_1164.all;
entity by5 is port (
  x, clk : in std_logic;
  y      : out std_logic);
end by5;
architecture state_machine of by5 is
  signal p_s, n_s : std_logic_vector(2 downto 0);
  signal p_sx : std_logic_vector(3 downto 0);
  constant state0 : std_logic_vector(2 downto 0) := "000";
  constant state1 : std_logic_vector(2 downto 0) := "001";
  constant state2 : std_logic_vector(2 downto 0) := "010";
  constant state3 : std_logic_vector(2 downto 0) := "110";
  constant state4 : std_logic_vector(2 downto 0) := "100";
begin
  state_clocked:process(clk) -- register
  begin
    if rising_edge(clk) then p_s <= n_s;
    end if;
  end process state_clocked;
```

```
-- combinational output spec.
y <= '1' when ((p_s = state4) or
  (p_s = state3) or
  ((p_s = state2) and (x = '1')))
  else '0';
-- combinational next state
specification
p_sx <= p_s & x;
with p_sx select
n_s <= state0 when "0000",
  state1 when "0001",
  state2 when "0010",
  state3 when "0011",
  state4 when "0100",
  state0 when "0101",
  state2 when "0110",
  state1 when "0111",
  state3 when "1000",
  state4 when "1001",
  state0 when others;
end architecture state_machine;
```

L7.11 A Simple FSM Wed. February 20, 2002



Q1 Q0	00	01	11	10
0	0	0	X	0
1	0	1	X	1

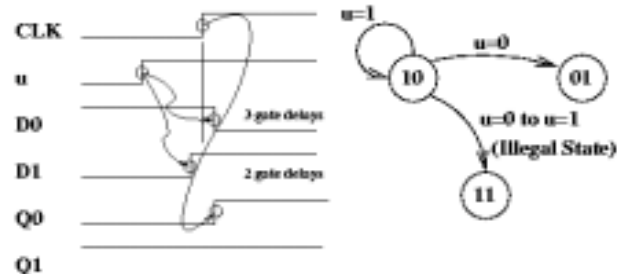
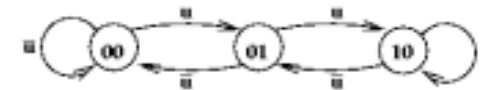
$$D1 = u \cdot Q0 + u \cdot Q1$$

Q1 Q0	00	01	11	10
0	0	0	X	1
1	1	0	X	0

$$D0 = u \cdot /Q0 \cdot /Q1 + /u \cdot Q1$$

L7.12 Problem Transition Wed. February 20, 2002

Consider the transition from state 10 (2) to state 01 (1), if u changes from 0 to 1 close to the clock edge:



Most of the time, this circuit will work just fine. It makes a mistake and enters an illegal state ONLY when the input transition is close to a clock edge.

Q1 Q0	00	01	11	10
0	0	0	X	0
1	0	1	X	1

$$D1 = u \cdot Q0 + u \cdot Q1$$

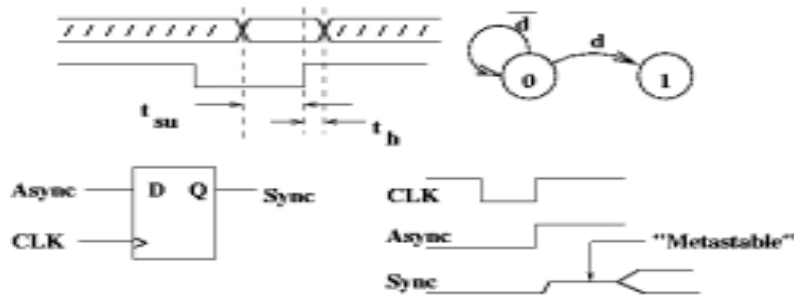
Q1 Q0	00	01	11	10
0	0	0	X	1
1	1	0	X	0

$$D0 = u \cdot /Q0 \cdot /Q1 + /u \cdot Q1$$

L7.13 Important Design Rule Wed. February 20, 2002

DESIGN RULE:

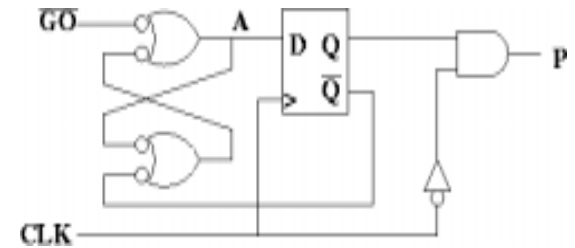
1. Synchronize ALL external signals.
2. Any asynchronous input must affect ONLY ONE flip-flop.



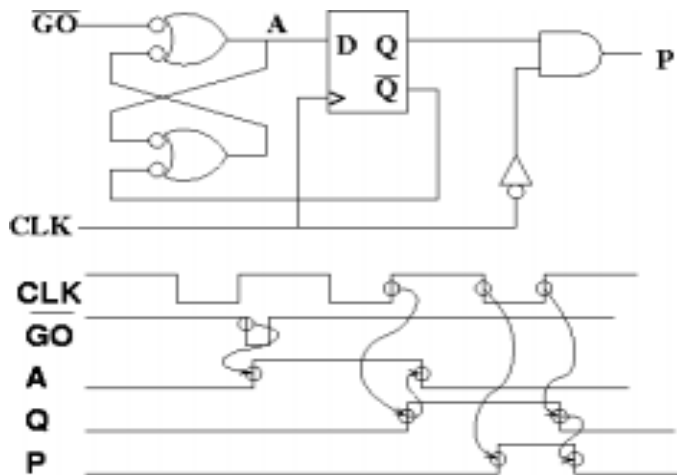
Any combinational logic with "Sync" as an input will be "glitchy" until after the metastable state has expired. In particular, do NOT use "Sync" as a CLK input.

6.111 Introductory Digital Systems Laboratory
 L7.14 VHDL Code for Short Pulse Catcher Wed. February 20, 2002

```
library ieee;
use ieee.std_logic_1164.all;
entity spulse is
    port(n_go, clk : in std_logic;
         p : out std_logic);
end spulse;
-- purpose: catch a short pulse
architecture behavioral of spulse is
    signal a, n_a, x, n_x, n_clk: std_logic;
    attribute synthesis_off of a: signal is true;
    attribute synthesis_off of n_a: signal is true;
begin -- behavioral
    a <= (not n_go) or (not n_a);
    n_a <= (not a) or (not n_x);
    n_x <= (not x);
    p <= x and (not clk);
ff: process(clk)
begin
    if rising_edge(clk) then
        x <= a;
    end if; end process ff;
end behavioral;
```

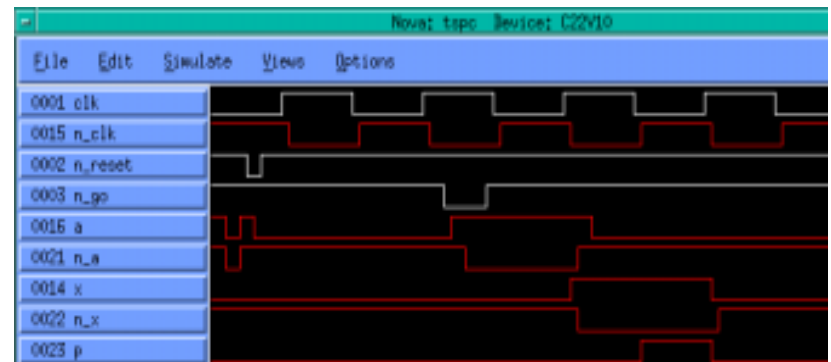


L7.15 Catch a Short Pulse Wed. February 20, 2002



6.111 Introductory Digital Systems Laboratory
 L7.16 Simulation of Catching a Short Pulse Wed. February 20, 2002

```
Begin -- behavioral
a <= (not n_go) or (not n_a);
n_a <= (not a) or (not n_x);
n_x <= (not x);
p <= x and (not clk);
ff: process(clk)
begin
    if rising_edge(clk) then
        x <= a;
    end if;
end process ff;
end behavioral;
```

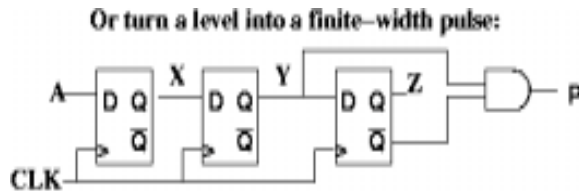


L7.17 VHDL Code for a Pulse Shaper Wed. February 20, 2002

```

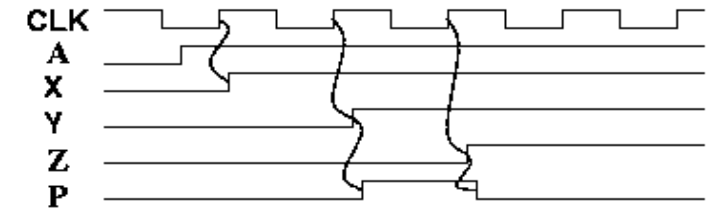
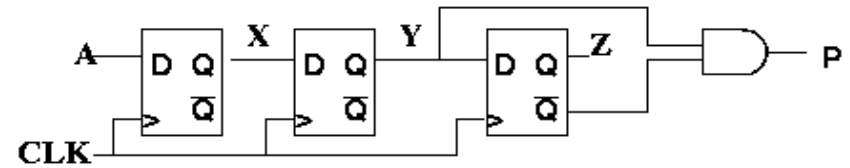
library ieee;
use ieee.std_logic_1164.all;
entity pform is
    port(A, CLK: in std_logic;
         P: out std_logic);
end pform;

-- purpose: Turn a level into a short pulse
architecture behavioral of pform is
    signal X, Y, Z: std_logic;
begin -- behavioral
    ff: process(CLK)
    begin
        if rising_edge(CLK) then
            X <= A;
            Y <= X;
            Z <= Y;
        end if;
    end process ff;
    P <= Y AND (not Z);
end behavioral;
    
```



L7.18 Turn a Level Into a Pulse Wed. February 20, 2002

Or turn a level into a finite-width pulse:



L7.19 Simulation of Level to Pulse Wed. February 20, 2002

```

begin -- behavioral
ff: process(CLK)
    begin
        if rising_edge(CLK) then
            X <= A;
            Y <= X;
            Z <= Y;
        end if;
    end process ff;
    P <= Y AND (not Z);
end behavioral;
    
```

