***NOTES2 ECE 2500 Digital Logic***
*To the Second & Final Exam*
***Prof. Dean R. Johnson***

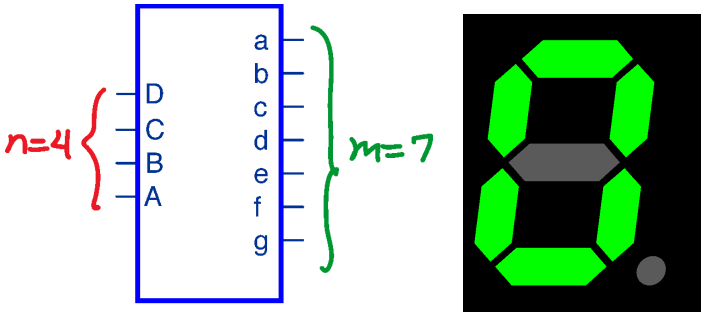
**Lecture Topics:**



## *DECODERS, MUXES, ENCODERS and DEMUXES*

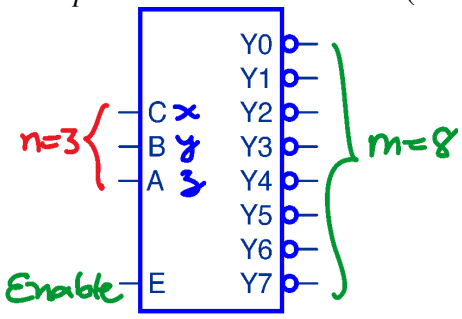
### Decoders

- Decoders translate an ***n*-bit input codeword** into a larger ***m*-bit output codeword**, where $m \leq 2^n$
  - *Example:* 7 Segment Decoder ($n = 4$, $m = 7$)



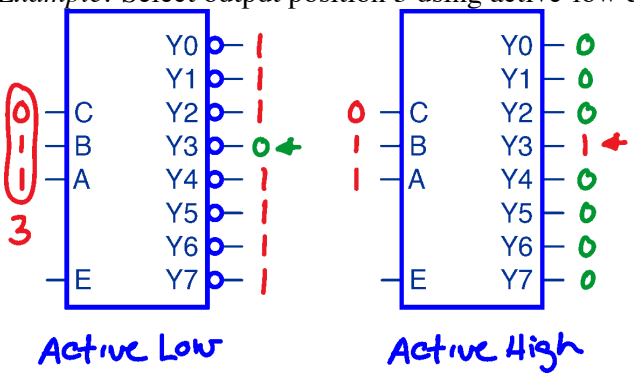



- *Binary* decoders ($m = 2^n$), are designed to *select an output position*
  - *Example:* 74138 3-to-8 Decoder ($n = 3$, $m = 8$)

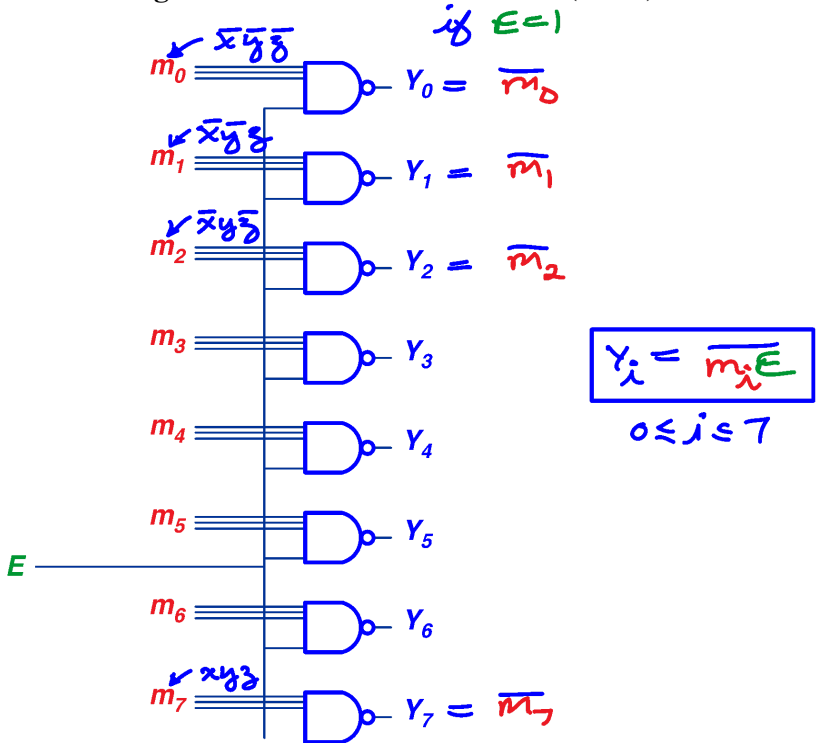



  - ***n*** select lines (i.e. ***C, B, A***) determine the *active* (selected) output $Y_i$, $0 \leq i \leq 2^n - 1$
  - The remaining ***m* -** 1 outputs are *inactive*

- *Example:* Select output position 3 using active-low & active high 3-to-8 decoders



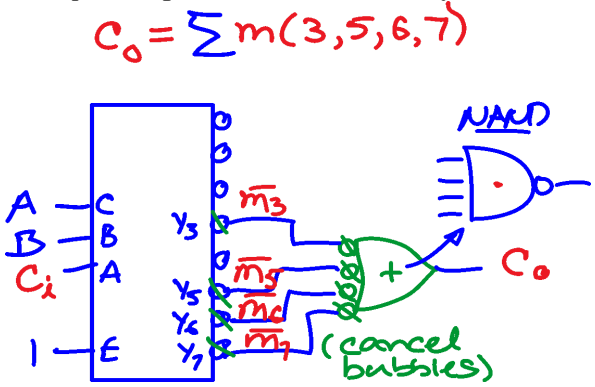

- Decoder gate architecture consists of
  - An array of ***m*** **ANDing elements** forming minterms of select variables
    - **AND** gates produce *active-high* outputs
    - **NAND gates** produce *active-low* outputs
  - An **enable signal** ***E*** must be in the active state (***E* = 1**) to enable the decoder




  - Active-low decoder output equations:
    $Y_i = (m_i E)'$, $0 \le i \le 2^n - 1$

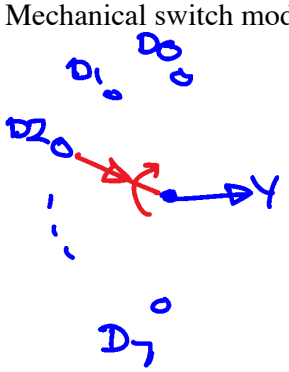
- Binary decoders may also be employed as **SOP Boolean function generators** by
  - Attaching an **ORing (summing) element** to decoder minterm outputs
    - **OR** gate summer for active-high decoders
    - **NAND** gate summer for active-low decoders

- *Example:* Implement full adder carry-out

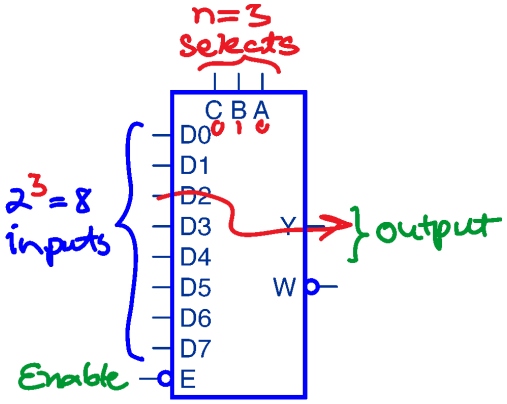



**Multiplexers (MUXes)**

- **MUXs** connect one of $2^n$ **data input lines** to one **output line**
  - Mechanical switch model:

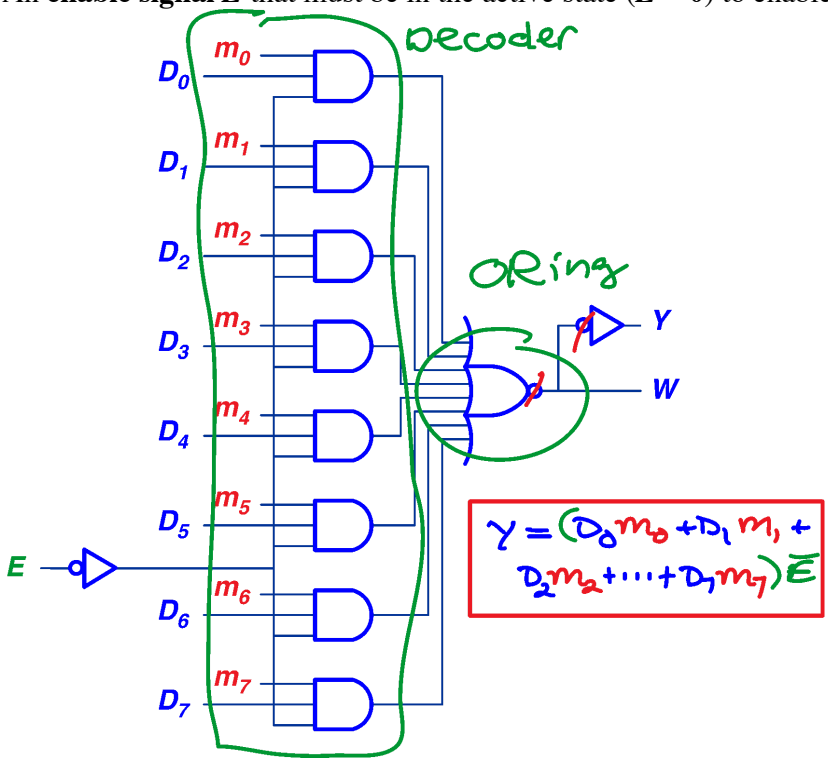



  - A 74151 8-to-1 **MUX** with active-low enable

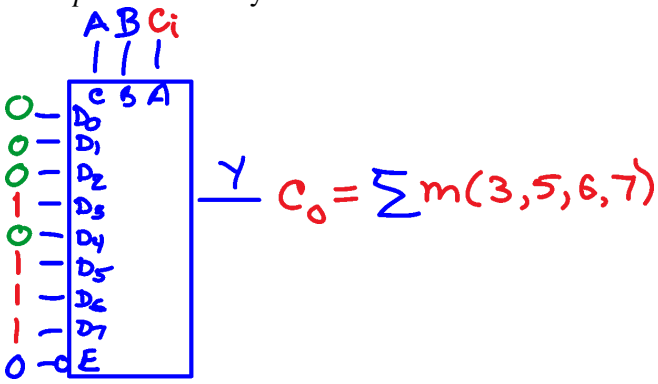



- The data line selected for connection is determined by a set of ***n*** input select lines, i.e. ***C, B, A***
- **MUX** gate architecture consists of
  - A **decoder** plus
  - An **ORing element**

- An **enable signal** ***E*** that must be in the active state (***E* = 0**) to enable the **MUX**




  - Data lines $D_i$ are **AND**ed with decoder minterms to provide selection of desired minterms.
  - **MUX** output equation (for an active-low ***E***) is:
    $Y_i = (D_0m_0 + D_1m_1 + D_2m_2 + \ldots + D_7m_7)E'$
- **MUX**s may also be employed as a **SOP Boolean function generator** of the input select lines
  - *Example:* More carry-out



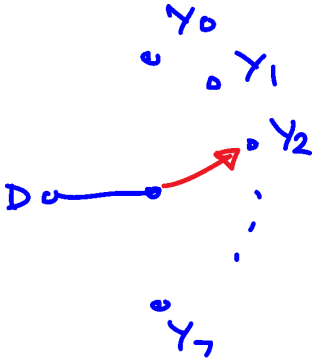

**Encoders**

- Translates an ***n* bit dataword** into a smaller **m-bit codeword**, where $n \leq 2^m$.
- Architecture consists of a linear array of ***m* ORing elements**. (Not shown)

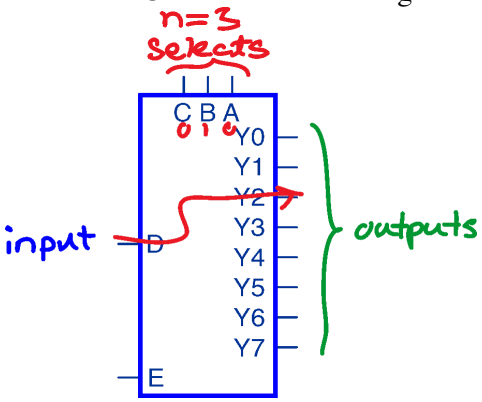
**Demultiplexers (DeMUXes)**

- DeMUXes connect one **input data line** to one of $2^n$ **output lines**

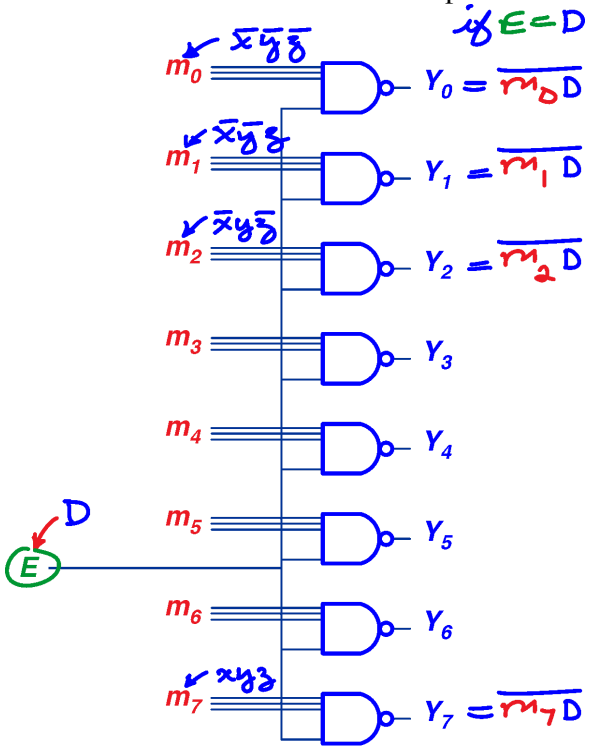
- Mechanical switch model:




- A 1-to-8 MUX with an active-high enable




- The output line selected for connection is determined by a set of ***n*** output select lines, i.e. ***C, B, A***.
- A **DeMUX** gate architiecture can be realized from a decoder circuit by utilizing enable ***E*** as the **DeMUX** data input line ***D***
  - A 74138 3-to-8 decoder with data input ***D*** connected to ***E***:



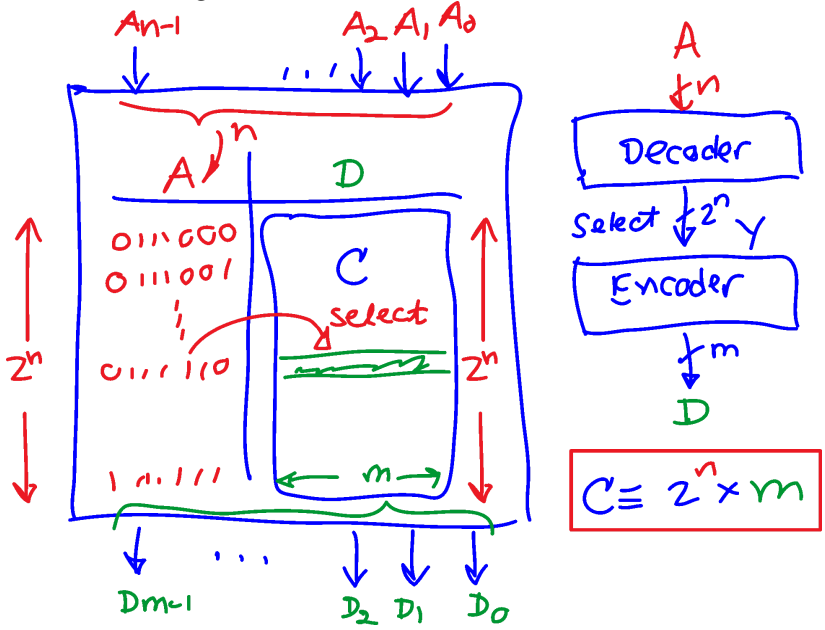

  - The **DeMUX** output equation is: $Y_i = (m_i D)'$, $0 \le i \le 7$

**Quiz #6 & selected solutions**

***READ-ONLY MEMORIES***

**ROM Basics**

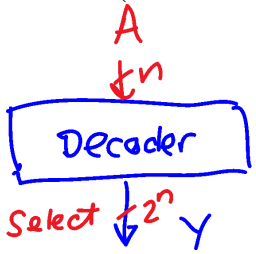
- A **ROM** is a programmable truth table consisting of a
  - ***n***-bit address input (selects a word of memory)
  - ***m***-bit data output (the data bits that fit into one word)
- ROM block diagram:



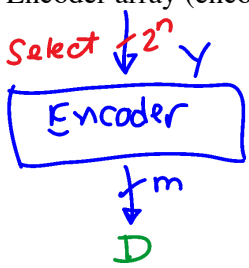

- ROM capacity is: $C = 2^n \times m$ (bit contents of the ROM)

**ROM Implementations**

- Decoder (selects a word)

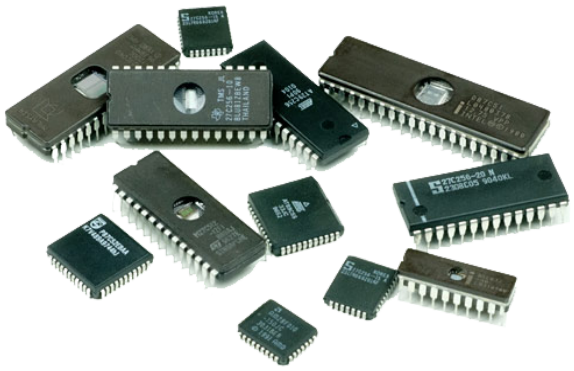



- Encoder array (encodes and stores the word)




- Historically 5 kinds of ROM implementations
  - ROM (mask)
  - PROM (programmable ROM):
  - EPROM (erasable-programmable ROM)
  - EEPROM (electrically-erasable PROM)

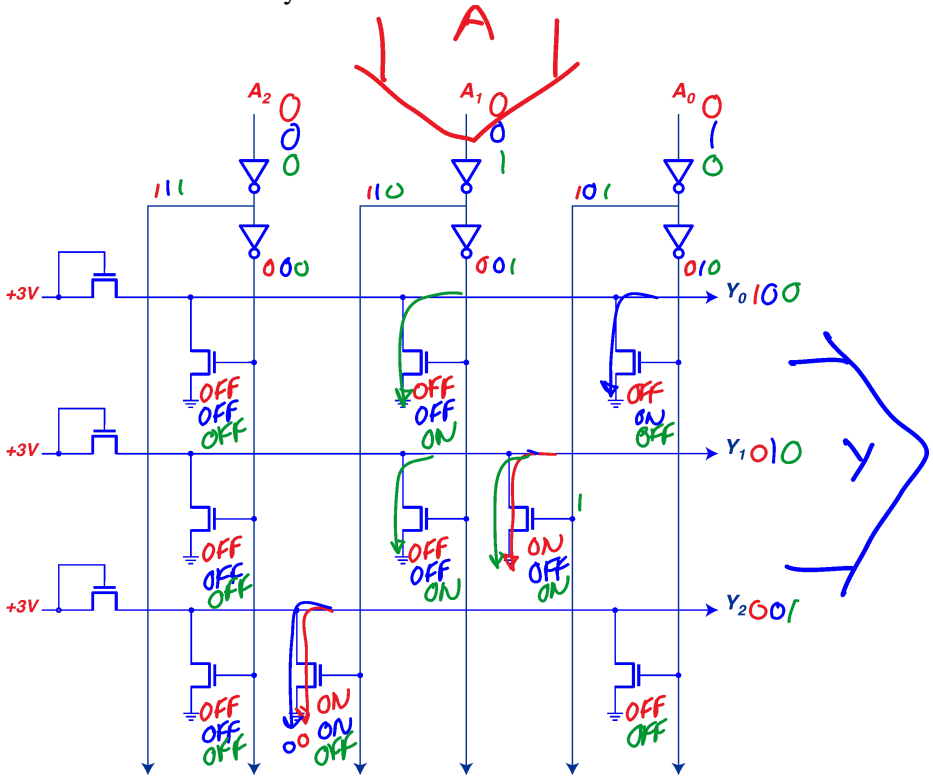
- Flash (fast block-based EEPROM)



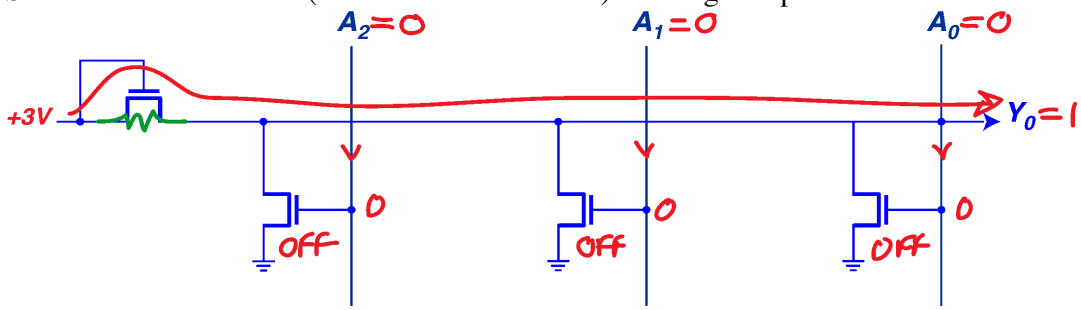
- Encoding Techniques
  - Diode-fuse array
    computer.howstuffworks.com/rom3.htm
  - MOS transistor-fuse array
  - Double gate MOS transistor array

**ROM Transistor Circuitry**
Three examples:

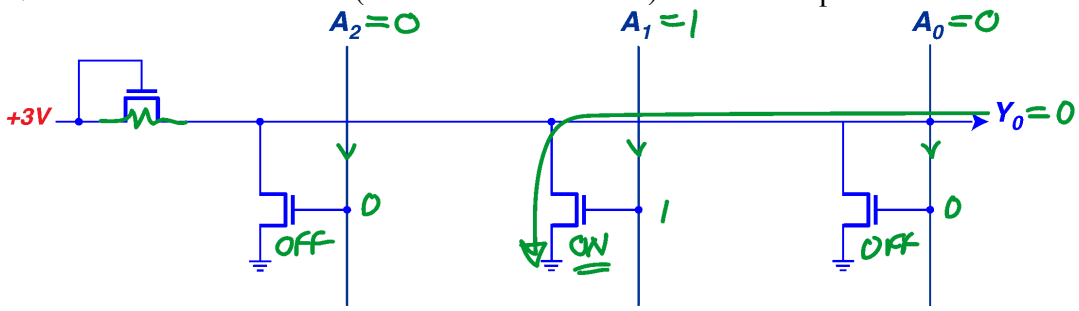
- **PROM**
  - Decoder transistor array:



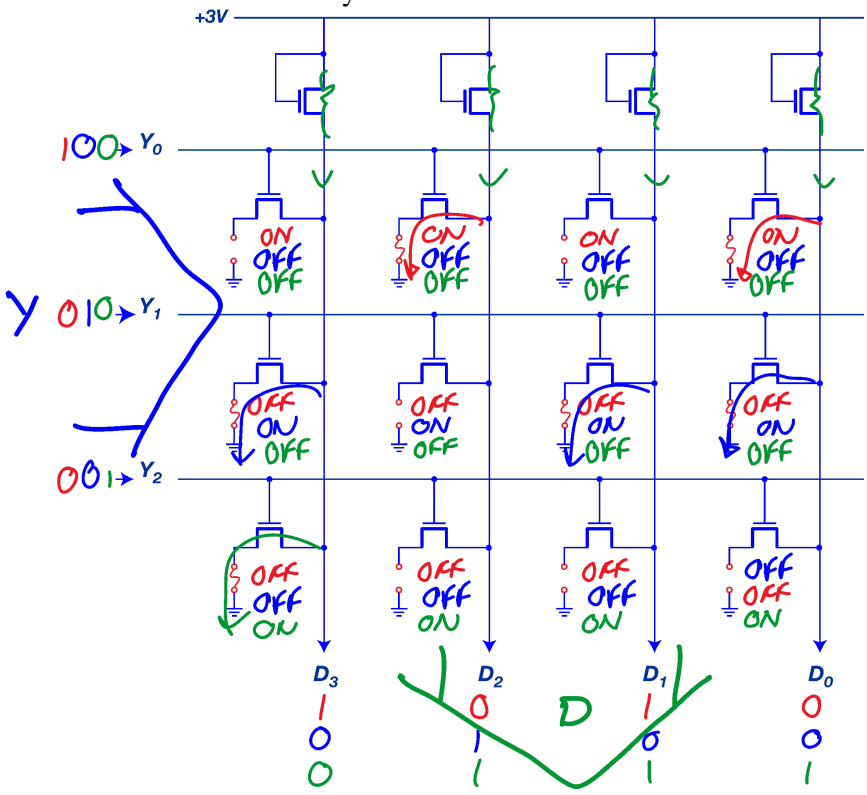

  - Selected row of decoder (row 0 with address 000) with high output:



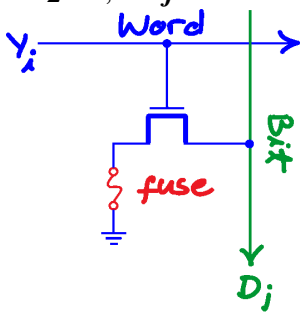
- Nonselected row of decoder (row 0 with address 010) with low output:



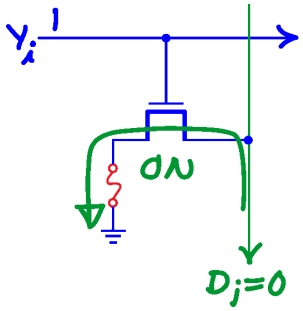

- Encoder transistor-fuse array:



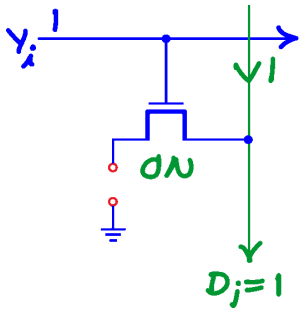

- **PROM** encoder cell, showing $i^{th}$ word & $j^{th}$ bit lines plus fuse:
  $\mathbf{0} \le i \le 2^n\text{-}1, \mathbf{0} \le j \le m\text{-}1$



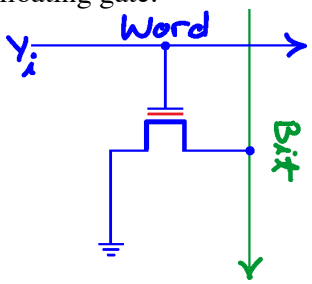

  - An intact fuse connects a **0**:



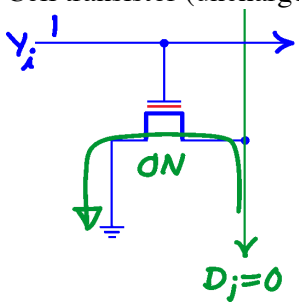
- A blown fuse connects a **1**:



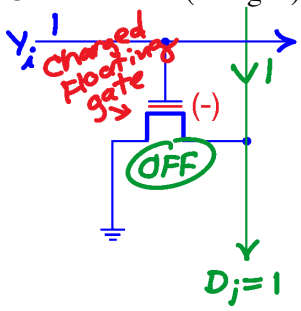

- **EPROM**
  - Decoder transistor array is used, as before.
  - Encoder transistors employ a floating gate to connect outputs:
    - **EPROM** encoder cell, showing $i^{th}$ word and $j^{th}$ bit lines plus a transistor having a chargeable red floating gate:



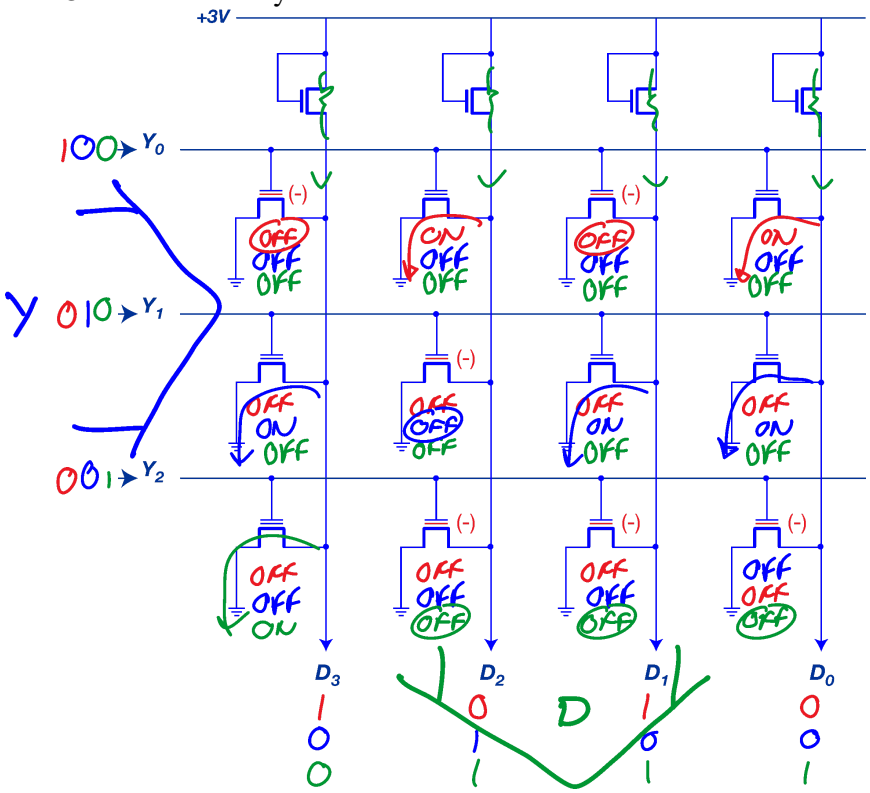

    - Cell transistor (uncharged) turns ON, connects a **0**




    - Cell transistor (charged) stays **OFF,** connects a **1**



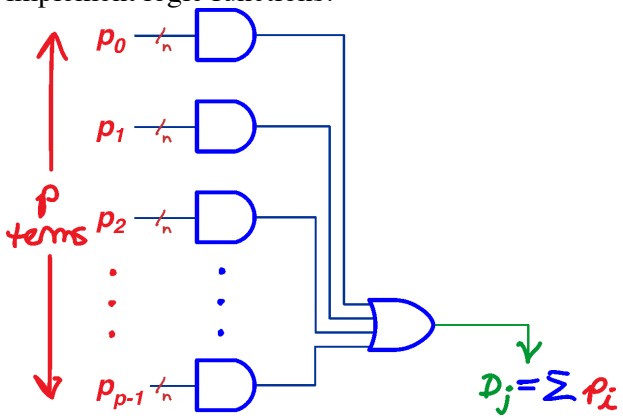
- EPROM Encoder array:




- **Flash** memories: [from wikipedia.com]
  - Double-gate implementation on transistors
  - NOR flash (like EPROM, can access individual bytes)
  - NAND flash (physically smaller, but can access only blocks of bytes). Current technology used by portable music players.

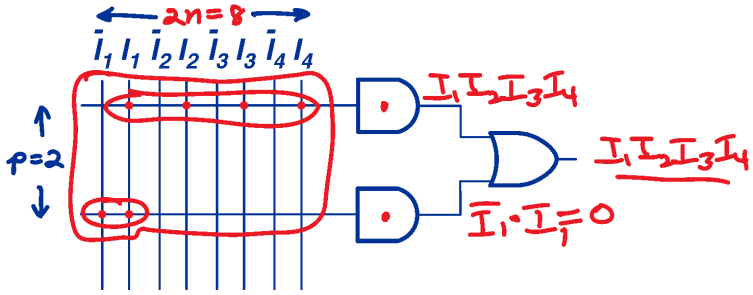
**Programmable Gate Arrays (PGAs)**

- A **PGA** is a programmable logic device that sums ***p*** configurable product terms (not just minterms) to quickly implement logic functions.

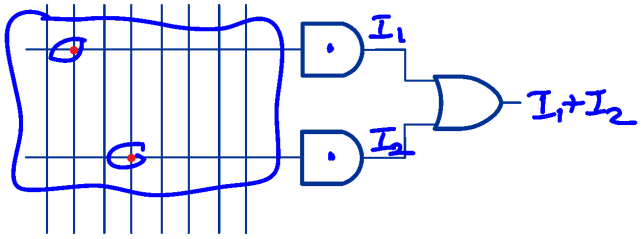



- Traditional **PGA**s are built using **AND-arrays**
  - Each **PGA** will sum ***p*** products.
  - Each *p*-term is formed from an **AND-array** having ***2*n*** input lines
  - An **AND-array** has size ***p x 2*n.***
  - A **PGA** has ***m* AND arrays** which implement ***m*** logic functions.
- A **PLD** is a P-logic device having one or more **PGA** implementations
- *Example:*A **PLD** (***p*** = 2, ***m*** = 5) implementing 5 simple functions

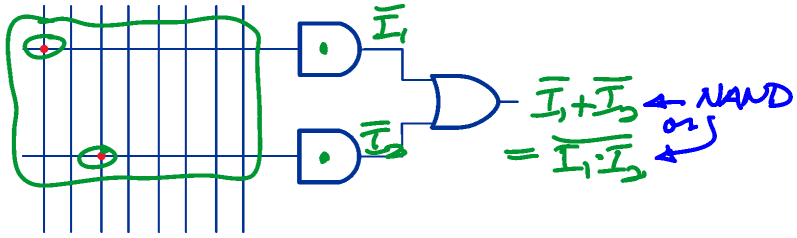
- **AND** function:



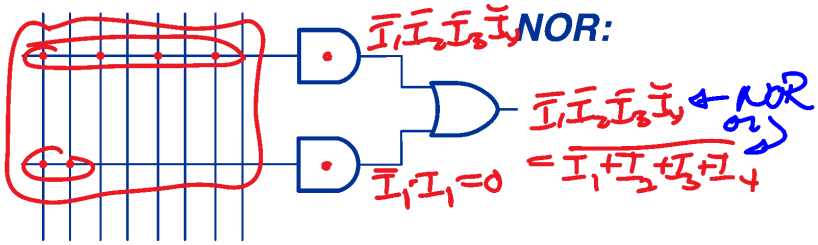

- **OR** function:



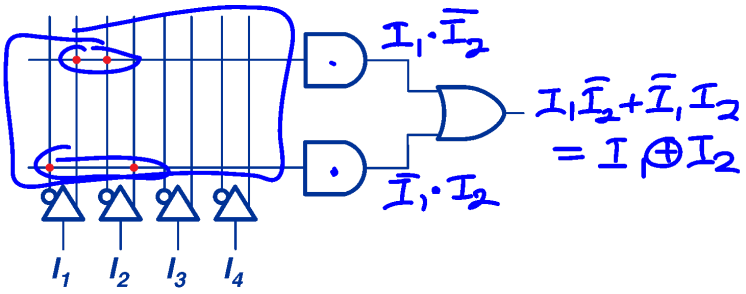

- **NAND** function:




- **NOR** function:




- **XOR** function:




- **Dots** (or **x**'s) show connections.
  - No **dot** (or **x**) implies no connection.
  - Dots are placed during the process of downloading a data bit stream
- The ***Artix-7*** FPGA (field PGA) used in the lab has a look-up table (LUT) organization
  - LUT has 6 inputs and 1 output
  - Constructed from a matrix of 5 by 4 = 20 configurable logic blocks (CLB). The CLB has 2 slices, each slice having
    - 24 inputs (***n = 24***)
    - 4 LUTs (***m = 4***) outputs

- 20 CLBs thus support 960 inputs and 160 outputs!

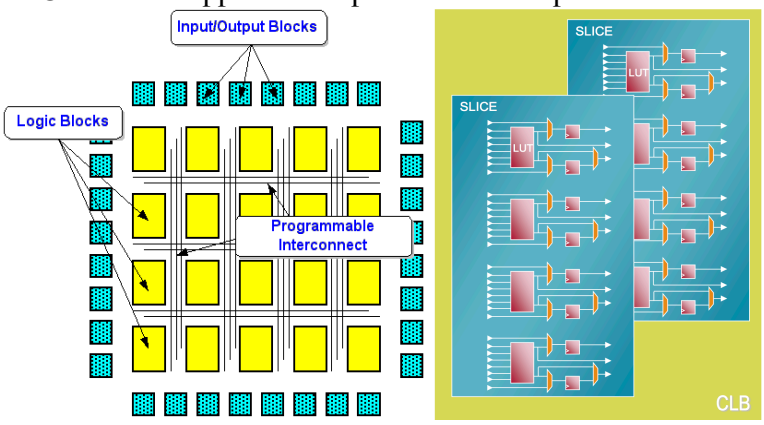


Image: Xilinx

- Advantages of **FPGA**s:
  - Can handle error sources such as latch ups in the logic by
    - Triple mode redundancy (TMR)
    - Epitaxial radiation hardening
    - Scrubbing (reprogramming)

**Quiz #7 & selected solutions**

## *RANDOM ACCESS MEMORIES*

### RAM Basics

- A **RAM** is a rewritable truth table that can
  - Read data (read a word from the table)
  - Write data (write a word to the table)
- ***n***-bit address input (selects a word)
- ***m***-bit data input/output determines *word length*:
  - ***m*** = 8 bits (1 byte)
  - ***m*** = 16 bits (2 bytes)
  - ***m*** = 32 bits (4 bytes)
  - ***m*** = 64 bits (8 bytes)
  - ***m*** = 128 bits (16 bytes)
- Binary data orders of magnitude: http://en.wikipedia.org/wiki/Orders_of_magnitude_(data)
- XBox Series X vs Playstation 5: Both use AMD Zen 2 CPU**,** and have a register size of ***m*** = 64 b and https://www.tomsguide.com/news/ps5-vs-xbox-series-x
- Capacity
  - Formulas:
    - $C = 2^n \times m$ **b** (in bits)
    - $C = 2^n$ **B** (in Bytes if ***m*** = 8)
    - $C = 2^n \times m/8$ **B** (in Bytes in general)
  - Prefixes:
    - $2^{10} = K$ (*Kilo*) [Alt acronym: *Kibi*]
    - $2^{20} = K*K = M$ (*Mega*) [Alt acronym: *Mebi*]
    - $2^{30} = K*M = G$ (*Giga*) [Alt acronym: *Gibi*]
    - $2^{40} = K*G = T$ (*Tera*) [Alt acronym: *Tibi*]
  - Sizes:
    - $C = (2^n \times m/8)/2^{10} \rightarrow C$ ***K*B** (in *Kilo* Bytes)
    - $C = (2^n \times m/8)/2^{20} \rightarrow C$ ***M*B** (in *Mega* Bytes)
    - $C = (2^n \times m/8)/2^{30} \rightarrow C$ ***G*B** (in *Giga* Bytes)
    - $C = (2^n \times m/8)/2^{40} \rightarrow C$ ***T*B** (in *Tera* Bytes)

- *Examples:*
  - $n = 14, m = 8$

$$C = 2^n \times m = 2^{14} \times 8 = 2^{17} \text{ bits}$$

$$\text{or } C = 2^n = 2^{14} \text{ Bytes} = \frac{2^{14}}{2^{10}} \text{ KB}$$

$$= 2^4 \text{ KB}$$

$$= 16 \text{ KB}$$

  - $n = 26, m = 8$

$$C = 2^n B$$

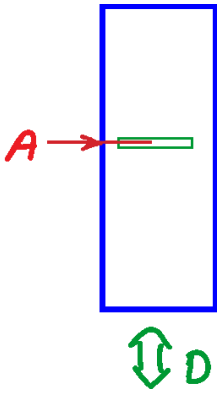
$$= 2^{26} B = \frac{2^{26}}{2^{20}} \text{ MB}$$

$$= 2^6 \text{ MB}$$

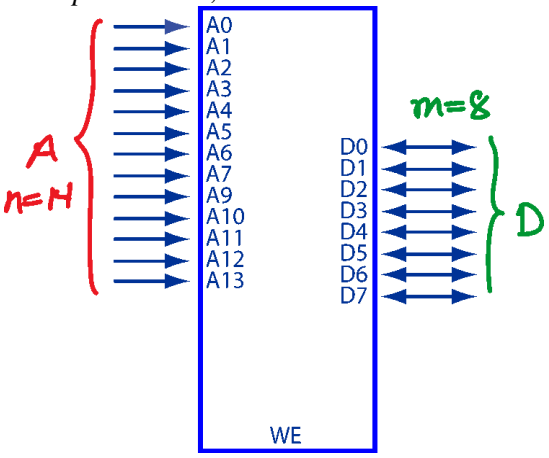
$$= 64 \text{ MB}$$

**RAM Organization and Type**

- Addressing (selecting a word)
  - **1D decoding** (Straight decoding)
    - $n$-bit physical address.
    - Address size = $n$
    - Row decoder selects $m$-bit word

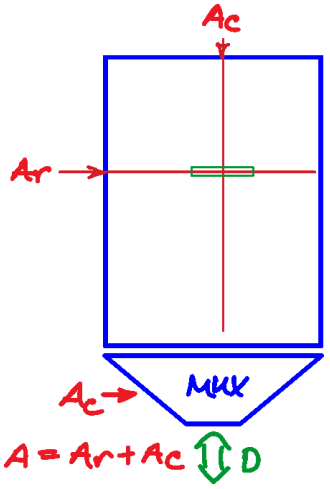



    - *Example:* $n = 14, m = 8$: $C = 16$ KB

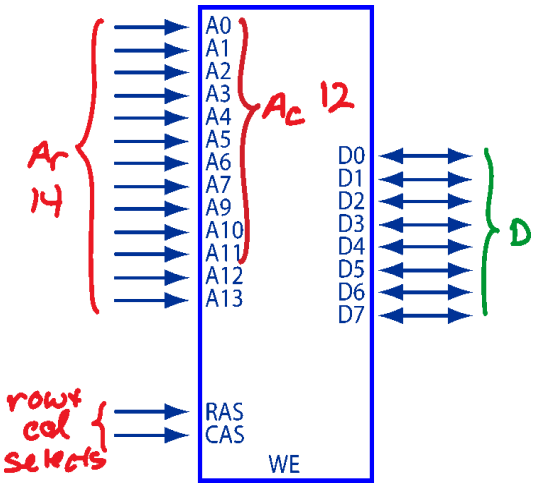



  - **2D decoding** (row decoders and column **MUX**)
    - $n_r$-bit row decoding
    - Reuse $n_c \le n_r$ for column decoding

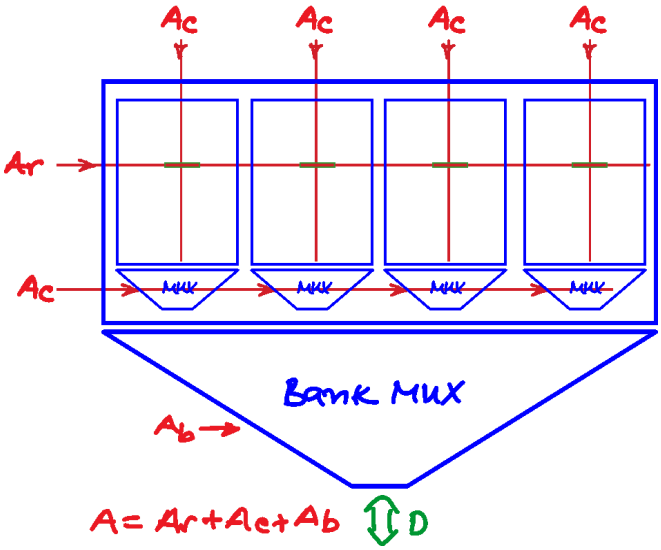
- Address size: $n = n_r + n_c$



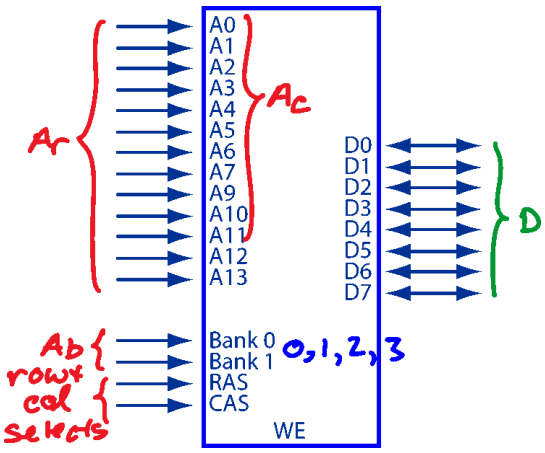

- *Example:* $n_r = 14$, $n_c = 12$, $m = 8$: $C = 64$ MB



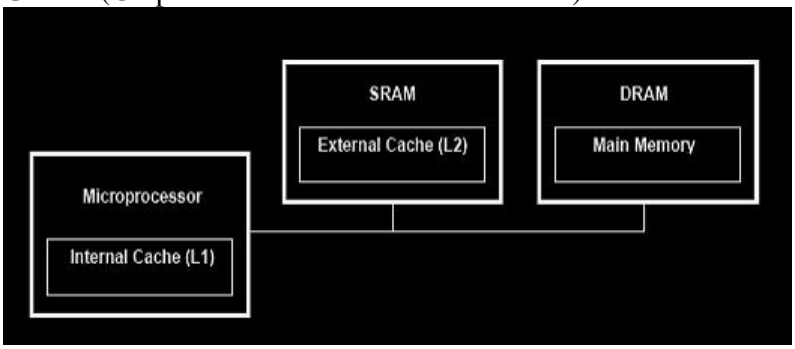

- **2D decoding with bank organization**
  - Address has $n_r$ and $n_c$ bit row decoding as before
  - Address also includes 1 or more bank lines $n_b$
  - Address size: $n = n_r + n_c + n_b$




- *Example:* $n_r = 14$, $n_c = 12$, $n_b = 2$, $m = 8$: $C = 256$ MB




- Memory packaging: computer.howstuffworks.com/ram4.htm
  - SIMM (single in-line memory module)
  - DIMM (dual in-line)
  - SODIMM (small outline dual in-line)
- Memory acronyms:
  - **SRAM** Static RAM (Used for registers, and caches)
  - **DRAM** Dynamic RAM (older memories)
  - **SDRAM** Synchronous Dynamic RAM (current memories)
  - **DDR SDRAM** Double Data Rate SDRAM (latest memories)
  - **LPDDR SDRAM** Low Power Double Data Rate SDRAM (latest memories)
  - **GDDR** (Graphics Double Data Rate memories)

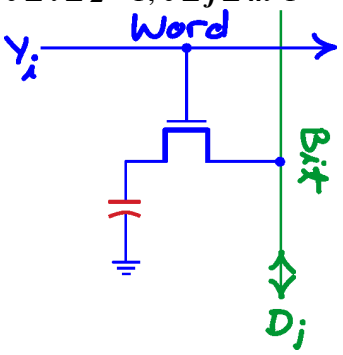



- Dynamic Memory
  - Synchronous Dynamic (SD) (self-refreshing)
  - Double Data Rate (DDR) SD (faster)
    - DDR2 = "2x more" than DDR
    - DDR4 = "4x more" than DDR2
    - DDR5 = "3x more" than DDR4
    - LPDDR4 & 5= Low power versions
  - Cell organization
    - Single transistor.

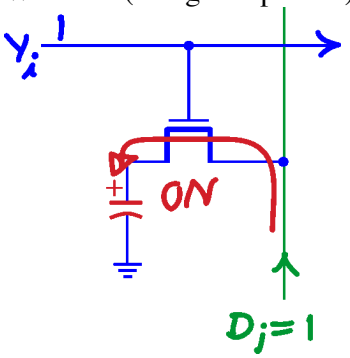
- Storage provided by capacitor, which leaks and requires refreshing by CPU.


Image: quora.com

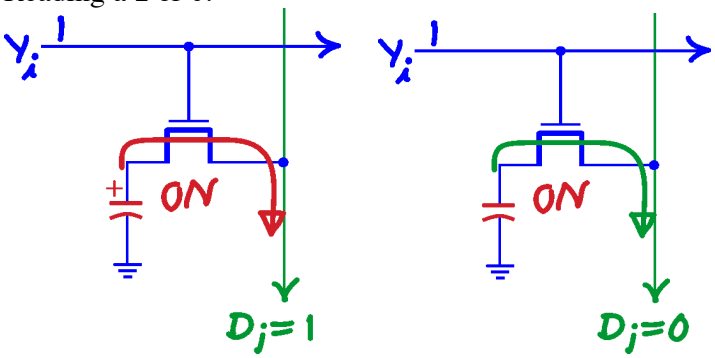
- DRAM cell showing $i^{th}$ word and $j^{th}$ bit lines plus capacitor:
  $\mathbf{0} \le i \le 2^n\text{-}1$, $\mathbf{0} \le j \le m\text{-}1$



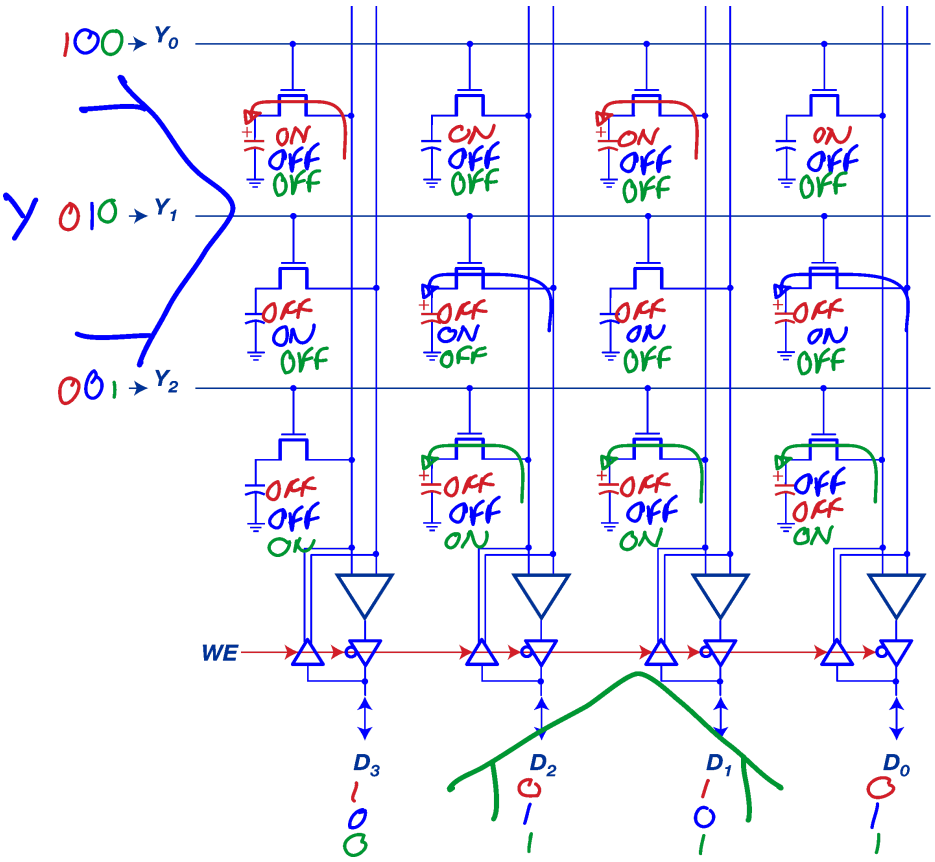

- Write a **1** (charges capacitor):



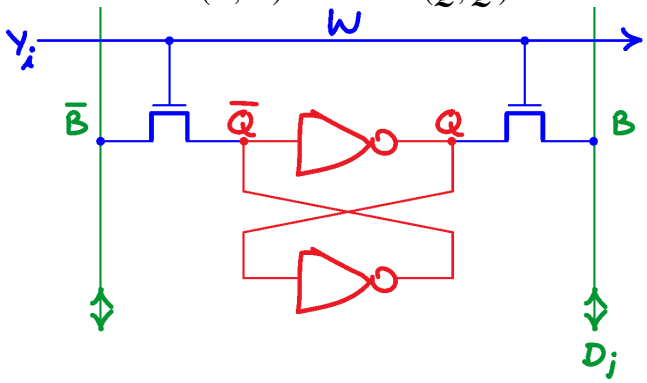

- Reading a **1** or **0**:




- **Dynamic RAM** (**DRAM**) layout example
  - Decoder transistor array, as before.
  - Encoder transistor array
    - Two data lines for each cell
    - Read: senses presense of charge in capacitors between selected and nonselected word. Charge = 1, no charge = 0
    - Write: charges capacitors for cells where data = 1 in selected word (write operations shown below)




- Static (no refresh, uses gates)
  - Uses two **cross-coupled NOT gates**.
  - $Q = 0, Q = 1$ are the 2 memory states
  - Cell access provided by transitors with word line ($W$)
  - Bidirectional bit lines ($B, B'$) connect to ($Q, Q'$)




***SMARTPHONES***

**iPhones and Androids**

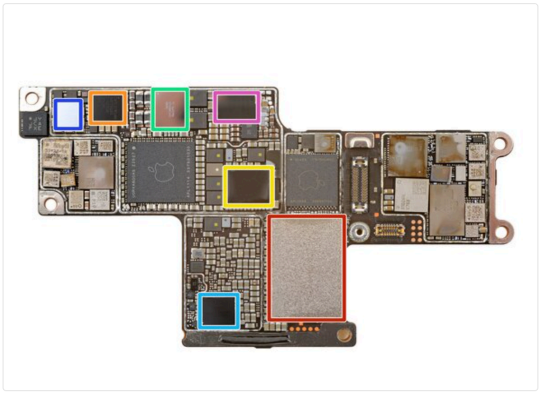
- Mobile Communication Device
  - Cell phone (voice/texting)
  - Media player (music/video)
  - Internet Browser (web/email/facebook/twitter)
  - Camera

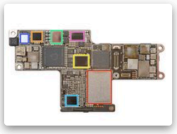
iPhone design

- iPhone 17 Air side view
  https://www.youtube.com/watch?v=woya8vjeFpo

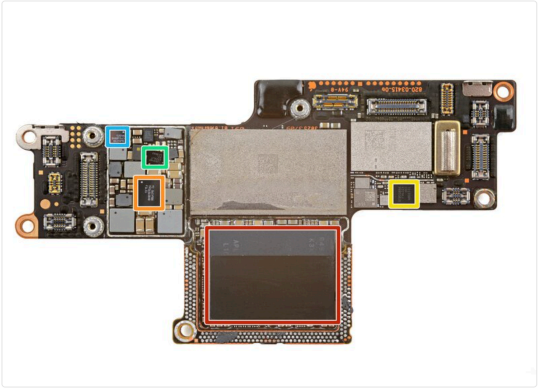

iphoneincanada.ca

- iPhone 17 Pro Main board top and back https://www.ifixit.com/Guide/iPhone+Air+Chip+ID/195913



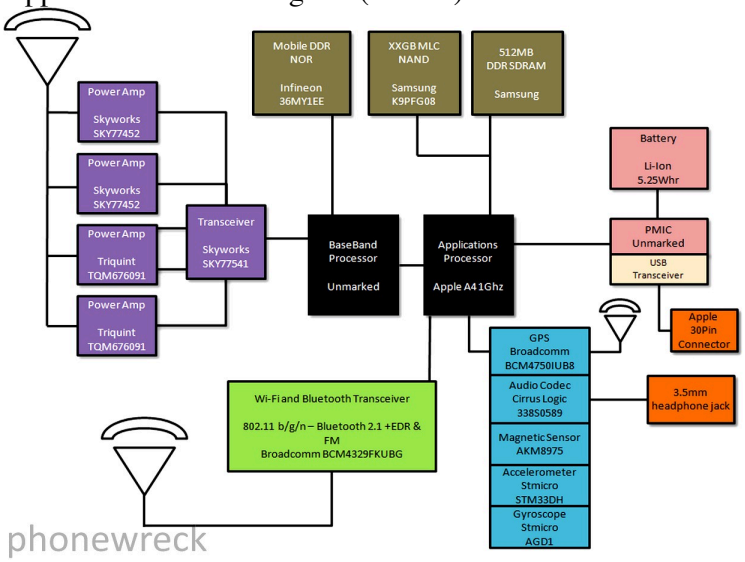


- Kioxia K8A7RJ5126 ? 256 GB NAND flash memory
- NXP Semiconductors CBTL1701B0 USB Type-C port controller
- Apple APL1064/338S01279 power management
- Texas Instruments CP3200C1H3 battery charger
- Apple/Cirrus Logic 338S00967 audio codec
- Apple/Cirrus Logic 338S01148 audio amplifier
- Broadcom BCM59367A1IUBG wireless charging controller



- Apple APL1V12/339S01839 hexa-core applications processor w/ GPU & Neural Engine layered under a Samsung K3KLMLM0FM-HGGV 12 GB LPDDR5X SDRAM
- Texas Instruments TPS65658A0 display power supply
- Analog Devices MAX11390A analog to digital converter
- Analog Devices MAX20348P DC-DC converter
- Possibly Texas Instruments CP5100A0 camera flash controller

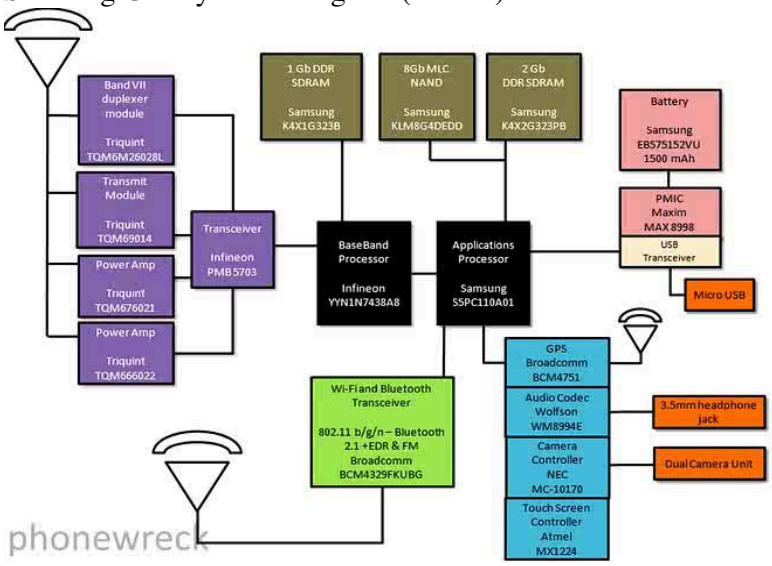
- Apple iPhone block diagram (7/2010)



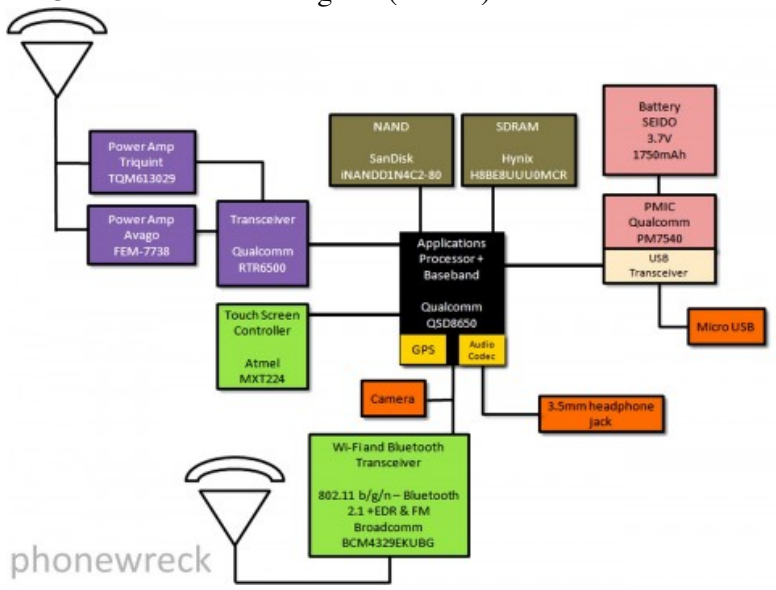

- iPhone Block Diagram Components
  - Application and Baseband Processors
  - DDR SDRAM
  - NAND Flash
  - NOR Pseudo SRAM
  - Cell phone Transceiver & Power Amps
  - Accelerometer/Magnetometer
  - WiFi and Bluetooth Transceiver
  - Magnetometer/Accelerometer/Gyroscope
  - 3-Axis Gyroscope introduced by Steve Jobs, 2010
  - USB & Audio Ports
  - Battery

Android design

- Samsung Galaxy block diagram (9/2010):

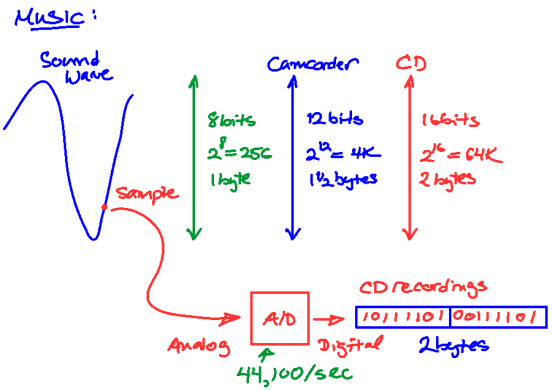


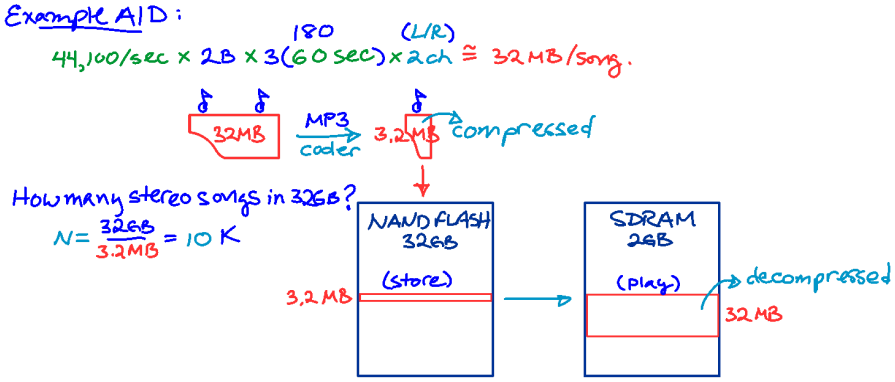
- HTC Incredible block diagram (1/2011)



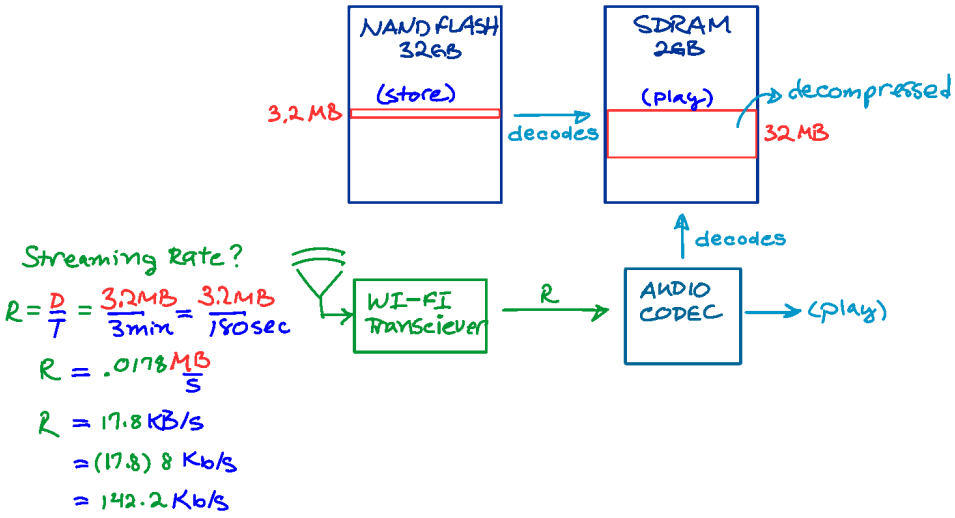

**Storing and Playing Music**

- Example: Storing Music
  - Music is sampled at 44,100/sec using 2B for each sample
  - Thus a three minute song file size is 32MB.
  - This song is then compressed to 3.2 MB to store in flash
  - 32GB of flash can store 10,000 3 minute songs
  - The song is decompressed to RAM where it is played as a 32MB song



- Example: Streaming Music
  - A 3.2 MB song is streamed on WiFi and is routed to Audio-Codec
  - A three minute song is streamed at a rate of 142.2 Kb/sec
  - The Audio-Code decodes the song back to 32 MB and is played in real time or is stored in RAM




**Terminology**:

- ARM: Advanced Risc Machines, Ltd.
  - Makes reduced instruction set processor templates
  - Licensed by virtually all major cell phone manufacturers
    - Processor license: Android
    - Architecture license: Apple (iPhone 5 and later)
  - Requires less transistors and power
- SoC: System on a Chip
  - Fabrication of major components of a smartphone onto a single chip
  - Typical integrations:
    - Communications (Baseband Processor)
    - Application Processor (CPU)
    - Graphics Processor (GPU)
    - Memory
- Applications Processor
  - CPU
  - GPU
  - NPU
  - Controller
  - Apple A17 chip -> 2023 iPhone 15 Pro

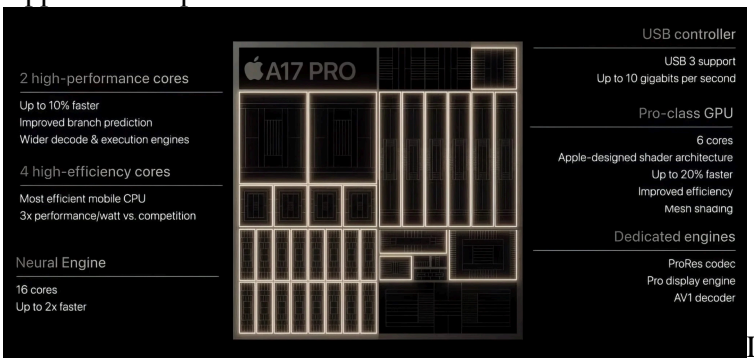


Image: Semi Analysis

- Apple A18 chip -> 2024 iPhone 16 Pro

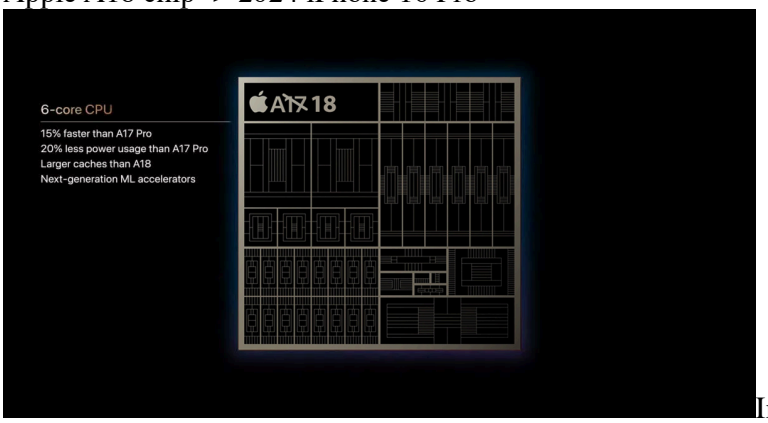


Image: MacWorld

- Apple A19 chip -> 2025 iPhone 17 Pro (same N3 family as above)

**Some History** www.anandtech.com/tag/smartphones

- iPhone 4 (2010)
  - A4 SoC = ARM Cortex A8 (800MHz) plus an
  - Imag Tech PowerVR SGX 535 GPU
  - 512 MB RAM
- Samsung Galaxy S1 (2010)
  - Exynos 3 SoC = ARM Cortex A8 (1 GHz), plus a
  - Power VR SGX540 GPU
  - 512 MB Mobile DDR RAM
- HTC Droid Incredible (2010)
  - Snapdragon S1 SoC = Qualcomm QSD8650 (1 GHz) plus a
  - Qualcomm Adreno 200 GPU
  - 512 MB RAM
- iPhone 4s (2011-12)
  - A5X SoC = ARM Cortex A9 (1 GHz, dual core) plus an
  - Imag Tech PowerVR SGX 543MP4 (4 GPUs)
  - 512 MB RAM
- Samsung Galaxy S3 (2012)
  - Snapdragon S3 SoC = Qualcomm MSM8960 (1.5 GHz, Krait 200 dual core), plus a
  - Qualcomm Adreno 250 GPU
  - 2 GB RAM
- HTC One S (2012)
  - Snapdragon S4 Krait SoC = Qualcomm MSM8260A (1.5 GHz, Krait 300 dual core) plus a
  - Qualcomm Adreno 225 GPU
  - 1 GB RAM
- Samsung Galaxy S6 (2015)
  - Exynos 7420 (4x 2.1/4x 1.5 GHz, Big/LP little hex core), plus a
  - Mali T760MP8 GPU (772 MHz)
  - 3 GB LPDDR4 RAM
- iPhone 6s (2015)
  - 64-bit A9 Enhanced Cyclone SoC = ARMv9 architecture (1.85 GHz, dual core) plus an
  - Imagination Tech PowerVR Series 4 GT7600 GPU (6 cores)
  - 3D Touch Display and Taptic Engine
  - 4 MB SRAM
  - 2 GB LPDDR4 SDRAM
- Samsung Galaxy S7 (2016)
  - Kyro (2x 2.15/2x 1.6 GHz, Big/LP little quad core), plus a
  - Adreno 530 GPU (650 MHz)
  - 4 GB LPDDR4 RAM
- iPhone 7 (2016)
  - 64-bit A10 Fusion SoC = ARMv10 architecture (4x 2.3 GHz Big/4x LP little hexa-core) plus an

    - Imagination Tech PowerVR GPU (6 cores)
    - 3D Touch Display and Taptic Engine
    - 8 MB SRAM
    - 2 GB LPDDR4 SDRAM
- Samsung Galaxy S8 (2016)
    - Qualcomm Snapdragon 835 Kyro 280 (2.35/1.9 GHz octa-core), 4x 2MB L2 cache, plus a
    - Adreno quad-core 540 GPU (670 MHz)
    - 4 GB LPDDR4 RAM
- iPhone X (2017)
    - 64-bit A11 Bionic SoC = ARMv11 architecture (2x 2.39 GHz 4x Big/LP little hexa-core) plus an
    - Apple custom GPU (3 cores)
    - 3D Touch Display and Taptic Engine
    - 2 core neural engine
    - 8 MB SRAM
    - 3 GB LPDDR4X SDRAM
- Samsung Galaxy S9 (2018)
    - Qualcomm Snapdragon 845 Kyro 385 (4x 2.8/4x 1.77 GHz octa-core), 4x 256KB L2, 4x 128KB L2, 2MB L3, 3MB L4 cache, plus a
    - Adreno dual-core 630 GPU (710 MHz)
    - 4 GB LPDDR4 RAM
- iPhone XS (2018)
    - 64-bit A12 Bionic G11P SoC = ARMv12 architecture (2x 2.5/4x 1.59 GHz Big/LP little hexa-core) plus an
    - Apple custom G11-P GPU (4 cores)
    - 8 core neural engine
    - 3D Touch Display and Taptic Engine
    - 8 MB SRAM
    - 4 GB LPDDR4X SDRAM
- Samsung Galaxy S10 (2019)
    - Qualcomm Snapdragon 855 Kyro 485 (1x 2.84/3x 2.42/4x 1.80 GHz octa-core), 1x 512KB L2, 3x 256KB L2, 4x 128KB L2, 2MB L3, 3MB L4 cache, plus a
    - Adreno dual-core 640 GPU (??? MHz)
    - 6 GB LPDDR4 RAM
- iPhone 11 Pro (2019)
    - 64-bit A13 Bionic G11P SoC = ARMv13 architecture (2x 2.66 (Lightning)/4x 1.73 (Thunder) GHz Big/LP little hexa-core) plus an
    - Apple GPU (4 cores)
    - 8 core neural engine
    - 16 MB SRAM Cache
    - 4 GB LPDDR4X SDRAM
- Samsung Galaxy S20 (2020)
    - Qualcomm Snapdragon 865 SoC (Cortex A77 1x 2.84/Kiro 3x 2.42/4x 1.80 GHz octa-core), plus a
    - Adreno 650 GPU (587 MHz)
    - 6 GB LPDDR4 RAM
    - Snapdragon 5G
- Google Pixel 5 (2020)
    - Qualcomm Snapdragon 765 SoC (1x 2.4/1x 2.2/6x 1.8 GHz), plus a
    - Arm Mali-G77 11-core GPU (??? MHz)
    - 8GB LPDDR4X RAM
    - Snapdragon 5G
- iPhone 12 Pro (2020)
    - 64-bit A14 Bionic SoC = ARMv14 architecture (2x 2.99 (Firestorm)/4x 1.82 (Icestorm) GHz Big/LP little) plus an
    - Apple GPU (4 cores)
    - 16 core neural engine
    - 16 MB SRAM Cache
    - 4 GB LPDDR4X SDRAM
    - 5G (sub-6 GHz and mmWave)

- Samsung Galaxy S21 Ultra (2021)
    - Qualcomm Snapdragon 888 Kyro SoC (Cortex X1 1x 2.84/Kiro 680 3x 2.42/4x 1.80 GHz), plus a
        - Adreno 660 GPU (750 MHz)
    - 6/8 GB LPDDR5 RAM
- Google Pixel 6 Pro (2021)
    - Google Tensor SoC (Cortex X1 1x 2.80/Cortex A76 2x 2.25/4x 1.80 GHz), plus a
    - Mail G78 GPU (20 cores)
    - Google EdgeTPU NPU
    - 8 GB LPDDR5X RAM
- iPhone 13 Pro (2021)
    - 64-bit A15 Bionic SoC = ARMv14 architecture (2x 3.24 (Avalanche)/4x 2.0 (Blizzard) GHz Big/LP little) plus an
    - Apple GPU (5 cores)
    - 16 core neural engine
    - 24 MB SRAM Cache
    - 6  GB LPDDR5X SDRAM
- Samsung Galaxy S22 (2022)
    - Qualcomm Snapdragon 888 Kyro SoC (Cortex X2 1x 3/A-710 3x 2.50/A-510 4x 1.80 GHz), plus a
    - Adreno 730 GPU (800 MHz)
    - 8 GB LPDDR5 RAM
- Google Pixel 7 Pro (2022)
    - Google Tensor SoC (Cortex X1 2x 2.85/Cortex A78 2x 2.35/CortexA55 4x 1.80 GHz, plus a
    - Mali-G710 MP7 GPU
    - Google EdgeTPU NPU
    - 8 GB LPDDR5X RAM
- iPhone 14 Pro (2022)
    - 64-bit A16 Bionic SoC = ARMv14 architecture (2x 3.46 (Everest)/4x 2.02 (Sawtooth) GHz) plus an
    - Apple GPU (5 cores)
    - 16 core neural engine
    - 24 MB SRAM Cache
    - 6 GB LPDDR5X SDRAM
- Samsung Galaxy S23 (2023)
    - Qualcomm 8550 AC Snapdragon 8 Gen 2 SoC: Octa-core CPU (1×3.36 GHz X3, 2×2.8 GHz A715, 2×2.8 GHz A710, 3×2.0 GHz A510) plus an
    - Adreno 740 GPU clocked at ~719 MHz
    - 8 GB LPDDR5 RAM
- Google Pixel 8 Pro (2023)
    - Google Tensor G3 SoC (Nona-core (1×2.91 GHz Cortex-X3 & 4×2.37 GHz Cortex-A715 & 4×1.70 GHz Cortex-A510), plus a
    - Immortalis-G715s MC10 GPU (~890 MHz)
    - Google Edge TPU NPU
    - 12 GB LPDDR5X RAM
- iPhone 15 Pro (2023)
    - The Apple A17 Pro SoC (Hexa-core (2×3.78 GHz Cortex-P + 4×2.11 GHz Cortex-E), plus an
    - Apple GPU (6 cores)
    - 16 core Neural Engine
    - 24 MB SRAM Cache
    - 8 GB LPDDR5 RAM
- Samsung Galaxy S24 (2024)
    - Qualcomm 8650 AC Snapdragon 8 Gen 3 SoC: Octa-core CPU (1 × 3.39 GHz Cortex-X4, 3 × 3.10 GHz Cortex-A720, 2 × 2.90 GHz Cortex-A720, 2 × 2.20 GHz Cortex-A520) plus an
    - Adreno 750 GPU (≈ 1.00 GHz)
    - 8 GB LPDDR5X RAM
- Google Pixel 9 Pro (2024)
    - The Google Tensor G4 SoC (Octa-core (1 × 3.10 GHz Cortex-X4 & 3 × 2.60 GHz Cortex-A720 & 4 × 1.92 GHz Cortex-A520), plus a
    - Mali-G715 (≈ 940 MHz) GPU

    - Google EdgeTPU/NPU
    - 16 GB LPDDR5X RAM
- iPhone 16 Pro (2024)
    - The Apple A18 Pro SoC (Hexa-core (2×4.04 GHz + 4×2.42 GHz) plus an
    - Apple GPU (6 cores)
    - 16 core Neural Engine
    - 24 MB SRAM Cache
    - 8 GB LPDDR5X RAM
- Samsung Galaxy S25 (2025)
    - Qualcomm 8750 AB Snapdragon 8 'Elite' for Galaxy SoC: Octa-core CPU (2 × 4.47 GHz Oryon (Prime) & 6 × 3.53 GHz Oryon (Efficiency)) plus an
    - Adreno 830 GPU
    - 12 GB LPDDR5X RAM
- Google Pixel 10 Pro (2025)
    - The Google Tensor G5 SoC (Octa-core (1×3.20 GHz Cortex-X5 & 3×2.70 GHz Cortex-A720 & 4×2.10 GHz Cortex-A520), plus an
    - Immortalis-G7xx GPU (≈ 1.10 GHz)
    - Google Edge TPU/NPU
    - 16 GB LPDDR5X RAM
- iPhone 17 Pro (2025)
    - The Apple A19 Pro SoC (Hexa-core (2×4.26 GHz & 4×2.60 GHz), plus an
    - Apple GPU (6 cores)
    - 16 core Neural Engine
    - 32 MB SRAM Cache
    - 12 GB LPDDR5X RAM

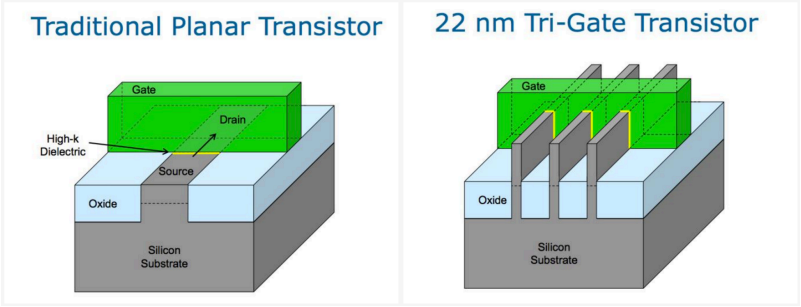
**Recent performance enhancements**

- Apple SoC Relative performance (Image: Anandtech.com)

| Apple SoC Evolution | | | |
|---|---|---|---|
| | Die Size | Transistors | Process |
| A5 | 122m2 | <1B | 45nm |
| A6 | 97mm$^2$ | <1B | 32nm |
| A7 | 102mm$^2$ | >1B | 28nm |
| A8 | 89mm$^2$ | ~2B | 20nm |
| A9 | 96mm$^2$/104.5mm$^2$ | >2B | 14nm/16nm |

    - Plus... A10 ----> 14/16 nm
    - and.... A11 ----> 10 nm
    - and.... A12 ----> 7 nm (6.8 B Transistors)
    - and.... A13 ----> 7 nm (8.5 B Transistors)
    - and.... A14 ----> 5 nm (11.8 B Transistors)
    - and.... A15 ----> 4 nm (12 B Transistors)
    - and.... A16 ----> 4 nm (13 B Transistors)
    - and.... A17 ----> 3 nm (19 B Transistors) N3B = N3 family
    - and.... A18 ----> 3 nm (?? B Transistors) N3E
    - and.... A19 ----> 3 nm (?? B Transistors) N3P

- Reduced power leakage and increased switching speed provided by finFET transistors (Image: Anandtech.com)



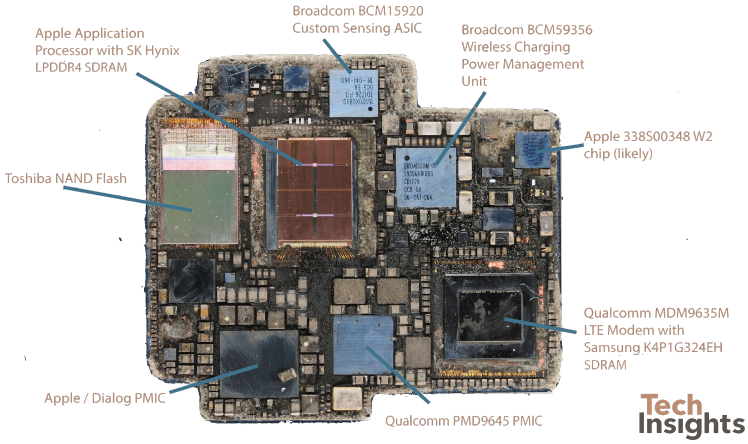

**Apple Watch**

- Combination smart device and jewelry (wearable device)
- Pairs with iPhone to run apps
- Independent cellular

- S6 Teardown:

(Image: ifixit.com)

- S3 SoC, encased in resin

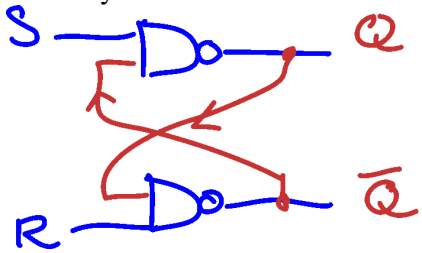



    - Apple Watch hardware Series 3 (2010)
        - 312 x 390 resolution display
        - The Apple S3 SiP Dual-Core CPU, plus a
        - 8-16 GB NAND
        - PowerVR GPU
        - Force Touch Display and Taptic Engine
        - 768 MB of DRAM
    - Apple Watch Series 11 (2025)
        - 374 x 446 resolution display
        - The Apple S10 SiP Dual-core CPU, plus an
        - GPU
        - 4 core Neural Engine
        - 64 GB NAND
        - 1 GB DRAM
    - The Series 3 roughly matches the iPhone 4 (2010):
        - A4 SoC = ARM Cortex A8 (800MHz) plus an
        - Imag Tech PowerVR SGX 535 GPU
        - 32 GB NAND
        - 512 MB RAM

## *LATCHES AND FLIP-FLOPS*

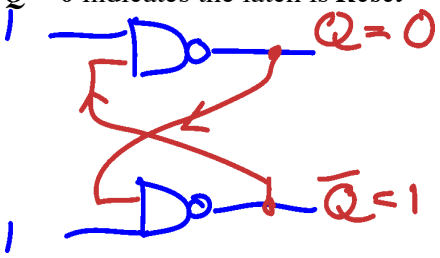
### Simple Latch Circuit

- Performs the simplest kind of **binary information storage**, using gates.
- Circuitry consists of two **cross-coupled NAND** (or **NOR**) **gates**. Inputs (***S, R***) are *active-low*.

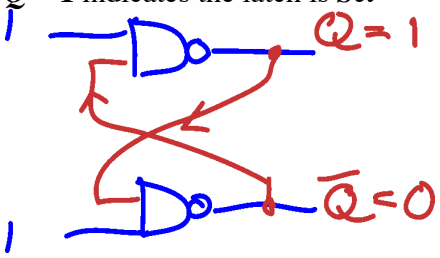



- Latch **state** is specified by the latch outputs;

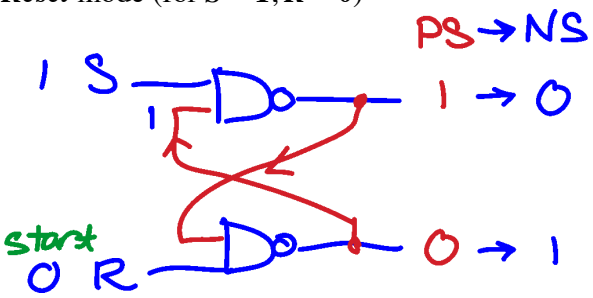
- $Q$ = **0** indicates the latch is ***Reset***



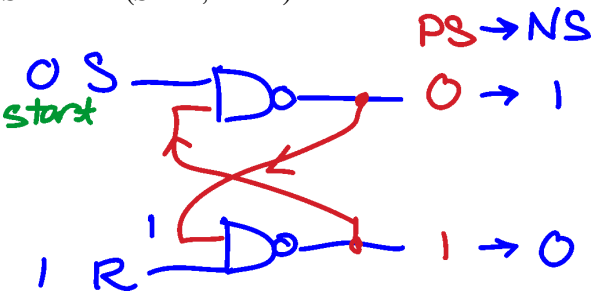

- $Q$ = **1** indicates the latch is ***Set***



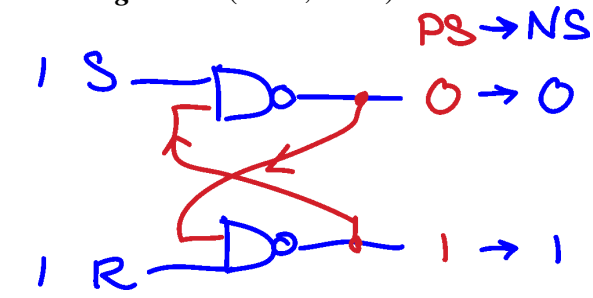

- There are ***three modes*** of operation (changing a **Present State** to a **Next State**):
  - ***Reset*** mode (for $S$ = **1**, $R$ = **0**)



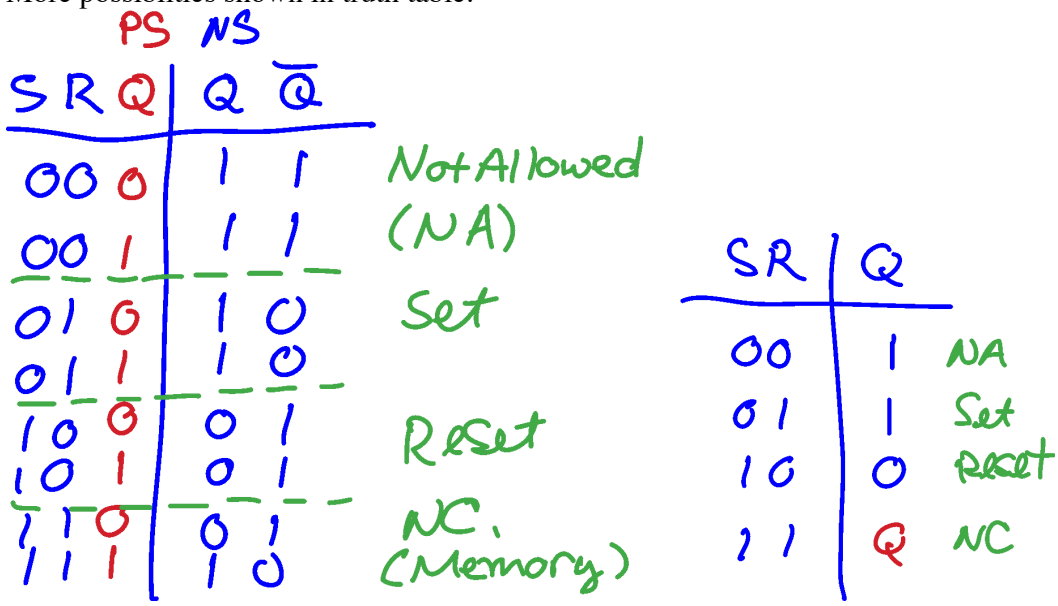

  - ***Set*** mode ($S$ = **0**, $R$ = **1**)



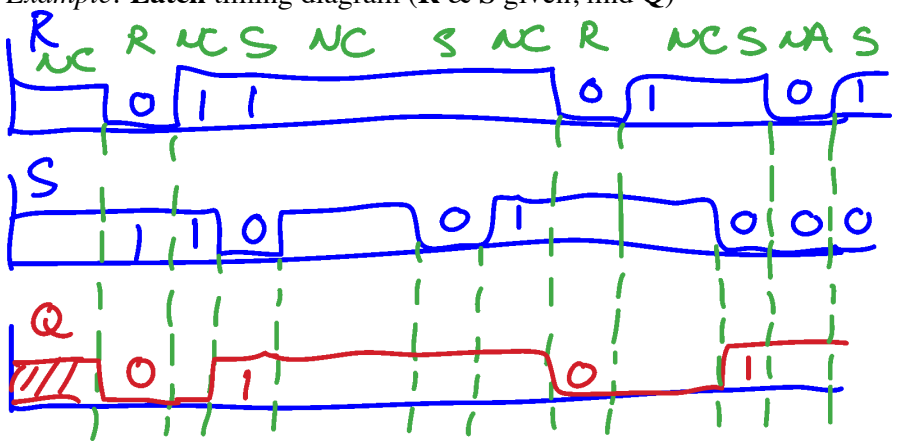

  - ***No Change*** mode ($S$ = **1**, $R$ = **1**)



- More possibilties shown in truth table:

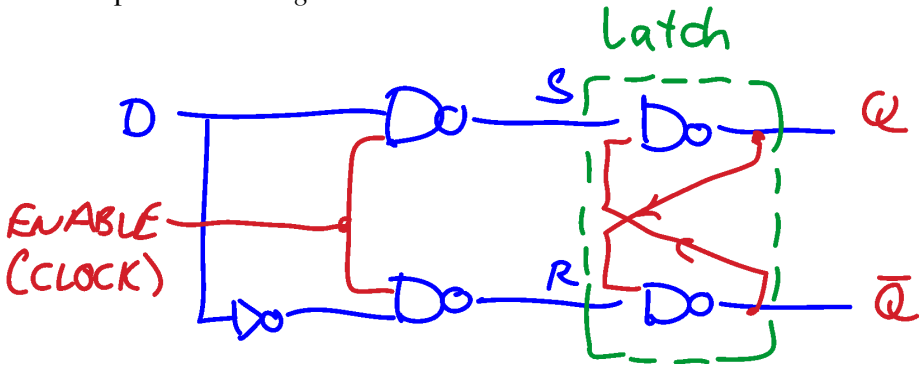



- *Example:* **Latch** timing diagram (**R** & **S** given, find **Q**)



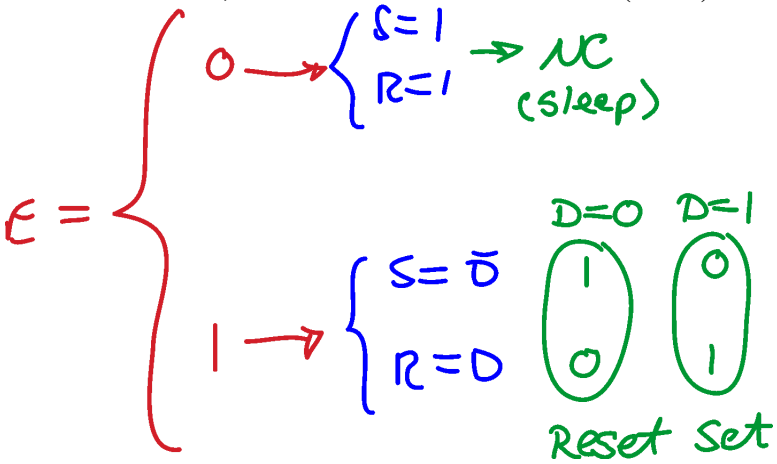

**Data Latch with Enable (D-Latch)**

- **Circuitry** consists of a **simple latch** plus additional logic under the control of an enable signal ***E*** and input ***D***. The ***D*** input is *active-high*.

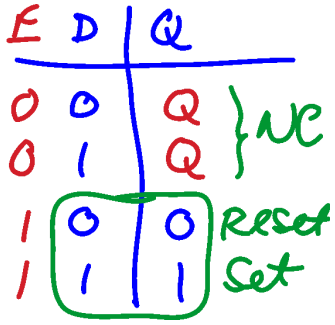



- There are ***three modes*** of operation:
  - When ***E* = LOW**, mode = ***No Change*** (this puts the latch to sleep)

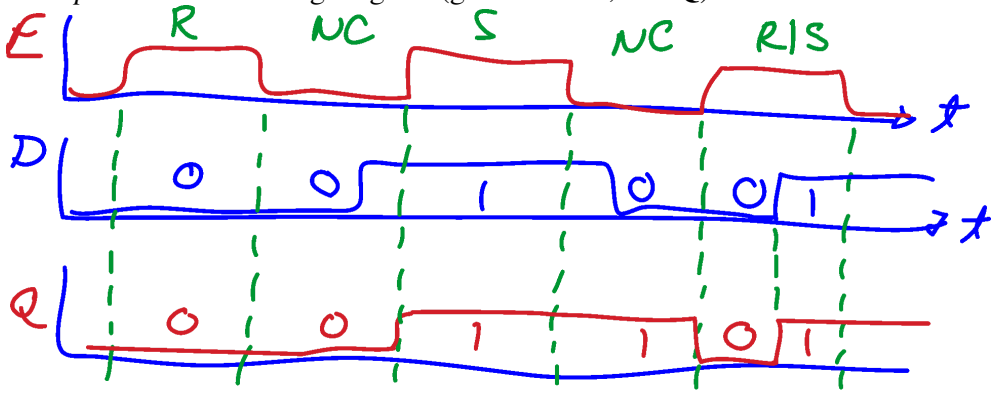
- When ***E* = HIGH**, the two ***active modes*** are ***Reset* (*D* = 0)** *or* ***Set* (*D* = 1)**, as determined by ***D***.



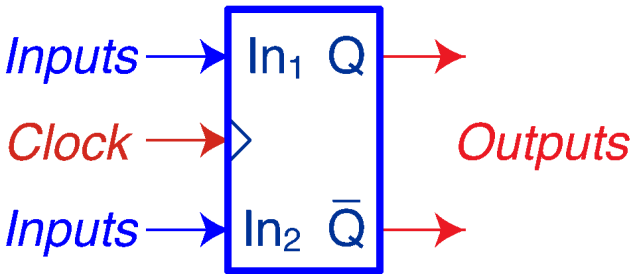

- The **enable input** forms the basis of a rudimentary clock signal.
- Truth table:



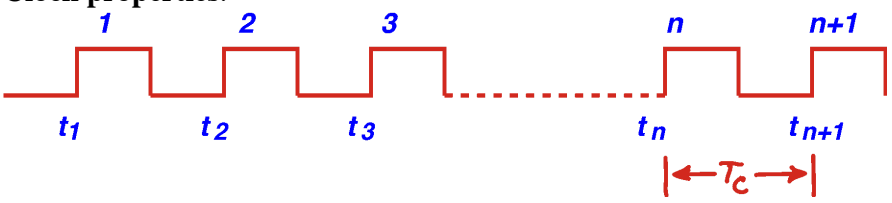

- *Example:* **D-Latch** timing diagram (given **E** & **D**, find **Q**)



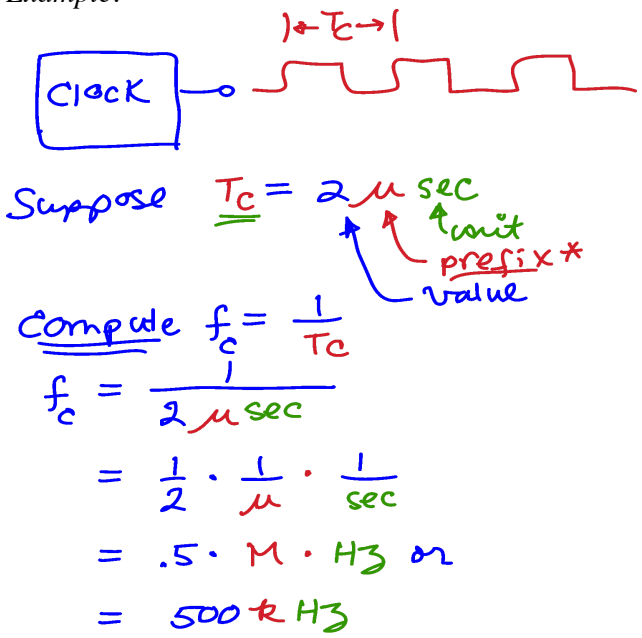

**Flip-Flop defined**: a **SLC** which stores one bit of binary information under the control of a set of **data input signals** and a **Clock**




**Clock properties**:



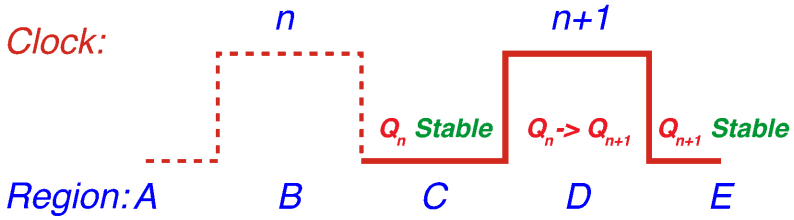
- The $n^{th}$ clock period begins at $t_n$.
- The $n^{th}$+1 clock period begins at $t_{n+1}$
- $T_c = t_{n+1} - t_n$ defines the length of the clock period
- $f_c = 1/T_c$, defines the clock frequency
- Time and frequency prefixes*:
  - $10^{-9} = n \leftrightarrow G = 10^{+9}$
  - $10^{-6} = \mu \leftrightarrow M = 10^{+6}$
  - $10^{-3} = m \leftrightarrow k = 10^{+3}$
- *Example*:




**Importance of Clocks:**

- Up part defines **daytime**, when flip-flops are active.
- Down part defines **nighttime**, when flipflops are sleeping.
- Flip-flop input signals (i.e. bread for breakfast) must come at night.

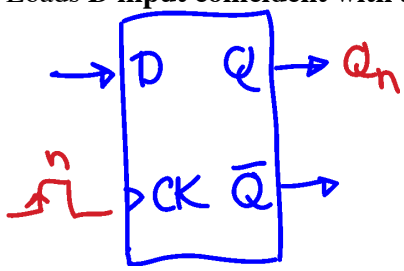
**Flip-Flop properties**



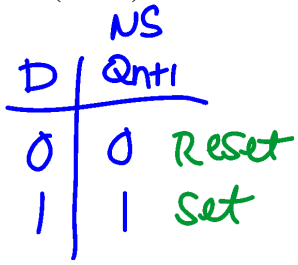

- $Q = Q_n$ defines the **F/F Present State** (Region **C**)
- $Q = Q_{n+1}$ defines the **F/F Next State** (Region **E**)
- $Q_n \rightarrow Q_{n+1}$ state transitions occur during the daytime (Region **D**)
- **F/F data inputs** must arrive the night before state transitions are made (Region **C**)

**Edge-Triggered D Flip-Flop (D F/F)**

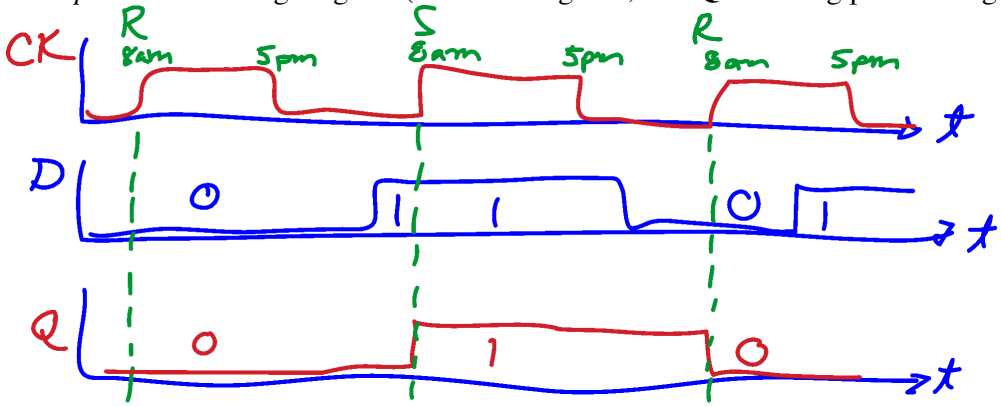
- Loads **D input coincident with a clock-pulse edge**; subsequent input changes are **locked out**



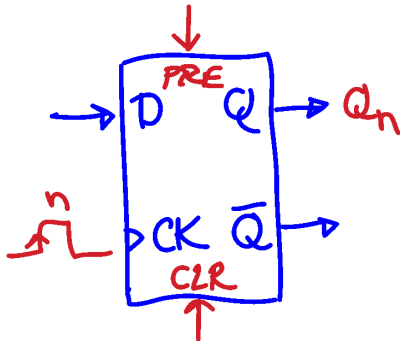

- There are ***two active modes*** of operation:
  - ***Reset*** **(*D* = 0)**
  - ***Set*** **(*D* = 1)**



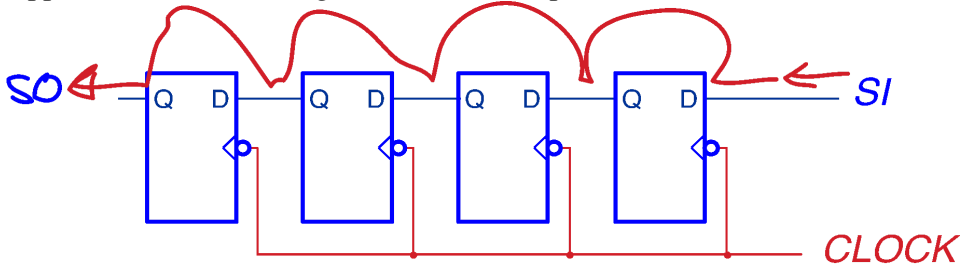

- State changes ($Q_n \rightarrow Q_{n+1}$) are triggered on **edges** of the clock:
  - Rising edges (↑) for positive edge triggered F/Fs
  - Falling edges (↓) for negative edge triggered F/Fs
- *Example:* **D** F/F timing diagram (**Clock** & **D** given, find **Q** assuming positive edge trig.)




- **D** F/F architecture is often augmented by an **asynchronous *S*** (or ***PRE)*** and ***R*** (or ***CLR***) **input overrides** on the slave latch.

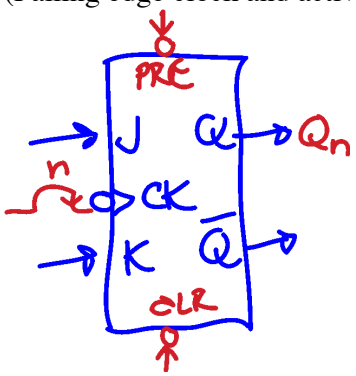



- Application: 4-bit shift register with serial input (see later)



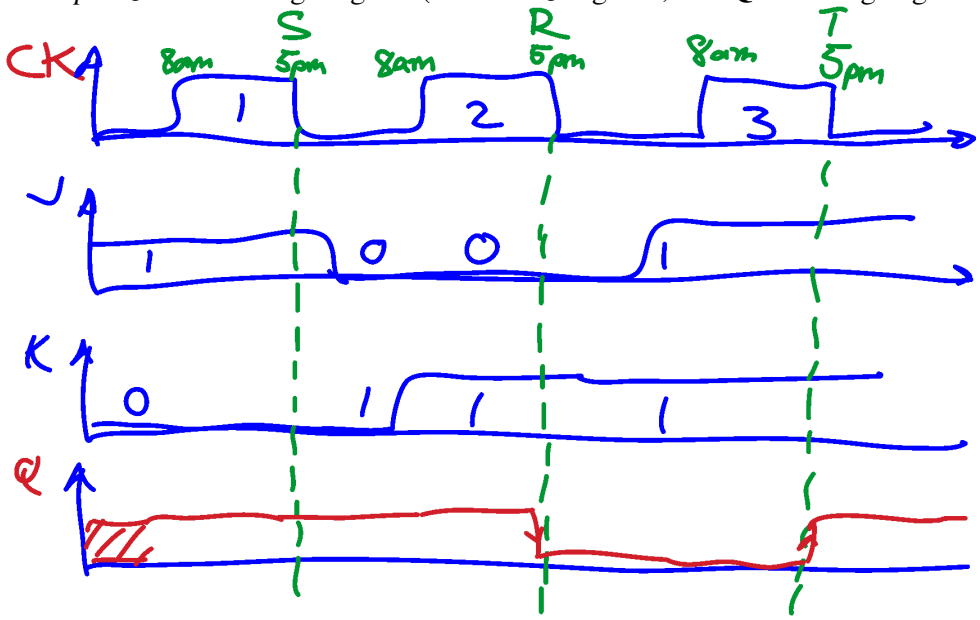
**Edge-Triggered Jack/Kill Flip-Flop (JK F/F)**

- Like **D** F/F, except with 2 inputs ***J*** and ***K***.
  (Falling edge clock and active-low preset inputs shown)

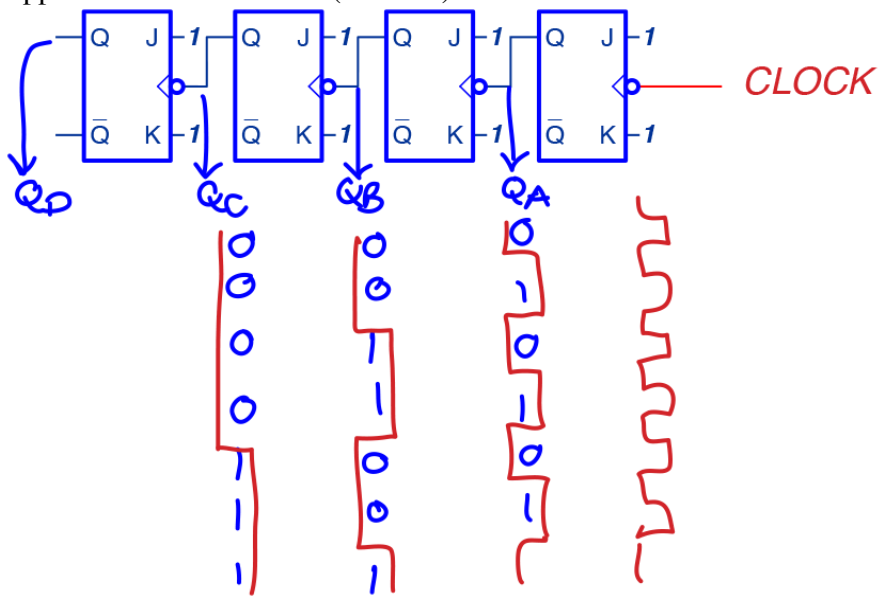



- There are ***four active modes*** of operation:
  - ***No Change*** ($J = 0, K = 0$)
  - ***Reset*** ($J = 0, K = 1$)
  - ***Set*** ($J = 1, K = 0$)
  - ***Toggle*** ($J = 1, K = 1$)


- *Example:* **JK** F/F timing diagram (**Clock** & **JK** given, find **Q** assuming negative edge trig.)



- Application: 4-bit Counter (see later)

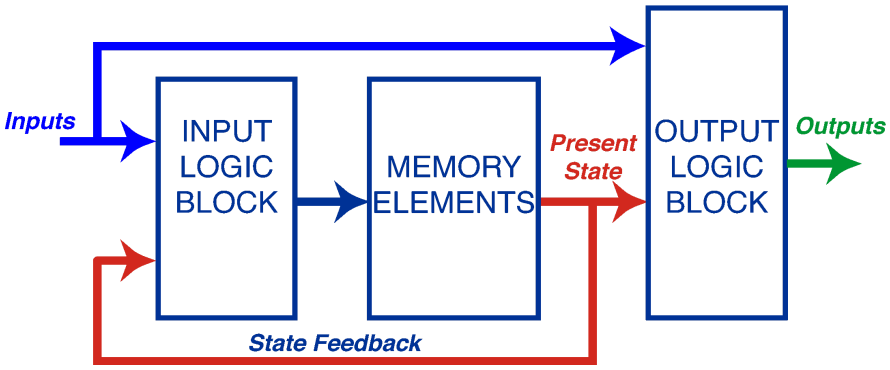



**Quiz #8 & selected solutions**

## *STATE MACHINE DESIGN*

**SLC defined**: logic circuits whose outputs depend upon **state**, also called a **state machine**.

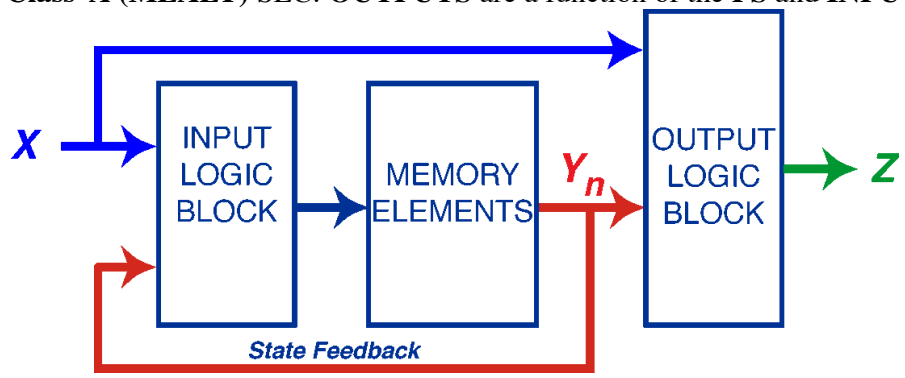
**General SLC Block Diagram**



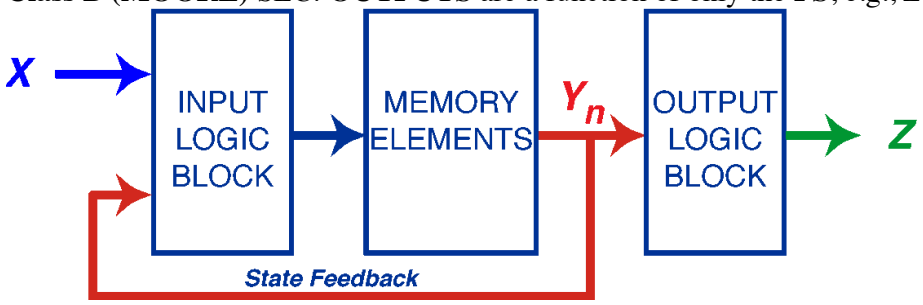

- The **Present State (PS)**: the bit pattern of the **MEMORY ELEMENTS** (a register) at a prescribed observation time.
- The **Next State (NS)**: the future state of the **MEMORY ELEMENTS**, formed by the **INPUT LOGIC BLOCK**.
- The **OUTPUTS**: the present output bit pattern, formed by the **OUTPUT LOGIC BLOCK**.

**SLC Classes**

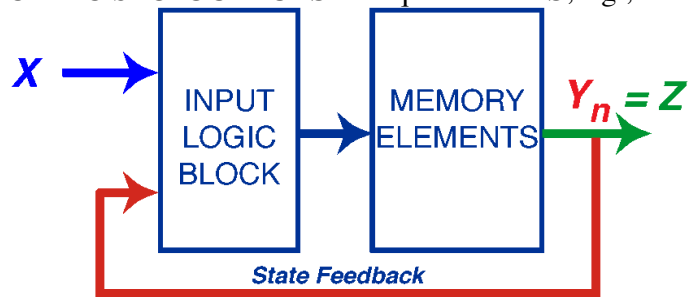
- **Class A (MEALY)** SLC: **OUTPUTS** are a function of the **PS** and **INPUTS**, e.g., $Z = f(X,Y)$.




- **Class B (MOORE)** SLC: **OUTPUTS** are a function of only the **PS**, e.g., $Z = f(Y)$.



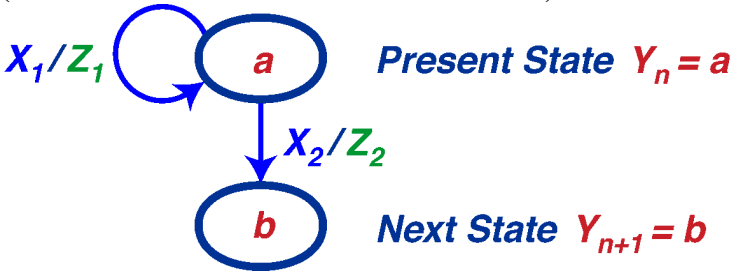

- **Class C** SLC: **OUTPUTS** are equal to the **PS**, e.g., $Z = Y$.




**SLC Variable Notation**

- $X$ = ***single input variable*** or ***vector of input variables*** ($X_1, X_2, X_3$,...).
- $Y$ = ***single state variable*** (representing flip-flop output $Q = Y$ ) or ***vector of state variables*** ($Y_1, Y_2, Y_3$,...).
- $Z$ = ***single output variable*** or ***vector of output variables*** ($Z_1, Z_2, Z_3$,...).
- $Y_n = Y$ **= PS**, $Y_{n+1}$ **= NS**

**The State Diagram (SD)**
(Shows the state to state transitions of a SLC)

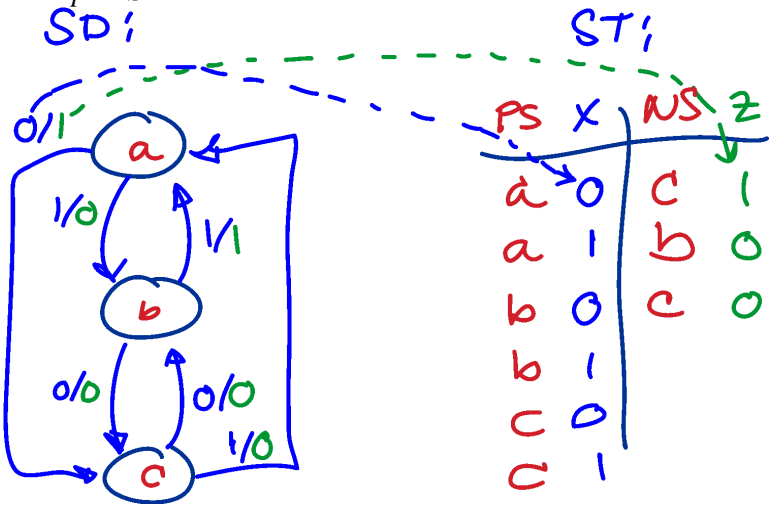



- **Bubbles**: represent the **SLC states** labeled with alphabetic letters (**a, b, c,** etc.) or with **state codes** (000, 001, 010,...)
- **Arrows** indicate allowable **state transitions** (Present to Next):
  - *Mealy*: Arrows are labeled with code ***X/Z*** (as above)
  - *Moore*: Arrows are labeled with ***X*** only; bubbles are labeled with code ***Y/Z***.

**The State Table (ST)**
(Tabulates **NS** and **Outputs** verses **PS** and **Inputs** of a **SD**)

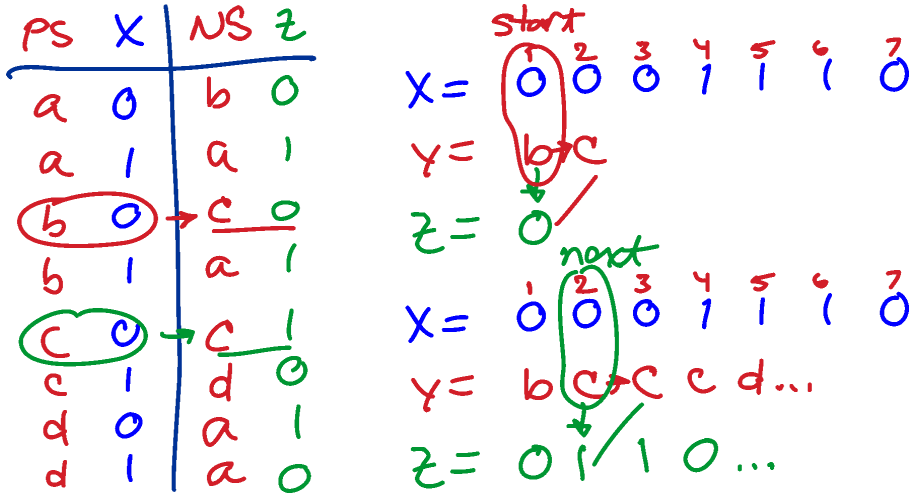
| PS In | | NS | Out | |
|---|---|---|---|---|
| $Y_n$ | $X$ | $Y_{n+1}$ | $Z$ | |
| a | $X_1$ | a | $Z_1$ | No Change |
| a | $X_2$ | b | $Z_2$ | Advance to b |

- Construction:
  - Examine **SD**: for every arrow, make a row in the **ST** having PS $Y_n$, Input $X$, NS $Y_{n+1}$ and Output $Z$.
  - *Example:* **ST** construction



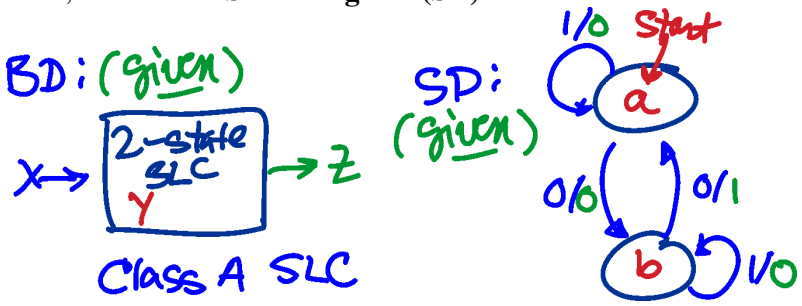

**State Trace**
(Lists states and outputs for a given sequence of input values)

- Trace format
  Similar to timing diagram, where $X$ is given and $Y$ and $Z$ need to be filled in:
  $X$: $X_1\ X_2\ X_3\ldots$ (sequence of inputs $X_n$)
  $Y$: $Y_1\ Y_2\ Y_3\ldots.$ (sequence of states $Y_n$)
  $Z$: $Z_1\ Z_2\ Z_3\ldots.$ (sequence of outputs $Z_n$)
- *Example:* Given ST & PS, construct trace ($X_1 = 0$ and $Y_1 =$ **b** are given)




**State Machine Design Procedure**
(The procedure will be illustrated by three examples)
***Example#1*:**

1. Given a set of specifications for a **SLC**, identify the **Inputs** and **Outputs** and draw a **Block Diagram (BD)**. Also, formulate a **State Diagram (SD)**.




2. Construct a **State Table (ST)** based on the **SD**.

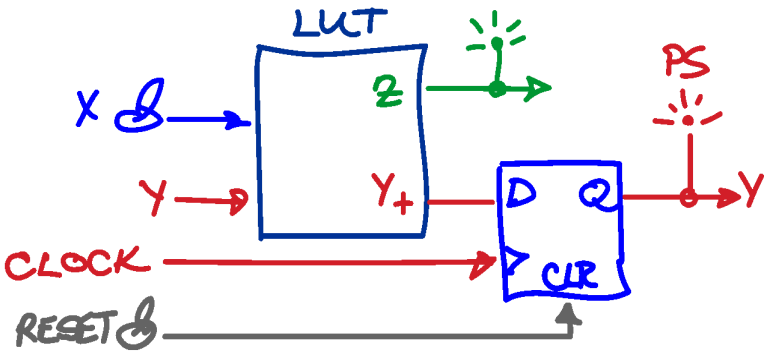
ST:

| PS IN | NS OUT |
|---|---|
| Yn X | Yn+1 Z |
| a 0 | b 0 |
| a 1 | a 0 |
| b 0 | a 1 |
| b 1 | b 0 |

a=0, b=1 →

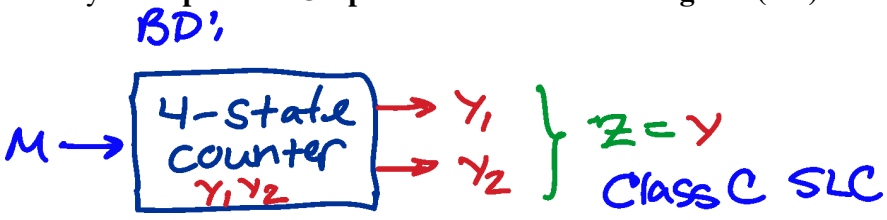
| A B | NS out |
|---|---|
| Yn X | Yn+1 Z |
| 0 0 | 1 0 |
| 0 1 | 0 0 |
| 1 0 | 0 1 |
| 1 1 | 1 0 |

3. Program the state table into a lookup table. Feed the next state output into a D flip-flop.

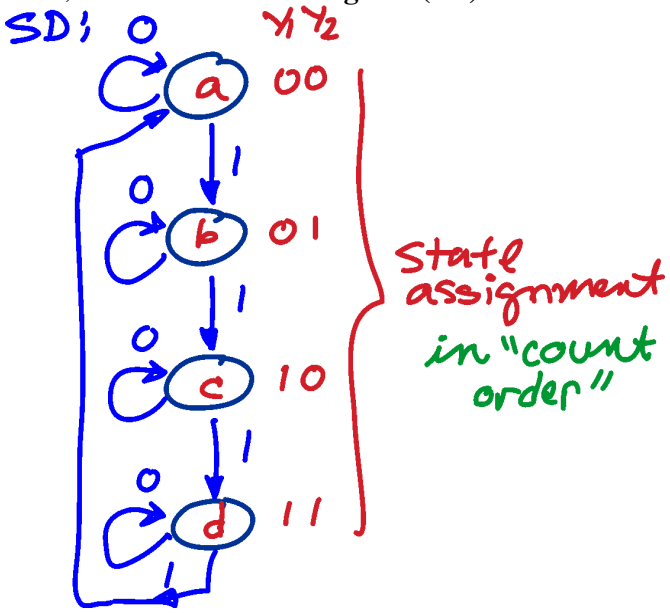



***Example#2*:**

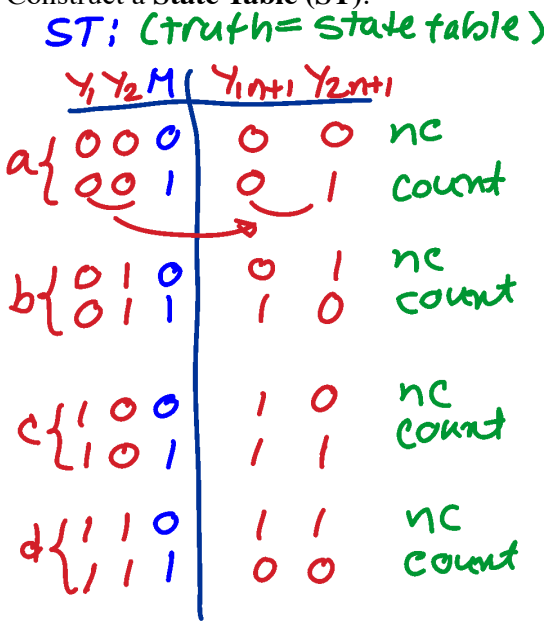
1. Given a set of specifications for a **SLC**:
   - Design a four state counter having a mode control input ***M***
     - When ***M*** = 0, do not count (no change)
     - When ***M*** = 1, count
   - Identify the **Inputs** and **Outputs** and draw a **Block Diagram (BD)**:



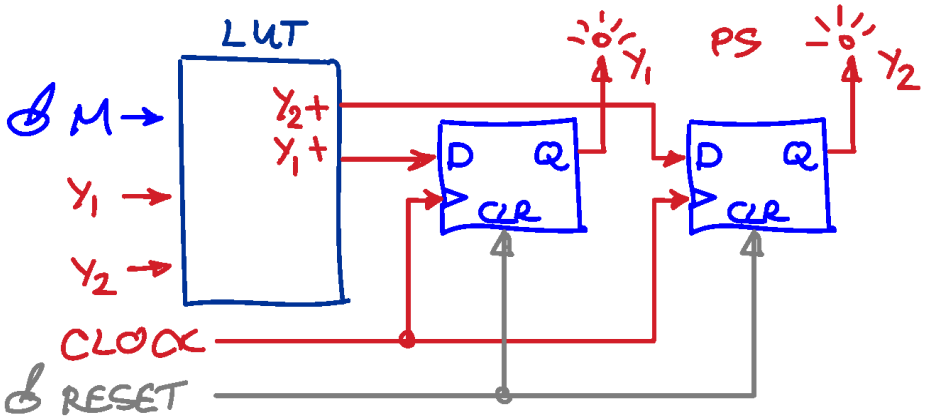
- Also, formulate a **State Diagram (SD)**:




2. Construct a **State Table (ST)**:



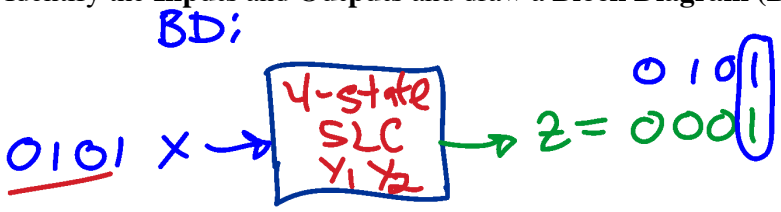

3. Program the state table into a lookup table. Feed the next state outputs to two D flip-flops.



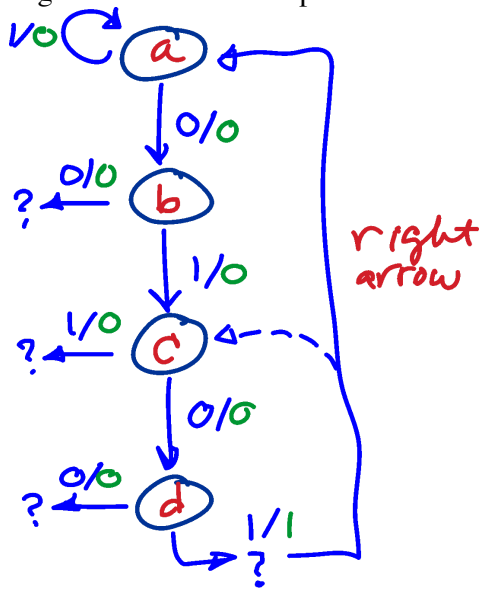

***Example#3*:**

1. Given a set of specifications for a **SLC**:

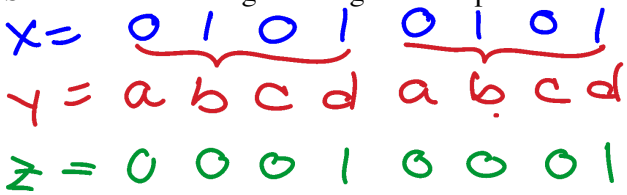
- Design a code sequence detector for the code ***X = 0101***
  - When $X = X_1X_2X_3X_4 = 0101$,
    then $Z = Z_1Z_2Z_3Z_4 = 0001$ ($Z_4 = 1$ when $X_4 = 1$)
  - Otherwise $Z_i$ stays ***0***
- Identify the **Inputs** and **Outputs** and draw a **Block Diagram (BD)**:



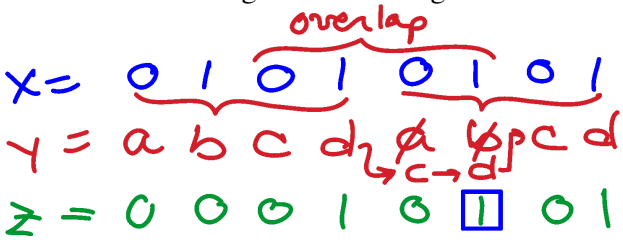

- Formulate a **State Diagram (SD)**
  - Right arrows show two possible correct sequences:



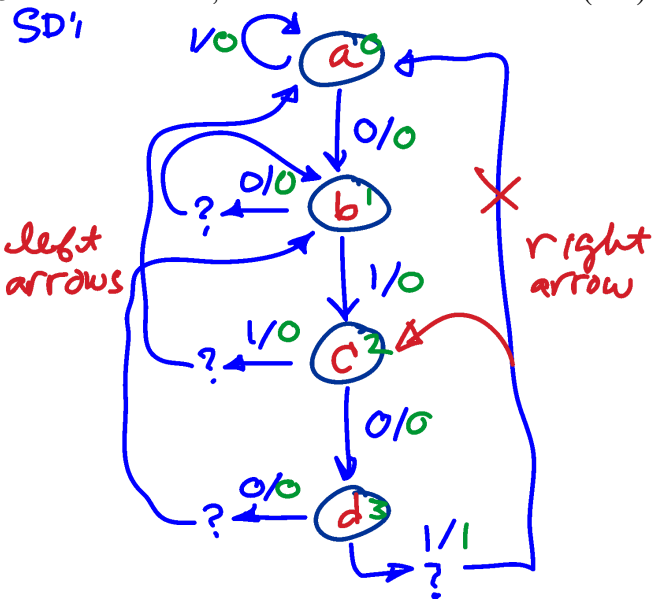

  - State trace following solid right arrow path back to state **a**:



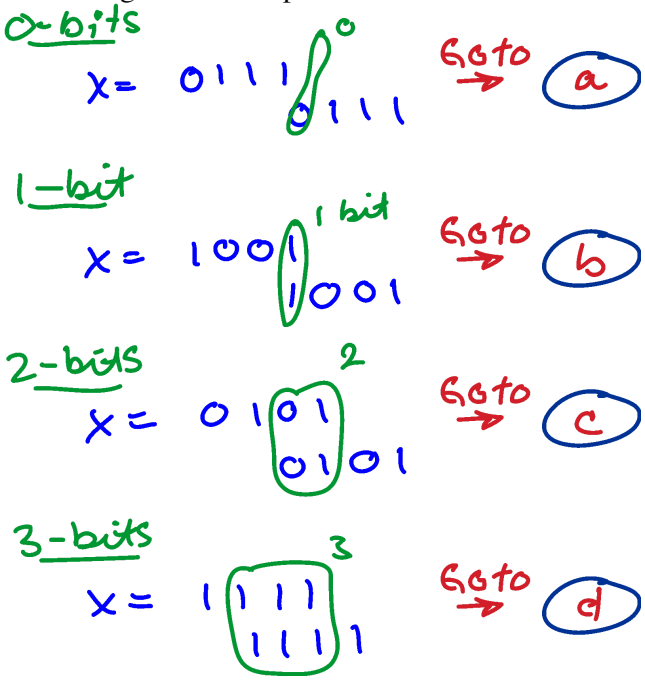

  - State trace following dashed-line right arrow to state **c**, allowing an overlapping sequence:



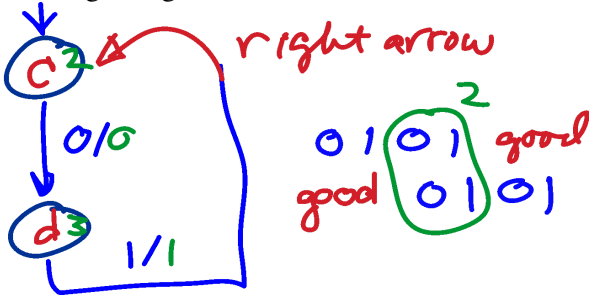
- On the other hand, left arrows indicate incorrect (bad) sequences, and where they should go:




- Where the right and left arrows are routed to is determined by the number of overlapping bits. Here are some general examples:

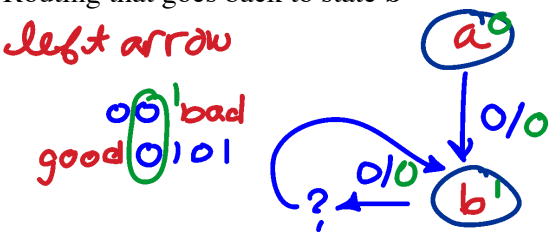



- More specifically, for the right arrow of ***Example #3***, overlapping the ***X = 0101*** good sequence with itself yields
  - 2 bits of overlap
  - Routing that goes to state **c**

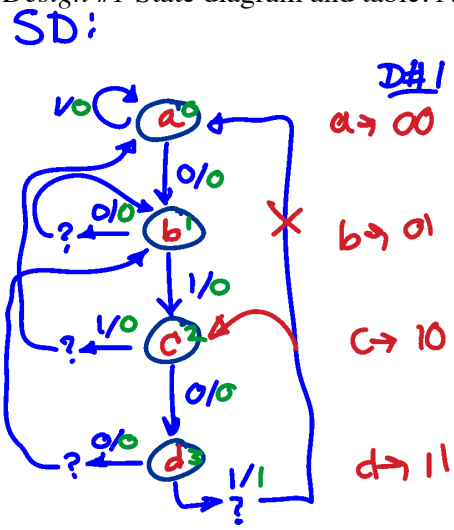



- Also, overlapping the bad sequence ***X = 00*** (from state **a** and out of state **b**) with the good sequence ***X = 0101*** yields:
  - 1 bit of overlap

- Routing that goes back to state **b**

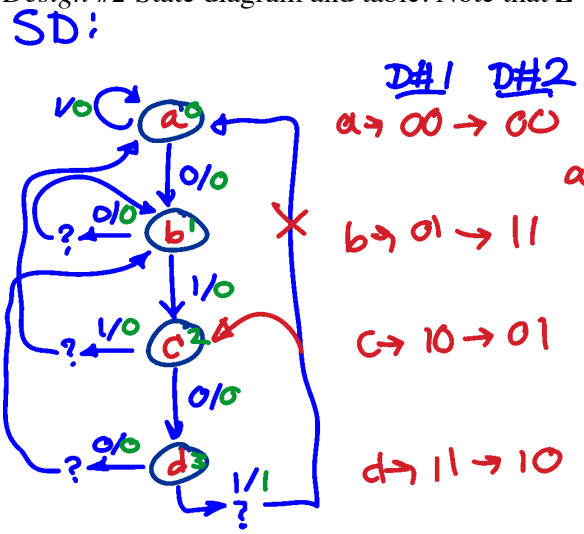


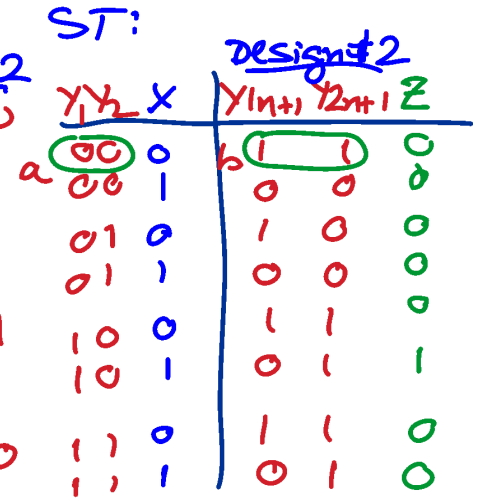

2. Construct **State Tables (ST)**: Lets make two designs based upon different state assignments.
   - *Design #1* State diagram and table. Note that ***Z= XY1Y2***



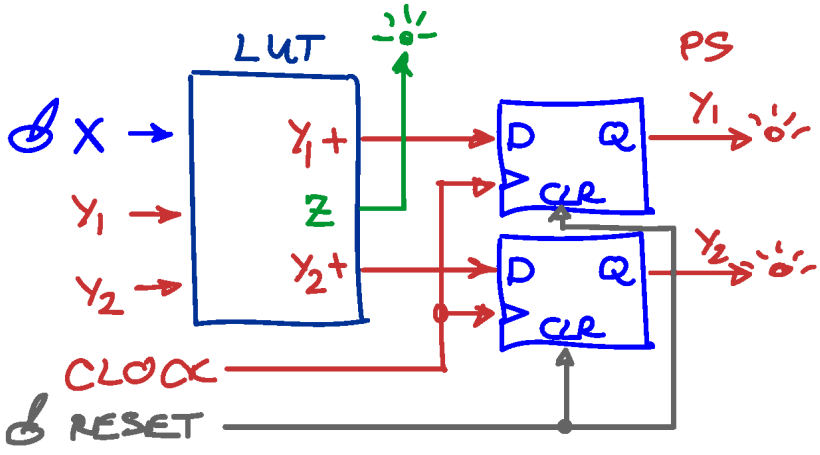


   - *Design #2* State diagram and table. Note that ***Z= XY1Y2'***




3. Program the state table into a lookup table. Feed the next state outputs to two D flip-flops.



**Quiz #9 & selected solutions**

## *REGISTERS AND COUNTERS*

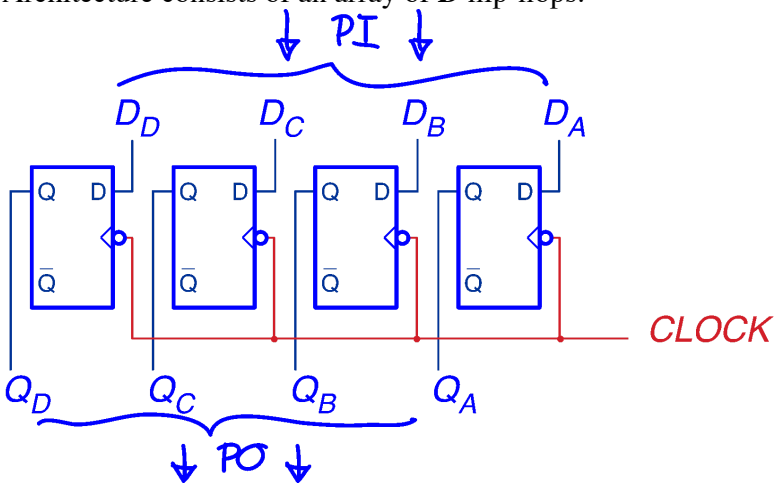
**Register defined**: a SLC which deals with the storage and transfer of multi-bit data.

### Data Transfer Modes

- PARALLEL-IN/PARALLEL-OUT = ***PIPO***
- SERIAL-IN/SERIAL-OUT = ***SISO***
- PARALLEL-IN/SERIAL-OUT = ***PISO***
- SERIAL-IN/PARALLEL-OUT = ***SIPO***

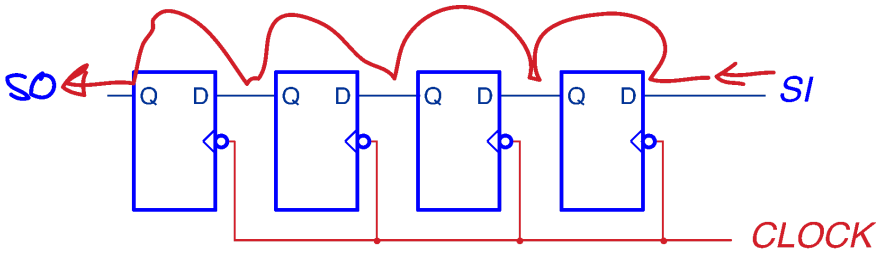
### Data Latch Register

- Performs ***PIPO*** data transfers.
- Architecture consists of an array of **D** flip-flops:

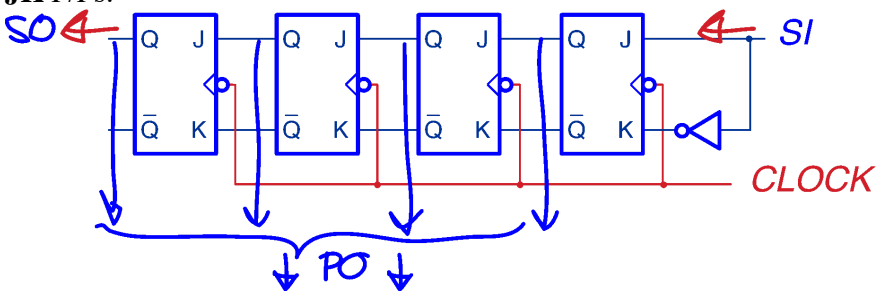



### Shift Register

- Performs primarily ***SISO*** data transfers.
- Architecture consists of an array of flip-flops connecting inputs and outputs of adjacent F/Fs.
  - **D** F/Fs:

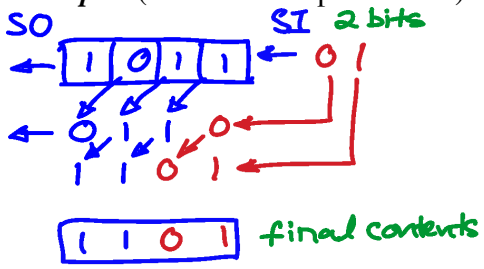


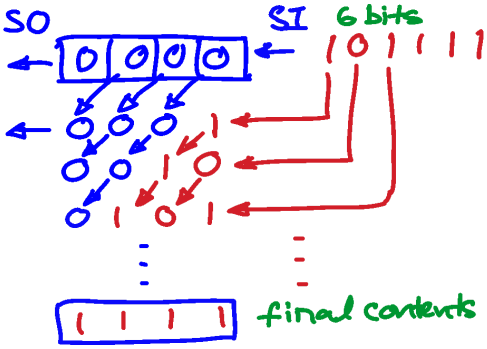

  - **JK** F/Fs:



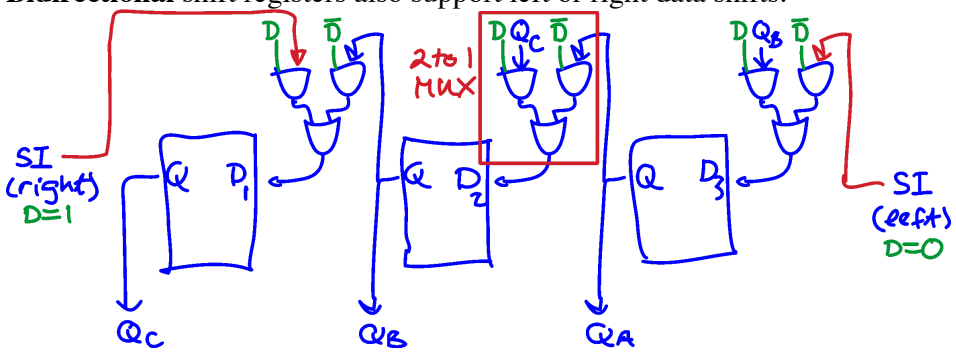

- **Multi-mode** shift registers in addition support ***SIPO***, ***PISO***, or even ***PIPO*** data transfers

- ***Examples*** (2 and 6 clock pulse shifts):




- **Bidirectional** shift registers also support left or right data shifts:




- A bidirectional shift register multiple additional modes of operation is known as a **multifunction** register.

**Counter defined**: a SLC which deals with the generation of multi-bit sequential codes.

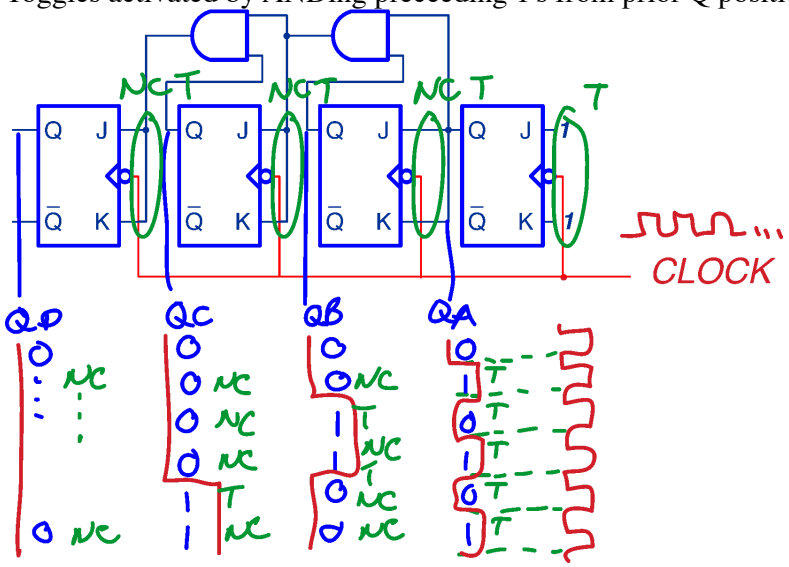
**Counter Circuit Characteristics**

- Single or multi-mode operation (up-down, for example)
- Synchronous or asynchronous operation
- Outputs provide **frequency division** of input clock signal
- Zero reset for termination of count sequence

**Single-Mode Synchronous Counter**

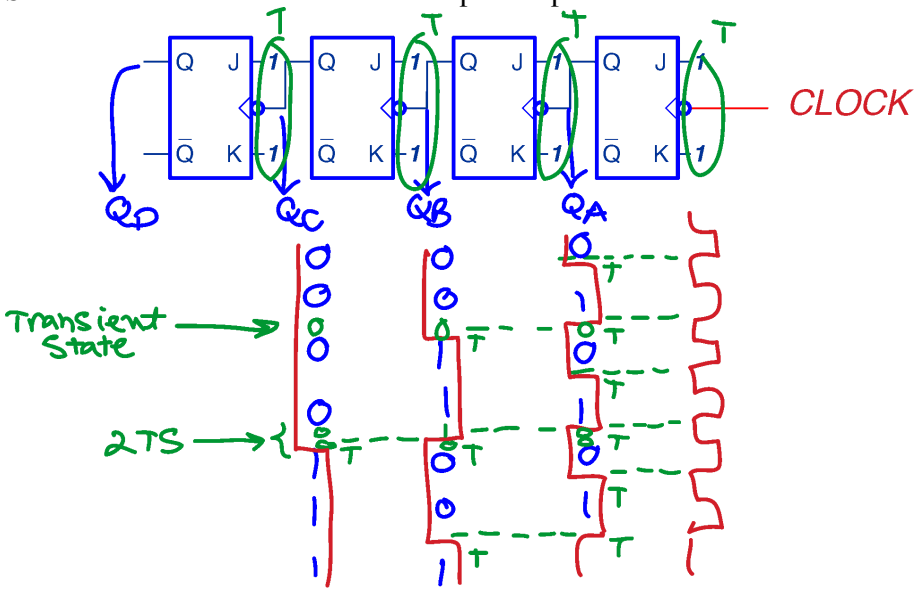
- A synchronous SLC which counts in ascending order.
- Architecture consists of an array of **JK** F/Fs
  - Driven by a common clock line
  - Configured to operate in the **Toggle (T)** or **No Change** (**NC**) modes

- Toggles activated by ANDing preceeding 1's from prior Q positions

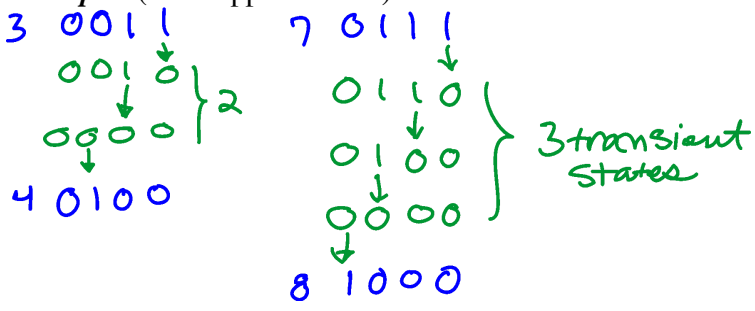



**Ripple Counter**

- An **asynchronous** SLC which counts in ascending order.
- Architecture consists of
  - An array of **JK** F/Fs configured entirely in **Toggle (T)** mode
  - Clock applied to first F/F only
  - Successive F/Fs are clocked from outputs of prior F/Fs:




- **Ripple effect**:
  - Output changes ripple from right to left F/F because...
  - Clocks (red) are slightly delayed between F/F stages
  - Transient states (green) appear between stable states (blue)
- ***Examples*** (4-bit ripple counter):



**Quiz #10 & selected solutions**