# Chapter 3: Sequential Logic

**Dr. Tim McGuire**
**Sam Houston State University**
*Based on notes by*
*Miles Murdocca*



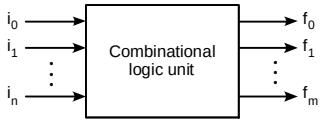
## Some Definitions

- ***Combinational logic*: a digital logic circuit in which logical decisions are made based only on combinations of the inputs (e.g., an adder).**
- ***Sequential logic*: a circuit in which decisions are made based on combinations of the current inputs as well as the past history of inputs (e.g., a memory unit).**
- ***Finite state machine*: a circuit which has an internal state, and whose outputs are functions of both current inputs and its internal state (e.g., a vending machine controller).**



## The Combinational Logic Unit

- **Translates a set of inputs into a set of outputs according to one or more mapping functions.**
- **Inputs and outputs for a CLU normally have two distinct (binary) values: high and low, 1 and 0, 0 and 1, or 5 v and 0 v, for example.**
- **The outputs of a CLU are strictly functions of the inputs, and the outputs are updated immediately after the inputs change. A set of inputs $i_0$–$i_n$ are presented to the CLU, which produces a set of outputs according to mapping functions $f_0$–$f_m$.**



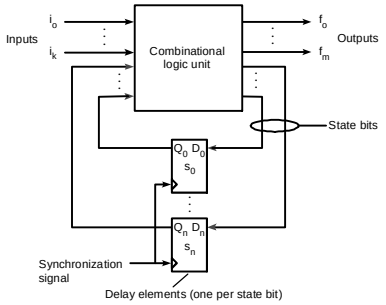



## Sequential Logic

- **The combinational logic circuits we have been studying so far have no memory. The outputs always follow the inputs.**
- **There is a need for circuits with a memory, which behave differently *depending upon their previous state.***
- **An example is the vending machine, which must remember how many and what kinds of coins have been inserted, and which behave according to not only the current coin inserted, but also upon how many and what kind of coins have been deposited previously.**
- **These are referred to as *finite state machines*, because they can have at most a finite number of states.**

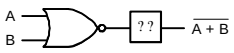


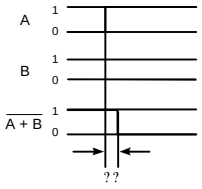
## Classical Model of a Finite State Machine






## A NOR Gate with a Lumped Delay

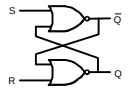


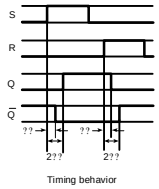

Timing behavior

**This delay between input and output is at the basis of the functioning of an important memory element, the *flip-flop*.**

# An R-S Flip-Flop




| $Q_t$ | $S_t$ | $R_t$ | $Q_{t+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | (disallowed) |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | (disallowed) |

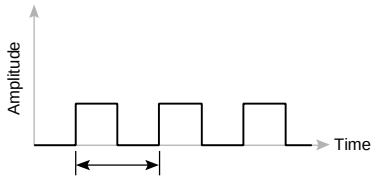



Timing behavior

**The R-S flip-flop is an active-high (positive logic) device.**



# A Clock Waveform

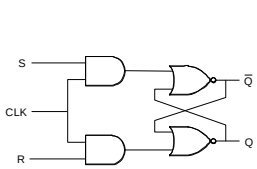


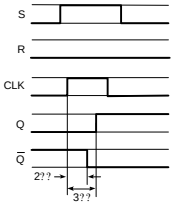

Cycle time = 25 ns

**In a positive logic system, the "action" happens when the clock is high, or positive. The low part of the clock cycle allows propagation between subcircuits, so their inputs are stable at the correct value when the clock next goes high.**



# A Clocked R-S Flip-Flop



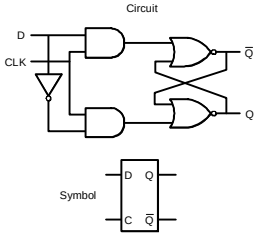



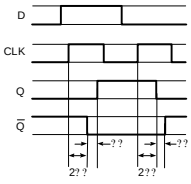

Timing behavior

**The clock signal, CLK, turns on the inputs to the flip-flop.**



# A Clocked D (Data) Flip-Flop



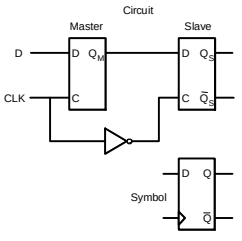



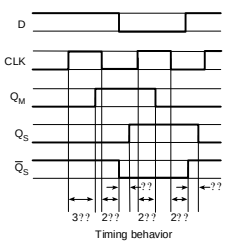

Timing behavior

**The clocked D flip-flop, sometimes called a latch, has a potential problem: If D changes while the clock is high, the output will also change. The Master-Slave flip-flop solves this problem.**



# A Master-Slave Flip-Flop



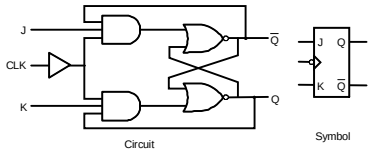




Timing behavior

**The rising edge of the clock clocks new data into the master, while the slave holds previous data. The falling edge clocks the new master data into the slave.**

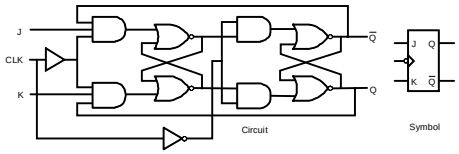


# The Basic J-K Flip-Flop



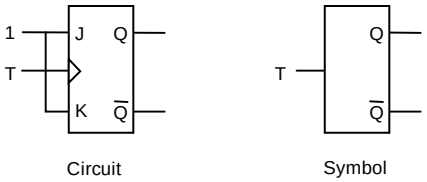

- **The J-L flip-flop eliminates the S = R = 1 problem of the S-R flip-flop, because Q enables J while Q' disables K, and vice versa.**
- **However there is still a problem. If J goes momentarily to 1 and then back to 0 while the flip-flop is active and in the reset, the flip-flop will "catch" the 1.**
- **This is referred to as "1's catching."**
- **The J-K master-slave flip-flop solves this problem.**

# A Master-Slave J-K Flip-Flop





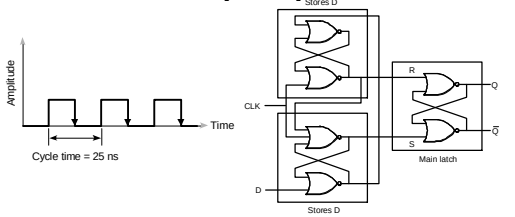

# A T Flip-Flop




- **The presence of a constant 1 at J and K means that the flip-flop will change its state from 0-1 or 1-0 each time it is clocked by the T (toggle) input.**



# Negative Edge-Triggered D Flip-Flop



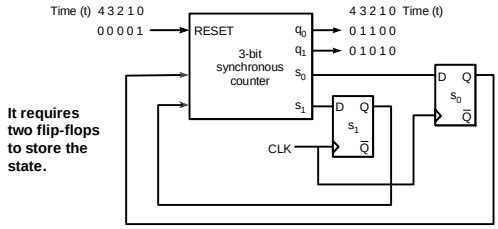

- **When the clock is high, the two input latches output 0, so the main latch remains in its previous state regardless of changes in D.**
- **When the clock goes high-low, values in the two input latches will affect the state of the main latch.**
- **While the clock is low, D cannot affect the main latch.**



# Finite State Machine Design

- **Counter has a clock input, CLK, and a RESET input.**
- **Has two output lines, which must take values of 00, 01, 10, and 11 on subsequent clock cycles.**

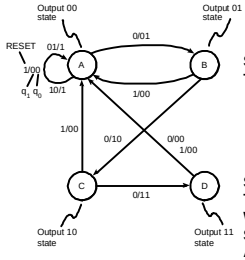
**A Modulo-4 Counter**




**It requires two flip-flops to store the state.**



# State Transition Diagram for a Modulo(4) Counter




**State Table**

| Present State | Next State RESET 0 | 1 |
|---|---|---|
| A | B/01 | A/00 |
| B | C/10 | A/00 |
| C | D/11 | A/00 |
| D | A/00 | A/00 |

**State Table With States Assigned**

| Present State | RESET 0 | 1 |
|---|---|---|
| A:00 | 01 | 00 |
| B:01 | 10 | 00 |
| C:10 | 11 | 00 |
| D:11 | 00 | 00 |

- **The state diagram and state table tell "all there is to know" about the FSM, and are the basis for a provably correct design.**



# Truth Table

| r(t) | $s_1(t)s_0(t)$ | $s_1s_0(t+1)$ | $q_1q_0(t+1)$ |
|---|---|---|---|
| 0 | 00 | 01 | 01 |
| 0 | 01 | 10 | 10 |
| 0 | 10 | 11 | 11 |
| 0 | 11 | 00 | 00 |
| 1 | 00 | 00 | 00 |
| 1 | 01 | 00 | 00 |
| 1 | 10 | 00 | 00 |
| 1 | 11 | 00 | 00 |

- **Develop equations from this truth table for $s_0(t+1)$, $s_1(t+1)$, $q_0(t+1)$, and $q_1(t+1)$ from inputs r(t), $s_0(t)$ and $s_1(t)$**

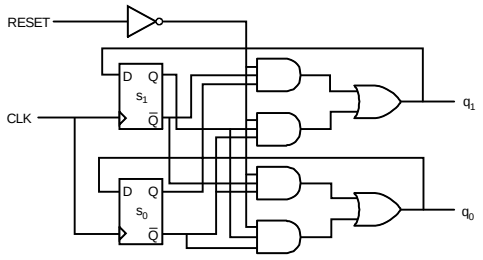
## Equations

$$s_0(t+1) = \overline{r(t)}s_1(t)\overline{s_0(t)} + r(t)s_1(t)\overline{s_0(t)}$$
$$s_1(t+1) = \overline{r(t)}s_1(t)s_0(t) + r(t)s_1(t)\overline{s_0(t)}$$
$$q_0(t+1) = \overline{r(t)}s_1(t)\overline{s_0(t)} + r(t)s_1(t)\overline{s_0(t)}$$
$$q_1(t+1) = \overline{r(t)}s_1(t)s_0(t) + r(t)s_1(t)\overline{s_0(t)}$$

**Implement these equations**



## Logic Design for a Modulo(4) Counter




**There are many simpler techniques for implementing counters.**



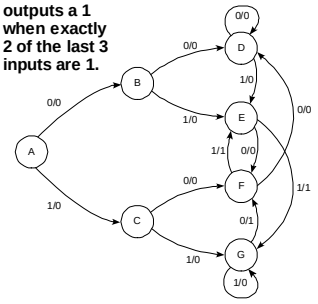
## Example: A Sequence Detector

- **Design a machine that outputs a 1 when exactly 2 of the last 3 inputs are 1.**
- **e.g. input sequence of 011011100 produces an output sequence of 001111010**
- **Assume input is a 1-bit serial line.**
- **Use D flip-flops and 8-1 multiplexers.**
- **Begin by constructing a state transition diagram.**



## State Transition Diagram for Sequence Detector

- **Design a machine that outputs a 1 when exactly 2 of the last 3 inputs are 1.**

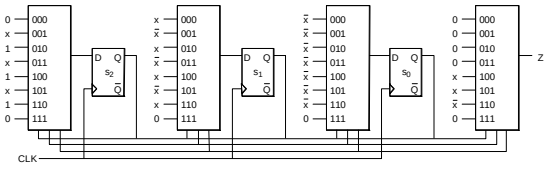



| Pres. State | X = 0 | X = 1 |
|---|---|---|
| $S_2S_1S_0$ | $S_2S_1S_0Z$ | $S_2S_1S_0Z$ |
| A=000 | 001/0 | 010/0 |
| B=001 | 011/0 | 100/0 |
| C=010 | 101/0 | 110/0 |
| D=011 | 011/0 | 100/0 |
| E=100 | 101/0 | 110/1 |
| F=101 | 011/0 | 100/1 |
| G=110 | 101/1 | 110/0 |

- **Discuss: the "meaning" of each state.**
- **Convert table to truth table (how?).**
- **Solve for $s_2$ $s_1$ $s_0$ and Z.**



## Logic Diagram for Sequence Detector

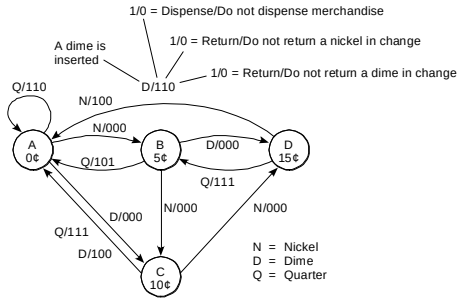





## Example: A Vending Machine Controller

- **Acepts nickel, dime, and quarter. When value of money inserted equals or exceeds twenty cents, machine vends item and returns change if any, and waits for next transaction.**
- **Implement with PLA and D flip-flops.**

# State Transition Diagram for Vending Machine Controller



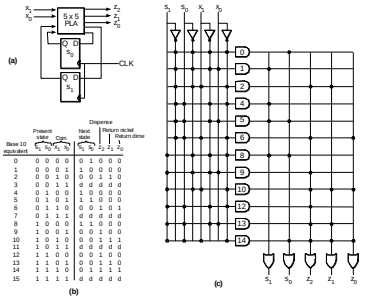



# Truth Table for Vending Machine Controller

| Base 10 equivalent | Present state $S_1$ | $S_0$ | Coin $X_1$ | $X_0$ | Next state $S_1$ | $S_0$ | Dispense $z_2$ | Return nickel $z_1$ | Return dime $z_0$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | d | d | d | d | d |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | d | d | d | d | d |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 | d | d | d | d | d |
| 12 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | d | d | d | d | d |



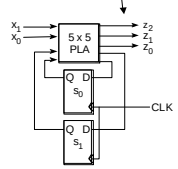
# (a)FSM Circuit, (b)Truth Table, and (c)PLA Realization for Vending Machine Controller



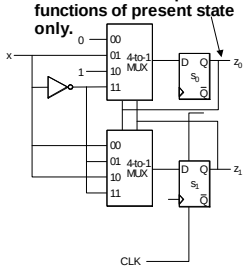



# Mealy versus Moore Machines

- **Mealy model: Outputs are functions of inputs and present state.**
- **Previous FSM designs were Mealy machines, because next state was computed from present state and inputs.**




- **Moore model: Outputs are functions of present state only.**




- **Both are equally powerful.**

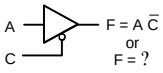


# Tri-State Buffers

| C | A | F |
|---|---|---|
| 0 | 0 | ? |
| 0 | 1 | ? |
| 1 | 0 | 0 |
| 1 | 1 | 1 |


**Tri-state buffer**

| C | A | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | ? |
| 1 | 1 | ? |



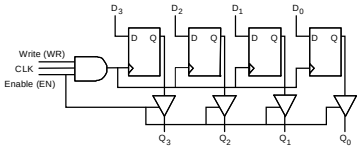

**Tri-state buffer, inverted control**

- **There is a third state: high impedance. This means the gate output is essentially disconnected from the circuit.**
- **This state is indicated by ? in the figure.**



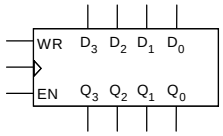
## A 4-Bit Register

**Gate-Level View**

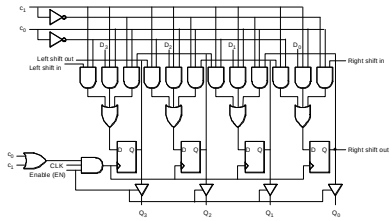



## Abstract Representation of a 4-Bit Register

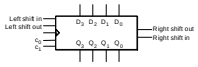
**Chip-Level View**



## Internal Layout and Block Diagram for Left-Right Shift with Parallel Read/Write Capabilities



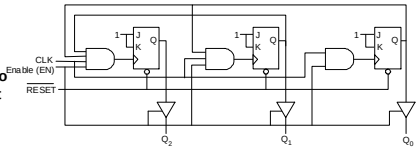

| Control | | Function |
|---|---|---|
| $c_1$ | $c_0$ | |
| 0 | 0 | No change |
| 0 | 1 | Shift left |
| 1 | 0 | Shift right |
| 1 | 1 | Parallel load |

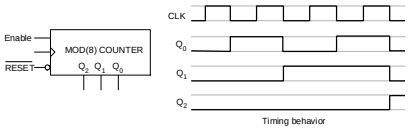





## A Modulo(8) Ripple Counter

**Note the use of the T flip-flops. They are used to toggle the input of the next flip-flop when its output is 1.**

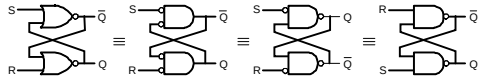



Timing behavior



## Converting a NOR S-R to an NAND S-R



Active-high NOR Implementation

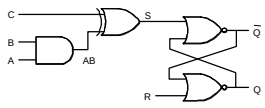
Push bubbles (DeMorgan's)

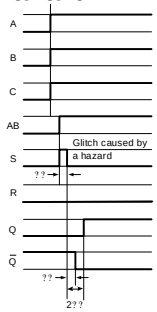
Rearrange bubbles

Convert from bubbles to active-low signal names



## A Circuit with a Hazard




Timing behavior

**It is desirable to be able to "turn off" the flip-flop so it does not respond to such hazards.**