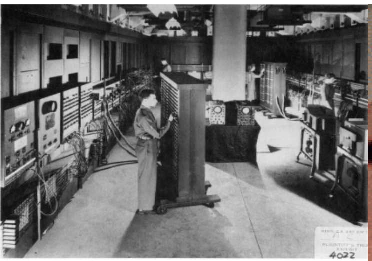
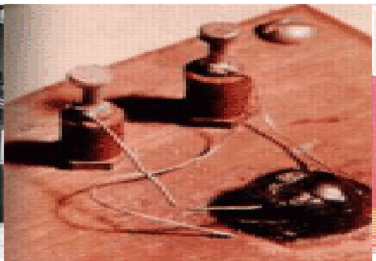
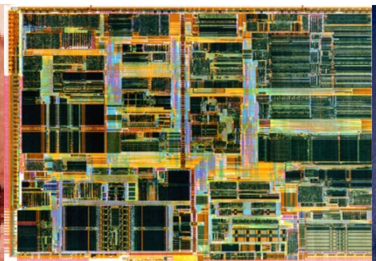
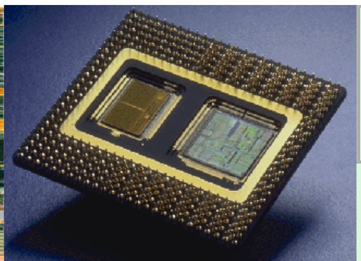
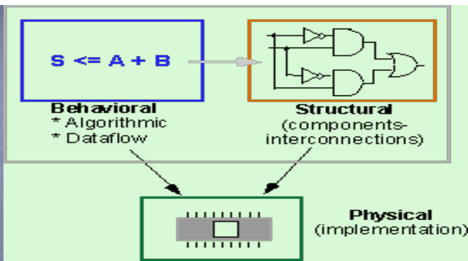
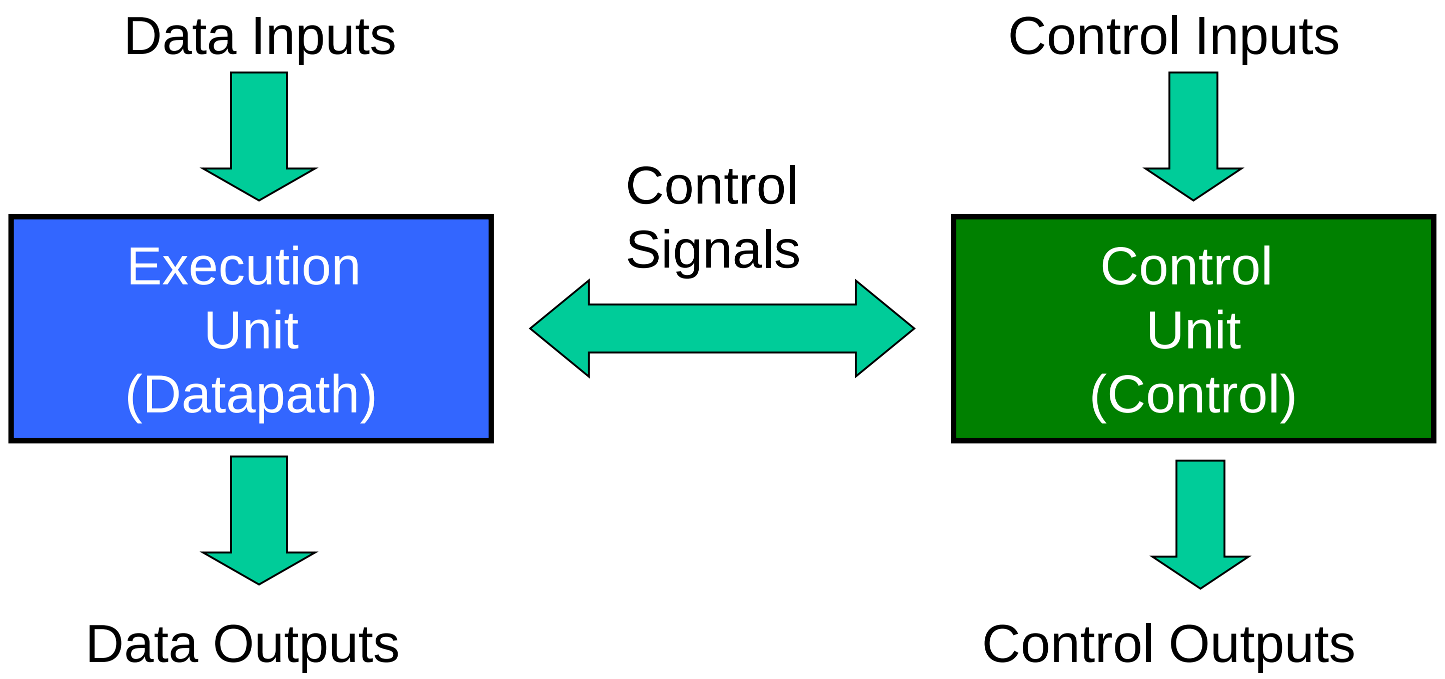
Lecture 13
Finite State Machines (FSM)

Bryan Ackland
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ 07030

Structure of Typical Digital System



Provides necessary resources & interconnect to perform specified task e.g.:

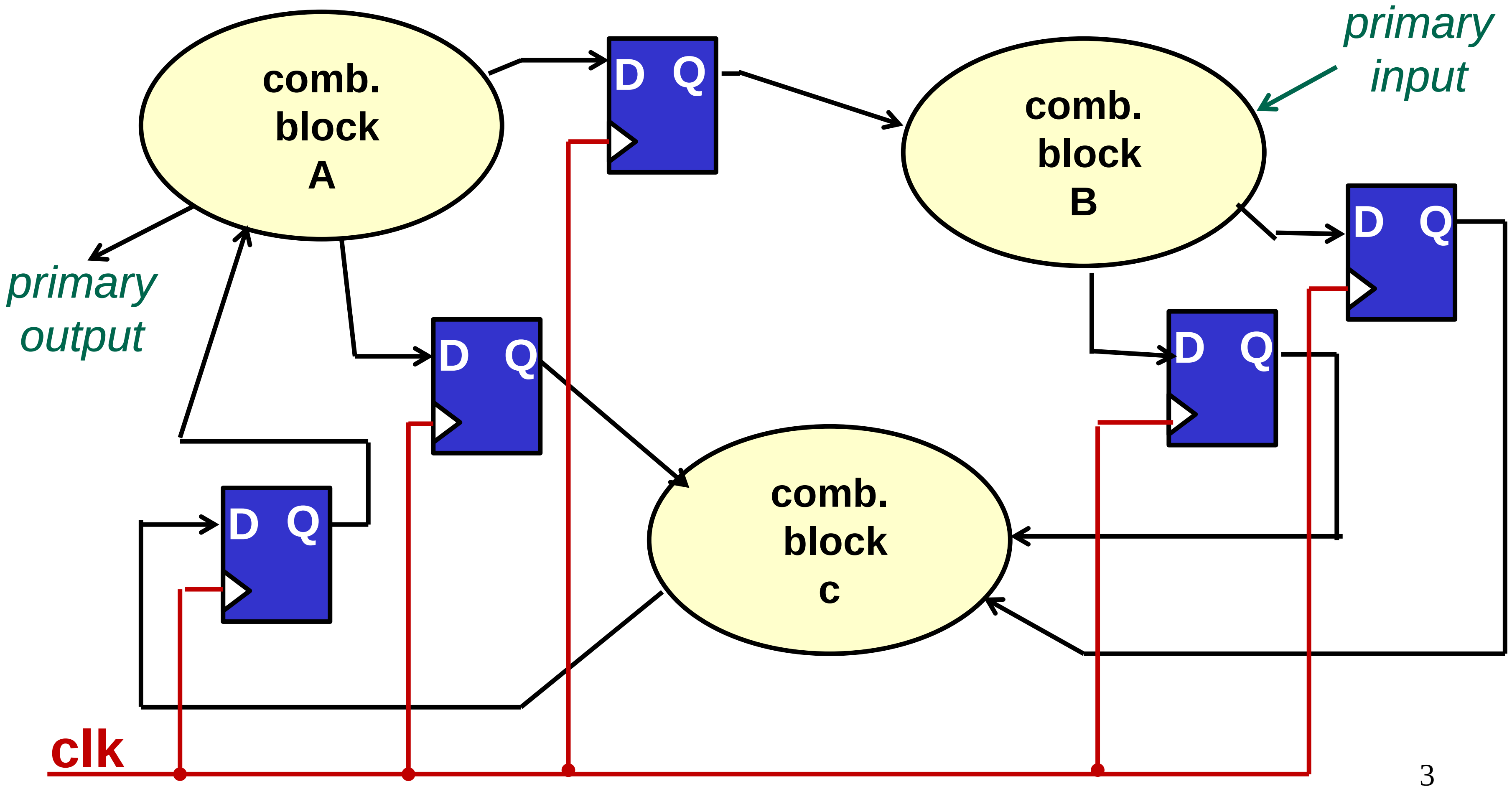
Adders, Multipliers, Shifters, Registers, Memories, etc.

Controls data movement and operation of execution unit. Usually follows some “program” or “sequence”.

Often implemented as one or more Finite State Machine(s)

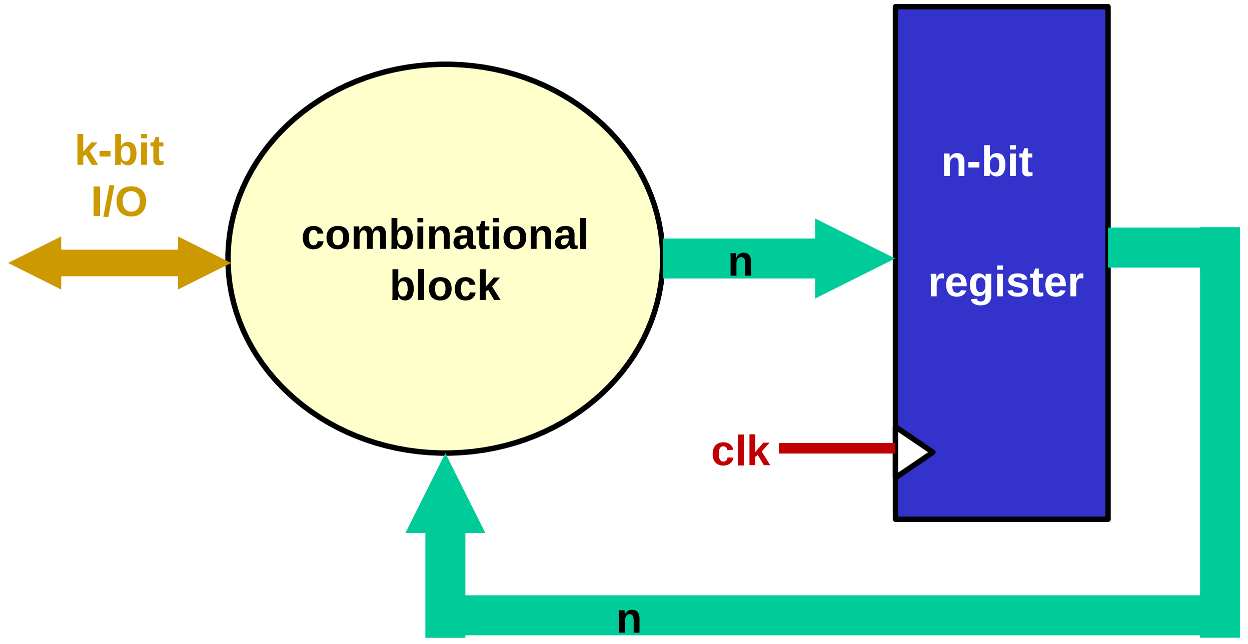
Synchronous (Single Clock) Digital Design

- Preferred design style is combinational circuit modules connected via positive (negative) edge-triggered flip-flops that all use a common clock.



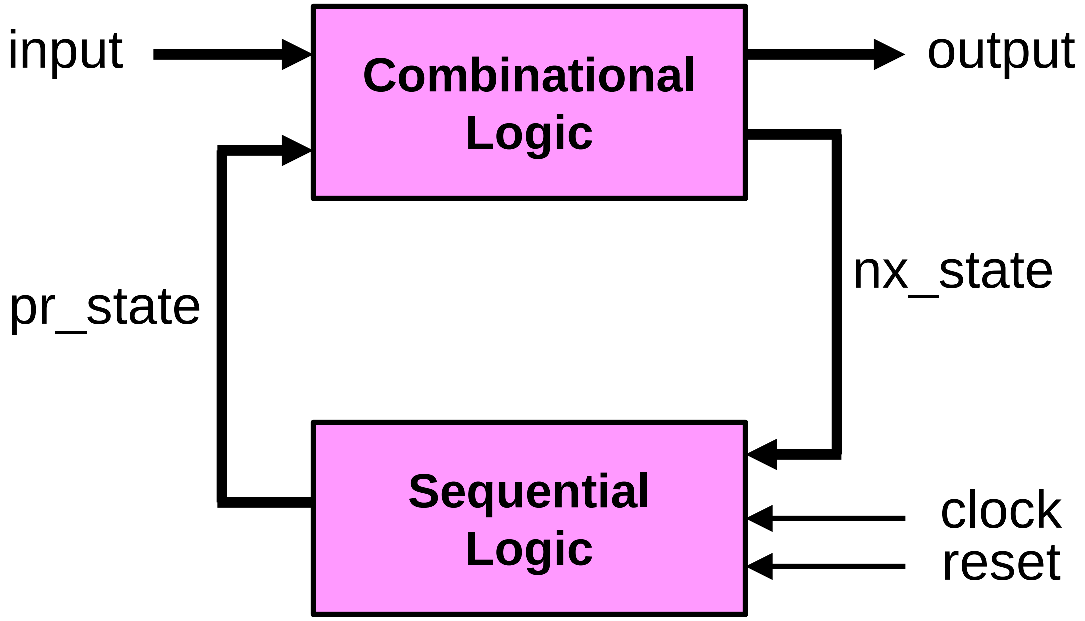
Finite State Machine

- Single clock synchronous system can be modeled as a single combinational block and a multi-bit register



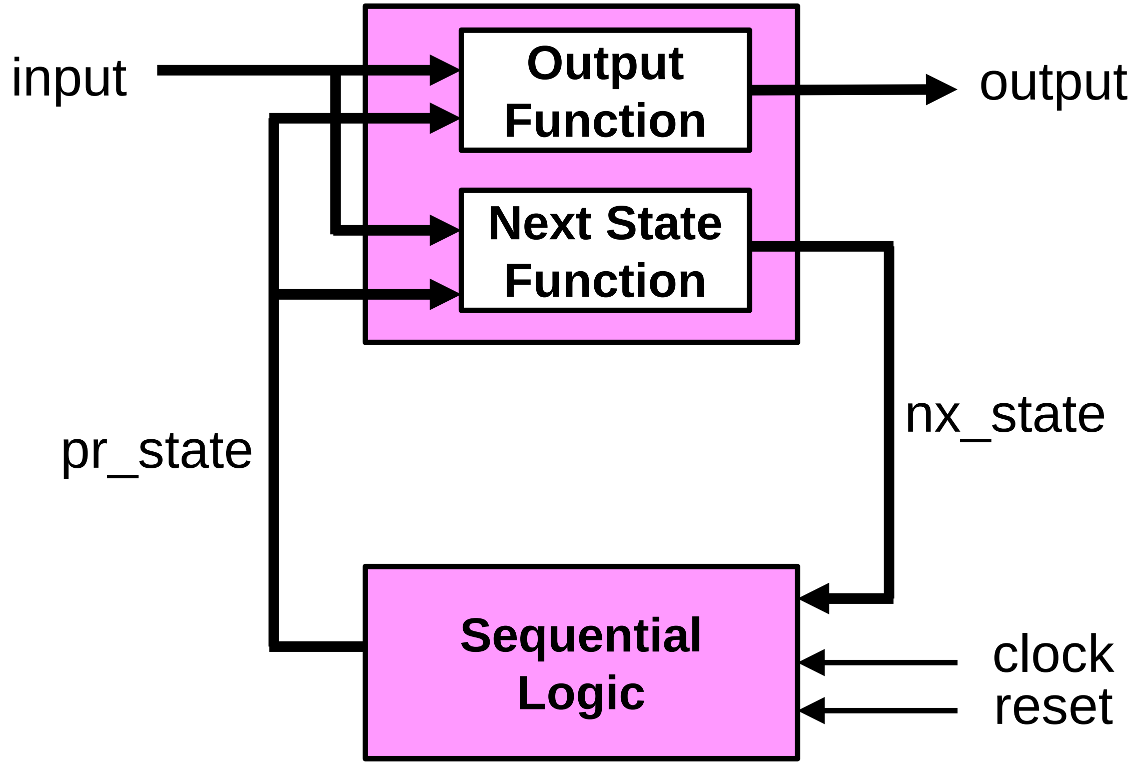
- Values stored in registers are state of the system
- Number of states is finite ($\leq 2^n$)
- State may change as a function of inputs
- Outputs are function of state and inputs

General Architecture of FSM



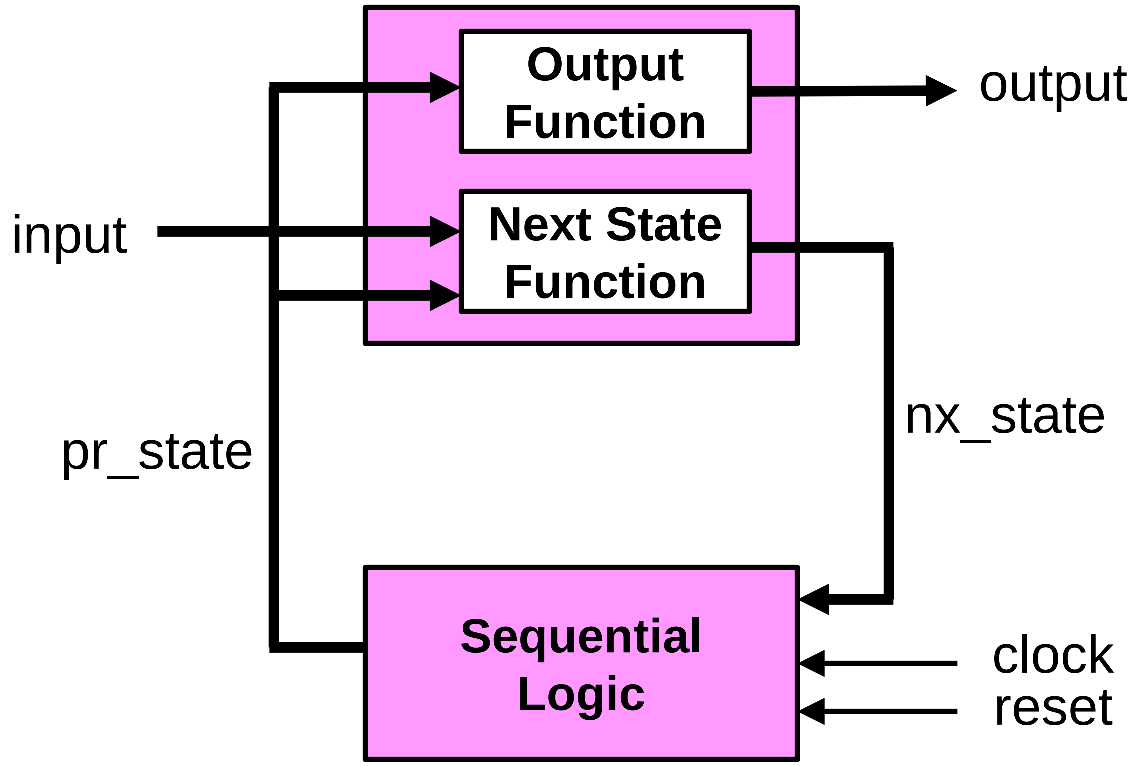
- Next state (nx_state) is a function of present state (pr_state) and inputs
- Output is a function of pr_state. May also be a function of inputs
- Reset allows system to be set to a known state

Mealy Machine



- Output is a function of pr_state *and* inputs
- Fast response (input -> output) – no FF’s in way
- Leads to a fewer number of states
- Propagates asynchronous behavior (e.g. glitches) from input to output

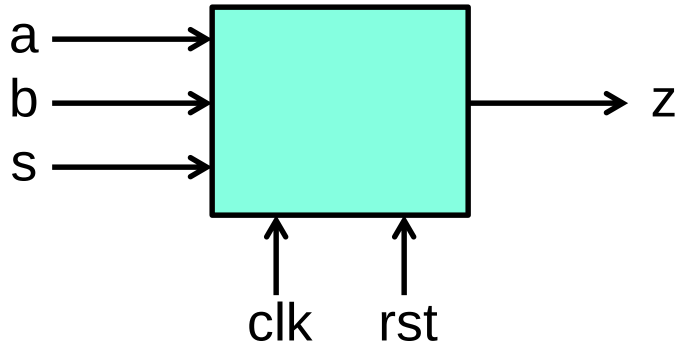
Moore Machine

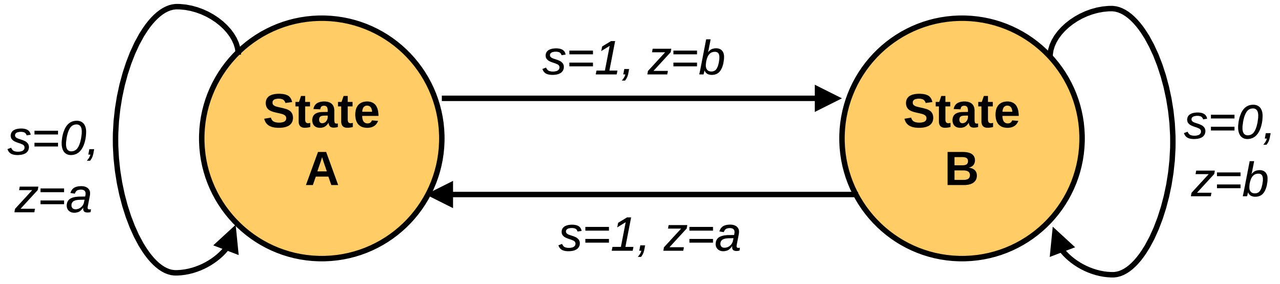


- Output is a function of pr_state *only*
- Slower response: (input -> output) requires clock transition
- Usually requires more states
- Operation is fully synchronous

Example: Toggle Multiplexer

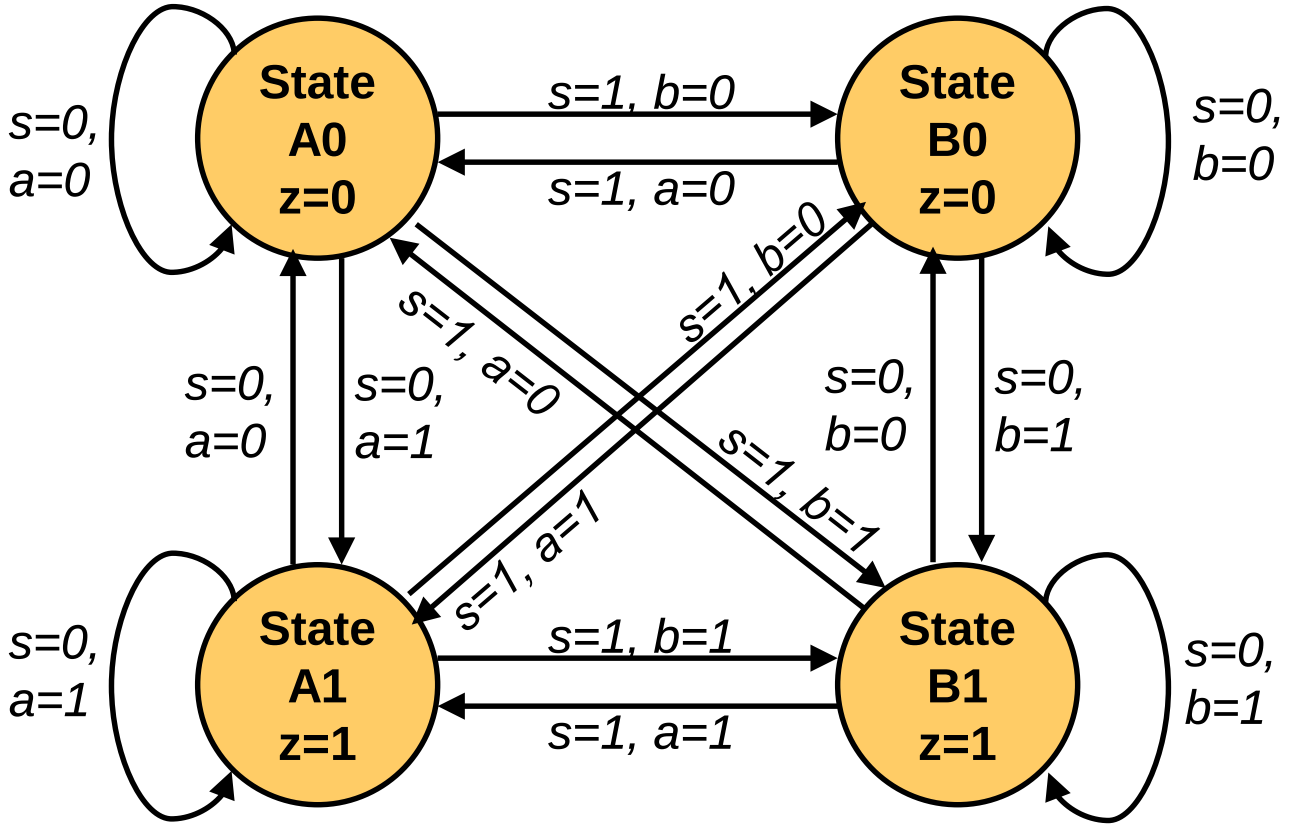
- Design a synchronous multiplexer that selects between two 1-bit inputs a and b. The multiplexer switches from one input to the other whenever a third input s is set to ‘1’





- This is a Mealy machine

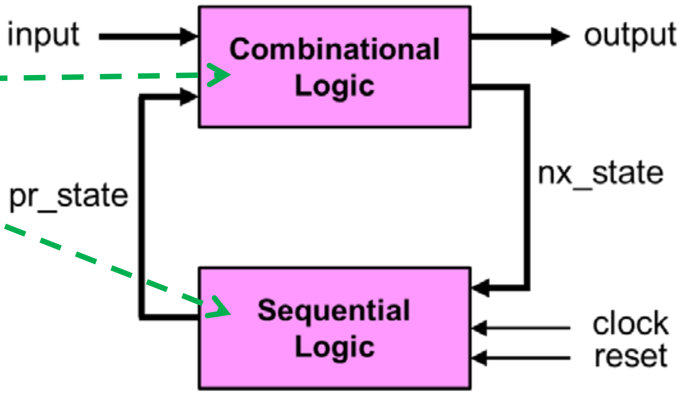
Example: As a Moore Machine



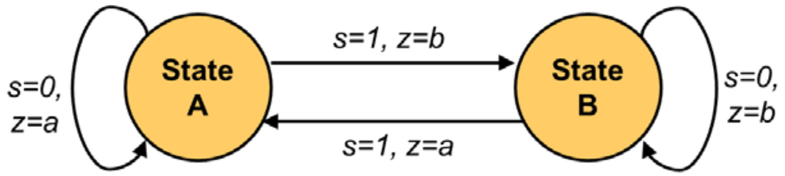
- This is a Moore machine

Toggle Multiplexer as Mealy Machine

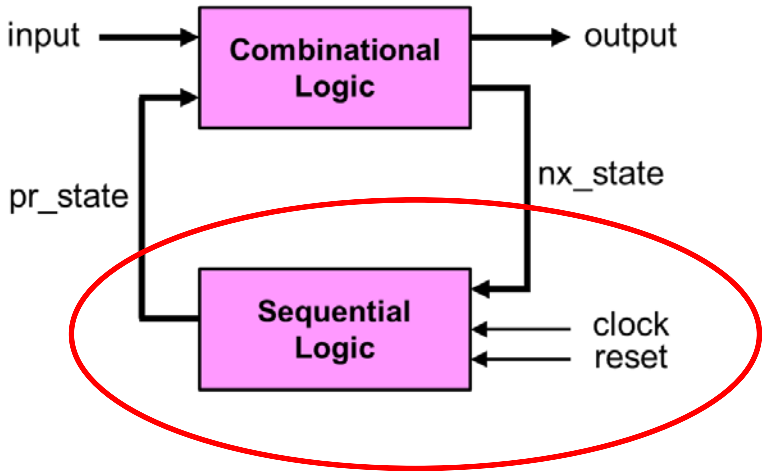
- General approach is to model FSM as two communicating concurrent processes
  - Combinational process
  - Edge triggered clock process



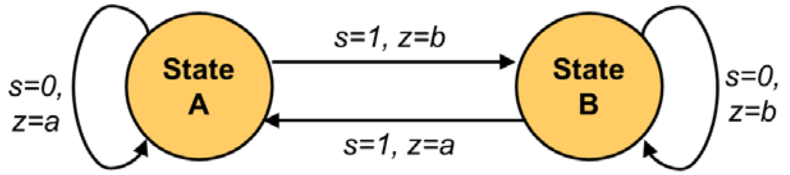
```
library  ieee;
use ieee.std_logic_1164.all;

entity tmpx is
port(a,b,s,clk,rst: in std_logic;
     z: out std_logic);
end entity tmpx;

architecture mealy of tmpx is
type state is (stateA, stateB);
signal pr_state, nx_state : state;
begin
```



Clock Process

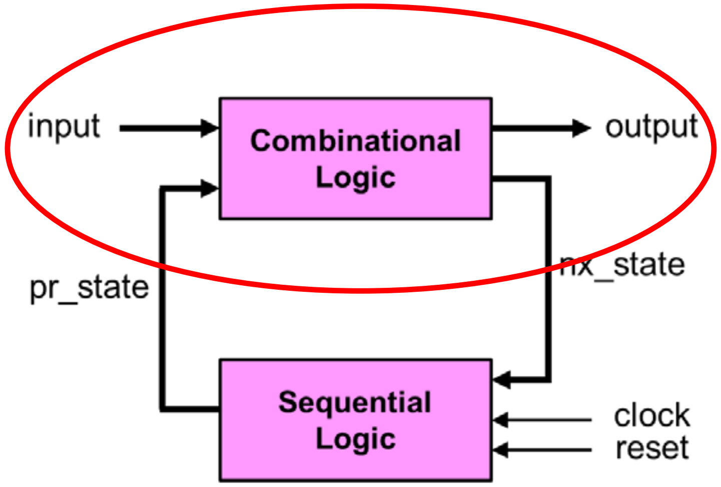


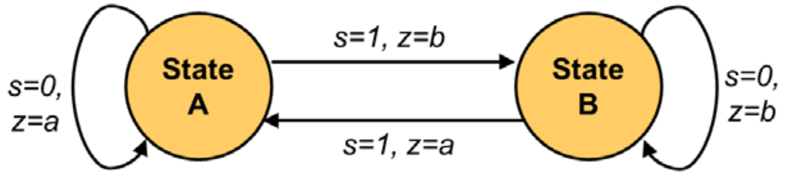
```
p_clk: process (rst, clk)
    begin
        if (rst = ‘1’) then
            pr_state <= stateA;
        elsif (clk’event and clk = ‘1’) then
            pr_state <= nx_state;
        end if;
    end process;
```



Combinational Process uses Case Statement

```
p_comb: process (a, b, s, pr_state)
   begin
      case pr_state is
      when stateA =>
         z <= a;
         if (s = ‘1’) then nx_state <= stateB;
            else nx_state <= stateA;
         end if;
      when stateB =>
         z <= b;
         if (s = ‘1’) then nx_state <= stateA;
            else nx_state <= stateB;
         end if;
      end case;
   end process;
end architecture mealy;
```

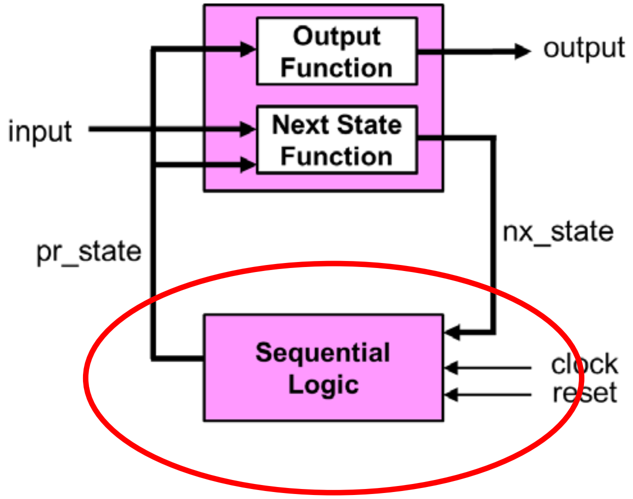


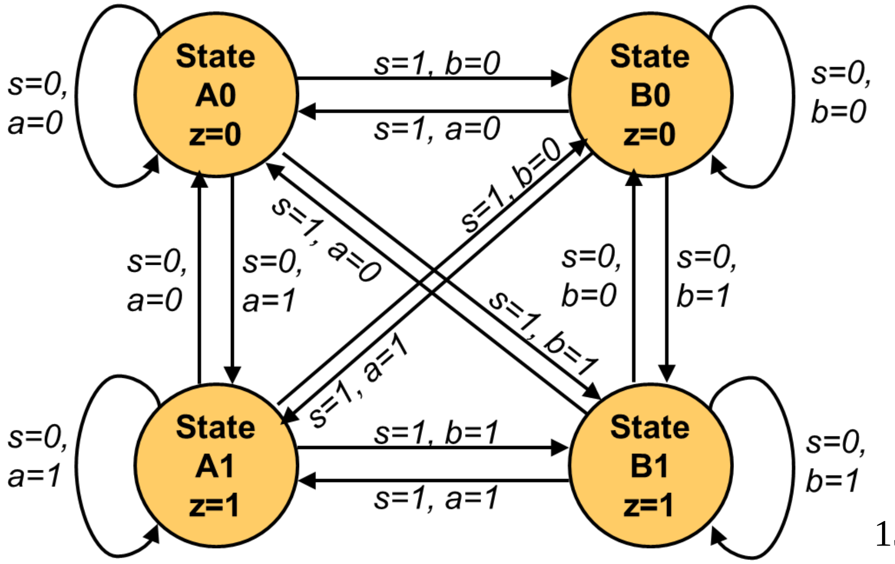


- Make sure all signals are assigned in all branches of case statement to avoid inferring latches

Moore Machine Implementation

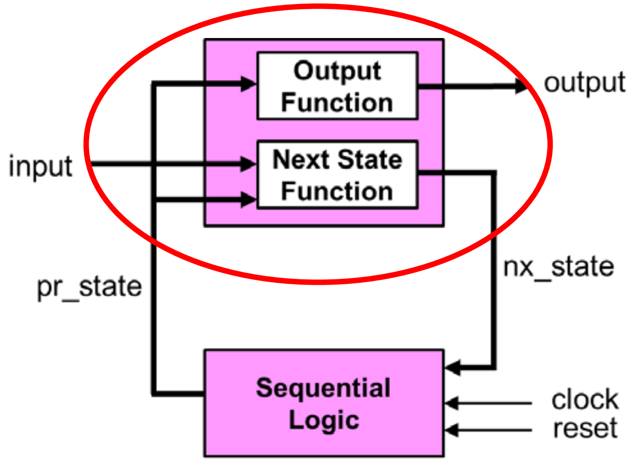
```
architecture moore of tmpx is
type state is (A0, A1, B0, B1);
signal pr_state, nx_state: state;

begin
p0: process (rst, clk)
     begin
        if (rst = ‘1’) then
              pr_state <= A0;
        elsif (clk’event and clk = ‘1’) then
              pr_state <= nx_state;
        end if;
   end process;
```

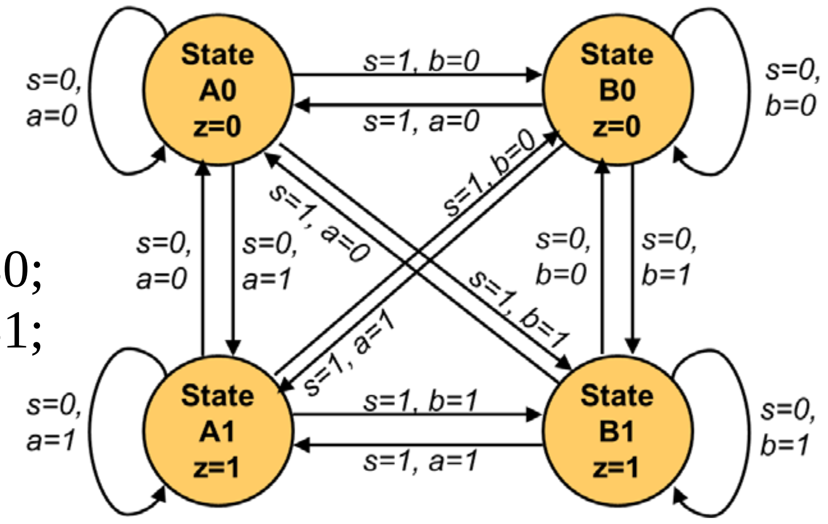




Moore Machine Implementation (2)

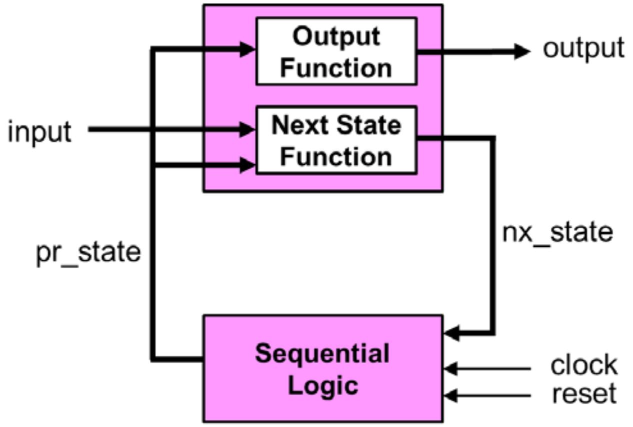
```
p_comb: process (a, b, s, pr_state)
    begin
        case pr_state is
        when A0 =>
            z<=‘0’;
            if s=‘0’ and a=‘1’ then nx_state<=A1;
            elsif s=‘1’ and b=‘0’ then nx_state<= B0;
            elsif s=‘1’ and b=‘1’ then nx_state<= B1;
            else nx_state<=A0;
            end if;
        when A1 =>
            z<=‘1’;
            if s=‘0’ and a=‘0’ then nx_state<=A0;
            elsif s=‘1’ and b=‘0’ then nx_state<= B0;
            elsif s=‘1’ and b=‘1’ then nx_state<= B1;
            else nx_state<=A1;
            end if;
```

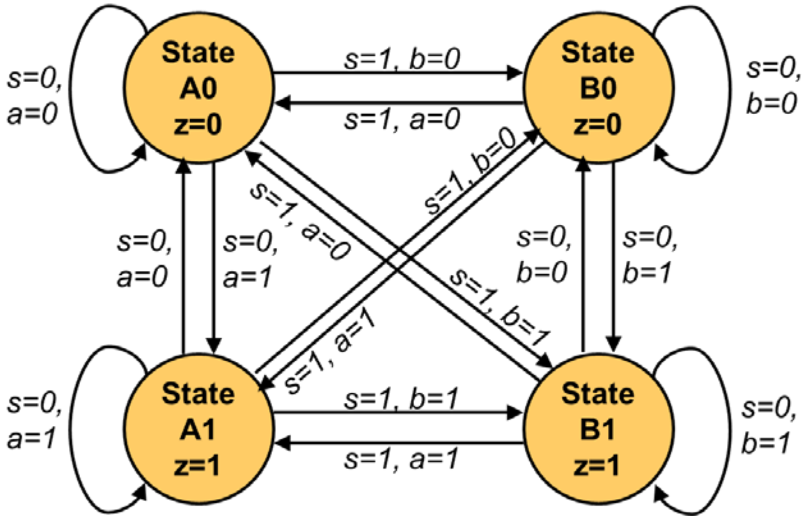




Moore Machine Implementation (3)

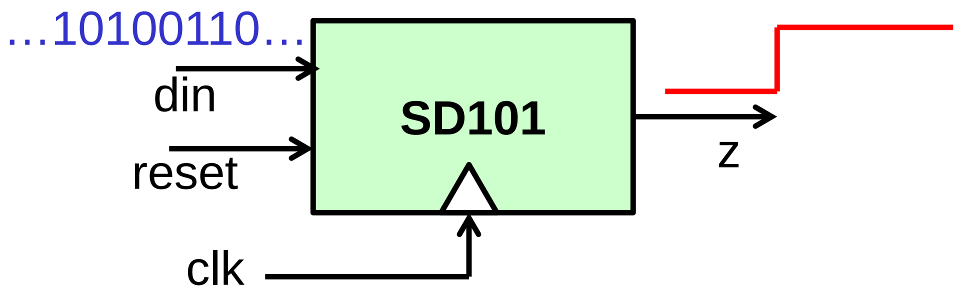
```
    when B0 =>
        z<=‘0’;
        if s=‘0’ and b=‘1’ then nx_state<=B1;
        elsif s=‘1’ and a=‘0’ then nx_state<= A0;
        elsif s=‘1’ and a=‘1’ then nx_state<= A1;
        else nx_state<=B0;
        end if;
    when B1 =>
        z<=‘1’;
        if s=‘0’ and b=‘0’ then nx_state<=B0;
        elsif s=‘1’ and a=‘0’ then nx_state<= A0;
        elsif s=‘1’ and a=‘1’ then nx_state<= A1;
        else nx_state<=B1;
        end if;
    end case;
end process;
end moore;
```

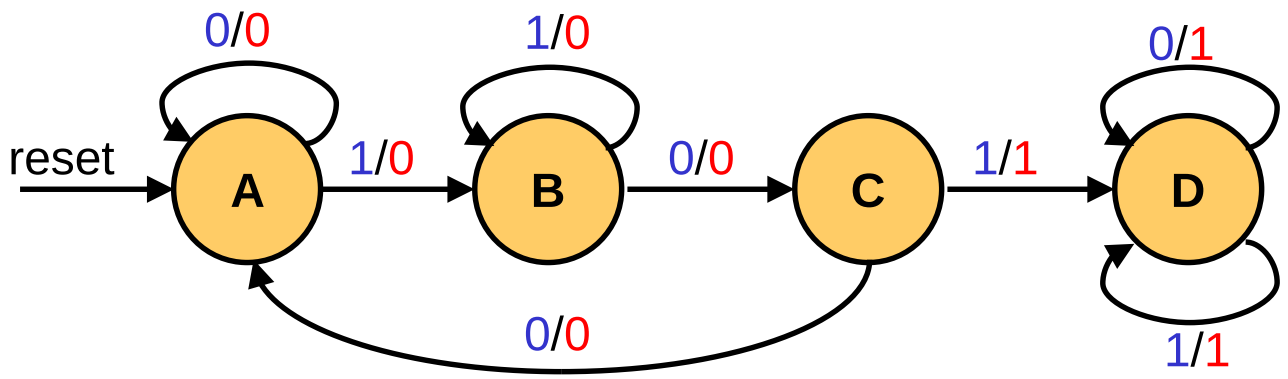




Example: Sequence Detection

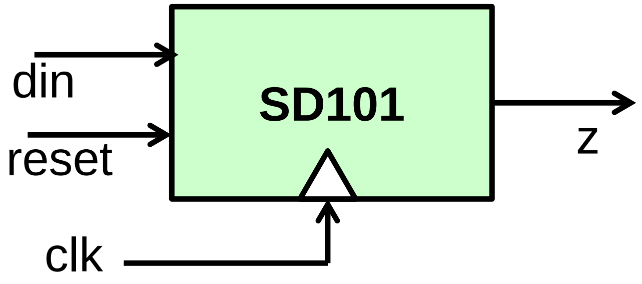
Build a Mealy FSM that has looks for sequence “101” in a serial input stream. When it detects the sequence, it outputs a 1 and holds that value until the machine is reset.





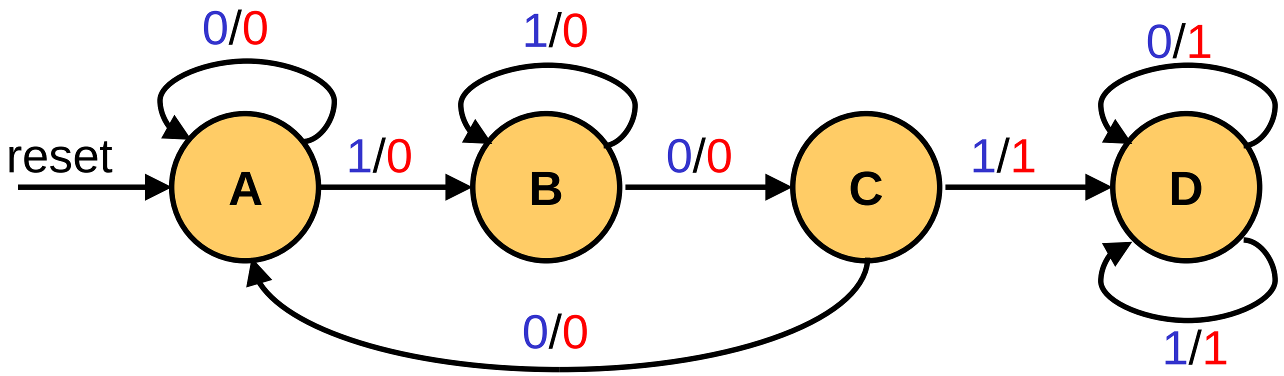
Sequence Detection: Clock Process

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

entity SD101 is
    port(din, reset,clk: in std_logic;
        z: out std_logic);
end entity SD101;

architecture mealy of SD101 is
type state is (stateA, stateB, stateC, stateD);
signal pr_state, nx_state : state;
begin
p_clk: process (rst, clk)
    begin
        if (reset = ‘1’) then
                pr_state <= stateA;
        elsif (clk’event and clk = ‘1’) then
            pr_state <= nx_state;
        end if;
    end process;
```



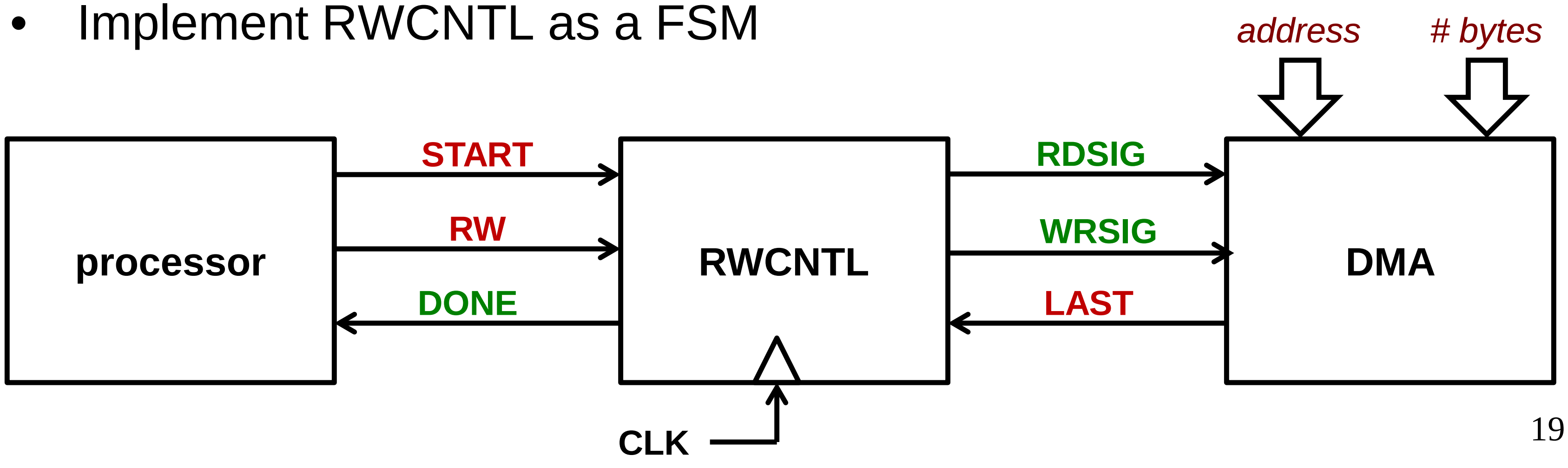
Sequence Detection: Combinational Process

```
p_comb: process (din,  pr_state)
    begin
        case pr_state is
        when stateA =>
            z <= ‘0’;
            if (din = ‘1’) then nx_state <= stateB;
            else nx_state <= stateA;
            end if;
        when stateB =>
            z <= ‘0’;
            if (din = ‘1’) then nx_state <= stateB;
            else nx_state <= stateC;
            end if;
        when stateC =>
            if (din = ‘1’) then z <= ‘1’;
                nx_state <= stateD;
            else z <= ‘0’;
                nx_state <= stateA;
            end if;
        when stateD =>
            z <= ‘1’;
            nx_state <= stateD;
        end case;
    end process;
```



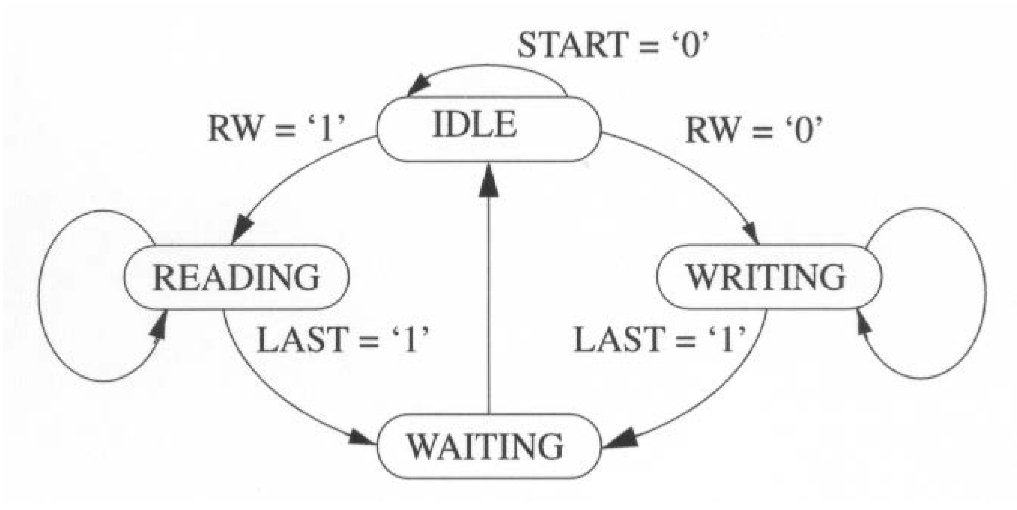
Example: Read-Write Controller

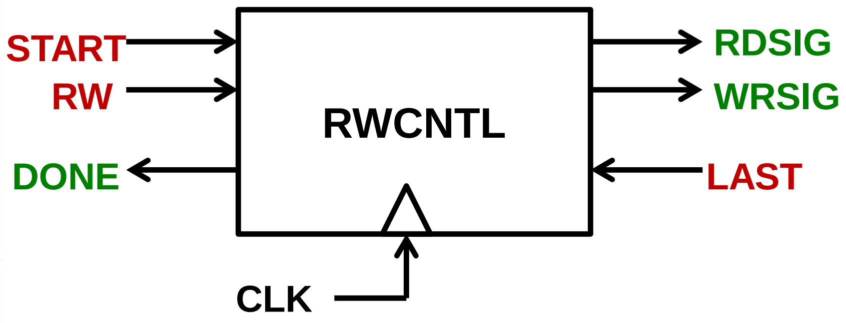
- Suppose we have a read-write controller RWCNTL that acts as an interface between a processor and a DMA unit.
  - The DMA unit is used to transfer blocks of data between the processors memory and an external storage device like a hard disk.
  - Assume the processor has already communicated the memory address and byte count information to the DMA unit (this is part of the “data path”)
- Operation:
  - Processor raises start signal with RW=1 for reading, RW=0 for writing
  - RWCNTL sets (and holds) RDSIG or WRSIG to enable data transfer
  - When transfer is complete, DMA raises signal LAST
  - RWCNTL then resets RDSIG/WRSIG and raises DONE flag
- Implement RWCNTL as a FSM



Read/Write Controller – State Transition Diag.

- RWCNTL starts at state IDLE, waiting for input signal START to go to ‘1’ and then changes to either state READING or state WRITING depending on the value of RW input.
- States READING and WRITING persist until input signal LAST goes to ‘1’ which changes state to WAITING. After one clock cycle, state WAITING always goes to state IDLE.
- The FSM has three outputs RDSIG, WRSIG, and DONE. They are ‘1’ when they are in state READING, WRITING, and WAITING, respectively, otherwise they are ‘0’.

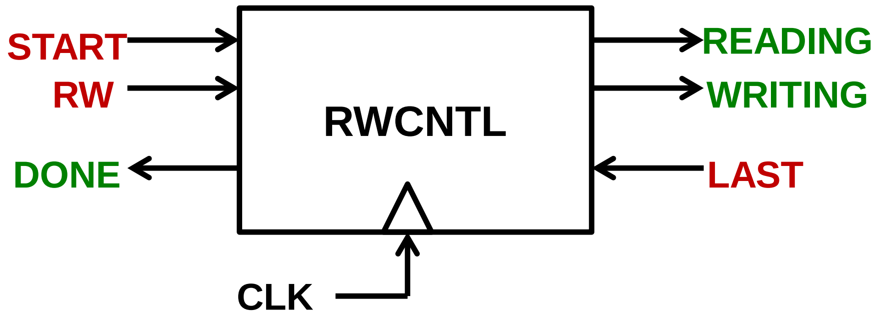




RWCNTL: Clock Process

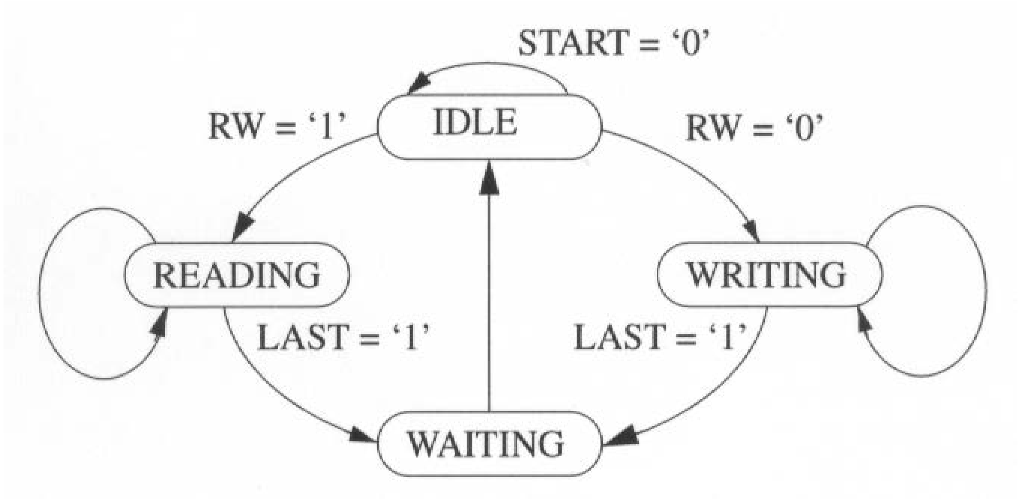
```
entity RWCNTL is
   port(
      CLK : in  std_logic;
      START : in  std_logic;
      RW    : in  std_logic;
      LAST  : in  std_logic;
      RDSIG : out std_logic;
      WRSIG : out std_logic;
      DONE  : out std_logic);
end entity RWCNTL;

architecture RTL of RWCNTL is
   type STATE is (IDLE, READING,
      WRITING, WAITING);
   signal PR_STATE, NX_STATE: STATE;
begin
```

```
   seq : process
     begin
       wait until CLK'event and CLOCK = '1';
       PR_STATE <= NX_STATE;
     end process;
  end RTL;
```

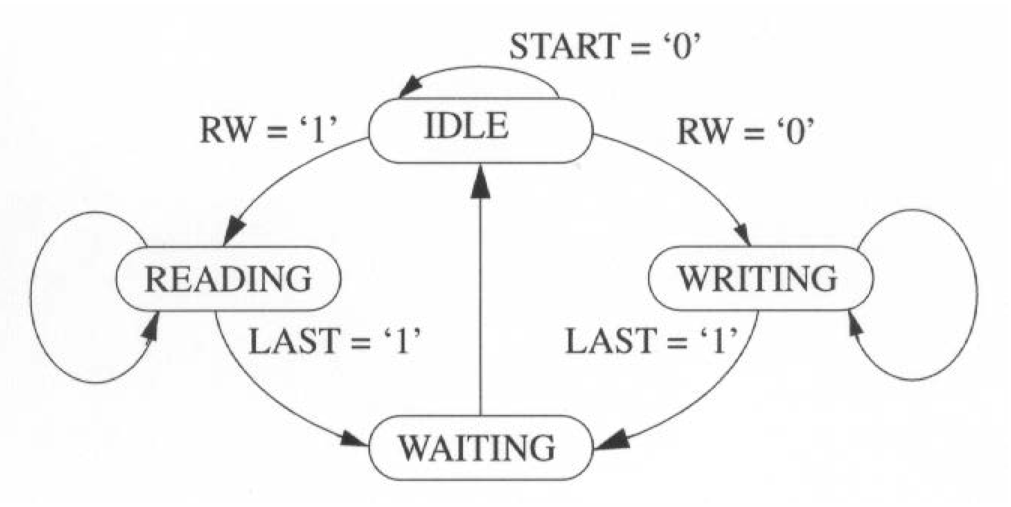


RWCNTL: Combinational Process

```
comb : process (PR_STATE, START, RW, LAST)
  begin
    DONE  <= '0'; RDSIG <= '0'; WRSIG <= '0';
    case PR_STATE is
      when IDLE =>
        if START = '0' then
          NX_STATE <= IDLE;
        elsif RW = '1' then
          NX_STATE <= READING;
        else
          NX_STATE <= WRITING;
        end if;
      when READING =>
        RDSIG <= '1';
        if LAST = '0' then
          NX_STATE <= READING;
        else
          NX_STATE <= WAITING;
        end if;
```

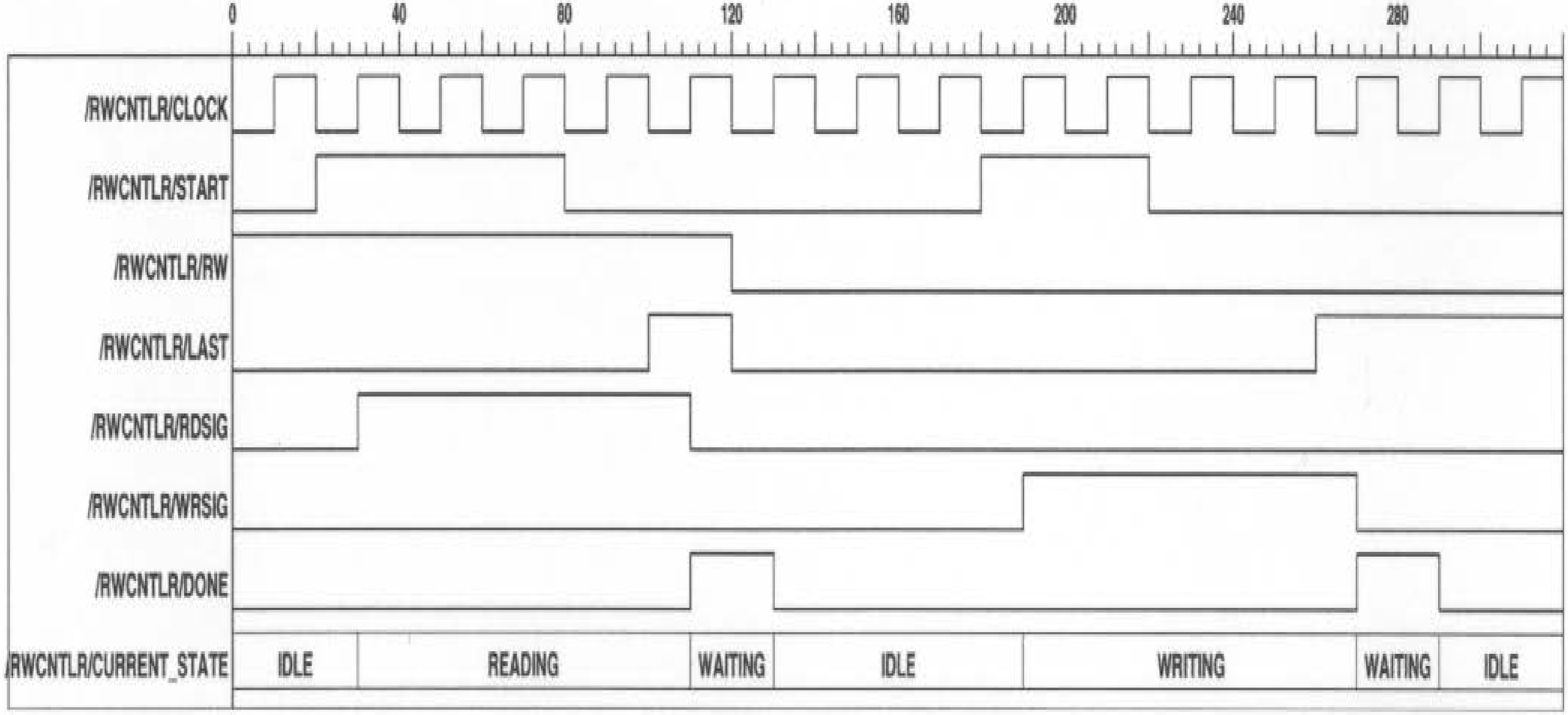


RWCNTL: Combinational Process (2)

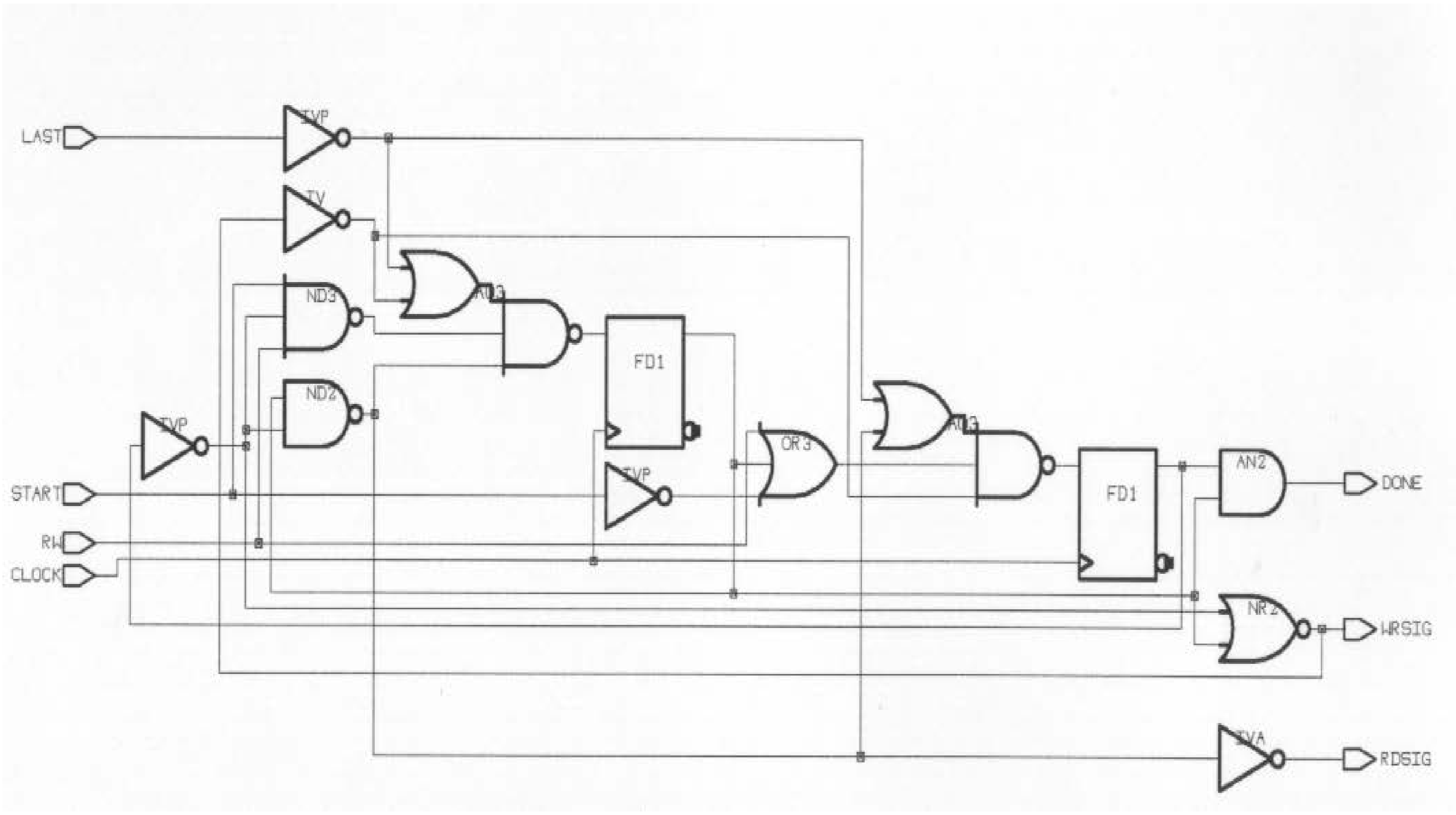


```
        when WRITING =>
          WRSIG <= '1';
            if LAST = '0' then
              NX_STATE <= WRITING;
          else
              NX_STATE <= WAITING;
          end if;
        when WAITING =>
          DONE <= '1';
          NX_STATE <= IDLE;
    end case;
  end process;
```

Simulation Waveforms

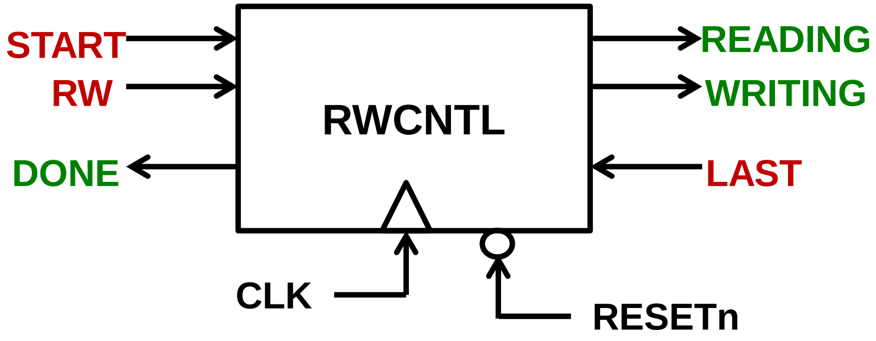


Synthesized RWCNTL

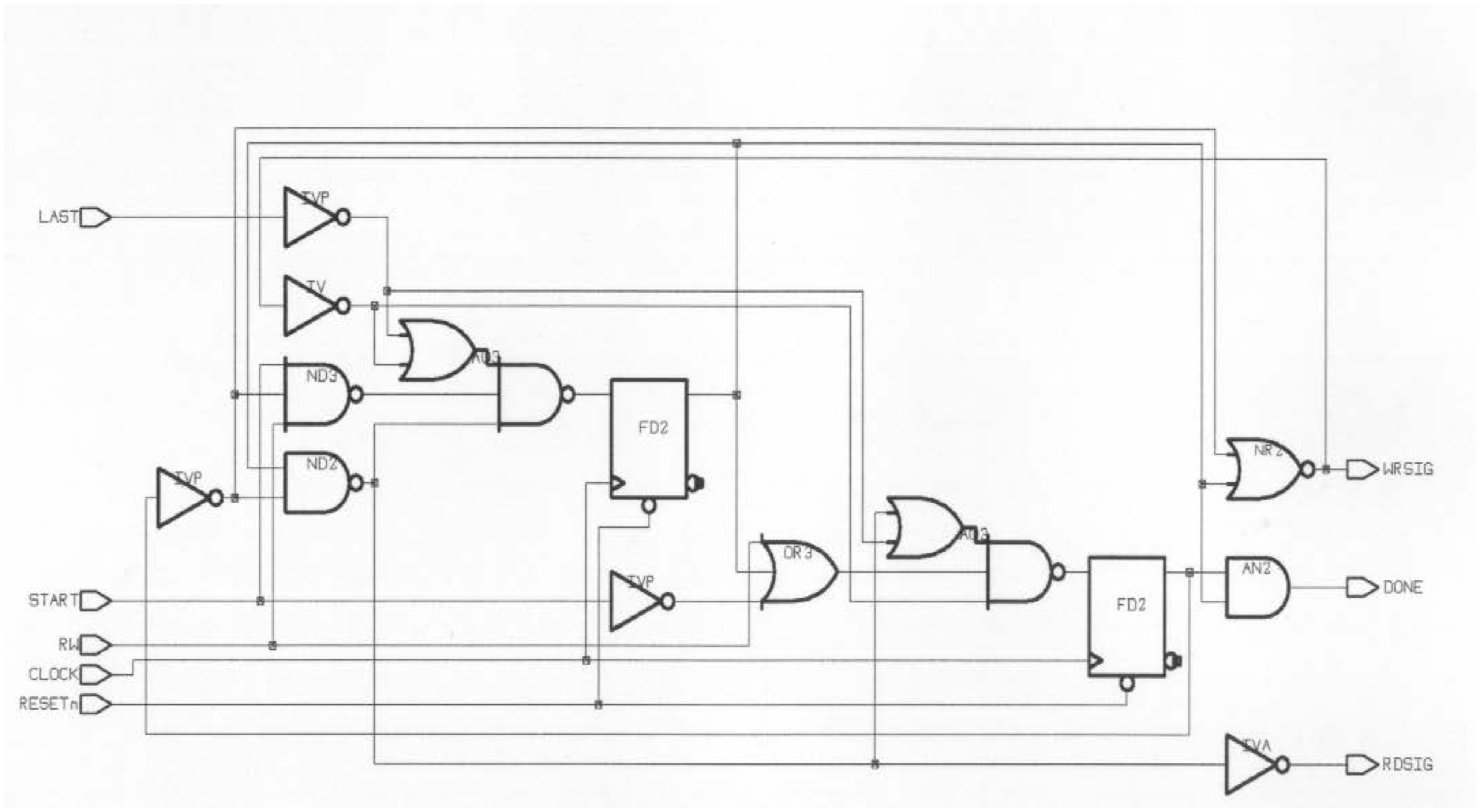


RWCNTL: Add Asynchronous Reset

```
seq : process
  begin
    if (RESETn = '0') then
       PR_STATE <= IDLE;
     elsif (CLOCK'event and CLOCK = '1') then
       PR_STATE <= NX_STATE;
     end if;
   end process;
end RTL;
```



Synthesized RWCNTL with Asynch. Reset



State Space Explosion

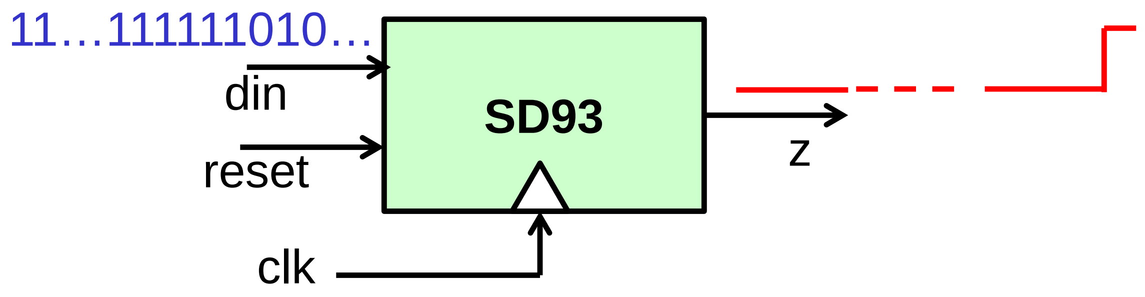
- Until now, we have considered state machines in which all possible states can be explicitly enumerated
  - e.g. stateA, stateB, etc.

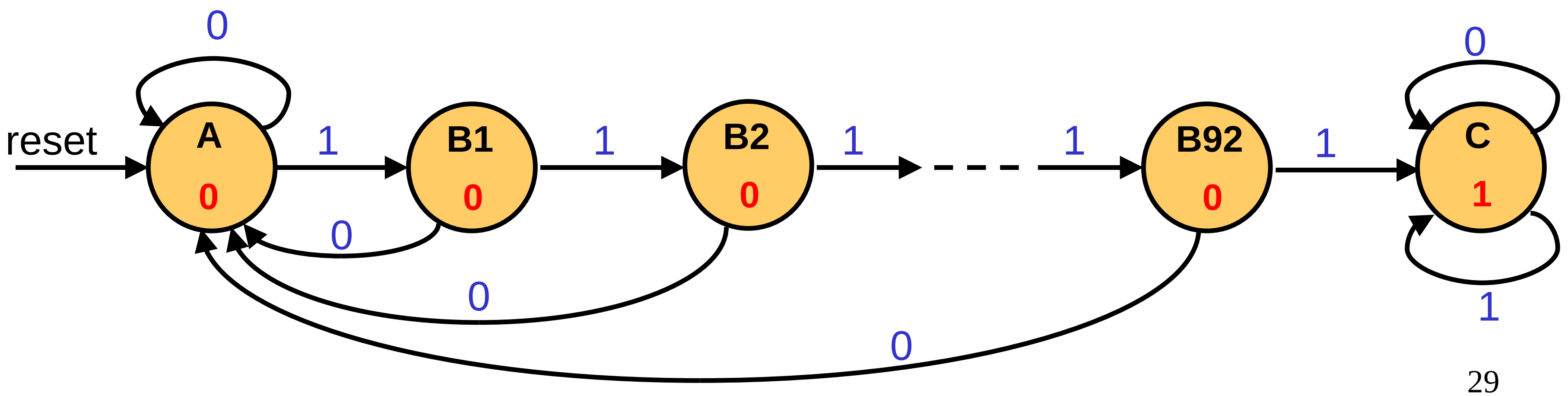
- Sometimes a FSM, in order to respond to a particular sequence of external events, will need to process and store some data – and the value of that data effectively becomes part of the state of the machine

- Because an n-bit data word can take on 2^n different values, this can lead to a state space explosion in which there are more states than can be explicitly enumerated

State Space Explosion - Example

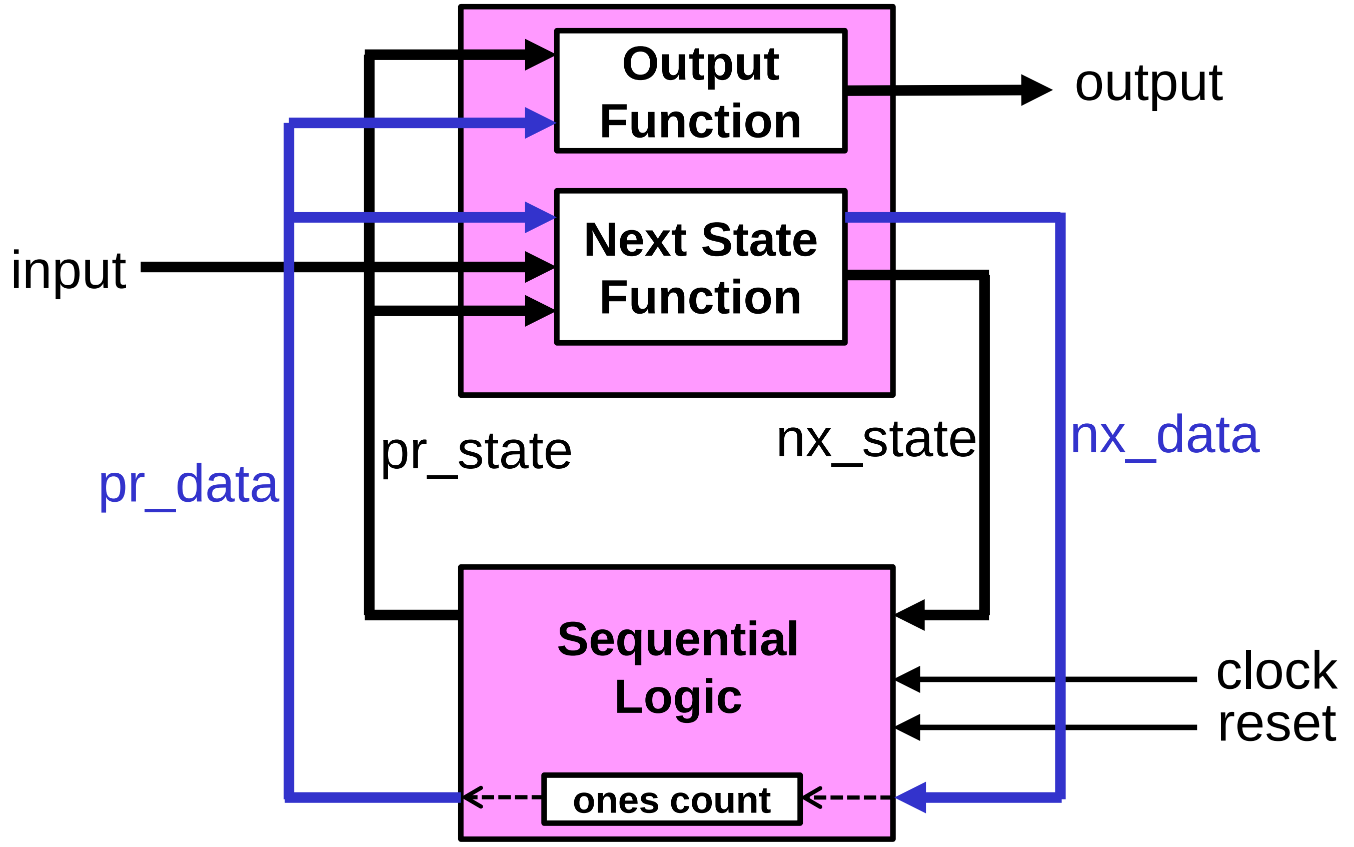
Build a Moore FSM that has looks for sequence of 93 successive ‘1’s in a serial input stream. When it detects the sequence, it outputs a 1 and holds that value until the machine is reset.





Augment State with Synchronous Data

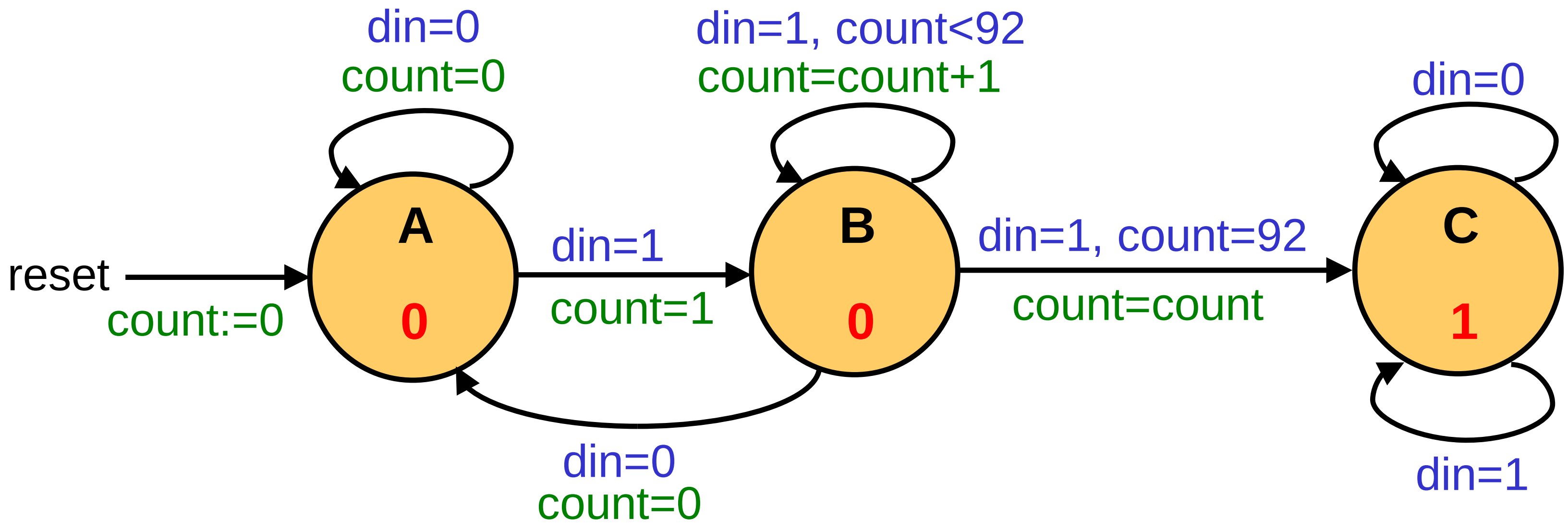
- A solution is to keep the ones count in a data word that effectively becomes part of the machine state



- Machine state is a combination of symbolic (pr_state) and numerical (pr_data) values

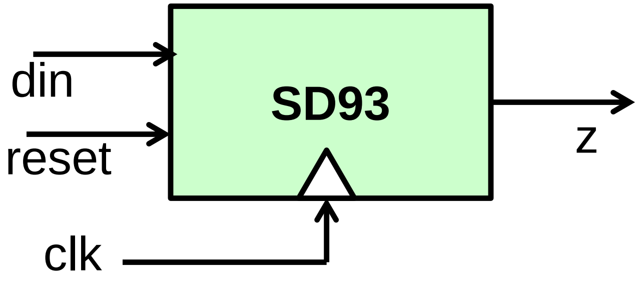
Augment State with Synchronous Data

- We have augmented the state with a computed data value



- “count” data is synchronously updated in clock process
- “count” becomes part of the machine state

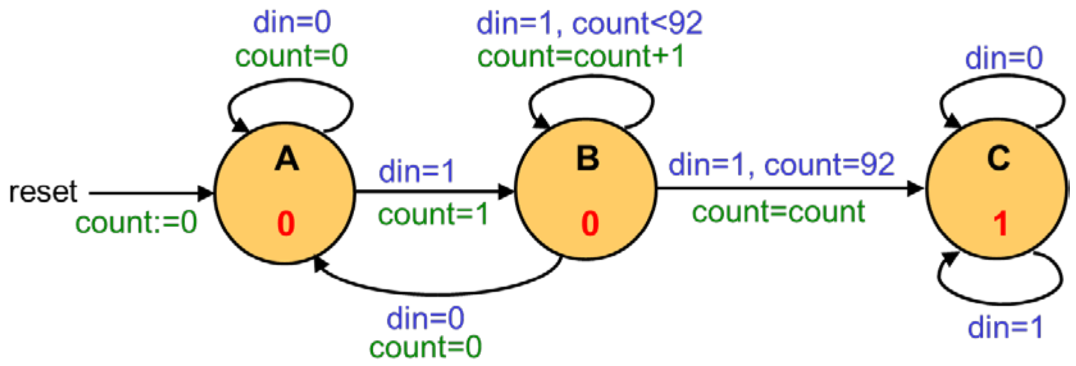
SD93 Sequence Detection: Clock Process



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

entity SD93 is
     port(din, reset,clk: in std_logic;
          z: out std_logic);
end entity SD93;

architecture moore of SD93 is
type state is (stateA, stateB, stateC);
signal pr_state, nx_state: state;
signal pr_count, nx_count: std_logic vector (6 downto 0);
begin
p_clk: process (rst, clk)
     begin
          if (reset = ‘1’) then
                    pr_state <= stateA;
                    pr_count <= (others=>’0’);
          elsif (clk’event and clk = ‘1’) then
               pr_state <= nx_state;
               pr_count <= nx_count;
          end if;
     end process;
```

SD93 Sequence Detection: Comb. Process

```
p_comb: process (din,  pr_state, pr_count)
     begin
          z <= '0'; nx_count <= pr_count;
          case pr_state is
          when stateA =>
               if din = '1' then
                    nx_state <= stateB;
                    nx_count <= "0000001";
               else
                    nx_state <= stateA;
                    nx_count <= (others=> '0')
               end if;
          when stateB =>
               if din = '1' and pr_count >= 92  then
                    nx_state <= stateC;
               elsif din = '1'  then
                    nx_state <= stateB;
                    nx_count <= pr_count + 1;
               else
                    nx_state <= stateA;
                    nx_count <= (others => '0');
               end if;
          when stateC =>
               z <= '1';
               nx_state <= stateC;
          end case;
          end process;
```



Example: Another Sequence Detector

- You are required to design a circuit that takes as input a serial bit stream and outputs a ‘1’ whenever there are two successive transitions, i.e. whenever the sequences 101 or 010 occur. Overlaps must be considered. For example:

Input: …11010011…

Output: …00011000…

Draw the Mealy and Moore FSM state diagrams.