

IMPORTANT TYPES OF COMBINATIONAL LOGIC CIRCUITS

Notes:

Decoder

1. Defined as a circuit which translates an **n-bit input codeword** into a larger **m bit output word**, where $m \leq 2^n$.
2. Useful for making *selections*. For simple decoders, the m bit output word consists of one selected (active) bit, and m-1 inactive bits. **n select lines** determine which output is selected.
3. Architecture consists of an array of **m ANDing elements** (*active-high* outputs: AND gates; *active-low* outputs: NAND gates)
4. For simple decoders, the outputs are minterms of the input select line variables.
5. The **enable signal E** must be in the active state to enable the decoder.
6. Simple decoders may also be employed as a **SOP Boolean function generator** of the n select lines. Sum desired minterms by attaching an **ORing element** to specific decoder outputs (each output line contributes one minterm.)

Multiplexer (MUX)

1. Defined as a circuit which connects one of 2^n **input data lines** to one **output line**.
2. The data source selected for connection is determined by a set of **n** input select lines.
3. Architecture consists of a **decoder** plus an **ORing element**.
4. Data lines D_j are ANDed with decoder minterms to provide selection of data sources.
5. The **enable signal E** is provided for activation.
6. Demonstration of multiplexing action as provided by a mechanical switch.
7. May also be employed as a **SOP Boolean function generator** of the input select lines (the data inputs specify the function values.)

Encoder

1. Defined as a circuit which translates **n bit dataword** into a **m-bit codeword**, where $n \leq 2^m$.
2. Useful for doing bit compression.
3. Architecture consists of a linear array of **m ORing elements**.

Demultiplexer (DeMUX)

1. Defined as a circuit which connects one **input data line** to one of 2^n **output lines**.
2. The output line selected for connection is determined by a set of **n** output select lines.
3. Demonstration of demultiplexing action as provided by a mechanical switch.
4. Demultiplexer circuits are typically realized from decoder circuits by utilizing the **decoder enable input line E** as the data input line.

Notice that decoder circuit is used in every important CLC except the encoder.

Read-Only Memory (ROM)

1. Programmable truth table
2. n-bit address input (selects a row)
3. m-bit data output (typically $m = 8$, for one row)
4. Capacity: $C = 2^n \times m$ (contents of truth table)

ROM Organization and Type

1. Decoder (selects a word)
2. Encoder array (stores the word)
 - a. Diode-fuse array
 - b. MOS transistor-fuse array
 - c. Double gate MOS transistor array
3. Historically 5 kinds of ROM implementations.
computer.howstuffworks.com/rom1.htm
 - a. ROM (mask)
 - b. PROM (programmable ROM):
 - c. EPROM (erasable-programmable ROM):
 - d. EEPROM (electrically-erasable PROM):
 - e. Flash (fast block-based EEPROM)

ROM Transistor Implementations

Three examples:

1. **PROM** computer.howstuffworks.com/rom3.htm
 - a. Decoder built from NMOS array.
 - b. Encoder uses transistor-fuse array.
 - c. **Fuses** are altered when ROM is programmed
 - i. A intact fuse represents a **0**
 - ii. A blown fuse represents a **1**.
2. **EPROM**
 - a. PROM transistor structure
 - b. Double-gate implementation
computer.howstuffworks.com/rom4.htm
3. **Flash** memories: [from wikipedia.com](http://from.wikipedia.com)
 - a. Double-gate implementation
 - b. NOR flash (like EPROM, can access individual bytes)

- c. NAND flash (smaller, but can access only blocks of bytes). Current technology used by portable music players.

Programmable Logic Devices (PLDs)

1. Consists of m **AND-arrays** and m **OR** gates having n input variables. There are p **AND** gates (product terms) in each **AND array**. A **PLD** is then defined as being of size $n \times m$ with p product terms.
2. A more simple definition comes from considering just the number of product terms (p) and vertical input lines ($2 \times n$) feeding one **OR** gate to define the $p \times 2 \times n$ **PLD**.
3. **Connections** between the inputs and the **AND** array are represented by **dots** (or **x**'s). No dot (or **x**) implies no connection. The dots are placed during the process of downloading a data bit stream into the **PLD** using special hardware and software.
4. **Xilinx XCR3064XL** complex XPA3 PLD (**CPLD**) structure. Consists of a number of interconnected **PLDs** (also called *function blocks*), each having 20 inputs ($n = 20$) and 8 *macrocells* ($m = 8$ outputs). There are 5 product terms ($p = 5$), per macrocell.
5. The **XCR3064XL** has 6 PLDs, each PLD having 8 macrocells. Each **PLD AND array** has $5 \times 8 = 40$ product terms and is connected to the macrocells through a *Zero-power Interconnect Array (ZIA)*.

SEQUENTIAL LOGIC CIRCUITS

Notes:

SLC defined: logic circuits whose outputs depend upon **state**.

Random Access Memory (RAM)

1. Rewritable truth table of cells
 - a. Read data (read 1 word of state from table)
 - b. Write data (write 1 word of state to table)
2. n -bit address input (selects a word)
3. m -bit data input/output determines *word length*:
 - a. $m = 8$ bits (1 byte)
 - b. $m = 16$ bits (2 bytes)
 - c. $m = 32$ bits (4 bytes)
 - d. $m = 64$ bits (8 bytes)
 - e. $m = 128$ bits (16 bytes)

RAM Organization and Type

1. Addressing (selecting a word).
 - a. Straight n-bit decoding. n physical address bits with array that has width m and decoder selects m-bit word. Address size = n.
 - b. Row decoders and column MUX. n_r bits for row selection of large array, and reuse $n_c \leq n_r$ bits for column decoding (via MUX) to select a word from that row. Address size: $n = n_r + n_c$
 - c. Bank organization. Address includes 1 or more bank lines for switching between memory banks in chip. Address size: $n = n_r + n_c + n_b$
2. Storage cell array technology
computer.howstuffworks.com/ram.htm
 - a. Static (no refresh; constructed from gates)
Dynamic (requires refreshing by CPU)
 - b. Synchronous Dynamic (SD) (self-refreshing)
 - c. Double Data Rate (DDR) SD (faster)
3. Refreshing cycle. Uses row and column decoders to select cells for refresh
computer.howstuffworks.com/ram1.htm
4. Depending upon cell type, we have
 - a. SRAM (Used for registers, and caches)
 - b. DRAM (older memories)
 - c. SDRAM (current memories)
 - d. DDR SDRAM (latest memories)
5. Memory modules:
computer.howstuffworks.com/ram4.htm
 - a. SIMM (single in-line)
 - b. DIMM (dual in-line)
 - c. SODIMM (small outline dual in-line)

Putting it all Together: Apple iPhone

1. Mobile Communication Device
 - a. Cell phone (voice/texting)
 - b. iPod (music/video)
 - c. Internet Browser (web/email/facebook/twitter)
2. iPhone Block Diagram
www.phonewreck.com/2009/06/19
 - a. Processor ARM Cortex A8
 - b. USB & Audio Ports
 - c. 256 MB DDR SDRAM
 - d. NOR Pseudo SRAM
 - e. 16 GB NAND Flash
 - f. Cell phone Transceiver & Power Amps
 - g. GPS
 - h. Accelerometer/Magnetometer
 - i. WiFi and Bluetooth Transceiver
 - j. Camera
 - k. Touch Screen

3. S5PC100 System on a Chip (SoC)
 - a. Processor ARM Cortex A8 (600 MHz)
 - b. GPU Power VR SGX (OpenGL ES 2.0)
 - c. 256 DDR RAM
4. Processor Details

www.arm.com/products/CPUs/

 - a. Dual instruction issue, in order.
 - b. 13 stage pipeline architecture, major steps are
 - i. iFetch
 - ii. iDecode
 - iii. iExecute
 - iv. Load/Store
 - c. L1 cache 32KB instructions/32KB data
 - d. L2 cache (256 KB)
 - e. Neon Unit (128 bit SIMD for multimedia)
5. Software Applications (Apps)

Writing an app tutorial: servin.com/iphone

Look at one bit now: Simple Latch Circuit

1. The latch performs the simplest kind of **binary information storage**. It is employed in some form in virtually all memory element circuits.
2. Circuitry consists of two **cross-coupled NAND** (or NOR) **gates**, with outputs (Q), fed back to inputs (S , R). Inputs are *active-low*.
3. Latch state is specified by the latch outputs; $(Q = 1, \bar{Q} = 0)$ indicates the latch is **Set**, $(Q = 0, \bar{Q} = 1)$ indicates the latch is **Reset**.
4. Latch **mode of operation** is specified by the latch inputs. Allowable **modes** include the **No Change**, **Set** and **Reset** modes.
5. Latch operation shall be demonstrated by use of truth table and timing diagrams.

Data Latch with Enable (Used in SRAM)

1. **Circuitry** consists of a simple latch plus additional logic which is under the control of an enable signal E and input D . The D input is *active-high*.
2. **Mode of operation** is specified by the input D , and the enable as follows: when $E = \text{LOW}$, mode = **No Change** (this puts the latch to sleep); when $E = \text{HIGH}$, mode = **Set** or **Reset**, as determined by D .
3. The enable input forms the basis of a rudimentary clock signal, which shall be examined more closely in the next section

FLIP-FLOPS AND CLOCKS

Notes:

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

Properties of Clocks (A general clock waveform will be presented and the following points should be noted.)

1. Clock outputs yield uniform **rectangular** waveforms when plotted against time.
2. $T_c = \text{clock period}$ = time between consecutive clock waveform rising (or falling) edges, expressed in units of seconds.
3. $f_c = \text{clock frequency}$ = number of complete clock oscillations (cycles) per second of time, expressed in Hertz or cycles per second (Hz=cycles/sec), where $f_c = \frac{1}{T_c}$.

Importance of Clocks

1. Up part defines **day time**, when flip-flops are active, and respond to inputs.
2. Down part defines **night time**, when flip-flops are sleeping, and are not responsive (put in the no change mode).
3. Flip-flop input signals (i.e. bread for breakfast) must come at night to be responded to.

Flip-Flop Notation and Characteristics (A general flip-flop circuit will be presented and the following points should be noted.)

1. $Q_n = Q$ is the stable **PS** Q value when the **CLOCK** is **LOW** *before* the $n^{th}+1$ clock pulse. Occurs during “night time”.
2. $Q_{n+1} = Q$ is the stable **NS** value when the **CLOCK** is **LOW** *after* the $n^{th}+1$ clock pulse, i.e. the Q value tomorrow night.
3. $Q_n \rightarrow Q_{n+1}$ is the **PS** to **NS** Q change when the **CLOCK** is **HIGH** during the $n^{th}+1$ clock pulse. This occurs during the “daytime”.
4. **F/F data inputs** (which control state changes) $Q_n \rightarrow Q_{n+1}$ must arrive *before* the $n^{th}+1$ clock pulse and remain stable “all day” while the **CLOCK** is **HIGH**.

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

1. Allows a **flip-flop to load its next D input only coincident with a clock-pulse edge**; subsequent input changes are **locked out** (and do not effect the output) until the next corresponding clock edge.
2. **Circuitry** consists of two latches (a **master** and a **slave** latch), using 8 gates. With some optimization, circuitry can be reduced to 6 gates, used by the commercial version.
3. **F/F state** is specified by the latch outputs; state changes ($Q_n \rightarrow Q_{n+1}$) are initiated on the **rising** edge of the clock.
4. Both **positive** (rising) and **negative-edge** (falling) triggering may be implemented.
5. **F/F mode of operation** is similar to that of the **D-Latch**: when **CLOCK = LOW**, mode = **No Change**; when **CLOCK = HIGH**, mode = **Set or Reset**, as determined by **D**. For positive edge triggered case, when **CLOCK = HIGH**, the **D** input is enabled; the master

latch is either set, reset, toggled, or not changed. When **CLOCK = LOW**, the **D** input is disabled and the output of the master latch is gated into the slave latch.

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

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

1. Like **D F/F**, except with 2 inputs **J** and **K**.
2. There are **four modes** of operation: **No Change**, **Set**, **Reset**, or **Toggle**, as specified by the **J** and **K** inputs.

SYNCHRONOUS SEQUENTIAL CIRCUIT ANALYSIS AND DESIGN

Notes:

General SLC Block Diagram

1. The **Present State (PS)** of the **SLC** is defined to be the output bit pattern of the **MEMORY ELEMENTS** (a register) at the prescribed observation time.
2. The **Next State (NS)** of the **SLC** is the future state of the **MEMORY ELEMENTS**, and is formed by the **INPUT LOGIC BLOCK**, which utilizes the **SLC INPUTS** as well as the **SLC Present State**.
3. The **OUTPUTS** of the **SLC** are determined by the **OUTPUT LOGIC BLOCK**, and are dependent upon the **SLC Present State** and **INPUTS**.

SLC Classes

1. **Class A (MEALY)** circuit: **OUTPUTS** are a function of the **PS** and **INPUTS**, e.g.,
 $Z = f(X, Y)$.
2. **Class B (MOORE)** circuit: **OUTPUTS** are a function of only the **PS**, e.g., $Z = f(Y)$.

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

SLC Variable Notation

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

The State Diagram (SD)

Shows the state to state transitions of a SLC

1. Components:
 - a. **Bubbles**: represent the **SLC states** labeled with alphabetic letters (**a, b, c**, etc.) or with **state codes** (000, 001, 010, ...)
 - b. **Arrows** indicate allowable **state transitions** (present to next.) Arrows specify the transition input conditions and resulting outputs.
2. Types:
 - a. *Mealy*: arrows are labeled with code **X/Z**
 - b. *Moore*: arrows are labeled with **X** only; bubbles are labeled with code **Y/Z**.

State Trace

A timing diagram which lists states and outputs for a given sequence of input values:

1. 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)

2. Construction:

Start with $n = 1$ (X_1 and Y_1 are given):

 - a. Examine **PS** Y_n and input X_n
 - b. Look up **NS** Y_{n+1} and Z_n in **SD**
 - c. Write Y_{n+1} and Z_n in trace
 - d. $n = n + 1$, Go to step a

Flip-Flop State Diagrams

1. D flip-flops
2. JK flip-flops

The State Table (ST)

A tabulation of **NS** and **outputs** verses **PS** and **inputs** of a **SD** in truth table fashion.

1. State Table format:

<u>PS In</u>		<u>NS Out</u>	
Y_n	X	Y_{n+1}	Z

2. Construction:
Examine **SD**: for every Y_n bubble, record Y_{n+1} and output Z indicated by arrow X .
3. There is a **ST** row for every **SD** state bubble.

Synchronous SLC Design Procedure (*This procedure is useful for designing SLCs, such as required in the ECE 250 Laboratory.*)

1. Given a set of specifications for a **SLC**, **identify the inputs and outputs** and **draw a block diagram**. (You may at this point be able to observe if the specifications stipulate a **Class A, B, or C SLC** design.) Also, **formulate a state diagram** that describes the functionality of the **SLC** according to the set of design specifications.
2. Construct a **state table** based on the information obtained in the state diagram of step (1), showing **PS** variables $Y = Y_n$ and inputs X verses **NS** variables Y_{n+1} and outputs $Z = f(X, Y)$.
3. Draw the **next state maps** (one for each Y_{n+1}) and **OUTPUT K-maps** (one for each Z) based upon the state table of step (2).
4. If **JK F/Fs** are to be employed, transform the **next state maps** into **JK maps** for each state variable Y . If **D F/Fs** are to be used, the **D maps** are the same as the **next state maps**.
5. Read the maps of steps (3, 4) to obtain **SOP expressions** for the **J, K, or D F/F** input variables as well as **Z**. Translate these equations into appropriate **circuits**

Synchronous SLC Analysis Steps (This procedure is useful for creating a state diagram for an already existing **SLC**, whose behavior needs to be analyzed.)

1. Examine the given **SLC** and determine the logic equation for each F/F input (**J**, **K**, or **D**) and circuit output **Z**.
2. Complete a K-map for the J, K, D, and Z variables. If JK F/Fs are used, consolidate the J and K maps into **next state** maps. If D F/Fs are used, the D maps are the **next state** maps. The **Z** variables generate the OUTPUT maps.
3. Transfer the contents of the **next state** maps and OUTPUT maps of step (2) into a single state table.
4. Construct a state diagram from the state table of step (3).
5. If an input sequence $X = \text{----}$ is given, complete a timing diagram for **X**, **Y**, **Z** and **CLOCK**.

SOME IMPORTANT TYPES OF SEQUENTIAL LOGIC CIRCUITS

Notes:

Register circuits defined: those logic circuits which deal with the storage and transfer of multi-bit data.

The most significant bit (MSB) of a register is usually reserved for the sign bit..

Data Transfer Modes

1. PARALLEL-IN/PARALLEL-OUT (PIPO)
2. SERIAL-IN/SERIAL-OUT (SISO)
3. PARALLEL-IN/SERIAL-OUT (PISO)
4. SERIAL-IN/PARALLEL-OUT (SIPO)

Data Latch Register

1. Defined as a register circuit capable of performing only **PIPO** data transfers.
2. Typical architecture consists of a linear array of **D** flip-flops driven by a common clock line.

Shift Register

1. Defined as any register circuit capable of performing **SISO** data transfers.
2. Typical architecture consists of a linear array of **D** or **JK** flip-flop with inputs and outputs of adjacent flip-flops tied together, driven by a common clock line.
3. **Multi-mode shift registers** are augmented by additional logic circuitry to support **SIPO**, **PISO**, or even **PIPO** data transfers.
4. **Bidirectional shift registers** contain additional logic circuitry to support left or right data shifts.
5. A bidirectional shift register capable of supporting additional modes of operation is known as a **multifunction** register.

Counter circuit defined: those logic circuits which deal with the generation and storage of multi-bit sequential codes.

Counter Circuit Characteristics

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

Single-Mode Synchronous Counter

1. Defined as a synchronous counter circuit which counts in one direction.
2. Typical architecture (for binary code sequence generator) consists of a linear array of **JK** or **D** flip-flops, driven by a common clock line, configured either in the **toggle** or **no change** modes as determined by additional decoding circuitry attached to the Q outputs.
3. Count outputs are guaranteed stable for a very short time following application of each clock pulse.

Ripple Counter

1. Defined as an **asynchronous counter circuit** because only the LSB flip-flop gets the clock signal.
2. Typical architecture consists of a linear array of **JK** flip-flops, configured in **toggle** mode, with the clock inputs and outputs of adjacent flip-flops tied together.
3. **Ripple effect:** count outputs are not guaranteed stable until the most significant F/F receives its clock signal.