N26F300 VLSI SYSTEM DESIGN (GRADUATE LEVEL)

Fall 2010

FSM and Controller

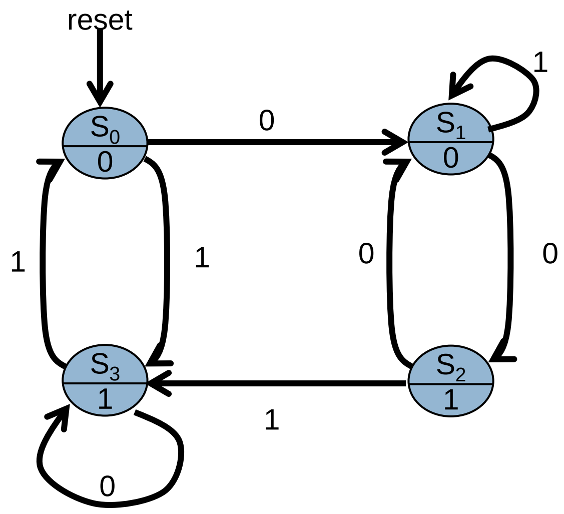
Outline

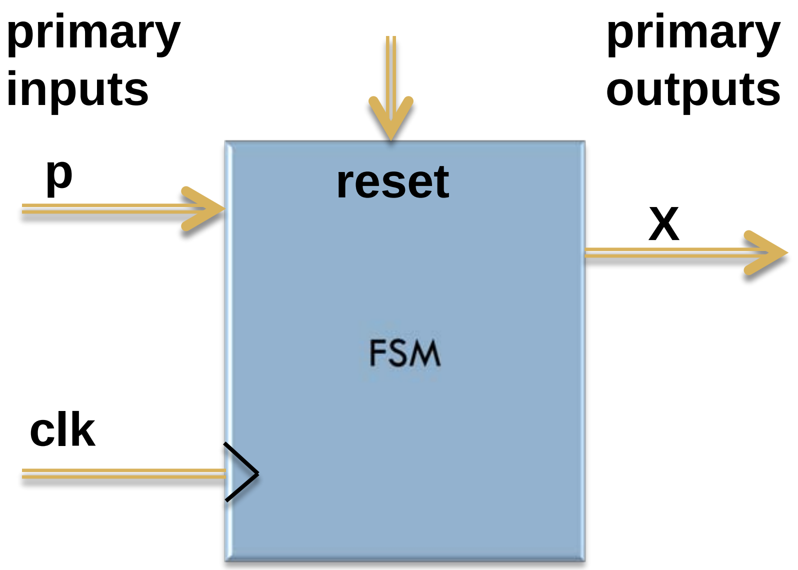
- Moore & Mealy Revisited
- Examples of FSM
- Control external hardwares
  - Controller for timer, ADC and memory access
  - WatchDog Timer
  - DMAC

[Material adapted from “FSM based Digital Design Using Verilog HDL”by Minns and Elliott]

Finite State Machine (FSM)

- A digital sequential block controlled by one or more inputs with predefined finite states. The machine can move from one state to another state.





Synchronous FSM

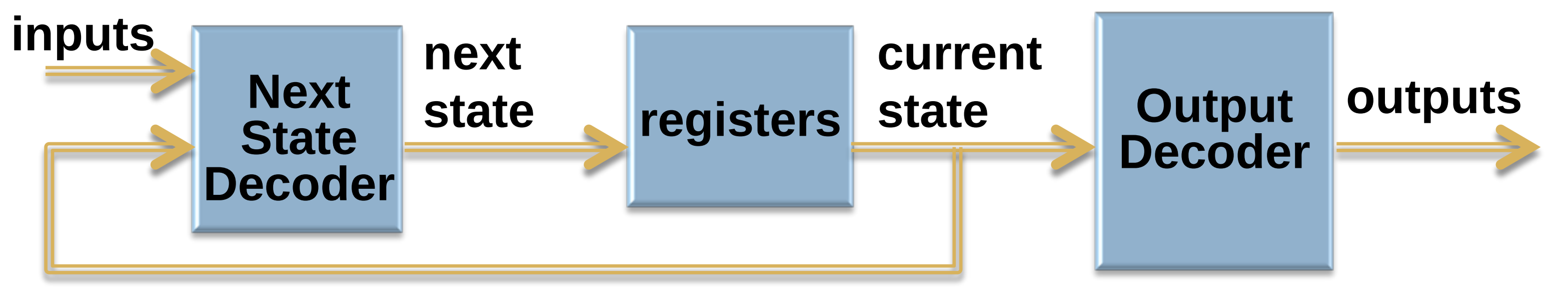
- An FSM that can only change states only if a clock pulse occurs.
- States can be identified by using a number of flip-flops inside the FSM block.

$$\# \text{ of states} = 2^{\text{Number of FFs}}$$

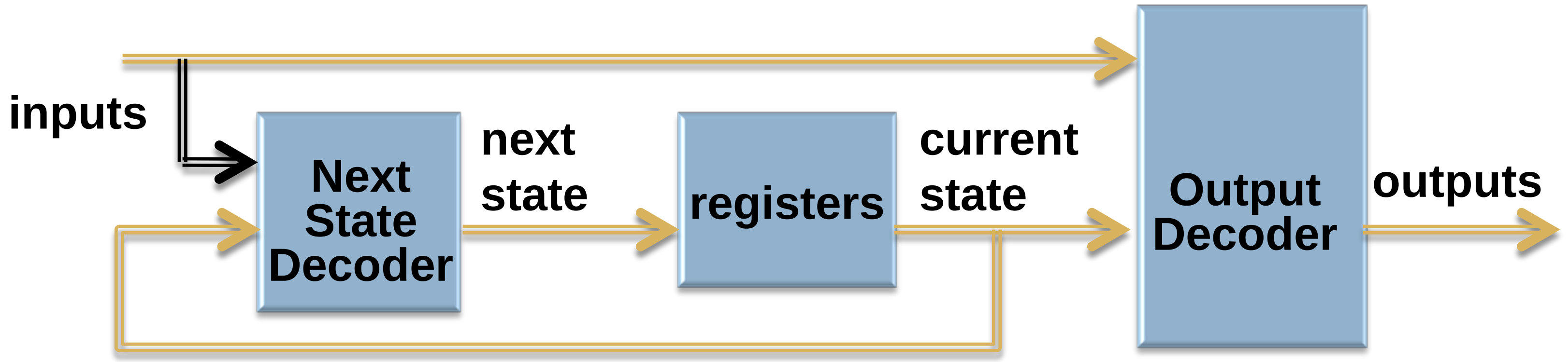
$$\Rightarrow \quad \# \text{ of FFs} = \log_2(\# \text{ of states})$$

Moore and Mealy Machines

- Moore model



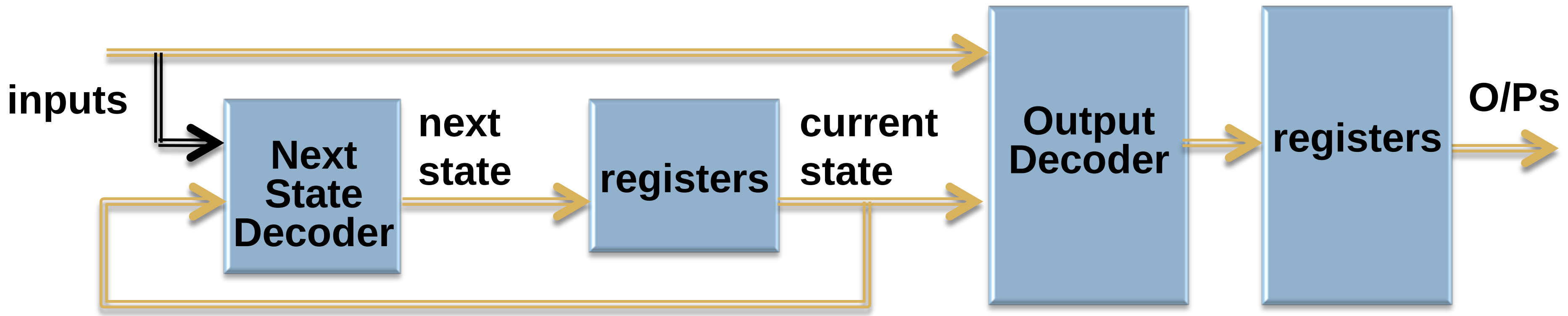
- Mealy model



Mealy vs. Moore models

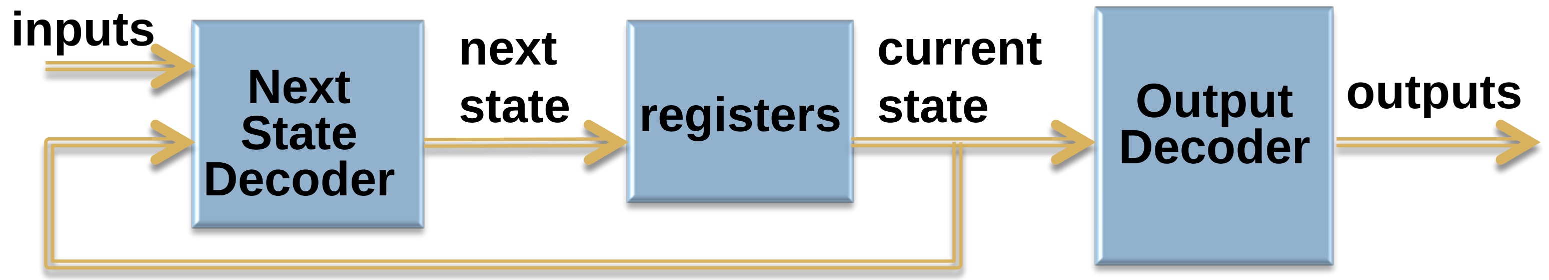
- Moore is safer to use
  - Outputs change at clock edge
  - Not like Mealy, output follow input in asynchronous way
-
  - Output = function of inputs and the present state
  - Not like Moore, output depends only on the present state
-
  - Complete in the same cycle
  - Not like Moore, more logic may be needed to decode state into outputs

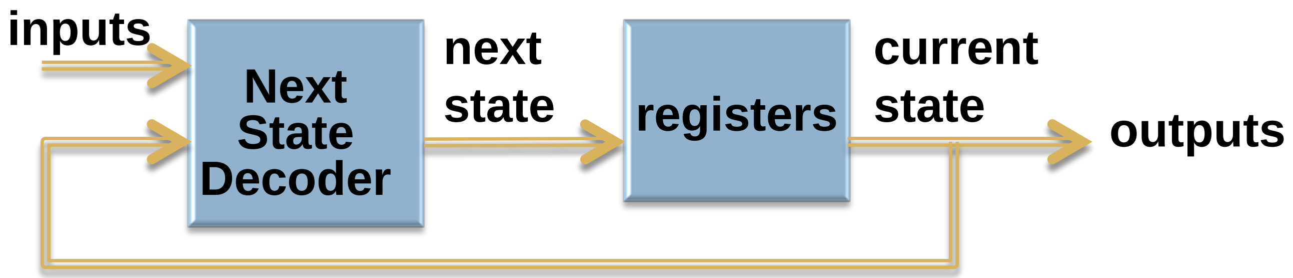
Synchronous Mealy Model



- Synchronous Mealy machines avoid the potential glitches and change of outputs asynchronously

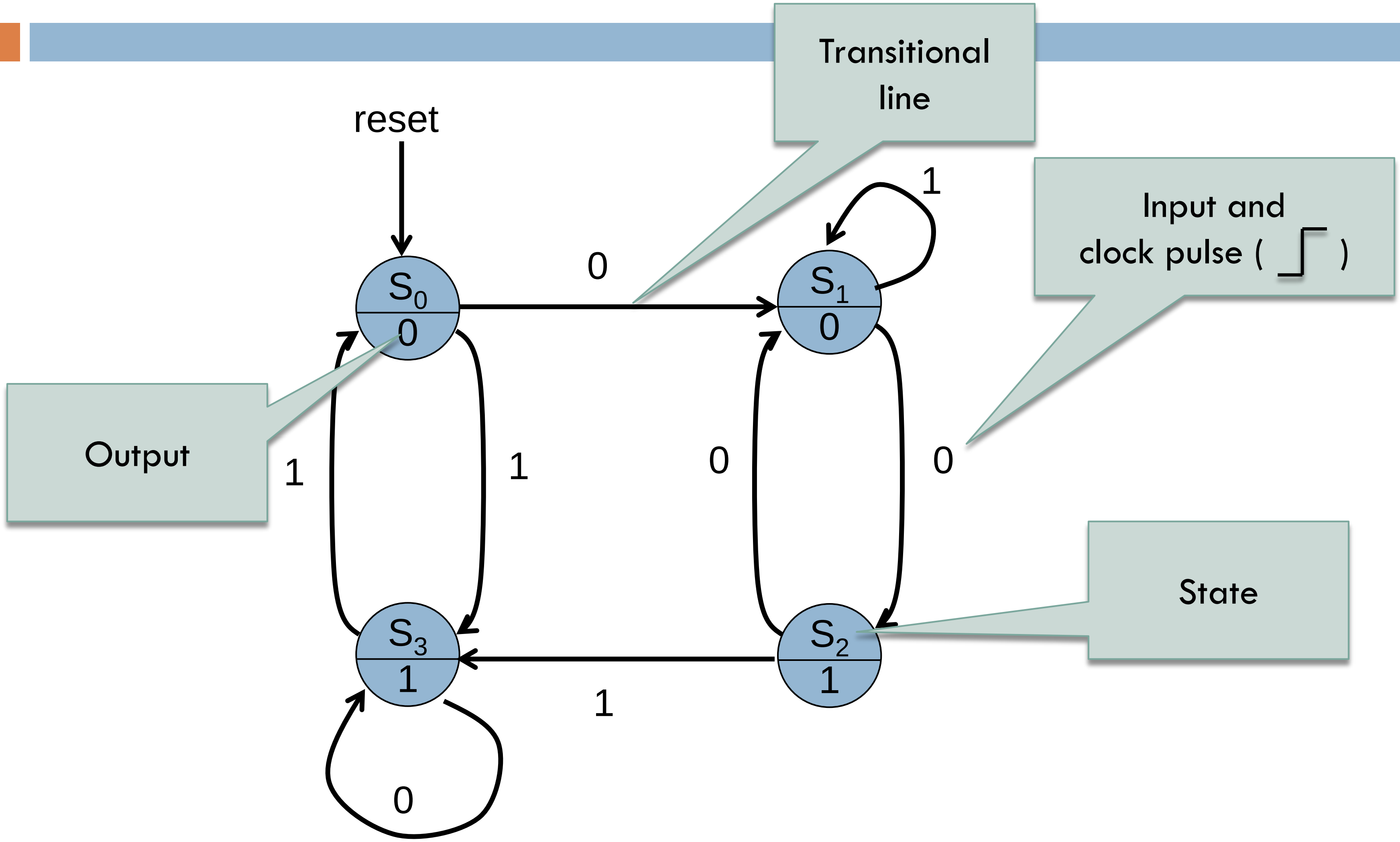
Moore Machines





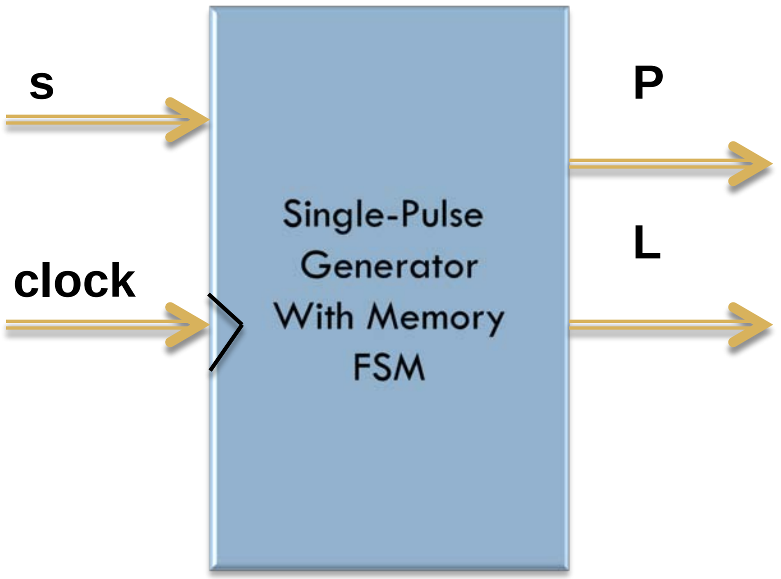
One of basic forms of many asynchronous counters

State Transition Graph (STG)

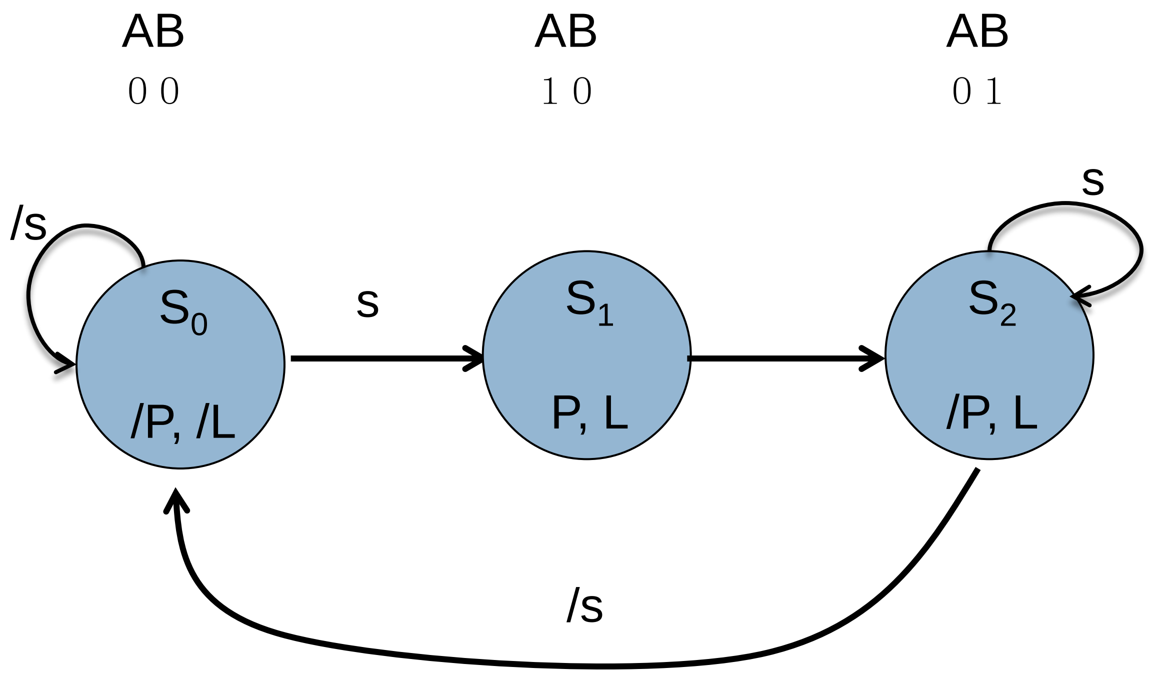


EXM1: A Single-Pulse Generator Circuit with Memory (SPGM)

- Problem:
  - When input s = 1,
    - produce a single output pulse at the output P
    - Set output L to 1
  - When output s = 0,
    - Clear output L to zero
  - L: memory indicator

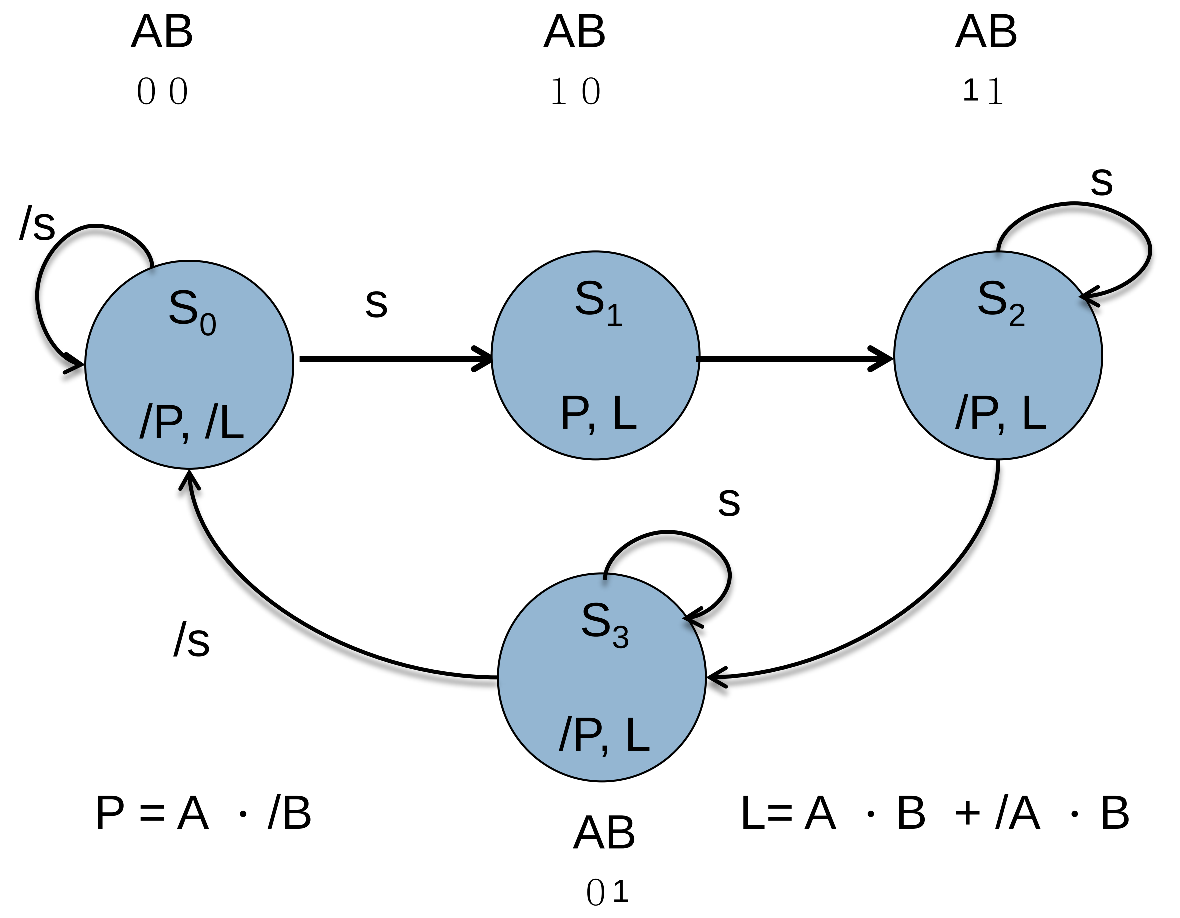


STG for SPGM (SPGM-1)



$P = A \cdot /B$ $\qquad$ $L = A \cdot /B + /A \cdot B$

STG for SPGM complying with Unit Distance Pattern (SPGM-2)

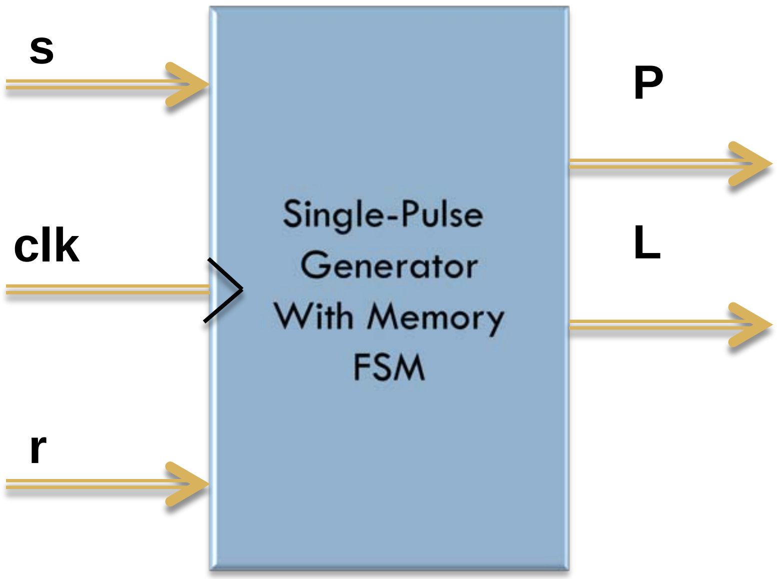


$P = A \cdot /B$

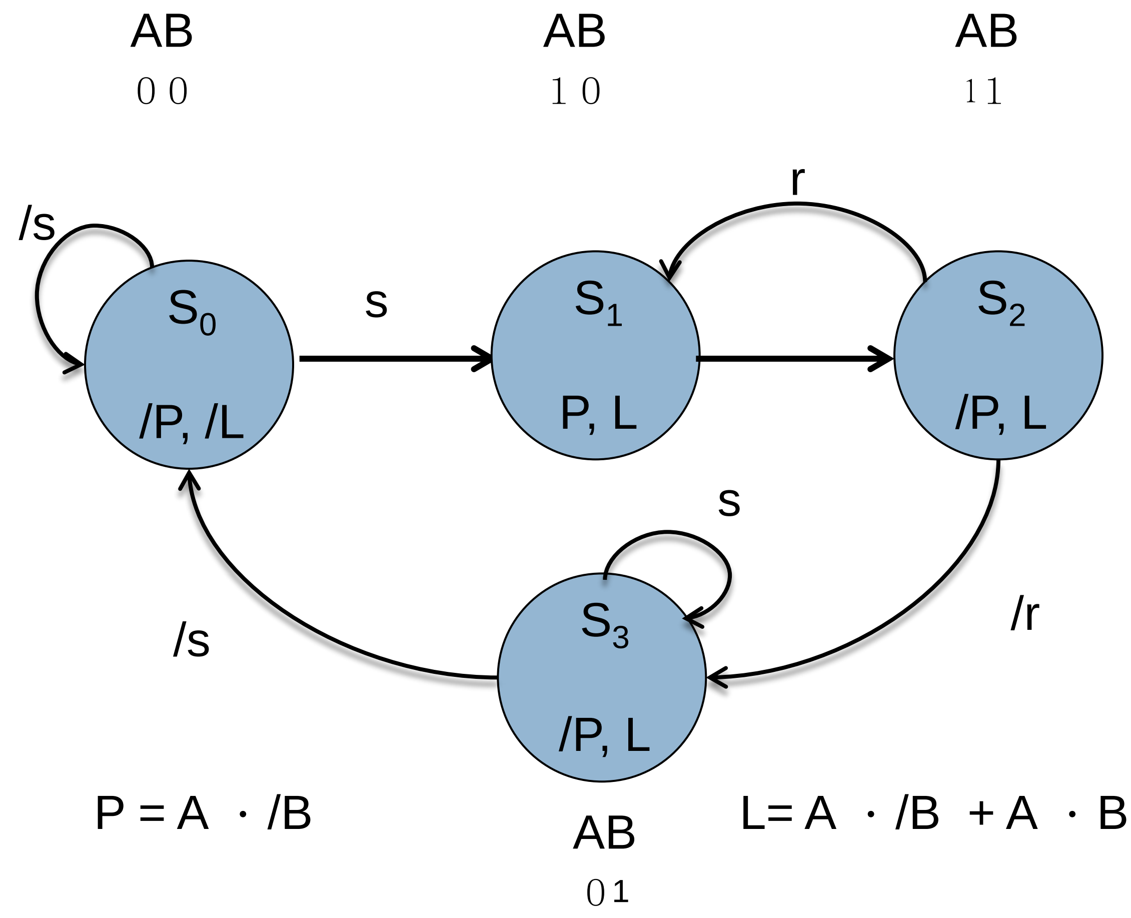
$L = A \cdot B + /A \cdot B$

EXM2: A Single-Pulse Generator Circuit with Memory (SPGM-3)

- Problem:
  - When input s = 1,
    - produce a single output pulse at the output P
    - Set output L to 1
  - When output s = 0,
    - Clear output L to zero
  - When input r =1,
    - Let P = clk
  - When input r =0,
    - Resume its single pulse



STG for SPGM complying with Unit Distance Pattern (SPGM-3MR)



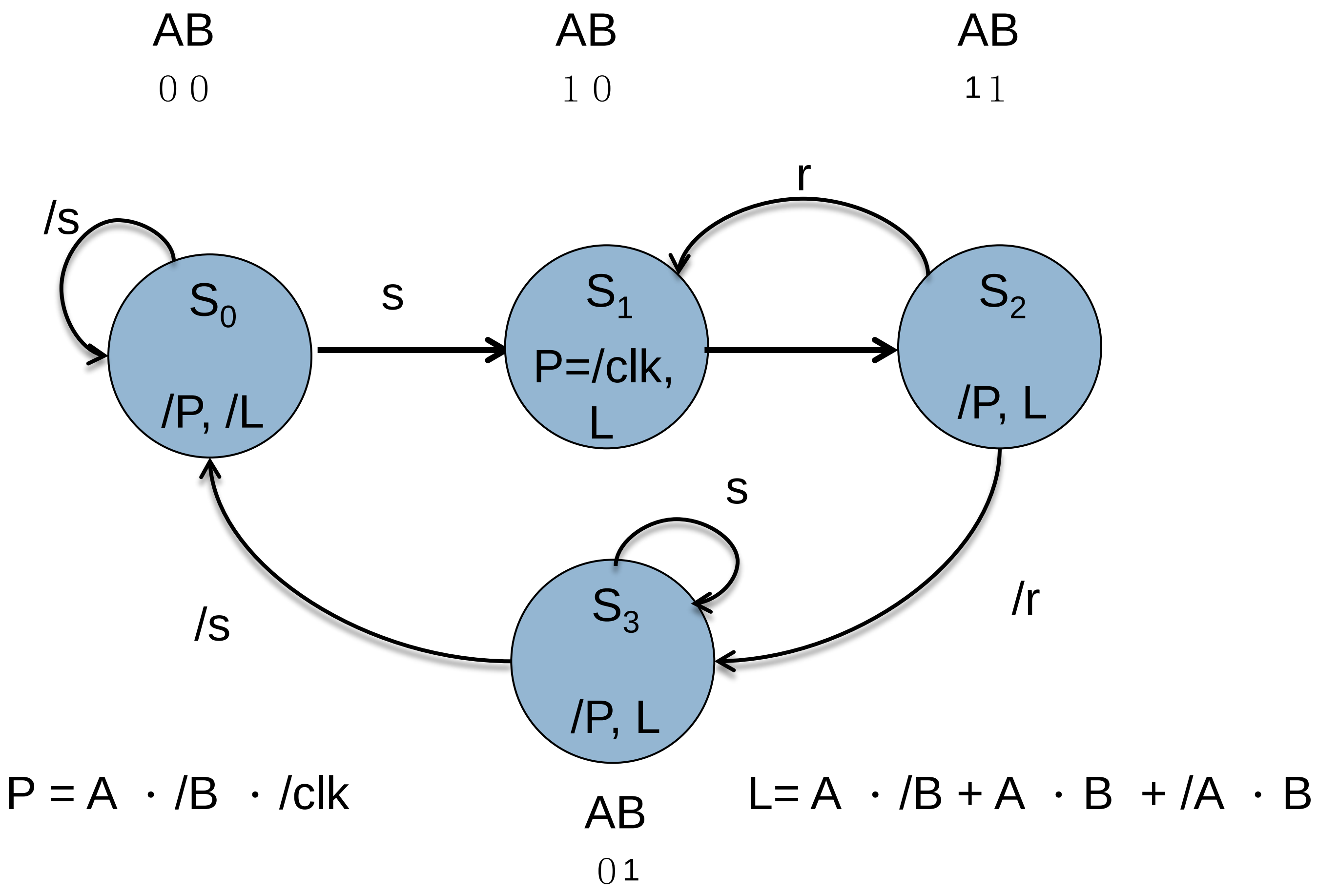
P = A · /B

L= A · /B + A · B + /A · B

Moore to Mealy

- Make the output P depend on the state1 as well as clk
- That is to say a direct control path from the input to the output

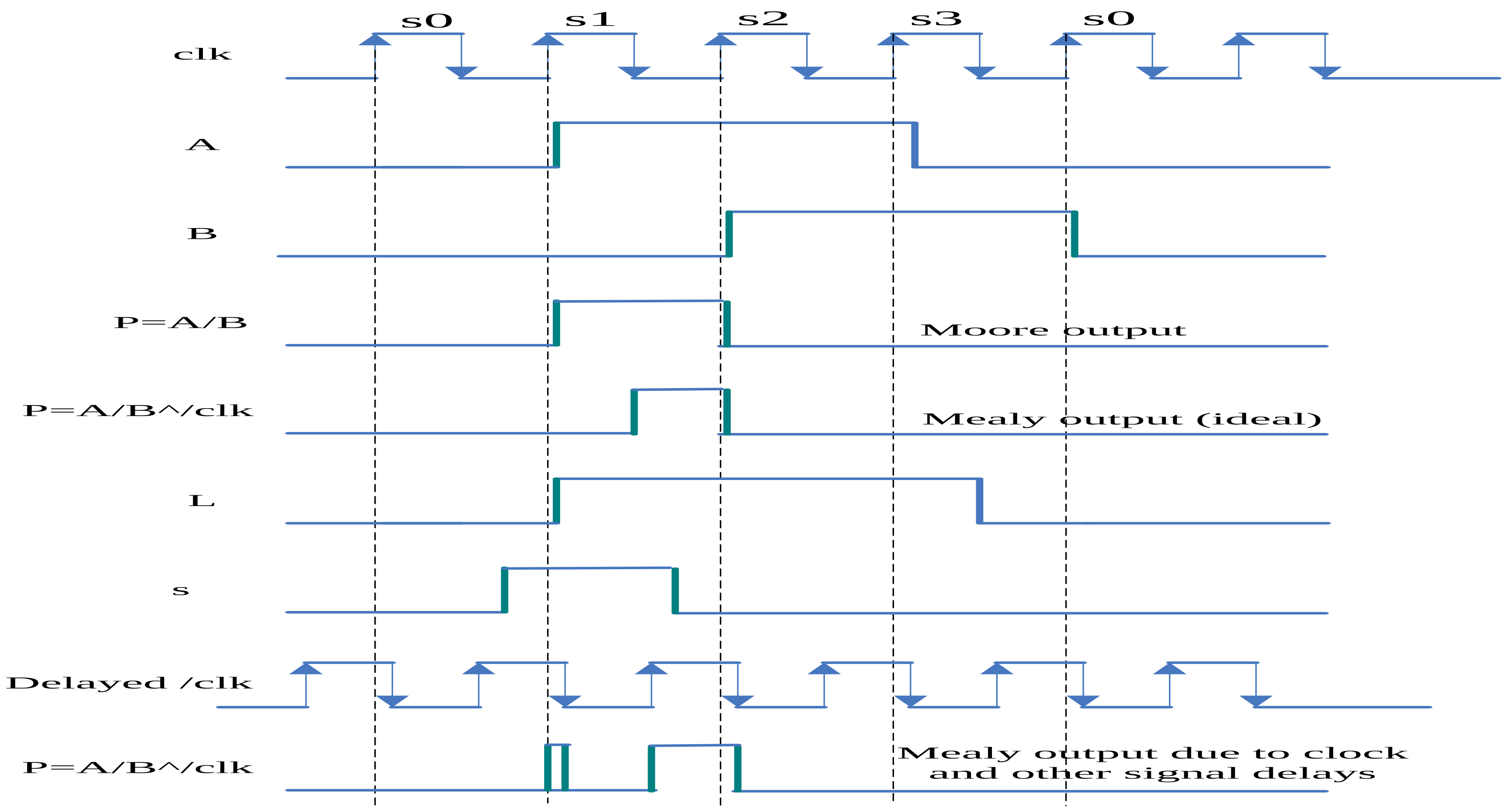
SPGM Using Mealy Model (SPGM-3ML)



$P = A \cdot /B \cdot /clk$

$L = A \cdot /B + A \cdot B + /A \cdot B$

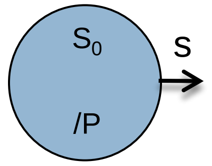
Waveform of Moore and Meanly



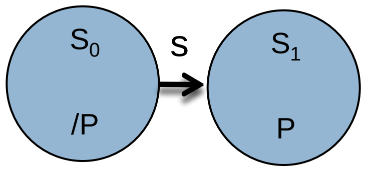
EXM3: 101 Pattern Generator

- Problem
  - Produce a state diagram for an FSM that will generate 101 pattern when input *s* goes high. The input *s* must be returned low before another 101 pattern can be produced.

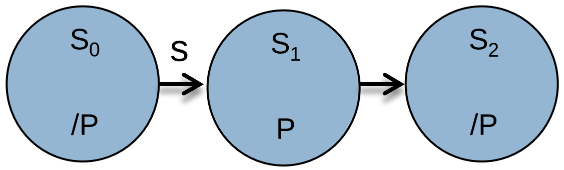
Development of 101 PG



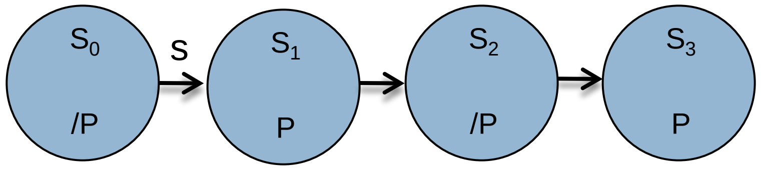
Wait for s goes high to start sequence



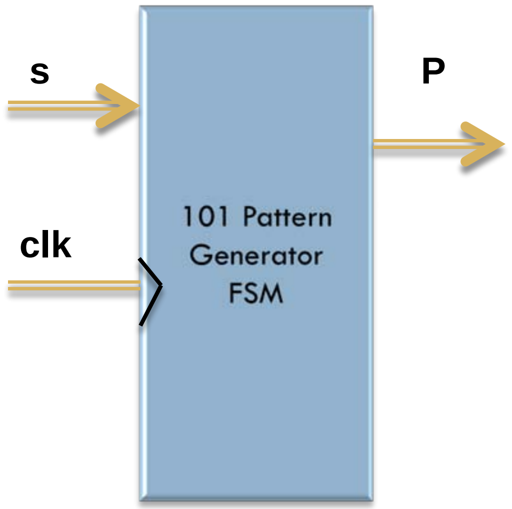
When s=1 move to s1 and raise P



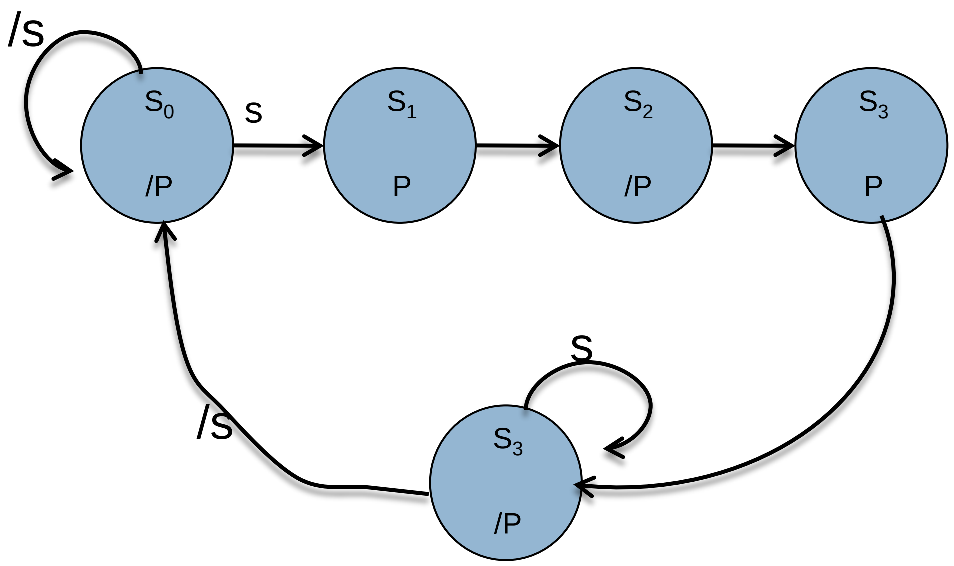
On next clock pulse, move to s2 and lower P



On next clock pulse, move to s3 and raise P

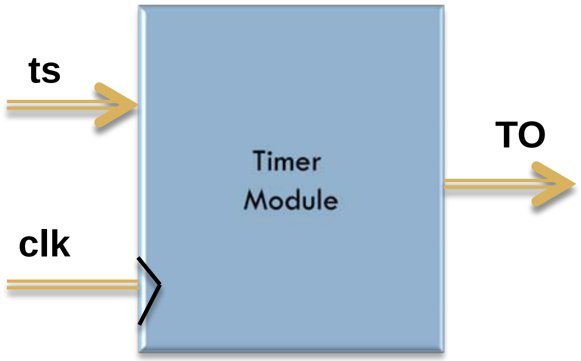


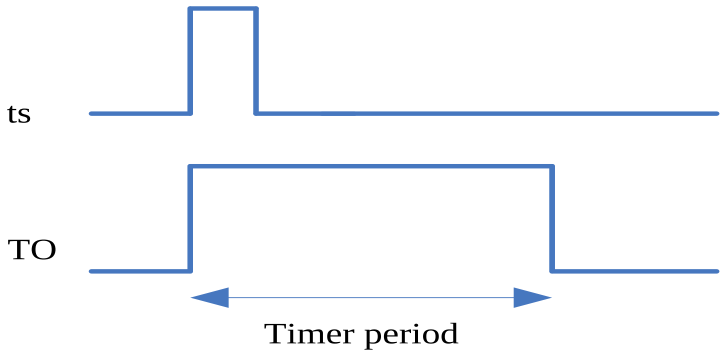
Complete State Diagram



Implementation of Wait

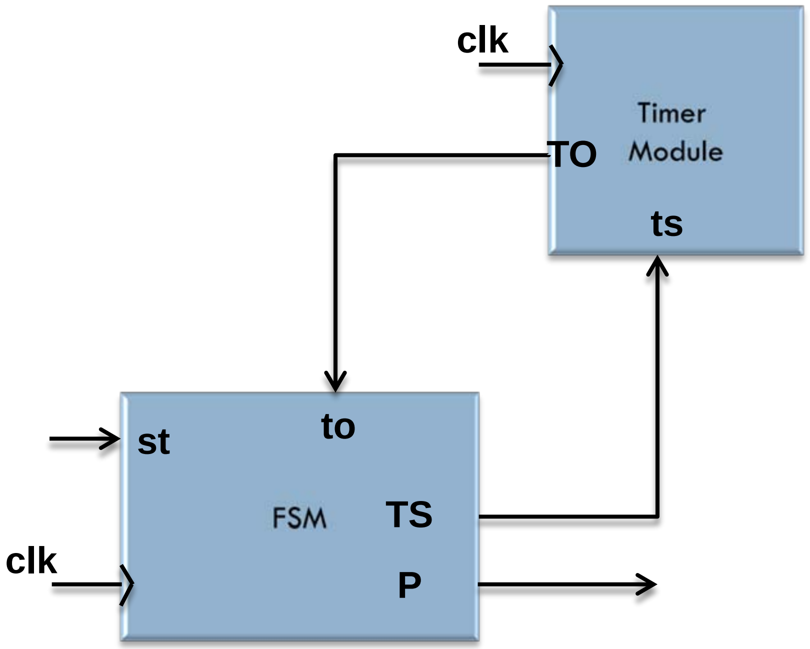
- How to let a FSM to wait in a state for predefined period?
  - Allocate a number of consecutive states
  - Use an external timer unit that can be controlled by the FSM

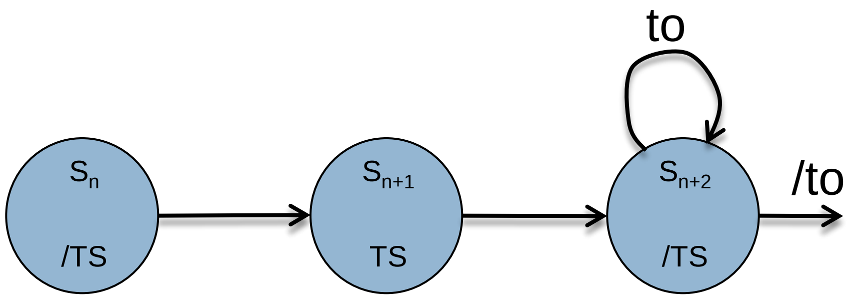




Timer Unit and FSM

- State sequence to control the timing module



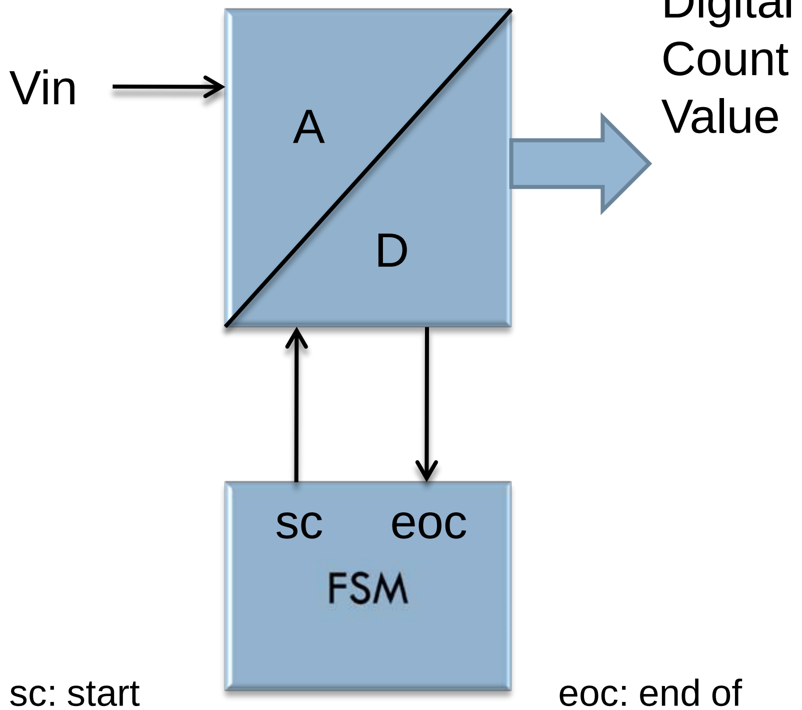


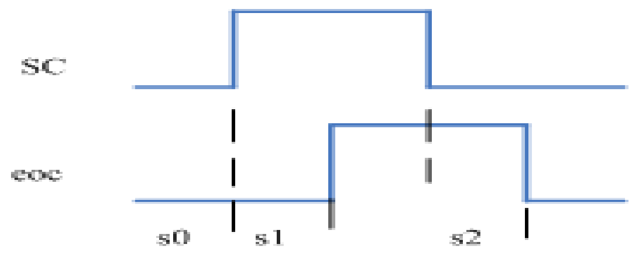
Prior to starting timer

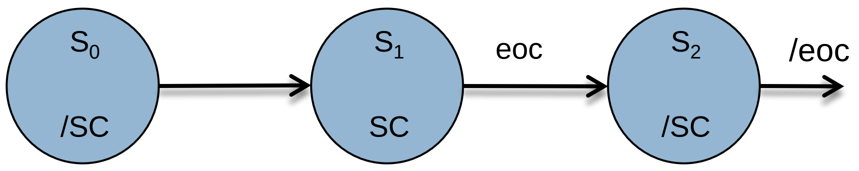
Start the timer

Time-out state wait here until timer times out

Controlling an Analog-to-Digital Conversion (ADC)







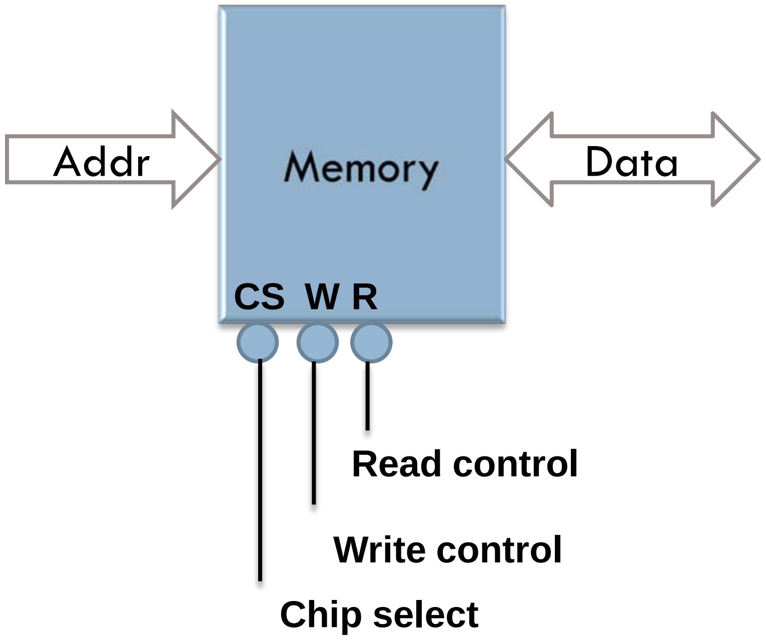
sc: start conversion

eoc: end of conversion

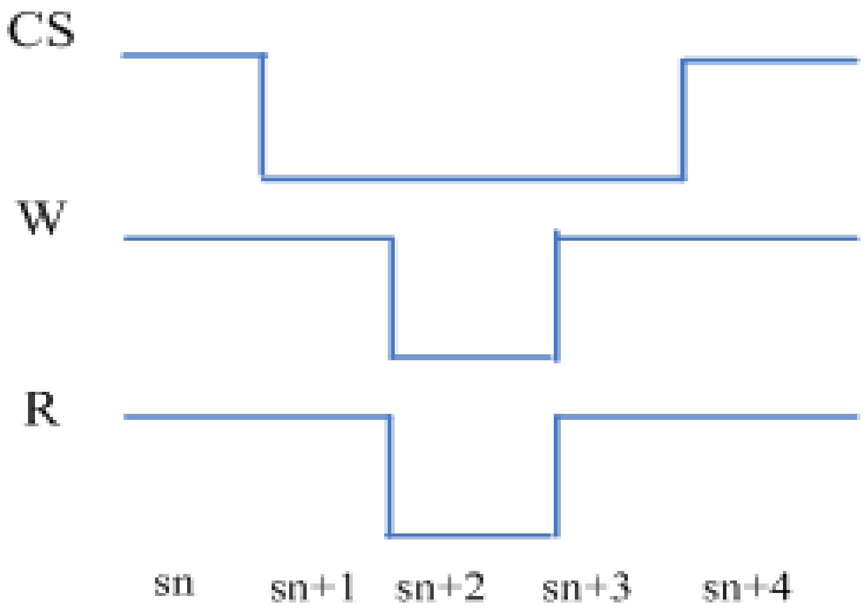
Generic Memory Device

I/O

R

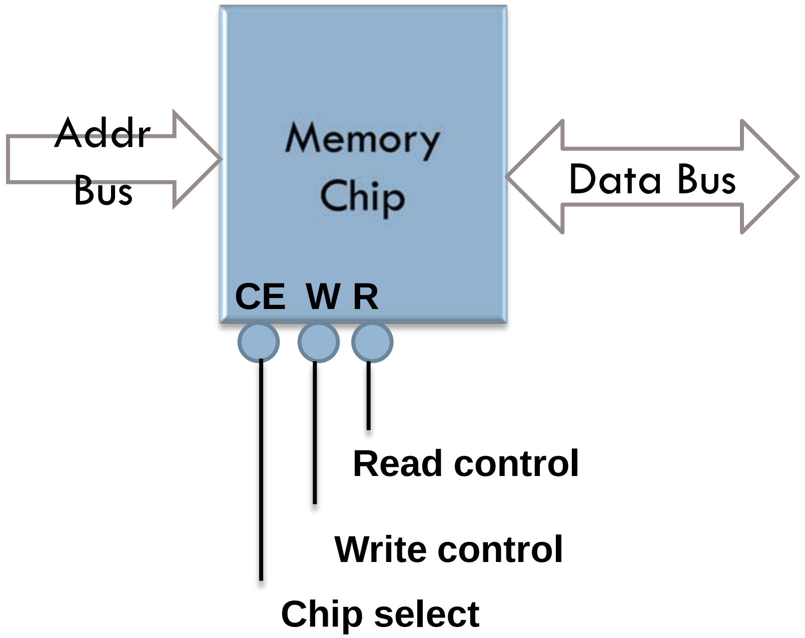


Memory Timing

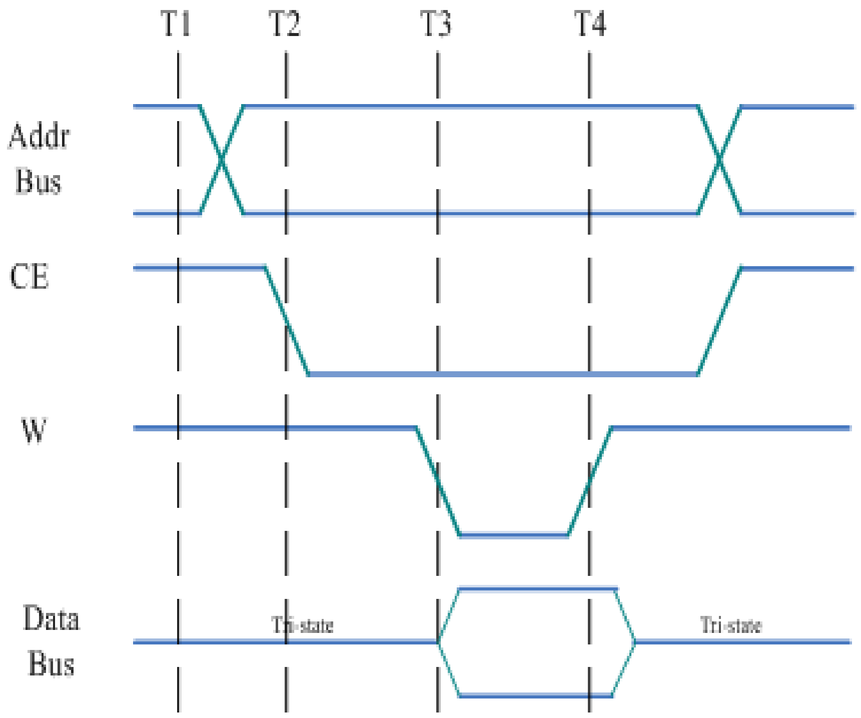


Timing Waveform of a Memory Device

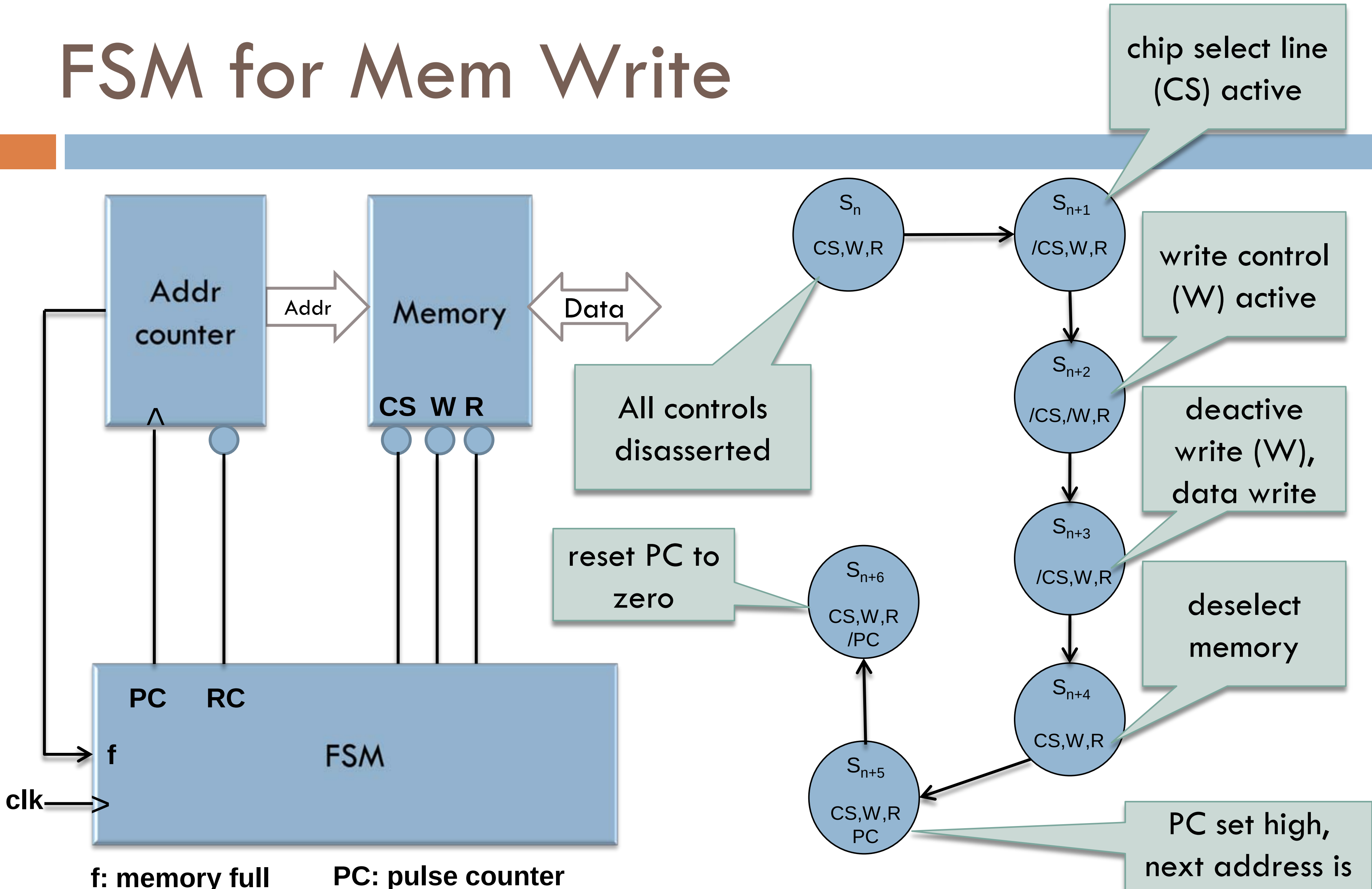
I/O



Memory Timing



FSM for Mem Write

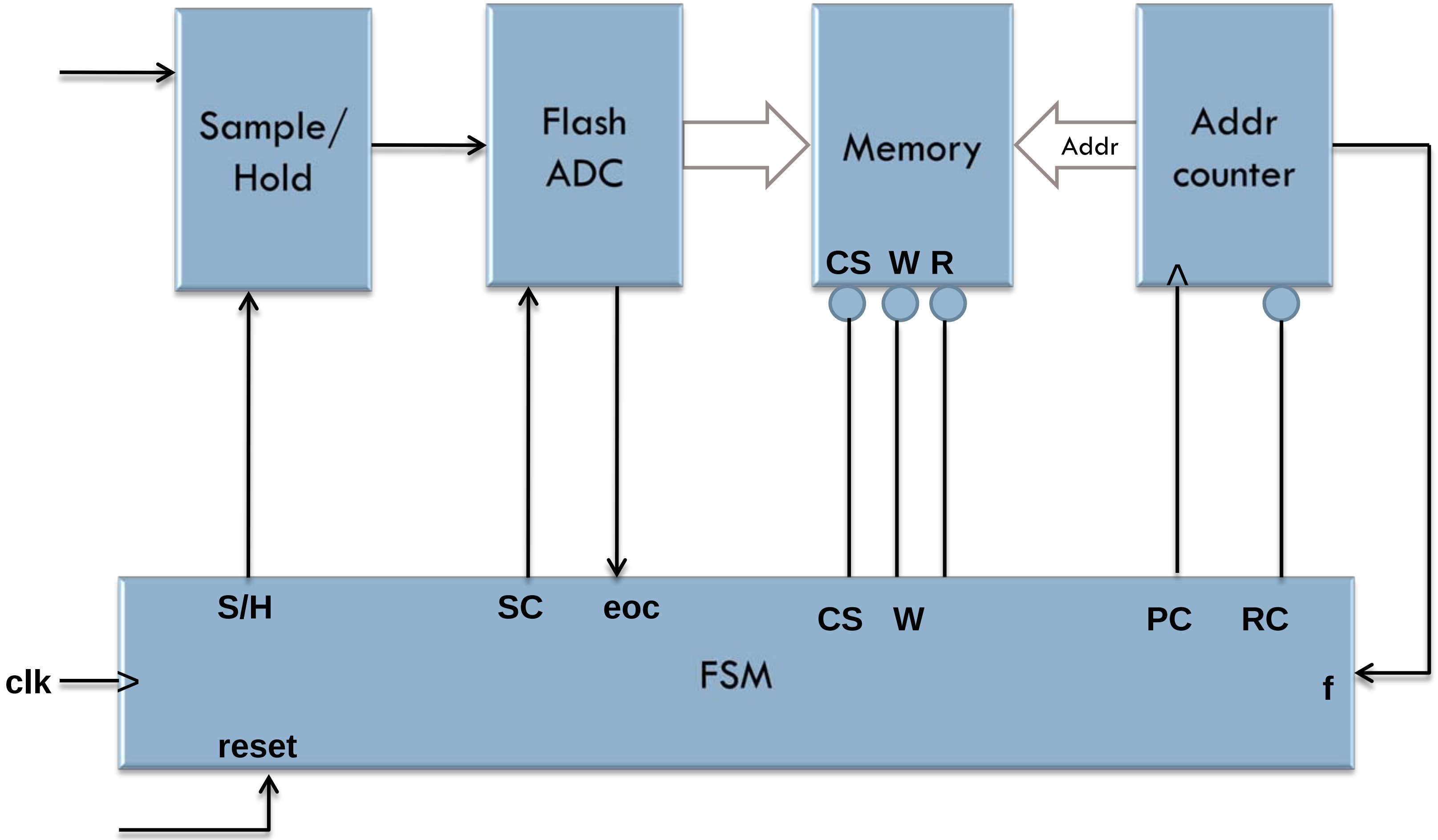


f: memory full

PC: pulse counter

RC: Reset counter

Small Data Acquisition System (DAS)

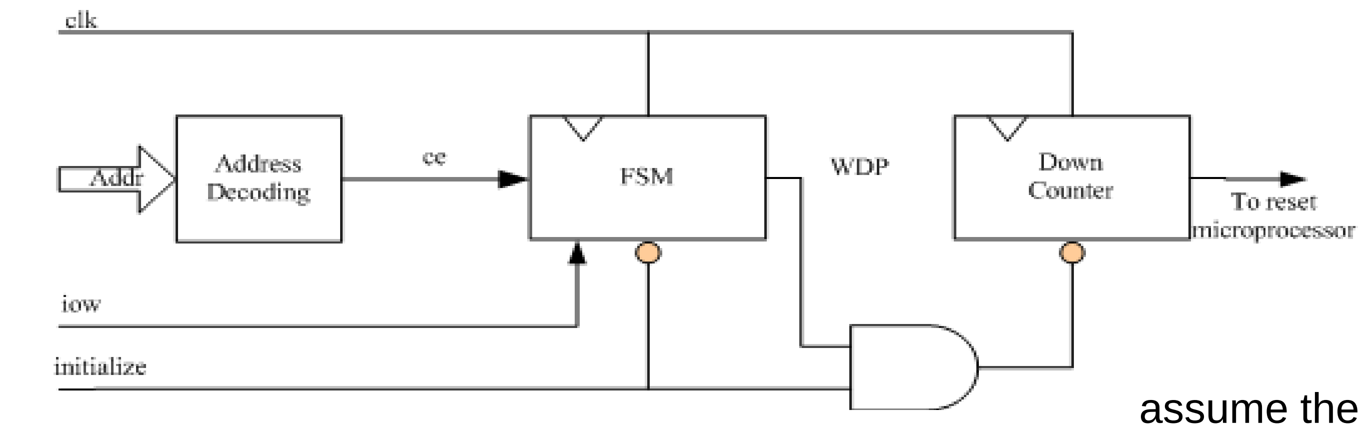


Exercise!

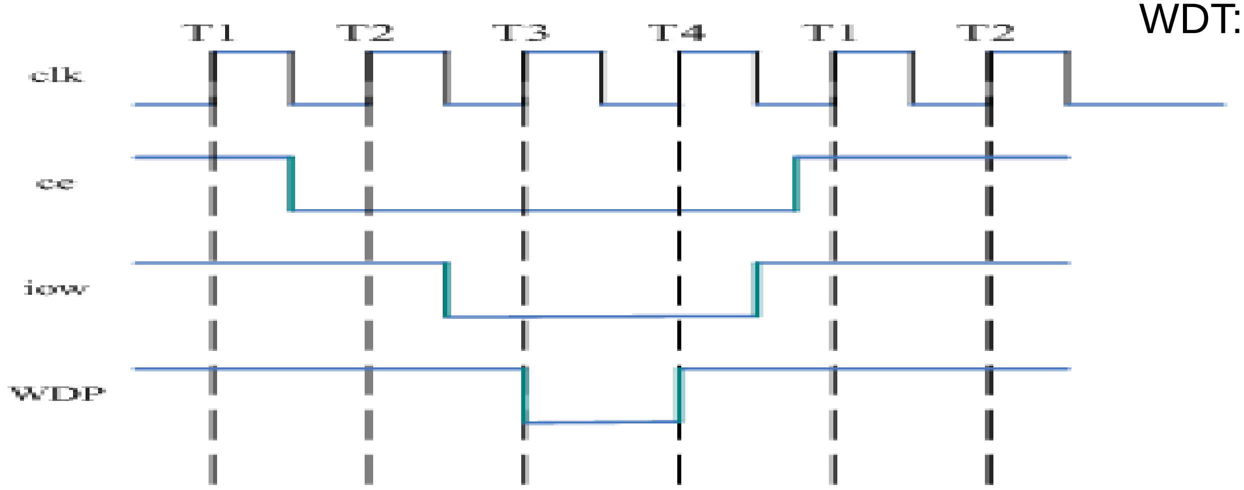
Clocked Watchdog Timer (WDT)

- Addressable device that can be written to on a regular basis
- Regularly reinitialize to a known count value
- If the μcontroller does not write to the WDT between counting-down period and WDT reach zero, the μcontroller will be reset.

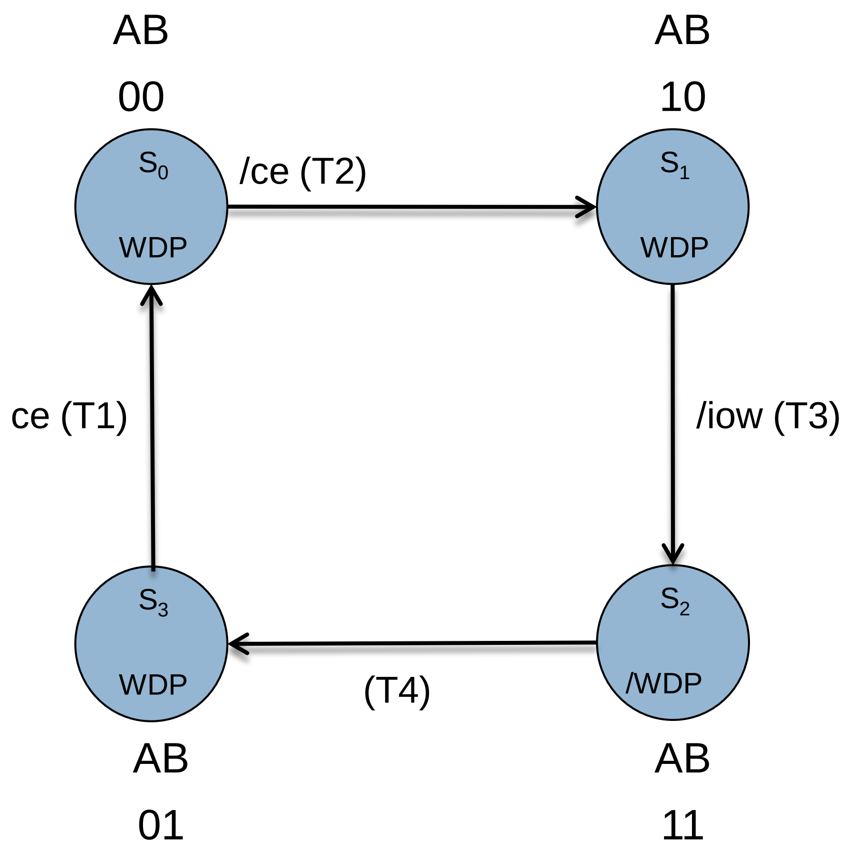
Block Diagram for a WDT



assume the address of WDT: 300h



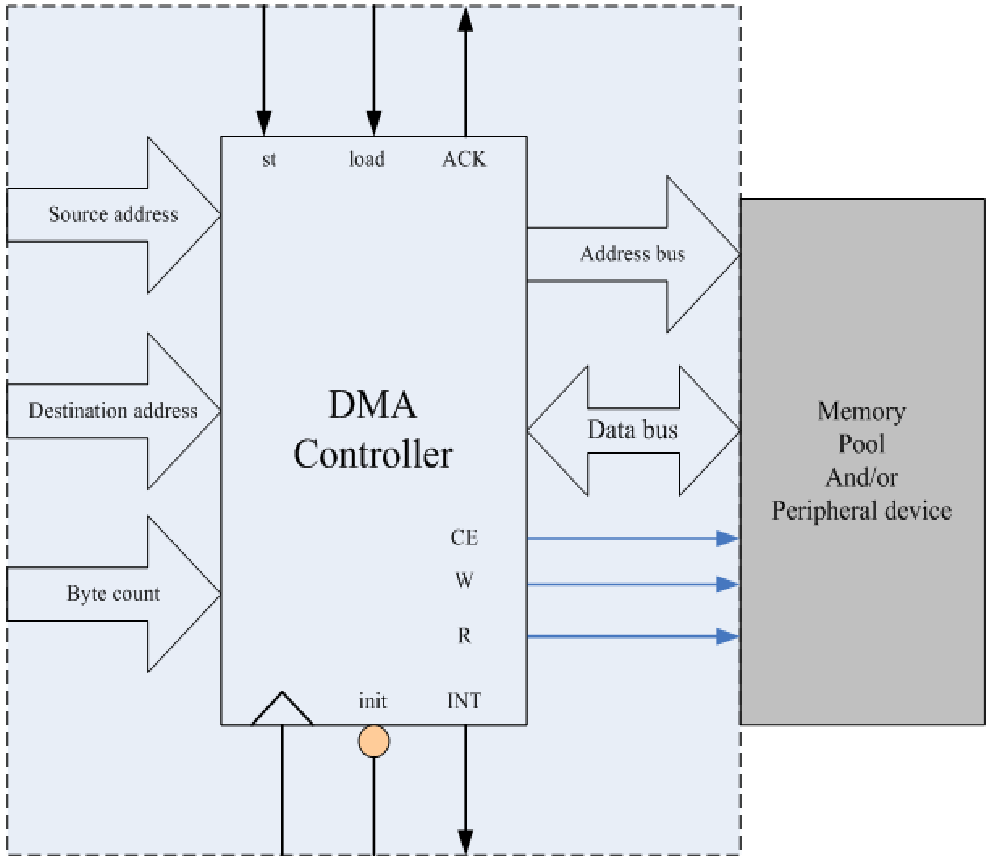
State Diagram for the WDT



One Hot Technique

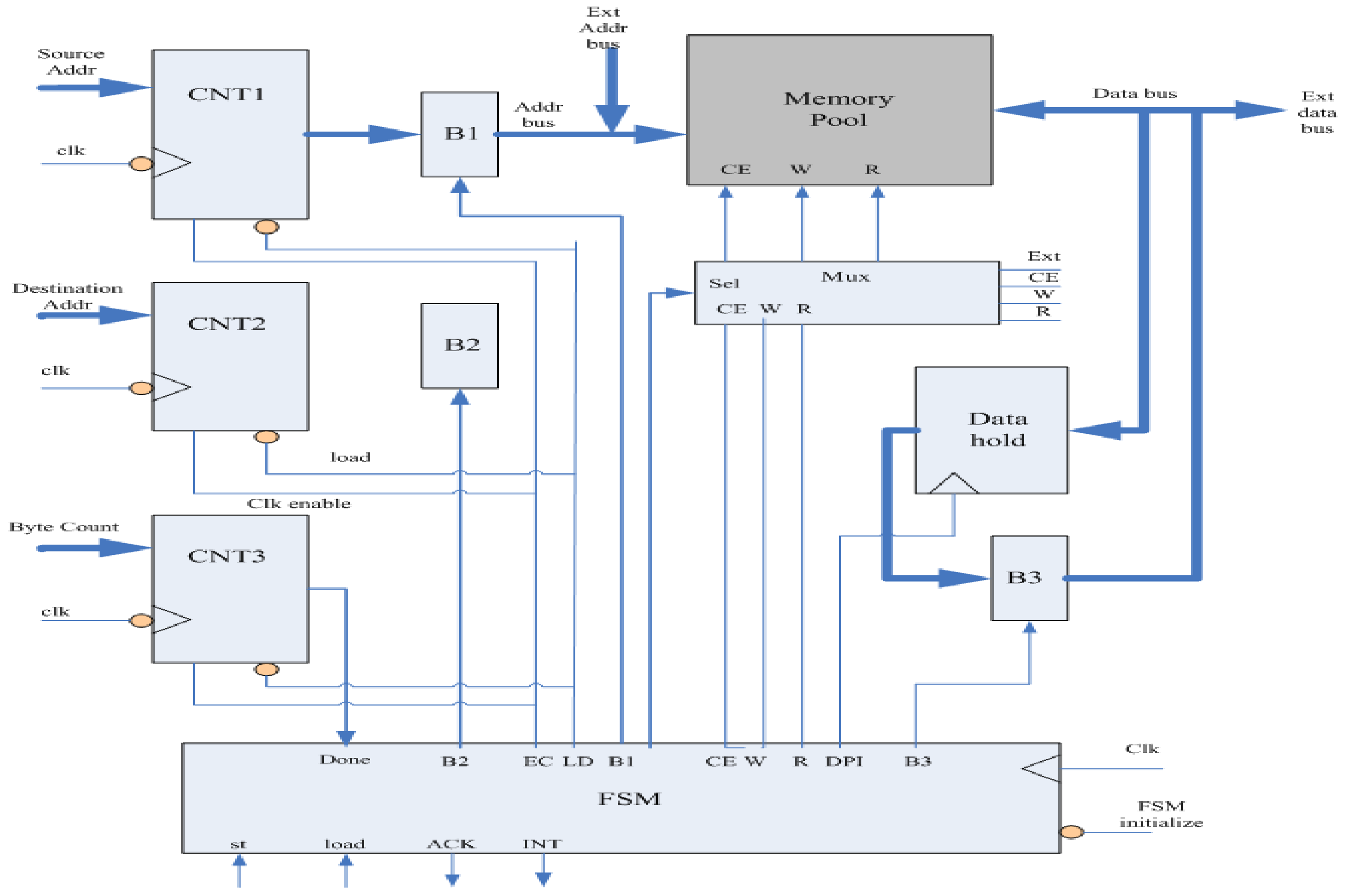
- Assign a flip-flop for each state
- Disadvantage: wasteful if the number of states is large
- Advantage:
  - in theory avoid the generation of output glitches
  - require fewer logic levels
- Often use in FPGAs since its architecture consists of many cells that can be programmed to be FFs

DMA Controller (DMAC)



- The DMAC can share the loading of the microprocessor
- Allow data to be move from one part of the memory system to another or to a peripheral device

Possible Detailed Block Diagram



General Steps

- Start (st) DMA
- Accept source, destination, and words/byte to be transferred
- Interrupt the microprocessor to let it know it is to take over the memory/peripheral
- Microprocessor isolates itself from these devices and send the load signal to DMA

Transactions

1. Select the source address and read its contents into a buffer
2. Select the destination address and deposit the buffer content into this address
3. Decrement the byte counter and advance the source & destination counter
4. Repeat 1 to 3 until all data transactions are complete (i.e., byte counter → 0)

Steps for FSM

1. Wait for the start signal st
2. Provide an interrupt to the µP to get it isolate itself from the memory
3. Wait for a load signal from the µP; when obtained, loading the source, destination, and byte count into the relevant counters
4. Source memory needs to be selected and data read from the memory into data holding registers

Steps for FSM

5. Source address needs to be isolated from the memory and the destination memory selected
6. Data in the holding register needs to be transferred into the output buffer B3 and store into the memory destination address

Steps for FSM

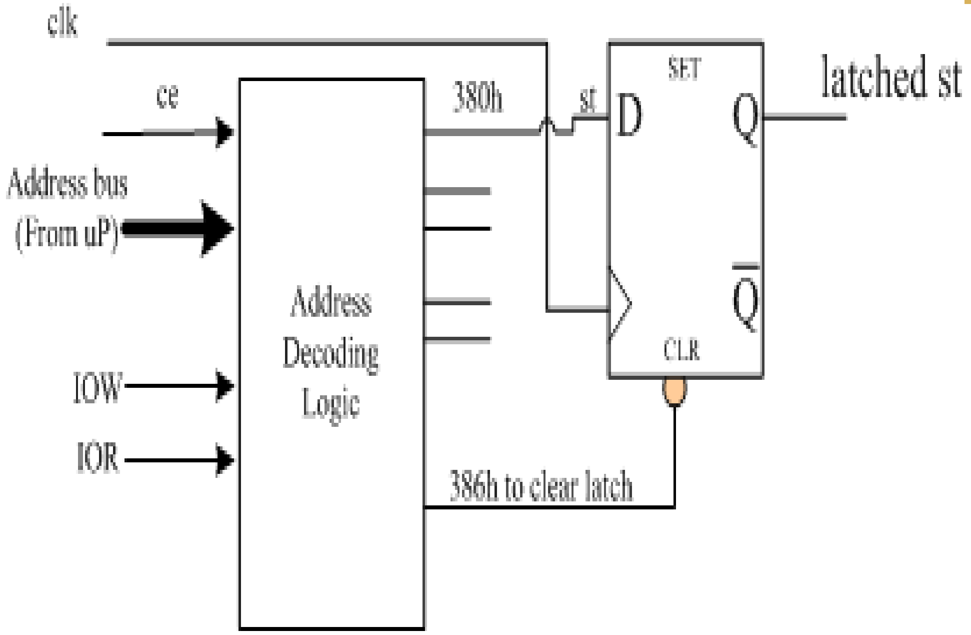
7. Decrement byte counter and checked if all bytes of data have been transferred
8. If there are more bytes to transfer, repeat 1 to 7 again. This is to continue until all bytes are transferred

Control DMAC from µP

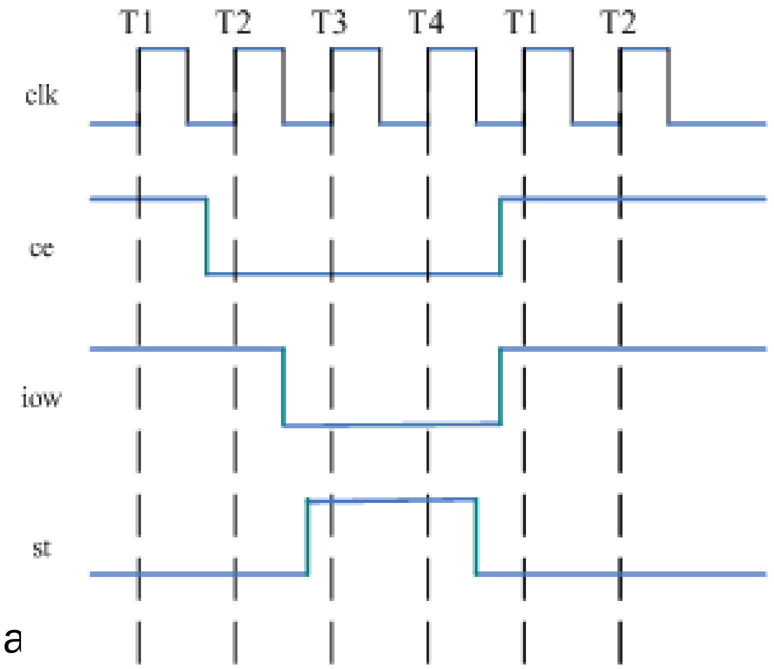
- Previous DMAC starts with a signal, st. Useful in avoiding address decoding logic
- A more appropriate way – via the memory (or I/O) map of the µP.
- In the following example, assume the spare address for DMAC is 380h

St from µP for the FSM

Block Diagram

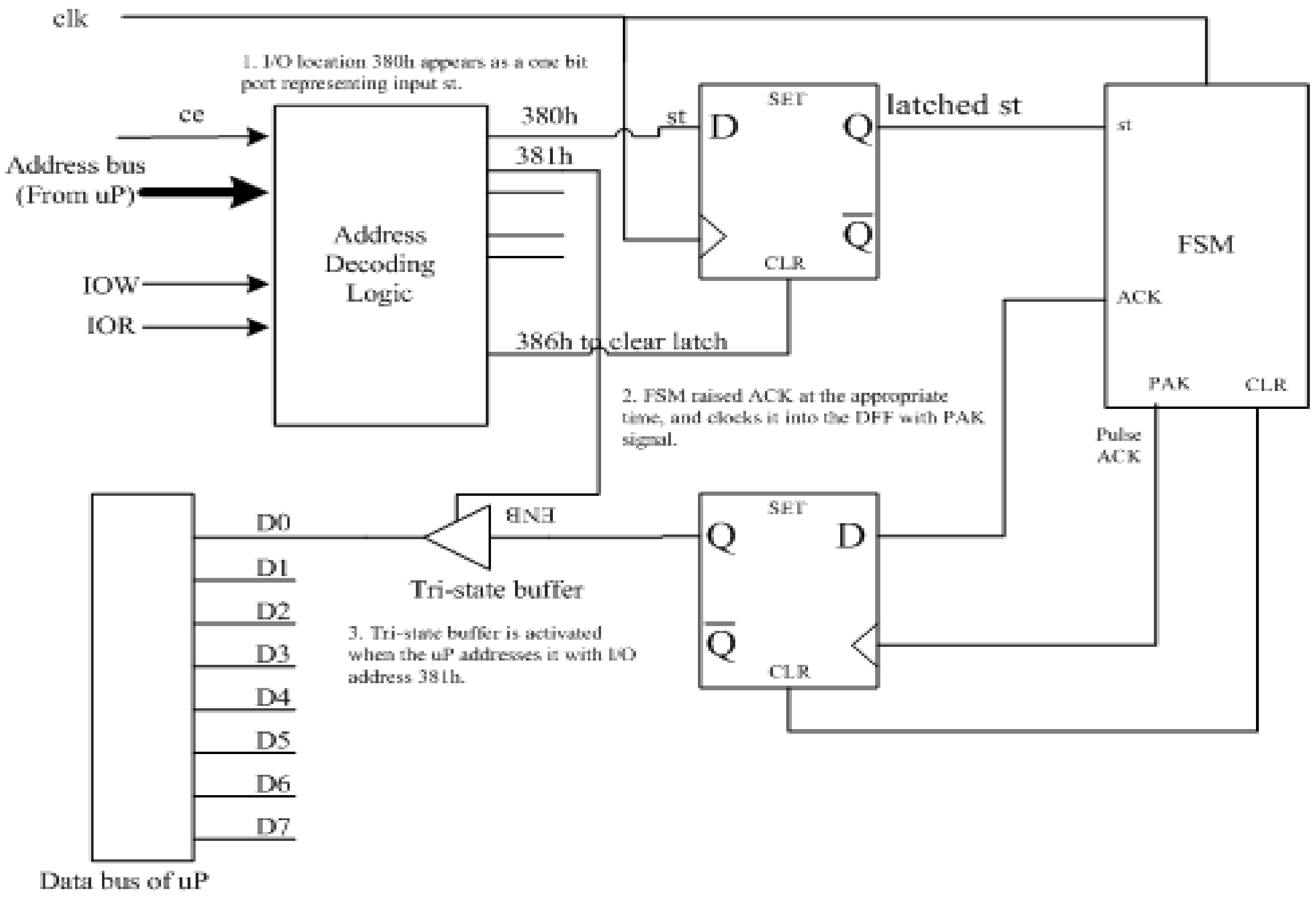


Timing waveform



I/O location 380h appears as a o/p port representing the input st. This needs to be latched into a 1-bit buffer so it can be ready by the FSM.

Whole Block Diagram

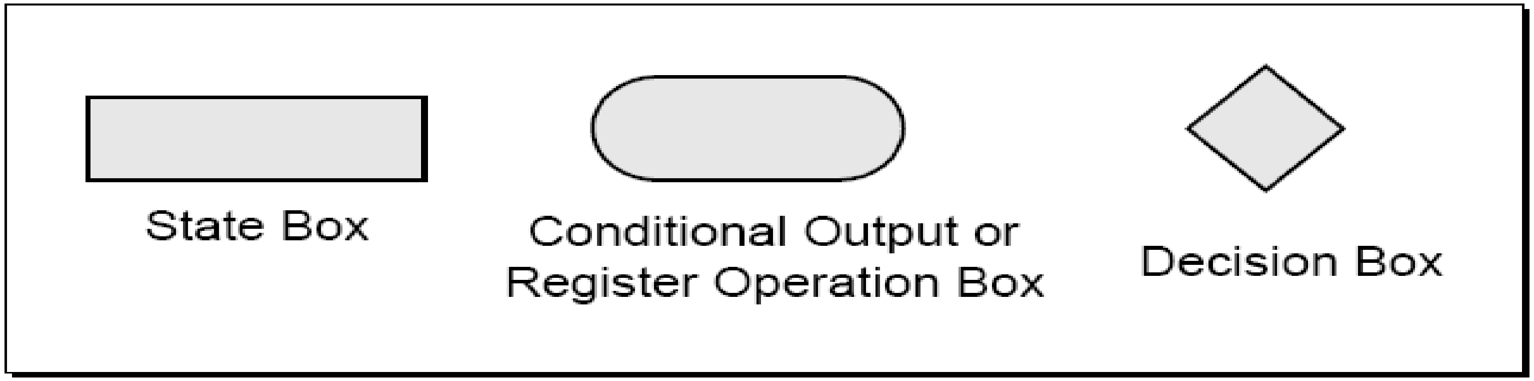

clk
1. I/O location 380h appears as a one bit port representing input st.
ce
Address bus (From uP)
IOW
IOR
Address Decoding Logic
380h
381h
386h to clear latch
st
D
SET
Q
CLR
latched st
st
FSM
ACK
PAK
CLR
2. FSM raised ACK at the appropriate time, and clocks it into the DFF with PAK signal.
Pulse ACK
ENB
Tri-state buffer
3. Tri-state buffer is activated when the uP addresses it with I/O address 381h.
D0
D1
D2
D3
D4
D5
D6
D7
Data bus of uP

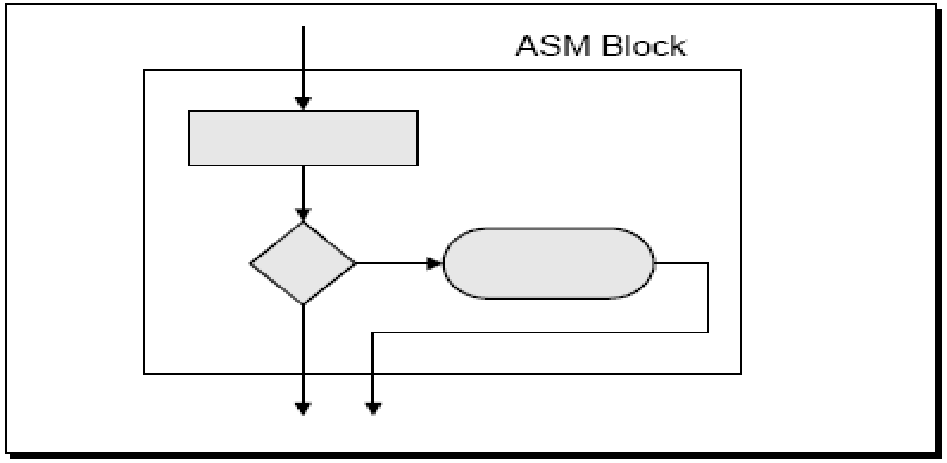
Algorithmic State Machine

Rationale for ASM Charts

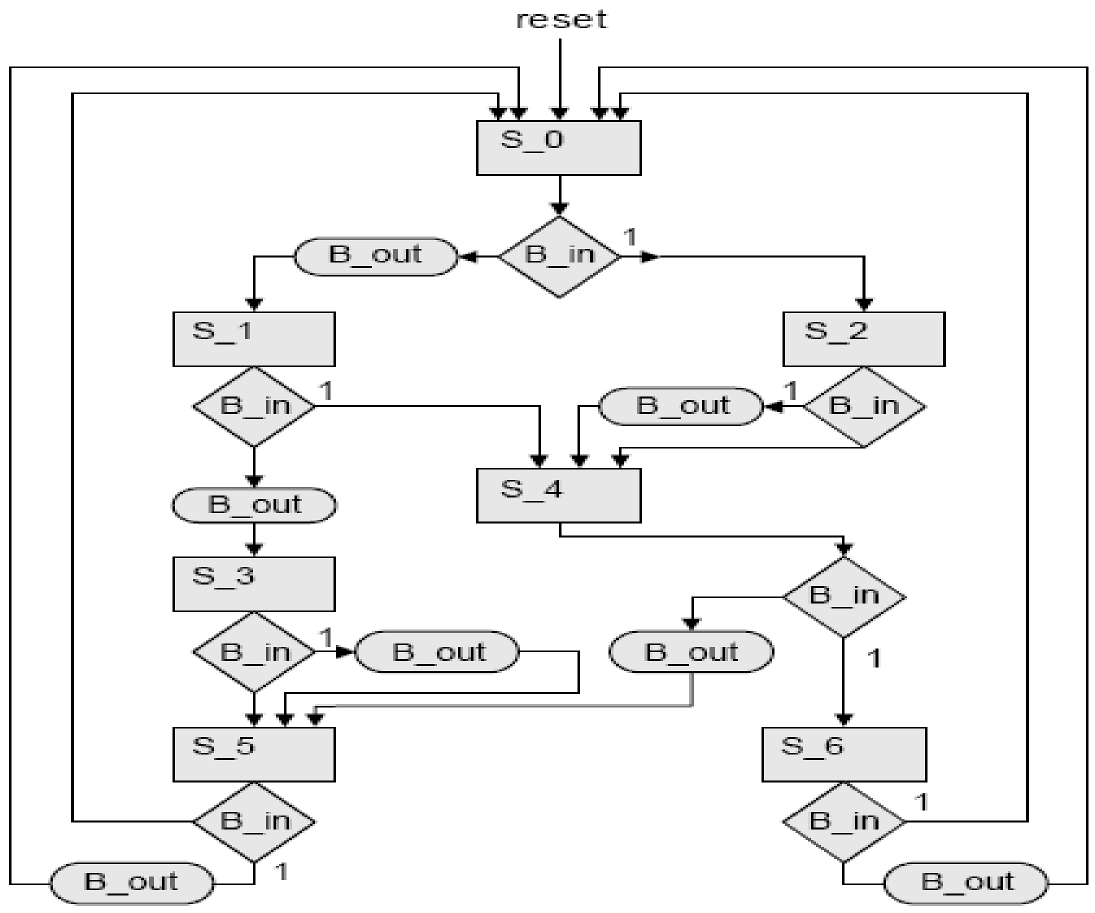
- STG do not directly display the evolution of states resulting from an input
- Algorithmic State Machine (ASM): an abstraction of the functionality of a sequential machine
- ASM charts reveal the sequential steps of a machine's activity
- Focus on machine's activity, rather than contents of registers
- ASM charts can represent Mealy and Moor machines

ASM Chart


State Box
Conditional Output or Register Operation Box
Decision Box


ASM Block

Excess-3 ASM Chart



BCD_to_Excess_3b

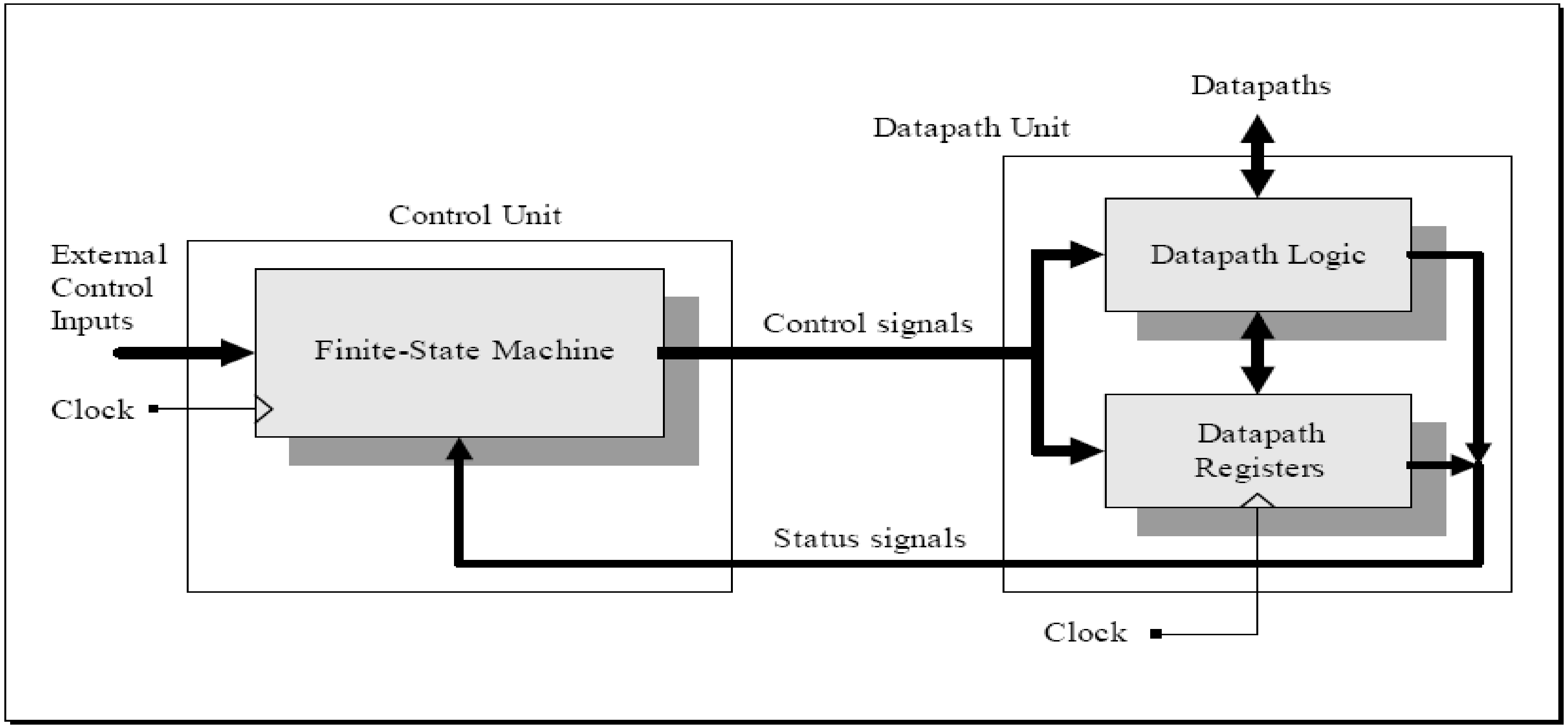
```
module BCD_to_Excess_3b (B_out, B_in, clk, reset_b);
 output              B_out;
 input               B_in, clk, reset_b;
 parameter           S_0 = 3'b000,      // State assignment
                     S_1 = 3'b001,
                     S_2 = 3'b101,
                     S_3 = 3'b111,
                     S_4 = 3'b011,
                     S_5 = 3'b110,
                     S_6 = 3'b010,
                     dont_care_state = 3'bx,
                     dont_care_out = 1'bx;

 reg  [2:0]          state, next_state;
 reg                 B_out;

 always @ (posedge clk or negedge reset_b)
  if (reset_b == 0) state <= S_0; else state <= next_state;
```

```
always @ (state or B_in) begin
   B_out = 0;
   case (state)
     S_0:  if (B_in == 0) begin next_state = S_1; B_out = 1; end
   else if (B_in == 1) begin next_state = S_2; end

     S_1: if (B_in == 0) begin next_state = S_3; B_out = 1; end
   else if (B_in == 1) begin next_state = S_4; end

     S_2: begin next_state = S_4; B_out = B_in; end

     S_3: begin next_state = S_5; B_out = B_in; end

     S_4: if (B_in == 0) begin next_state = S_5; B_out = 1; end
   else if (B_in == 1) begin next_state = S_6; end

     S_5:  begin next_state = S_0; B_out = B_in; end

     S_6:  begin next_state = S_0; B_out = 1; end
  endcase
  end
endmodule
```

Reading: M. Ciletti Chap 7

Controller for Datapath

FSM Controller for a Datapath



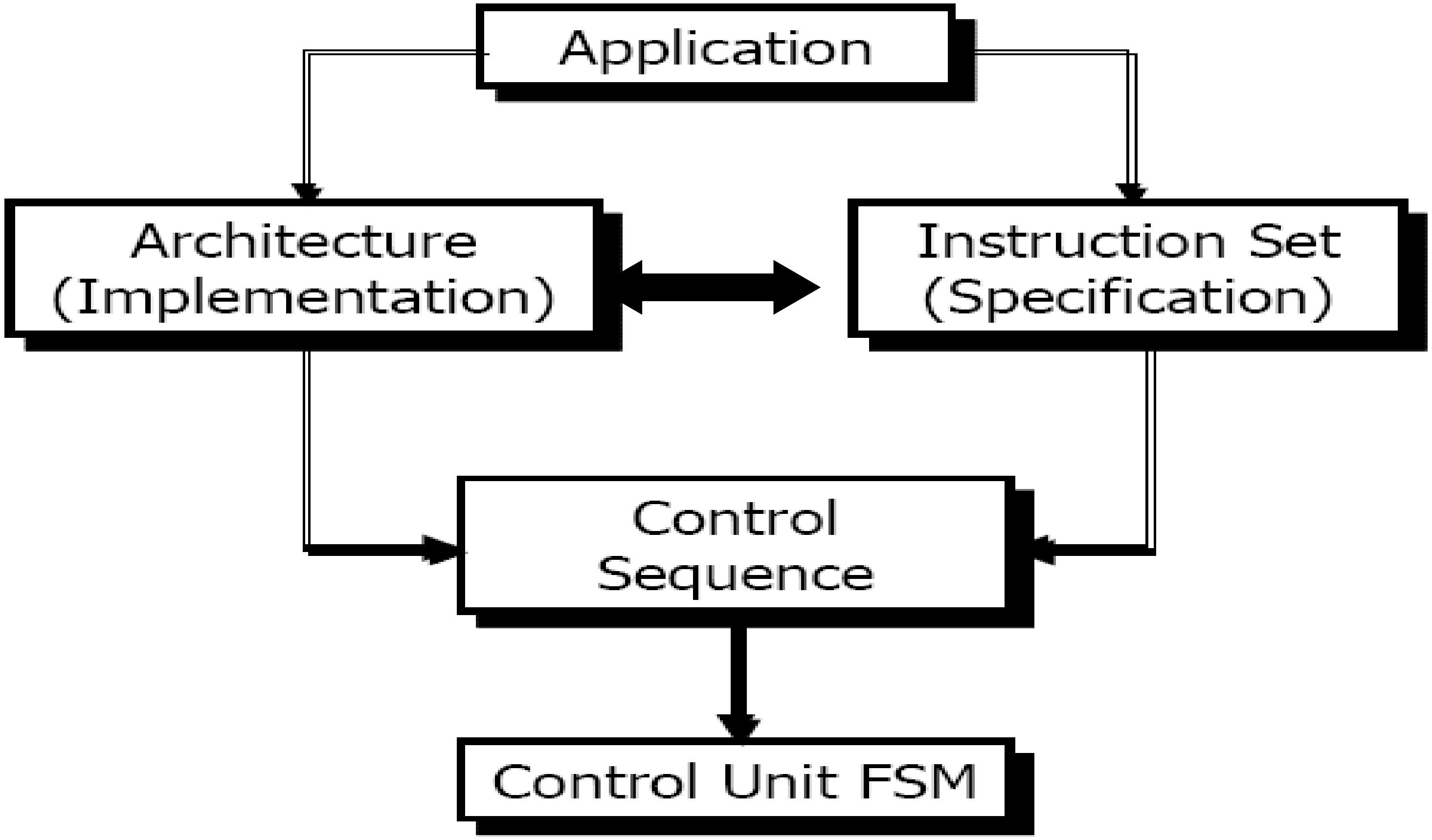
The controller dictates the timing of all activity in a system.

Datapath Control Signals

For stored-program computer, control signals need to

- Load, read, shift contents of registers
- Fetch instruction from memory
- Store data in memory
- Steer signals through muxes
- Control 3-state devices
- Select/execute ALU operations

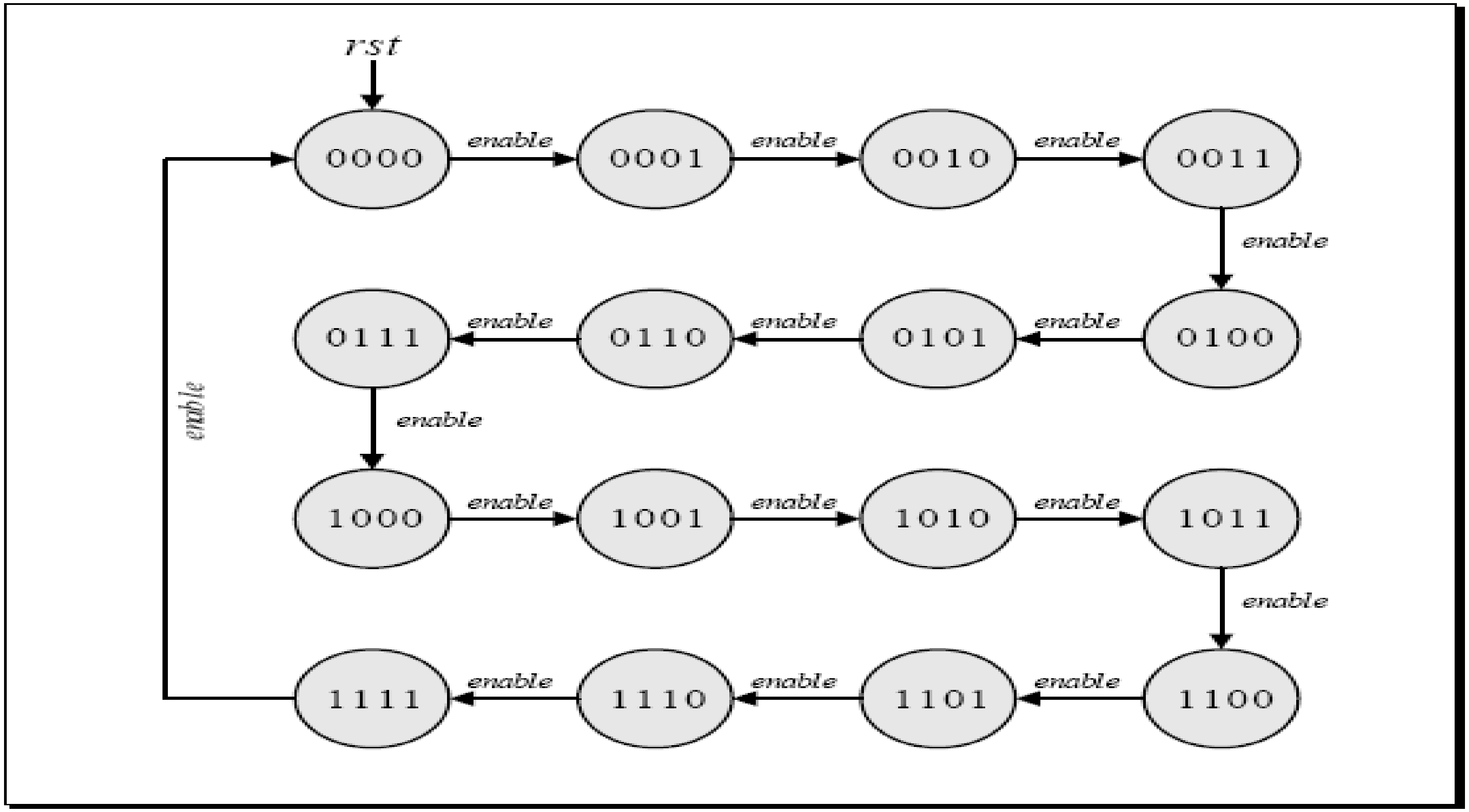
Conceptual Steps of Designing Application-Driven HW Systems


Application
Architecture (Implementation)
Instruction Set (Specification)
Control Sequence
Control Unit FSM

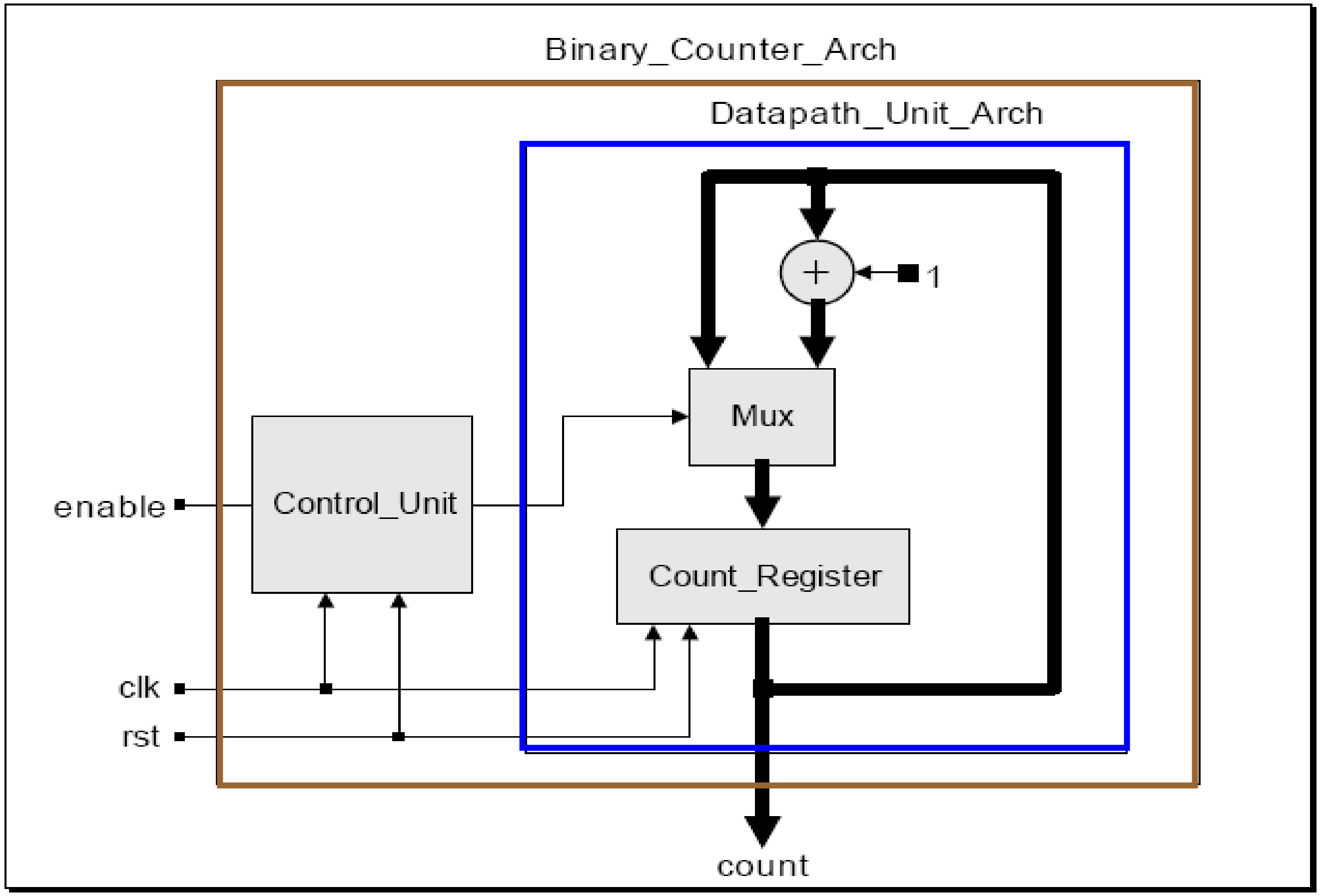
Design Example: Binary Counter

- Problem statement:
  - Design a 4-bit binary counter with features
    - Incremented by 1 at each active edge of the clock
    - Wrapped around to 0 when reaching 1111_2
- Ideas to write the controller's code
  - Use implicit state machine
  - Use explicit state machine
  - Separate control unit and datapath
    - RTL structure model
    - RTL behavioral model

Approach One: Explicit-state Machine



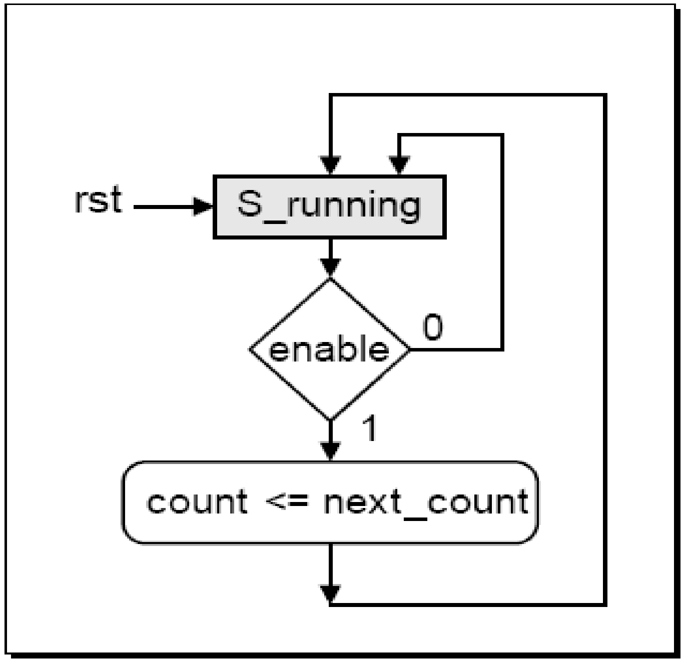
Approach Two: Datapath and Controller

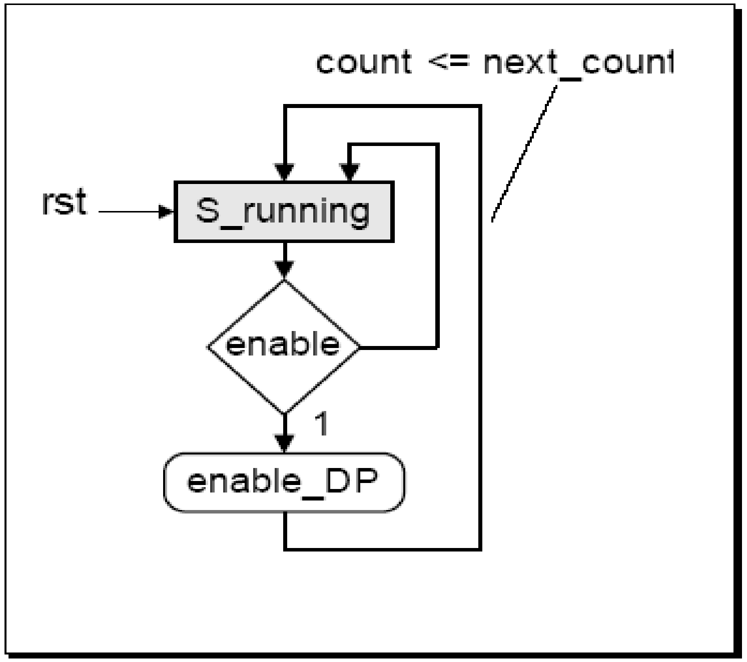


Approach Three: Implicit State Machine

```
module binary_counter_imp(cnt, enable, reset, clock);
input enable, reset, clock;
output cnt;

reg [3:0]cnt;

always @(posedge reset or posedge clock)
    if(reset==1)
        cnt <= 4'b0;
    else
       if (enable==1)
              cnt <= cnt+1;

endmodule
```

ASM Views of Two Machines





Confusion: Mixed Datapath Operations and FSM

Clarity: Partitioned Datapath and Controller

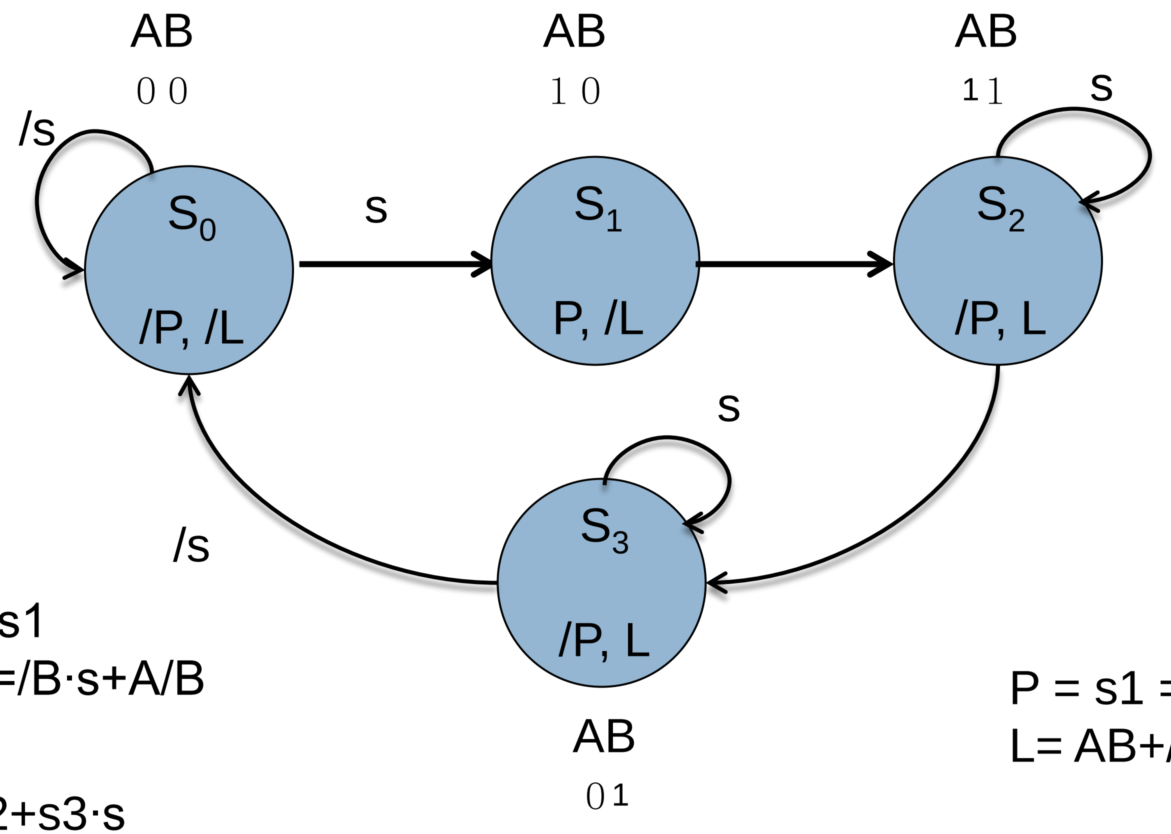
ASM-Based Verilog Model (1/2)

```
module Binary_Counter_Part_RTL (count, enable, clk, rst);
parameter       size = 4;
output  [size -1: 0]    count;
input           enable;
input           clk, rst;
wire            enable_DP;

Control_Unit M0 (enable_DP, enable, clk, rst);
Datapath_Unit M1 (count, enable_DP, clk, rst);
endmodule

 module Control_Unit  (enable_DP, enable, clk, rst);
   output       enable_DP;
   input        enable;
   input        clk, rst;      // Not needed

                wire enable_DP = enable;

 endmodule
```

ASM-Based Verilog Model (2/2)

```
module Datapath_Unit (count, enable, clk, rst);
parameter   size = 4;
output     [size-1: 0] count;
input       enable;
input       clk, rst;
reg         count;
wire       [size-1: 0] next_count;

always @ (posedge clk)
   if (rst == 1) count <= 0;
   else if (enable == 1) count <= next_count(count);

  function [size-1: 0] next_count;
  input [size-1:0] count;
    begin
              next_count = count + 1;
    end
  endfunction
endmodule
```

State Diagram for Implementation using DFFs (for SPGM-2)

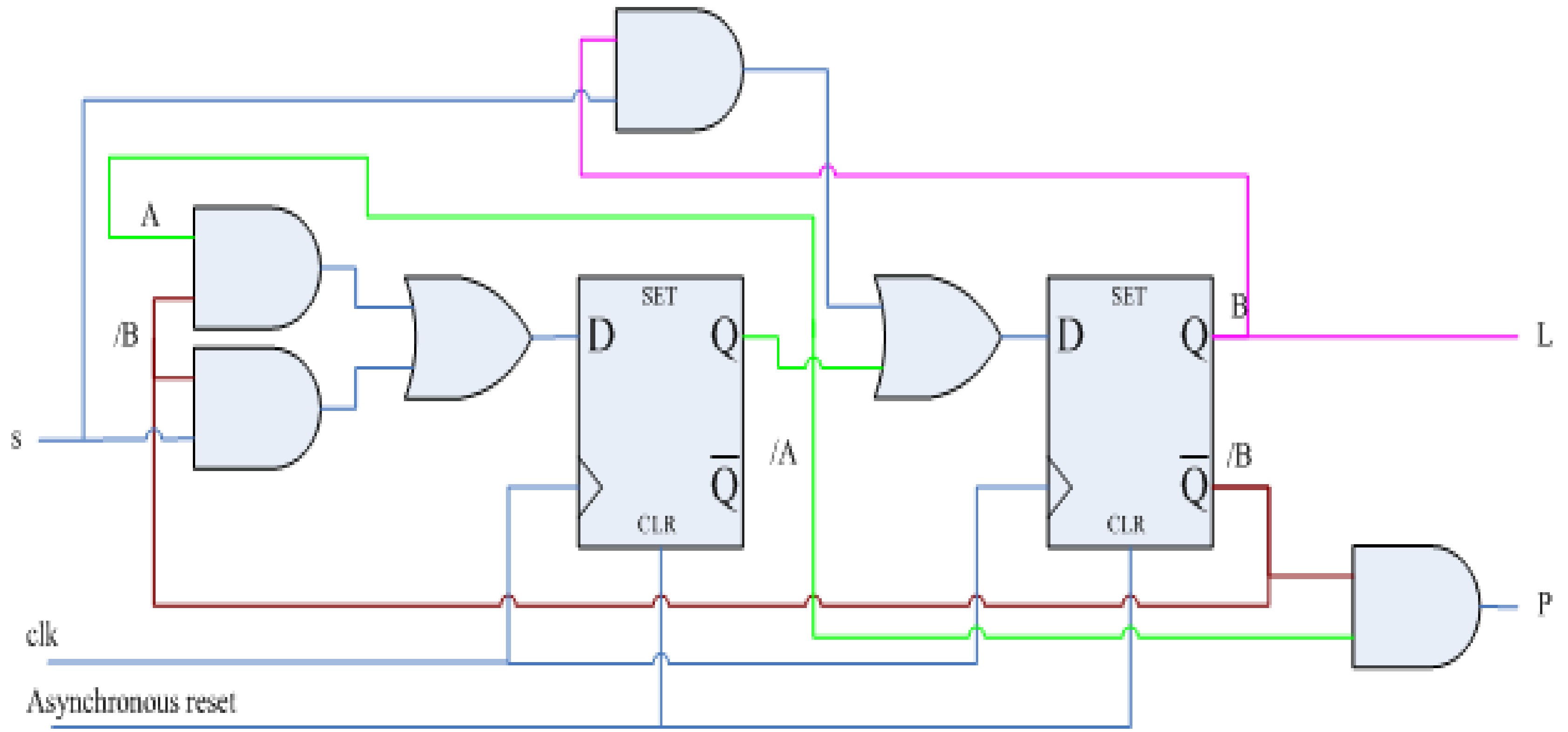


A->D = s0·s+s1
=/A/B·s+A/B =/B·s+A/B

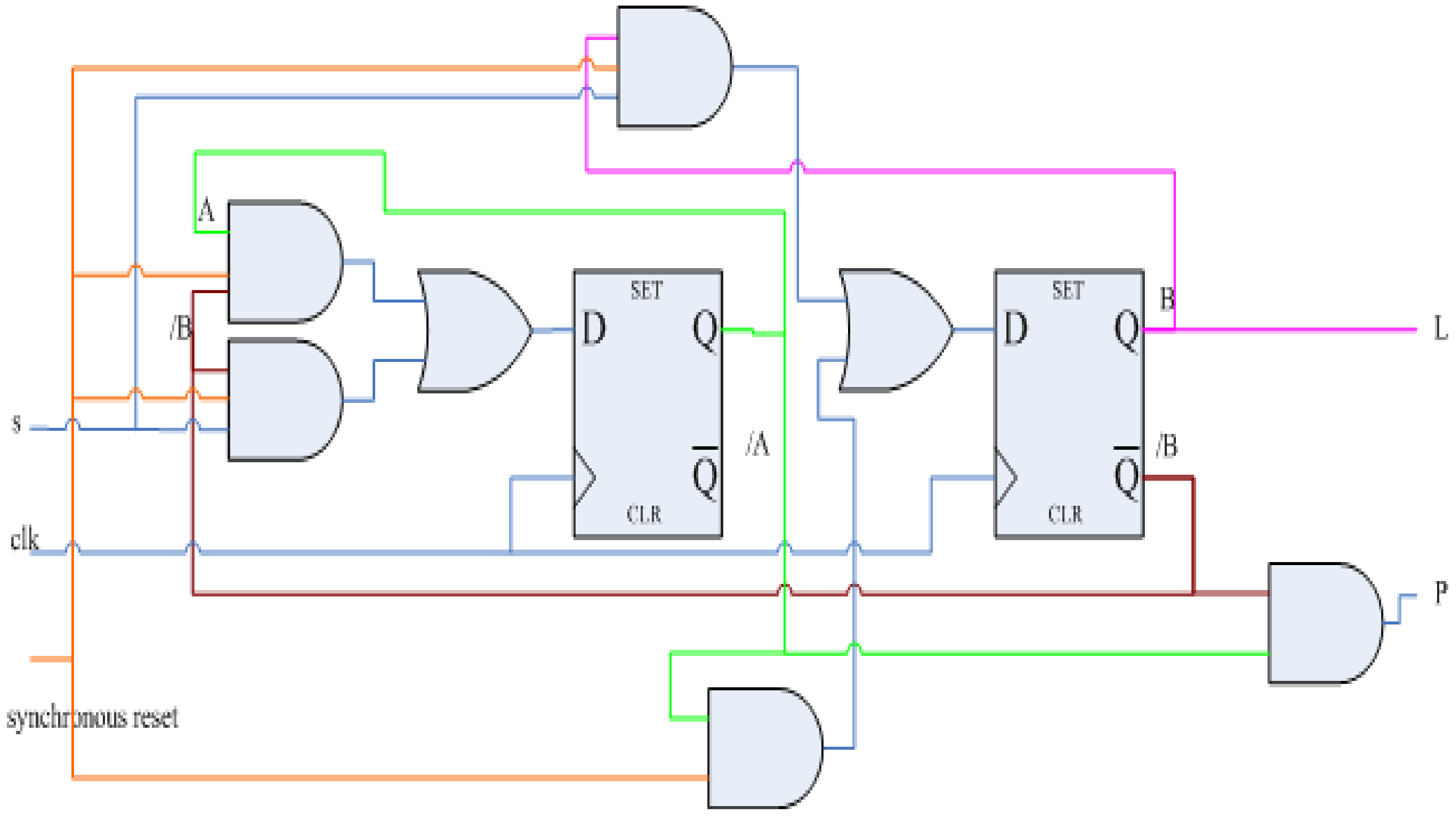
B->D = s1+s2+s3·s
=A/B+AB+/AB ·s = A+B ·s

P = s1 = A/B
L= AB+/AB = B

RTL for SPG with Asyn. Reset



RTL for SPG with Syn. Reset


A
/B
s
clk
synchronous reset
SET
D
Q
CLR
/A
SET
D
Q
CLR
B
L
/B
P