Digital Design

Ethan Anthony

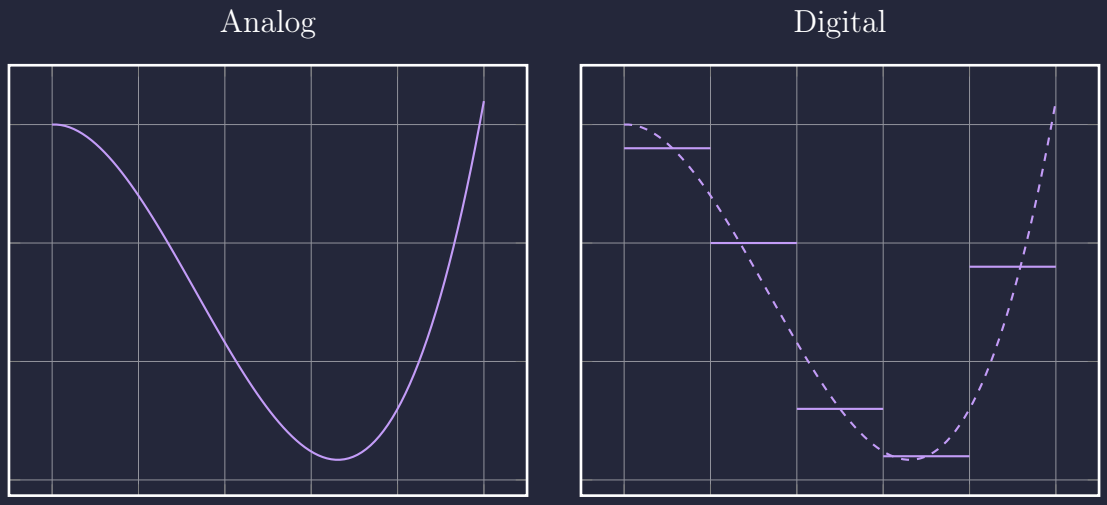
January 3, 2025

Contents

1 Foundational Concepts

1.1 Analog and Digital

The world we live in is **analog**, meaning signals and information are variable and continuous. Digital systems, on the other hand, are represented by discrete values.



(a) Analog (b) Digital

Figure 1: Analog vs. Digital

Digital is used for many different reasons:

- Represent and manipulate discrete elements of information
- Perform numeric computations on information with digital computers
- Use digital systems to manipulate elements of information
- Signals as elements of information
- Two-valued (binary) number system
- Integrated Circuit (IC) technology

1.2 Basic Definition

On the very basic level, there are a few definitions that will be used throughout the course.

Digital Systems	**1.1**
Manipulate **discrete elements** of information	

Hardware System **1.2**

A system whose physical components are constructed from electronic building blocks or modules.

Binary Digital System **1.3**

Constrains the values of **digital** inputs and outputs to two distinct values (1s and 0s).

1.3 Positional Notation

Positional Notation **1.4**

Used to represent a number as a position. In base ten, every value to the left of the decimal point

$$N_r = (a_n \cdot a_{n-1}...a_1 \cdot a_0.a_{-1}...a_{-m})_r$$

$$\text{decimal value}(N_r) = \sum_{j=-m}^{n} (a_j \times r^j)$$

Where:

$r =$ is the **radix** or **base** of the number system
$N_r =$ is the number in base-**r**
$a_j =$ is a coefficient of the number with a decimal value in the range of 0 to $(r-1)$
$j =$ is the place (or position) of the digit

Base (r)	5									
Number (N_r)	4021.23_5									
N_r										
Position (j)	5	4	3	2	1	0	.	-1	-2	-3
Digit (a_j)	0	0	4	0	2	1	.	2	3	0
Position Value (r^j)	5^5	5^4	5^3	5^2	5^1	5^0	.	5^-1	5^-2	5^-3
Digit Position Value $(a_j \times r^j)$	0	0	500	0	10	1	.	0.4	0.12	0
Decimal Value $(\sum a_j \times r^j)$	511.52									

Figure 2: Base Five Representation

1.4 Binary Number System

Digital systems manipulate binary calues. The digits in a binary number are called **bits** (**b**inary dig**its**). A group of 8 bits is called a **byte**.

When in the binary number system, some unit prefixes take on a different meaning. This can be seen in Figure 3.

2^10	**K**	Binary Kilo
2^20	**M**	Binary Mega
2^30	**G**	Binary Giga
2^50	**T**	Binary Tera

Figure 3: Special Powers of 2

1.5 Number Base Conversion

Conversion of decimal numbers $\rightarrow$ base-r numbers.

1. Integer Conversion
   (a) Divide the number by the base r; save the remainder and the quotient.
   (b) Divide the quotient from the previous step by r; save the remainder and *new* quotient.
   (c) Repeat step (a) until the *new* quotient is 0.
   (d) The integer in base-r is the **remainders** written down in **reverse** order from which they were generated.
2. Fractional Conversion
   (a) Multiply the fractional portion of the number by r; save the result and the number to the left of hte decimal point
   (b) Multiply the fractional portion the result from the previous step by r; save the result and the *new* number to the left of the decimal point
   (c) Repeat step (b) until the fractional portion is zero or the desired precision is reached.
   (d) The fraction in base-r is the digits save that were left of the decimal point in order.

$$47_{10} =?_2$$
$$47 \div 2 = 23 \text{ remainder } 1$$
$$23 \div 2 = 11 \text{ remainder } 1$$
$$11 \div 2 = 5 \text{ remainder } 1$$
$$5 \div 2 = 2 \text{ remainder } 1$$
$$2 \div 2 = 1 \text{ remainder } 0$$
$$1 \div 2 = 0 \text{ remainder } 1$$
$$47_{10} = 101111_2$$

(a) Integer Conversion

$$0.3_{10} =?_2$$
$$0.3 \times 2 = 0.6$$
$$0.6 \times 2 = 1.2$$
$$0.2 \times 2 = 0.4$$
$$0.4 \times 2 = 0.8$$
$$0.8 \times 2 = 1.6$$
$$0.3_{10} = 0.0\overline{1001}_2$$

(b) Fractional Conversion

Figure 4: Fractional and Integer Conversion

This method of converting between base systems works universally. Additionally, they can be used in tandem to convert between a number with an integer and fraction part.

1.6 Octal and Hexadecimal

These are very important within digital design. Notably, both octal (base 8) and hexadecimal (base 16) are both powers of binary (base 2). When any number base is a power of another number base a quicker conversion method can be used.

1. Starting from the binary point and proceeding to the left and to the right, partition the binary number into groups of x bits, where x is the power to which the lower base is raised to reach the higher base.
    - For example, converting from octal to binary, $x = 3$ since $2^3 = 8$.
2. Convert each group of x bits into the equivalent binary representation.

2 Truth Tables

Truth tables are useful for depicting logical functions. A truth table shows the output value of a logic function or expression for every combination of values of the input variables.

In Figure 5,

Row #	i_0	i_1	i_n	Output Values
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
$2^n - 1$	1	1	1	0

Figure 5: Truth Table Template

2.1 Boolean Algebra

Boolean algebra consists of:

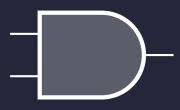
- A set of elements/variables (such as x, y, and z) that represent a condition or a physical quantity.
- Values for elements/variables (i.e. 0 = FALSE, 1 = TRUE) that represent the state of the variable.
- Primitive logical operations (**AND**, **OR**, **NOT**) for combining the values of the variables.
- A set of axioms, postulates, and theorems that define the algebra.

Boolean Operators are foundational to boolean algebra. They take a set of inputs, whether it be a single input or a pair, and return a value based on the states of those inputs. These are the logical operations that combine values of variables in boolean algebra.

AND Operator **2.1**

When all inputs are 1, the output value of **and** is 1. Otherwise, the output is 0.

AB, $A \cdot B$, $A \wedge B$, A & B, or A and B.



OR Operator **2.2**

When any input is 1, the output value of **or** is 1. Otherwise, the output is 0.

$X + Y$, $X\#Y$, $X|Y$, $X \vee Y$, or $XorY$.

NOT Operator **2.3**

The output of **not** is the *inverse* of the output; 1 becomes 0, and 0 becomes 1.

$\overline{Y}$, Y', $!Y$, $\neg Y$, or not Y.

XOR Operator **2.4**

The output of **XOR** is 1 is **only** one of the inputs is 1, but not both and not neither.

$A \oplus B$ or A XOR B.

NAND and NOR Operators **2.5**

By adding **N** to the beginning of a gate, the gate is converted from what it was to the inverse of it. **AND** becomes **NOT AND**, **OR** becomes **NOT OR**. Functionally, the outputs of either of the gates is just the inverse of the original gate. To convert a drawn AND or OR gate to NAND or NOR respectively, simply add a circle to the output.

Just like "regular" algebra, boolean algebra can be operated on through methods such as factoring, as well as logical properties such as those listen in Figure 6. Using these, we can simplify complex equations to simpler ones:

$$
\begin{aligned}
F &= (\overline{x} \wedge \overline{y} \wedge z) \vee (x \wedge y \wedge z) \vee (\overline{x} \wedge y \wedge z) \vee (x \wedge \overline{y} \wedge z) \\
F &= (\overline{x} \wedge \overline{y} \wedge z) \vee (\overline{x} \wedge y \wedge z) \vee (x \wedge y \wedge z) \vee (x \wedge \overline{y} \wedge z) \\
F &= [\overline{x} \wedge z(\overline{y} \vee y)] \vee [x \wedge z(y \vee \overline{y})] \\
F &= (\overline{x} \wedge z \wedge 1) \vee (x \wedge z \wedge 1) \\
F &= (\overline{x} \wedge z) \vee (x \wedge z) \\
F &= z \wedge (\overline{x} \vee x) \\
F &= z \wedge 1 \\
F &= z
\end{aligned}
$$

Thus, through algebraic manipulation, it can be seen that the original equation:

$$F = (\overline{x} \wedge \overline{y} \wedge z) \vee (x \wedge y \wedge z) \vee (\overline{x} \wedge y \wedge z) \vee (x \wedge \overline{y} \wedge z)$$

is equivalent to the direct state of z:

$$F = z$$

Mathematical Representation	Name of the Law
$p \vee \neg p \equiv T$	Law of Excluded Middle
$p \wedge \neg p \equiv F$	Law of Non-Contradiction
$p \wedge T \equiv p$	Identity Laws
$p \vee F \equiv p$	
$p \wedge F \equiv F$	Domination Laws
$p \vee T \equiv T$	
$p \vee p \equiv p$	Idempotent Laws
$p \wedge p \equiv p$	
$\neg\neg p \equiv p$	Laws of Double Negation Elimination
$p \vee q \equiv q \vee q$	Commutative Laws
$p \wedge q \equiv q \wedge q$	
$(p \vee q) \vee r \equiv p \vee (q \vee r)$	Associative Laws
$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	Distributive Laws
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	
$\neg(p \wedge q) \equiv \neg p \vee \neg q$	DeMorgan's Laws
$\neg(p \vee q) \equiv \neg p \wedge \neg q$	
$p \vee (p \wedge q) \equiv p$	Absorption Laws
$p \wedge (p \vee q) \equiv p$	
$p \Rightarrow q \equiv q \vee \neg p$	$\vee$ Restatement of Implication
$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$	Contraposition
$(p \Rightarrow q) \wedge (p \Rightarrow r) \equiv p \Rightarrow (q \wedge r)$	Conjunction of Implications
$(p \Rightarrow r) \wedge (q \Rightarrow r) \equiv (p \vee q) \Rightarrow r$	Will Show This Below
$(p \Rightarrow q) \vee (p \Rightarrow r) \equiv p \Rightarrow q \vee r$	Disjunction of Implications
$(p \Rightarrow r) \vee (q \Rightarrow r) \equiv (p \wedge q) \Rightarrow r$	
$p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$	Conjunction of Implications
$p \Leftrightarrow q \equiv \neg p \Leftrightarrow \neg q$	Negation Restatement
$\neg(p \Leftrightarrow q) \equiv \neg p \Leftrightarrow q$	Negation

Figure 6: List of Logical Equivalences

Definitions

A **literal** is a variable or the complement of a variable.

A **Product Term** is a single literal or the logical product of two or more literals.

A **Sum of Products (SOP)** expression is a logical sum of product terms.

A **Sum Term** is a single literal or a logical sum of two or more literals.

A **Product of Sums (POS)** expression is a logical product of sum terms.

A **Normal Term** is a product or sum term in which no variable appears more than once.

2.2 Canonical Notation

Product Term **2.6**

Denoted with m_n, a product term is the product (and) of each input. Inputs default to 1, with 0 being the complement of the input.

For example, if $A = 1$, $B = 0$, $C = 1$, then $m_5 = A\overline{B}C$. $n = 5$ because 101 occurs on the 5^{th} row. See Figure 7.

Sum Term **2.7**

Denoted with M_n, a sum term is the sum (or) of each input. Input defaults to 0, with 1 being the complement of the input.

For example, if $A = 1$, $B = 0$, $C = 1$, then $M_5 = \overline{A} + B + \overline{C}$. $n = 5$ because 101 occurs on the 5^{th} row. See Figure 7.

Canonical Notation **2.8**

Either Sum of Products (SOP) or Product of Sums (POS), canonical notation is used to express a boolean equation by showing where the outputs of a single type are.

In Canonical Notation, the **Sum of Products** shows where the 1s are in the truth table, with the **Product of Sums** showing where the 0s are.

Row #	A	B	C	Product Term	Sum Term
0	0	0	0	$\overline{A}\overline{B}\overline{C} = m_0$	$A + B + C = M_0$
1	0	0	1	$\overline{A}\overline{B}C = m_1$	$A + B + \overline{C} = M_1$
2	0	1	0	$\overline{A}B\overline{C} = m_2$	$A + \overline{B} + C = M_2$
3	0	1	1	$\overline{A}BC = m_3$	$A + \overline{B} + \overline{C} = M_3$
0	1	0	0	$A\overline{B}\overline{C} = m_4$	$\overline{A} + B + C = M_4$
1	1	0	1	$A\overline{B}C = m_5$	$\overline{A} + B + \overline{C} = M_5$
2	1	1	0	$AB\overline{C} = m_6$	$\overline{A} + \overline{B} + C = M_6$
3	1	1	1	$ABC = m_7$	$\overline{A} + \overline{B} + \overline{C} = M_7$

Figure 7: Product and Sum Terms

$$F = \sum_{A,B,C} (1, 3, 4, 6) \qquad F = \prod_{A,B,C} (0, 2, 5, 7)$$

$$F(A, B, C) = \sum m(1, 3, 4, 6) \qquad F(A, B, C) = \prod m(0, 2, 5, 7)$$

(a) List of Product Terms

(b) List of Sum Terms

Canonical Sum (SOP):

$$F(A, B, C) = m_1 + m_3 + m_4 + m_6 = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}\overline{C} + AB\overline{C}$$

Canonical Product (POS):

$$F(A, B, C) = m_0 + m_2 + m_5 + m_7 = (\overline{A} + \overline{B} + C)(A + \overline{B} + C) + A\overline{B}\overline{C} + AB\overline{C}$$

A **Don't Care** is any row that has input that is unimportant for the relevant use case. To notate a "Don't Care", *dc* is followed by each unimportant row in parentheses:

$$F(A, B, C) = \prod m(0, 2, 5, 7) + dc(1, 3, 4, 8)$$

---

3 Karnaugh Maps

A **Karnaugh Map** is a graphical representation of a boolean function's truth table. A function of n variables will have a map of 2^n cells, with each cell representing a unique combination of the inputs of the function, similar to a truth table.

Grey Code **3.1**

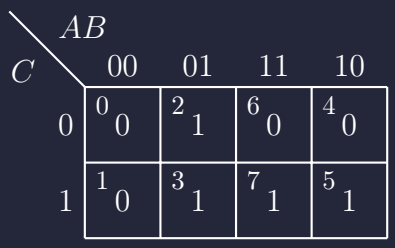
A method of sequencing binary data such that each successive number only changes by a single bit. For example, with a 3-digit binary number, the sequence would follow as: 000, 001, 011, 010, 110, 100, etc.

When constructing a Karnaugh Map, the row and column labels must follow a **Grey Code** sequence, this is so that the map is able to "wrap around" (the end of a row/column is adjacent to the beginning).

Each cell is doubly labeled, first with its corresponding row number in the truth table in the upper right, and second with its logical value in the center.

Row	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

(a) Truth Table



(b) Karnaugh Map

Figure 9: Karnaugh Map Synthesis

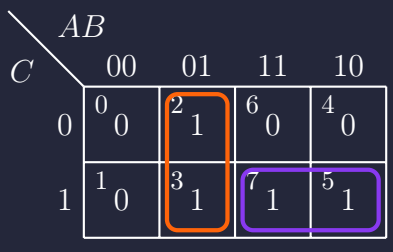


Figure 10: Grouping Implicants

To construct a SOP expression from a Karnaugh Map, 1-cells are group together into **prime implicants** in groups of size 2^k. Once grouped, expressions can be derived for each prime implicant based on the input values that don't change throughout the group.

$$F(A, B, C) = \sum(2, 3, 5, 7) = \overline{A}B + AC$$

Implicant **3.2**

A product term of a complete function such that the product term implies the function. A **prime implicant** is the largest set that can be made to include any specific 1-cell in a SOP-expression. An **essential prime implicant** is a prime implicant that covers one or more *distinguished 1-cells*.

Similarly, to construct a POS expression from a Karnaugh Map, 0-cells are group together into **prime implicates** in groups of size 2^k. Once grouped, expressions can be derived for each prime implicant based on the input values that don't change throughout the group.

$$F(A,B,C) = \prod(0,1,4,6) = (A+B)(\overline{A}+C)$$

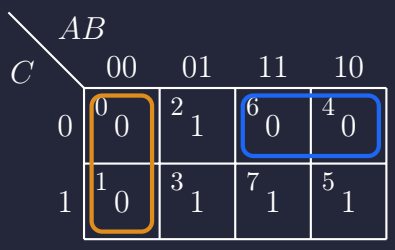


Figure 11: Grouping Implicates

Implicate **3.3**

A sum term of a complete function such that the sum term implies the function.

Distinguished 1/0-Cell **3.4**

An input combination that is only covered by a single prime implicant/implicate.

$$f(w,x,y,z) = \sum m(0,1,2,4,5,6,9,10,11,13,14,15)$$

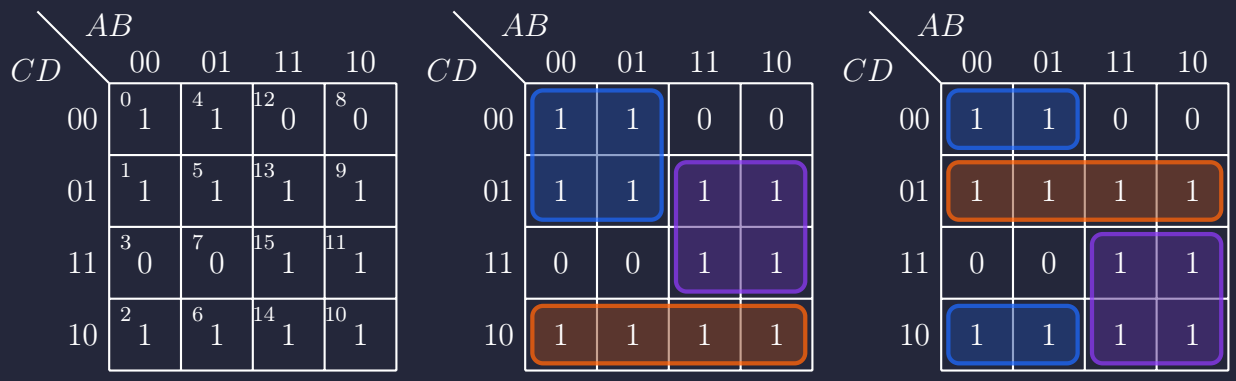


Figure 12: Boolean Expression with Multiple Solutions

It is possible that a boolean expression can be grouped into a Karnaugh Map in different ways, providing multiple solutions. Assuming the maps are grouped correctly, there is no single best solution.

When constructing a Karnaugh Map of an expression that includes "don't cares", two rules should be followed to construct the minimum SOP or POS expression:

- Include don't cares in the prime implicates/implicants to make them as big as possible
- Don't create an implicate/implicant containing **only** don't cares

$$F(w,x,y,z) = \sum(1,2,3,5,7) + d(10,11,12,13,14,15)$$

yz \ wx	00	01	11	10
00	$^{0}0$	$^{4}0$	$^{12}*$	$^{8}0$
01	$^{1}1$	$^{5}1$	$^{13}*$	$^{9}0$
11	$^{3}1$	$^{7}1$	$^{15}*$	$^{11}*$
10	$^{2}1$	$^{6}0$	$^{14}*$	$^{10}*$

yz \ wx	00	01	11	10
00	0	0	*	0
01	1	1	*	0
11	1	1	*	*
10	1	0	*	*

Figure 13: Don't Care Karnaugh Map

4 Binary Systems and Encoding

4.1 Number Format

Inside of a computer, numbers are represented as a stream of digits in the computer's memory. The format the number is stored must be known for a computer to be able to use the value properly.

Number Format **4.1**

The components of the way a number is expressed that determine how it is encoded and decoded. These components include:

- Number base
- Number of digits in the number
- Integer or floating point
- Where the radix point is
- The complement system, if any, being used

Signed Magnitude

Binary Signed-Magnitude

Complement Systems

A complement system *negates* a number by taking the complement of it according to the rules of a specific complement system.

Given a (fixed) number of ways we uniquely can represent values in a number system, we will assign a portion of the representations to represent negative (or inverse) values and develop a system of arithmetic using this new number system.

Complement systems are useful because addition of One's or Two's Complemented numbers is done simply by adding each digit as usual. This type of binary addition is sign-agnostic.

One's Complement **4.2**

The complement of a number according to One's Complement follows the pattern of subracting each digit of the number from the radix (r) minus one ($r-1$).

$0 = 0000_2$	$2 = 0010_2$	$7 = 0111_2$	$5 = 0101_2$
$-0 = 1111_2$	$-2 = 1101_2$	$-7 = 0111_2$	$-7 = 1010_2$

One's complement is a symmetric number system. Both 0 and -0 have different expressions under One's complement despite being equivalent. Thus, the system is symmetric in terms of the number of negatives and positives that can be expressed given a bit size. Two's complement is asymmetric as it expresses one more negative than it does positive.

Two's Complement **4.3**

Given the number a, the radix r, and the number of digits n, the Two's complement of a is defined as any of the following:

$$\begin{aligned}(-a)_2 &= [\text{one's complement of } a] + 1 \\ &= r^n - a \\ &= [\text{each digit of } a \to (r-1) - \text{digit}] + 1 \\ &= \text{complement each digit to the left of the rightmost one}\end{aligned}$$

$0 = 0000_2$	$2 = 0010_2$	$7 = 0111_2$	$5 = 0101_2$
$-0 = 0000_2$	$-2 = 1110_2$	$-7 = 1001_2$	$-7 = 1011_2$

Decimal	Two's Complement	One's Complement	Signed Magnitude
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	—	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	—	—

Figure 14: Binary Number Systems in 4-Bits

4.2 Binary Encoding

ASCII Character Codes

ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) is a 7-bit coding standard that identifies letters, numbers, and various symbols of a teletype machine. With 2^7, or 128, possible combinations, that many possible values can be encoded as binary.

When encoding in ASCII, a **parity bit** can be added to the front to ensure that there is a number of 1s in the code that is either always even or always odd. This is done as a check to make sure that the codes being sent and received make sense.

5 Adders and Comparators

5.1 Adder

An **Adder** performs addition of bits using the rules for addition defined in the binary number system. With an adder, subtraction (addition of the complement), multiplication (repeated addition), and division (repeated addition of the complement) can all be performed.

Given two inputs, a **Half Adder** combines the bits of each input to produce two outputs: the sum bit (S) and the carry bit (C). With any two bits, the sum will only be 1 if one of the two is a 1, thus a XOR gate would be used for the sum. The carry will only result in a 1 if both inputs are 1, thus an AND gate would be used. The circuit is shown in Figure 16.

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 15: Half Adder Truth Table

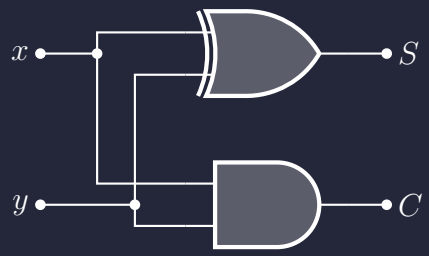
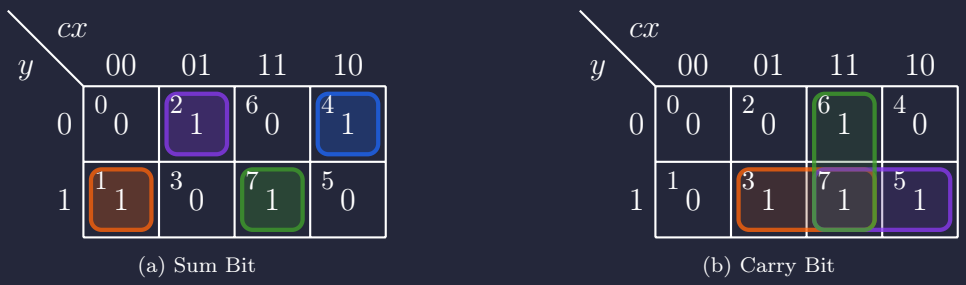


Figure 16: Half Adder

A **Full Adder** performs the addition of three digits. Conceptually, adders can be linked in a chain such that each adder has three inputs: two unique inputs and the carry bit from the previous link in the chain. Thus, a full adder must accommodate three inputs. A full adder can be seen in Figure 18.

The Karnaugh Map representation of each term is as in Figure 17. Two different outputs are defined: one for the sum bit (S) and one for the carry bit (C). Based on these K-Maps: $S = c \oplus (x \oplus y)$ and $C = xy + yc + cx$.



(a) Sum Bit

(b) Carry Bit

Figure 17: Karnaugh Maps of the Sum and Carry Bits

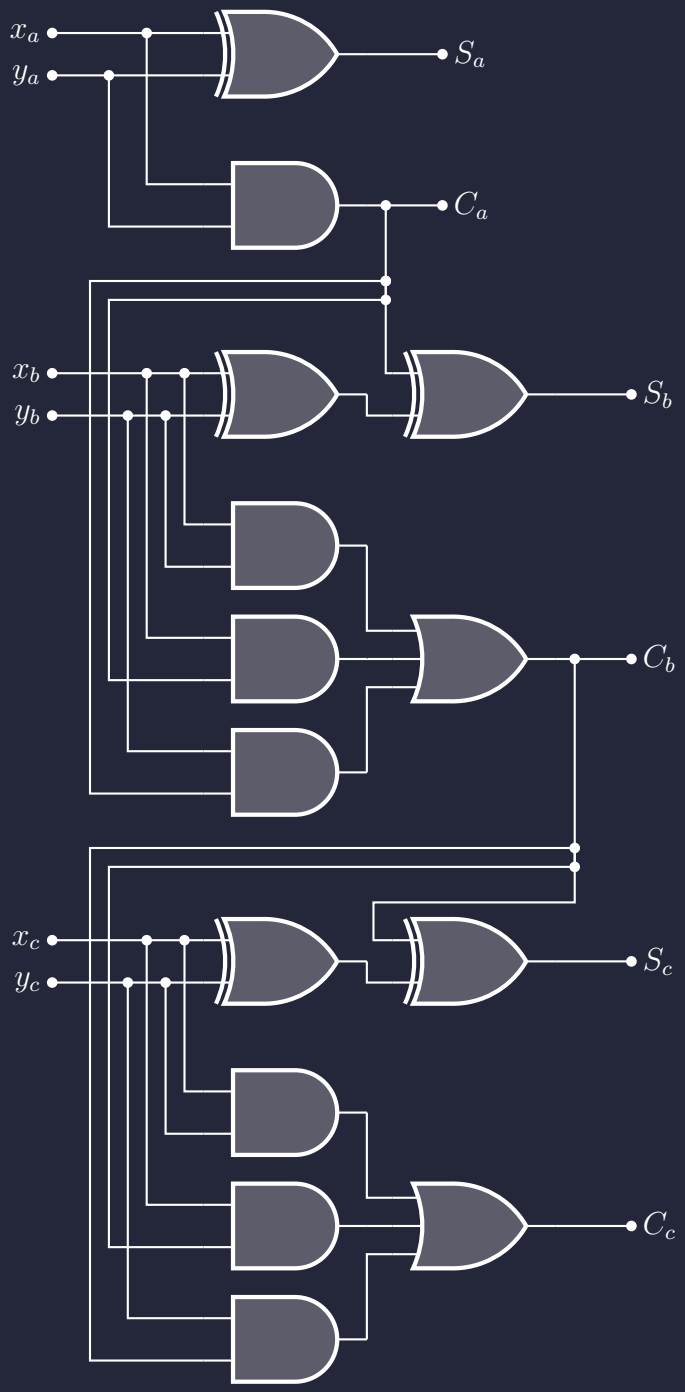


Figure 18: Full Adder

6 Latches and Flip-Flops

6.1 Latch Gate

Latch **6.1**

A latch is a bi-stable memory element, meaning that it can store data in two different states (1 or 0). Additionally, it is also level-sensitive, meaning that the state that it stores is dependent on the inputs to the latch.

A latch can be implemented in many ways, one of which being with two NOR gates as seen in Figure 19.

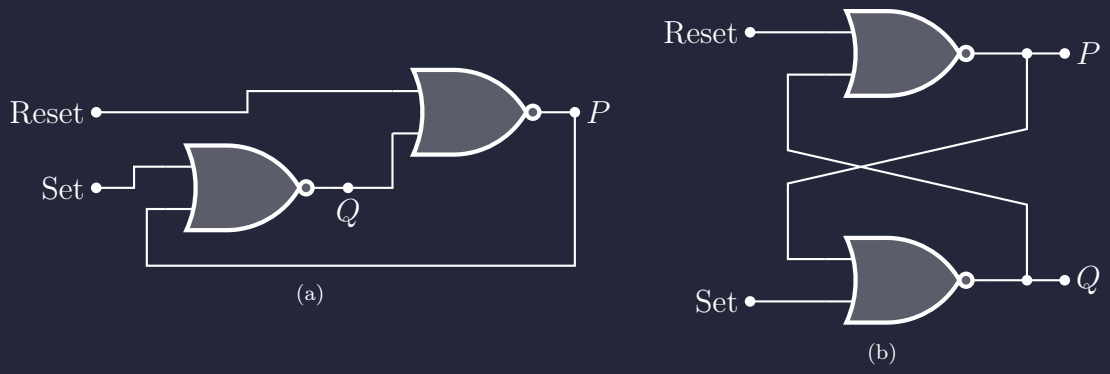


Figure 19: Latch Implementation With Nor Gates

Operating a Latch Gate

The operation of a latch follows two simple rules:

1. For all states, $P \equiv \overline{Q}$. P is always the complement of Q
2. The inputs R and S must never change simultaneously

Following these simple rules, manipulating the values of R and S changes the outputs of P and Q according to the following table.

S	R	P	Q
0	0	P	Q
0	1	0	1
1	0	1	0
1	1	0	0

(a) Characteristic Table

$$Q^+ = S \vee (\overline{R} \wedge Q)$$

(b) Characteristic Equation

Figure 20

Consider Figure 21. Starting in the initial state of $R, S = 0$, changing the value of R to 1 results in the output of P changing from 0 to 1, and the output of Q changing from 1 to 0.

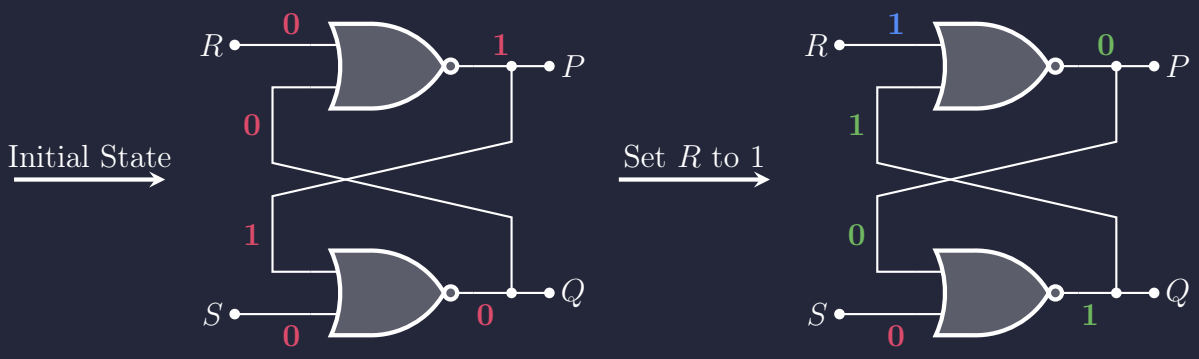


Figure 21

However, going from the resulting state in Figure 21 and changing R back to 0, the state of P and Q are unaffected. Thus, we have a situation in which the inputs of $R = 0$ and $S = 0$ correspond to outputs of $P = 0$ and $Q = 1$. Previously, the same inputs corresponded to the complement of the outputs.

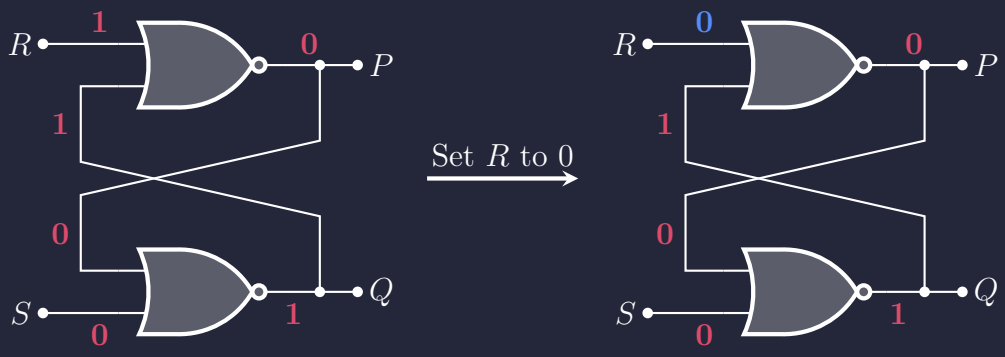


Figure 22

Starting from the resulting state of Figure 21, changing R back to 0 has no effect on the outputs of the latch. This is a demonstration of the latch's "latching" behavior; in certain states, it retains a memory of its outputs independentally of its inputs.

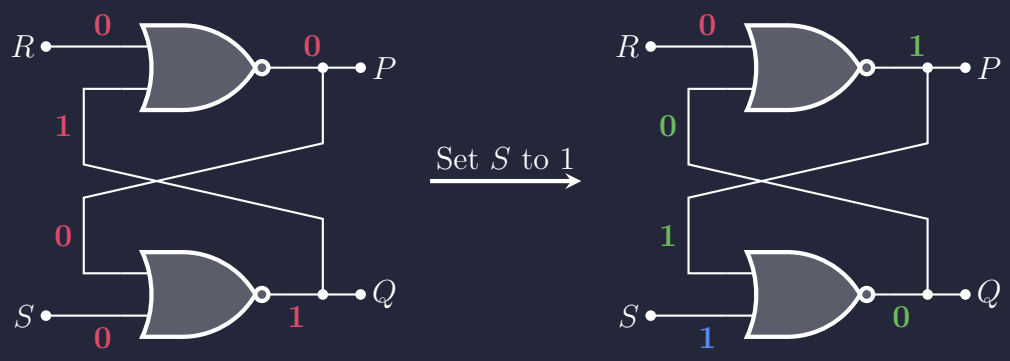


Figure 23

From the result of Figure 22, raising S to 1 has the effect of complementing both of the outputs; P goes to 1 from 0, and Q goes to 0 from 1.

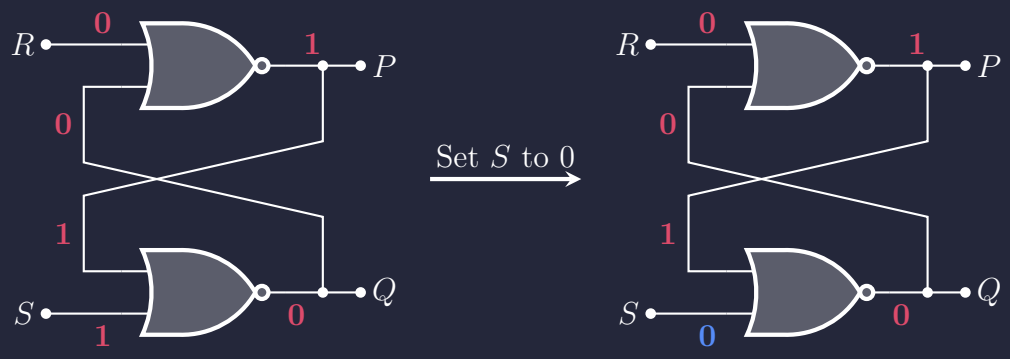


Figure 24

After lowering S back to 0, again, the latch retains its outputs.

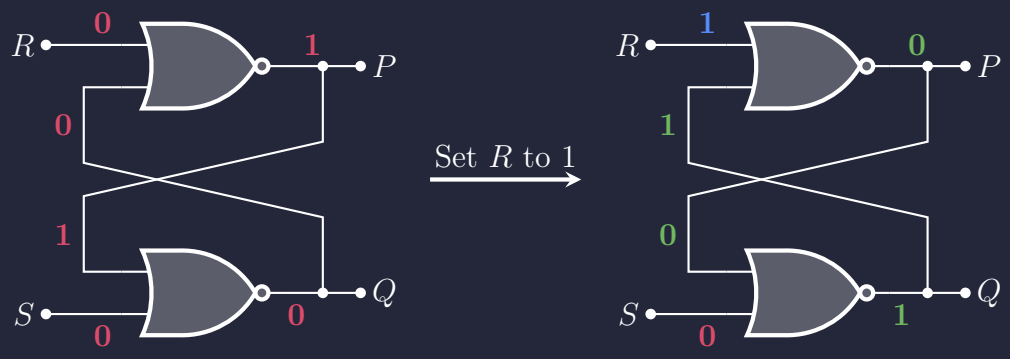


Figure 25

Lastly, by raising R back to 1, the outputs of the latch are the same as they were in the initial state. If R were to be lowered again to 0, the latch would retain its memory of the state and would find itself back in the same initial state as it was in Figure 21.

Notice that the outputs only ever change when either S or R is raised to 1. When either is lowered, the outputs remain constant.

Latches are useful for signal "debouncing". When a physical button is pressed, often there is excess kinetic energy that causes to circuit to flicker, leading to several input changes despite the button being pressed only a single time.

Figure 26: Bouncing Signal

Since a latch only changes its output when an input is **raised**, but not when it is lowered, only the initial rise will change the output. Each lowering of the input due to the bouncing is negated.

Figure 27: Debounced Signal

6.2 Gated Latch

Clk	S	R	P	Q
0	x	x	P	Q
1	0	0	P	Q
1	0	1	0	1
1	1	0	1	0
1	1	1	x	x

(a) Characteristic Table

$$Q = Clk\big(S + \overline{R} \cdot Q\big) + \overline{Clk} \cdot Q$$

(b) Characteristic Equation

Figure 28

A gated latch operated similarly to a regular latch in that it uses two inputs R and S to manage a circuit capable of storing memory. However, the difference is that it connects each input through an AND gate with a clock.

Since the state of a latch can only change when R or S is raised, and R or S can only be raised when the clock ticks, the changing of P or Q is essentially restricted to when the clock ticks. In other words, the gated latch functions identically to the regular latch, but it only can change states during the tick of a clock.

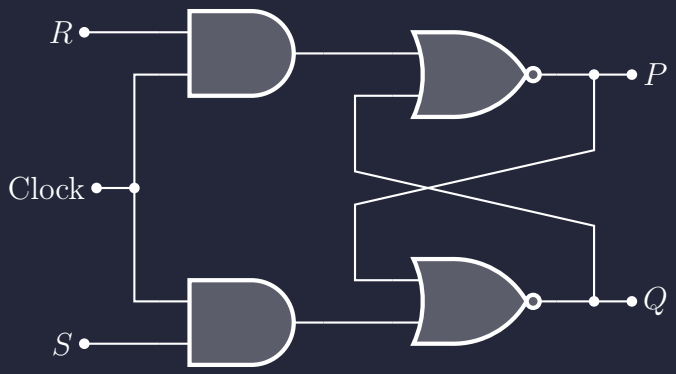


Figure 29: Gated Latch

6.3 Timing Diagrams

Timing Diagram **6.2**

A timing diagram is a graphical view of a circuits behavior that shows how the outputs of a circuit change in response to the inputs which are varying as a function of time.

For a circuit such as a three-input AND gate, a possible timing diagram would look like Figure 30.

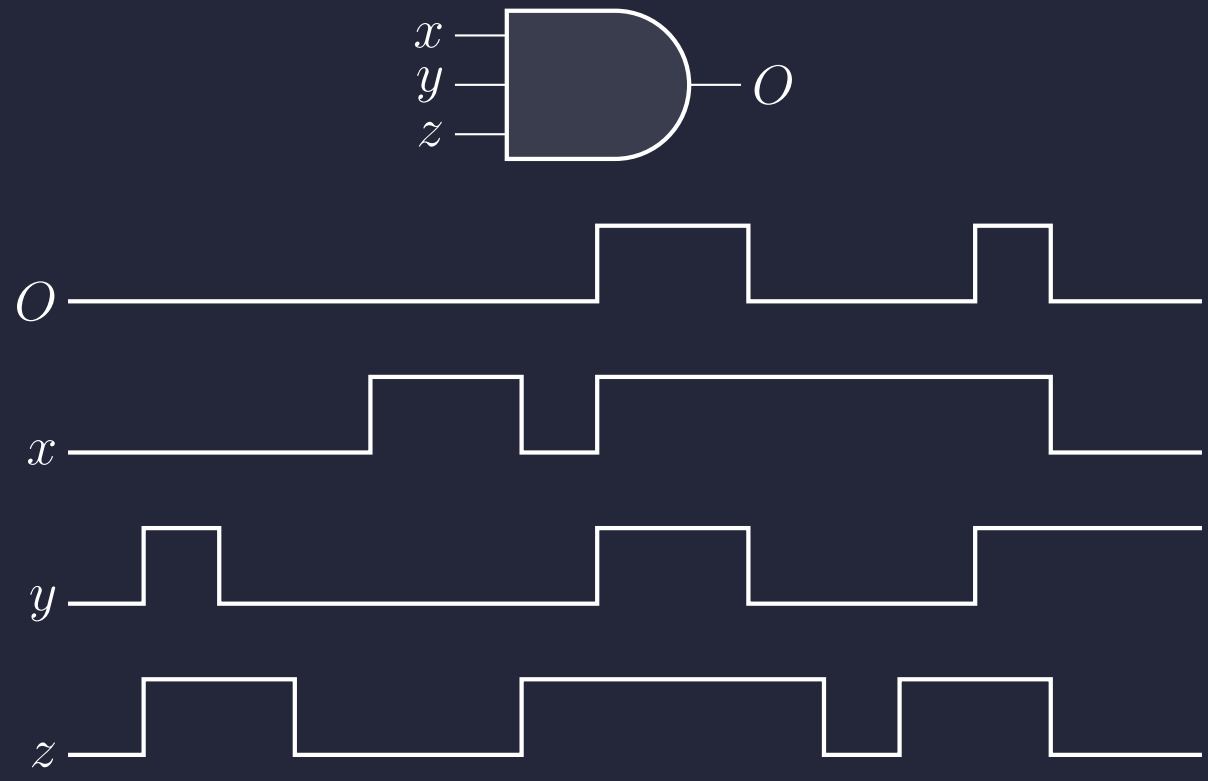


Figure 30: AND Timing Diagram

When given arbitrary high and low values for the inputs x, y, and z, the output O is only high when all three inputs are high. This is just a visual representation of the operation of an AND gate.

With more complex circuits, timing diagrams get a little harder to parse. However, the same basic concept applies: each line for each input, output, or state should logically correspond to each other line of the diagram. Consider the timing diagram for some D Flip-Flops in Figure 31.

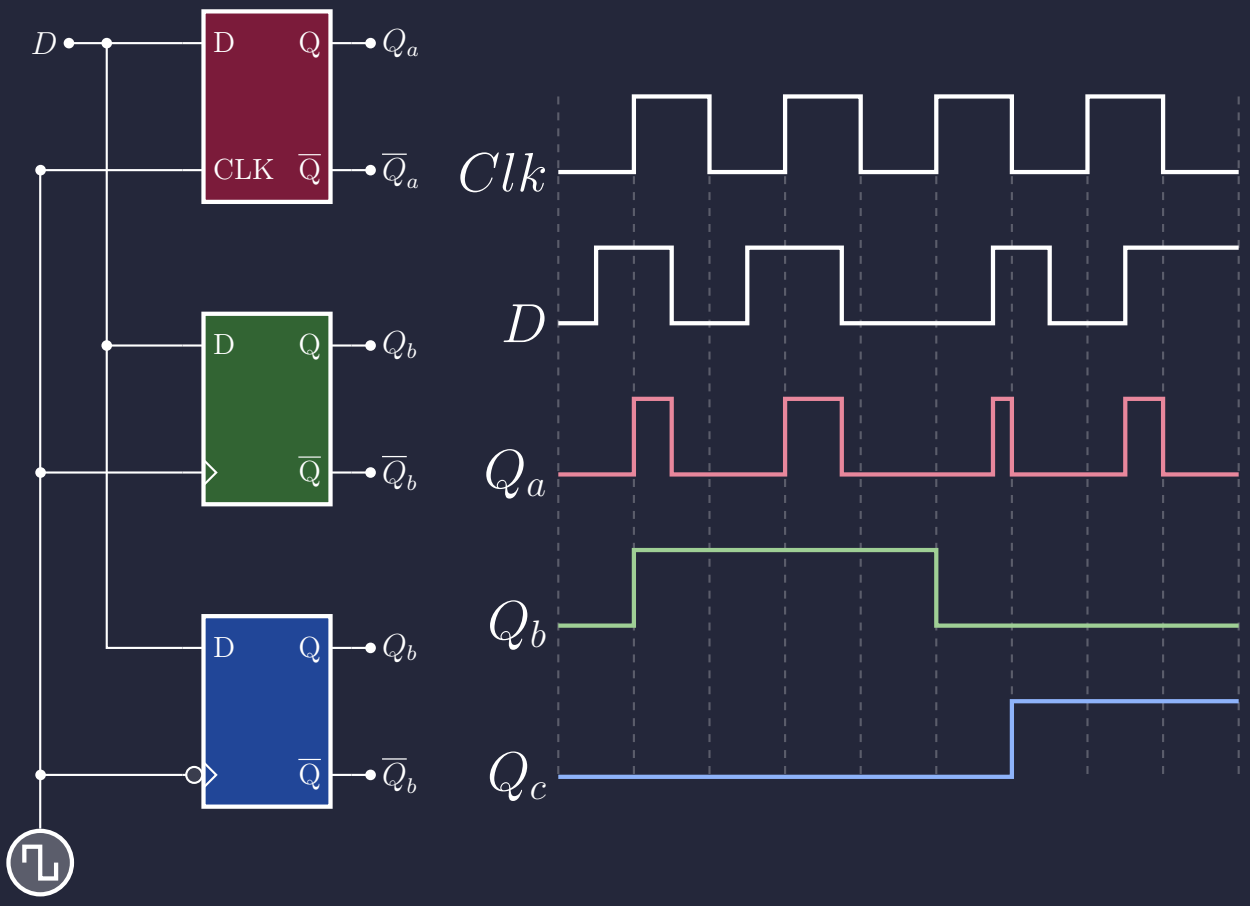


Figure 31: Timing Diagram for D Flip-Flops

Flip-flop a has a direct Clk input, meaning it just takes the value of the clock and combined it with input D. Flip-flop b triggers only on the rising edge, and flip-flop c triggers only on a falling edge.

7 Sequential Circuits

A **combinational circuit** is the result of interconnecting logic gates. The functioning of a combinational circuit is instentaneous. The current state of the inputs entirely determines the state of the outputs.

Sequential Circuit **7.1**

Contains **combinational circuits** and **memory elements** with a feedback path from the memory element to the combinational circuit.

The output of a sequential circuit will be dependent on **both** the current inputs as well as the memory, or past history, of the circuit stored in the memory elements. The data stored in the memory is referred to as **the state** of the circuit.

7.1 Synchronous Sequential Circuits

A **synchronous sequential circuit** is a circuit whose behavior can be defined from the knowledge of its signals at discrete instants of time. A clock signal is used to control and synchronize the circuit behavior. These circuits are called **clocked sequential circuits**.

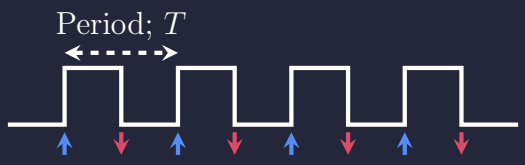


Figure 32: Clock Signals

These clock signals "ticks" each time there is a change from low to high or from high to low. A circuit can use either the leading (↑) or trailing (↓) edge of a pulse to mark the "tick" of the clock pulse.

7.2 Sequential Circuit Analysis

Analyzing a sequential circuit follows three steps:

1. Derive the **state equations** and the **output equations** for the circuit
   - State equations model the next state as a function of the current states and any inputs
   - Output equations model the outputs as a function of the current states and any inputs
2. Derive the **transition table** and the **state table** for the circuit

3. Derive the **state diagram** for the circuit

7.2.1 State and Output Equations

8 Sequential Circuit Synthesis

The steps to synthesizing a circuit from a description of its behavior are as follows:

1. Draw a **state diagram**
2. Construct a **state/output table**, then assign unique binary values to each state and output in the table
3. Create a **transition/output table**
4. Choose a flip-flop
5. Construct an **excitation table**
6. Use K-Maps (or another method) to **derive the logic equations**
7. Construct the circuit

To illustrate this process more clearly, the following design specifications will be used to derive a fully functioning circuit. The circuit will operate as a stop light rotating through green, red, and yellow:

1. All changes in the circuit occur on the positive edge of a clock signal
2. The circuit has three outputs (r, y, and g)
   (a) The output r is equal to 1 if the previous output was y
   (b) The output y is equal to 1 if the previous output was g
   (c) The output g is equal to 1 if the previous output was r or there is no previous output

Clockcycle	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
r	0	0	1	0	0	1	0	0	1	0	0
y	0	1	0	0	1	0	0	1	0	0	1
g	1	0	0	1	0	0	1	0	0	1	0

8.1 State Diagram

In a simple circuit with no inputs (other than a clock cycle), the conditions for each state are straightforward: if the current state is red (S_2), then the next state is green (S_0).

Note that a starting state is required, otherwise the initial state of the circuit will be undefined behavior.

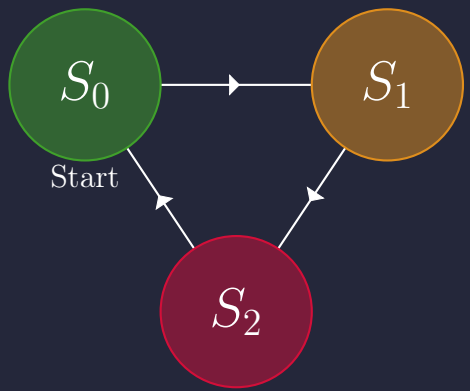


Figure 33: State Diagram

8.2 State/Output Table

S	S^+	O
S_0	S_1	g
S_1	S_2	y
S_2	S_0	r

(a) Abstract Representation

S		S^+		O		
0	0	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	1
1	1	*	*	*	*	*

(b) Binary Representation

Figure 34: State/Output Table

Based on the state diagram (Figure 33), a state/output table can be derived. This is essentially a tabular representation of the state diagram, showing the current state (S), the next state (S^+), and the output corresponding to each state (O).

In Figure 34a, a simple and abstracted table is shown, directly mirroring the information in the state diagram. In Figure 34b, each state and output is replaced with actual binary values to represent them.

8.3 Transition/Output Table

S		S^+		O		
S_0	S_1	S_0^+	S_1^+	O_0	O_1	O_2
0	0	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	1
1	1	*	*	*	*	*

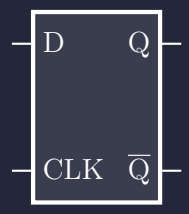
Figure 35: Transition Table

Notably, this transition table is essentially identical to the table in Figure 34b. This is because, when assigning binary values to each state/output in the circuit, the transitions become apparent anyway.

8.4 Choose a Flip-Flop

There are many Flip-Flops to choose from. However, three important ones (each with its own derivations) exist: the **T Flip-Flop**, the **D Flip-Flop**, and the **JK Flip-Flop**.

8.4.1 D Flip-Flop



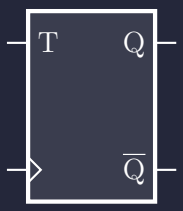
D	C	Q^+	$\overline{Q}^+$
0	↑	0	1
1	↑	1	0
X	1	Q	$\overline{Q}$

The D Flip-Flop is described by the characteristic equation:

$$Q^+ = D$$

The D Flip-Flop captures the value of the D-input at a definite portion of the clock cycle (generally at the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change.

8.4.2 T Flip-Flop



T	C	Q^+	$\overline{Q}^+$
0	↑	Q	$\overline{Q}$
1	↑	$\overline{Q}$	Q
X	1	Q	$\overline{Q}$
X	0	Q	$\overline{Q}$

If the T input is high, the T Flip-Flop changes state ("toggles") whenever the clock input rises. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation:

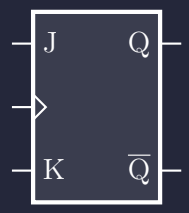
$$Q^+ = T \oplus Q = (T \wedge \overline{Q}) \vee (\overline{T} \wedge Q)$$

The toggle only occurs on the rising edge of a clock signal, meaning any input is meaningless when not paired with a rising edge. This is seen in rows 2 and 3 of the table.

8.4.3 JK Flip-Flop

The JK Flip-Flop, augments the behavior of the SR flip-flop (J: Set, K: Reset) by interpreting the $J = K = 1$ condition as a "flip" or toggle command.

$J = 1, K = 1 \rightarrow$ Toggle
$J = 0, K = 1 \rightarrow$ Reset
$J = 1, K = 0 \rightarrow$ Set
$J = 0, K = 0 \rightarrow$ No Change



J	K	C	Q^+	$\overline{Q}^+$
0	0	↑	Q	$\overline{Q}$
0	1	↑	0	1
1	0	↑	1	0
1	1	↑	$\overline{Q}$	Q
X	X	0	Q	$\overline{Q}$
X	X	1	Q	$\overline{Q}$

Similarly to the T Flip-Flop, the changes can only occur on the rising edge of a clock signal, making any input outside of that rising edge meaningless. The JK Flip-Flop is described by the characteristic equation:

$$Q^+ = (J \wedge \overline{Q}) \vee (\overline{K} \wedge Q)$$

For the purposes of this circuit, the **D Flip-Flop** is sufficient and will be used.

8.5 Construct an Excitation Table

An excitation table is shows the information required to change (or "excite") the circuit into moving from one state to the next. Since flip-flops all function differently, an excitation table for a circuit cannot be generalized to any flip-flop.

For the current design, the following excitation table can be constructed from the transition/output table.

S		S^+		O		
S_0	S_1	S_0^+	S_1^+	O_0	O_1	O_2
0	0	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	1
1	1	*	*	*	*	*

(a) Transition/Output Table

S		S^+		D		O		
S_0	S_1	S_0^+	S_1^+	D_0	D_1	O_0	O_1	O_2
0	0	0	1	0	1	1	0	0
0	1	1	0	1	0	0	1	0
1	0	0	0	0	0	0	0	1
1	1	*	*	*	*	*	*	*

(b) Excitation Table

Clearly, both the transition table and excitation table are incredibly similar. The only difference is the addition of the D column (referring to each of the two D Flip-Flops required in the design). This column holds the required inputs to each of the Flip-Flops to properly transition the circuit between states.

8.6 Derive the Logic Equations

From the excitation table in Figure 39b, a K-Map can be created for each of the outputs as well as each of the states. From the K-Maps, the state and output equations can be derived.

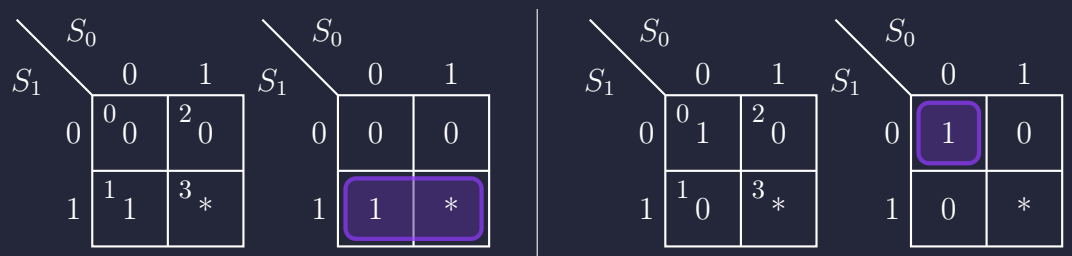
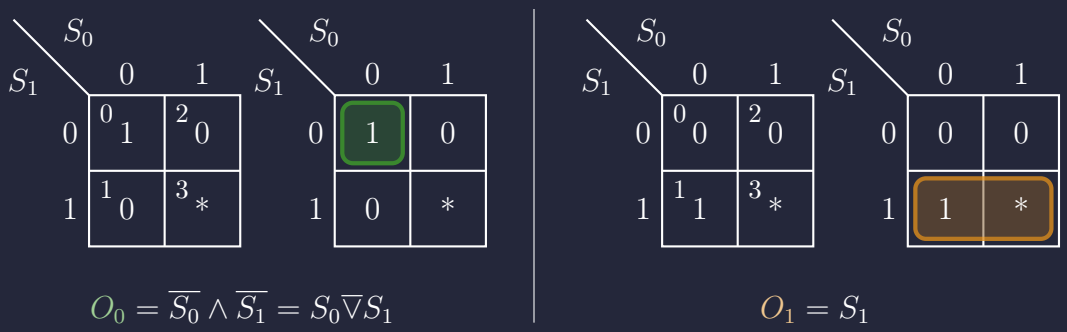


Figure 40: State K-Maps



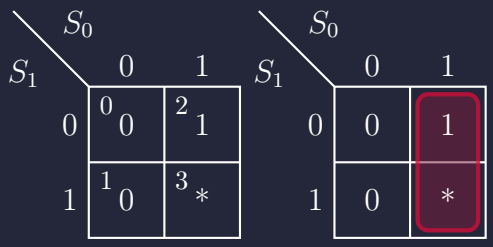


Figure 41: State K-Maps

8.7 Construct the Circuit

Using the equations derived in the previous step, all that is left is to construct the circuit accordingly:

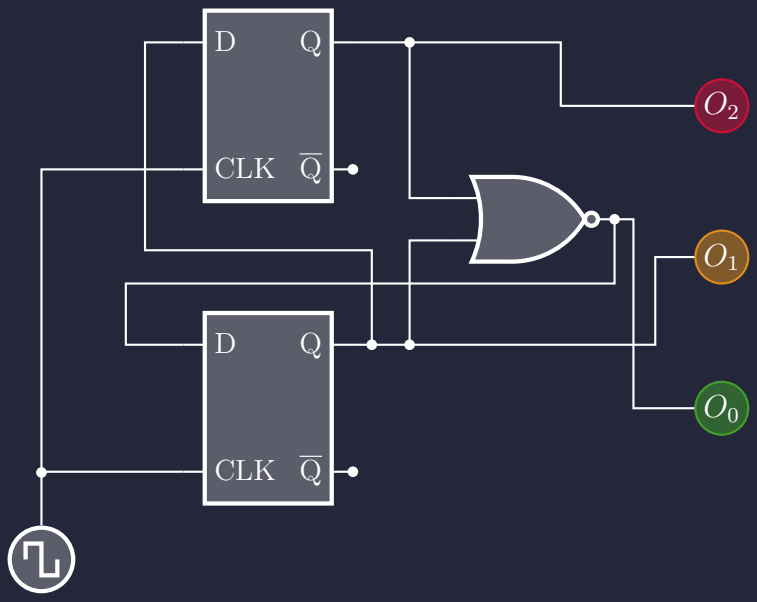


Figure 42: Fully Constructed Circuit