# Chapter #8: Finite State Machine Design

***Contemporary Logic Design***



**University of Calif eley**



**June 1993**

## Motivation

- ***Counters:*** **Sequential Circuits where State = Output**

- ***Generalizes to Finite State Machines:***
  **Outputs are Function of State (and Inputs)**
  **Next States are Functions of State and Inputs**
  **Used to implement circuits that control other circuits**
  **"Decision Making" logic**



- ***Application of*** **niques**
  **Word Proble**
  **Mapping into formal representa ehavior**
  **Case Studies**

## Chapter Overview

***Concept of the State Machine***

- **Partitioning into Datapath and Control**
- **When Inputs are Sampled and Outputs Asserted**

***Basic Design Approach***

- **Six Step Design Process**



***Alternative State M***

- **State Diagram, ASM Notation, VH cription Language**

***Moore and Mealy Machines***

- **Definitions, Implementation Examples**

***Word Problems***

- **Case Studies**

## Concept of the State Machine

**Computer Hardware = Datapath + Control**

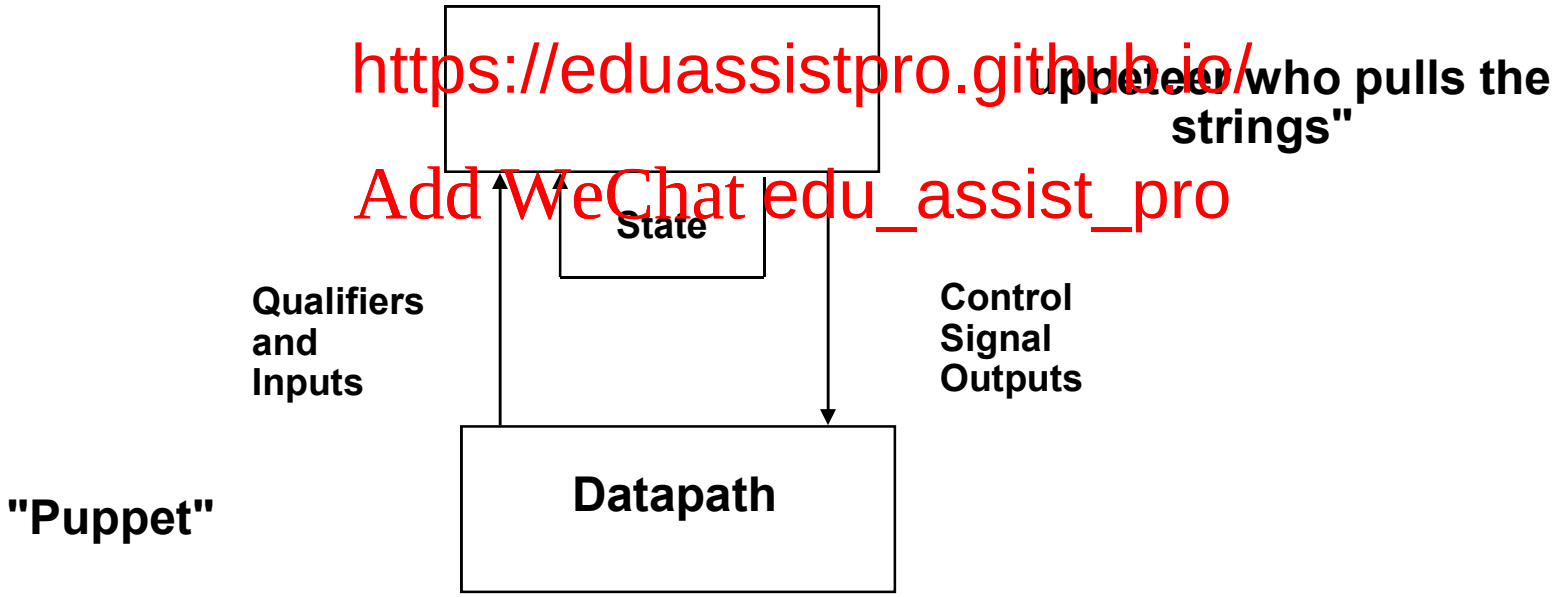
**Registers**
**Combinational Functional Units (e.g., ALU)**
**Busses**

***Qualifiers*** →

← ***Control***

**FSM generating sequences of control signals**
**Instructs datapath what to do next**

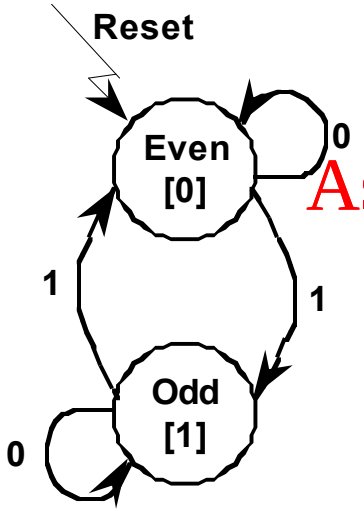




## Concept of the State Machine

***Example: Odd Parity Checker***

**Assert output whenever input bit stream has odd # of 1's**




**State Diagram**

| Present State | Input | Next State | Output |
|---|---|---|---|
| Even | 0 | Even | 0 |
| Even | 1 | Odd | 0 |
| Odd | 0 | Odd | 1 |
| Odd | 1 | Even | 1 |

**ransition Table**

| Present Stat | | xt State | Output |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |

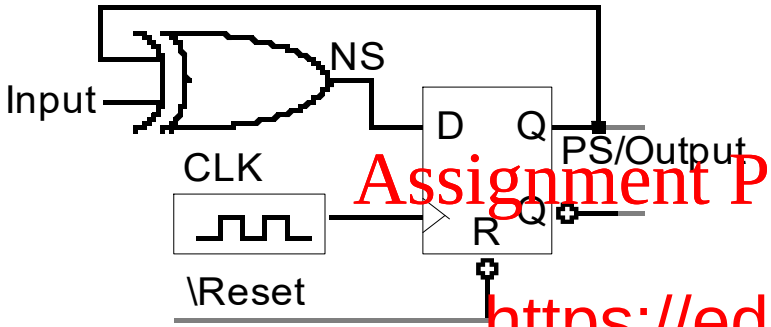
**Encoded State Transition Table**

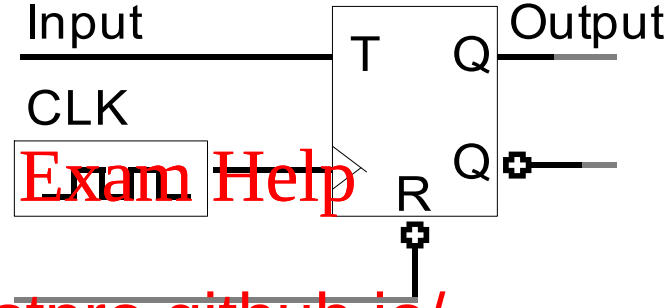
# Concept of the State Machine

***Example: Odd Parity Checker***

**Next State/Output Functions**

**NS = PS xor PI;  OUT = PS**



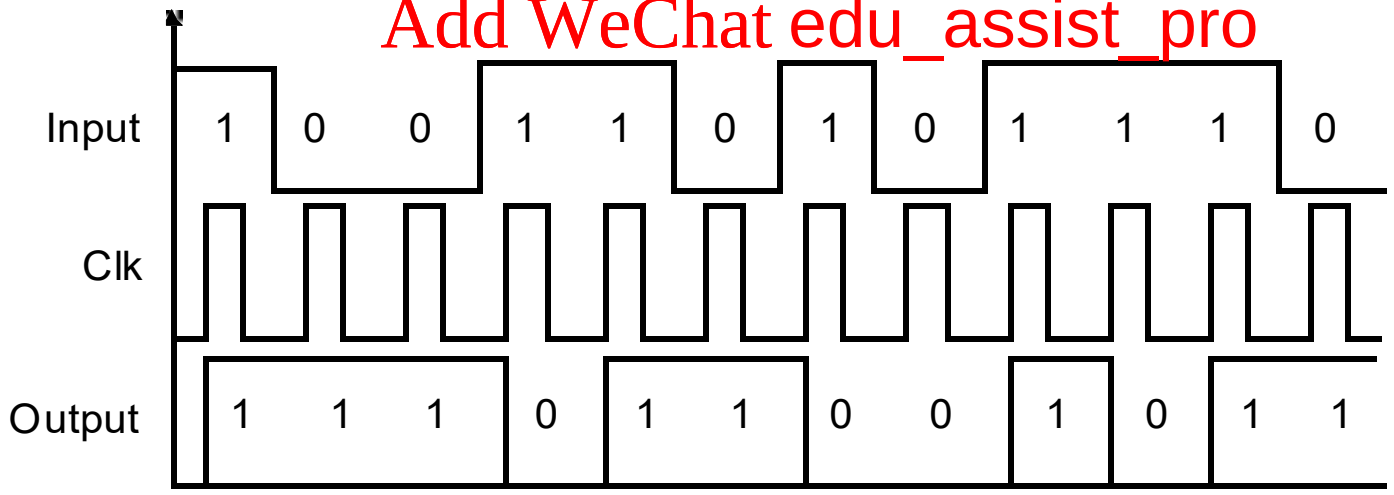






**D FF Implementation**

**Implementation**






**Timing Behavior: Input 1 0 0 1 1 0 1 0 1 1 1 0**

## Concept of State Machine

***Timing:***

**When are inputs sampled, next state computed, outputs asserted?**

***State Time:*** **Time between clocking events**

- **Clocking event causes state/outputs to transition, based on inputs**
- **For set-up/hold time considerations:**

  **Inputs sho** **event**
- **After propagation delay, Next St** **utputs are stable**

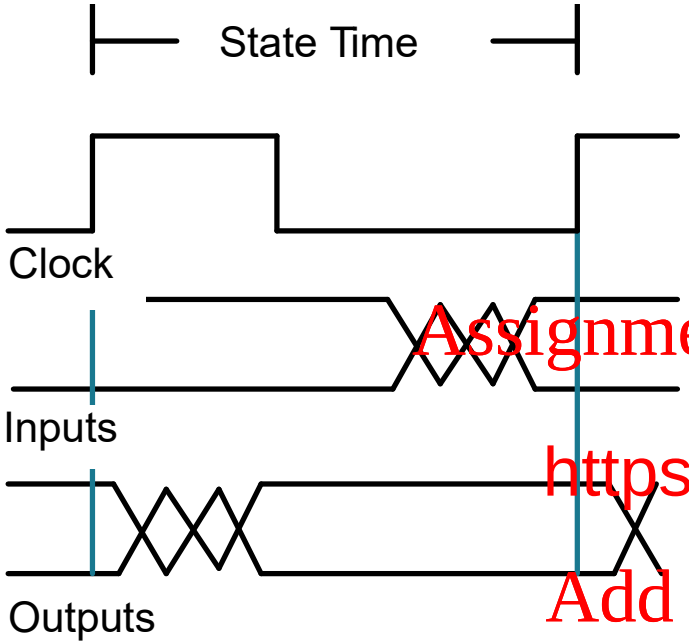


**NOTE: Asynchronous signals take effect immediately**
**Synchronous signals take effect at the next clocking event**

**E.g., tri-state enable: effective immediately**
**sync. counter clear: effective at next clock event**

## Concept of State Machine

***Example:*** **Positive Edge Triggered Synchronous System**




**On rising edge, inputs sampled**
**outputs, next state computed**

**After propagation delay, outputs and**
**next state are stable**

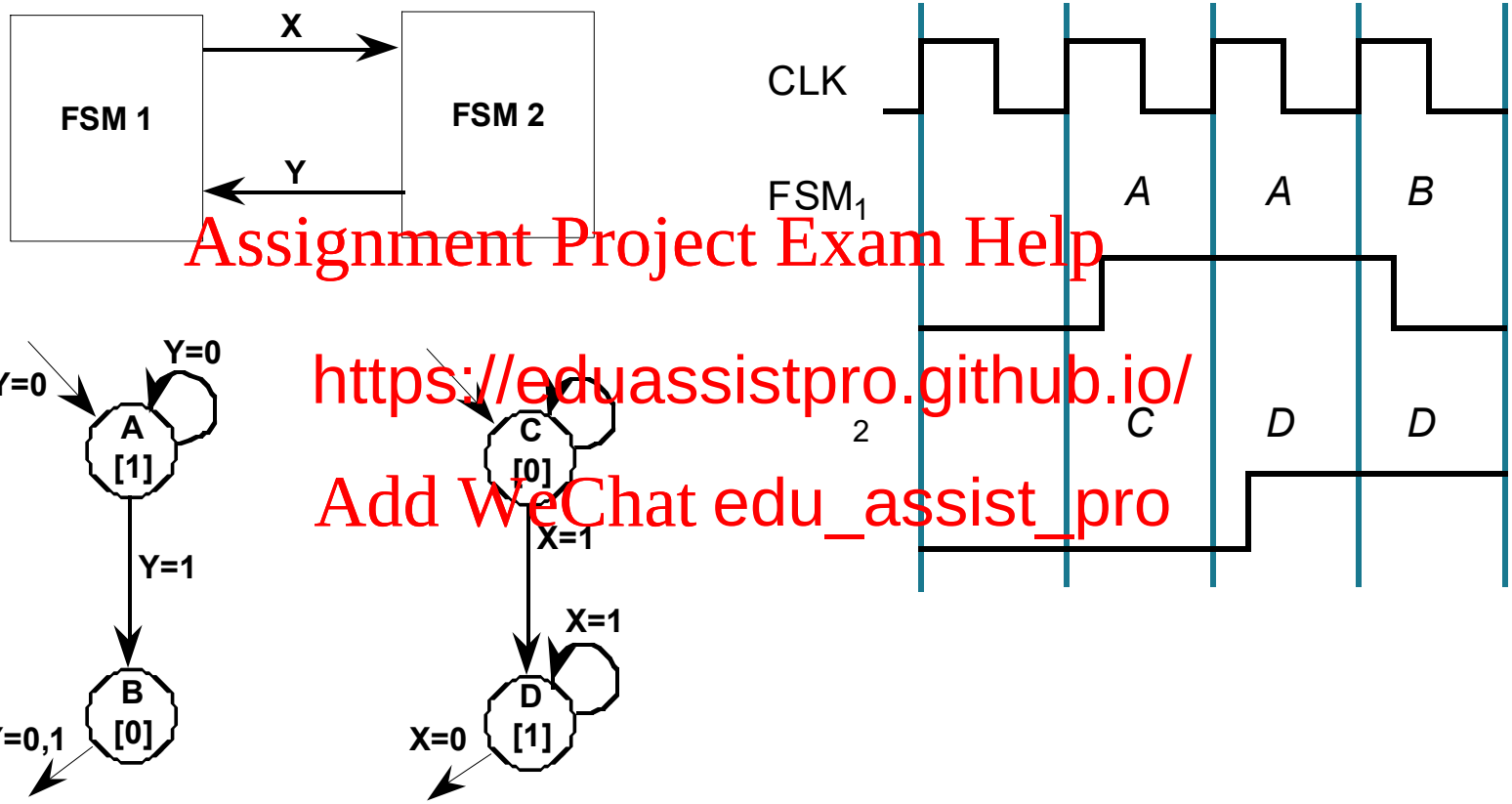
***Immediate Outputs:***
**affect datapath immediately**
**from datapath to change**

**take eff ock edge**
**propag ust exceed hold times**

# Concept of the State Machine

## *Communicating State Machines*

**One machine's output is another machine's input**






**Machines advance in lock step**

**Initial inputs/outputs: X = 0, Y = 0**

## Basic Design Approach

***Six Step Process***

1. **Understand the statement of the Specification**
2. **Obtain an abstract specification of the FSM**
3. **Perform a state mininimization**
4. **Perform state**
5. **Choose FF types to implement F ter**
6. **Implement the FSM**



***1, 2 covered now; 3, 4, 5 covered later;***
***4, 5 generalized from the counter design procedure***

## Basic Design Approach

***Example:*** **Vending Machine FSM**

**General Machine Concept:**

**deliver package of gum after 15 cents deposited**

**single coin slot for dimes, nickels**

**no change**



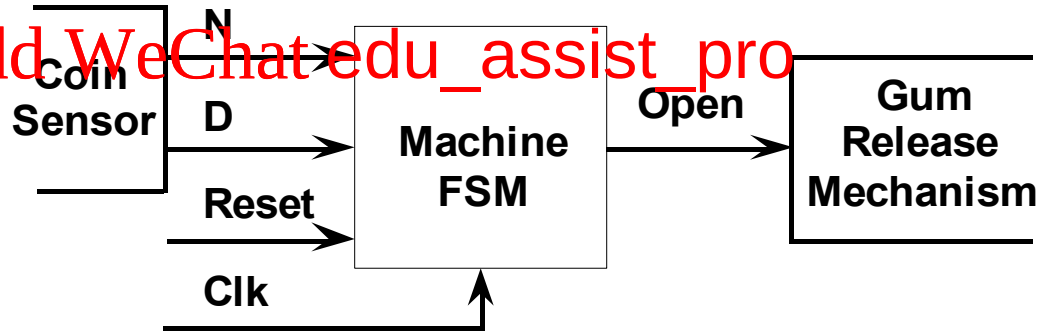
**Step 1. *Understa***

**Draw a pictu**



***Block Diagram***





## Vending Machine Example

**Step 2.** ***Map into more suitable abstract representation***

***Tabulate typical input sequences:***

**three nickels**
**nickel, dime**
**dime, nickel**
**two dimes**
**two nickels, dime**



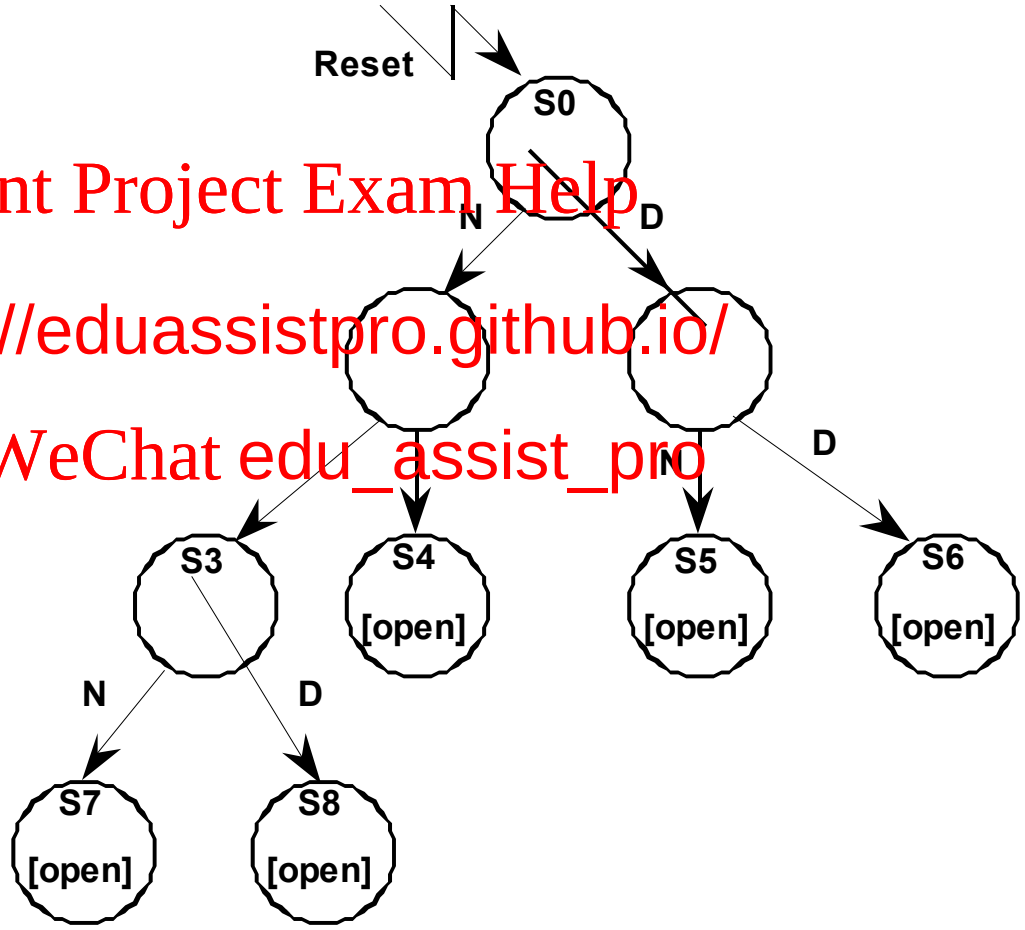
***Draw state diagra***

**Inputs: N, D, re**



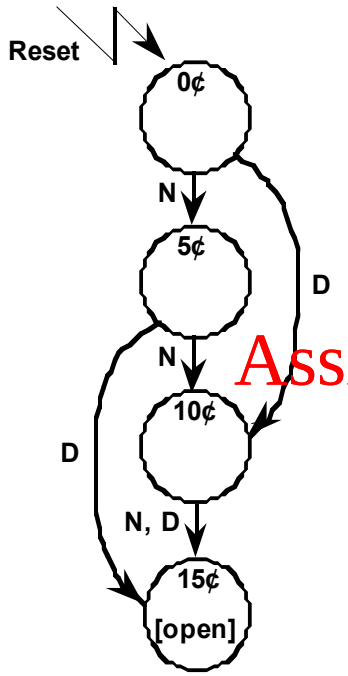
**Output: open**





# Vending Machine Example

## Step 3: State Minimization




**reuse states whenever possible**

| Present State | Inputs D | Inputs N | Next State | Output Open |
|---|---|---|---|---|
| 0¢ | 0 | 0 | 0¢ | 0 |
| | 0 | 1 | 5¢ | 0 |
| | 1 | 0 | 10¢ | 0 |
| | 1 | 1 | X | X |
| 5¢ | 0 | 0 | 5¢ | 0 |
| | 0 | 1 | 10¢ | 0 |
| | 1 | 0 | 15¢ | 0 |
| | | | X | X |
| 10¢ | | | 10¢ | 0 |
| | | | 15¢ | 0 |
| | | | 15¢ | 0 |
| | | | X | X |
| 15¢ | X | X | 15¢ | 1 |

**Symbolic State Table**

# Vending Machine Example

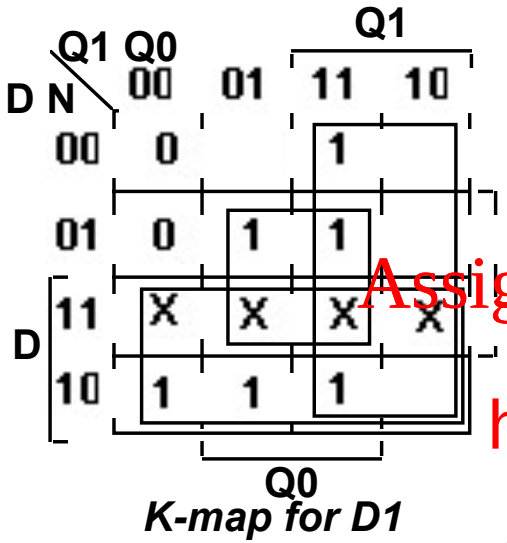
## Step 4: State Encoding

| Present State $Q_1$ | Present State $Q_0$ | Inputs D | Inputs N | Next State $D_1$ | Next State $D_0$ | Output Open |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 0 | 1 | 0 | 1 | 0 |
| | | 1 | 0 | 1 | 0 | 0 |
| | | 1 | 1 | X | X | X |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| | | 0 | 1 | 1 | 0 | 0 |
| | | | | | | 0 |
| | | | | | | X |
| 1 | 0 | | | | | 0 |
| | | 0 | 1 | | | 0 |
| | | 1 | 0 | | | 0 |
| | | 1 | 1 | | | X |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| | | 0 | 1 | 1 | 1 | 1 |
| | | 1 | 0 | 1 | 1 | 1 |
| | | 1 | 1 | X | X | X |

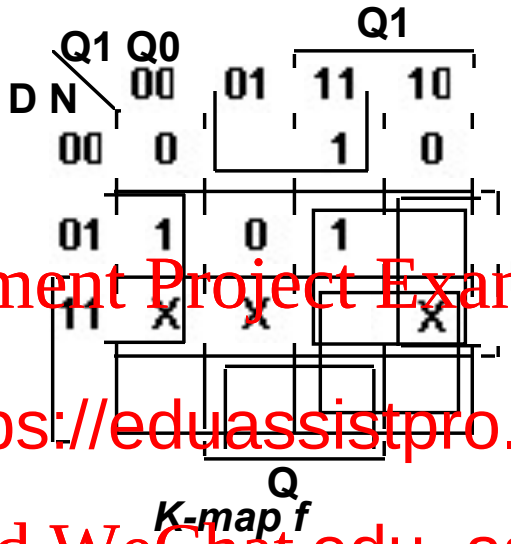
# Vending Machine Example

## Step 5. *Choose FFs for implementation*

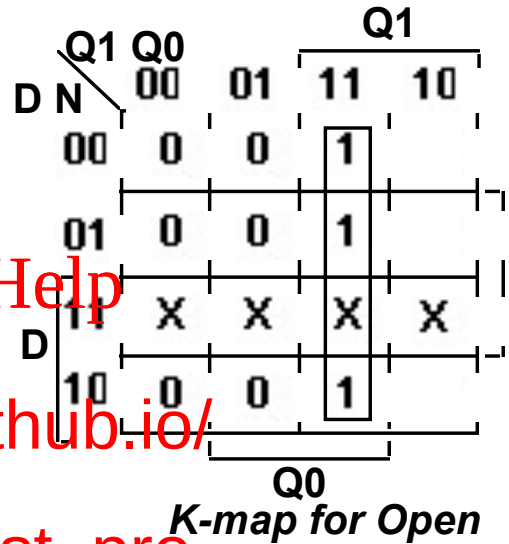
**D FF easiest to use**




*K-map for D1*

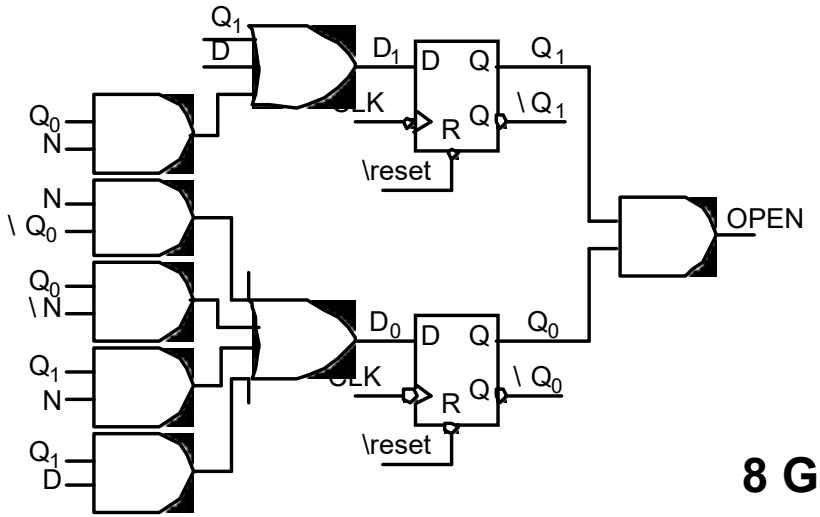



*K-map f*




*K-map for Open*






$$D1 = Q1 + D + Q0\,N$$

$$D0 = N\,\overline{Q0} + Q0\,\overline{N} + Q1\,N + Q1\,D$$

$$OPEN = Q1\,Q0$$

**8 Gates**

## Vending Machine Example

**Step 5.** ***Choosing FF for Implementation***

**J-K FF**

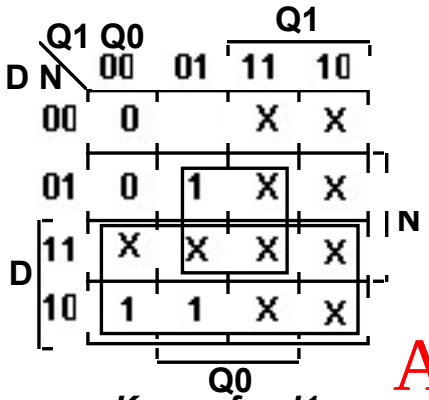
| Present State $Q_1$ | Present State $Q_0$ | Inputs D | Inputs N | Next State $D_1$ | Next State $D_0$ | $J_1$ | $K_1$ | $J_0$ | $K_0$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | 0 | X |
| | | 0 | 1 | 0 | 1 | 0 | X | 1 | X |
| | | 1 | 0 | 1 | 0 | 1 | X | 0 | X |
| | | | | | | | | X | X |
| 0 | 1 | | | | | | | X | 0 |
| | | | | | | | | X | 1 |
| | | 1 | 0 | 1 | | | | X | 0 |
| | | 1 | 1 | X | X | | | X | X |
| 1 | 0 | 0 | 0 | 1 | | | | 0 | X |
| | | 0 | 1 | 1 | 1 | X | 0 | 1 | X |
| | | 1 | 0 | 1 | 1 | X | 0 | 1 | X |
| | | 1 | 1 | X | X | X | X | X | X |
| 1 | 1 | 0 | 0 | 1 | 1 | X | 0 | X | 0 |
| | | 0 | 1 | 1 | 1 | X | 0 | X | 0 |
| | | 1 | 0 | 1 | 1 | X | 0 | X | 0 |
| | | 1 | 1 | X | X | X | X | X | X |



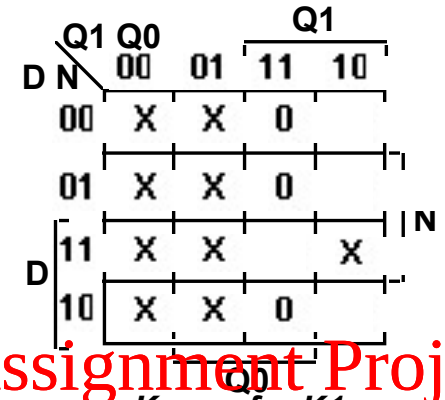
**Remapped encoded state transition table**

# Vending Machine Example

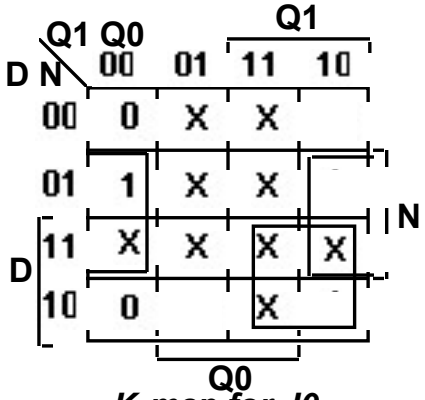
***Implementation:***



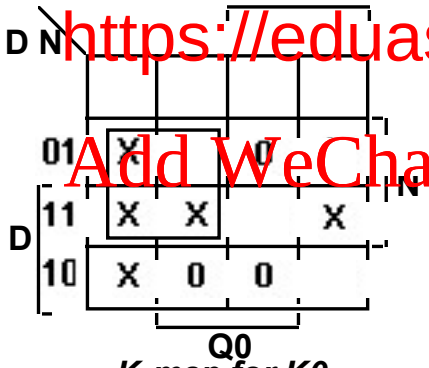

*K-map for J1*




*K-map for K1*




*K-map for J0*




*K-map for K0*

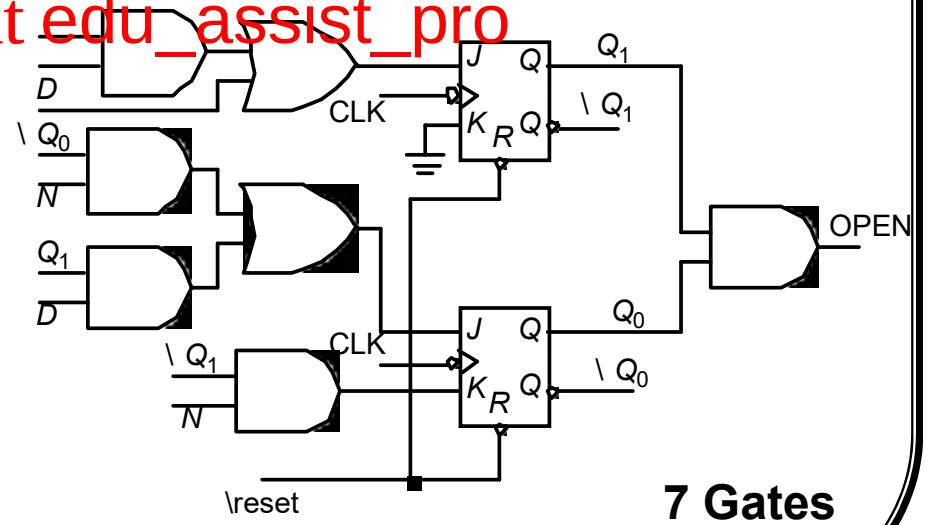
**J1 = D + Q0 N**

**K1 = 0**

**J0 = $\overline{Q0}$ N + Q1 D**

**K0 = Q1 N**

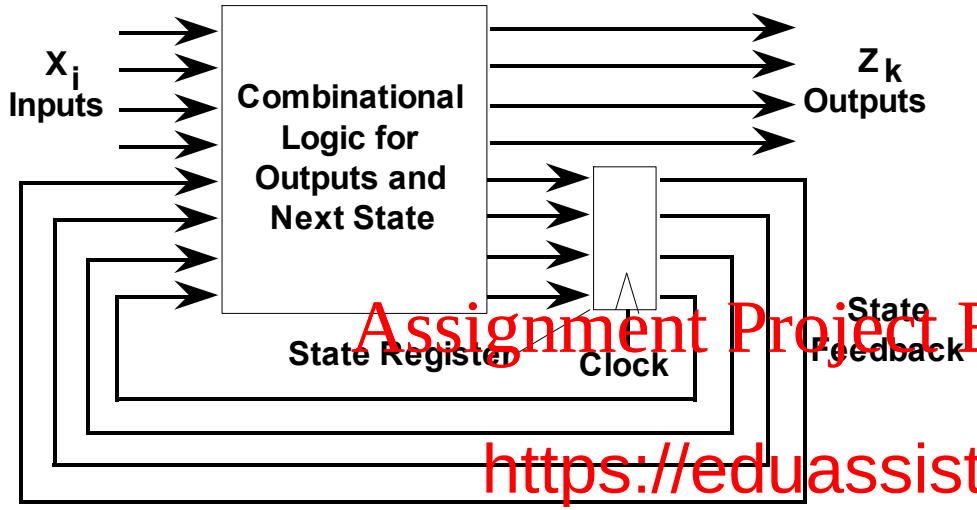





**7 Gates**

# Moore and Mealy Machine Design Procedure

## *Definitions*



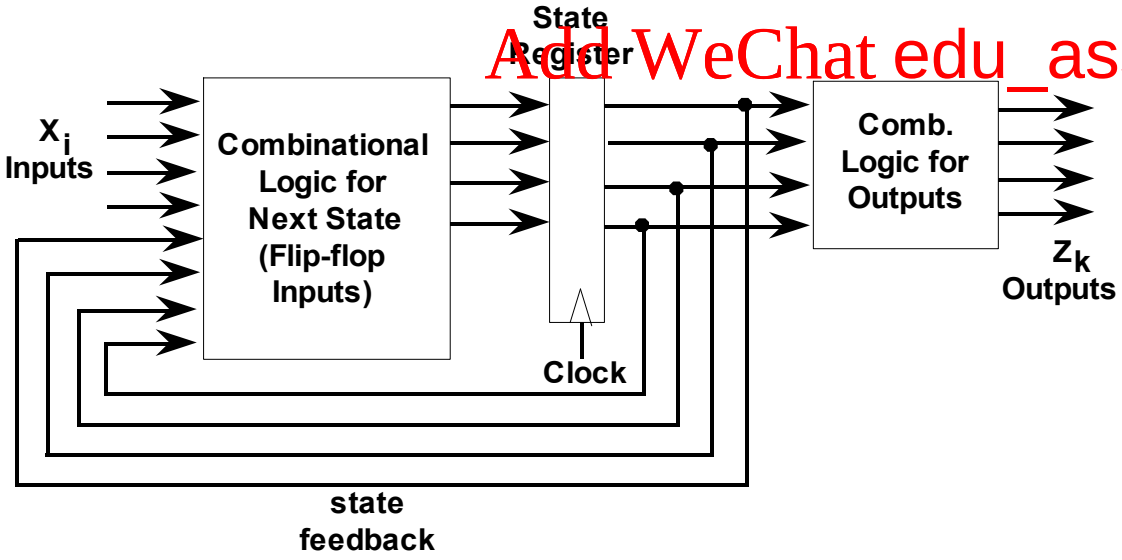

**Mealy Machine**

***Outputs depend on state AND inputs***

***Input change causes an immediate output change***

***Asynchronous signals***






**Moore Machine**

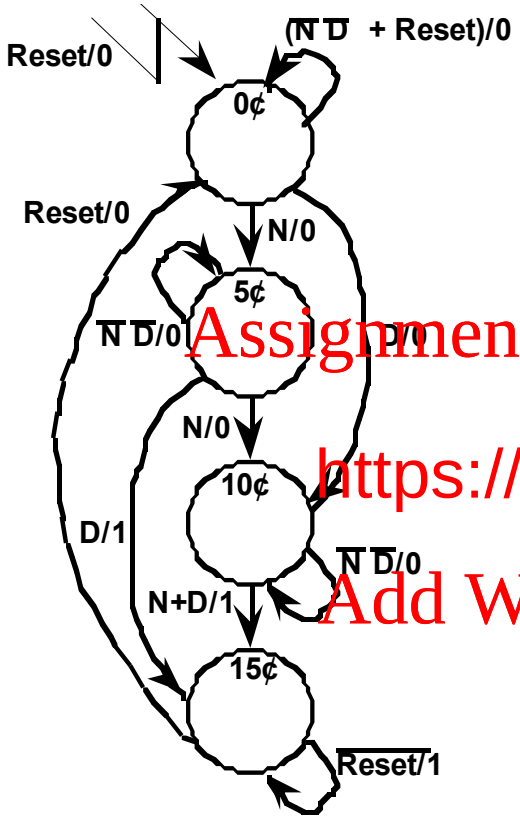
***Outputs are function solely of the current state***

***Outputs change synchronously with state changes***

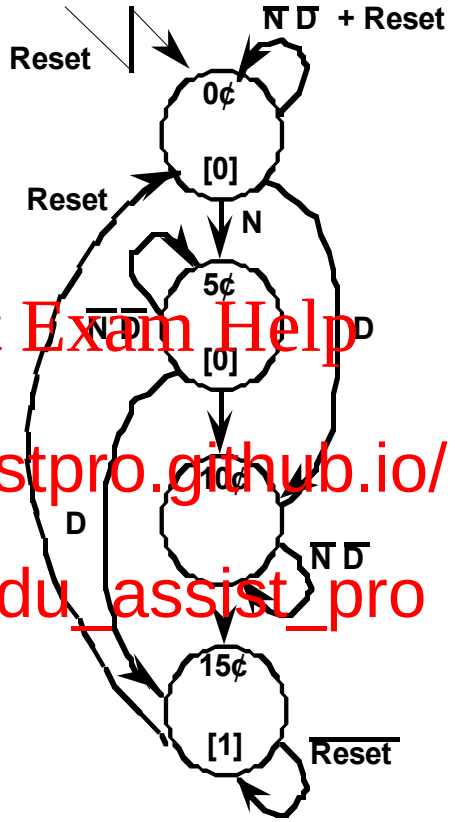
# Moore and Mealy Machines

## *State Diagram Equivalents*

**Mealy Machine**




**Outputs are associated with Transitions**




**Moore Machine**

**Outputs are associated with State**

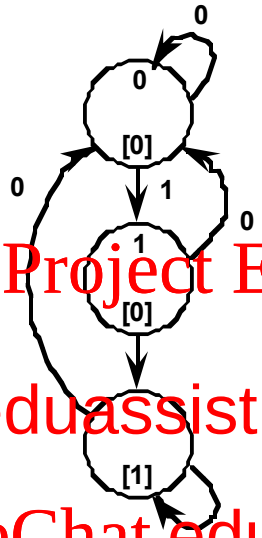
# Moore and Mealy Machines

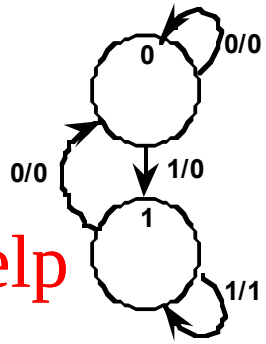
## *States vs. Transitions*

**Mealy Machine typically has fewer states than Moore Machine for same output sequence**

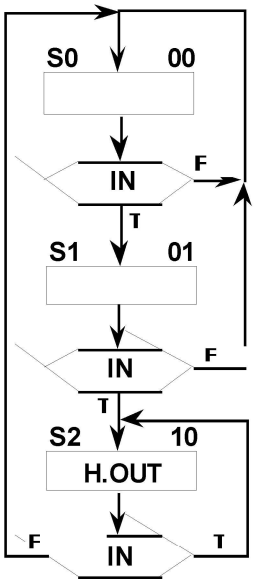
**Same I/O behavior: Assert a single output whenever at least two 1's have been received in a sequence.**

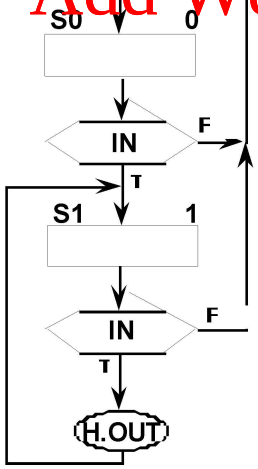
**Different # of state**





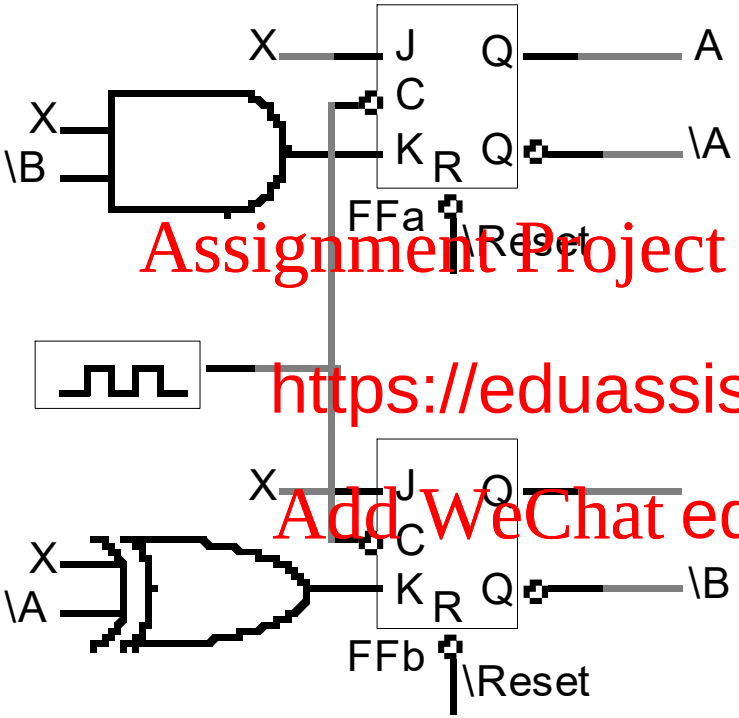




**Equivalent ASM Charts**

# Moore and Mealy Machines

## *Timing Behavior of Moore Machines*

**Reverse engineer the following:**




**Input X**
**Output Z**
**State A, B = Z**



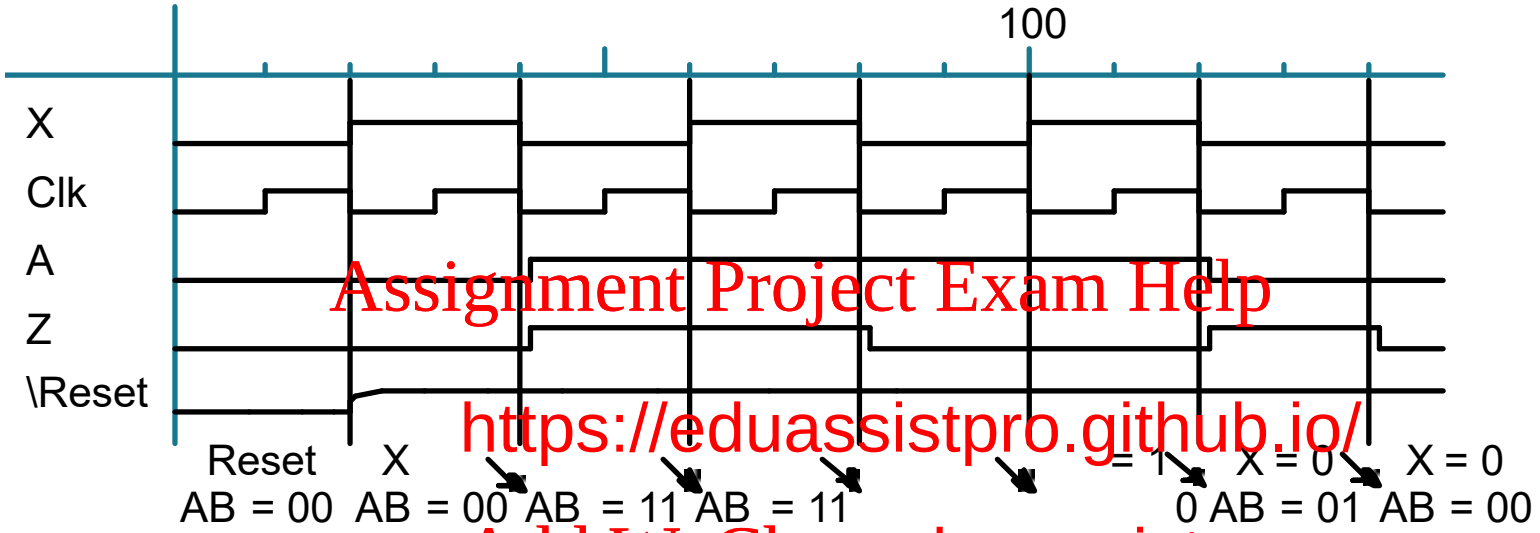
**Two Techniques for Reverse Engineering:**

- **Ad Hoc: Try input combinations to derive transition table**
- **Formal: Derive transition by analyzing the circuit**

# Moore and Mealy Machines

## *Ad Hoc Reverse Engineering*

**Behavior in response to input sequence 1 0 1 0 1 0:**






***Partially Derived State Transition Table***

| A | B | X | A+ | B+ | Z |
|---|---|---|---|---|---|
| 0 | 0 | 0 | ? | ? | 0 |
| | | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| | | 1 | ? | ? | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| | | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| | | 1 | 1 | 0 | 1 |

# Moore and Mealy Machines

## *Formal Reverse Engineering*

**Derive transition table from next state and output combinational functions presented to the flipflops!**

$Ja = X$ $\quad$ $Ka = X \cdot \overline{B}$ $\quad$ $Z = B$

$Jb = X$ $\quad$ $Kb = X \text{ xor } A$

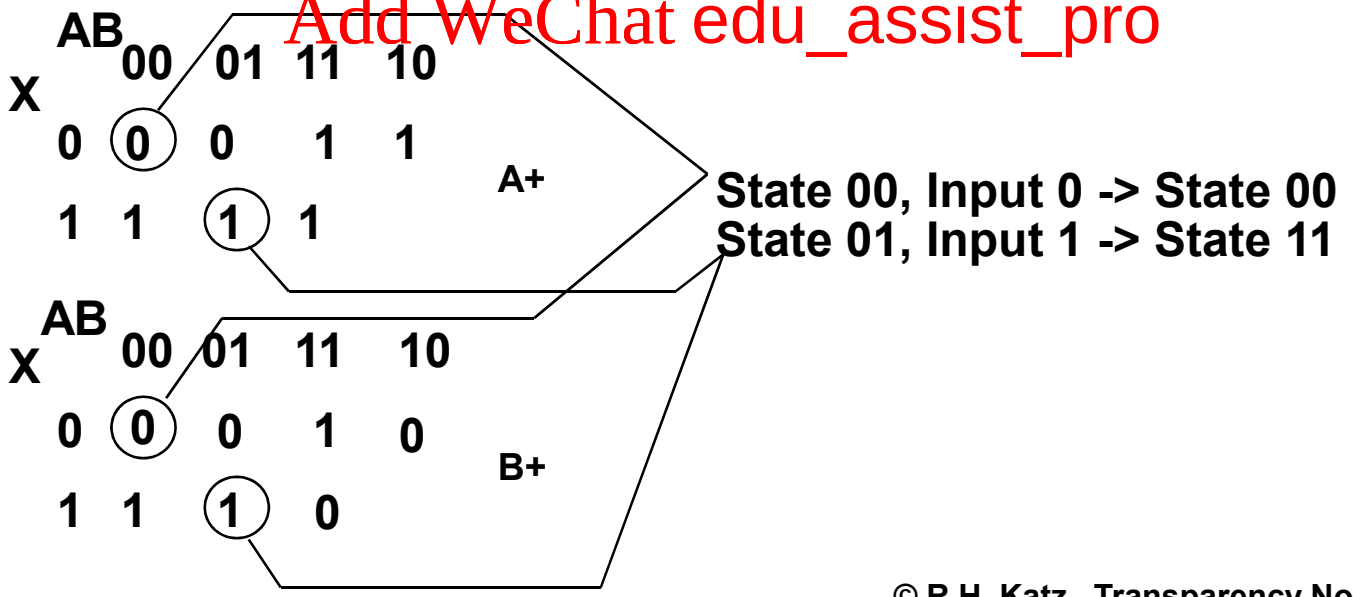
**FF excitation equations for J-K flipflop:**

$A+ = Ja \cdot \overline{A} + \overline{Ka} \cdot A = X \cdot \overline{A} + (\overline{X} + A) \cdot A$

$B+ = Jb \cdot \overline{B} + \overline{Kb} \cdot B = X \cdot \overline{B} + (\overline{X \text{ xor } A}) \cdot B$

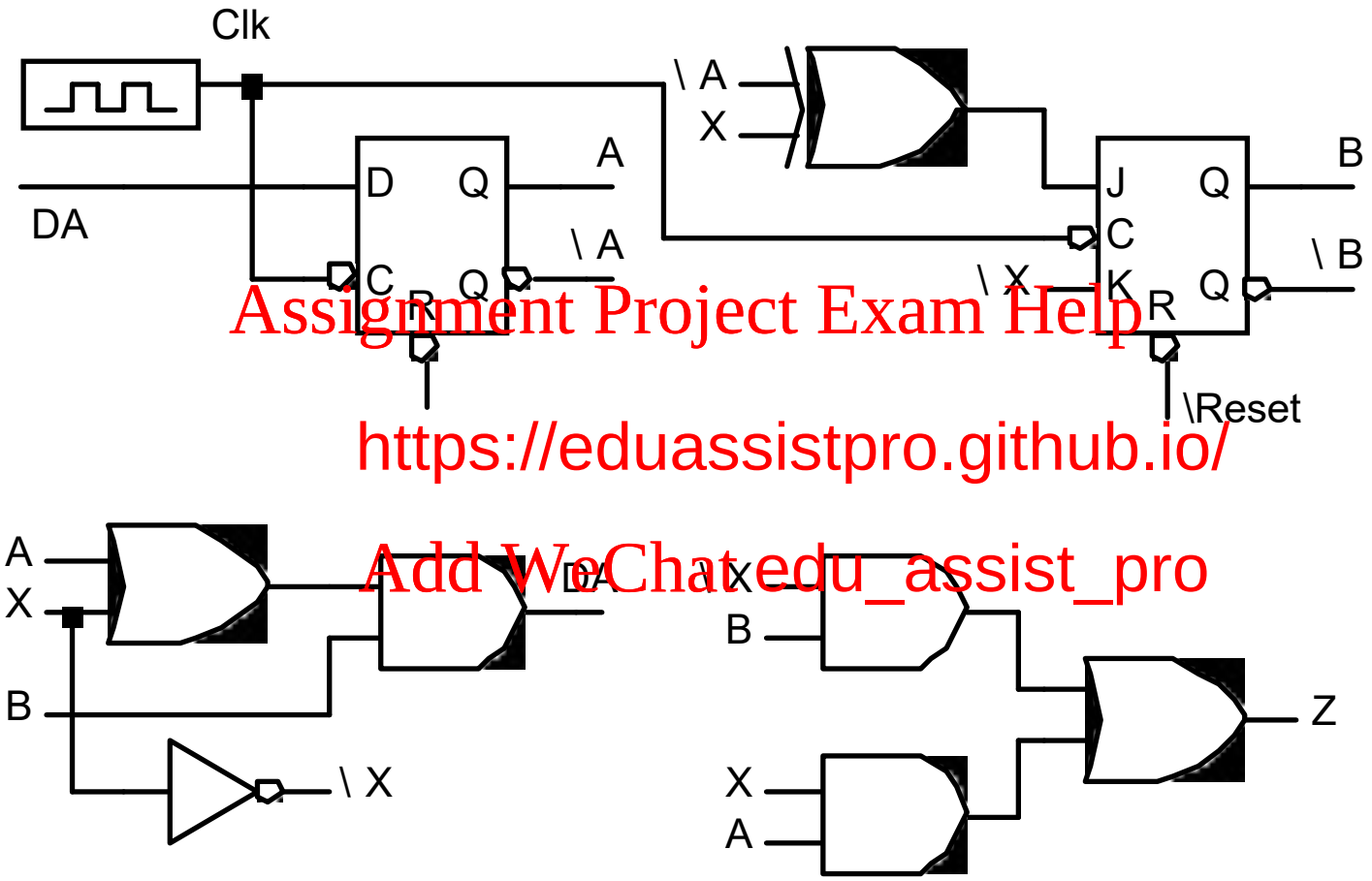


**Next State K-Maps:**



# Moore and Mealy Machines

## *Reverse Engineering a Mealy Machine*





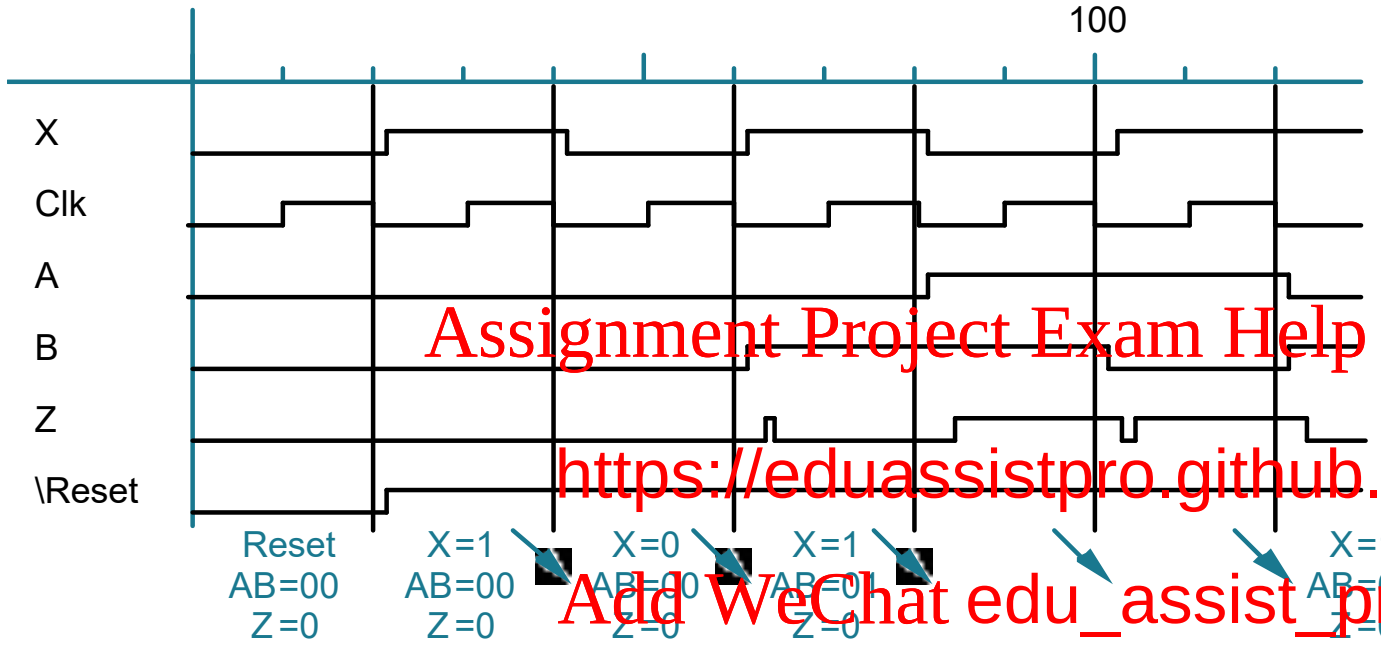

**Input X, Output Z, State A, B**

**State register consists of D FF and J-K FF**

# Moore and Mealy Machine

## *Ad Hoc Method*

### Signal Trace of Input Sequence 101011:






***Note glitches in Z!***

***Outputs valid at following falling clock edge***

***Partially completed state transition table based on the signal trace***

| A | B | X | A+ | B+ | Z |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 |
| | | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | ? | ? | ? |
| | | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | ? | ? | ? |
| | | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| | | 1 | ? | ? | ? |

## Moore and Mealy Machines

### *Formal Method*

$$A+ = B \cdot (A + X) = A \cdot B + B \cdot X$$

$$B+ = Jb \cdot \overline{B} + \overline{Kb} \cdot B = (\overline{A} \text{ xor } X) \cdot \overline{B} + X \cdot B$$

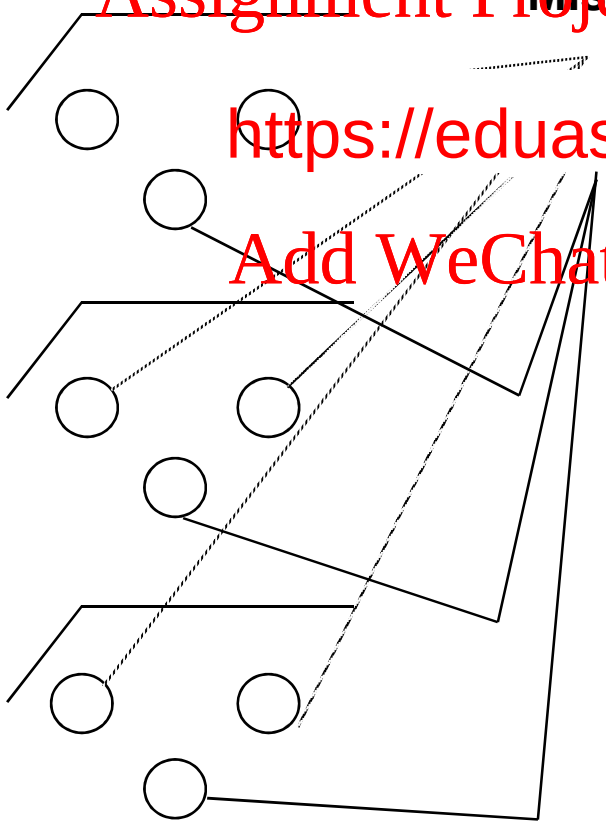
$$= A \cdot \overline{B} \cdot X + \overline{A} \cdot \overline{B} \cdot \overline{X} + B \cdot X$$

$$Z = A \cdot X + B \cdot \overline{X}$$



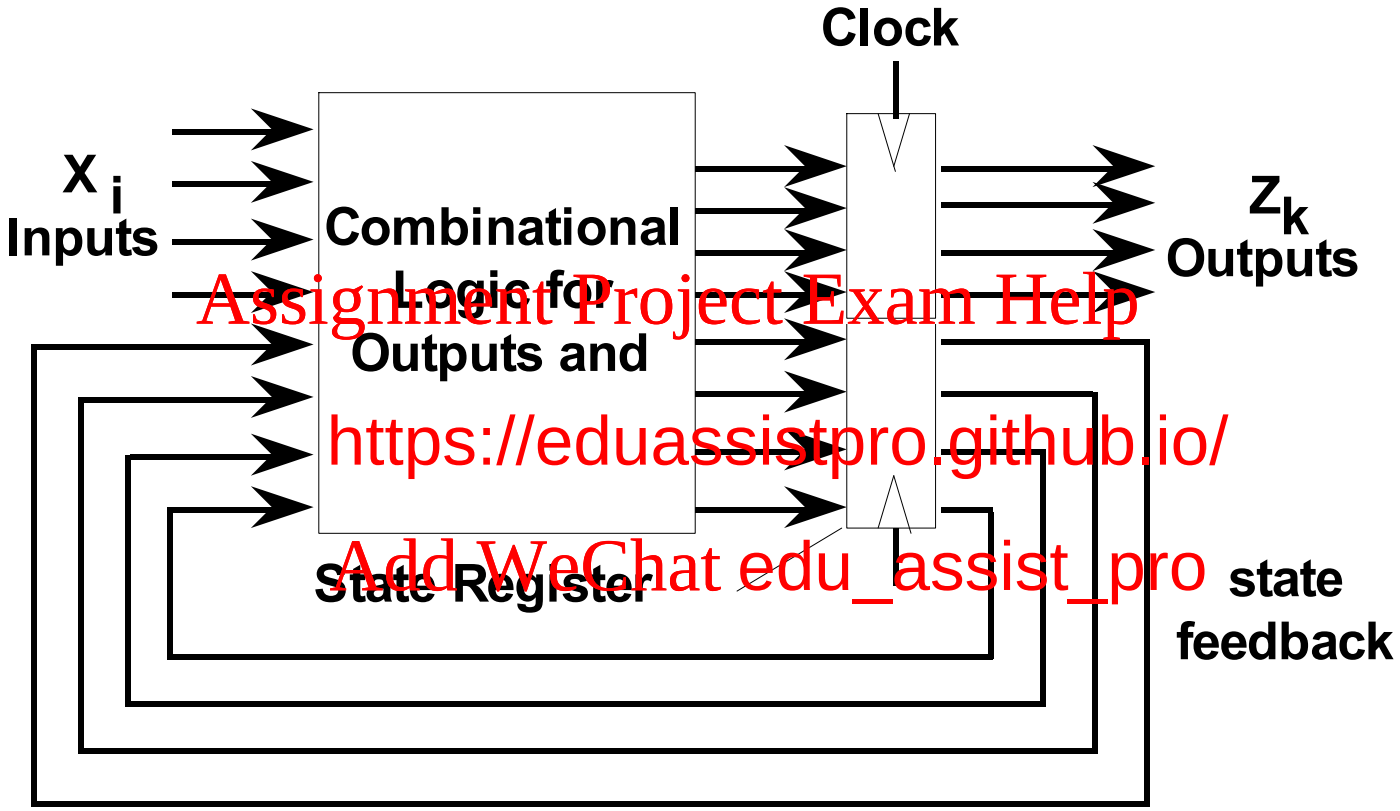
Missing Transitions and Outputs:

ut 0 -> State 00, Output 1
ut 0 -> State 00, Output 0
ut 1 -> State 11, Output 1



## Moore and Mealy Machines

### *Synchronous Mealy Machine*






**latched state AND outputs**

**avoids glitchy outputs!**

## Finite State Machine Word Problems

***Mapping English Language Description to Formal Specifications***

**Four Case Studies:**

- **Finite String Pattern Recognizer**
- **Complex Counter with Decision Making**
- **Traffic Light Controller**
- **Digital Co**



***We will use state diagrams and A***

# Finite State Machine Word Problems

## *Finite String Pattern Recognizer*

**A finite string recognizer has one input (X) and one output (Z). The output is asserted whenever the input sequence …010… has been observed, as long as the sequence 100 has never been seen.**

**Step 1. Understanding the problem statement**



**Sample input/output behavior:**

**X:**
**Z:**

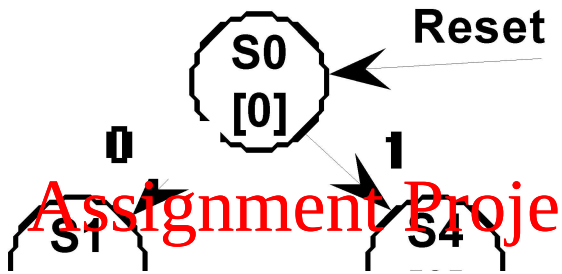


**X: 1001101001 0…**
**Z: 00000001000…**

# Finite State Machine Word Problems

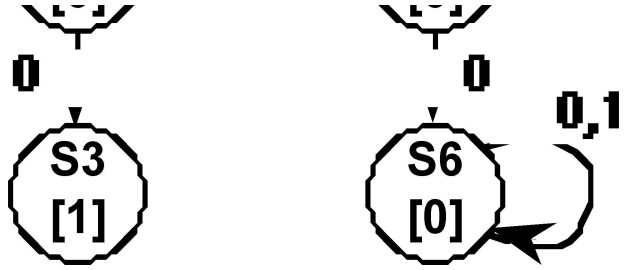
## *Finite String Recognizer*

**Step 2. Draw State Diagrams/ASM Charts for the strings that must be recognized. I.e., 010 and 100.**




**Moore State Diagram**
**Reset signal places FSM in S0**






***Outputs***

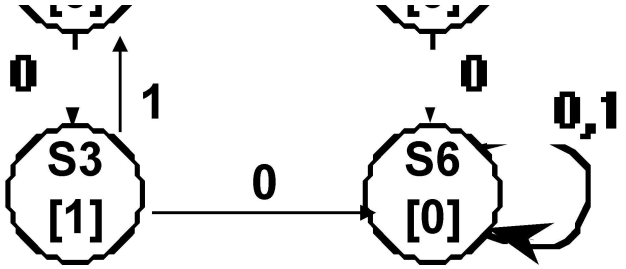
***Loops in State***

## Finite State Machine Word Problems

### *Finite String Recognizer*

**Exit conditions from state S3: have recognized …010**
**if next input is 0 then have …0100!**
**if next input is 1 then have …0101 = …01 (state S2)**





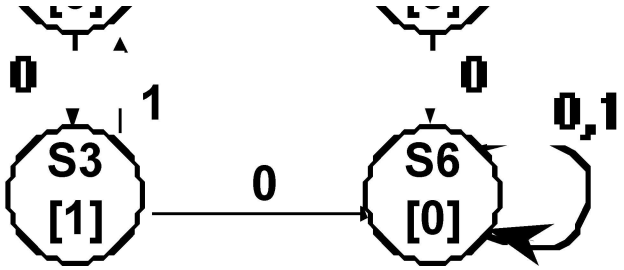
# Finite State Machine Word Problems

## *Finite String Recognizer*

**Exit conditions from S1: recognizes strings of form …0 (no 1 seen)**
**loop back to S1 if input is 0**

**Exit conditions from S4: recognizes strings of form …1 (no 0 seen)**
**loop back to S4 if input is 1**





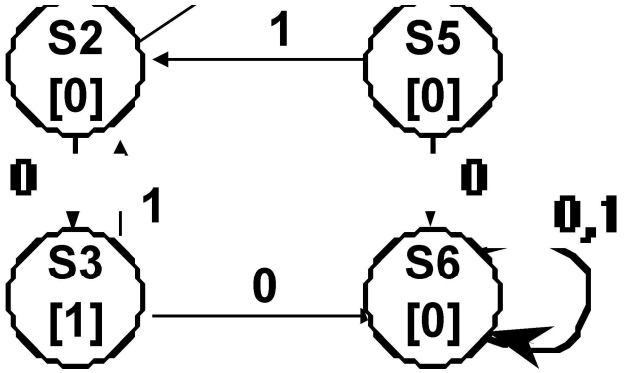
## Finite State Machine Word Problems

### *Finite String Recognizer*

**S2, S5 with incomplete transitions**

**S2 = …01; If next input is 1, then string could be prefix of (01)1(00)**
**S4 handles just this case!**





## Finite State Machine Word Problems

### *Finite String Recognizer*

**Review of Process:**

- **Write down sample inputs and outputs to understand specification**
- **Write down sequences of states and transitions for the sequences to be recognized**
- **Add missin** **much as possible**
- **Verify I/O behavior of your sta** **insure it functions like the specification**

# Finite State Machine Word Problems

## *Complex Counter*

**A sync. 3 bit counter has a mode control M. When M = 0, the counter counts up in the binary sequence. When M = 1, the counter advances through the Gray code sequence.**

**Binary: 000, 001, 010, 011, 100, 101, 110, 111**
**Gray: 000, 001, 011, 010, 110, 111, 101, 100**



***Valid I/O behavior:***



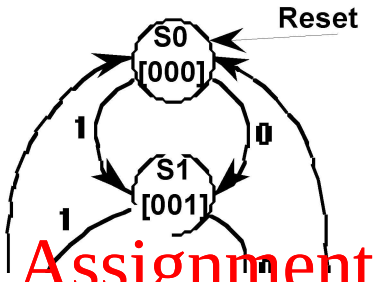


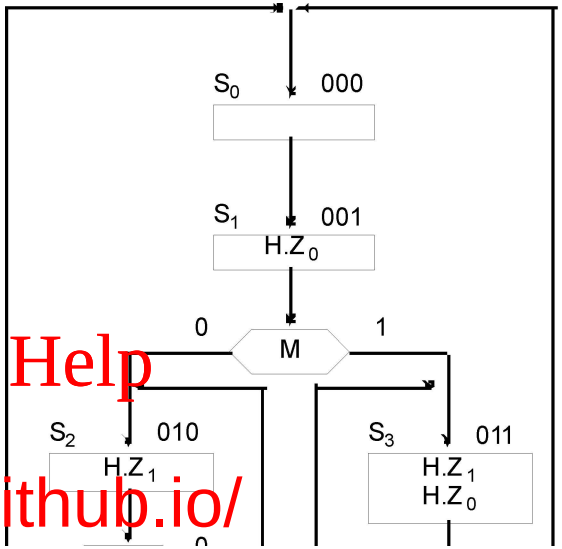
| Mode Input | Present State (Z2 Z1 Z0) | Next State (Z2 Z1 Z0) |
|---|---|---|
| 0 | 001 | 010 |
| 0 | 010 | 011 |
| 1 | 011 | 010 |
| 1 | 110 | 111 |
| 1 | 111 | 101 |
| 0 | 101 | 110 |
| 0 | 110 | 111 |

# Finite State Machine Word Problems

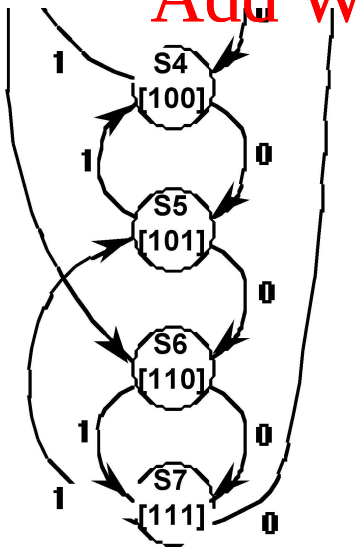
## *Complex Counter*

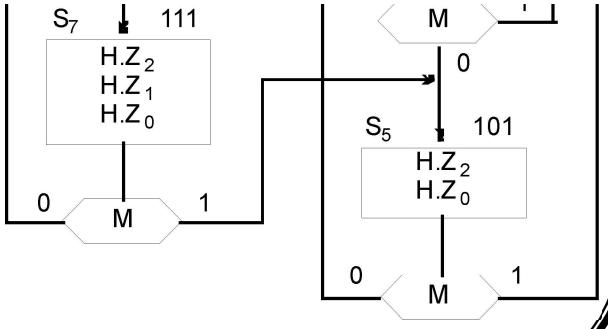
**One state for each output combination**
**Add appropriate arcs for the mode control**





# Finite State Machine Word Problems

## *Traffic Light Controller*

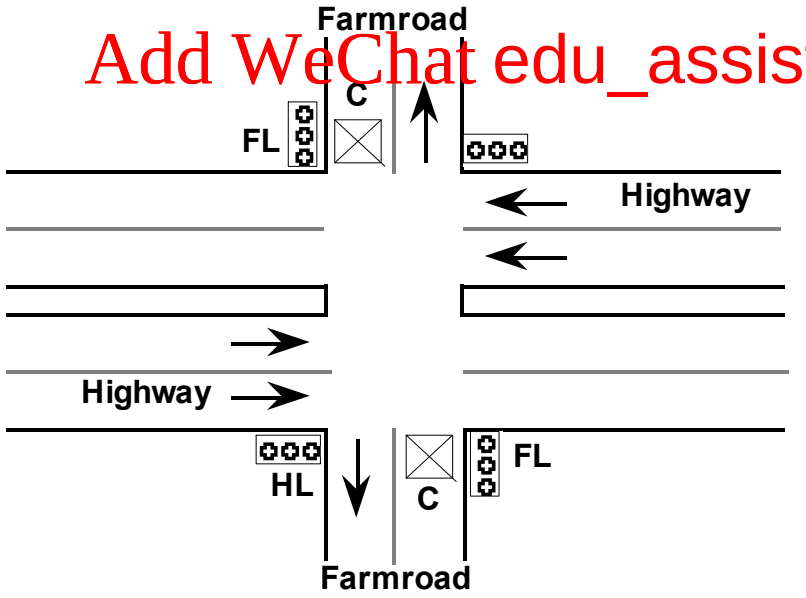
A busy highway is intersected by a little used farmroad. Detectors C sense the presence of cars waiting on the farmroad. With no car on farmroad, lights remain green in highway direction. If vehicle on farmroad, highway lights go from Green to Yellow to Red, allowing the farmroad lights to become green. These stay green only as long as a farmroad car is detected but never longer than a set interval. When these are met, farm lights transition from Green to Yellow to Red, allowing highway to return to green. Even if farmroad vehicles are waiting, highway gets at least a set interval as green.



Assume you have time pulse (TS) and a long ti t (ST) signal. TS is to be used for reen lights.



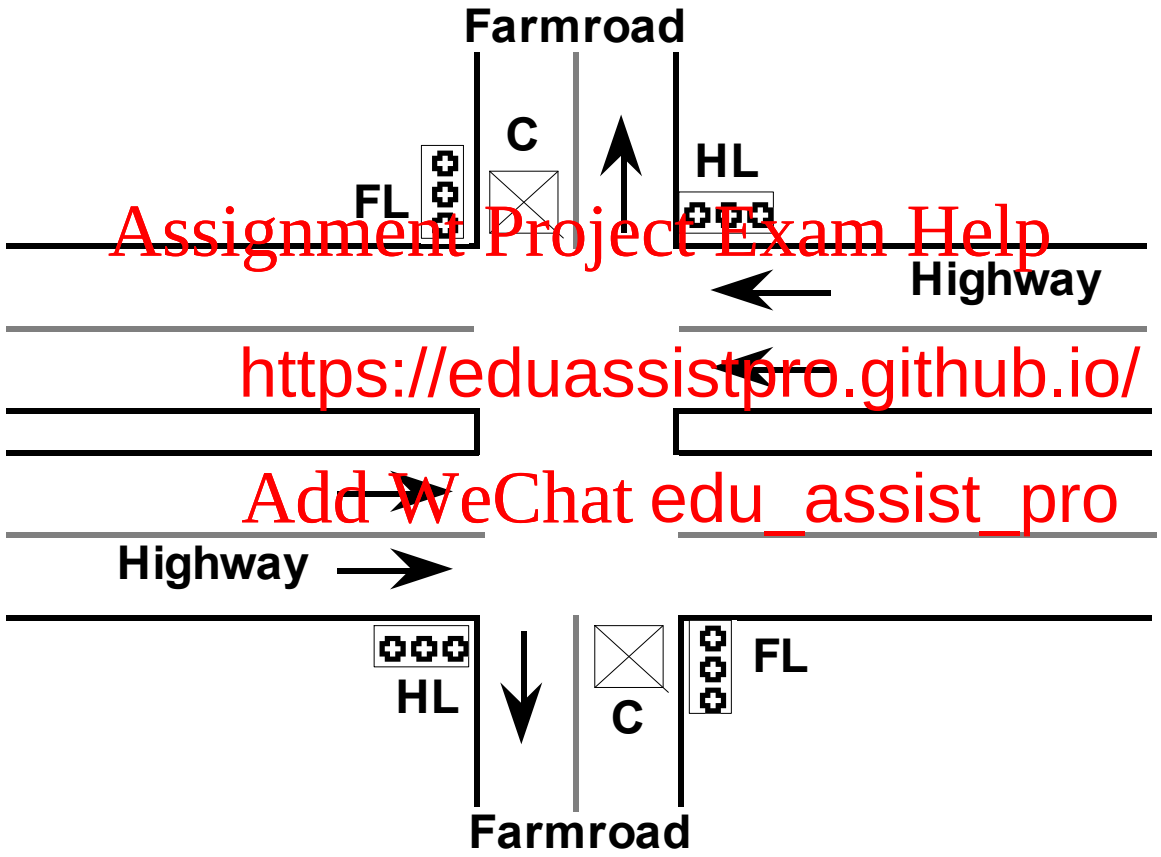




# Finite State Machine Word Problems

## *Traffic Light Controller*

**Picture of Highway/Farmroad Intersection:**



# Finite State Machine Word Problems

## *Traffic Light Controller*

- **Tabulation of Inputs and Outputs:**

| *Input Signal* | *Description* |
|---|---|
| reset | place FSM in initial state |
| C | detect vehicle on farmroad |
| TS | short time interval expired |
| TL | long time interval expired |

| *Output Signal* | *Description* |
|---|---|
| HG, HY, HR | /red highway lights |
| FG, FY, FR | /red farmroad lights |
| ST | or long interval |



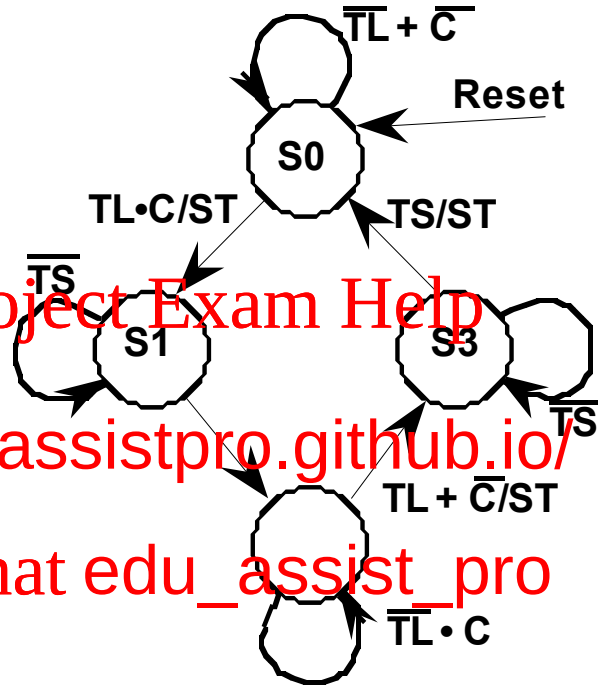
- **Tabulation of Unique States: So** **guration imply others**

| *State* | *Description* |
|---|---|
| S0 | Highway green (farmroad red) |
| S1 | Highway yellow (farmroad red) |
| S2 | Farmroad green (highway red) |
| S3 | Farmroad yellow (highway red) |

# Finite State Machine Word Problems

## *Traffic Light Controller*

**Compare with state diagram:**




**S0: HG**

**S1: HY**

**S2: FG**

**S3: FY**



***Advantages of State Charts:***

- **Concentrates on paths and conditions for exiting a state**
- **Exit conditions built up incrementally, later combined into single Boolean condition for exit**
- **Easier to understand the design as an algorithm**

# Finite State Machine Word Problems

## *Digital Combination Lock*

**"3 bit serial lock controls entry to locked room. Inputs are RESET, ENTER, 2 position switch for bit of key data. Locks generates an UNLOCK signal when key matches internal combination. ERROR light illuminated if key does not match combination. Sequence is: (1) Press RESET, (2) enter key bit, (3) Press ENTER, (4) repeat (2) & (3) two more times."**



***Problem specifi***

- **how do you**



- **exactly when is the ERROR lig**



***Make reasonable assumptions:***

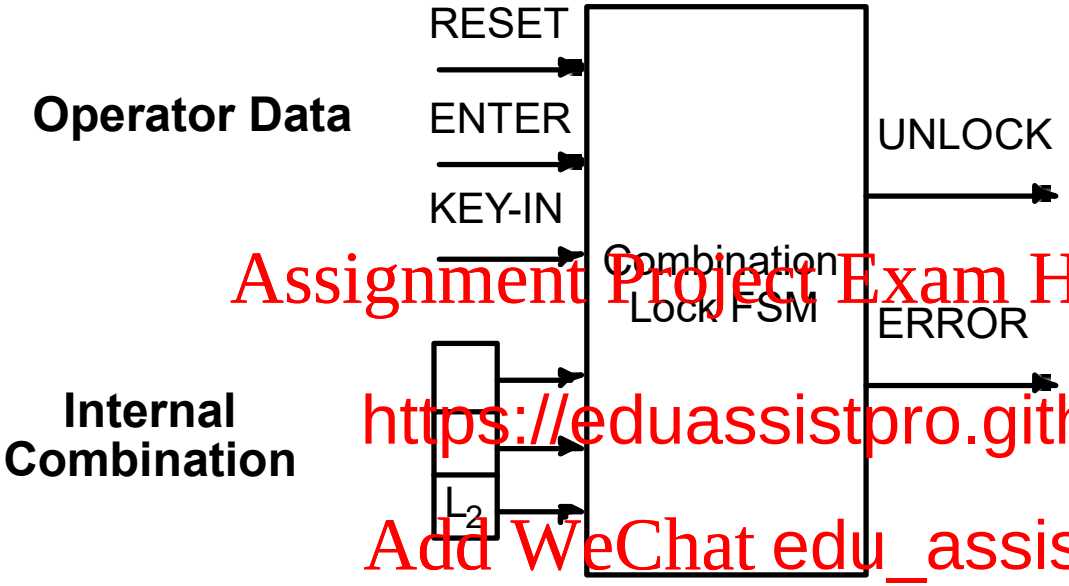
- **hardwired into next state logic vs. stored in internal register**
- **assert as soon as error is detected vs. wait until full combination has been entered**

**Our design: registered combination plus error after full combination**

# Finite State Machine Word Problems

## *Digital Combination Lock*

**Understanding the problem: draw a block diagram …**



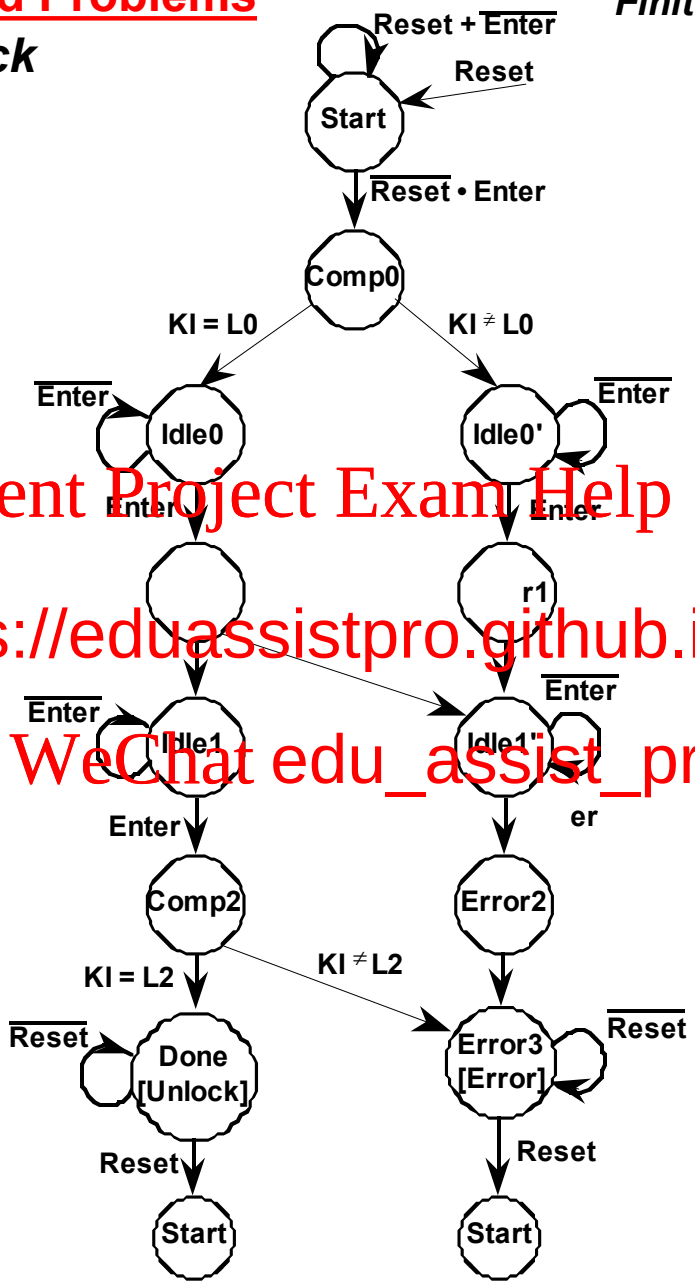



***Inputs:***
**Reset**
**Enter**
**Key-In**
**L0, L1, L2**

***Outputs:***
**Unlock**
**Error**

# Finite State Machine Word Problems

## *Digital Combination Lock*




**Equivalent State**

## Chapter Review

***Basic Timing Behavior an FSM***

- when are inputs sampled, next state/outputs transition and stabilize
- Moore and Mealy (Async and Sync) machine organizations
  outputs = F(state) vs. outputs = F(state, inputs)

***First Two Steps of the Six Step Procedure for FSM Design***



- understanding the problem
- abstract repres



***Abstract Representations of an FSM***

- ASM Charts, Hardware Description Languages

***Word Problems***

- understand I/O behavior; draw diagrams
- enumerate states for the "goal"; expand with error conditions
- reuse states whenever possible