

Chapter 1

Introduction to Digital Systems Design

This text deals with the design of digital systems. The purpose of this chapter is to introduce the notion of digital systems in preparation for the remainder of the text. The main concepts you should get from this chapter include an understanding of a few of the differences between digital and analog systems.

1.1 Digital vs. Analog

It is often said that we live in a digital world, but what does that really mean? What is the opposite of or alternative to digital? In its simplest form, a digital system is one that manipulates and stores binary values. A binary value can have only one of two different values: TRUE or FALSE. The values TRUE and FALSE are often represented by 1 and 0 respectively. We use the term *bit* (short for *binary digit*) to refer to a single value consisting of either a 1 or a 0. Using a single binary value you can only represent notions such as ON vs. OFF, HIGH vs. LOW, HUNGRY vs. NOT-HUNGRY, etc.

The physical representation of a binary value in a digital system might be done using a voltage on a wire. In this case a high voltage would correspond to a 1 (TRUE) and a low voltage would correspond to a 0 (FALSE). The digital system would then contain circuits to operate on those voltages (values) to do computations.

In contrast, an *analog* value is one that can take on any value from a continuous range. An example of an analog quantity might be the representation of the current temperature which could take on any value from the continuous range of -120 to +120 degrees and would be mapped to the range of 0V to 2.4V. In this case, each 0.01V change on the wire voltage might correspond to a 1 degree change in temperature. Clearly, this quantity could represent the temperature much more accurately than a single bit would be able to. The temperature could (conceivably) take on any real number while in the digital case, the temperature would be represented simply as one of two different quantities such as HOT vs. NOT-HOT (COLD) where the dividing line between HOT and COLD might be arbitrarily set at 80 degrees.

The reader may be inclined to conclude that analog values can more accurately represent information than digital values. This is true when the digital value is a single bit, but to represent a wider range of values a binary word can be created from multiple bits, thereby providing accuracy comparable to that achievable with an analog representation. To better understand this, a review of positional number systems is in order.

1.2 Positional Number Systems

In elementary school we all learned the notion of a positional number system. We all know that the number 735 can be interpreted as follows:

$$735 = 7 \times 10^2 + 3 \times 10^1 + 5 \times 10^0$$

That is, the digits of the number 735 can be interpreted as being applied to powers of the base (10 in this case). Further, in base 6 the following is true:

$$325_6 = 3 \times 6^2 + 2 \times 6^1 + 5 \times 6^0 = 108 + 12 + 5 = 125_{10}$$

The positional number system can be applied to any number base. In the case of binary data the number base used is base 2. This has the interesting property that the only available digits are 0 and 1. This corresponds exactly to our definition of binary values above. The interpretation of a base 2 number follows exactly from the examples given above:

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 8 + 4 + 1 = 13_{10}$$

The largest quantity a 4-bit binary number can represent is 15 (1111) and the smallest is 0 (0000). The range of values that a number can represent can be enlarged by simply adding more bits. Thus, an 8-bit binary number can represent values in the range 0 – 255 and a 12-bit number can represent values in the range 0 – 4,095. The result is that by adding additional bits to a binary word we can create a way of representing information just as accurately as analog values do.

1.3 Digital vs. Analog Continued

At this point you are prepared to now understand a few of the differences between analog and digital. A first difference has to do with how values are represented. In an analog world, the current temperature might be represented by a voltage on a wire as discussed above. Alternatively, the voltage might be represented by some other physical quantity such as the amount of current flowing through a wire, the pressure in a vacuum, or the strength of a magnetic field. In each case, a continuously variable physical quantity is used to represent the analog value.

In a digital world the current temperature would be converted to a binary number to represent it. For example, a binary value of 1000011 would represent the number 67 in base 10, and could be used to represent the fact that the current temperature was 67 degrees. It would require 7 wires in this case to represent the current temperature in the system, each of which would contain either a 1 (a high voltage) or a 0 (a low voltage).

To further motivate the difference between analog and digital storage and processing of information let us consider an example from the area of photography, particularly the process of taking and processing a black and white picture using both analog and digital means.

Many of the earliest photographs were taken using a simple camera as shown in Figure 1.1. A sheet of light-sensitive paper (film) was placed at the back of a box. A pinhole was then opened in the front of the box. The light penetrating the box through the pinhole would strike the film at the back of the box and an image would be formed. The photographic paper, being coated with light-sensitive chemicals (the emulsion), would record an analog image of the scene. Emulsion in the areas on the photographic paper which received light would undergo a chemical change and those areas not receiving light would not. Developing the photo consisted of using chemicals to wash away the parts of the emulsion which were thus chemically changed while leaving behind the other emulsion (or vice versa). The result was what we call a black-and-white photograph.

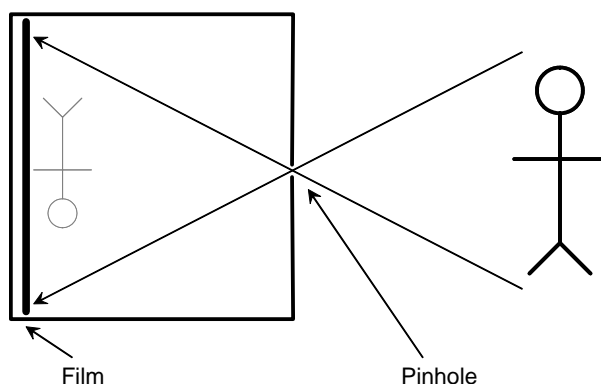


Figure 1.1: Basic Analog Camera

This description of the photographic process is still not quite complete since it has described the emulsion as either being completely chemically altered by light striking it or not (a digital concept!). In actuality, the amount of chemical change experienced by any *area* of the emulsion layer is in proportion to the intensity of light it is exposed to and thus we perceive it as a gray-scale image with shades ranging from black through various shades of gray all the way

through white. This is an example of recording analog data - the concentration of emulsion remaining in any area on the developed film is essentially a continuous function of the amount of light exposure received by that portion of the film¹.

Once an image is captured on a piece of film, what can then be done with it? If the film base were transparent then light could be projected through it and a copy of the image could be created on a second piece of film. In the process various filters could be placed between the original and copied image to effect a change such as blurring or contrast enhancement. The entire process, however, is an analog process — essentially continuous physical quantities (light intensity, chemical concentrations) are used to store and manipulate the original data (the image). An important consideration with analog processes is that they are sensitive to variations such as chemical intensity, lens aberrations, and even dust particles or scratches on the filter surface.

In contrast, consider the process of taking a digital photograph as shown in Figure 1.2. A camera lens is used to focus the light from a scene onto a silicon detector (a sensor circuit). The sensor surface is divided into a 2-dimensional grid of locations called *pixels* or picture-elements. The light intensity at each pixel (an analog quantity) is sensed and converted to a multi-bit binary number. As an example, the pixel intensities might be represented by 8-bit binary numbers with 0 (00000000 in binary) representing black and 255 (11111111 in binary) representing white.

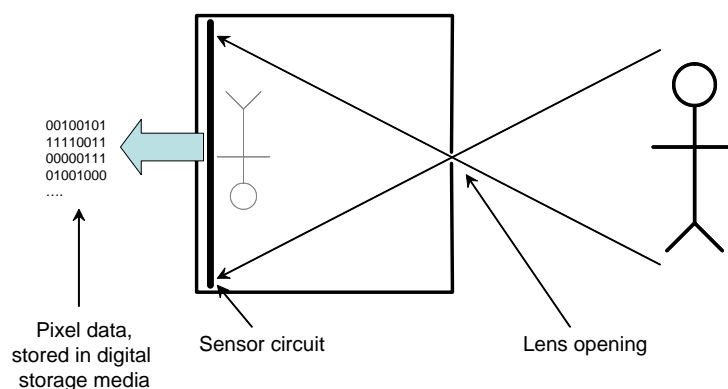


Figure 1.2: Basic Digital Camera

The digital photograph then, is simply a listing of the light intensities at each and every pixel location. These intensity values (binary words) are transmitted from the sensor circuit to a digital storage circuit and can then be stored on a computer. The 'GIF' file format is an example of such a file format and essentially contains a list of the light intensities at each pixel location, each represented as one or more binary words. To view the picture, the pixels on a computer monitor are used to display the pixel intensities stored in the GIF file. When we view such a computer display we 'see' the original image.

This is an example of *digital quantization* of two kinds. First, the image itself is *spatially quantized* or sampled. That means that a finite set of pixel locations are chosen and the image intensities *only* at those locations are recorded. If the number of pixels is large enough (the spatial sampling rate is high enough) we see no visible degradation in the final image. If the number of pixels is small (the spatial sampling rate is low) we then see a 'blocky' image. An example of this is shown in Figure 1.3 which is made up of an 8-by-8 array of pixels and might be used as a pointer icon in a windows-based operating system. The effects of spatial quantization are very evident. A 64-by-64 pixel version of the pointer would look much better (have less blockiness). In a 1,024-by-1,024 pixel version the blockiness would not be visible.



Figure 1.3: An 8×8 Image

¹ At the atomic level, even photographic emulsion has spatial quantization effects (called *grain*) and also sensitivity limitations. Nevertheless, it is common to view the intensities stored on the film as continuous analog values both spatially and in intensity.

The second kind of quantization that occurs in this process relates to quantization of the intensity values. In a real image the intensity at any given pixel location can take on any continuous value from a range of values. By choosing an 8-bit binary word to represent each pixel intensity we are limiting the intensities that we can represent to 256 different values. If the quantization is fine enough (there are enough different intensity values) we don't perceive a difference between the recorded image and the original image. But, if the quantization of the intensities (brightness values) is coarse enough we see artifacts in the image which indicate to us it is a digitally captured image. This can be seen in the image of Figure 1.4 which is an image which has been quantized to both 8-bit intensities (256 different brightness values) and to 3-bit intensities (only 8 different brightness values).

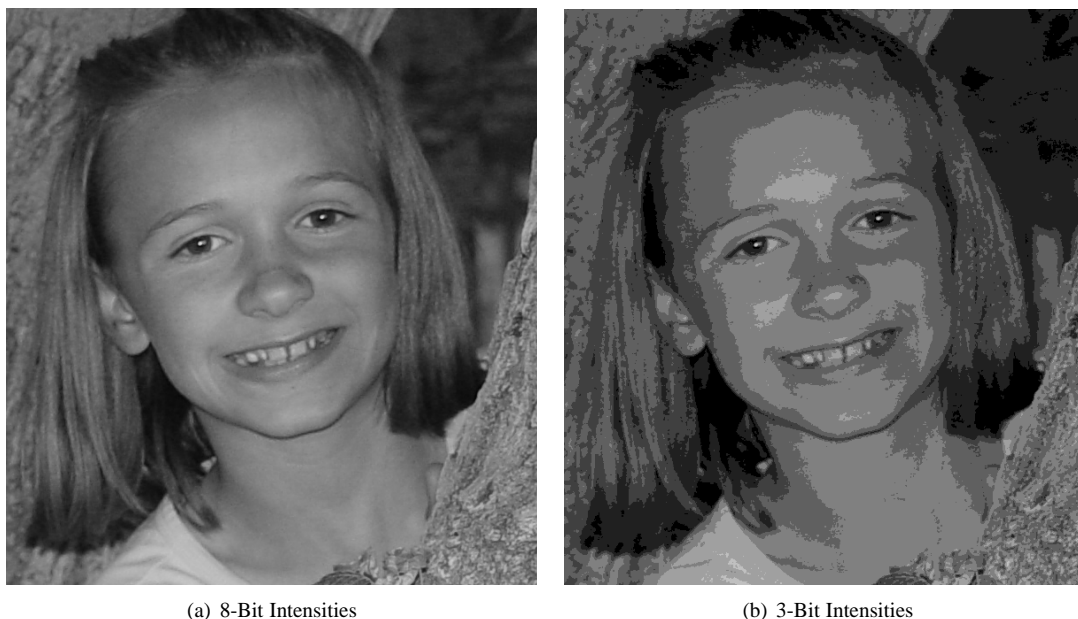


Figure 1.4: Wendy Quantized to Both 8-bits Per Pixel and 3-bits Per Pixel

1.3.1 Analog vs. Digital Data Storage

Next, consider the storage of images. A photographic image often will fade over time due to gradual chemical changes in the emulsion and film. While the rate of fading varies with the film type, long term changes in the emulsion, film, and paper are inevitable. Analog photos are thus not completely permanent.

In contrast, once the pixel intensities have been recorded as binary data in a GIF file it should be possible to preserve the exact image (the pixel intensities) indefinitely. We are careful to say 'should be possible' in that new digital media is constantly being introduced and it can be difficult to find equipment to read old-format digital media (try to find a $5\frac{1}{4}$ inch floppy drive). Independent of that, however, if the 1's and 0's of the original data can be retrieved from the media, that original image can be precisely reproduced (more correctly, the originally sampled and quantized version can be precisely reproduced).

Further, consider the process of making copies. A copy of a digital image (the GIF file) can be bit-for-bit exact and so there is no limit to the number of identical digital copies which can be made. In the case of analog copies, the copying process itself may introduce changes due to imperfections in the lenses used, in the chemicals used to develop the copies, or in any one of a number of other parts of the copying process. The result is that n-th generation analog copies may differ noticeably from the original.

1.3.2 Analog vs. Digital Data Processing

Continuing on with our digital photo example, processing an analog photo (to alter its color in the case of a color photo) can be done by projecting it through a lens or filter of some kind and recording the new image on film. In the case of a digital image, the contents of the GIF file (or a copy) can be altered by a computer program. Users of

the popular Adobe Photoshop program are aware of the many, many changes that can be made to a digital photo in this way including: sharpening, color changing, contrast enhancement, blurring, etc. Further, parts of various images can be combined into one via cutting and pasting. All of this done by manipulating the binary data from the GIF file representation of the image.

1.3.3 Analog vs. Digital - A Summary

In summary, digital representations of data have a number of benefits over analog representations for storage, duplication, transmission, and processing. Does this mean there is no longer a need for analog circuits? Absolutely not. The world we live in is largely an analog world and analog techniques are needed to deal with it. That said, a general trend over the past few decades has been to increasingly design systems which convert these analog quantities into equivalent digital representations, and which then store and manipulate the resulting values using digital rather than analog techniques. As will be shown in the remainder of this text, digital circuits are easily designed and built to operate on digital (binary) data and which can manipulate, store, and retrieve binary data.

1.4 Combinational vs. Sequential Digital Circuits

The above discussion focused on the use of digital techniques for recording, manipulating, and storing data. Another important use of digital systems is for control circuits. For example, consider the creation of a machine to control a car wash. The machine must react to the insertion of coins into its coin box, it must sequence the car wash pumps in order to properly spray the car, apply soap, rinse the car, etc. Further, it must time these various steps since each takes a different amount of time. Finally, it must flash lights to communicate with the user. A digital system is a good way of implementing such a controller.

Digital circuits can be divided into two broad categories — *combinational* circuits and *sequential* circuits. Combinational circuits are the simpler of the two and are circuits whose outputs are a simple and direct function of their inputs at all times. A typical problem which could be solved using a combinational circuit would be a cooling fan control circuit for a computer:

Create a circuit whose output is TRUE any time the current temperature in the computer case is HOT or any time the CPU power is turned ON.

This would be a circuit with two inputs and one output, each of them 1-bit wide. The first input would represent the temperature in the computer case (HOT=1 vs. NOT-HOT=0). The second input would signify whether the power to the CPU was ON or OFF (ON=1, OFF=0). This circuit would ensure that any time the case was hot the fan would be turned on (even if the CPU was not being powered). It would do this by asserting its output to be TRUE (high) in this case. It would also ensure that the fan was on any time the CPU was receiving power by asserting its output. The first part of this textbook is devoted to the design and implementation of combinational circuits such as this — they continuously monitor their inputs and continuously update their outputs in response.

In contrast, a sequential digital system is one which possesses some form of memory. At each time instant it makes a decision on what to generate for its outputs based not only on the current value of its inputs (presence of a coin in a coin slot for example), but also on its current state (what step it is at in the process it is performing). For a car wash controller that is required to perform 5 steps to complete a car wash, we would say that the sequential machine would have 5 states.

By querying the value of its state, such a sequential machine can determine what action to perform in response to changes on its inputs. For example, consider our hypothetical car wash controller. The action it should take when a coin is inserted into its coin box is different depending on what state it is in. If it were in the IDLE state it would interpret that action as a request to start the car wash and thus start the 5-step process to wash the car. If it were in a state indicating it was in the middle of the car wash's rinse cycle, it would interpret the insertion of a coin as a mistake by the customer and generate the necessary outputs to return the coin to the user. It is only by using this notion of state that we can design a sequential system such as this 5-step car wash controller.

The design of sequential circuits has, as a prerequisite, the ability to design and implement combinational circuits. Thus, sequential circuit design is postponed until you have mastered combinational logic design. By the end of this course, you will have enough knowledge to design and build complex digital systems for a variety of uses. These digital systems will contain both combinational and sequential parts.

1.5 Chapter Summary

This chapter has attempted to begin our discussion of digital systems by introducing the notion of binary values, binary number representations, and the use of binary data to represent real-world quantities. Further, a few words about analog vs. digital and combinational vs. sequential circuits were provided. At this point, rather than further delay beginning our study of digital systems design with more overview and examples, let's move on to the next chapter and begin to lay the foundational elements necessary to design and build real digital systems.

Chapter 3

Boolean Algebra and Truth Tables

We continue our study of digital systems design by examining Boolean Algebra - the basic method of describing and manipulating combinations of binary values. After that we turn our attention, in the next chapter, to the circuits which implement the computations described by Boolean Algebra.

3.1 Introduction to Boolean Algebra and Truth Tables

Boolean algebra is named after George Boole, the 19th century mathematician who developed it. It is an *algebra* over the values TRUE and FALSE. In developing this, Boole was not pursuing a method to reason about and design digital systems but rather was studying formal logic. Later, Claude Shannon pioneered the use of Boole's algebra to describe and manipulate binary variables with the goal of describing digital switching systems. Today, Boolean Algebra is the foundation upon which the specification and design of digital systems rests.

In Boolean Algebra, we map TRUE to the value 1 and FALSE to the value 0. A variable or constant whose range is taken from the set $\{0, 1\}$ is called a boolean variable or boolean constant.

Because it is an algebra, Boolean Algebra further includes operators for operating on the values 0 and 1. These operators include AND, OR, and NOT. The results of applying these operators to boolean values are fairly intuitive. For example, the expression " X AND Y " will be TRUE only when X is TRUE and Y is TRUE. The expression " X OR Y " is TRUE whenever X is TRUE or Y is TRUE. Finally, if X is TRUE then "NOT X " is FALSE and if X is FALSE then "NOT X " is TRUE.

Since boolean variables can take on only one of two values, it is possible to describe a boolean function by exhaustively enumerating all possible combinations of its input values and the corresponding output values. This is in direct contrast to conventional high-school algebra where enumerations are generally not useful. An enumeration in this respect can be done using a *truth table*. For example, the function of the AND operator can be described using the truth table of Figure 3.1(a).

A	B	$A \bullet B$	A	B	$A + B$	A	A'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

(a) AND (b) OR (c) NOT

Figure 3.1: The AND, OR, and NOT Operators

The arguments of the AND operator are shown on the left side of the truth table (A and B), and the result is shown on the right. The AND operator is often represented using the \bullet symbol and so the AND of A and B is $A \bullet B$. There are four possible combinations of values for the A and B arguments: (00, 01, 10, and 11). The only case where $A \bullet B$ is TRUE is when *all* of the function's inputs are true. In all other cases, $A \bullet B$ is FALSE. This matches precisely the word definition given previously.

Similarly, the function of an OR operation is TRUE whenever *any* of its inputs are TRUE as shown in the truth table of Figure 3.1(b). Here, the $+$ symbol is used to represent the OR operator, and $A + B$ is TRUE any time either A or B is TRUE. Once again this matches the word definition given previously.

Finally, the NOT operator is shown in Figure 3.1(c). It negates or *inverts* its argument. When writing boolean expressions a number of symbols have traditionally been used for the NOT operator. Examples of this include A' and \bar{A} . In this text, an apostrophe will be used. Thus, A' is the same as NOT A and \bar{A} .

3.2 Truth Tables for Arbitrary Functions

Truth tables can be used to describe any binary (boolean) function beyond simple AND, OR, and NOT. For example, the truth table of Figure 3.2 shows a function which is TRUE when its inputs are equal to one another.

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Figure 3.2: The Equivalence Function

There are two cases where the function is TRUE. The first case is the top row of the truth table and represents the case when both inputs are FALSE ($A' \bullet B'$). The second case is the bottom row of the truth table and represents the case when both inputs are TRUE ($A \bullet B$). In the other two cases the function is FALSE. A boolean expression representing this could be written as: $F = A' \bullet B' + A \bullet B$. Thus, an equation taken from a truth table is simply the OR of the rows with TRUE outputs.

As another example, consider the function shown in Figure 3.3. This function is only TRUE in a single case — when A is TRUE and B is FALSE. The function is FALSE for the other three cases. This would be written as: $F = A \bullet B'$.

A	B	F
0	0	0
0	1	0
1	0	1
1	1	0

Figure 3.3: Another Boolean Function

Finally, consider the function shown in Figure 3.4. In this truth table, the function is always TRUE. Following our methodology of listing the *product terms* corresponding to TRUE outputs for the function, this would be written as $F = A' \bullet B' + A' \bullet B + A \bullet B' + A \bullet B$. However, another way to write this would be: $F = 1$ (F is always TRUE).

A	B	F
0	0	1
0	1	1
1	0	1
1	1	1

Figure 3.4: A Trivial Boolean Function

Note that in the equations written for these functions, an implied precedence of operators was used. The order is that NOT has the highest precedence, followed by AND, and then followed by OR. This is similar to the precedence of multiply before add found in conventional arithmetic. If a different interpretation than the default precedence is desired, parentheses can be used.

Truth tables can be created for functions with more than two inputs. In this case the process is the same as above: (1) Enumerate all possible combinations of the function's inputs and then (2) for each such combination (row in the truth table) specify the function's output value. For a k -input function there are 2^k possible combinations and thus there will be 2^k rows in the truth table.

A 3-input function is shown in Figure 3.5. This function is TRUE any time at least two of its inputs are TRUE. A boolean equation representing it could be written as: $F = A' \bullet B \bullet C + A \bullet B' \bullet C + A \bullet B \bullet C' + A \bullet B \bullet C$.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 3.5: A 3-Input Function

3.3 Converting Truth Tables to Boolean Equations

From these examples it should be clear how to construct a boolean equation from a truth table. For each row which has a TRUE output (a '1'), write down the AND of the input values which it corresponds to. A careful examination of the previous figures will show how this is done in each case (begin by examining Figure 3.3 since there is only one product term). These product terms are then OR-ed together to form the final result.

The AND symbol (\bullet) is often omitted for clarity and so the function of Figure 3.5 could be written like this: $F = A'BC + AB'C + ABC' + ABC$. Additional examples of converting truth tables to boolean equations are shown in Figure 3.6. In each case, the number of *product terms* in the boolean equation is equal to the number of rows in the truth table which contain a '1' in the output column.

A	B	F
0	0	1
0	1	1
1	0	0
1	1	1

(a) $F = A'B' + A'B + AB$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

(b) $F = A'B + AB'$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

(c) $F = A'B' + A'B + AB'$

Figure 3.6: Some Additional Examples

3.4 Converting Boolean Functions to Truth Tables

Converting a boolean equation to a truth table is a relatively straightforward process. For example, consider the function: $F = A'B + AB$. This corresponds to a truth table with two rows that are TRUE (the '01' row and the '11' row). The corresponding truth table is shown in Figure 3.7. To ensure you understand how this is done, go back and review the previous figures and ensure you see how to map both directions (truth table \rightarrow equation as well as equation \rightarrow truth table.)

A	B	F
0	0	0
0	1	1
1	0	0
1	1	1

Figure 3.7: Truth Table for $F = A'B + AB$

A more interesting problem is mapping the following function to a truth table: $F = A + A'B$. In this equation, the first product term does not contain a B in either inverted or non-inverted form. Thus, the process is a bit different and results in the truth table shown in Figure 3.8.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Figure 3.8: Truth Table for $F = A + A'B$

Here, the A term corresponds to all rows for which A is TRUE. This includes the '10' row as well as the '11' row. The term $A'B$ corresponds to the '01' row. As another example, consider the 3-variable function shown in Figure 3.9 and its truth table.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
BC {	0	1	1
	1	0	0
	1	0	0
	1	1	0
BC {	1	1	1
	1	1	1

Figure 3.9: Truth Table for $F = AB + BC$

Here, the product terms overlap. That is the '111' row is covered by both AB as well as BC .

3.5 Boolean Identities and Theorems

As with regular algebra, boolean algebra has an associated set of identities and theorems. These are useful for doing algebraic manipulations on boolean expressions, usually with the goal of simplifying those expressions. A set of commonly used boolean identities and theorems are introduced in this section.

3.5.1 Single-Variable Theorems

$$A \bullet 0 = 0$$

This first theorem can be proven using a truth table as shown in Figure 3.10. This is done by listing the possible combinations of A in the truth table and AND-ing them with a '0'. Using the truth table of Figure 3.1(a) as a guide, the output column values can be determined. The resulting output column (all 0's) shows that indeed $A \bullet 0 = 0$.

A	0	A•0=0
0	0	0
1	0	0

Figure 3.10: Proof for $A \bullet 0 = 0$

In like manner, the following identity can be proven as shown in Figure 3.11.

$$A \bullet 1 = A$$

A	1	A•1=A
0	1	0
1	1	1

Figure 3.11: Proof for $A \bullet 1 = A$

Similarly, the following can be shown to be true:

$$A + 0 = A$$

$$A + 1 = 1$$

An additional set of theorems include the following which are proven in Figure 3.12

$$A' \bullet A = 0$$

$$A' + A = 1$$

A'	A	A'•A=0	A'	A	A'+A=1
1	0	0	1	0	1
0	1	0	0	1	1

(a) $A' \bullet A = 0$

(b) $A' + A = 1$

Figure 3.12: Proofs for Additional Theorems

3.5.2 Two-Variable Theorems

The following theorem is useful for simplifying boolean expressions:

$$A + A'B = A + B$$

When presented with a boolean expression, you can simplify it (reduce the number of terms and variables it contains) by applying this theorem. An expression with fewer terms and variables is preferable to one containing more terms and variables since it will often result in a smaller circuit. The proof of the theorem $A + A'B = A + B$ is shown in Figure 3.13.

A	B	A'B	A+A'B	A+B
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1



 $A+A'B = A+B$

Figure 3.13: Proof for $A + A'B = A + B$

In this figure, additional columns are added to truth table to build up the proof, piece-by-piece, using the basic definitions of AND, OR, and NOT. In the end, the columns labeled $A + A'B$ and $A + B$ contain the same combination of 0's and 1's. This shows that the two expressions have the same output for all possible input combinations and, are therefore, equivalent.

This theorem can also be used for simplifying more complex expressions using variable substitution:

$$(N + M + K') + (N + M + K')' \bullet WXY = (N + M + K') + WXY$$

where $(N + M + K') \rightarrow A$ and $WXY \rightarrow B$. A similar theorem which is easily proven using a truth table is given by:

$$A(A' + B) = AB$$

3.5.3 Commutative, Associative, and Distributive Theorems

Theorems similar to the commutative and associative laws hold true in Boolean Algebra:

$$A \bullet B = B \bullet A$$

$$(A \bullet B) \bullet C = A \bullet (B \bullet C)$$

$$A + B = B + A$$

$$(A + B) + C = A + (B + C)$$

These are easily extended to more than three variables. Boolean algebra also has *two* theorems similar to the distributive law:

$$A(B + C) = AB + AC$$

$$A + BC = (A + B)(A + C)$$

These distributive laws are proven in Figure 3.14.

A	B	C	B+C	A(B+C)	AB	AC	AB+AC	A	B	C	BC	A+BC	A+B	A+C	(A+B)(A+C)
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0
0	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1
1	0	1	1	1	0	1	1	1	0	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(a) $A(B + C) = AB + AC$ (b) $A + BC = (A + B)(A + C)$

Figure 3.14: Proofs for Distributive Theorems

3.5.4 The Simplification Theorems

Two of the most useful theorems for simplification of boolean expressions are these:

$$AB' + AB = A$$

$$(A + B')(A + B) = A$$

While the first can be proven using a truth table, it can also be proven using previously introduced theorems:

$$AB' + AB = A(B' + B) = A(1) = A$$

The second of these theorems can also be proven by *multiplying it out* and then applying the other theorems:

$$\begin{aligned}
 (A + B')(A + B) &= AA + AB + B'A + B'B \\
 &= A + AB + AB' + 0 \\
 &= A + A(B + B') + 0 \\
 &= A + A + 0 = \\
 &= A
 \end{aligned}$$

3.5.5 The Consensus Theorems

The final theorem which will be introduced is the consensus theorem:

$$AC + A'B + BC = AC + A'B$$

where the 3rd term in the left hand side expression is redundant and can be eliminated. A companion theorem is:

$$(A + C)(A' + B)(B + C) = (A + C)(A' + B)$$

While the consensus theorems are correct theorems, identifying opportunities to apply them when simplifying boolean expressions can be difficult.

3.6 Duality

An interesting feature of boolean algebra is the notion of *duality*. For any valid equality, its dual is also TRUE. The dual is formed by replacing AND operators with OR operators, replacing OR operators with AND operators, replacing '1' constants with '0' constants, and replacing '0' constants with '1' constants (do not change the variables). Since the following is TRUE:

$$A \bullet 1 = A$$

we immediately know that the following is also TRUE:

$$A + 0 = A$$

Further, since we know that this is TRUE:

$$A(B + C) = AB + AC$$

we immediately know that the following is also TRUE:

$$A + BC = (A + B)(A + C)$$

This is done by replacing the left side of the equation with its dual and setting that equal to the dual of the right side. It is important when forming the dual of an equality to correctly enforce precedence of operators (this is a common source of errors made by newcomers when forming duals). Since AND has a higher precedence than OR in expressions, parentheses are often needed as shown in this last example. A simple rule to follow is to place parenthesis around any AND terms in the original expression prior to doing the dual conversion. This will result in a correct dual.

Another common error is to mistakenly assume that “the dual of an expression is equal to the original expression.” This is not true. Rather, what is true is that “if an equality is true, the dual of the equality is also true.” This implies you must take the dual of both sides.

3.7 Summary of the Boolean Theorems

A set of useful boolean theorems is summarized in Table 3.1. Note the symmetry in the table — for every theorem in the left half of the table its dual is listed in the right half of the table. This table summarizes the set of boolean theorems which will be used throughout the remainder of this text.

Table 3.1: Some Useful Boolean Identities and Theorems

$A \bullet 0 = 0$	$A + 1 = 1$
$A \bullet 1 = A$	$A + 0 = A$
$A \bullet A' = 0$	$A + A' = 1$
$A + A'B = A + B$	$A(A' + B) = AB$
$AB = BA$	$A + B = B + A$
$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
$A(B + C) = AB + AC$	$A + BC = (A + B)(A + C)$
$AB' + AB = A$	$(A + B')(A + B) = A$
$AC + A'B + BC = AC + A'B$	$(A + C)(A' + B)(B + C) = (A + C)(A' + B)$

3.8 Chapter Summary

The basics of Boolean Algebra have been introduced in this chapter. It has been shown that every boolean expression has a corresponding truth table and that every truth table can be reduced to a boolean expression. A set of theorems, useful for simplifying boolean expressions was introduced. Proofs using truth tables were also introduced.

The key high-level points to master from this chapter include the following:

1. Boolean algebra is an algebra over the set $\{0, 1\}$ and the operators AND, OR, and NOT.
2. The boolean operators can be described by simple truth tables.
3. A truth table is an enumeration of all possible input combinations and the corresponding function output.
4. Truth tables for arbitrary boolean functions can be easily created.
5. For every truth table there is a corresponding boolean equation relating the truth table output to its inputs.
6. For every boolean equation there is a corresponding truth table.
7. Truth tables can be used to prove or disprove the truthfulness of a proposed equality.
8. A set of boolean theorems exist which can be used to simplify boolean expressions.
9. This set of boolean theorems can also be used to prove or disprove a proposed equality.
10. For every boolean theorem a dual theorem exists which is also true.

The skills that should result from a study of this chapter include the following:

1. The ability to create a truth table from a simple word description of a boolean function for any number of inputs.
2. The ability to write the boolean equation corresponding to a truth table.
3. The ability to create a truth table from a boolean equation.
4. The ability to prove or disprove a proposed equality using a truth table.
5. The ability to understand and apply the boolean theorems presented in this chapter.

3.9 Exercises

- 3.1. Draw the truth table for a 3-variable function whose output is TRUE any time an odd number of its inputs is TRUE.
- 3.2. Draw the truth table for a 4-variable function whose output is TRUE any time an even number of its inputs is TRUE. For purposes of this problem, zero is an even number.
- 3.3. Draw the truth table for a 4-variable function whose output is TRUE any time its inputs, when interpreted as the bits of a 4-bit unsigned binary number, is a multiple of 3 (consider 0 to be a multiple of 3).
- 3.4. For each of the problems above, write the boolean equation for the function by reading it off the truth table.
- 3.5. Prove that the following identity is TRUE using a truth table: $AC + A'B + BC = AC + A'B$. This theorem has a name. What is its name?
- 3.6. Prove that the following is TRUE by *multiplying it out* and then simplifying: $(A+BC)(A+DE) = A+BCDE$
- 3.7. Write the dual for the equality in the previous problem.