

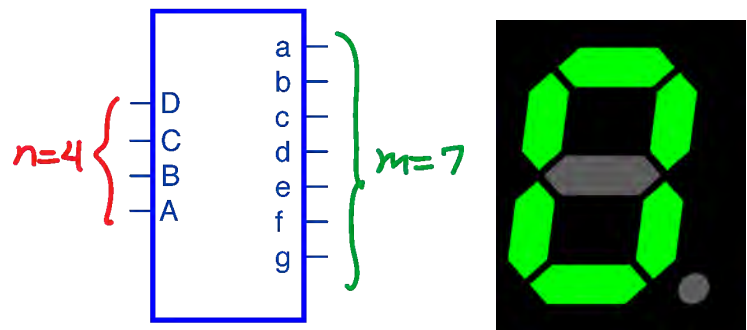
Lecture Topics:

- [Decoders, MUXes, Encoders and DeMUXes](#)
- [Read-Only Memories](#)
- [Random Access Memories](#)
- [Smartphones](#)
- [Latches and Flip-Flops](#)
- [State Machine Design](#)
- [Registers and Counters](#)

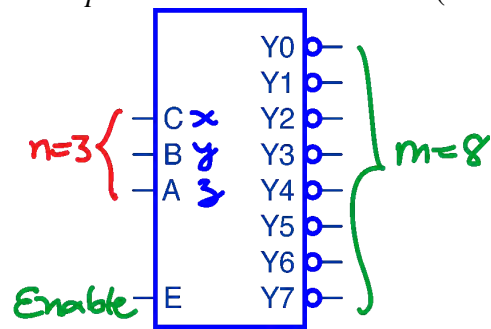
DECODERS, MUXES, ENCODERS and DEMUXES

Decoders

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

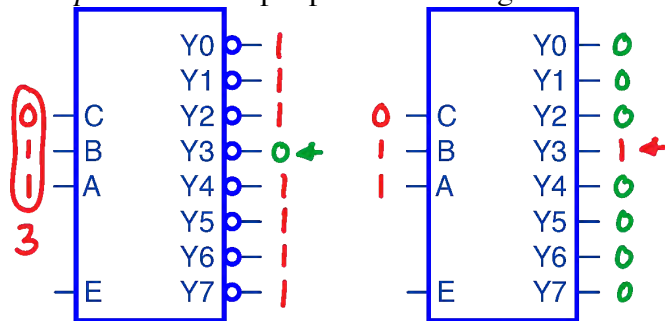


- Binary decoders ($m = 2^n$), are designed to *select an output position*
 - Example: 74138 3-to-8 Decoder ($n = 3, m = 8$)



- 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*

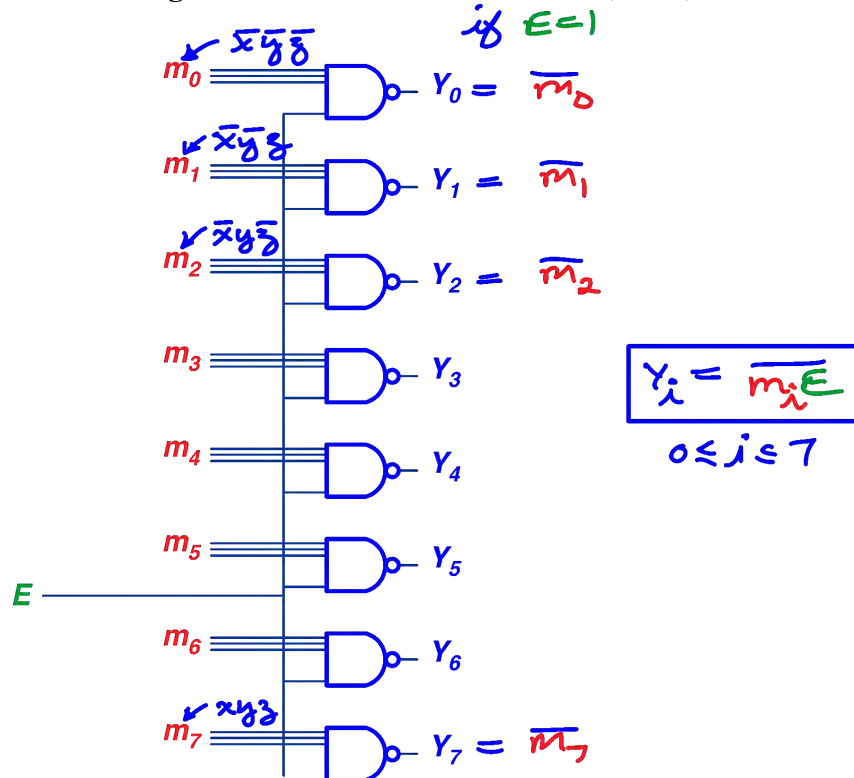
- Example: Select output position 3 using active-low & active high 3-to-8 decoders



Active Low

Active High

- 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

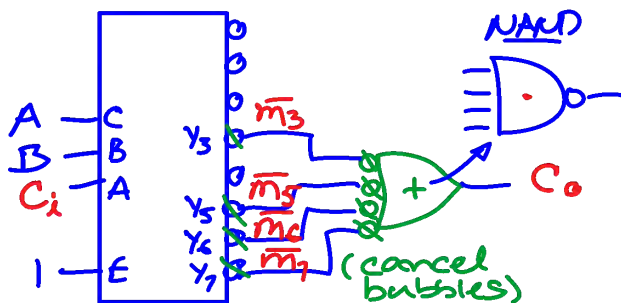


- Active-low decoder output equations:

$$Y_i = (m_i E)', 0 \leq i \leq 2^n - 1$$

- 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

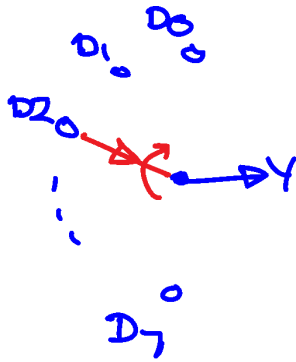
$$C_o = \sum m(3, 5, 6, 7)$$



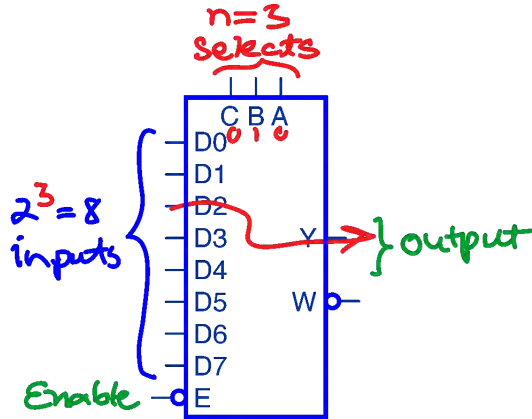
Multiplexers (MUXes)

- MUXs connect one of 2^n data input lines to one output line

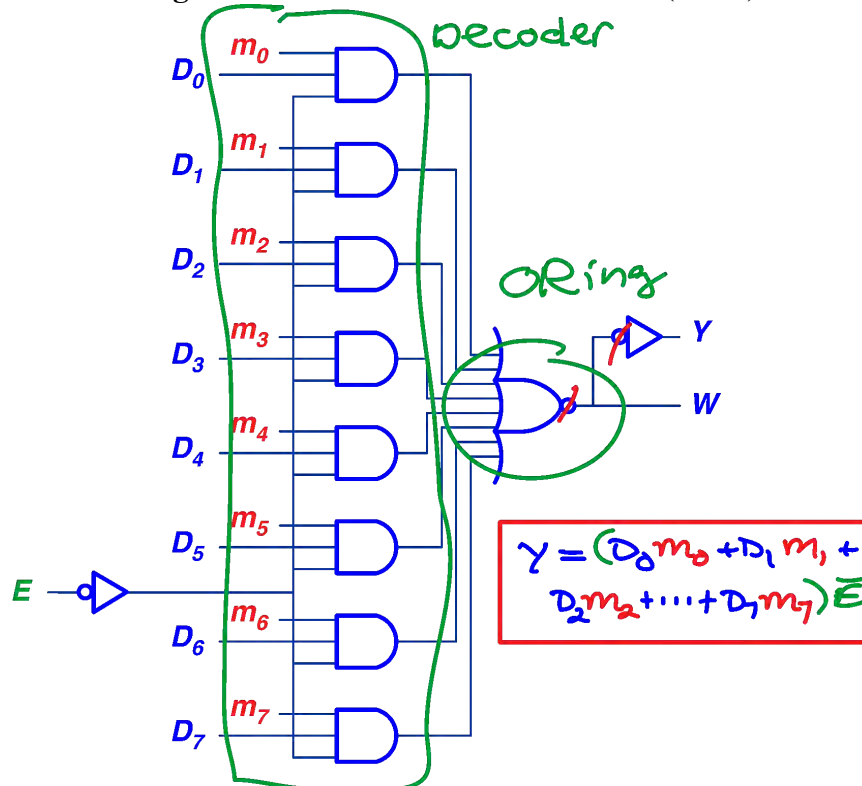
- Mechanical switch model:



- 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

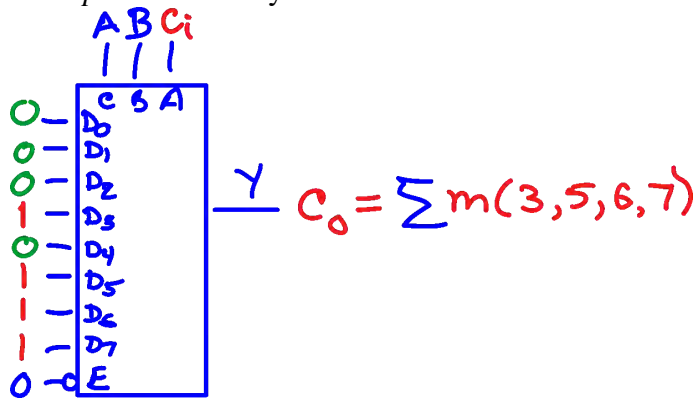


- Data lines D_i are ANDed with decoder minterms to provide selection of desired minterms.
- MUX output equation (for an active-low E) is:

$$Y_i = (D_0m_0 + D_1m_1 + D_2m_2 + \dots + D_7m_7)E'$$

- MUXs may also be employed as a SOP Boolean function generator of the input select lines

- Example: More carry-out

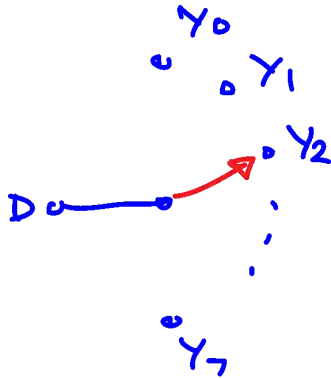


Encoders

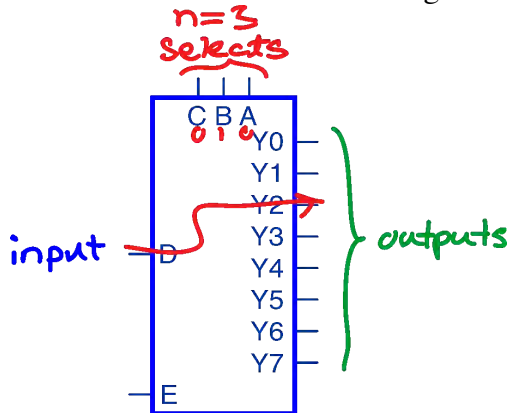
- Translates an n bit dataword into a smaller m -bit codeword, where $n \leq 2^m$.
- Architecture consists of a linear array of m ORing elements. (Not shown)

Demultiplexers (DeMUXes)

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

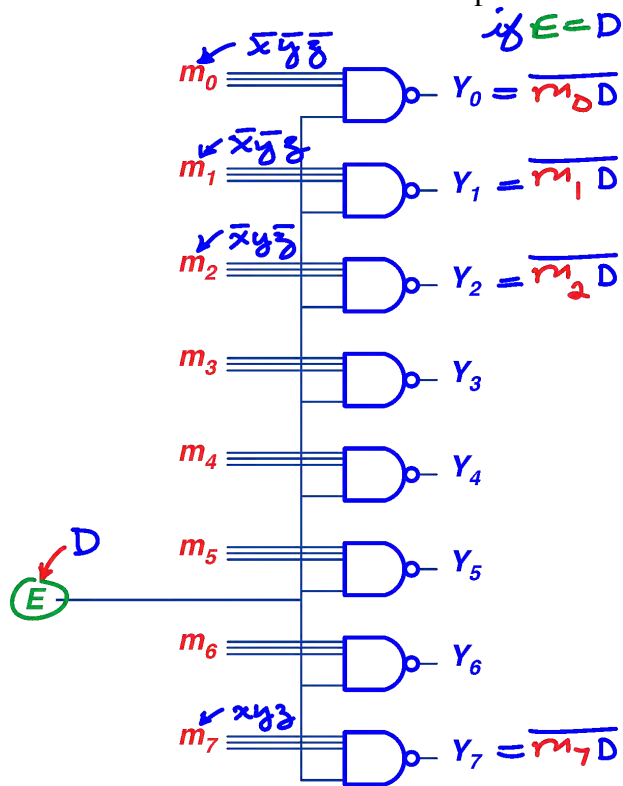


- A 1-to-8 MUX with an active-high enable



- The output line selected for connection is determined by a set of n output select lines, i.e. C, B, A .
- A DeMUX gate architecture 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 :



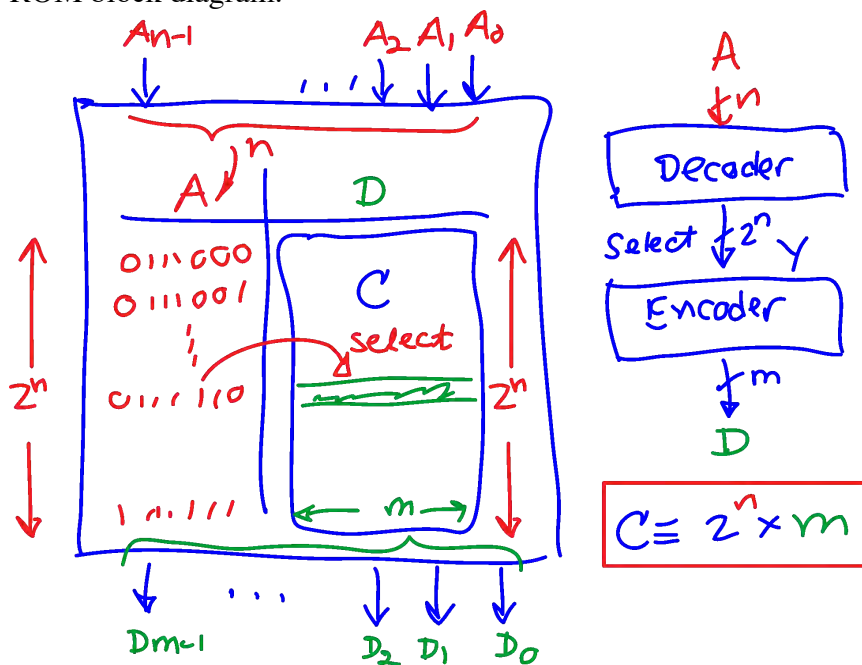
- The DeMUX output equation is: $Y_i = (m_i D)', 0 \leq i \leq 7$

Quiz #6 & selected solutions

READ-ONLY MEMORIES

ROM Basics

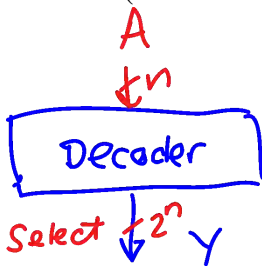
- A **ROM** is a programmable truth table consisting of a
 - n -bit address input (selects a word of memory)
 - m -bit data output (the data bits that fit into one word)
- ROM block diagram:



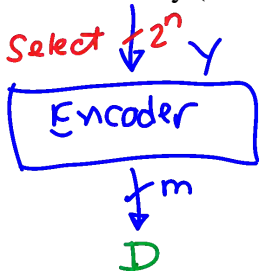
- ROM capacity is: $C = 2^n \times m$ (bit contents of the ROM)

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)



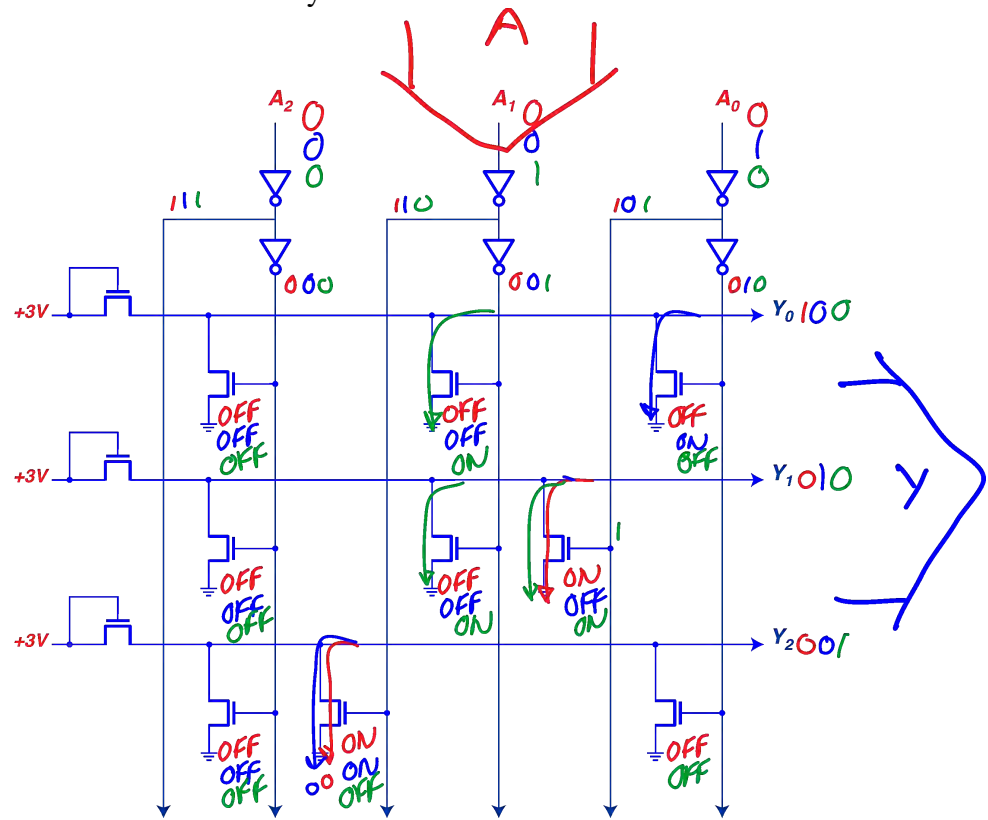
- Encoding Techniques
 - Diode-fuse array
computer.howstuffworks.com/rom3.htm
 - MOS transistor-fuse array
 - Double gate MOS transistor array

ROM Transistor Circuitry

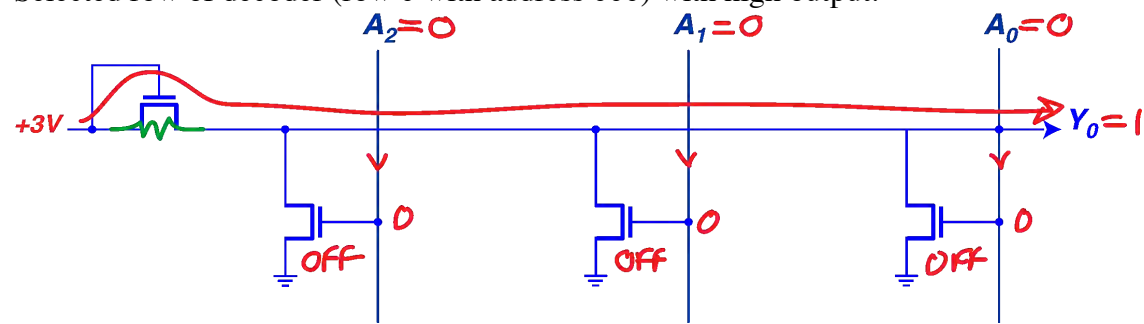
Three examples:

- **PROM**

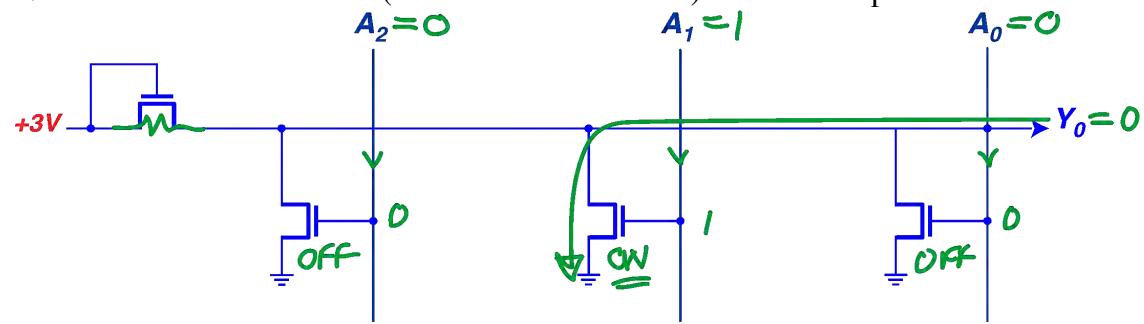
- Decoder transistor array:



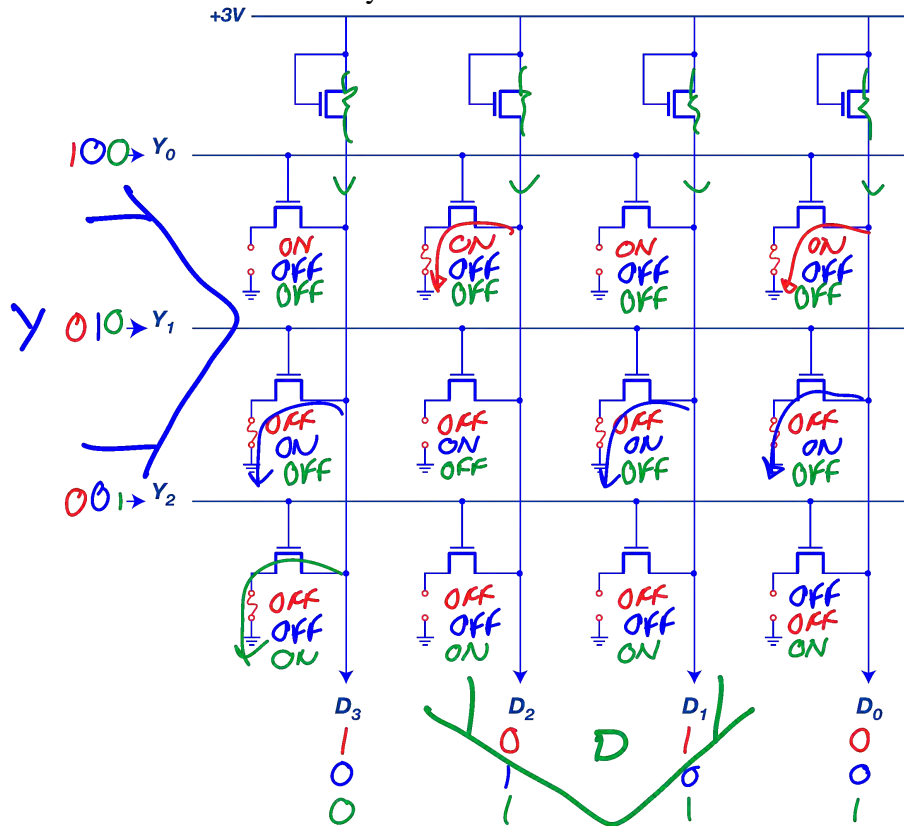
- Selected row of decoder (row 0 with address 000) with high output:



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

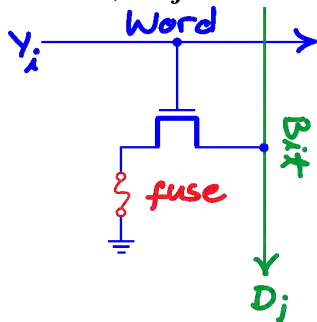


- Encoder transistor-fuse array:

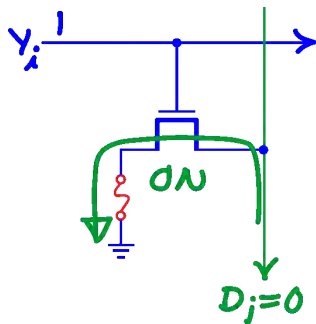


- PROM encoder cell, showing i^{th} word & j^{th} bit lines plus fuse:

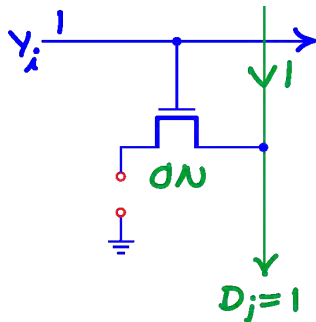
$$0 \leq i \leq 2^n - 1, 0 \leq j \leq m - 1$$



- An intact fuse connects a 0:

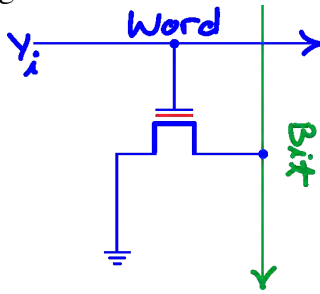


- A blown fuse connects a 1:

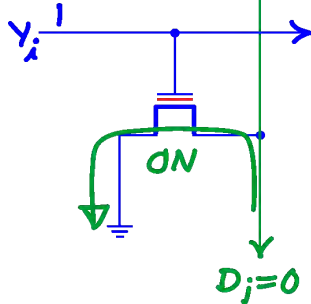


• EPROM

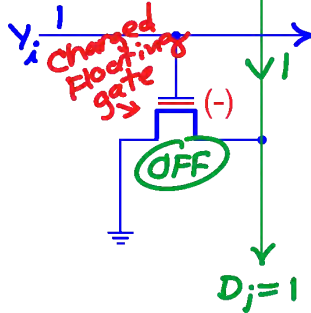
- Decoder transistor array is used, as before.
- Encoder transistors employ a floating gate to connect outputs:
 - **EPROM** encoder cell, showing i^{th} word and j^{th} bit lines plus a transistor having a chargeable red floating gate:



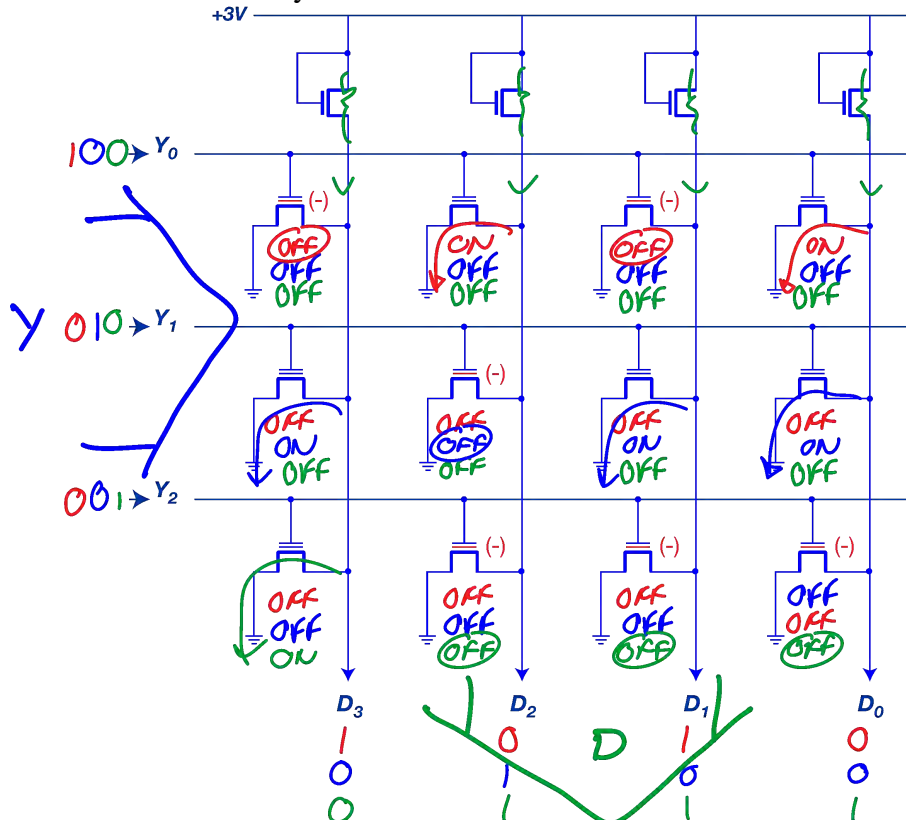
- Cell transistor (uncharged) turns ON, connects a 0



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



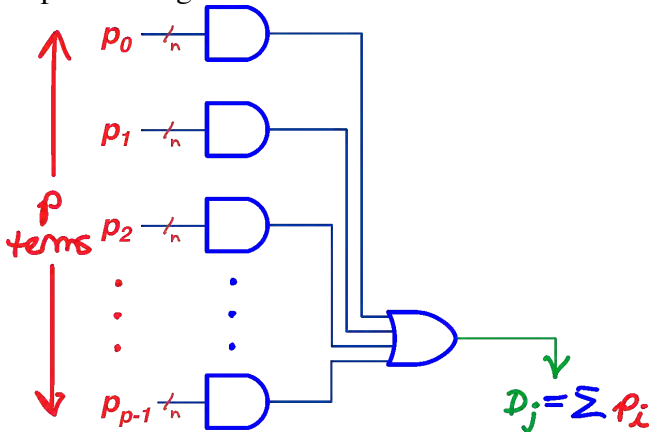
- EPROM Encoder array:



- **Flash** memories: [from wikipedia.com](https://en.wikipedia.org/wiki/Flash_memory)
 - 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.

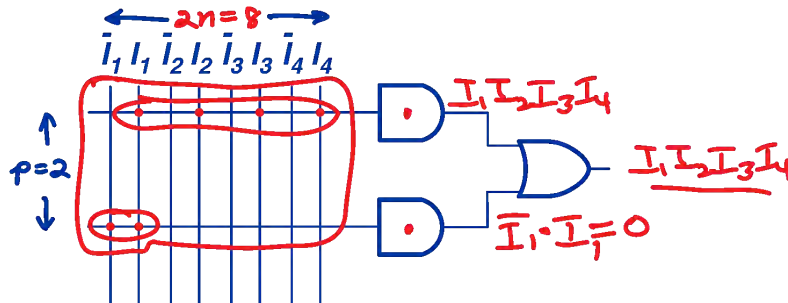
Programmable Gate Arrays (PGAs)

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

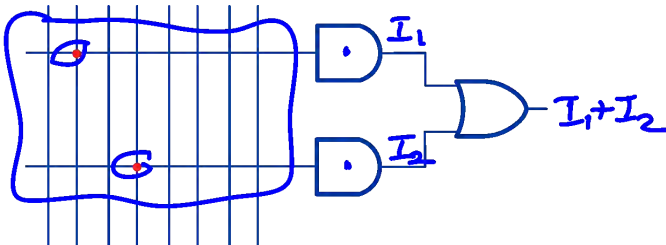


- Traditional **PGAs** are built using **AND-arrays**
 - Each **PGA** will sum p products.
 - Each p -term is formed from an **AND-array** having $2*n$ input lines
 - An **AND-array** has size $p \times 2*n$.
 - A **PGA** has m **AND** arrays which implement m logic functions.
- A **PLD** is a P-logic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

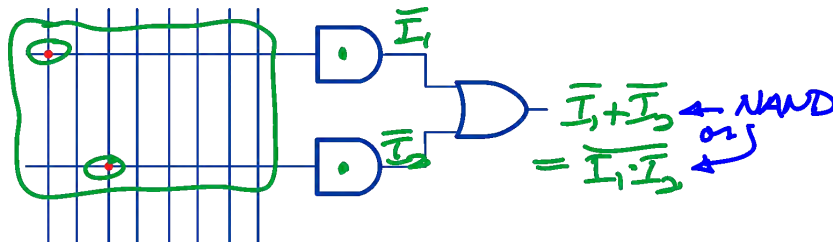
- **AND** function:



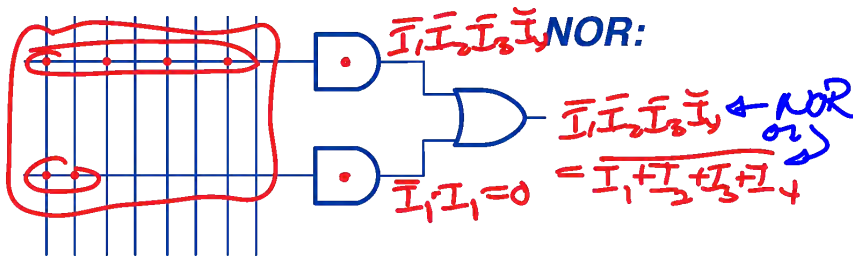
- **OR** function:



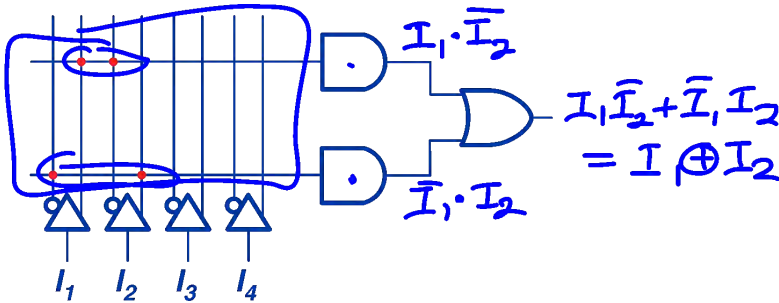
- **NAND** function:



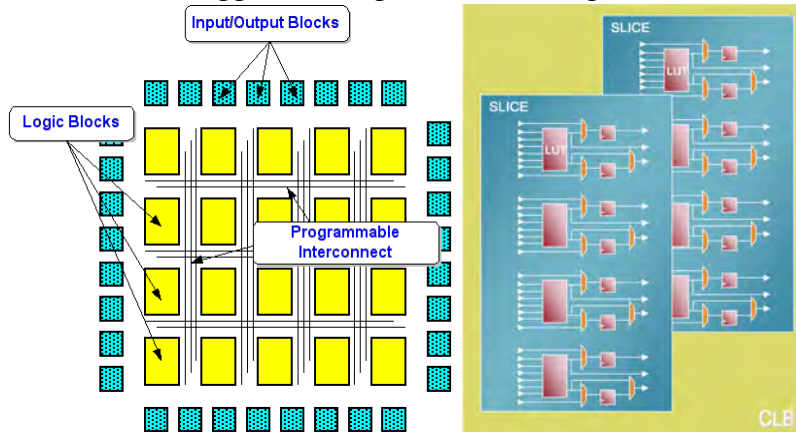
- **NOR** function:



- **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!



Graphics by Xilinx

- Advantages of **FPGAs**:
 - 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_\(data\)](http://en.wikipedia.org/wiki/Orders_of_magnitude_(data)).
- Xbox Series X vs Playstation 5: Both AMD Zen 2 CPU, $m = 256$ (32 B) and <https://www.tomsguide.com/news/ps5-vs-xbox-series-x>
- Capacity
 - Formulas:
 - $C = 2^n \times m \text{ b}$ (in bits)
 - $C = 2^n \text{ B}$ (in Bytes if $m = 8$)
 - $C = 2^n \times m/8 \text{ B}$ (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:
 - $C = (2^n \times m/8)/2^{10} \rightarrow C \text{ KB}$ (in Kilo Bytes)
 - $C = (2^n \times m/8)/2^{20} \rightarrow C \text{ MB}$ (in Mega Bytes)
 - $C = (2^n \times m/8)/2^{30} \rightarrow C \text{ GB}$ (in Giga Bytes)
 - $C = (2^n \times m/8)/2^{40} \rightarrow C \text{ TB}$ (in Tera Bytes)
 - Examples:
 - $n = 14, m = 8$

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

$$\text{or } C = 2^n = 2^{14} \text{ Bytes} = \frac{2^{14}}{2^{10}} \text{ KB} \\ = 2^4 \text{ KB} \\ = 16 \text{ KB}$$

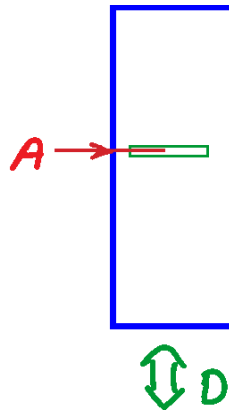
- $n = 26, m = 8$

$$C = 2^n \text{ B} \\ = 2^{26} \text{ B} = \frac{2^{26}}{2^{20}} \text{ MB} \\ = 2^6 \text{ MB} \\ = 64 \text{ MB}$$

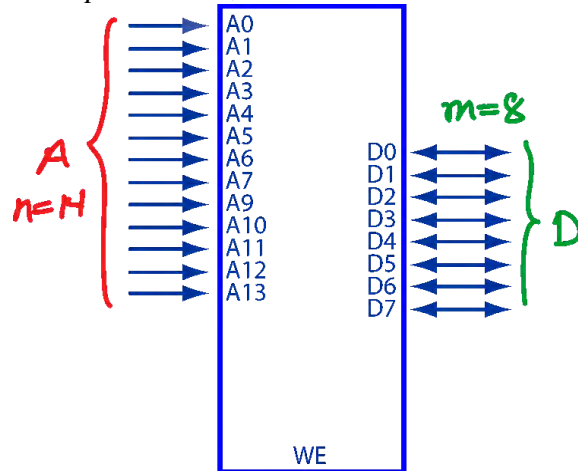
RAM Organization and Type

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

- Row decoder selects m -bit word

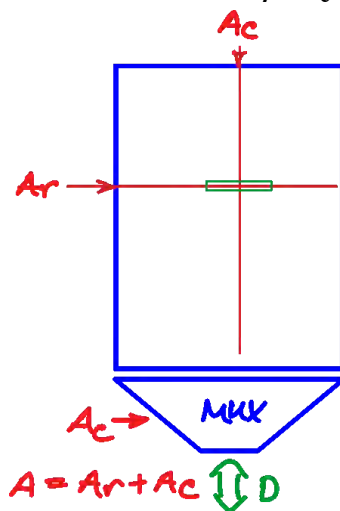


- Example: $n = 14$, $m = 8$: $C = 16$ KB

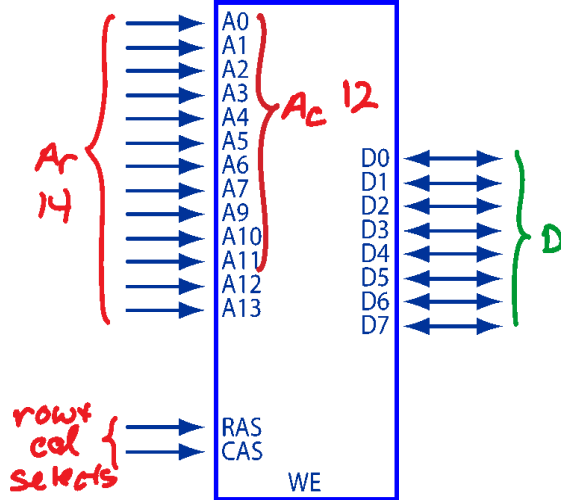


◦ **2D decoding** (row decoders and column MUX)

- n_r -bit row decoding
- Reuse $n_c \leq n_r$ for column decoding
- Address size: $n = n_r + n_c$

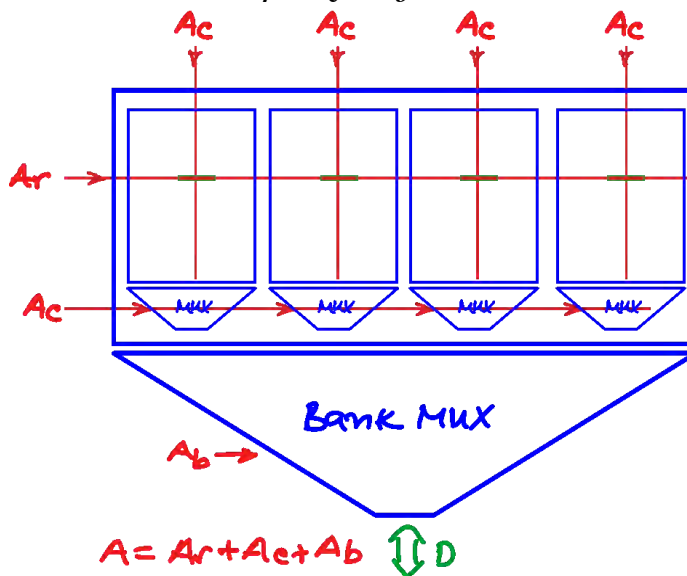


- Example: $n_r = 14$, $n_c = 12$, $m = 8$: $C = 64$ MB

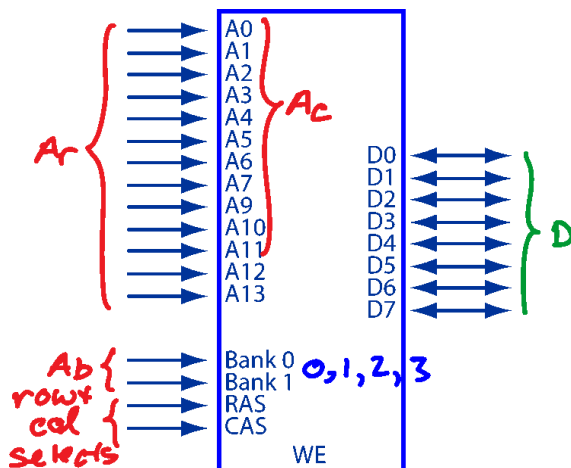


◦ **2D decoding with bank organization**

- 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:

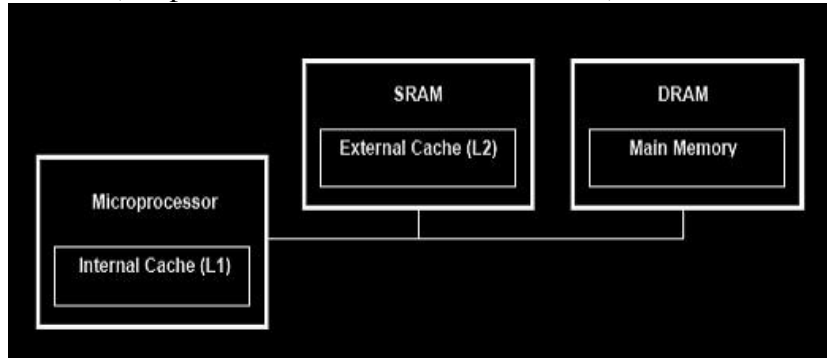
computer.howstuffworks.com/ram4.htm

- 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)



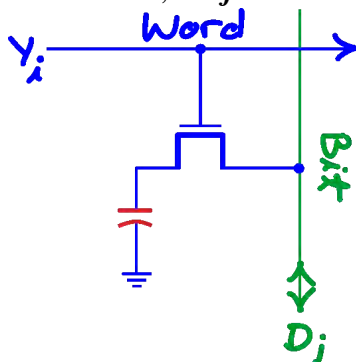
- Dynamic Memory
 - Synchronous Dynamic (SD) (self-refreshing)
 - Double Data Rate (DDR) SD (faster)
 - DDR2 = "2x more"
 - DDR4 = "4x more"
 - LPDDR4 = Low Power "4x more"
 - Cell organization
 - Single transistor.
 - Storage provided by capacitor, which leaks and requires refreshing by CPU.



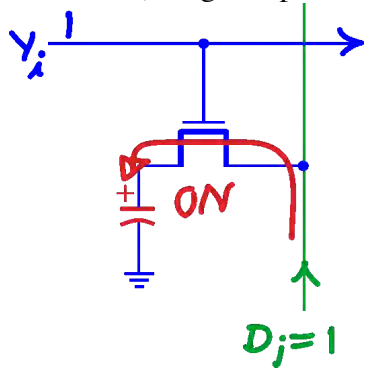
Graphics by quora.com

- DRAM cell showing i^{th} word and j^{th} bit lines plus capacitor:

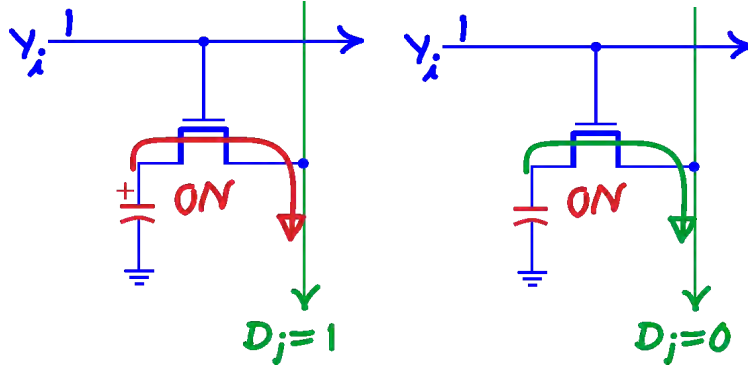
$$0 \leq i \leq 2^n - 1, 0 \leq j \leq m - 1$$



- Write a 1 (charges capacitor):

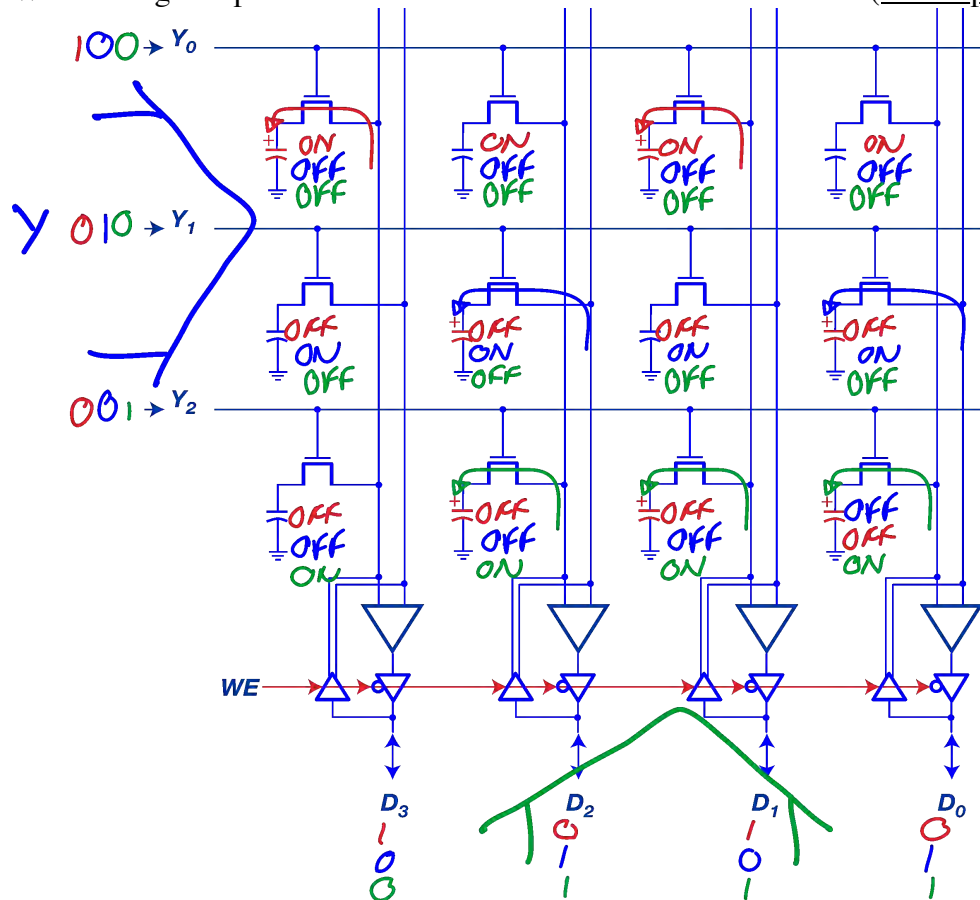


- Reading a 1 or 0:

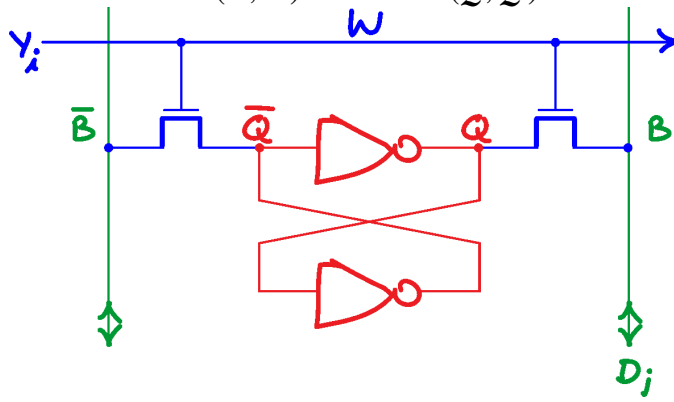


Dynamic RAM (DRAM) layout example

- Decoder transistor array, as before.
- Encoder transistor array
 - Two data lines for each cell
 - Read: senses presence 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)



- Cell access provided by transistors 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



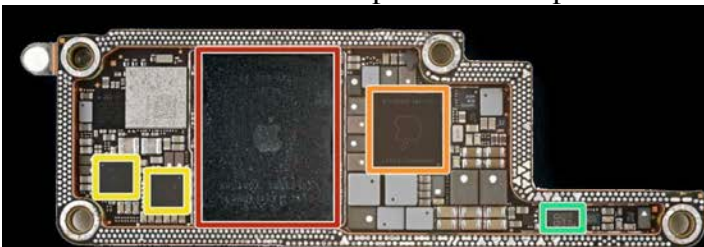
iPhone design

- iPhone 16 Pro Max inside view
<https://www.youtube.com/watch?v=PHQD4FcsOvs>

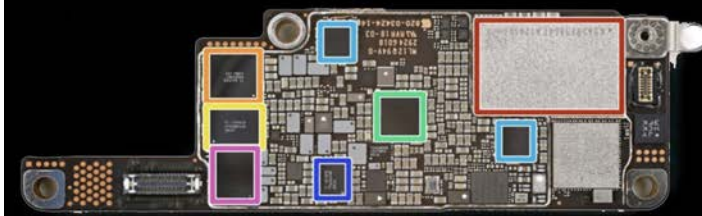


ifixit.com

- iPhone 16 Pro Main board top and back <https://www.ifixit.com/Guide/iPhone+16+Pro+Chip+ID/177358>

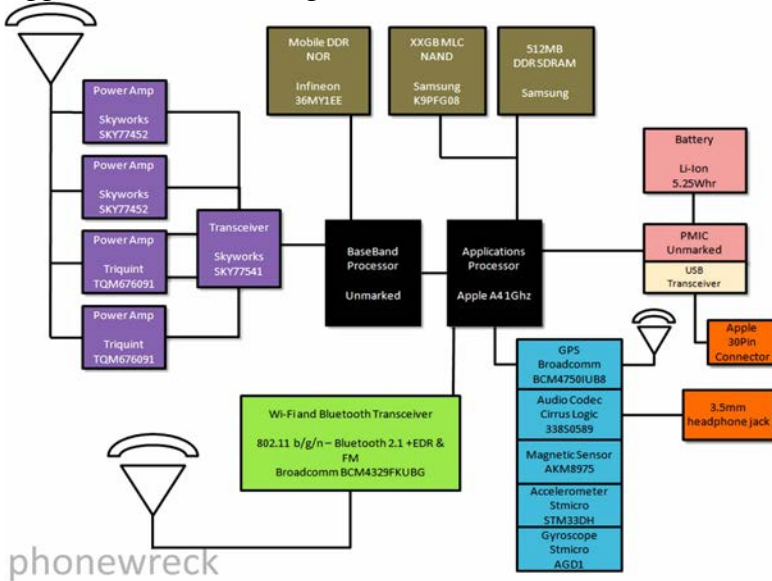


- Micron MT62F1G64D4AQ-031 XT:C LPDDR5X SDRAM layered over an Apple APL1V07 / 339S01527 A18 Pro hexa-core applications processor w/ GPU & Neural Engine
- Apple APL109A / 338S01119 power management
- STMicroelectronics STPMIA3C power management
- GigaDevice GD25Q80E 1 MB serial NOR flash memory



- Kioxia K5A3RF9864 128 GB NAND flash memory
- Likely Texas Instruments SN2012027 USB type-C controller
- Likely Texas Instruments CP3200B1G0 battery charger
- Cirrus Logic 338S00967 audio codec
- Cirrus Logic 338S01087 audio amplifier
- Texas Instruments TPS65657B0 display power supply
- Likely Cirrus Logic 338S00843 power management

• Apple iPhone block diagram (7/2010)



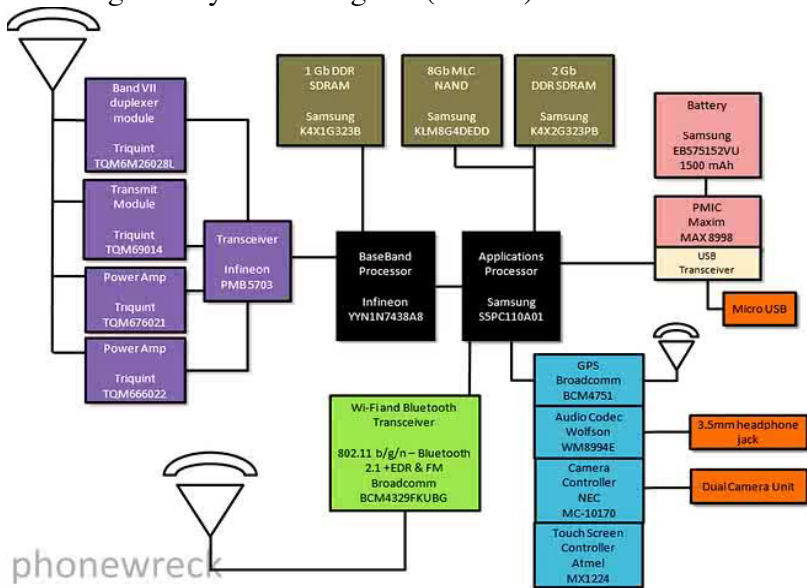
• iPhone Block Diagram Components

- 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

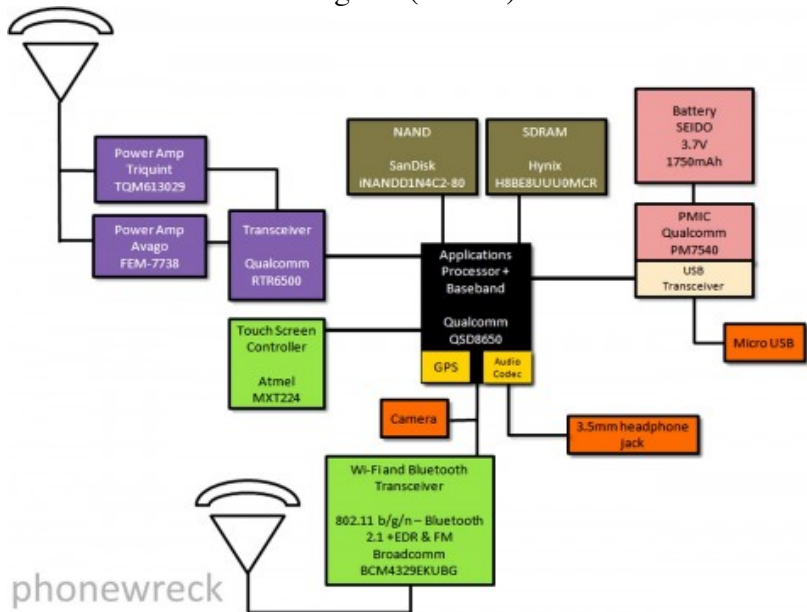


Android design

- Samsung Galaxy block diagram (9/2010):



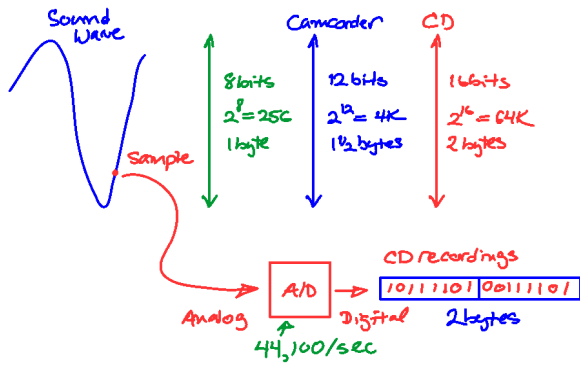
- HTC Incredible block diagram (1/2011)



Storing and Playing Music

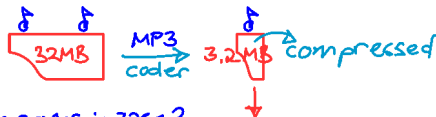
- Example: Storing Music
 - Music is sampled at 44,100/sec using 2B for each sample
 - Thus a three minute song file size is 32MB.
 - This song is then compressed to 3.2 MB to store in flash
 - 32GB of flash can store 10,000 3 minute songs
 - The song is decompressed to RAM where it is played as a 32MB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

Music:



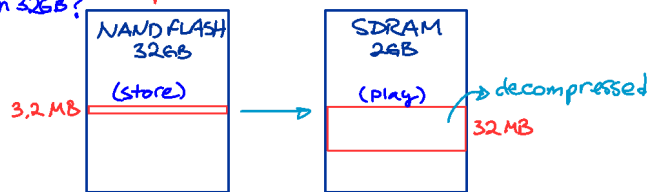
Example A/D:

$$44,100/\text{sec} \times 2B \times 3(60\text{sec}) \times 2\text{ch} \approx 32\text{MB/song.}$$



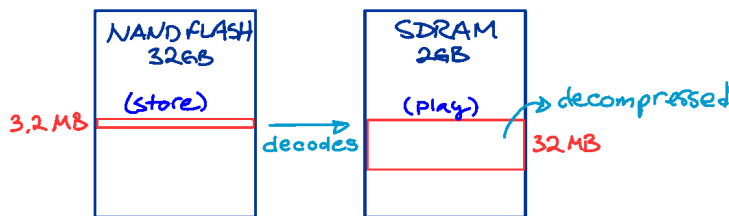
How many stereo songs in 32GB?

$$N = \frac{32GB}{3.2MB} = 10K$$



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-Codec decodes the song back to 32 MB and is played in real time or is stored in RAM



Streaming Rate?

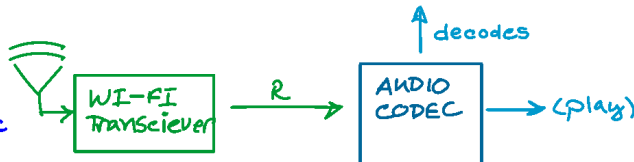
$$R = \frac{D}{T} = \frac{3.2MB}{3\text{min}} = \frac{3.2MB}{180\text{sec}}$$

$$R = .0178\frac{MB}{s}$$

$$R = 17.8\frac{KB}{s}$$

$$= (17.8) 8\frac{Kb}{s}$$

$$= 142.2\frac{Kb}{s}$$



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

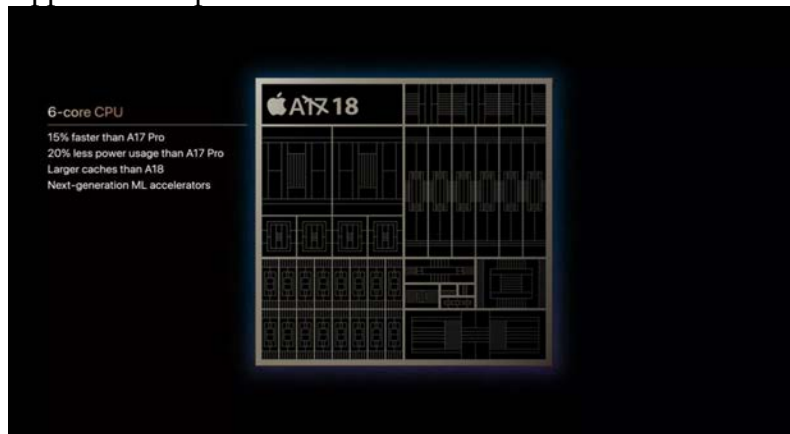


Image: MacWorld

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 Kryo SoC (Cortex X1 1x 2.84/Kryo 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
 - Mali 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
 - 32 MB SRAM Cache
 - 6 GB LPDDR5X SDRAM
- Samsung Galaxy S22 (2022)
 - Qualcomm Snapdragon 888 Kryo 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
 - 32 MB SRAM Cache
 - 6 GB LPDDR5X SDRAM
- iPhone 15 Pro (2023)
 - 64-bit A17 Bionic SoC = ARMv14 architecture (2x 3.78/4x 2.11 GHz HP/EE cores) plus an
 - Apple GPU (6 cores)
 - 16 core neural engine
 - 32 MB SRAM Cache
 - 8 GB LPDDR5X SDRAM

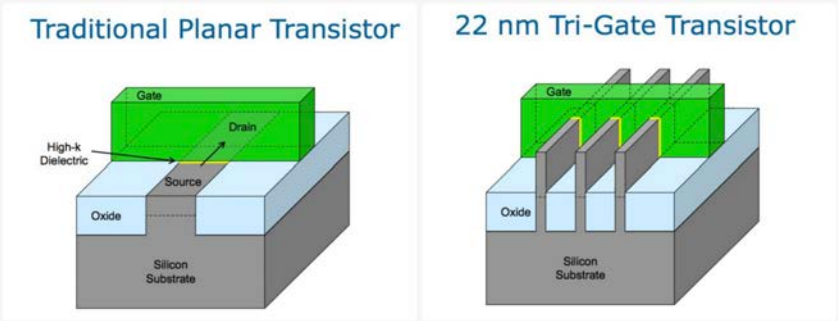
- Samsung Galaxy S23 (2023)
 - Qualcomm 8550 AC Snapdragon 8 Gen 2 SoC Octacore (1x3.36 GHz Cortex-X3 & 2x2.8 GHz Cortex-A715 & 2x2.8 GHz Cortex-A710 & 3x2.0 GHz Cortex-A510 GHz), plus a
 - Adreno 740 8-core GPU (??? MHz)
 - 8 GB LPDDR5 RAM
- Google Pixel 8 Pro (2023)
 - Google Tensor G3 SoC (Nona-core (1x3.0 GHz Cortex-X3 & 4x2.45 GHz Cortex-A715 & 4x2.15 GHz Cortex-A510), plus a
 - Immortalis-G715s MC10 8-core GPU (??? MHz)
 - Google EdgeTPU NPU
 - 8 GB LPDDR5X RAM

Recent performance enhancements

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

Apple SoC Evolution			
	Die Size	Transistors	Process
A5	122mm ²	<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)
- 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

- Apple Watch Series 1-10 (2018-24)

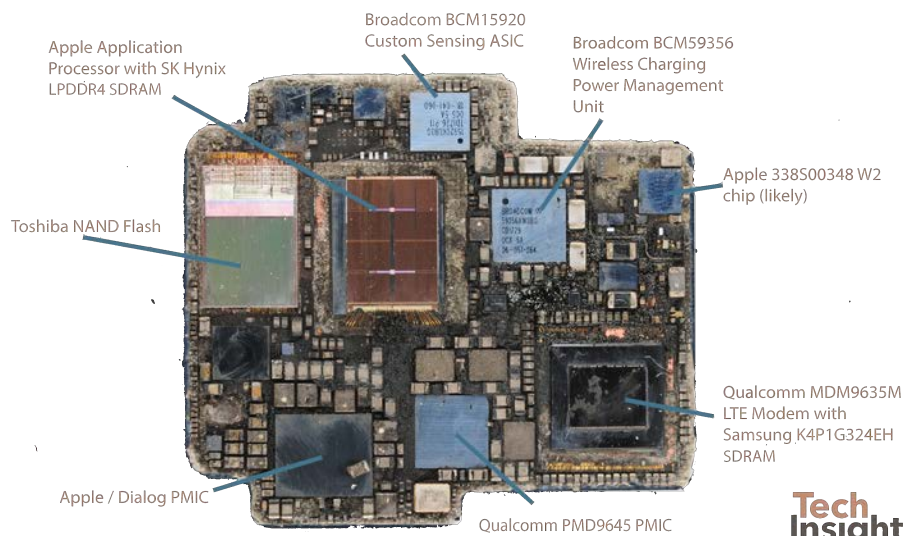


- S6 Teardown:



(Image: ifixit.com)

- S3 SoC, encased in resin



Tech
Insights

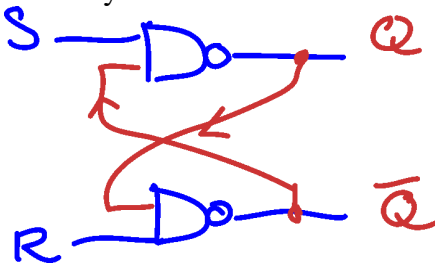
- Apple Watch hardware Series 3

- 390 x 312 resolution display
 - S3 SoC ARM Cortex A9 (Dual Core, 800 MHz) plus a
 - 8 (16/S4) GB NAND
 - PowerVR GX??? GPU
 - 3D Touch Display and Taptic Engine
 - 768 MB LPDDR4 SDRAM
- Apple Watch Series 4-7 hardware unknown except they have
 - 448 x 368 resolution display
 - SiP 64 bit dual core processor
- Compare S3 with iPhone 4 (2010):
 - A4 SoC = ARM Cortex A8 (800MHz) plus an
 - Imag Tech PowerVR SGX 535 GPU
 - 512 MB RAM

LATCHES AND FLIP-FLOPS

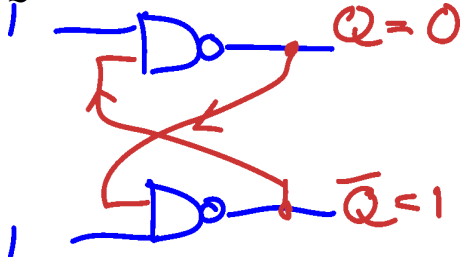
Simple Latch Circuit

- 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*.

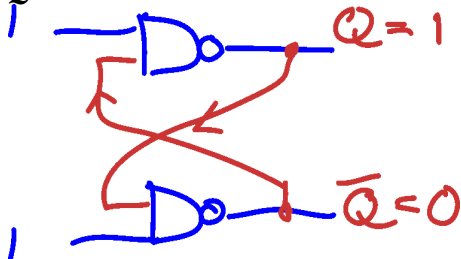


- Latch **state** is specified by the latch outputs;

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

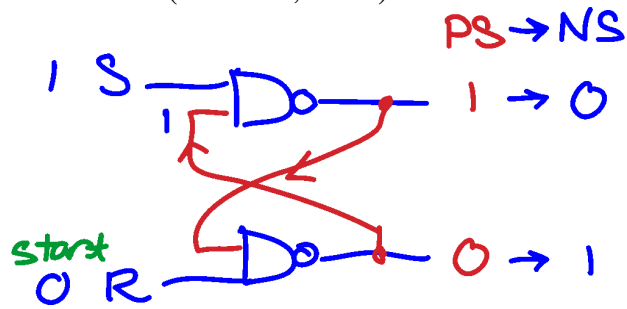


- $Q = 1$ indicates the latch is **Set**

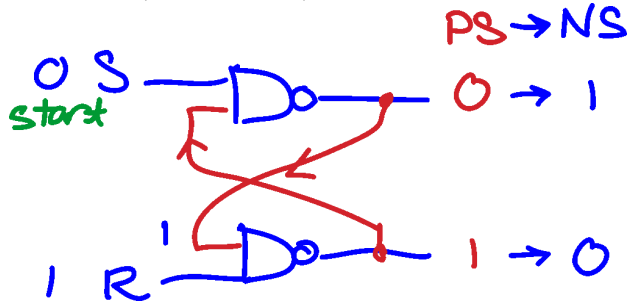


- There are **three modes** of operation (changing a **Present State** to a **Next State**):

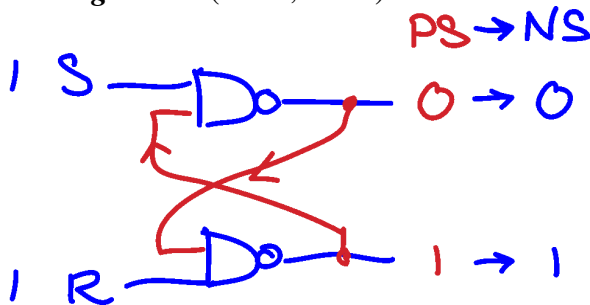
- **Reset mode** (for $S = 1, R = 0$)



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



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

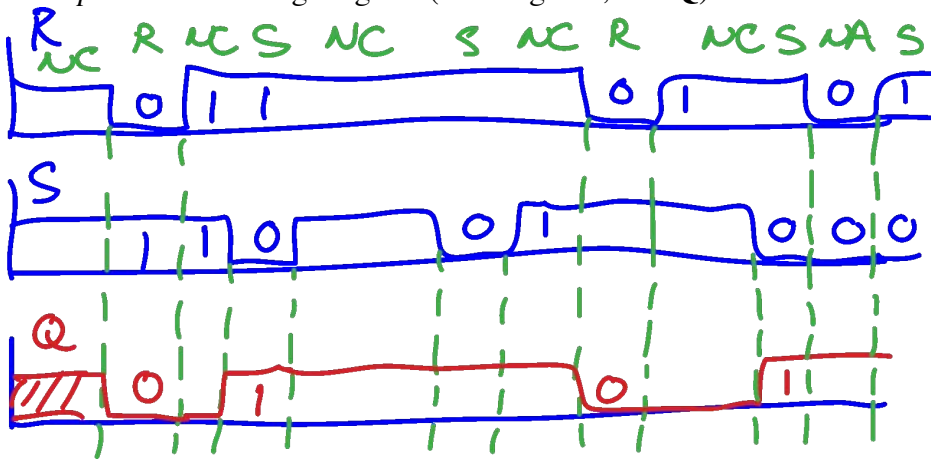


- More possibilities shown in truth table:

			PS NS		
S	R	Q	Q	Q̄	
0	0	0	1	1	Not Allowed (NA)
0	0	1	1	1	
0	1	0	1	0	Set
0	1	1	1	0	
1	0	0	0	1	Reset
1	0	1	0	1	
1	1	0	0	1	NC, (Memory)
1	1	1	1	0	

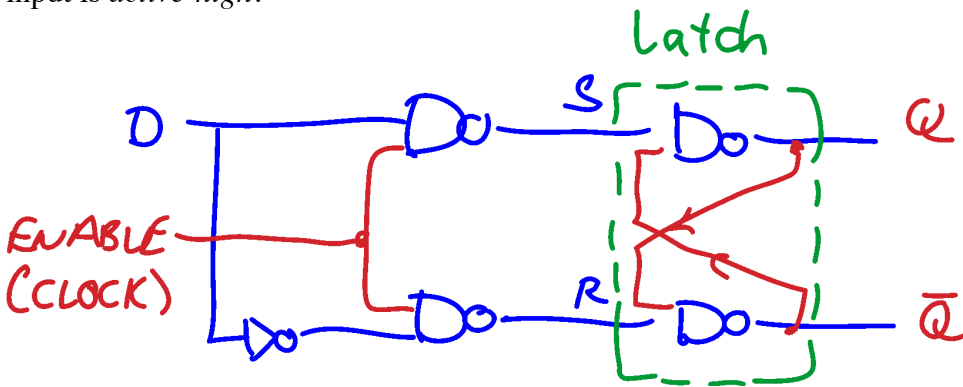
S	R	Q	
0	0	1	NA
0	1	1	Set
1	0	0	Reset
1	1	Q	NC

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

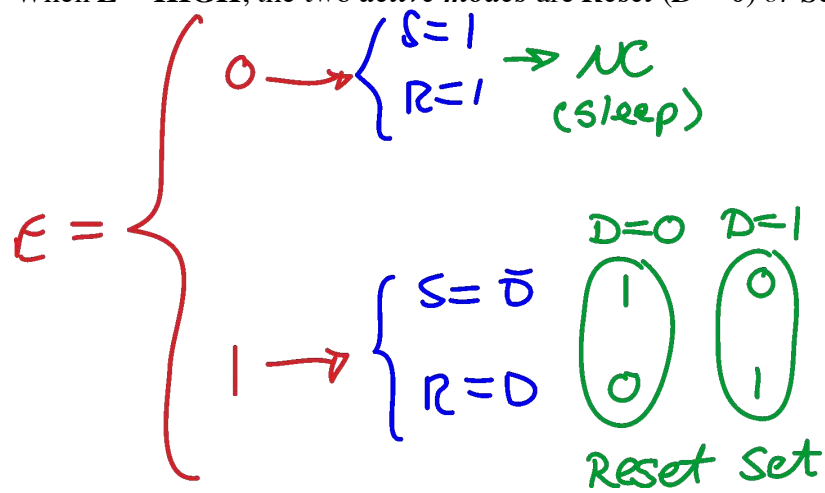


Data Latch with Enable (D-Latch)

- **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**.

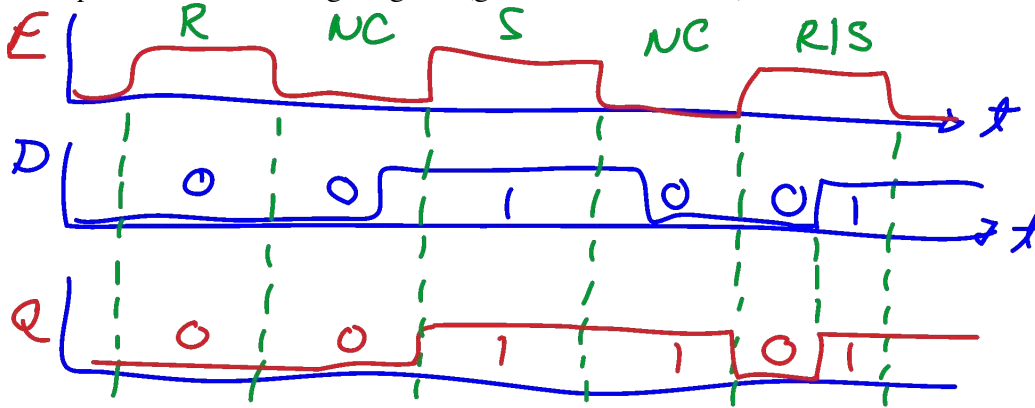


- The **enable input** forms the basis of a rudimentary clock signal.
- Truth table:

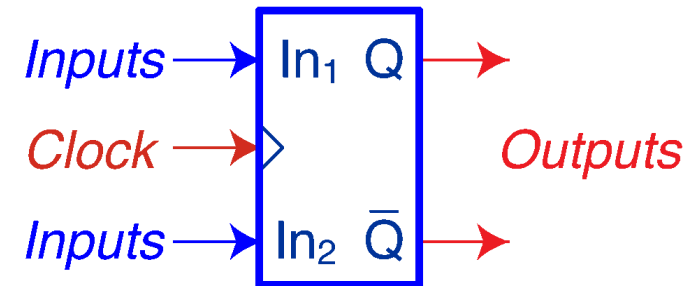
E	D	Q
0	0	Q
0	1	Q
1	0	0
1	1	1

} NC
 Reset
 Set

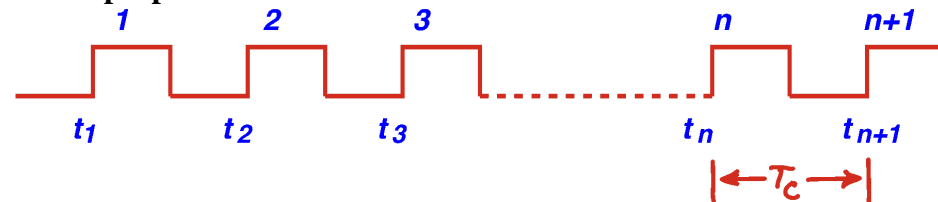
- Example: **D-Latch** timing diagram (given E & D, find Q)



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

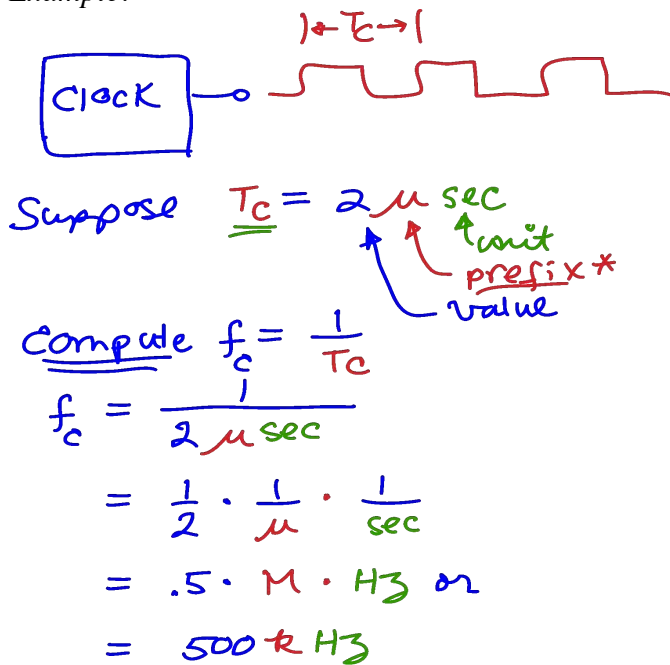


Clock properties:



- 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*:
 - $10^{-9} = n \leftrightarrow G = 10^{+9}$
 - $10^{-6} = \mu \leftrightarrow M = 10^{+6}$
 - $10^{-3} = m \leftrightarrow k = 10^{+3}$

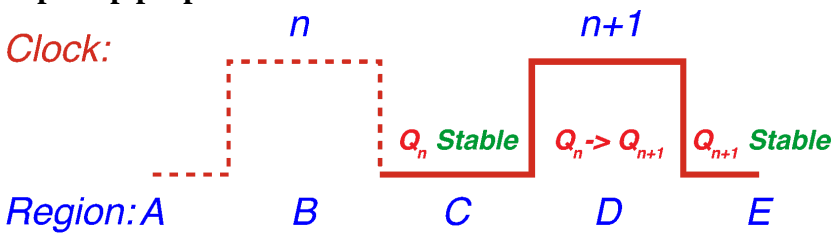
- Example:



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.

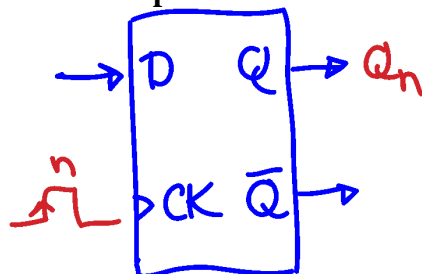
Flip-Flop properties



- $Q = Q_n$ 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**

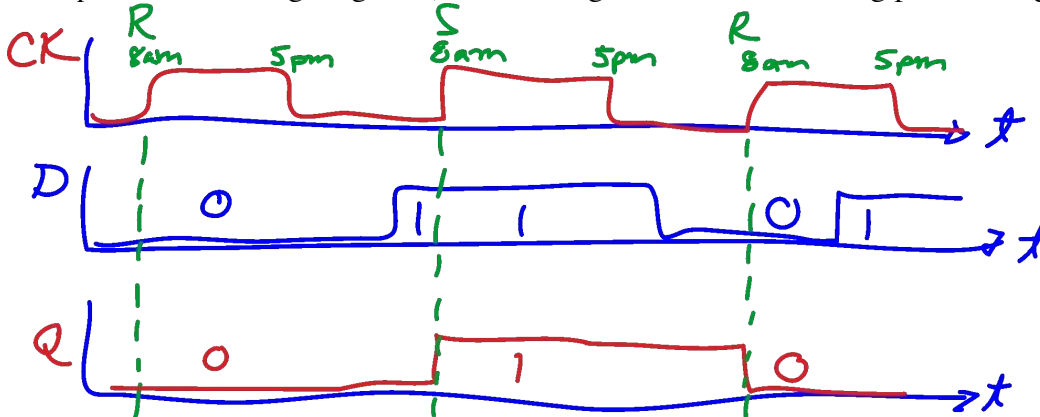


- There are **two active modes** of operation:
 - **Reset** ($D = 0$)

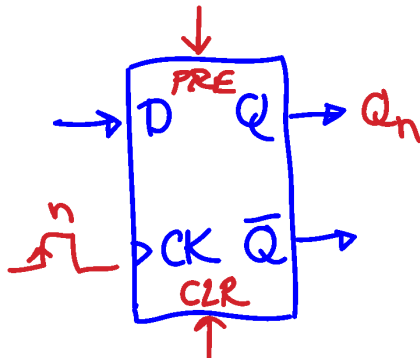
- Set ($D = 1$)

NS	
D	Q_{n+1}
0	0 Reset
1	1 Set

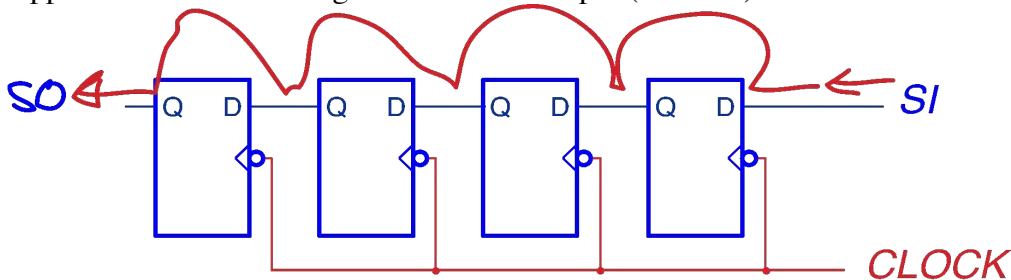
- State changes ($Q_n \rightarrow Q_{n+1}$) are triggered on **edges** of the clock:
 - Rising edges ($\uparrow$) for positive edge triggered F/Fs
 - Falling edges ($\downarrow$) for negative edge triggered F/Fs
- Example: D F/F timing diagram (Clock & D given, find Q assuming positive edge trig.)



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

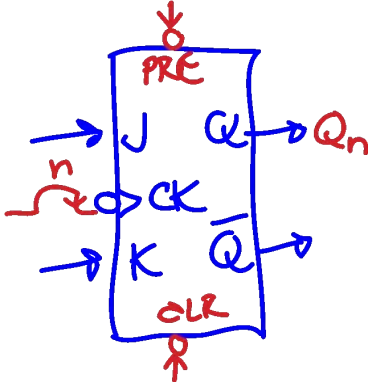


- Application: 4-bit shift register with serial input (see later)



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

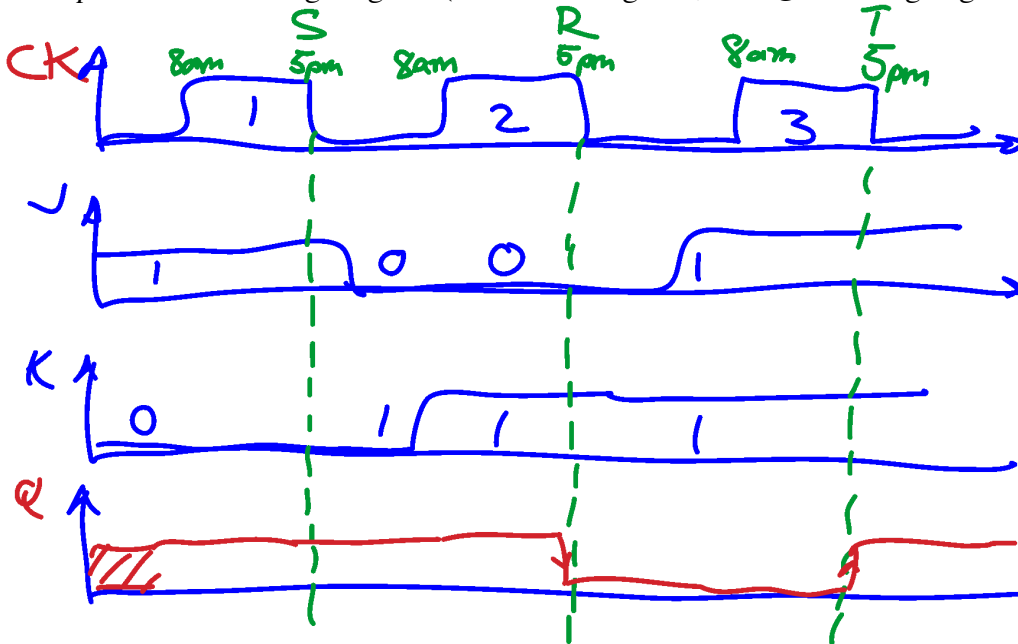
- Like D F/F, except with 2 inputs **J** and **K**.
(Falling edge clock and active-low preset inputs shown)



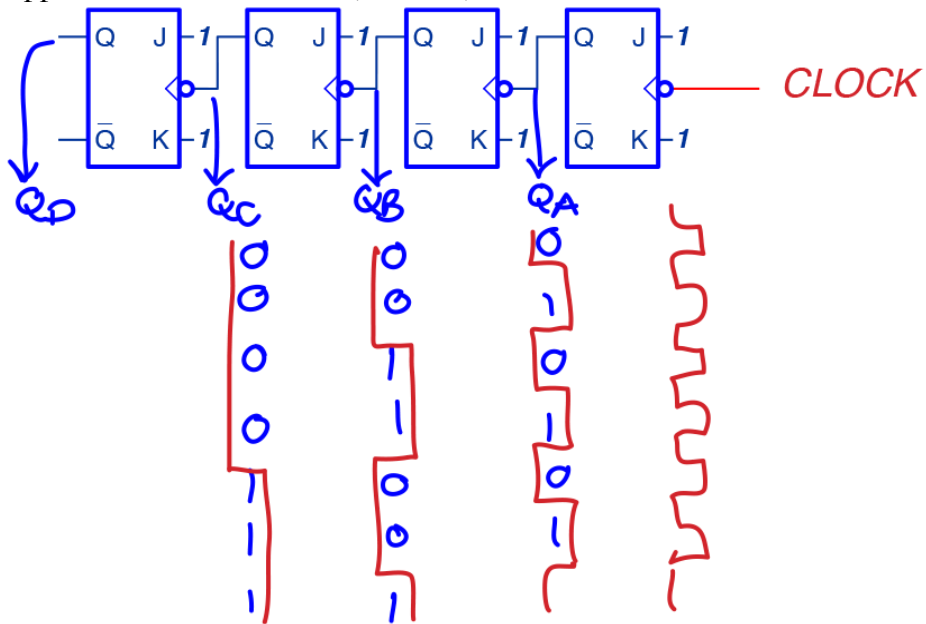
- There are **four active modes** of operation:
 - No Change ($J = 0, K = 0$)
 - Reset ($J = 0, K = 1$)
 - Set ($J = 1, K = 0$)
 - Toggle ($J = 1, K = 1$)

		NS
J	K	Q_{n+1}
0	0	Q_n NC
0	1	0 Reset
1	0	1 Set
1	1	$\bar{Q}_n$ Toggle

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



- Application: 4-bit Counter (see later)

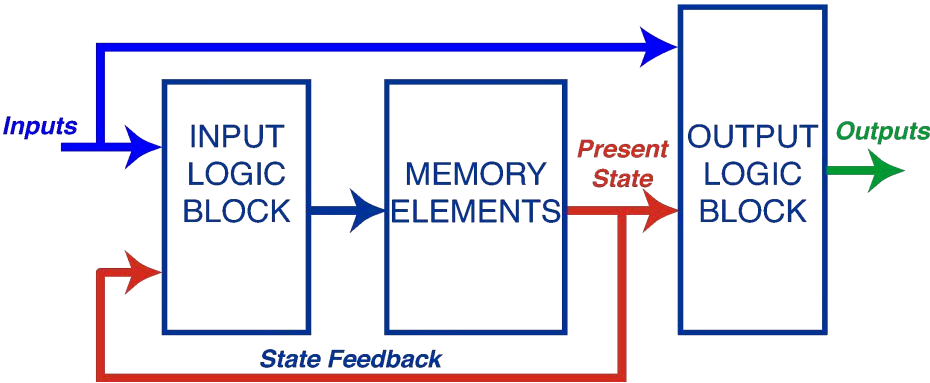


Quiz #8 & selected solutions

STATE MACHINE DESIGN

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

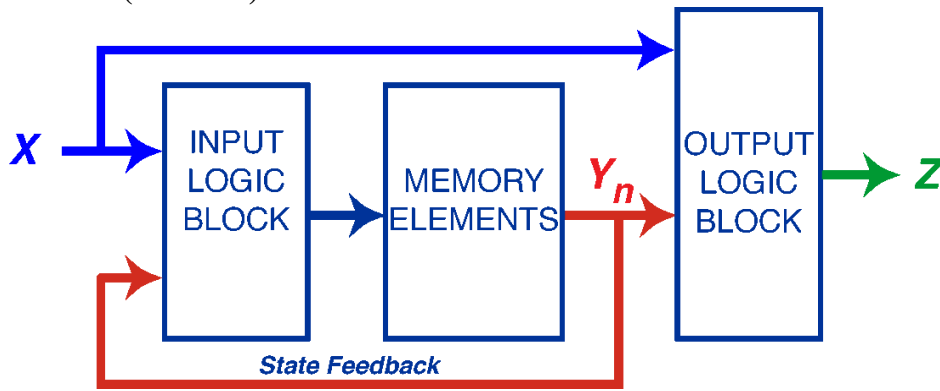
General SLC Block Diagram



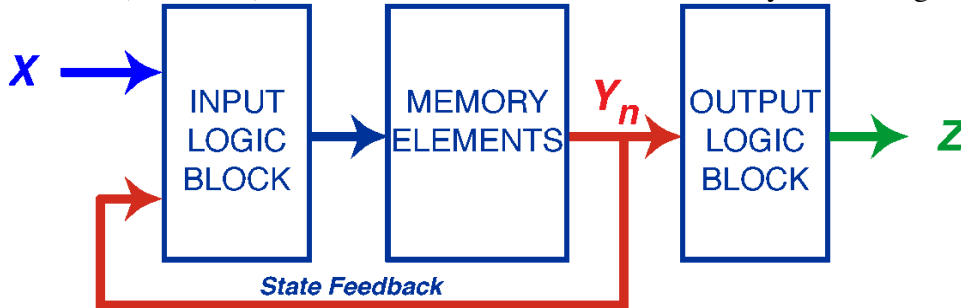
- The **Present State (PS)**: the bit pattern of the **MEMORY ELEMENTS** (a register) at a prescribed observation time.
- The **Next State (NS)**: the future state of the **MEMORY ELEMENTS**, formed by the **INPUT LOGIC BLOCK**.
- The **OUTPUTS**: the present output bit pattern, formed by the **OUTPUT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

SLC Classes

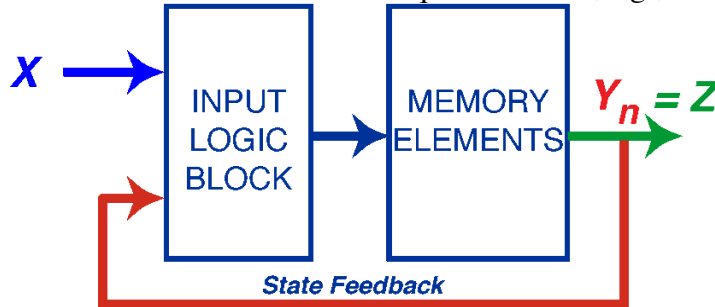
- **Class A (MEALY) SLC:** **OUTPUTS** are a function of the **PS** and **INPUTS**, e.g., $Z = f(X, Y)$.



- **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., $Z = Y$.

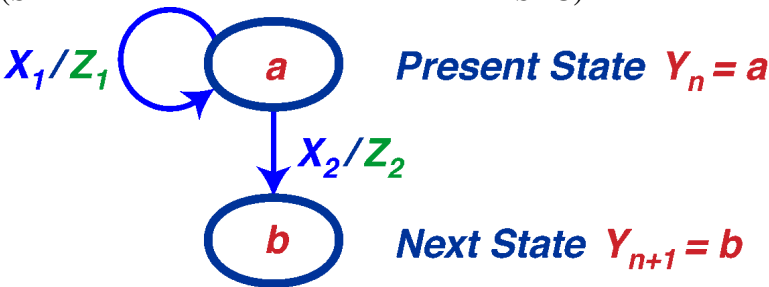


SLC Variable Notation

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

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)

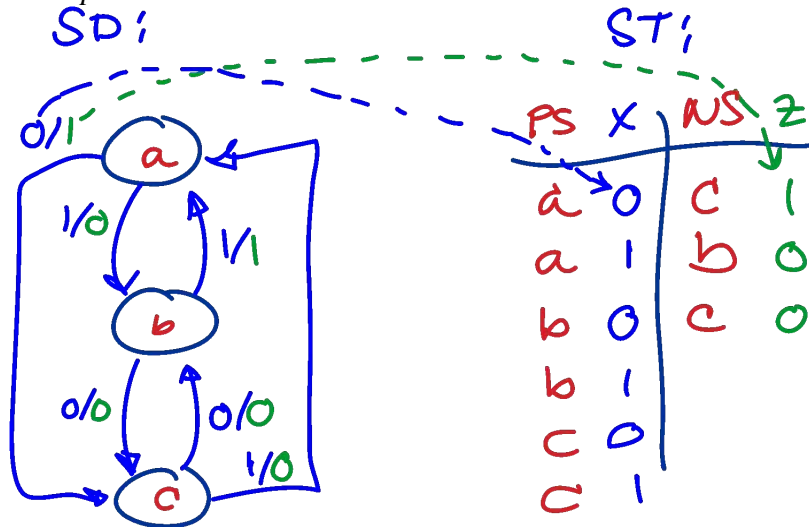
(Tabulates **NS** and **Outputs** verses **PS** and **Inputs** of a **SD**)

PS	In	NS	Out
Y_n	X	Y_{n+1}	Z
a	X_1	a	Z_1
a	X_2	b	Z_2

No Change
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

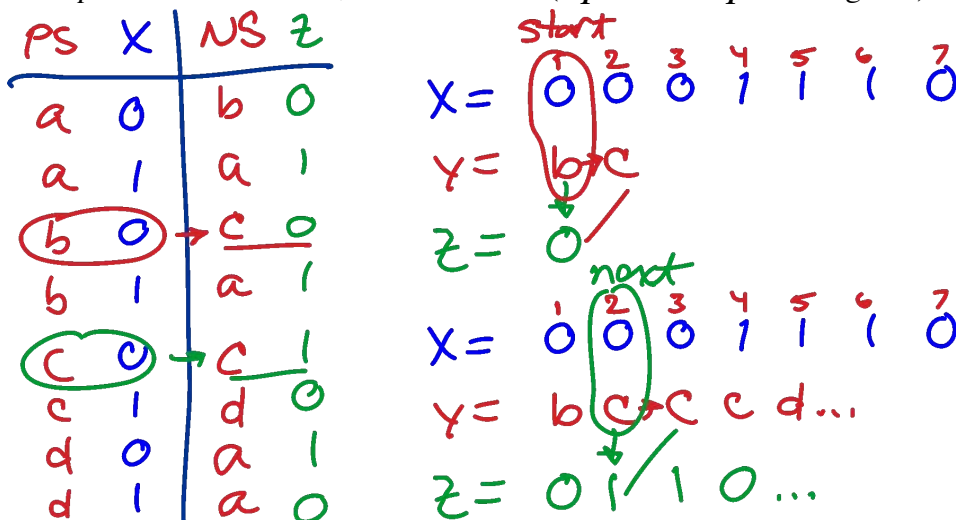
Similar to timing diagram, where X is given and Y and Z need to be filled in:

X : $X_1 X_2 X_3 \dots$ (sequence of inputs X_n)

Y : $Y_1 Y_2 Y_3 \dots$ (sequence of states Y_n)

Z : $Z_1 Z_2 Z_3 \dots$ (sequence of outputs Z_n)

- Example: Given ST & PS, construct trace ($X_1 = 0$ and $Y_1 = b$ are given)

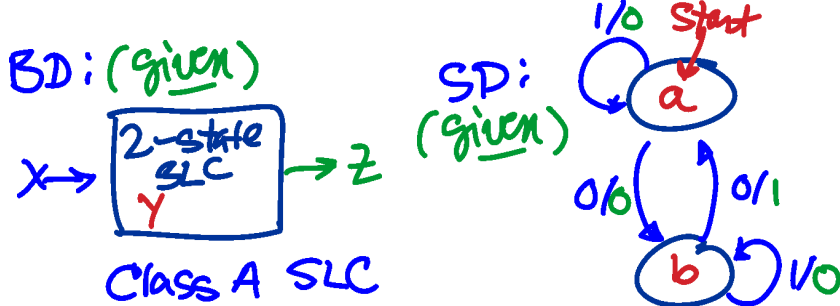


State Machine Design Procedure

(The procedure will be illustrated by three examples)

Example#1:

- 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.

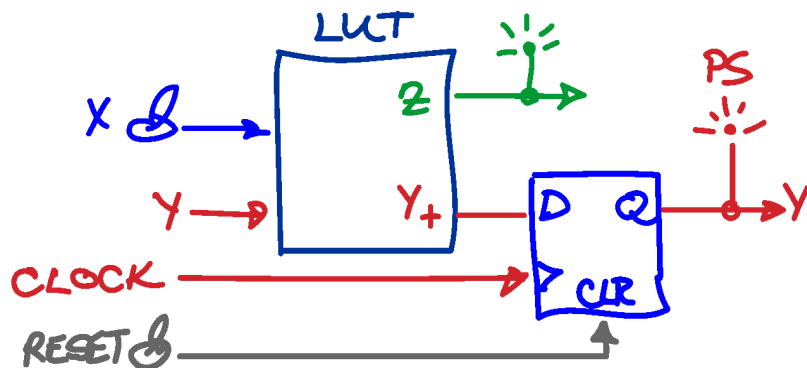
ST:

PS	IN	NS	OUT
Y_n	X	Y_{n+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
Y_n	X	Y_{n+1}	Z
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	0

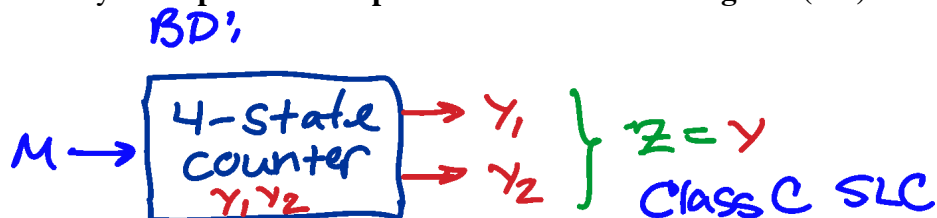
3. Program the state table into a lookup table. Feed the next state output into a D flip-flop.



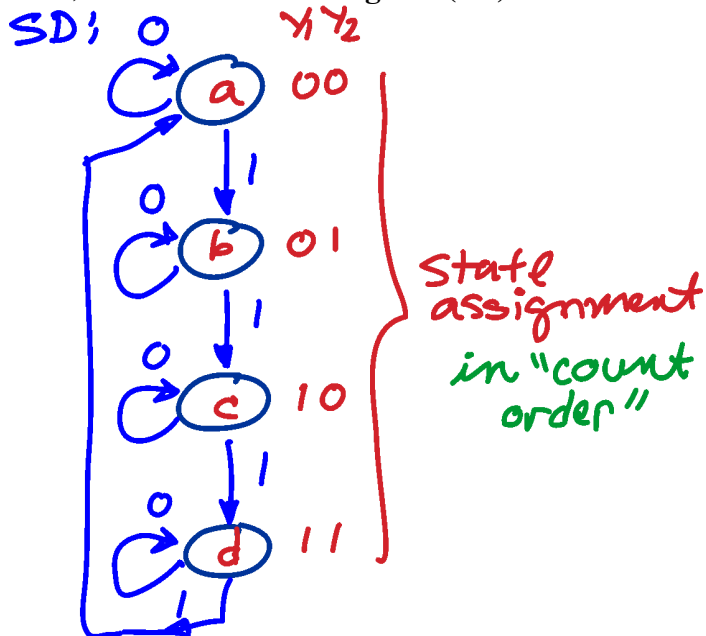
Example#2:

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)**:

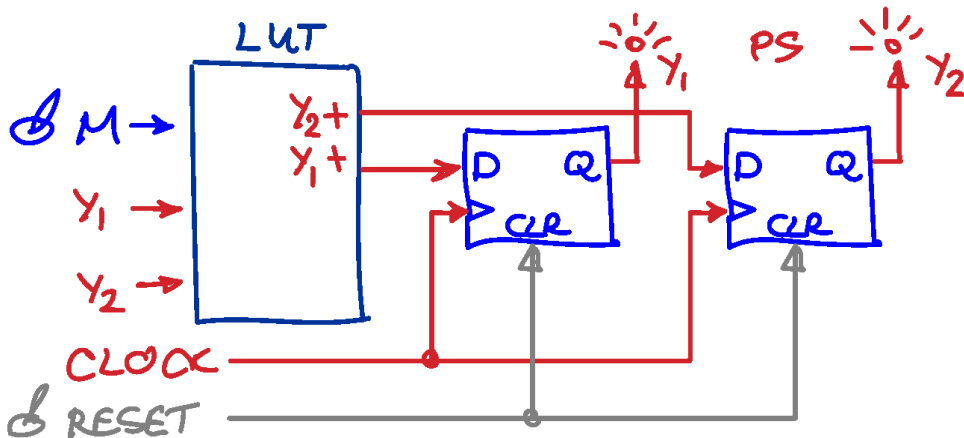


- Construct a **State Table (ST)**:

ST: (truth = state table)

$y_1 y_2 M$	y_{1n+1}	y_{2n+1}	
a { 00 0	0	0	nc
00 1	0	1	count
b { 01 0	0	1	nc
01 1	1	0	count
c { 10 0	1	0	nc
10 1	1	1	count
d { 11 0	1	1	nc
11 1	0	0	count

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



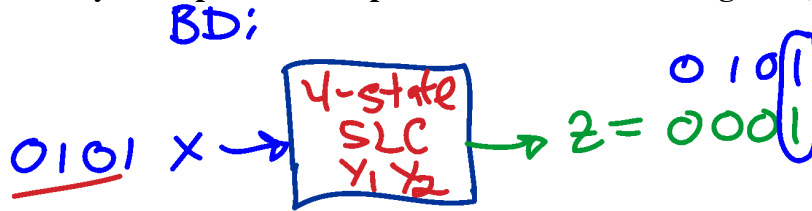
Example#3:

- Given a set of specifications for a **SLC**:

- Design a code sequence detector for the code $X = 0101$

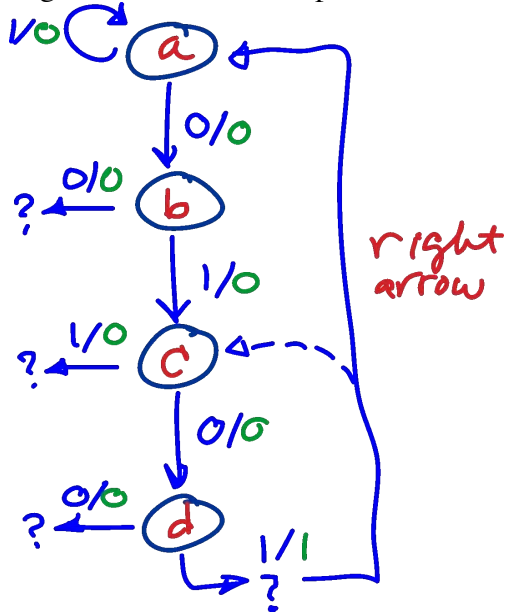
- When $X = X_1X_2X_3X_4 = 0101$,
then $Z = Z_1Z_2Z_3Z_4 = 0001$ ($Z_4 = 1$ when $X_4 = 1$)
- Otherwise Z_i stays 0

- 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:

$X = 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$
 $Y = a \ b \ c \ d \ a \ b \ c \ d$
 $Z = 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1$

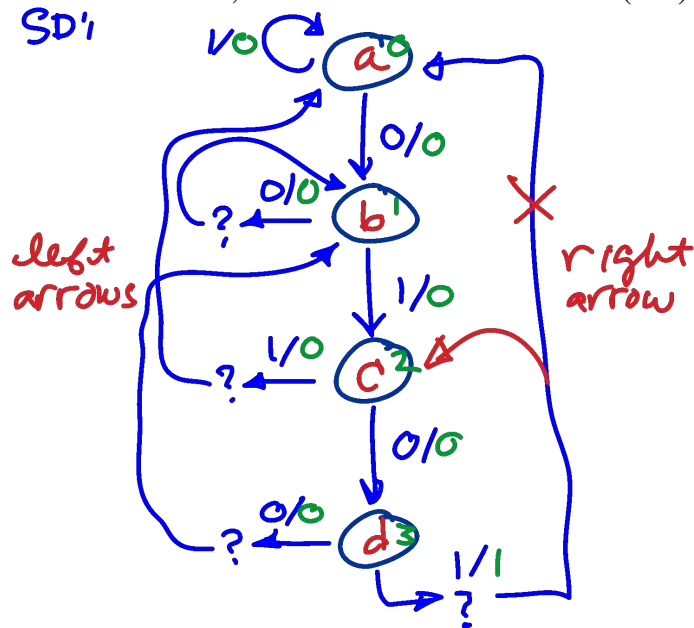
- State trace following dashed-line right arrow to state c, allowing an overlapping sequence:

$X = 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$
 $Y = a \ b \ c \ d \ a \ b \ c \ d$
 $Z = 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$

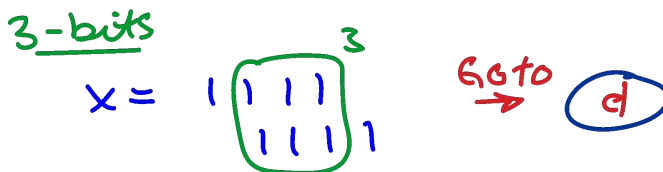
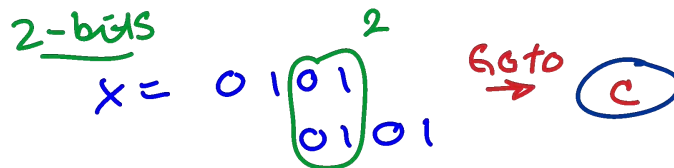
overlap

$c \rightarrow d$

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

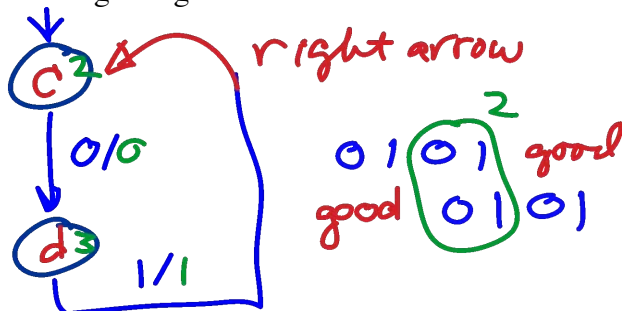


- Where the right and left arrows are routed to is determined by the number of overlapping bits. Here are some general examples:



- 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



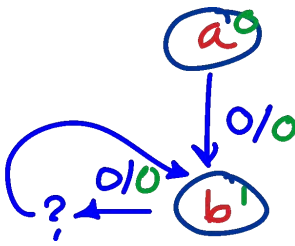
- Also, overlapping the bad sequence $X = 00$ (from state **a** and out of state **b**) with the good sequence $X = 0101$ yields:

- 1 bit of overlap

- Routing that goes back to state b

left arrow

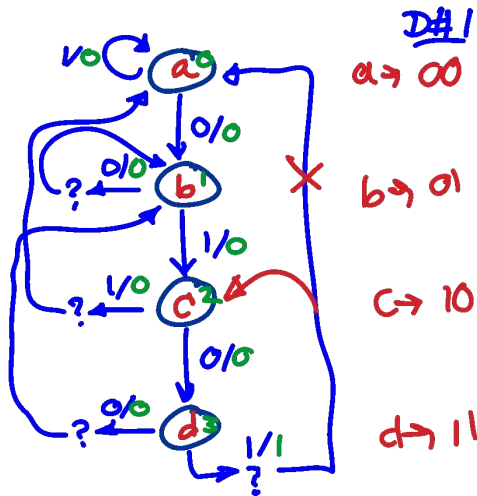
bad
good



2. Construct **State Tables (ST)**: Lets make two designs based upon different state assignments.

- Design #1 State diagram and table. Note that $Z = XY1Y2$

SD:

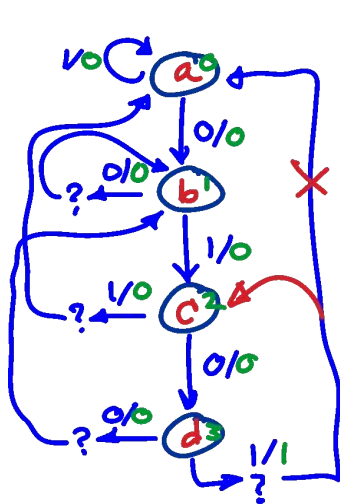


ST:

Design #1		
$Y_1 Y_2 X$	Y_{1n+1}	$Y_{2n+1} Z$
a 00 0	b 0 1	0
00 1	0 0	0
01 0	0 1	0
01 1	1 0	0
10 0	1 1	0
10 1	0 0	0
11 0	0 1	0
11 1	1 0	1

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

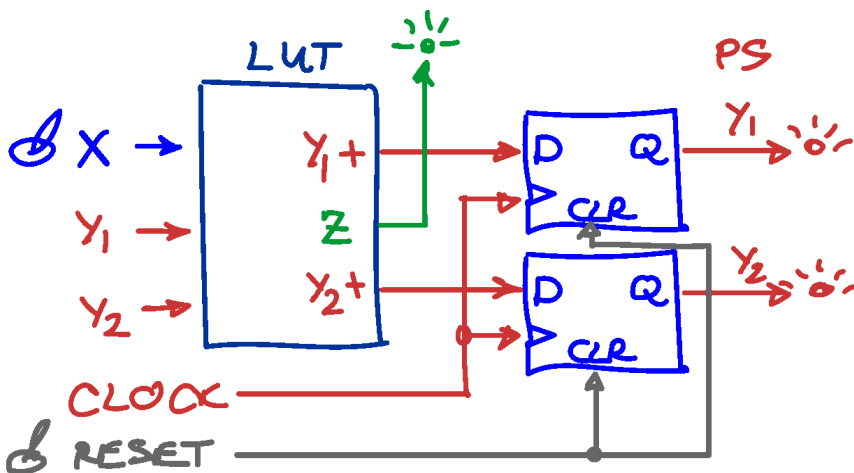
SD:



ST:

Design #2		
$Y_1 Y_2 X$	Y_{1n+1}	$Y_{2n+1} Z$
a 00 0	b 1 1	0
00 1	0 0	0
01 0	1 0	0
01 1	0 0	0
10 0	1 1	0
10 1	0 1	1
11 0	1 1	0
11 1	0 1	0

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



REGISTERS AND COUNTERS

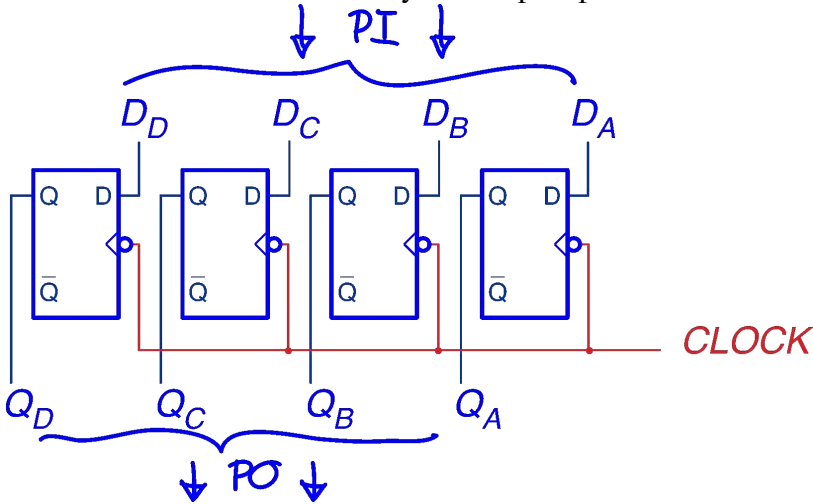
Register defined: a SLC which deals with the storage and transfer of multi-bit data.

Data Transfer Modes

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

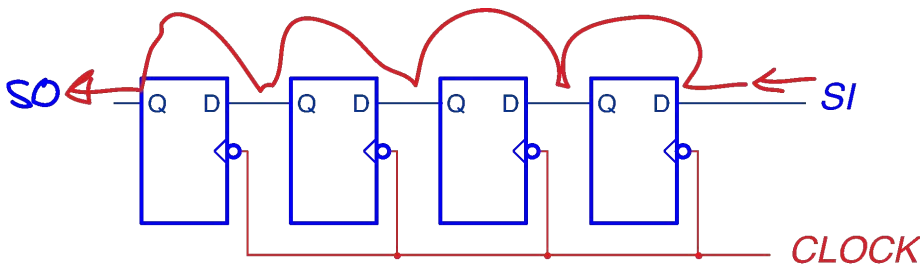
Data Latch Register

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

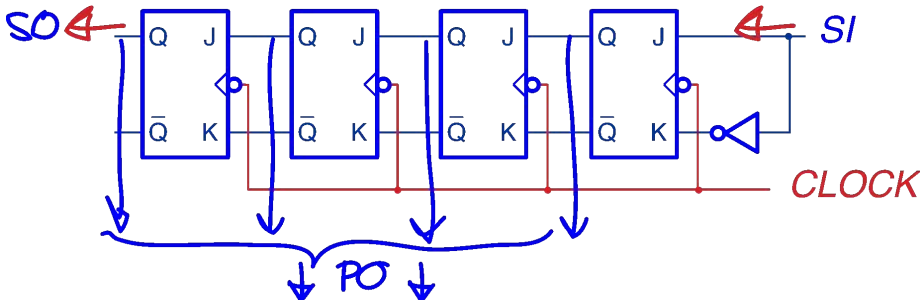


Shift Register

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

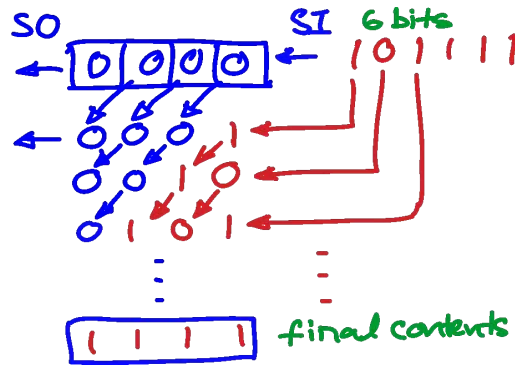
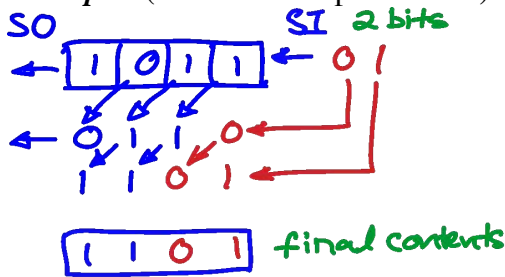


- **JK** F/Fs:

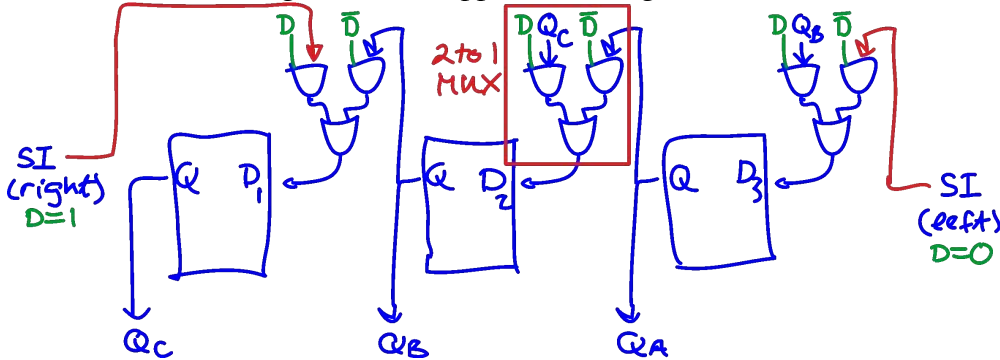


- **Multi-mode** shift registers in addition support **SIPO**, **PISO**, or even **PIPO** data transfers

- **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.

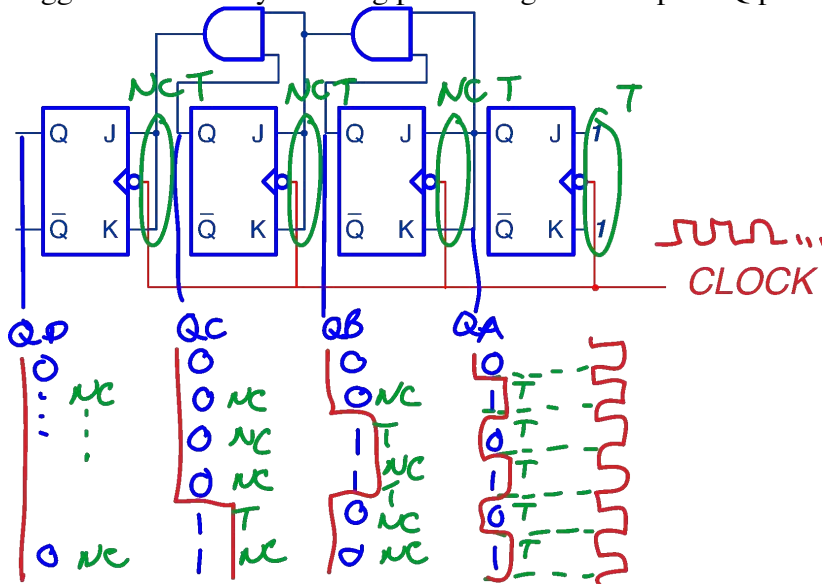
Counter defined: a SLC which deals with the generation of multi-bit sequential codes.

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

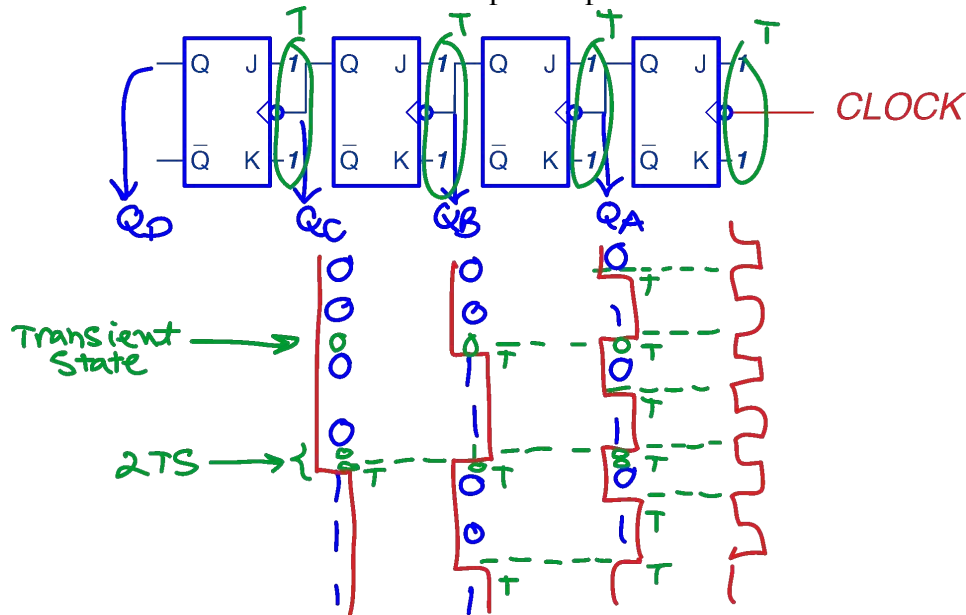
Single-Mode Synchronous Counter

- 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 preceding 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):

