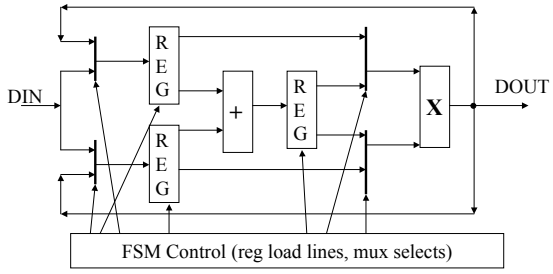
Finite State Machines

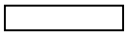
- The job of a finite state machine is to sequence operations on a datapath



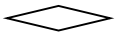
Algorithmic State Chart (ASM)

- An ASM chart can be used to describe FSM behavior

Only three action signals can appear within an ASM chart:



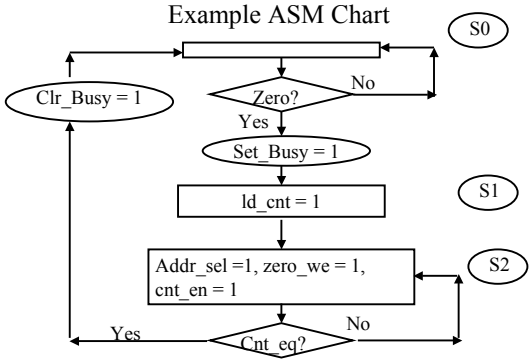
State box. Each box represents a state. Outputs within a state box is an UNCONDITIONAL output (always asserted in this state).



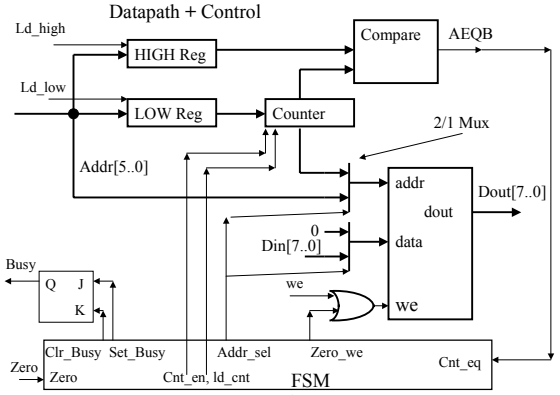
Decision box. A condition in this box will decide next state condition.

Conditional output box. If present, will always follow a decision box; output within it is conditional.

Example ASM Chart



Datapath + Control



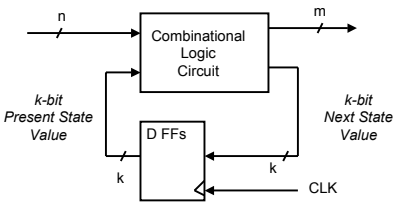
Comments about ASM Example

- How many states?
  - Three states, count the boxes
- How many inputs?
  - Two inputs (Zero, Cnt_eq). Count signals within decision boxes. Inputs ALWAYS appear within decision boxes.
- How many outputs?
  - 4 unconditional outputs (count signals within state boxes)
  - 2 conditional output (count signals within conditional output boxes
  - Outputs ALWAYS appear in either state boxes or conditional output boxes.

FSM Implementation

Will always use VHDL to implement FSMs in this class.

Most common method is to use ONE process for implementing state registers, ONE process for implementing logic.



State Encoding

- How we encode the states is an implementation decision
- For K states, need a minimum of $\log_2(K)$ Dffs.
- Minimal encoding for example is two FFs
  - S0 = 00, S1 = 01, S2 = 10 (counting order)
  - S0 = 00, S1 = 01, S2 = 11, (Gray code for S0->S1->S2) Gray code usually faster, less logic than counting order
- One Hot encoding, one FF per state
  - S0 = 001, S1 = 010, S2 = 100
  - For large FSMs (> 16 states), one hot can be faster than minimal encoding

FSM Entity Declaration, Part of Architecture

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

-- FSM entity for RAM Zero example
entity ramfsm is
 port (  clk, reset:  in std_logic;
          zero, cnt_eq:  in std_logic;  -- control inputs
          set_busy, clr_busy:  out std_logic; -- control outputs
         addr_sel, cnt_en, ld_cnt, zero_we: out std_logic;
         state: out std_logic_vector(1 downto 0)   -- state out for debugging
  );

end ramfsm;

architecture a of ramfsm is
 signal pstate: std_logic_vector(1 downto 0);
 CONSTANT S0 : std_logic_vector(1 downto 0) := "00";   --- state encoding
 CONSTANT S1 : std_logic_vector(1 downto 0) := "01";
 CONSTANT S2 : std_logic_vector(1 downto 0) := "10";
```


FSM Architecture, One process (cont)

```
begin
 state <= pstate;          -- look at present state for debugging purposes
 stateff:process(clk)      -- process has state transitions ONLY
  begin
   if (reset = '1') then   pstate <= S0;
   elsif (clk'event and clk='1') then  -- rising edge of clock
           CASE pstate IS
             WHEN S0 =>  if (zero = '1') then   pstate <= S1;  end if;
             WHEN S1 =>  pstate <= S2;
              WHEN S2 =>   if (cnt_eq = '1') then pstate <= S0 ;  end if;
              WHEN others => pstate <= S0;
           end case;
    end if;
 end process stateff;

 set_busy <= '1' when (pstate  = S0 and zero = '1') else '0';
 ld_cnt <= '1' when (pstate = S1) else '0';
 addr_sel <= '1' when (pstate = S2) else '0';
 zero_we <= '1' when (pstate = S2) else '0';
 cnt_en <= '1' when (pstate = S2) else '0';
 clr_busy <= '1' when (pstate = S2 and cnt_eq = '1') else '0';
end a;
```


Comments on One Process Implementation

- *Stateff* process defines state FFs and transistions between states
- Outputs of FSM are separate concurrent statements outside of process
- Can be confusing since you separate out the FSM outputs from their state definitions within the CASE statement
- If output code is placed within CASE statement then they would be protected by the clock check and thus would have DFFs placed on their outputs
  - 1 clock cycle of latency to output assertion

FSM Architecture, Two processes

```
architecture a of ramfsm is
 signal pstate, nstate: std_logic_vector(1 downto 0);

 CONSTANT S0 : std_logic_vector(1 downto 0) := "00";   --- state encoding
 CONSTANT S1 : std_logic_vector(1 downto 0) := "01";
 CONSTANT S2 : std_logic_vector(1 downto 0) := "10";

begin
  state <= pstate;          -- look at present state for debugging purposes


  stateff:process(clk)      -- process has DFFs only
   begin
    if (reset = '1') then   pstate <= S0;
    elsif (clk'event and clk='1') then
            pstate <= nstate;    -- updated present state with next state
      endif;
   end process stateff;
```

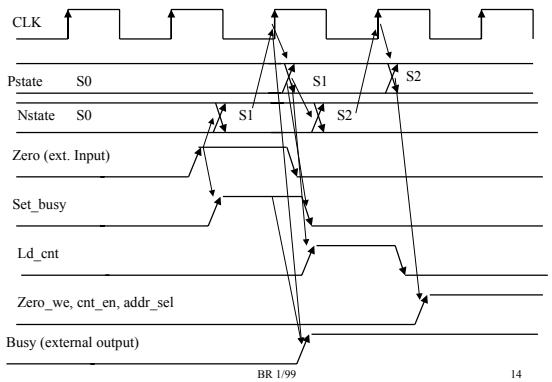

FSM Architecture, Two processes (cont)

```
comblogic: process (zero, cnt_eq, pstate)
 begin
    -- default assignments
    nstate <= pstate;
    set_busy <= '0'; clr_busy <= '0';
    ld_cnt <= '0';
    addr_sel <= '0';
    zero_we <= '0';
    cnt_en <= '0';
    CASE pstate IS
          WHEN S0 =>  if (zero = '1') then
                        set_busy <= '1'; nstate <= S1;
                      end if;
          WHEN S1 => ld_cnt <= '1'; nstate <= S2;
           WHEN S2 => zero_we <= '1' ; cnt_en <= '1' ; addr_sel <= '1';
                      if (cnt_eq = '1') then
                        clr_busy <= '1'; nstate <= S0 ;
                      end if;
           WHEN others => nstate <= S0;
         end case;
  end if;
 end process comblogic;
end a;
```

Comments on Two Process Implementation

- *Stateff* process defines only FFs
- *Comblogic* process defines
  - State transitions
  - Output assertions
  - Has natural mapping from ASM chart to CASE statement
- Default assignments to outputs in *Comblogic* process very important
  - A combinational process; do not want latches synthesized on outputs
  - The assignment "pstate <= nstate" says to not change state unless directed to from within CASE statement.

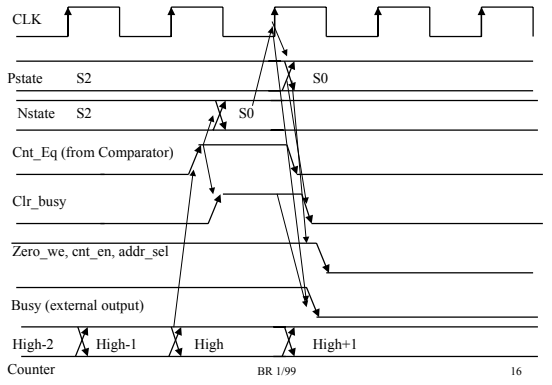
FSM Timing: Start Zero Operation



Comments

- Note that Pstate changes on active clock edge
- Conditional outputs will change based on present state AND external inputs
- Unconditional outputs change on clock edge and remain true as long as in the current state
- In order for BUSY to go high in State S1, 'set_busy' must be asserted in S0 since BUSY comes from JK FF.

FSM Timing: Finish Zero Operation



Comments

- Note that for BUSY to go low in S0, then "clr_busy" had to be asserted in State S2.
- Note that the 'cnt_en' signal stays true for one clock edge after 'cnt_eq' goes true
  - This means that the COUNTER will increment to HIGH+1, sometimes this makes a difference, need to be aware of it