PDHonline Course E326 (2 PDH)

Introduction to Digital Logic

Instructor: John Tuman, PE

2020

PDH Online | PDH Center

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone: 703-988-0088
www.PDHonline.com

An Approved Continuing Education Provider

An Introduction to Digital Logic

John Tuman, P.E.

COURSE CONTENT

1. Introduction

This course provides a concise overview of digital logic design. For those already familiar with logic design this course will serve as a review, for those new to the topic the course will serve as an introduction. This course has no prerequisite class; however a basic understanding of Boolean algebra may be helpful.

The course will first introduce the logical operations AND, NOT, NAND, OR, NOR, XOR, XNOR as well as their Boolean expression and schematic symbols. A few examples will be provided to illustrate how to determine the output behavior of a given logic circuit then how to develop logic given a truth table or a Boolean expression.

The course then provides an overview of TTL and CMOS chip families highlighting key differentiators. The popular "74" series of chip identifiers will be introduced as well as common TTL and CMOS sub-families. Differences in speed and power dissipation between family and sub-family types are examined by reviewing excerpts from manufacturer specification sheets.

Next the course covers some fundamental design principles like positive and negative logic, input/output voltage switching thresholds, propagation delay, and fan-out.

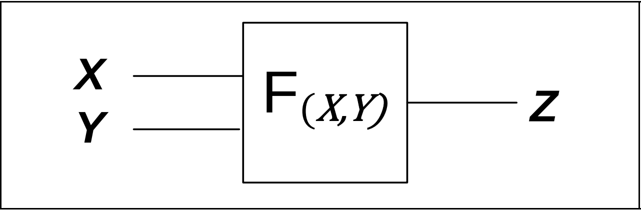
Building from the information provided the course introduces the half and full-adder to demonstrate a solution to a commonly encountered requirement in logic design; binary addition. Lastly the course concludes by demonstrating a simple circuit to perform parity generation.

2.0 Logic Gates Summary

The basic buildings blocks of digital logic design are logic gates. A logic gate will accept one or more inputs, perform a logical operation and produce a single output as shown in figure 2.0.1.

An input to a logic gate can have only two valid states, these two discrete states are true and false. The nomenclature which will used throughout this course will be to denote the true state as 1 (one) and the false condition as a 0 (zero). It is important to emphasize that used in the context of logic design 1 (one) and 0 (zero) represent two discrete logic states and should not be confused with binary numbers.

In a like manner, the output from a logic gate also has two valid states, 1 (true) or 0 (false).



- The output of a digital logic gate ***Z*** is a function of its inputs ***X*** & ***Y***.
- The gate function will define its behavior.
- The input(s) and output can only be discrete states of 1 (true) or 0 (false).

Figure 2.0.1. A logical operation

2.1 Truth Tables

A truth table is often useful in documenting the operation of a digital logic gate or system.

The behavior of a logic gate can be described by a truth table. In other words for any given input(s), the corresponding output can be determined by consulting a simple table. The truth table shows all possible combinations of input states on the left and the corresponding output condition on the right.

The truth table shown in figure 2.1.1 describes the behavior of a two input logical **AND** operation.

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

*Truth table describing a two input **AND** operation. X and Y are inputs, Z is the output.*

Figure 2.1.1. A truth table can be used to describe the behavior of a logic gate.

Example: Using the truth table in figure 2.1.1

If input ***X***= 0 and input ***Y*** =1, then output ***Z*** = 0, likewise if the inputs ***X*** and ***Y*** both equal 1 then the output ***Z*** equals 1.

Logical functions can also be described with Boolean expressions

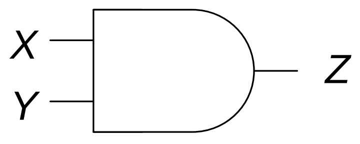
The operation of ***X* AND *Y*** to yield output ***Z*** can be written

$$X \bullet Y = Z$$

It is acceptable and typical to drop the dot when expressing the **AND** operation, thus the following expression also describes operation ***X* AND *Y*** to yield output ***Z***.

$$XY = Z$$

In order to illustrate a logical operation on a schematic diagram each logic gate has a specific symbol. Figure 2.1.2 shows the schematic symbol used to represent a logical **AND** gate.



In order to illustrate a logical operation on a schematic diagram each logic gate has a specific symbol.

Figure 2.1.2. The schematic symbol for the **AND** function.

2.2 Logic Gates

The following chart provides a concise summary of the logical **AND**.

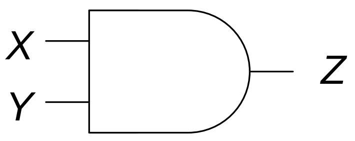
Operation	Expression	Truth Table	Schematic Symbol
AND	$X \bullet Y = Z$	X Y | Z 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	

Figure 2.2.1. The logical **AND**

Another common logical operation is the logical **NOT.** Also known as the compliment or inverse, the logical **NOT** will accept a 1 or 0 input and produce the opposite output. In other words an input of 1 (true) will produce and output of 0 (false). The concise summary of the logical **NOT** operation is shown in figure 2.2.2.

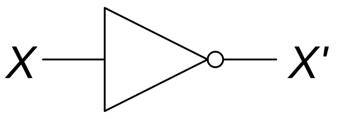
Operation	Expression	Truth Table	Schematic Symbol
NOT	X' or $\overline{X}$	X | X' 0 | 1 1 | 0	

Figure 2.2.2. The logical **NOT**

The next operation is the logical **NAND,** the **NAND** operation can be easily recognized as the compliment (or inverse) of the logical **AND** operation**.** The schematic symbol for a **NAND** gate is similar to the **AND** with the addition of a small circle at the output to denote the compliment as shown in figure 2.2.3.

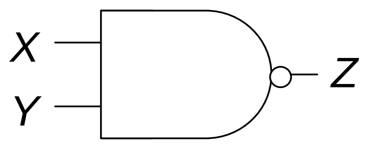
Operation	Expression	Truth Table	Schematic Symbol
NAND	$(X \bullet Y)' = Z$ *or* $\overline{X \bullet Y} = Z$	X Y | Z 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	

Figure 2.2.3. The logical **NAND.**

The logical **OR** operation is summarized in figure 2.2.4.

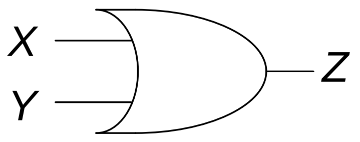
Operation	Expression	Truth Table	Schematic Symbol
OR	$X + Y = Z$	X Y | Z 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	

Figure 2.2.4. The logical **OR.**

The compliment of the logical **OR** operation is the logical **NOR** described in figure 2.2.5.

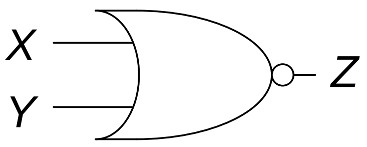
Operation	Expression	Truth Table	Schematic Symbol
NOR	$(X + Y)' = Z$ or $\overline{X + Y} = Z$	X Y | Z 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	

Figure 2.2.5. The logical **NOR.**

Two other common logical operations are the exclusive OR commonly called **XOR** and its compliment **XNOR**, or exclusive NOR. **XOR** and **XNOR** as shown in figures 2.2.6 and 2.2.7.

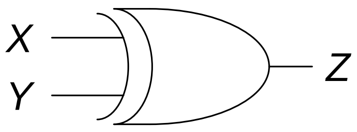
Operation	Expression	Truth Table	Schematic Symbol
XOR	$X \oplus Y = Z$	*X* *Y* | *Z* 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	

Figure 2.2.6. The logical **XOR.**

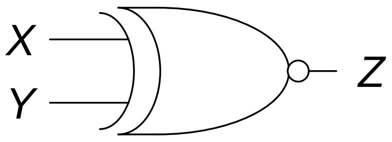
Operation	Expression	Truth Table	Schematic Symbol
XNOR	$(X \oplus Y)' = Z$	*X* *Y* | *Z* 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1	

Figure 2.2.7. The logical **XNOR.**

The above examples involved no more than two inputs for each gate, however there are three, four (and higher) input logic gates. A quad input NAND and a quad input NOR are shown in figure 2.2.8 and figure 2.2.9.

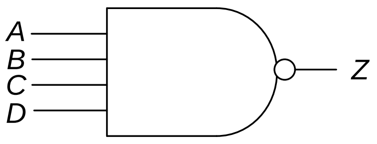
Operation	Expression	Truth Table					Schematic Symbol
		A	*B*	*C*	*D*	*Z*	
NAND	(ABCD)’ = Z	0	0	0	0	1	
		0	0	0	1	1	
		0	0	1	0	1	
		0	0	1	1	1	
		0	1	0	0	1	
		0	1	0	1	1	
		0	1	1	0	1	
		0	1	1	1	1	
		1	0	0	0	1	
		1	0	0	1	1	
		1	0	1	0	1	
		1	0	1	1	1	
		1	1	0	0	1	
		1	1	0	1	1	
		1	1	1	0	1	
		1	1	1	1	0	

Figure 2.2.8. The Quad input **NAND**.

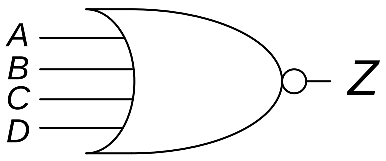
Operation	Expression	Truth Table					Schematic Symbol
		A	*B*	*C*	*D*	*Z*	
NOR	(A+B+C+D)’ = Z	0	0	0	0	1	
		0	0	0	1	0	
		0	0	1	0	0	
		0	0	1	1	0	
		0	1	0	0	0	
		0	1	0	1	0	
		0	1	1	0	0	
		0	1	1	1	0	
		1	0	0	0	0	
		1	0	0	1	0	
		1	0	1	0	0	
		1	0	1	1	0	
		1	1	0	0	0	
		1	1	0	1	0	
		1	1	1	0	0	
		1	1	1	1	0	

Figure 2.2.9. The Quad input **NOR**

3.0 Combinational Logic

Combinational logic is formed when we combine logical elements together and do not employ any type of feedback, that is, the output of a combinational logic circuit will be a function of its inputs. Conversely the output of a sequential logic system may depend on previous output states, in other words some type of feedback has been employed.

Consider the combinational logic shown in figure 3.0.1.
For the given input what is the output **Z**?

A = 0
B = 1
C = 0
D = 0

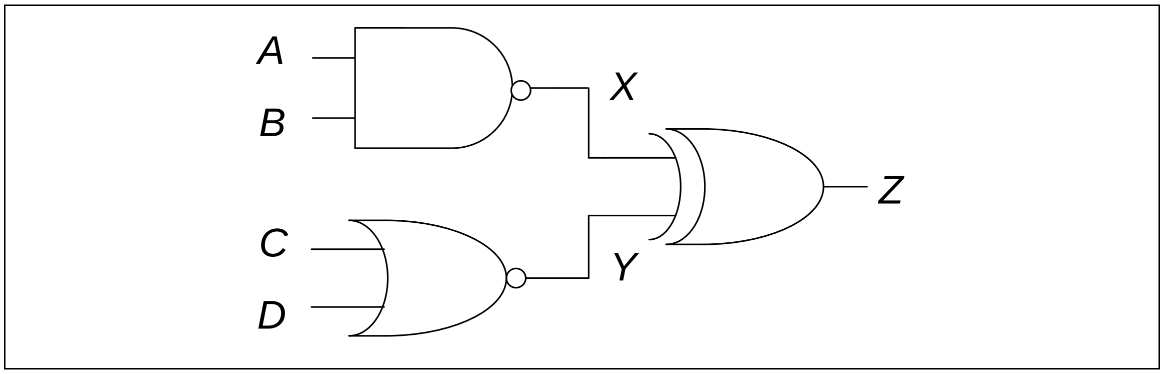


Figure 3.0.1. Combinational logic

This logic design can also be written as a Boolean expression

$(A \bullet B)' \oplus (C + D)' = Z$

To determine the output of this design, consult the individual logic gate truth tables and work your way from left to right through each stage of the design. Determining the output of the previous stage will define the input to the next.

The **NAND** operation:

$(\boldsymbol{A \bullet B})' = \boldsymbol{X}$; where $\boldsymbol{A} = 0$ and $\boldsymbol{B} = 1$, therefore $\boldsymbol{X} = 1$

The **NOR** operation:

$(\boldsymbol{C} + \boldsymbol{D})' = \boldsymbol{Y}$; where $\boldsymbol{C} = 0$ and $\boldsymbol{D} = 0$, therefore $\boldsymbol{Y} = 1$

Using the output from the first stage as the input to the **XNOR** in the second stage.

$$\mathbf{X} \oplus \mathbf{Y} = \mathbf{Z}$$

Where $\boldsymbol{X} = 1$ and $\boldsymbol{Y} = 1$, therefore $\boldsymbol{Z} = 0$.

The following truth table (figure 3.0.2) describes the digital logic in figure 3.0.1 for all possible input values.

A	*B*	*C*	*D*	*X*	*Y*	*Z*	
0	0	0	0	1	1	0	Truth table describing the behavior of the logic shown in figure 3.0.1
0	0	0	1	1	0	1	
0	0	1	0	1	0	1	
0	0	1	1	1	0	1	
0	1	0	0	1	1	0	
0	1	0	1	1	0	1	
0	1	1	0	1	0	1	
0	1	1	1	1	0	1	
1	0	0	0	1	1	0	
1	0	0	1	1	0	1	
1	0	1	0	1	0	1	
1	0	1	1	1	0	1	
1	1	0	0	0	1	1	
1	1	0	1	0	0	0	
1	1	1	0	0	0	0	
1	1	1	1	0	0	0	

Figure 3.0.2. The Truth table describing the digital logic in figure 3.0.1

A Note about precedence of operation:

When evaluating Boolean expressions Be sure to observe the rules of precedence of Boolean algebra; combine terms in parentheses first, followed by the **NOT**, **AND**, then **OR** operations.

Next suppose you want to create a digital design to achieve the logical behavior shown in truth table 3.0.3.

A	*B*	*C*	*Z*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 3.0.3.

The Boolean expression describing this logic can be derived directly from the truth table by combining all terms where the output ***Z*** is true as shown in figure 3.0.4.

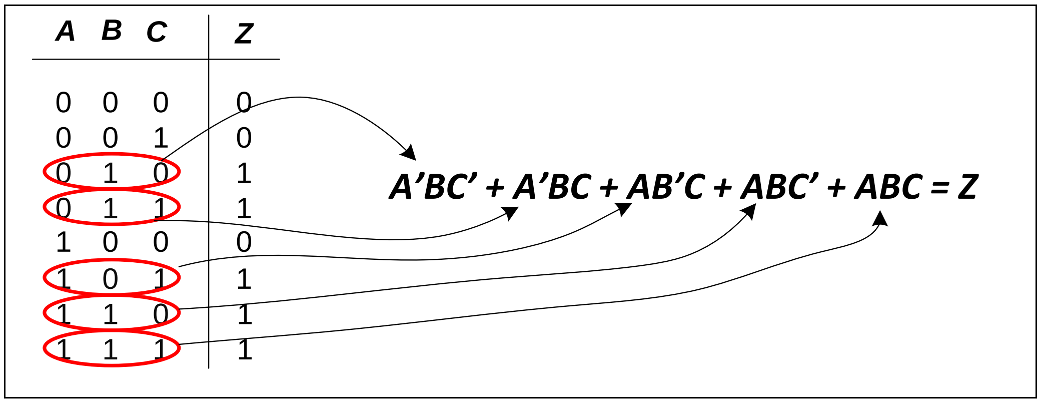


Figure 3.0.4.

This effort generates the following Boolean expression:

$$A'BC' + A'BC + AB'C + ABC' + ABC = Z$$

We can build the logic design directly from this expression with five, three input **AND** gates and one **OR** gate with 5 inputs, as shown in figure 3.0.5.

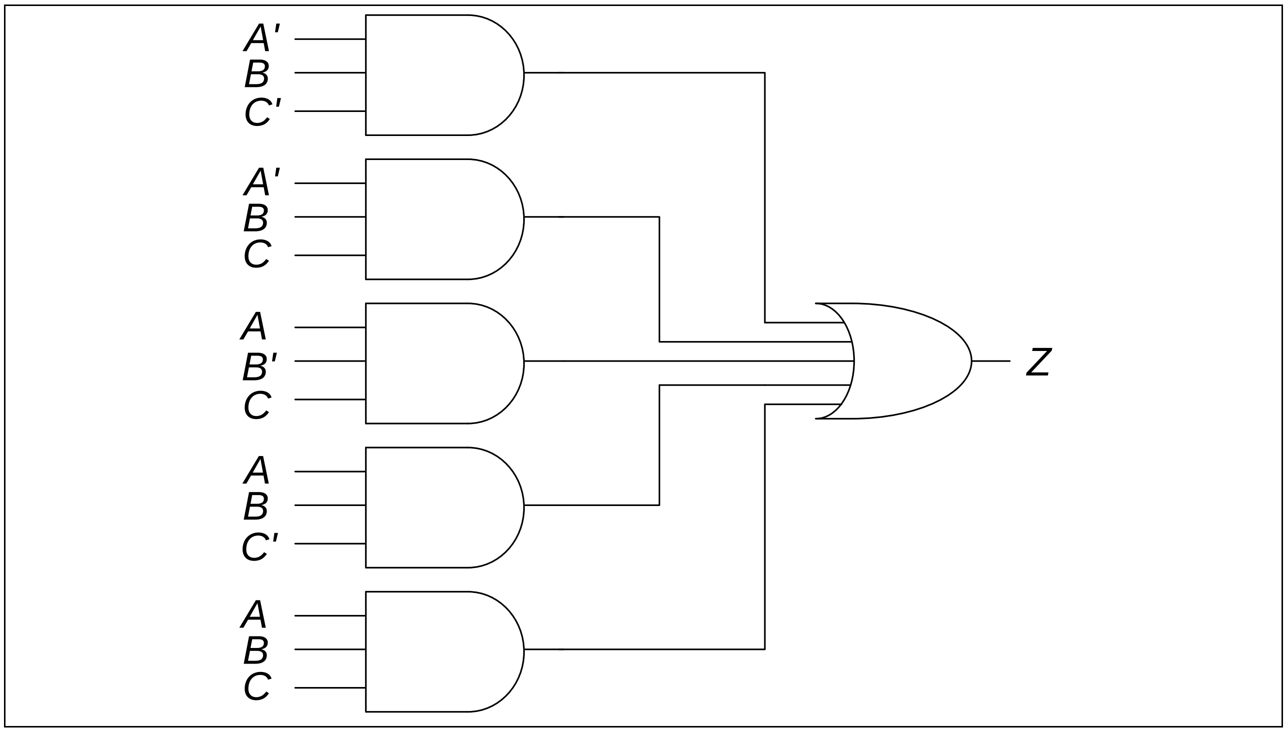


Figure 3.0.5.

It should be noted that while this combination of logic yields a valid solution the same solution can be achieved with far fewer logic gates. Simplification, a task important in keeping power dissipation and overall propagation delay to a minimum can be achieved through Boolean algebraic methods or with Karnaugh maps (both covered in another course). In the above case a careful review of the truth table is all that is required to notice that ***Z*** is only true when ***B*** is true or when ***A*** and ***C*** are true and ***B*** is false or in Boolean algebra terms:

$$B + AB'C = Z$$

Directly from the above Boolean expression the reduced equivalent design is shown in figure 3.0.6.

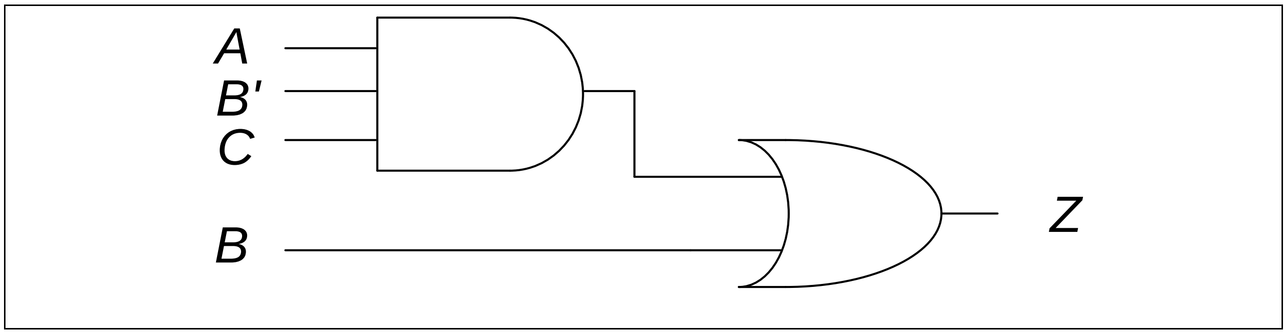


Figure 3.0.6.

3.1 Timing Diagrams

Timing diagrams are useful in detailing the behavior of a circuit. A timing diagram will show the input transitions into the circuit and the resulting output. When no units of measurements or axis labels are provided it is generally accepted that the horizontal axis are units of time and any vertical transitions from a lower position to a higher position indicate a transition from false to True. The timing diagram is useful because it clearly illustrates how the inputs and outputs change in relation to each other. Figure 3.1.1 demonstrates the use of a timing diagram to show how an **AND** gate is used to selectively enable a clock input. The switching diagram shows the clock input is passed through to the output only when the select line is enabled.

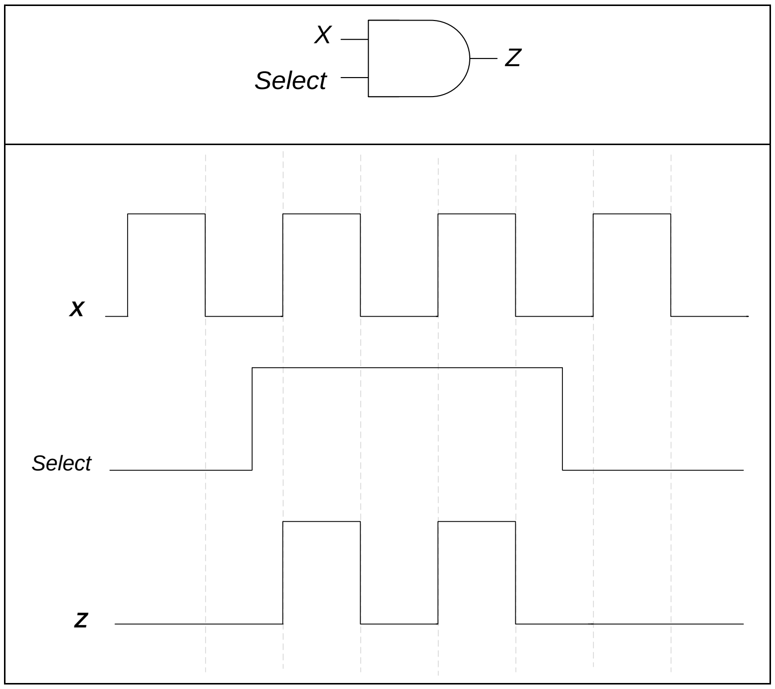
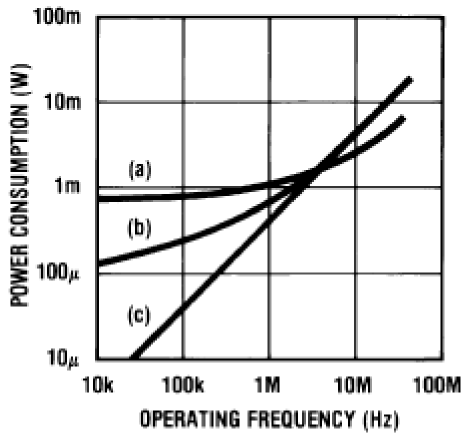


Figure 3.1.1. A timing diagram

4.0 Digital Logic Families

The two most recognized families of digital logic gates are TTL (transistor-transistor logic) and CMOS (Complementary Metal-Oxide-Silicon). While identical logic can be realized with either family, TTL and CMOS have different operating characteristics because of the underlying technology upon which they are built; TTL integrated circuits are built with bipolar transistors while CMOS integrated circuits use MOSFETs (metal oxide field effect transistors). There are many other logic families like the predecessors to TTL, RTL (resistor-transistor logic) and DTL (diode-transistor logic) to more recent forms like BiCMOS which actually combines aspects of both TTL and CMOS. This course will focus on the TTL and CMOS families.

The main advantage that CMOS offers to digital logic design is that of very low power dissipation. CMOS uses complementary P-MOS and N-MOS FETs which draw extremely low power levels. In design scenarios where low power consumption is important, as is the case with portable electronic devices, CMOS offers a large advantage over TTL. However there is a design tradeoff when high switching speeds are required. TTL generally requires more power than CMOS but TTL can support much higher switching frequencies. TTL will maintain a fairly uniform supply current draw across a wide range of switching frequencies, while CMOS will have power characteristics measured in micro amps at quiescence and rise to TTL levels (milliamps) and beyond as switching frequencies increase as shown in figure 4.0.1.



Power Consumption of 74HCT00 Being Driven by a) Worst-Case TTL Levels; b) Typical TTL Levels; c) CMOS Levels

CMOS power dissipation increases as switching frequencies increase

Figure 4.0.1.
Excerpt from Fairchild Semiconductor Application Note 368 March 1984 **"An Introduction to and Comparison of 74HCT TTL Compatible CMOS Logic" used with permission from Fairchild Semiconductor.**

4.1 Logic identifiers

Digital logic semiconductors are often referenced by an identifying number like 74LS02. This identifier will reveal the family, subfamily and function of the chip and possibly more information depending on the manufacturer. The TTL family of logic gate is typically denoted with the identifying number beginning with "74" while military grade series are denoted with "54". Within each family of technology there are various sub-families with different operating characteristics.

4.2 Digital Logic Sub-families

The sub-family can be identified by the next few letters after the "74", for example, some common TTL subfamilies are L, LS, S, AS, ALS, and F. The subfamily is followed by the chip number which identifies the logic function. Figure 4.2.1 defines some common TTL sub-families.

TTL sub family designation	TTL sub family description
L	Low Power
LS	Lower power Schottky
S	Schottky
AS	Advanced Schottky
ALS	Advanced Low Power Schottky
F	Fast

Figure 4.2.1. TTL subfamilies

Many CMOS chips also follow the "74" series of logic gate numbering; however CMOS has different subfamily types, a few are shown in figure 4.2.2. There is also a slightly older family of CMOS with the 4000 series chip designation, this series typically features wide (3v-18v) supply voltage ranges.

"74" series CMOS sub- family designations	CMOS sub-family description
AC	Advanced CMOS
ACH	High Speed Advanced CMOS
LVC	Low Voltage CMOS
ALVC	Advanced Low Voltage CMOS
HC	High Speed CMOS
HCT	High Speed CMOS w/ TTL compatible switching thresholds

Figure 4.2.2. CMOS sub-families

"74" Series Identifier	Logic Gate Description
74xx00	Quad 2-Input NAND
74xx02	Quad 2-Input NOR
74xx04	Hex Inverter
74xx08	Quad 2-Input AND
74xx10	Triple 3-Input NAND
74xx20	Dual 4-Input NAND
74xx30	8-Input NAND
74xx32	Quad 2-Input OR
74xx86	Quad XOR

Figure 4.2.3. Common logic gate identifiers

Given a logic gate identifier, the above tables can be used to determine the logical family, subfamily and function of the most common gates. The following table provides three examples.

Type	Description
74ALS02	TTL Advanced Low Power Schottky quad 2 input NOR gate
74AC02	Advanced CMOS quad 2-Input NOR gate
74HCT32	CMOS High Speed w/ TTL compatible switching thresholds quad 2-Input OR gate

Figure 4.2.4.

Figure 4.2.5 is instrumental in highlighting the differences between TTL and CMOS subfamily types. The diagram was created by graphing the properties of a 74xx00 **NAND** gate across different TTL and CMOS sub-family types. This exercise clearly shows the significantly lower (by an order of magnitude) power dissipation of the CMOS subfamily.

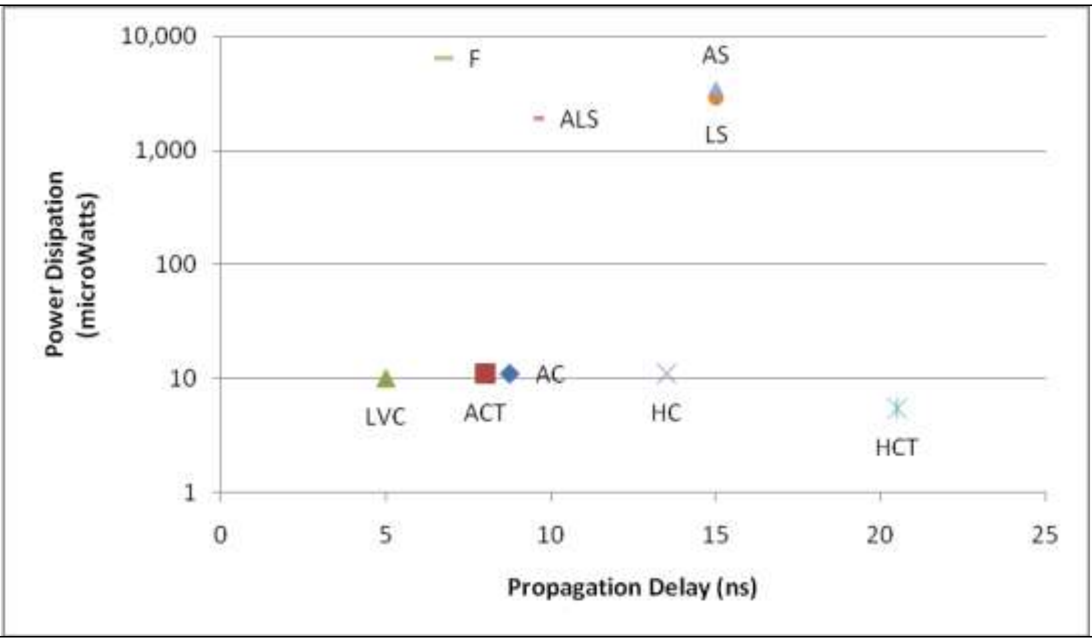


Figure 4.2.5. Comparison of 74xx00 NAND gate across TTL and CMOS sub-families

Notes:
The supply current for CMOS subfamilies is listed at quiescence.
Listed supply current for TTL subfamilies is the average of typical and guaranteed max under typical operating conditions.
Propagation delay average values are average of Max $\mathbf{t_{PHL}}$ and Max $\mathbf{t_{PLH}}$ for both subfamilies.

4.3 Positive and Negative logic.

A logic gate will use different voltage levels to indicate the logic states of true and false.

In the case of positive logic a 0 (false) is defined as a lower voltage like 0^v whereas 1(true) is defined as a higher voltage like 5.0^v.

Positive logic is very common and will be the used throughout this course. Negative logic simply reverses the definition such that a high voltage indicates a false condition while a lower voltage level indicates a true condition.

4.4 Input and output voltage ranges

The device data specification sheet should always be consulted for the proper operating parameters however most TTL devices have accepted input and output ranges for logic levels as show in Figure 4.4.1.

Input 0 (false)	**Input 1** (true)	**Output 0** (false)	**Output 1** (true)
V_{IL} Min – Max	V_{IH} Min – Max	V_{OL} Min – Max	V_{OH} Min – Max
$0.0^v – 0.8^v$	$2.0^v – 5.0^v$	$0.0^v – 0.4^v$	$3.0^v – 5.0^v$

Figure 4.4.1. Positive logic, common TTL voltage levels

4.5 Propagation Delay

Thus far in our examination of digital logic we have assumed that the output of a logic gate was generated at the same instant that the input signal was applied. However in practice there is a finite period of time between the presence of a valid and stable input and the resulting output. This period of time is called propagation delay (t_{PD}) as shown in figure 4.5.1.

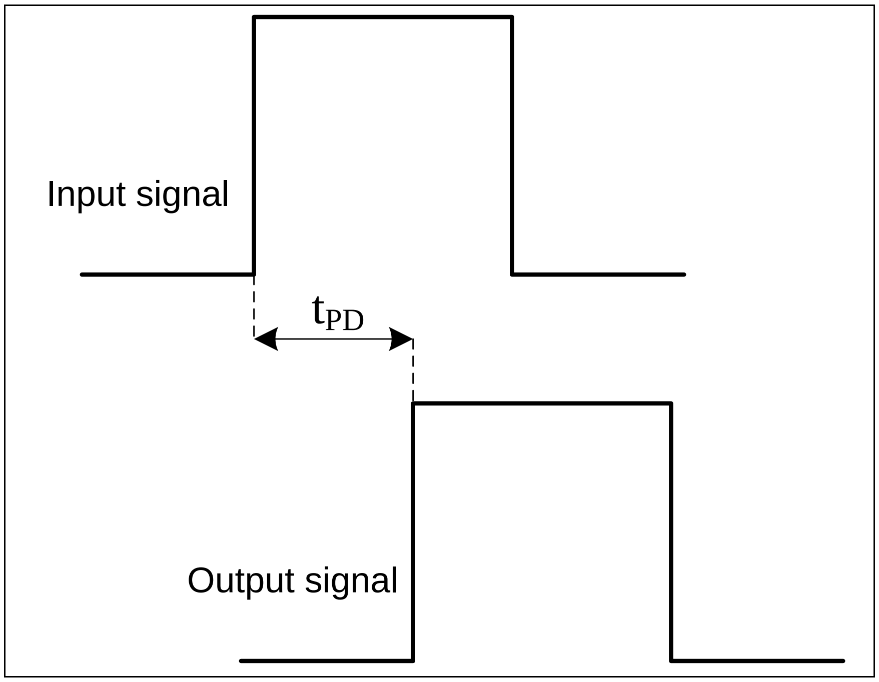


Figure 4.5.1. Propagation delay

Propagation delay may differ significantly between logic families and sub-families as detailed in figure 4.2.5. Propagation delay is not a static parameter; it will vary depending on the operating conditions of your circuit. For example your circuit may behave as expected in the controlled 23C° temperature of your test lab, but deliver unintended behavior when operating in the field where temperatures vary from 0C° to 50C°. The chip's data specification sheet will typically provide propagation delay characteristics across a range of operating temperatures however other conditions will also impact propagation delay. Output load capacitance will typically have the largest impact, while other factors like changes in power supply voltage, or the quantity of outputs switching concurrently can also impact propagation delay.

Propagation delay can be measured when an output transitions from low-to-high (t_{PLH}) or from high-to-low (t_{PHL}) see figure 4.5.2. It should be noted that t_{PLH} and t_{PHL} often differ.

Where timing is not important in your design propagation delay can be safely ignored, however where there are tight timing tolerances propagation delay must be properly accounted otherwise your design may produce intermittent or unintended results.

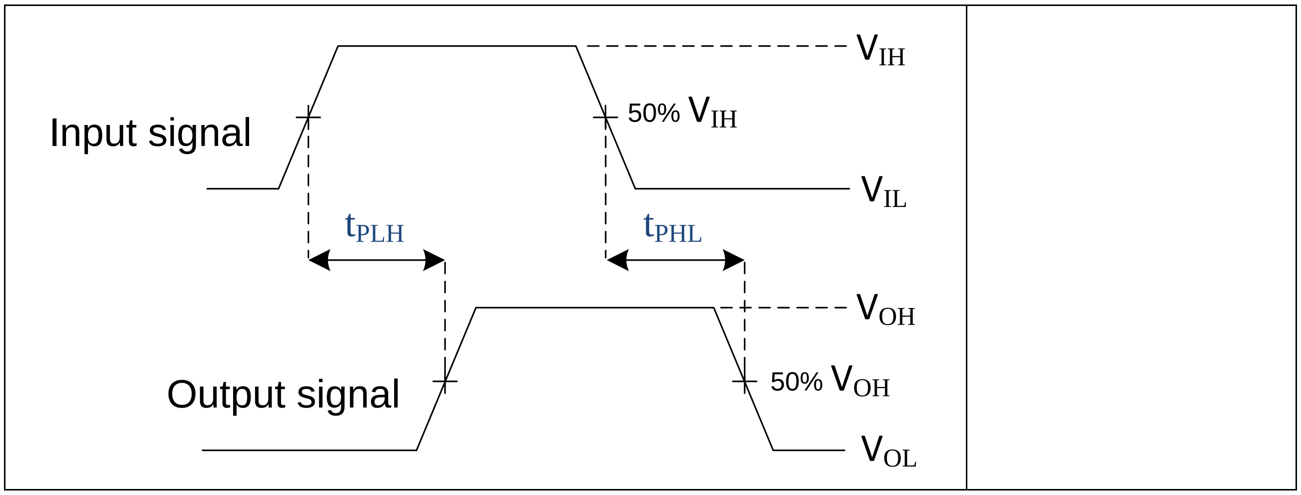
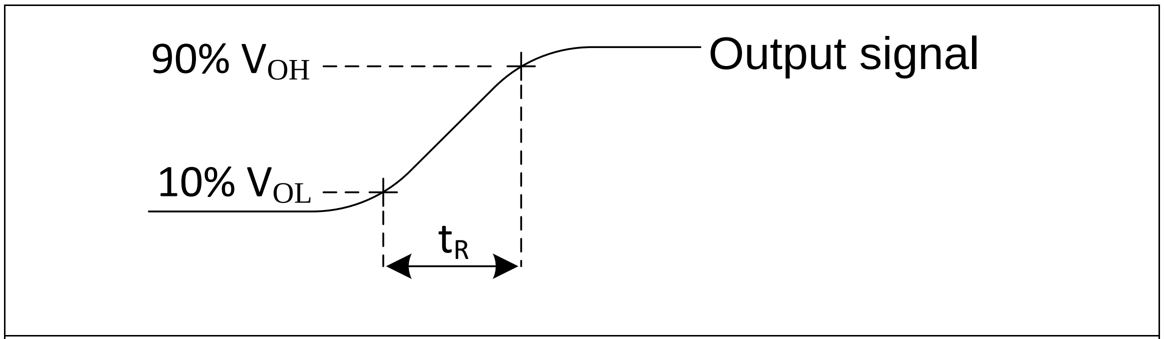


Figure 4.5.2.

4.6 Slew Rate

Slew rate is defined as the rate of change of an output signal from 10% to 90% of its final value as shown in figure 4.6.1. Slew rate is a measure of how fast an output can transition from low-to-high or high-to-low. Slew rate will change depending on the capacitance seen at the output; intuitively a slower low-to-high transition will occur if a capacitor must be charged. Slew rate measured from a low-to-high transition may differ from that of a high-to-low transition.



$$\text{Slew Rate} = \frac{90\%\ V_{OH} - 10\%\ V_{OL}}{t_R}$$

Figure 4.6.1.

4.7 Fan-out & Fan-in

In our examination thus far we have not placed any limit on the output drive capability of a logic gate. The maximum number of logic gate inputs that can be driven from a single logic gate's output is called fan-out. For example if a gate output can supply (or sink) $\mathbf{I_{out}}$ and another gate's input requires $\mathbf{I_{in}}$, fan-out is $\mathbf{I_{out}} / \mathbf{I_{in}}$. Fan-out is especially important when designing with TTL because TTL gates have larger input current characteristics. Conversely CMOS logic gates feature very low input current requirements which allow a single CMOS gate to drive many downstream CMOS gates. In fact the typical input current requirement for CMOS is so low that output current fan-out has been replaced by load capacitance as the limiting factor. As additional CMOS gates are connected to a single CMOS output the capacitance is cumulative; this capacitance will serve to limit the actual fan-out. However even with this limitation CMOS fan-out is typically far superior to TTL. Figure 4.7.1 emphasizes the fan-out differences between TTL and CMOS by comparing the fan-out capabilities of the HC family of CMOS chips to that of common TTL families.

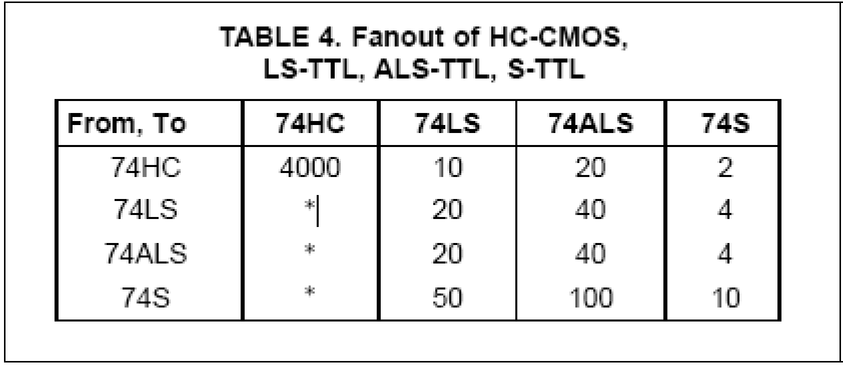


TABLE 4. Fanout of HC-CMOS, LS-TTL, ALS-TTL, S-TTL

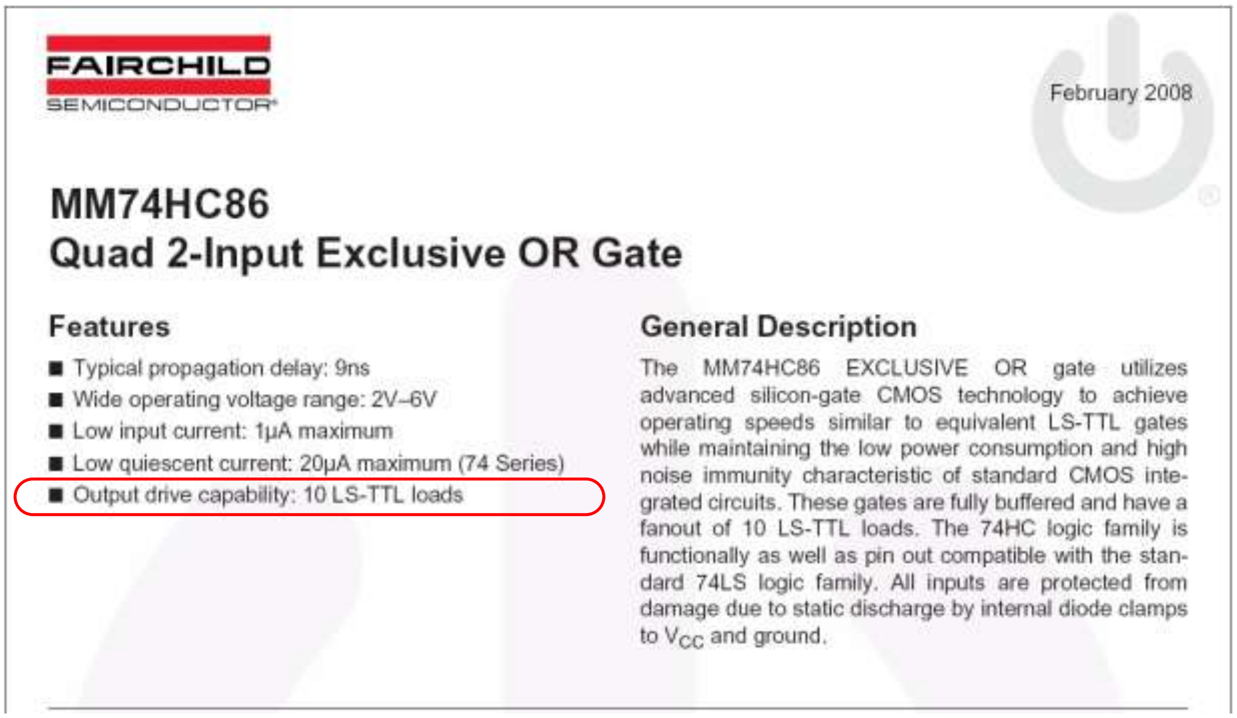
From, To	74HC	74LS	74ALS	74S
74HC	4000	10	20	2
74LS	*	20	40	4
74ALS	*	20	40	4
74S	*	50	100	10

The 74HC series of Fairchild CMOS gates has a fan-out rating of 4000 downstream 74HC gates essentially eliminating fan-out as a design limitation. It should be noted that the capacitance of the connected CMOS gates will serve to limit the actual number that can be connected well before the fan-out limit is reached.

Figure 4.7.1. Example of fan-out differences between CMOS and TTL

Excerpt from Fairchild Semiconductor Application Note 319 June 1983 "Comparison of MM74HC to 74LS, 74S and 74ALS Logic" used with permission from Fairchild Semiconductor.

Sometimes the specification sheets are more explicit, figure 4.7.2 shows an excerpt from the specification sheet for a Fairchild MM74HC86, a quad 2-Input Exclusive OR Gate indicating this CMOS chip can drive up to 10 TTL devices.



FAIRCHILD SEMICONDUCTOR

February 2008

MM74HC86
Quad 2-Input Exclusive OR Gate

Features

- Typical propagation delay: 9ns
- Wide operating voltage range: 2V–6V
- Low input current: 1µA maximum
- Low quiescent current: 20µA maximum (74 Series)
- Output drive capability: 10 LS-TTL loads

General Description

The MM74HC86 EXCLUSIVE OR gate utilizes advanced silicon-gate CMOS technology to achieve operating speeds similar to equivalent LS-TTL gates while maintaining the low power consumption and high noise immunity characteristic of standard CMOS integrated circuits. These gates are fully buffered and have a fanout of 10 LS-TTL loads. The 74HC logic family is functionally as well as pin out compatible with the standard 74LS logic family. All inputs are protected from damage due to static discharge by internal diode clamps to V_{CC} and ground.

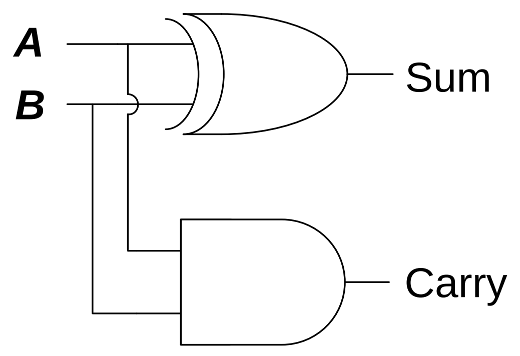
Figure 4.7.2. An example of CMOS to TTL Fan-out documented on the product spec sheet.
Excerpt taken from http://www.fairchildsemi.com/ds/MM%2FMM74HC86.pdf with permission from Fairchild Semiconductor.

A similar concept of fan-In indicates the number of logic gates that can be connected to a logic gate input.

5.0 Common Circuits

5.1 The half-adder

The need to add two binary numbers together is a requirement frequently encountered in digital logic. The most basic solution to this problem can be accomplished with the half-adder. Figure 5.1.1 shows a circuit to add two single bit binary numbers together. The exclusive OR is used to sum the bits while the AND gate is used to handle the carry bit. Because this circuit has no logic to handle a carry from an earlier stage it is referred to as a half-adder.

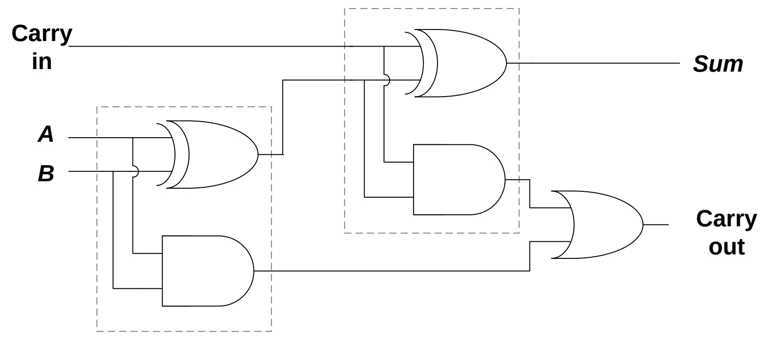


A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 5.1.1. A circuit to perform simple arithmetic of two binary numbers is the half-adder

5.2 The Full-Adder

A full-adder can be created by combining two half-adders as shown in figure 5.2.1. The full-adder serves the same purpose as the half-adder, that is to sum two single bit binary numbers, however, the full-adder can accept a carry input from a previous stage.



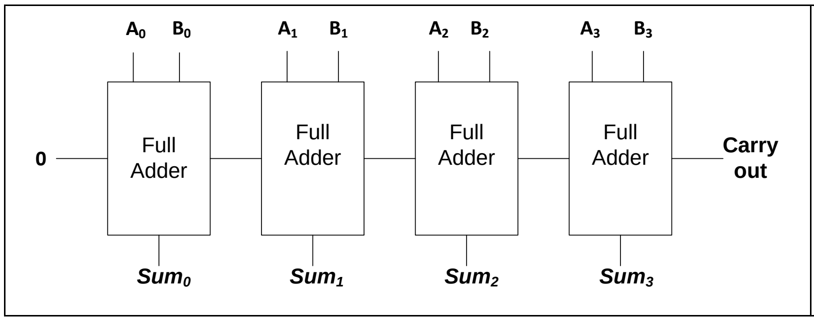
Carry in	A	B	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 5.2.1. The full-adder circuit. Two half-adders, shown within the dashed lines, are combined to form a full-adder capable of handling a carry-in bit.

5.3 Ripple Carry Adder

While adding single bit binary numbers is interesting, we often need to add larger binary numbers; 4, 8, 16 bit adders can be accomplished by interconnecting multiple stages of the full-adder. With each additional stage we can accomplish one more bit of binary addition.

If we set the carry-in of the first stage (bit zero) to false and connect the carry-out of each subsequent stage to the carry-in of the next stage we can build a larger adder as shown in figure 5.3.1. This form of counter is commonly referred to as the ripple carry adder.



By chaining full-adder circuits we create a circuit to add two four-bit binary numbers A_{0-4} & $B_{0-4.}$

Figure 5.3.1. The ripple carry adder

Propagation delay can become a limiting factor with ripple carry adders because each successive stage must "wait" for the carry bit from the previous stage. For large binary number addition where fast response is required other summing techniques must be employed.

5.4 Parity Generator

Parity generation & checking is a common method of error checking. This next circuit (figure 5.4.1) will demonstrate a simple parity generator made with XOR gates.

The timing chart shown in Figure 5.4.1 provides the inputs and the resulting output when applied to the cascaded XOR combination. While reviewing the input and output state transitions it should be noticed that this circuit functions as a simple even-parity generator. An even number of high inputs will result in a low output, an odd number of high inputs will result in a high output. Parity is commonly used as a simple error checking method in data transmission. The concept is that the parity on the transmitting side should match the parity on the receiving side; if it does not, an error must have occurred and that data should be discarded. In our case **Z** is the parity bit, the parity bit will be counted along with the inputs **A**, **B**, **C** & **D** to ensure the number of high states is even. A similar circuit on the receiving end of a transmission could be used to verify the parity was always even.

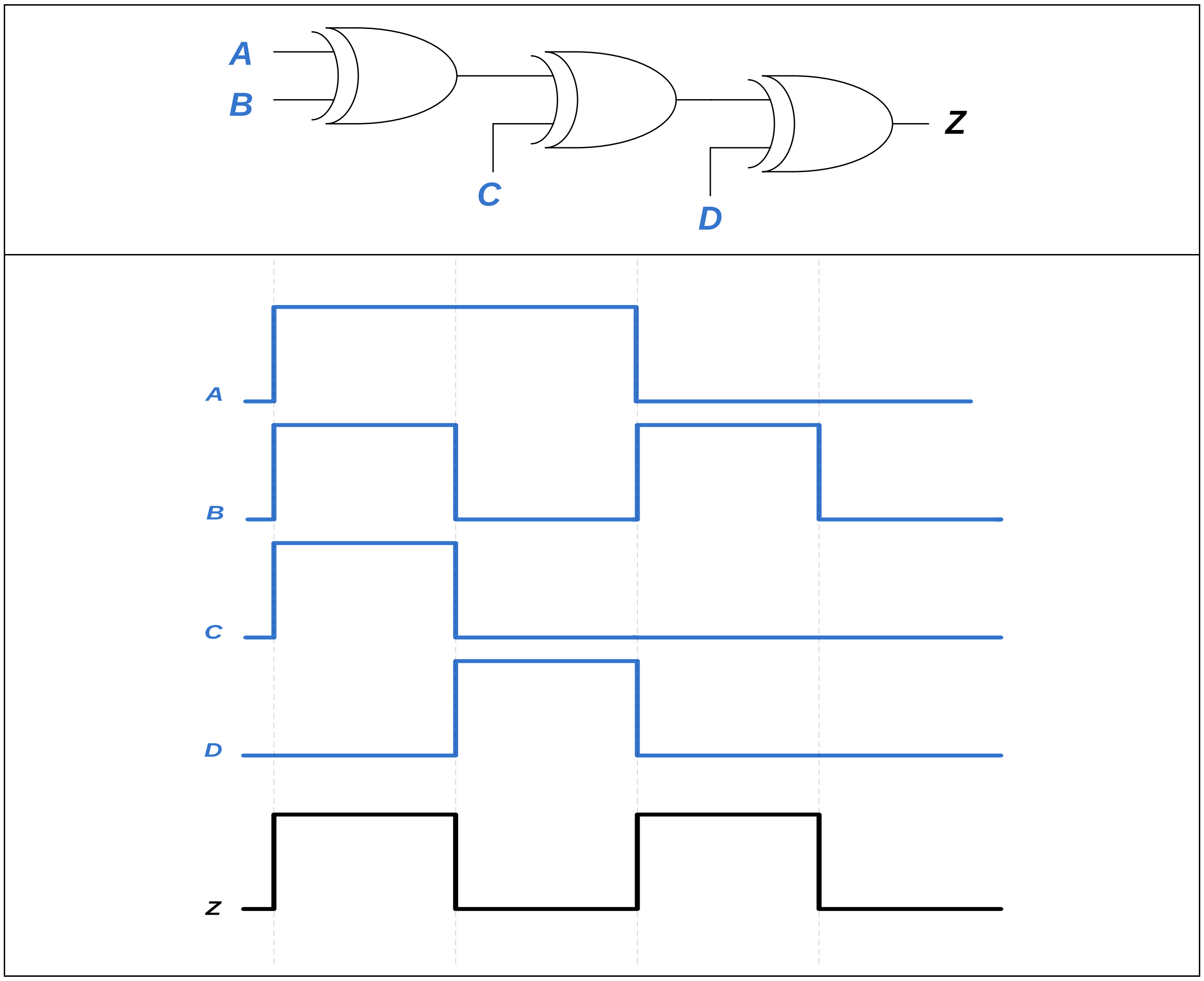


Figure 5.4.1. The timing diagram detailing the behavior of the parity generator.

The input values of ***A***, ***B***, ***C*** & ***D*** are given in the timing chart along with the corresponding output value ***Z***. Like the ripple carry-adder in the previous example because of each successive stage of the parity generator depends on the previous, propagation delay will become a limiting factor especially as speeds increase. With each additional bit that is added to the parity generator the propagation delay increases by t_{PD} (the propagation delay of the XOR).