EE 457 Unit 1

Overview of Digital System Design

Credits

- These slides were derived from Gandhi Puvvada's EE 457 Class Notes

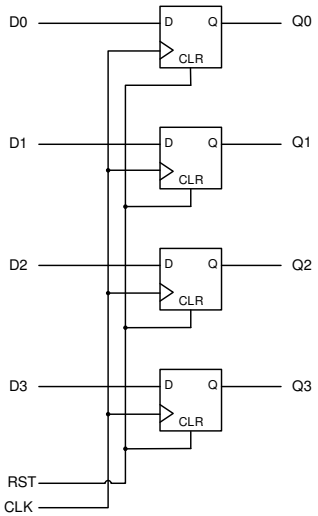
Clocking Strategies

REGISTERS & DATA ENABLES

Registers

- A Register is a group of D-FF's tied to a common clock and clear (reset) input
  - Clear can be asynchronous or synchronous
- Used to store multiple bit values on each clock cycle

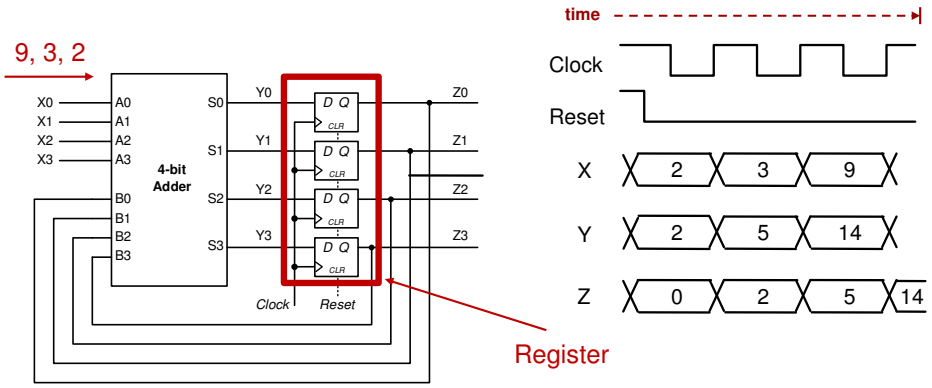
CLK	RST	D_i	Q_i*
1,0	X	X	Q_i
↑	1	X	0
↑	0	0	0
↑	0	1	1



4-bit Register

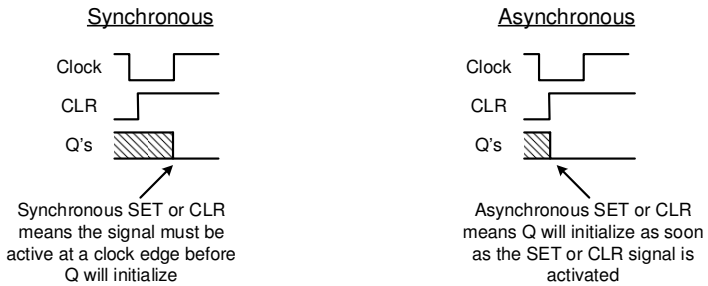
Example: Accumulator

- Sum a time-based sequence of numbers
- A register usually stores a single logic value (i.e. a number)



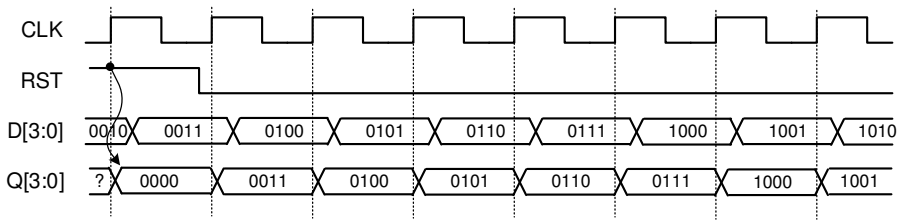
Synchronous vs. Asynchronous

- The set/preset and clear inputs can be built to be ***synchronous*** or ***asynchronous***
- These terms refer to when the initialization takes place
  - Asynchronous Reset (AR): Initialization of Q takes effect immediately regardless of the CLK
  - Synchronous Reset (SR): Initialization of Q takes effect only at an edge (clear must be active at the edge)



Registers

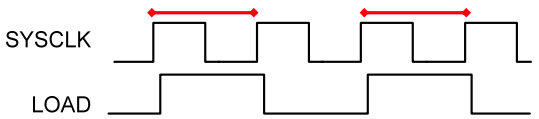
- Whatever the D value is at the clock edge is sampled and passed to the Q output until the next clock edge



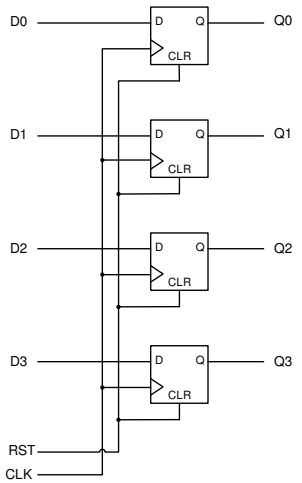
4-bit Register – On clock edge, D is passed to Q

Selective Loading/Registering of Data

- What if we only want a register to capture data on *selective* clocks (and not on *EVERY* clock)
  - Clocks are indicated with a "LOAD" signal



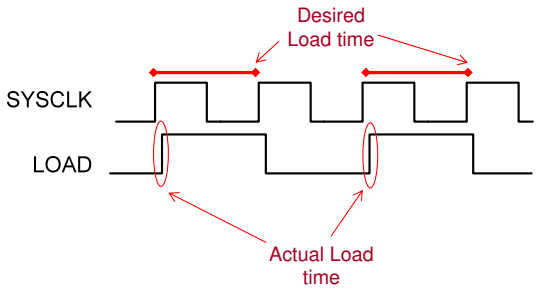
Want to load the register on the indicated clock cycles and have it retain its value in the other cycles

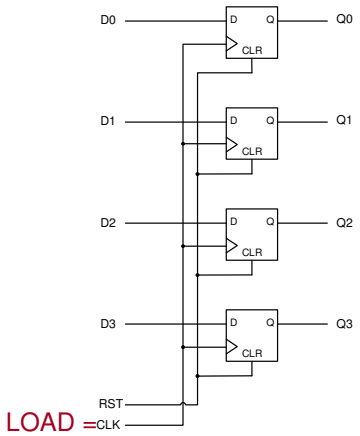


4-bit Register

Clocking Option 1

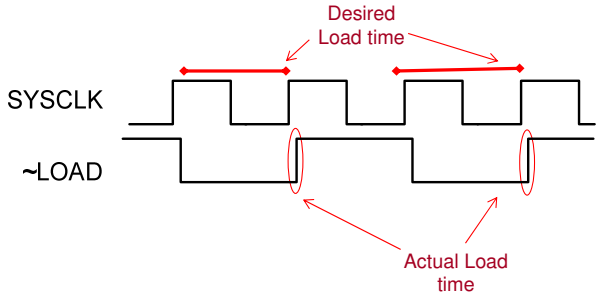
- Use Load as the clock signal
- **(Does/Doesn't)** Work.
- _____

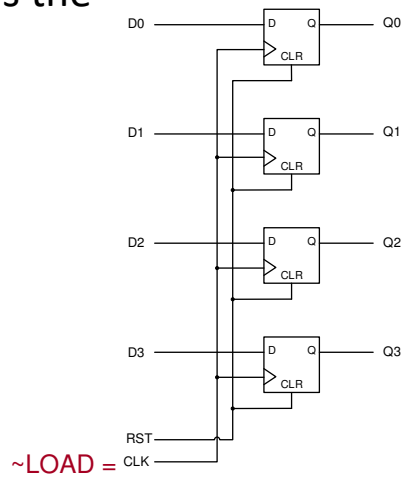




Clocking Option 2

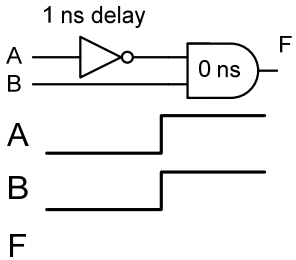
- Use ~Load (inverted Load) as the clock signal
- **(Does/Doesn't)** Work.
- _____





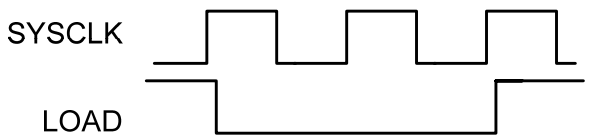
Glitches

- Temporary (transient) incorrect / toggling output values due to differing delay paths of the inputs
  - Eventually output settles to correct value
  - Unless a circuit is specially designed, glitches are possible on all circuits



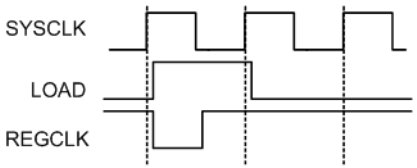
Successive Loading Clocks

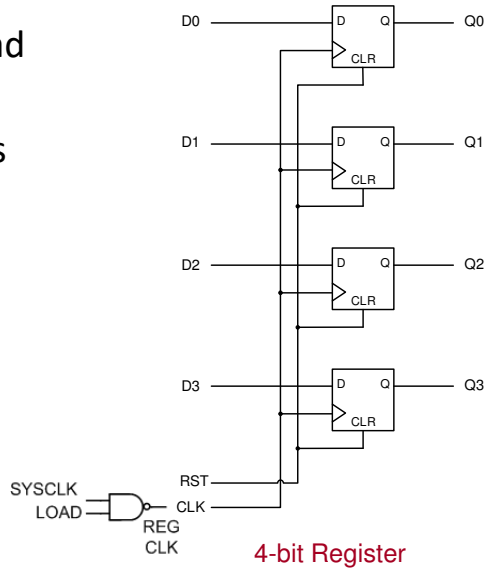
- If load is held high on two successive clock cycles will get _____ edge(s)



Option 3

- Gate the clock with the load signal
- Also susceptible to glitches

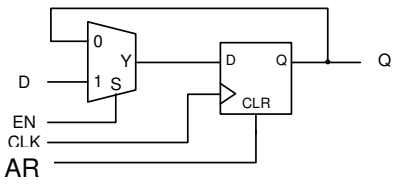




4-bit Register

Option 4: Feedback mux

- Registers (D-FF's) will sample the D bit every clock edge and pass it to Q
- Sometimes we may want to hold the value of Q and ignore D even at a clock edge
- We can add an enable input and some logic in front of the D-FF to accomplish this

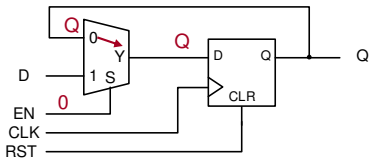


FF with Data Enable
(Always clocks, but selectively chooses old value, Q, or new value D)

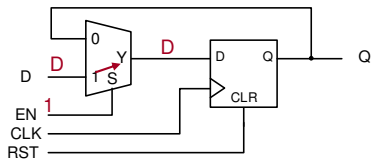
CLK	AR	EN	D_i	Q_i^*
X	1	X	X	0
0,1	0	X	X	Q_i
↑	0	0	X	Q_i
↑	0	1	0	0
↑	0	1	1	1

Registers w/ Enables

- When EN=0, Q value is passed back to the input and thus Q will maintain its value at the next clock edge
- When EN=1, D value is passed to the input and thus Q will change at the edge based on D



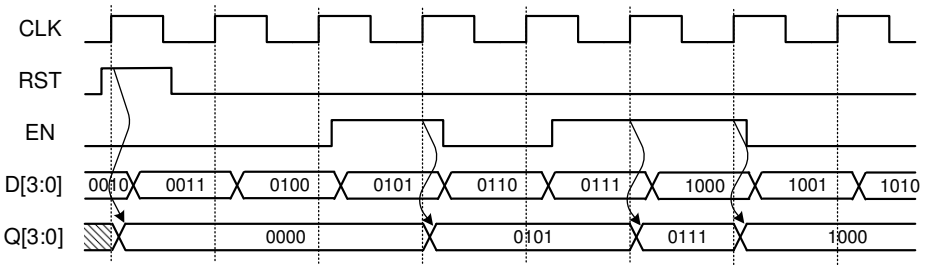
When EN=0, Q is recycled back to the input



When EN=1, D input is passed to FF input

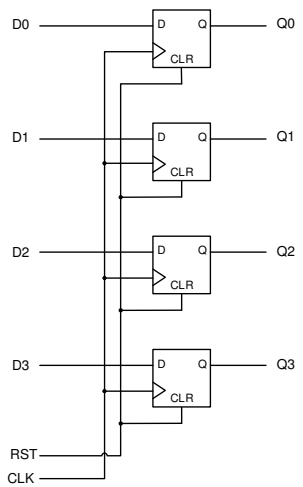
Registers w/ Enables

- The D value is sampled at the clock edge only if the enable is active
- Otherwise the current Q value is maintained

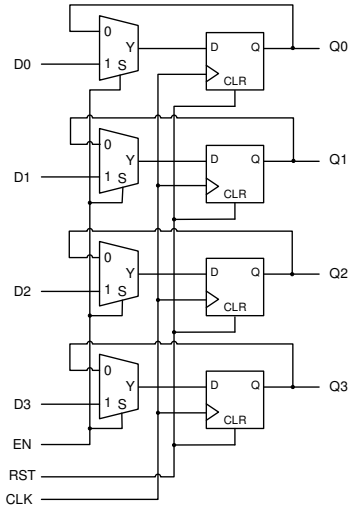


Register With or Without An Enable

Free-Running Register



Register with Load (Data) Enable

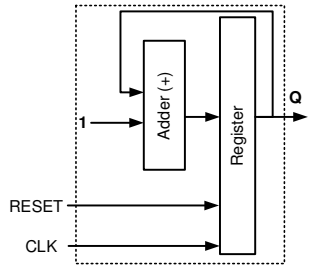


When to use one vs. the other?

- Free-running register: Do you want to update the stored value EVERY edge
- Register w/ Enable: In all other cases…

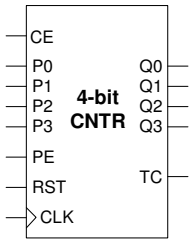
Counters

- Increment (Add 1 to Q) at each clock edge
  - Up Counter: Q* = Q + 1
- Standard counter components include other features
  - Enables: Will not count at edge if EN=0
  - Resets: Reset count to 0
  - Parallel Load Inputs: Can initialize count to a value P (i.e. Q* = P rather than Q+1)



Sample 4-bit Counter

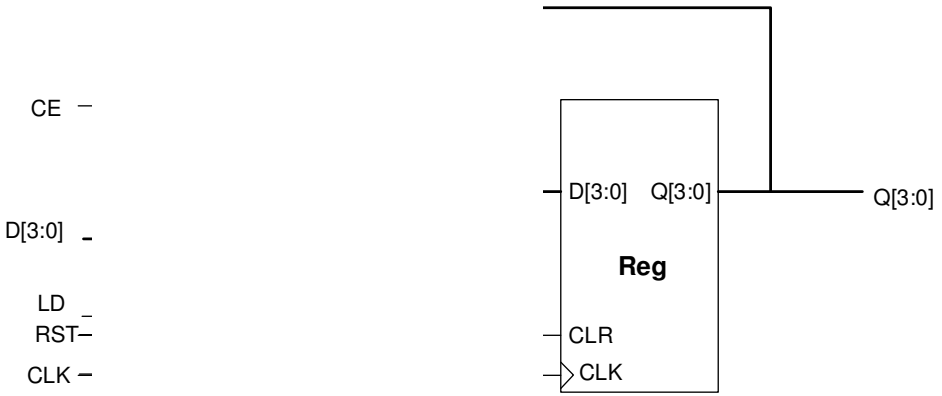
- 4-bit Up Counter
  - RST: synchronous reset input
  - PE and P_i inputs: loads Q with P when PE is active
  - CE: Count Enable
    - Must be active for the counter to count up
  - TC (Terminal Count) output
    - Active when Q=1111 AND counter is enabled
    - TC = EN•Q3•Q2•Q1•Q0
    - Indicates that on the next edge it will roll over to 0000
    - Used to create 8-, 12-, 16-bit, etc. counters from these 4-bit building blocks



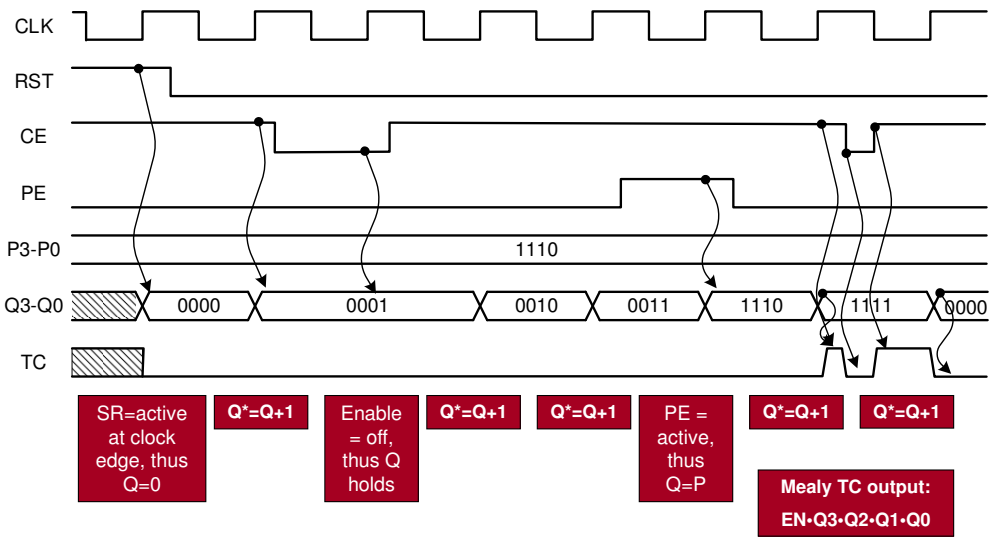
CLK	RST	PE	CE	Q*
0,1	X	X	X	Q
↑	1	X	X	0
↑				
↑				
↑				

Counter Design

- Sketch the design of the 4-bit counter presented on the previous slides



Counters



Reference Verilog

- Verilog description of a register with enable and counter with load and count enable

```
module reg16e(
  input clk,
  input reset,
  input en,
  input [15:0] d,
  output reg [15:0] q
);

  always @(posedge clk)
  begin
    if(reset)
       q <= 16'd0;
    else if(en)
       q <= d;
  end
endmodule
```

16-bit Register w/ Enable

```
module cntr16ce(
    input clk,
    input reset,
    input load,
    input ce,
    input [15:0] d,
    output reg [15:0] q
);

  always @(posedge clk)
  begin
  if(reset == 1)
    q <= 16'd0;
  else if(load == 1)
    q <= d;
  else if(ce == 1)
    q <= q+1;
  end
endmodule
```

16-bit Counter w/ Load and Count Enable

Data Register Summary

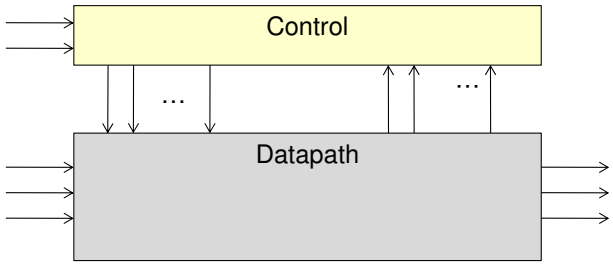
- Understand the operation of a free-running register, register w/ data/load enable, and a counter with count enables, etc.

Datapath and Control Unit Decomposition

SYNCHRONOUS SYSTEM DESIGN TECHNIQUES

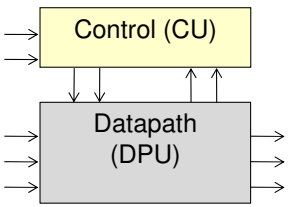
Digital System Design

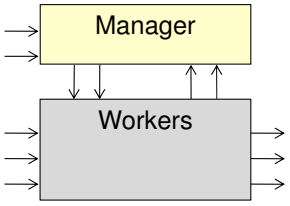
- Control **(CU)** and Datapath Unit **(DPU)** paradigm
  - Separate logic into datapath elements that operate on data and control elements that generate control signals for datapath elements
  - Datapath: Adders, muxes, comparators, counters, registers (w/ enables), memories, FIFO's
  - Control Unit: State machines/sequencers



Datapath + Control

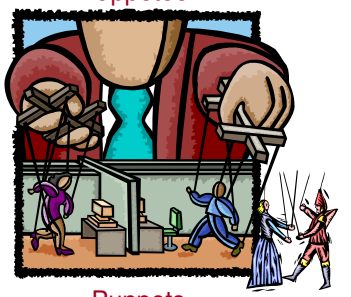
- The control unit acts as scheduler and manager while the datapath "does" the work
  - Similar division of labor in many other areas
  - Control signals include: _____, _____, _____ enables, output enables, etc.





Construction Company

Puppeteer



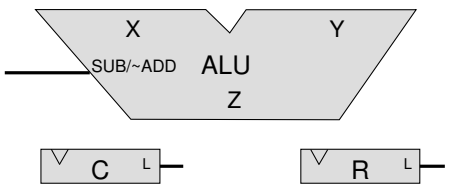
Puppets

Identifying Control Signals

- Design a datapath to support the following RTL (Register Transfer Level) operations
  - Identify the control signals & other datapath components

Desired Operations:

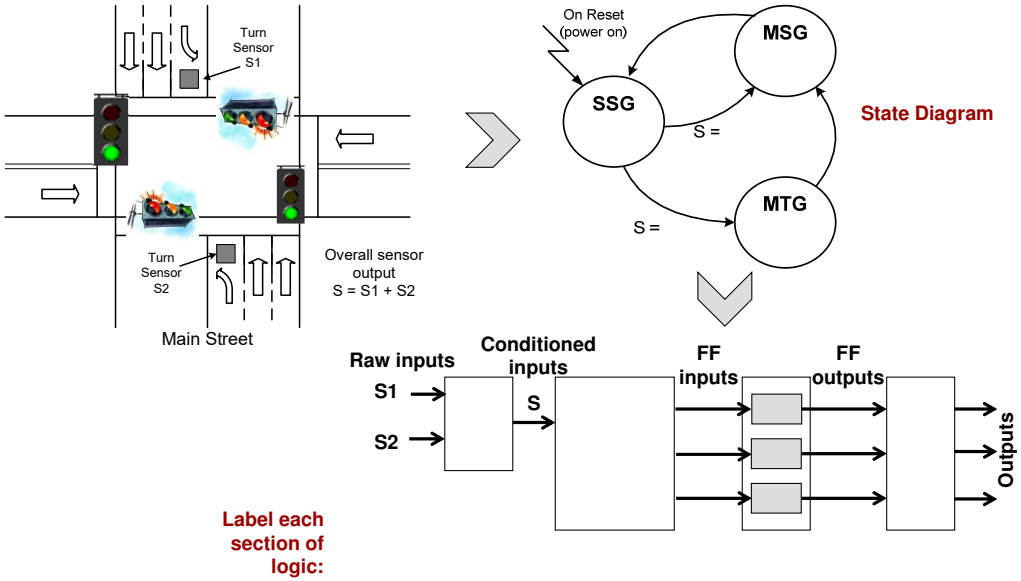
C ← A+B,	if F,G=0,0
C ← A-B,	if F,G=0,1
R ← P+Q,	if F,G=1,0
R ← P-Q,	if F,G=1,1



One-hot State Machine Design

STATE MACHINE (CONTROL UNIT) DESIGN

Digital System Representation



State Machine Review

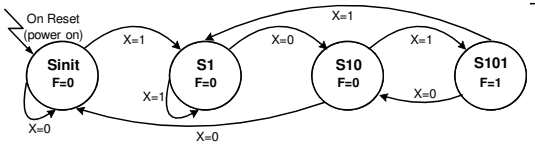
State Diagrams

1. States
2. Transition Conditions
3. Outputs

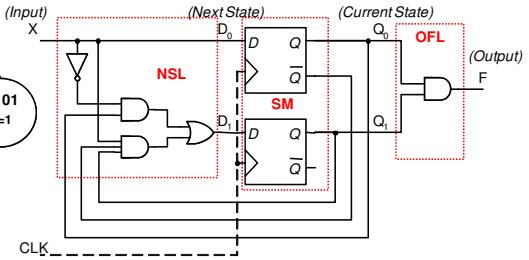
State Machine

1. State Memory => FF's
   - n-FF's => 2^n states
2. Next State Logic (NSL) + Input Function Logic (IFL)
   - combinational logic for FF inputs
3. Output Function Logic (OFL)
   - MOORE: f(state)
   - MEALY: f(state + inputs)

State Machines require sequential logic to remember the current state (w/ just combo logic we could only look at the current value of X, but now we can take 4 separate actions when X=0)

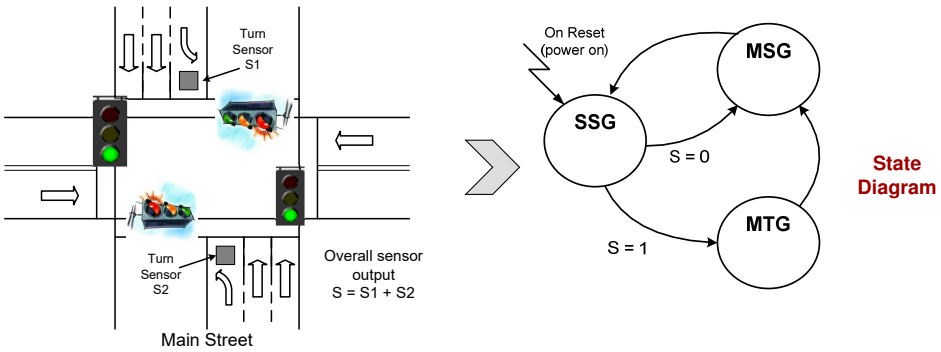


State Diagram for "101" Sequence Detector



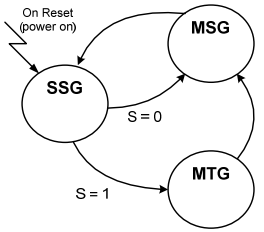
State Assignment

- Design of the traffic light controller with main turn arrow
- Represent states with some binary code, but what kind?
  - Encoded: 3 States => 2 bit code: 00=SSG, 01=MSG, 10=MTG
  - One-hot: Separate FF per state: 100=SSG, 010=MSG, 001=MTG



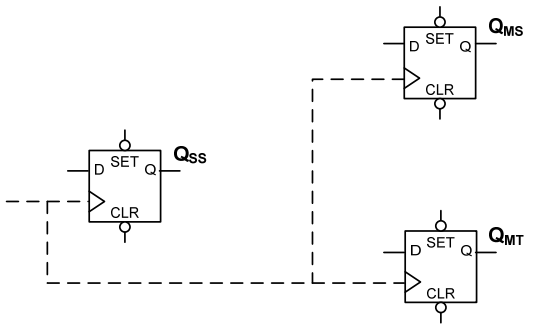
NSL Implementation in 1-Hot Method

- In one-hot assignment, NSL is designed by simple observation
- For each state, examine each incoming transition
  - Each incoming arrow will be one case in our logic
  - We can just OR each condition together
- Describe each transition as a combination of what state it originates from & any associated conditions
- Ex. Two arrows converge on MS: "Q_{MS} should be '1' on the next clock when…
  - Current state is MT ***…OR…***
  - Current stat is SS **AND** S=0



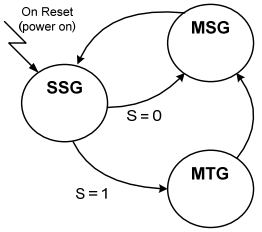
	Q_{SS}	Q_{MT}	Q_{MS}
SS	1	0	0
MT	0	1	0
MS	0	0	1

One-hot State Assignment



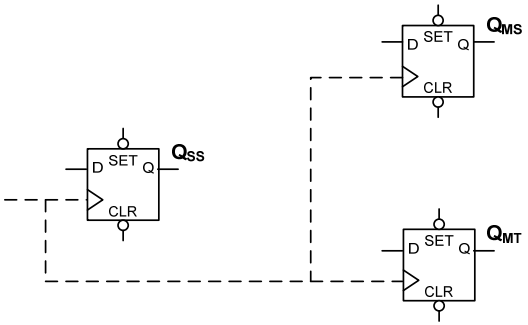
NSL Implementation in 1-Hot Method

- Two arrows converge on MS: "Q_{MS} should be '1' on the next clock when…
  - Current state is MT ***…OR…***
  - Current state is SS **AND** S=0
- **$Q^*_{MS} = D_{MS} = Q_{MT} + Q_{SS}$•S'**
- **$Q^*_{MT} = D_{MT} =$**
- **$Q^*_{SS} = D_{SS} =$**
- **What about initial state? Preset the appropriate flop.**



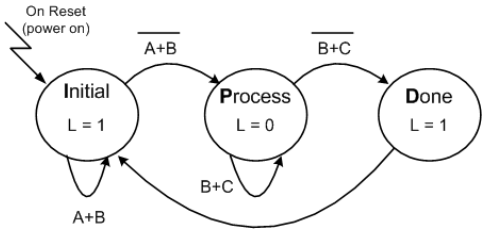
	Q_{SS}	Q_{MT}	Q_{MS}
SS	1	0	0
MT	0	1	0
MS	0	0	1

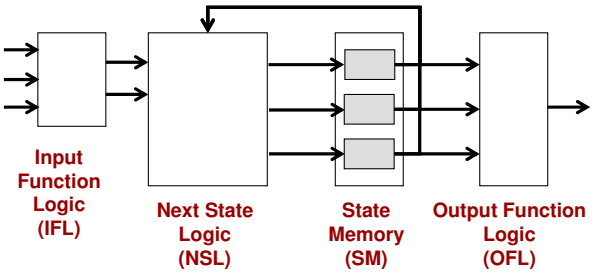
One-hot State Assignment



Illustrative Example

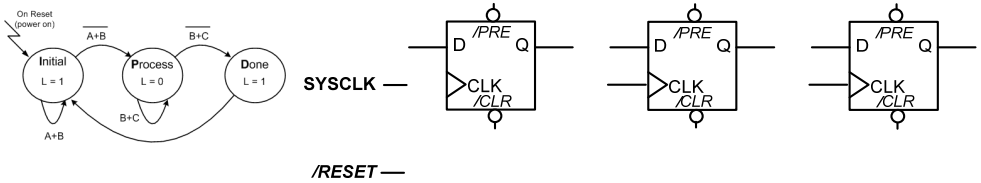
- Consider the following state diagram





State Assignment & NSL

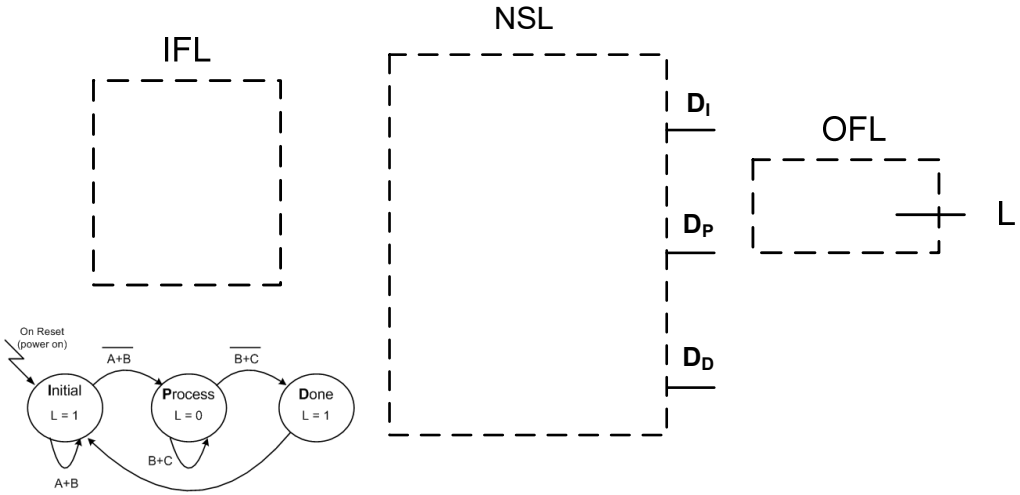
- Let us choose a one-hot state assignment (one FF per state)



- Next State Equations:
  - $D_I = Q_I^* =$
  - $D_P = Q_P^* =$
  - $D_D = Q_D^* =$

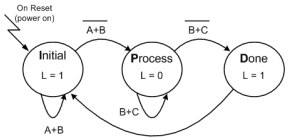
IFL, NSL, & OFL

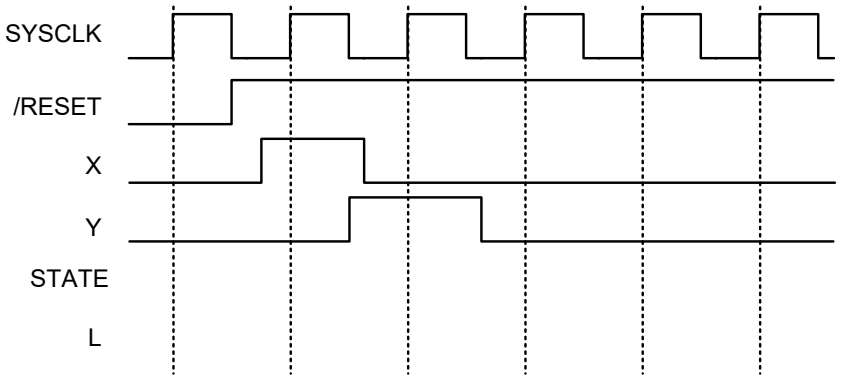
- Now we can draw the logic for each section



Waveform

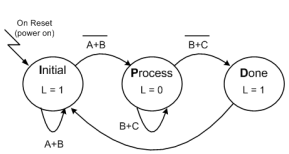
- Recall, X = A+B and Y = B + C
- L is a combinational function of the current state

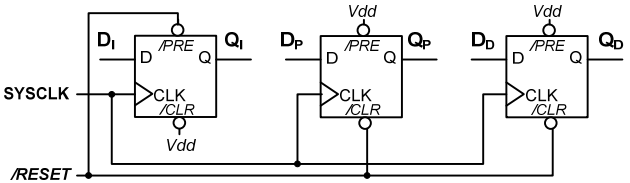




State Assignment & NSL

- Let us choose a one-hot state assignment (one FF per state)

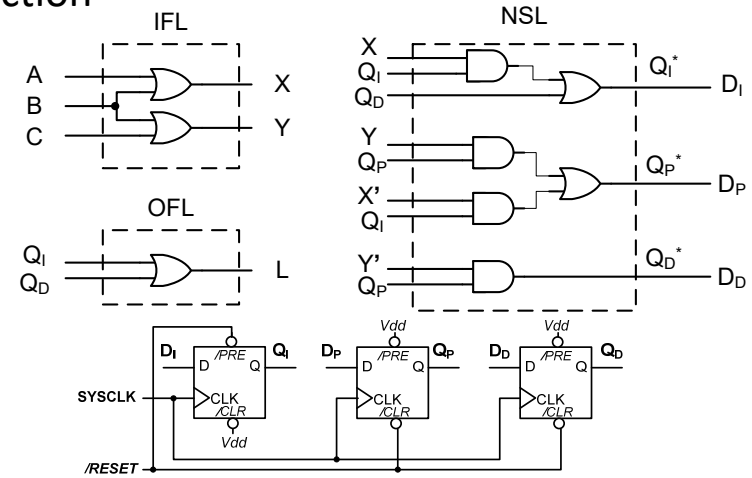




- Next State Equations:
  - $D_I = Q_I^* = Q_I \bullet (A+B) + Q_D$
  - $D_P = Q_P^* = Q_P \bullet (B+C) + Q_I \bullet (A+B)'$
  - $D_D = Q_D^* = Q_P \bullet (B+C)'$

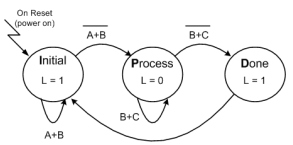
IFL, NSL, & OFL

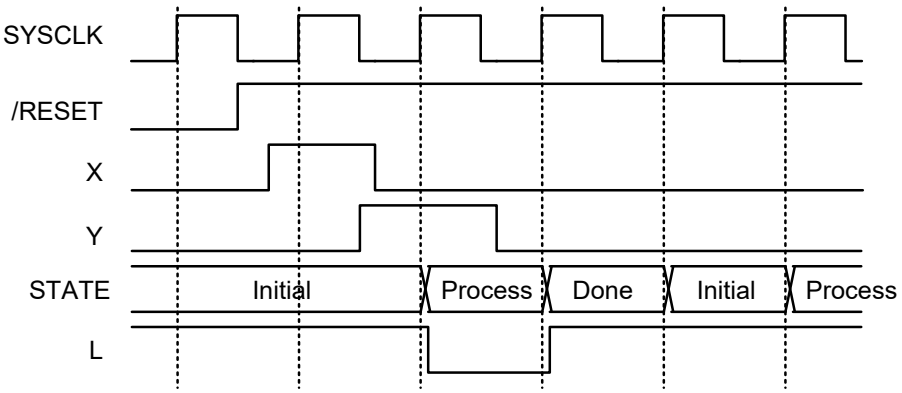
- Now we can easily draw the logic for each section



Waveform

- Recall, X = A+B and Y = B + C
- L is a combinational function of the current state





State Machine Summary

- 4 sections of state machine circuitry: _____
- In one-hot state encoding, a system with 5 states requires ____ flip-flops (____ flip-flops per _____)
- In an encoded state encoding, a system with 5 states requires ___ flip-flops (_____ FF's for n states)
- When designing the NSL for a state, enumerate the conditions associated with the (**incoming/outgoing**) transitions to that state
- To implement the power-on reset condition in a one-hot state encoding, connect the RESET signal to the _____ input for the ____ FF associated with the initial state, and to the _____ signals of the other FF(s).

Mealy- vs. Moore-style outputs

STATE MACHINE OUTPUTS

State Machine Outputs

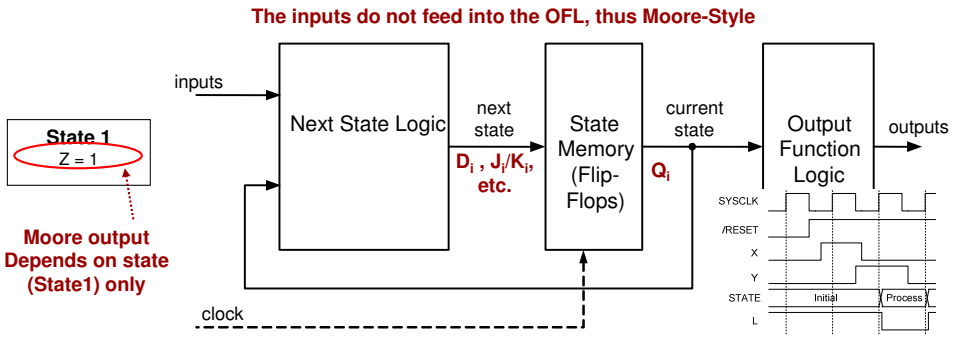
- State Machine outputs can be classified according to how the outputs are produced

– If *Outputs = f(current state, other inputs)*…
_____-Style

– If *Outputs = f(current state)*…
_____-Style

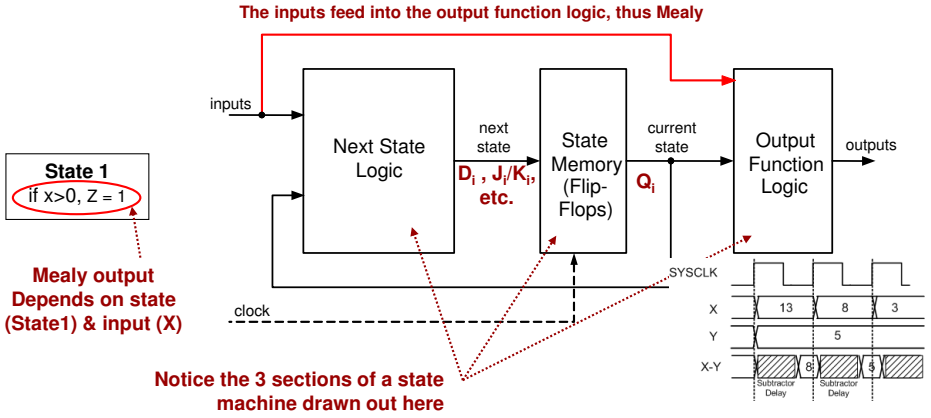
Moore-Style Outputs

- Moore-style outputs only depend on the current state
- Thus, they are valid _____ in the clock cycle and ***stay*** _____ nearly the entire
- Often requires extra states compared to Mealy-style implementations



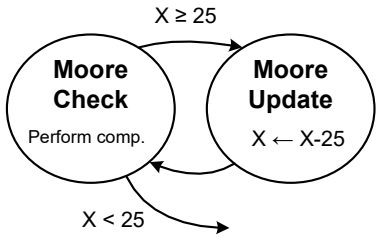
Mealy-Style Outputs

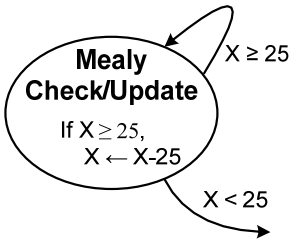
- Mealy-style outputs depend not only on the current state but the external inputs
- Thus, they may not be valid until _____ in the clock cycle and ***may*** _____ during the cycle if the inputs change



Mealy vs. Moore Update

- Consider the update/loading of a register, X, with X-25
- Need to generate an X_LOAD signal
  - Can be Moore or Mealy-style



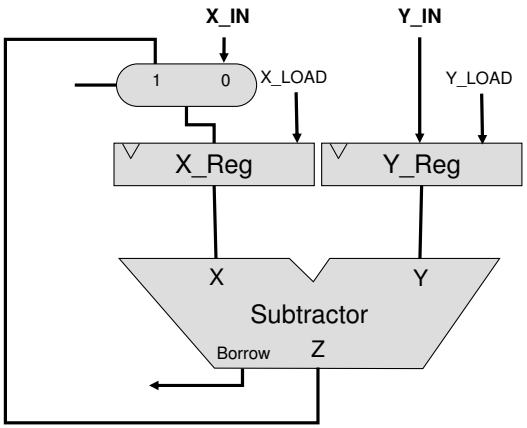


Divider

- Consider design of a sequential divider, (X / Y)
- Algorithm:
  - Repeatedly subtract Y from X while X-Y≥0 (or really X≥Y).
  - Quotient, Q, is simply how many subtractions were performed (i.e. count how many times we performed X=X-Y)
  - Use a subtractor to compute X-Y (if subtractor needs to borrow, then we know X-Y < 0 (or really X < Y)
- Sample Operation: X = 13, Y = 5
  - Q = 0
  - X = X – Y = 13 – 5 = 8, Q = 1
  - X = X – Y = 8 – 5 = 3, Q = 2
  - Remainder = X = 3

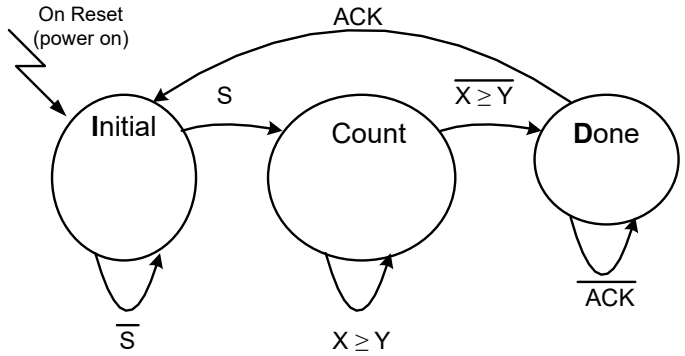
Divider Datapath

- Datapath for Divider



Divider Control Unit

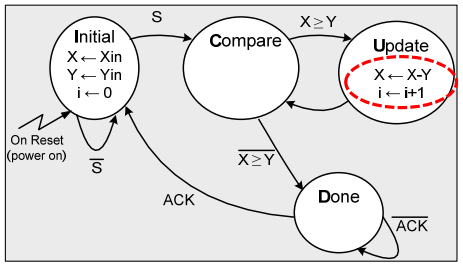
- Complete the state diagram
  - What is the logic for X_Load



Mealy vs. Moore Comparison

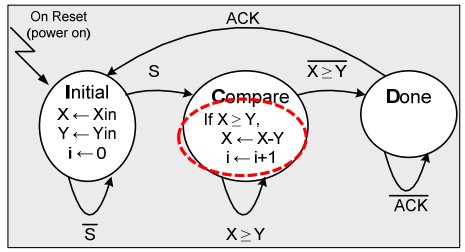
Moore Implementation

- We need to compare X with Y to determine if we should increment our quotient and update X
- If we want Moore-style enable and increment signals, we need a separate compare & update state



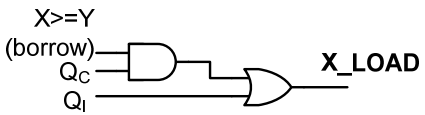
Mealy Implementation

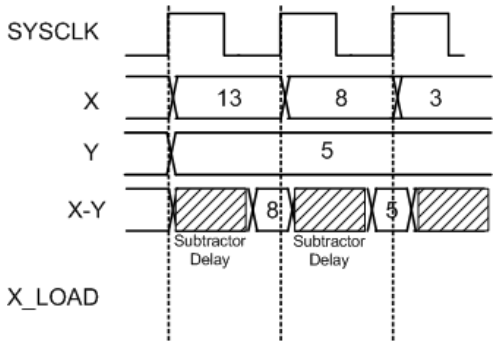
- In a Mealy-style implementation we can compare and use the result to produce the enable and increment signals in the same clock



Mealy Timing

- In Mealy-style implementation, we must ensure the clock is long enough for control signals to be produced





Control Signal Timing

- When identifying control signals in the datapath, be sure to consider if the signal should be valid

–"During the clock"

- Must be valid shortly after the beginning of the clock (e.g. mux selects, etc.)
- Usually must be produced as Moore-style output or fast, Mealy output

–"At the end of the clock"

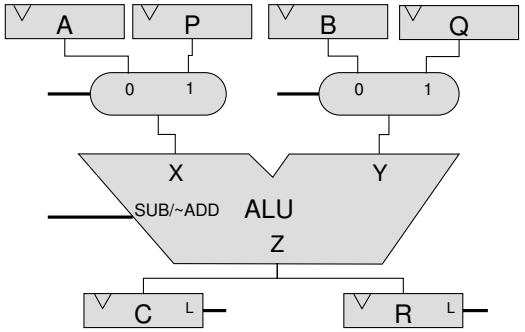
- Must be valid by the end of the clock (e.g. register load enables)
- Can be produced by combo logic in datapath

Datapath Design Example

- Design a datapath to support the following RTL (Register Transfer Level) operations
  - Identify the control signals & other datapath components

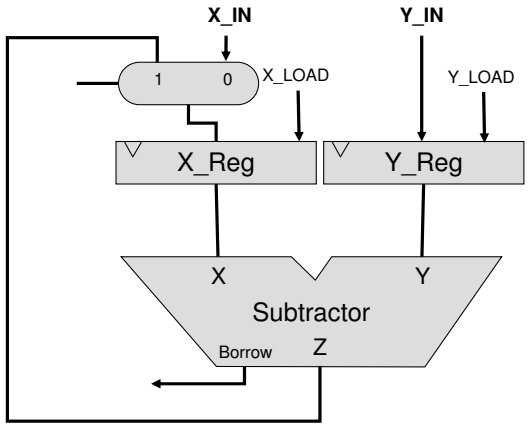
Desired Operations:

C ← A+B,	if F,G=0,0
C ← A-B,	if F,G=0,1
R ← P+Q,	if F,G=1,0
R ← P-Q,	if F,G=1,1



Divider Datapath

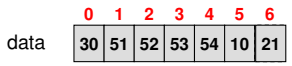
- Datapath for Divider



MIN/MAX FINDER

Min/Max Finder Description

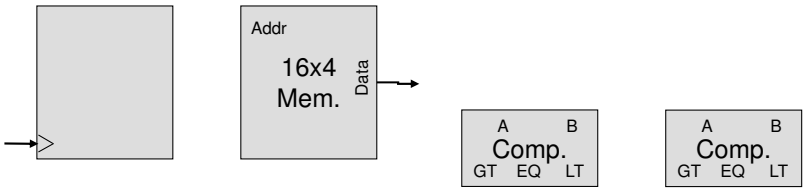
- Sixteen 4-bit unsigned numbers are stored in a 16x4 (16 rows/addresses of 4-bits each)
- Iterate over all numbers and determine the maximum (largest) and minimum (smallest) number
- First implement assuming (2) 4-bit comparators are available
- Repeat the implementation assuming (1) 4-bit comparator is available
- Remember in HW we try to perform as many operations in parallel as possible to achieve speed (e.g. perform iteration counter increment in same clock as iteration)
- How many clocks do you think we need?




```
int min = data[0];
int max = data[0];
for(int i=1; i < N; i++)
{
   if(data[i] < min)
      min = data[i];
   if(data[i] > max)
      max = data[i];
}
```

Datapath Components

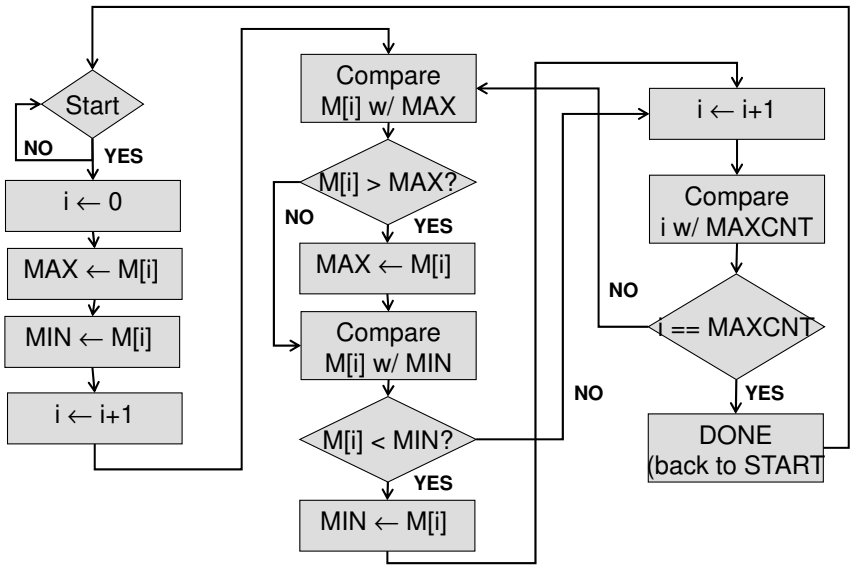
- The datapath requires…
  - Two comparators (as per our first implementation description)
  - A 16x4 memory
  - _____
  - _____



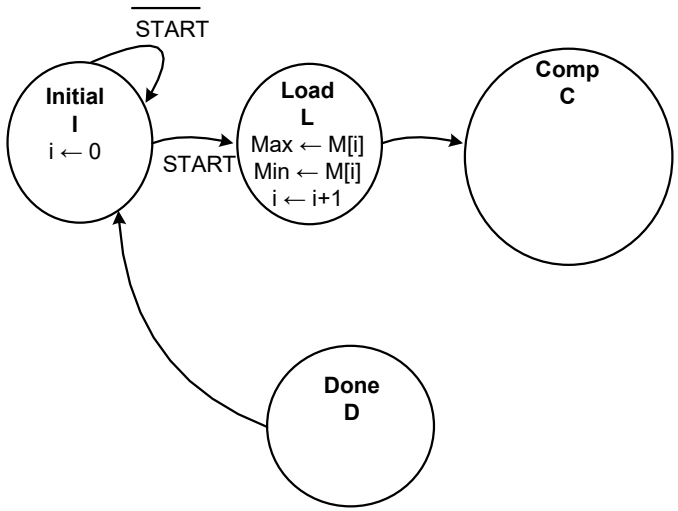
Datapath Components

- Algorithm
  - After a START signal is applied, load the zero-th (0-th) number in the memory in both MAX and MIN registers.
  - Enter an iterative loop for the first thru fifteenth numbers with both the current known MAX and MIN values, updating the registers appropriately
  - When all iterations are done (or about to be done?) go to a DONE state
- Draw a flow chart or state diagram

Flow Chart



MinMax Control Unit



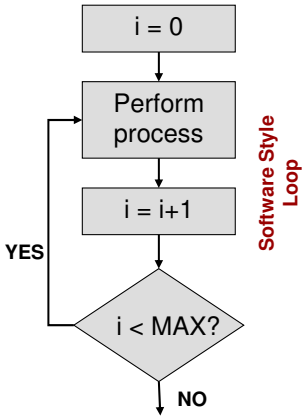
Add transition conditions Comp state operations

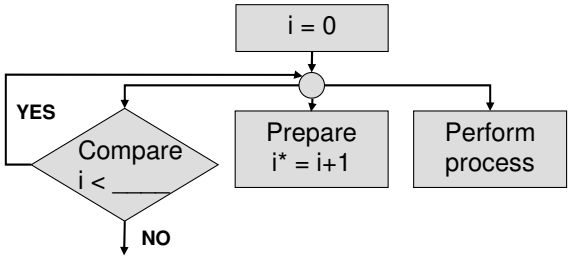
Control Signal Timing Example

- Consider the following statement of building a counter such that i increments (i ← i+1) on each clock
  - (During the clock / At the end of the clock) you enable the counter so that (during the clock / at the end of the clock) it actually increments
- Consider designing a minutes & seconds counter circuit with separate counters for each
  - **Option 1**: After 60 seconds (during 61st second) enable the minutes counter
  - **Option 2**: During the 60th second (after 59 seconds) enable the minutes counter

Early or Late?

- Consider a counting loop to iterate MAX times
- In hardware we try to perform as many operations in parallel as possible
- To iterate MAX times, what should we compare with i?

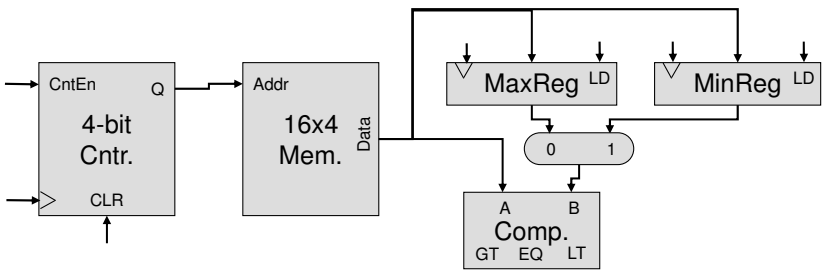




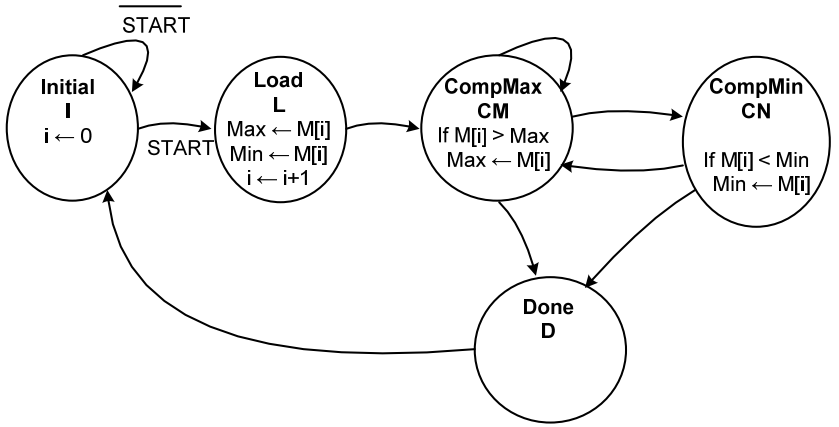
Hardware Style Loop

1 Comparator Datapath

- The datapath now requires…
  - One comparator
  - A 16x4 memory
  - (2) registers to store current min / max
  - (1) 4-bit counter to counter iterations & address memory



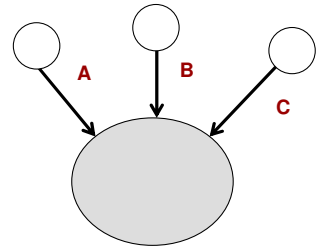
1-Comparator Implementation

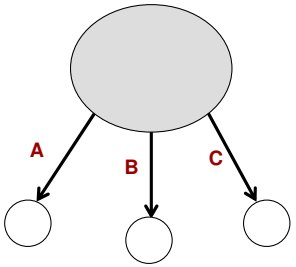


Add transition conditions and add counter increment statements

Transition Conditions

- Is there any relationship between conditions associated with incoming transitions or outgoing transitions?
- Outgoing transitions must be
  - _____ (____ conditions true)
  - _____ (____ conditions true)



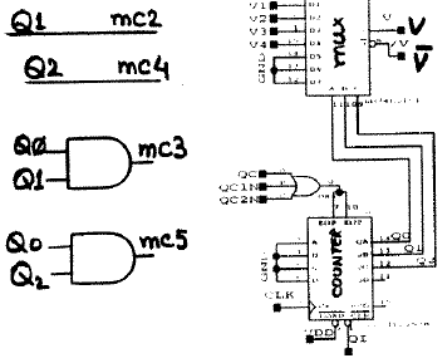


Example: Vote Counting Machine

1.1 Majority vote for a **5-vote** case: In your homework #8, you have designed a state machine to inspect *four votes* **serially** to find majority vote. This is a similar problem with **FIVE votes** *instead of* *FOUR votes*. Majority means **three YES votes** in the five-vote case also. The six states are:

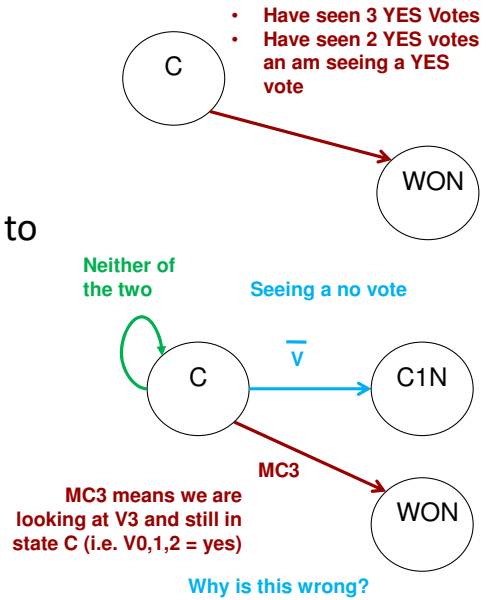
I	=	INITIAL
C	=	COUNTING votes
C1N	=	continuing COUNTING votes *after* seeing one (1) NO vote
C2N	=	continuing COUNTING votes *after* seeing two (2) NO votes
W	=	WON the majority vote
L	=	LOST the majority vote

Complete the state diagram.



Vote State Machine Conditions

- Remain in C until…
  - Find a NO vote…
    - …then go to C1N
  - Have enough YES votes to guarantee victory…
    - …then go to WON



Control / Datapath Interaction

- We have used the analogy of the control unit as a manager and datapath as workers
- Consider the DPU as individual workers (plumbers, electricians,…) who need to be told what to do each hour (each clock cycle)
- The control unit uses the current state (updated each clock cycle) to determine what work (control signals) should be performed each hour (clock cycle)
- Control signals generated by state machines may fall into one of two sets:
  - Mealy-style outputs
  - Moore-style outputs

Control Output Summary

- A _____-style output is conditioned upon the current state AND other input signals
- A _____-style output is conditioned only upon the current state
- In synchronous digital systems, you must prepare a control signal _____ so the effect takes place at the _____ (beginning of next clock)

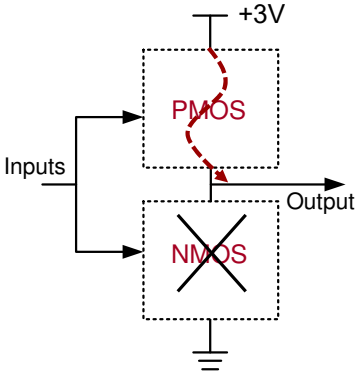
TRI-STATE OUTPUTS & BUSES

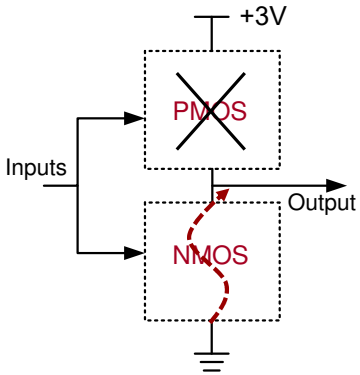
Typical Logic Gate

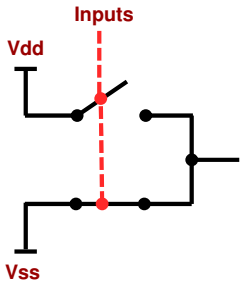
- Gates can output two values: 0 & 1
  - Logic '1' (Vdd = 3V), or Logic '0' (Vss = GND)
- Analogy: a sink faucet
  - 2 possibilities: Hot ('1') or Cold ('0')
- Inputs cause *EITHER* a pathway from output to VDD *OR* VSS

Hot Water = Logic 1

Cold Water = Logic 0

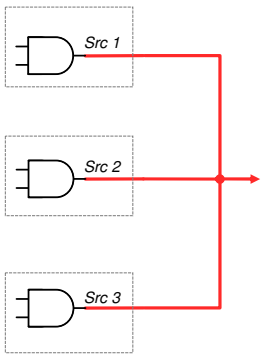


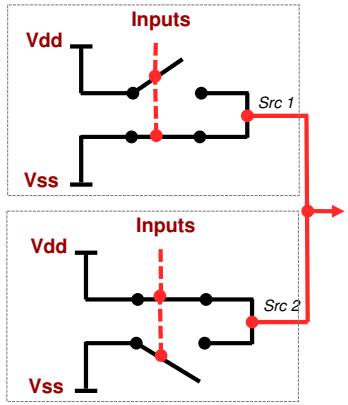




Output Connections

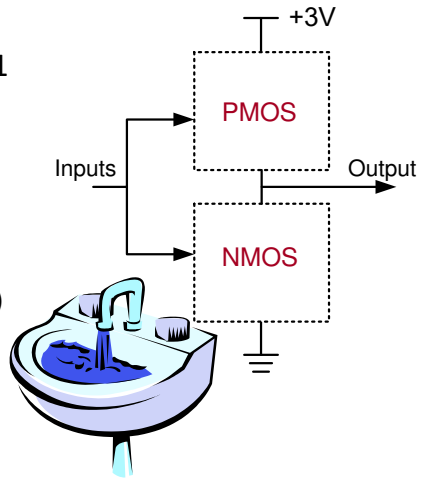
- Can we connect the output of two logic gates together?





Tri-State Buffers

- Gates can output two values: 0 & 1
  1. Logic 0 = 0 volts
  2. Logic 1 = 5 volts
- Tristate buffers can output a third value:
  3. ____ = _____
     (no connection to any voltage source)
- Analogy: a sink faucet
  - 3 possibilities:
    1.) Hot water,
    2.) Cold water,
    3.) _____

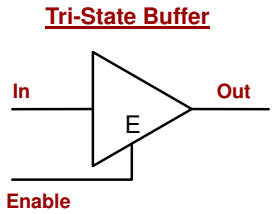


Hot Water = Logic 1

Cold Water = Logic 0

Tri-State Buffers

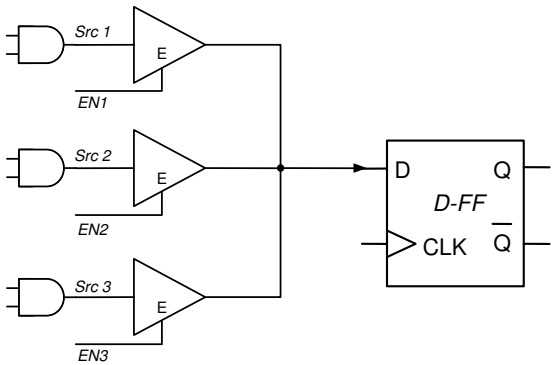
- Tri-state buffers have an extra enable input
- When disabled, output is Z
- When enabled, normal buffer



En	In	Out
0	x	Z
1	0	0
1	1	1

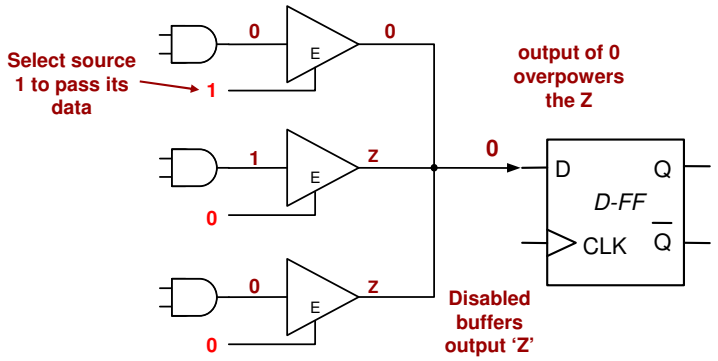
Tri-State Buffers

- We use tri-state buffers to share one output amongst several sources
- Rule: _____



Tri-State Buffers

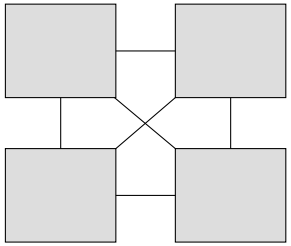
- We use tri-state buffers to share one output amongst several sources
- When 1 buffer enabled, its output overpowers the Z's (no connection) from the other gates

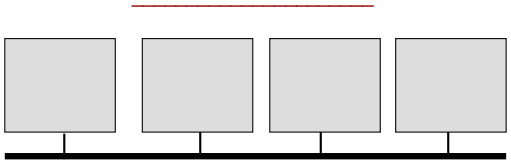


Communication Connections

- Multiple entities need to communicate
- We could use
  - Point-to-point connections
  - A _____

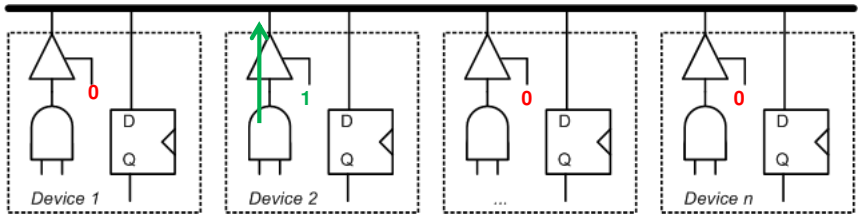
Separate point to point connections





Bidirectional Bus

- ___ transmitter (otherwise bus contention)
- N receivers



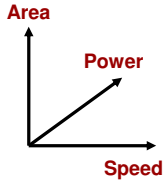
Tri-State Buffer / Bussing Summary

- Provide a 3rd output value: _____
- Allows multiple outputs to be wired together
- Only _____ bus driver can be enabled at a time

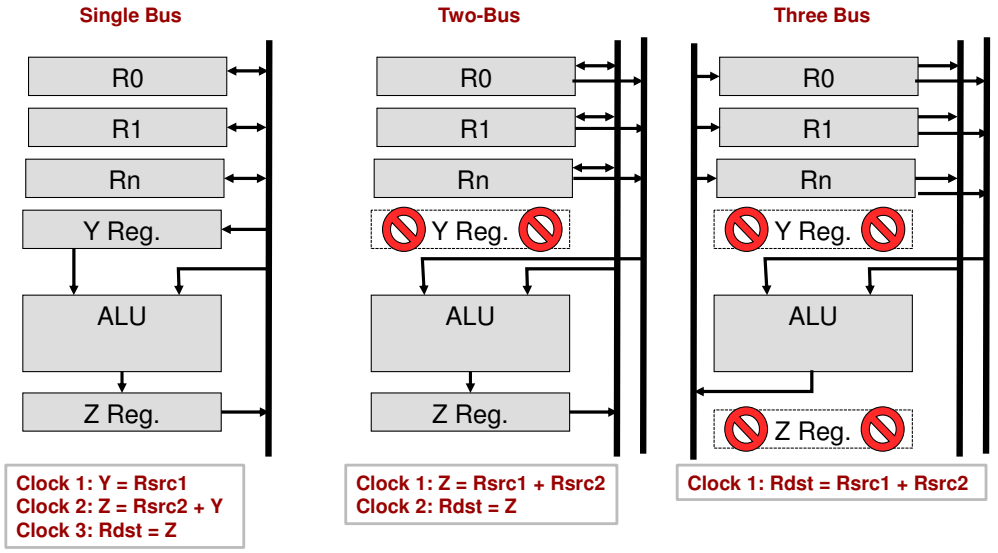
MICROARCHITECTURE EXAMPLE

Digital Design Goals

- Digital systems seek to optimize a design along these three axes:
  - Area (size)
  - Speed
  - Power Consumption
- Can often only optimize one or two of these without sacrificing the other(s)
  - Just as in software design, there is a classic time/space trade-off
  - Microarchitecture can determine where a design falls in this trade space



Different Architectures



General Implications: Less Resources => More Clock Cycles (Time)