8

ARITHMETIC CIRCUITS

What to Expect

This chapter develops various types of digital arithmetic circuits, like adders and subtractors. It also introduces Arithmetic-Logic Units (ALUs), which combine both arithmetic and logic functions in a single integrated circuit. The following topics are included in this chapter.

- Developing and using half-adders, full adders, and cascading adders
- Developing and using half-subtractors, full subtractors, and cascading subtractors
- Combining an adder and subtractor in a single IC
- Describing ALUs
- Creating a comparator

8.1 ADDERS AND SUBTRACTORS

8.1.1 *Introduction*

In the Binary Mathematics chapter, the concept of adding two binary numbers was developed and the process of adding binary numbers can be easily implemented in hardware. If two one-bit numbers are added, they will produce a sum and an optional carry-out bit and the circuit that performs this is called a "half adder." If two one-bit numbers along with a carry-in bit from another stage are added they will produce a sum and an optional carry-out, and the circuit that performs this is function called a "full adder."

Subtractors are similar to adders but they must be able to signal a "borrow" bit rather than send a carry-out bit. Like adders, subtractors are developed from half-subtractor circuits. This unit develops both adders and subtractors.

8.1.2 *Half Adder*

Following is the truth table for a half adder.

Inputs		Output	
A	B	Sum	COut
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 8.1: Truth Table for Half-Adder

The *Sum* column in this truth table is the same pattern as an XOR) so the easiest way to create a half adder is to use an XOR gate. However, if both input bits are high a half adder will also generate a carry out (*COut*) bit, so the half adder circuit should be designed to provide that carry out. The circuit in Figure 8.1 meets those requirements. In this circuit, *A* and *B* are connected to XOR gate U1 and that is connected to output *Sum*. *A* and *B* are also connected to AND gate U2 and that is connected to carry-out bit *COut*.

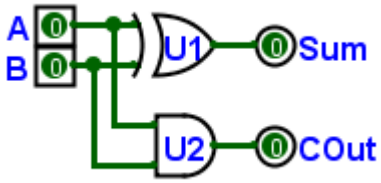

Figure 8.1: Half-Adder

8.1.3 *Full Adder*

A full adder sums two one-bit numbers along with a carry-in bit and produces a sum with a carry-out bit. Truth table 8.2 defines a full adder.

Inputs			Output	
A	B	CIn	Sum	COut
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

Table 8.2: Truth Table for Full Adder

Following are Karnaugh Maps for both outputs.

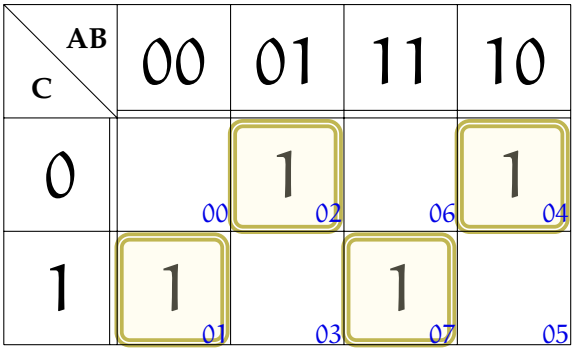

Figure 8.2: K-Map For The SUM Output

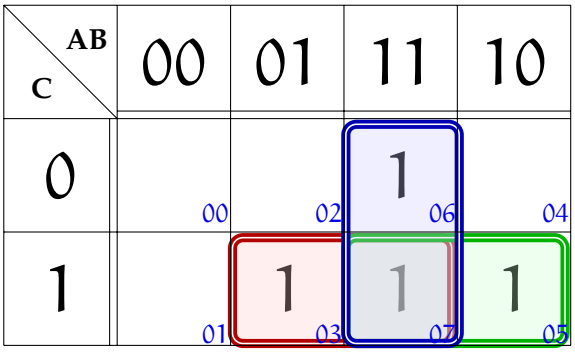

Figure 8.3: K-Map For The COut Output

Karnaugh map 8.2 is a Reed-Muller pattern that is typical of an XOR gate. Karnaugh map 8.3 can be reduced to three Boolean expressions. The full adder circuit is, therefore, defined by the following Boolean equations.

$$A \oplus B \oplus CIn = Sum \tag{8.1}$$
$$(A * B) + (A * CIn) + (B * CIn) = COut$$

Figure 8.4 is a full adder. In essence, this circuit combines two half-adders such that U1 and U2 are one half-adder that sums *A* and *B* while U3 and U4 are the other half-adder that sums the output of the first half-adder and *CIn*.

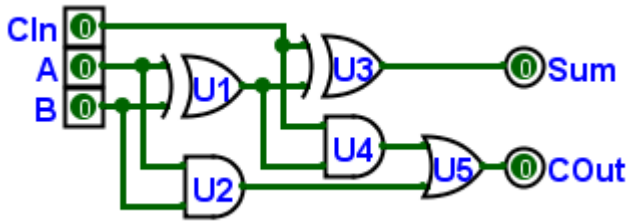

Figure 8.4: Full Adder

8.1.4 *Cascading Adders*

The full adder developed above will only add two one-bit numbers along with an optional carry-in bit; however, those adders can be cascaded such that an adder of any bit width can be easily created. Figure 8.5 shows a four-bit adder created by cascading four one-bit adders.

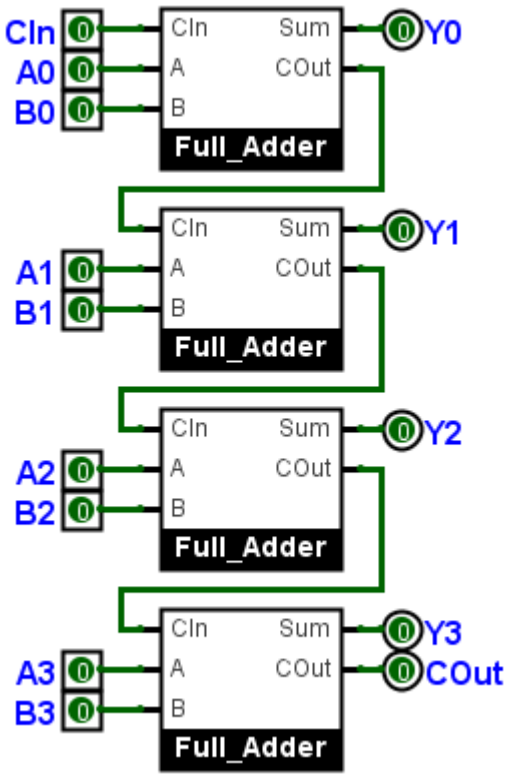

Figure 8.5: 4-Bit Adder

This circuit would add two four-bit inputs, *A* and *B*. Stage zero, at the top of the stack, adds bit zero from both inputs and then outputs bit zero of the sum, *Yo*, along with a carry-out bit. The carry-out bit from stage zero is wired directly into the stage one's carry-in port. That adder then adds bit one from both inputs along with the carry-in bit to create bit one of the sum, *Y1*, along with a carry-out bit. This process continues until all for bits have been added. In the end, outputs *Yo* - *Y3* are combined to create a four-bit sum. If there is a carry-out bit from the last stage it could be used as the carry-in bit for another device or could be used to signal an overflow error.

8.1.5 *Half Subtractor*

To understand a binary subtraction circuit, it is helpful begin with subtraction in a base-10 system.

```
 83
-65
 18
```

Since 5 cannot be subtracted from 3 (the least significant digits), 10 must be borrowed from 8. This is simple elementary-school arithmetic

but the principle is important for base-2 subtraction. There are only four possible one-bit subtraction problems.

```
  0      1      1      10
 -0     -0     -1     -1
  0      1      0      1
```

The first three examples above need no explanation, but the fourth only makes sense when it is understood that it is impossible to subtract 1 from 0 so 10_2 was borrowed from the next most significant bit position. The problems above were used to generate the following half-subtractor truth table.

Inputs		Outputs	
A	B	Diff	BOut
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 8.3: Truth Table for Half-Subtractor

Diff is the difference of *A* minus *B*. *BOut* ("Borrow Out") is a signal that a borrow is necessary from the next most significant bit position when *B* is greater than *A*. The following Boolean equations define the calculations needed for a half-subtractor.

$$\begin{aligned} A \oplus B &= Diff \\ A' * B &= BOut \end{aligned} \tag{8.2}$$

The pattern for *Diff* is the same as an XOR gate so using an XOR gate is the easiest way to generate the difference. *BOut* is only high when *A* is low and *B* is high so a simple AND gate with one inverted input can be used to generate *BOut*. The circuit in figure 8.6 realizes a half-subtractor.

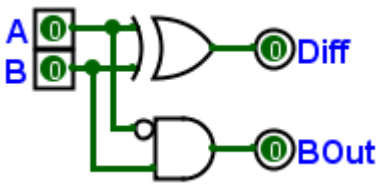

Figure 8.6: Half-Subtractor

8.1.6 *Full Subtractor*

A full subtractor produces a difference and borrow-out signal, just like a half-subtractor, but also includes a borrow-in signal so they can be cascaded to create a subtractor of any desired bit width.

Truth table 8.4 is for a full subtractor.

Inputs			Output	
A	B	BIn	Diff	BOut
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 8.4: Truth Table for Subtractor

Diff is the difference of *A* minus *B* minus *BIn*. *BOut* ("Borrow Out") is a signal that a borrow is necessary from the next most significant bit position when *A* is less than *B* plus *BIn*.

Following are Karnaugh Maps for both outputs.

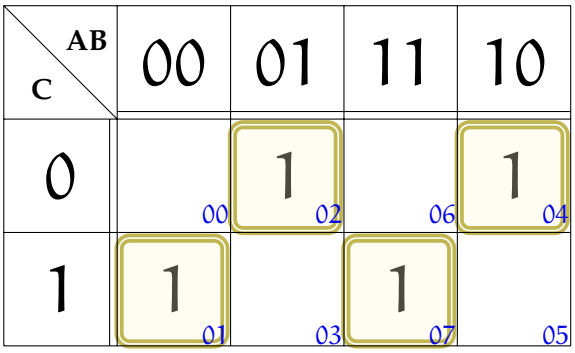

Figure 8.7: K-Map For The Difference Output

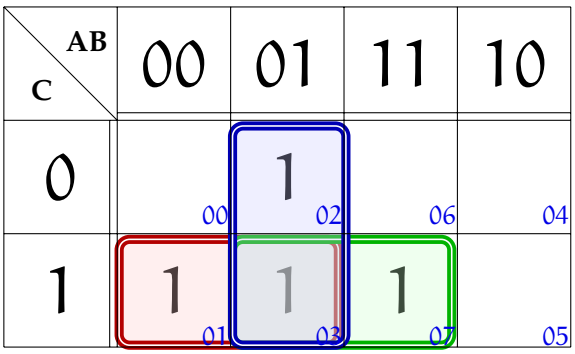

Figure 8.8: K-Map For The BOut Output

Karnaugh map 8.7 is a Reed-Muller pattern that is typical of an XOR gate. Karnaugh map 8.8 can be reduced to three Boolean expressions. The full subtactor circuit is, therefore, defined by the following Boolean equations.

$$\begin{aligned} A \oplus B \oplus BIn &= Diff \\ A'B + (A' * BIn) + (B * BIn) &= BOut \end{aligned} \tag{8.3}$$

The circuit in figure 8.9 realizes a subtractor.

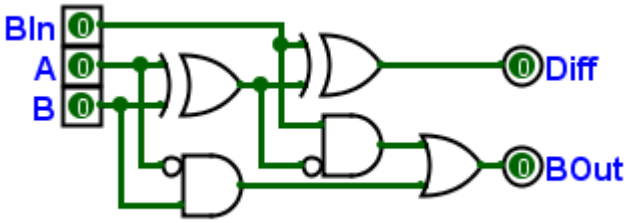

Figure 8.9: Subtractor

8.1.7 *Cascading Subtractors*

The full subtractor developed above will only subtract two one-bit numbers along with an optional borrow bit; however, those subtractors can be cascaded such that a subtractor of any bit width can be easily created. Figure 8.10 shows a four-bit subtractor created by cascading four one-bit subtractors.

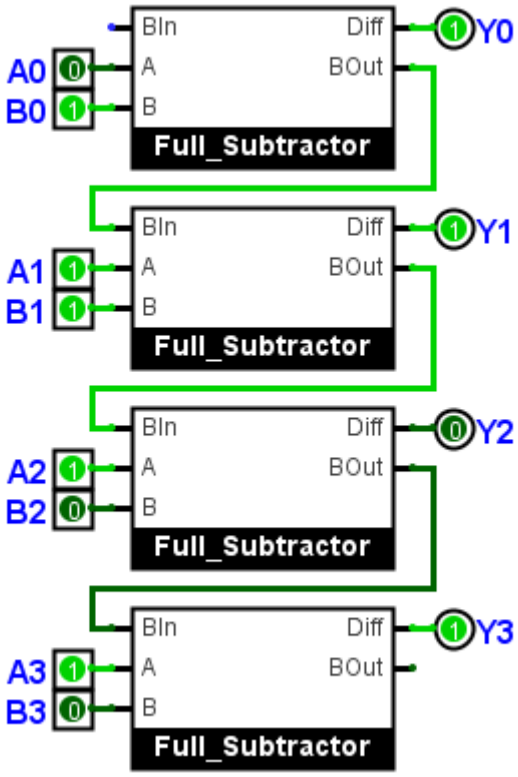

Figure 8.10: 4-Bit Subtractor

This circuit would subtract a four bit number B from A. The subtractor is set up to solve $1110_2 - 0011_2 = 1011_2$. Stage zero, at the top of the stack, subtracts bit zero of input B from bit zero of input A and then outputs bit zero of the difference, Yo, along with a borrow-out bit. The borrow-out bit from stage zero is wired directly into the stage one's borrow-in port. That stage then subtracts bit one of input B from bit one of input A along with the borrow-in bit to create bit one of the sum, $Y1$, along with a borrow-out bit. This process continues until all for bits have been subtracted. In the end, outputs Yo - $Y3$ are combined to create a four-bit difference. The borrow-out bit of the last stage is not connected to anything but it could be used as the borrow-in bit for another device.

8.1.8 *Adder-Subtractor Circuit*

It is remarkably easy to create a device that both adds and subtracts based on a single-bit control signal. Figure 8.11 is a 4-bit adder that was modified to become both an adder and subtractor. The circuit has been set up with this problem: $0101_2 - 0011_2 = 0010_2$.

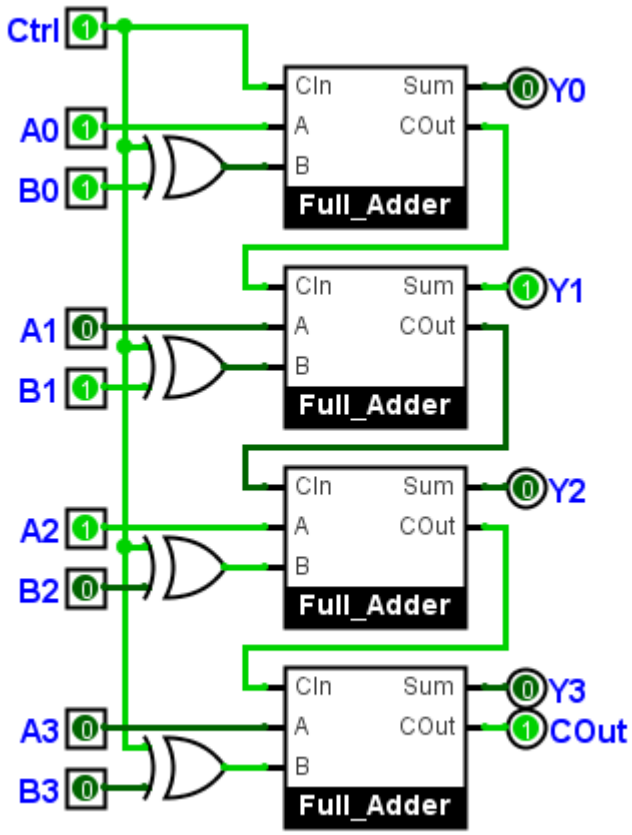

Figure 8.11: 4-Bit Adder-Subtractor

To change an adder to an adder-subtractor makes use of the binary mathematics concept of subtracting by adding the twos complement (see Section 3.2.5.2 on page 45). The "trick" is to use the XOR gates on input *B* to convert that input to its complement then the adder will subtract *B* from *A* instead of add.

To create the twos complement of a binary number each of the bits are complemented and then one is added to the result (again, this process is described in Section 3.2.5.2). Each of the *B* input bits are wired through one input of an XOR gate. The other input of that gate is a *Ctrl* ("Control") bit. When *Ctrl* is low then each of the *B* inputs are transmitted through an XOR gate without change and the adder works as an adder. When *Ctrl* is high then each of the *B* inputs are complemented by an XOR gate such that the ones complement is created. However, *Ctrl* is also wired to the *CIn* input of the first stage which has the effect of adding one to the result and turn input *B* into a twos complement number. Now the adder will subtract input *B* from input *A*.

In the end, the designer only needs to set *Ctrl* to zero to make the circuit add or one to make the circuit subtract.

8.1.9 *Integrated Circuits*

In practice, circuit designers rarely build adders or subtractors. There are many different types of manufactured low-cost adders, subtractors, and adder/subtractor combinations available and designers usually find it easiest to use one of those circuits rather than re-invent the proverbial wheel. A quick look at Wikipedia[1] found this list of adders:

- 7480, gated full adder
- 7482, two-bit binary full adder
- 7483, four-bit binary full adder
- 74183, dual carry-save full adder
- 74283, four-bit binary full adder
- 74385, quad four-bit adder/subtractor
- 74456, BCD adder

In addition to adder circuits, designers can also opt to use an ALU IC.

8.2 ARITHMETIC LOGIC UNITS

An ALU is a specialized IC that performs all arithmetic and logic functions needed in a device. Most ALUs will carry out dozens of different functions like the following few examples from a 74181 ALU (assume that the ALU has two inputs, A and B, and one output, F):

- F = NOT A
- F = A NAND B
- F = (NOT A) OR B
- F = B
- F = (NOT A) AND B
- F = A − 1
- F = A − B
- F = AB − 1
- F = −1

1 https://www.wikipedia.com/en/List_of_7400_series_integrated_circuits

ALUs are very important in many devices, in fact, they are at the core of a CPU. Because they are readily available at low cost, most designers will use a commercially-produced ALU in a project rather than try to create their own.

A quick look at Wikipedia[2] found this list of ALUs:

- 74181, four-bit arithmetic logic unit and function generator
- 74381, four-bit arithmetic logic unit/function generator with generate and propagate outputs
- 74382, four-bit arithmetic logic unit/function generator with ripple carry and overflow outputs
- 74881, Arithmetic logic unit

8.3 COMPARATORS

A comparator compares two binary numbers, A and B. One of three outputs is generated by the comparison: $A = B$, $A > B$, $A < B$. A one-bit comparator uses a combination of AND gates, NOT gates, and an XNOR gate to generate a *True* output for each of the three comparisons:

$A = B$	$(A \odot B)'$
$A > B$	AB'
$A < B$	$A'B$

Table 8.5: One-Bit Comparator Functions

Figure 8.12 is the logic diagram for a one-bit comparator.

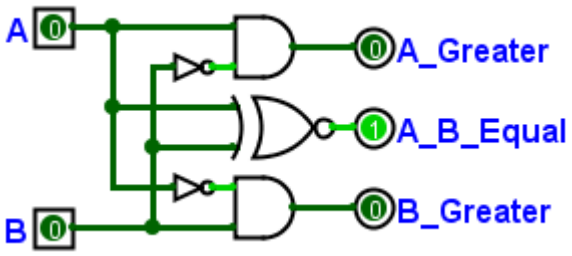

Figure 8.12: One-Bit Comparator

To compare numbers larger than one bit requires a more involved analysis of the problem. First, a truth table is developed for every possible combination of two 2-bit numbers, A and B.

2 https://www.wikipedia.com/en/List_of_7400_series_integrated_circuits

Inputs				Outputs		
A1	A0	B1	B0	$A < B$	$A = B$	$A > B$
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Table 8.6: Truth Table for Two-Bit Comparator

Next, Karnaugh Maps are developed for each of the three outputs.

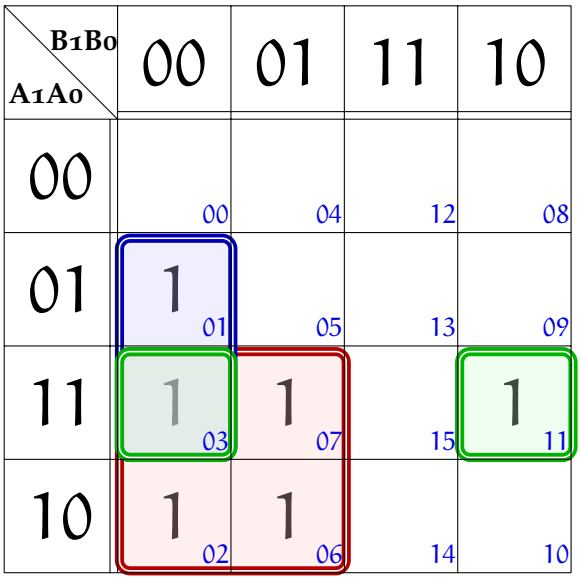

Figure 8.13: K-Map For $A > B$

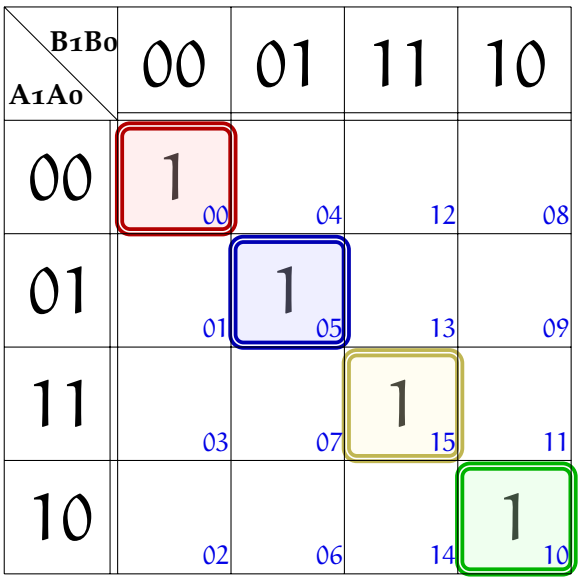

Figure 8.14: K-Map For $A = B$

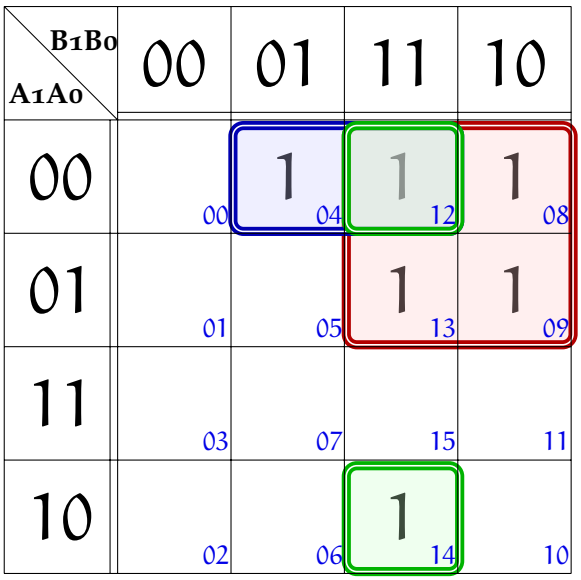

Figure 8.15: K-Map For $A = B$

Given the above K-Maps, the following Boolean Equations can be derived.

$$A < B : A1'B1 + A0'B1B0 + A1'A0'B0 \tag{8.4}$$
$$A = B : (A0 \odot B0)(A1 \odot B1)$$
$$A > B : A1B1' + A0B1'B0' + A1A0B0'$$

The above Boolean expressions can be used to create the circuit in Figure 8.16.

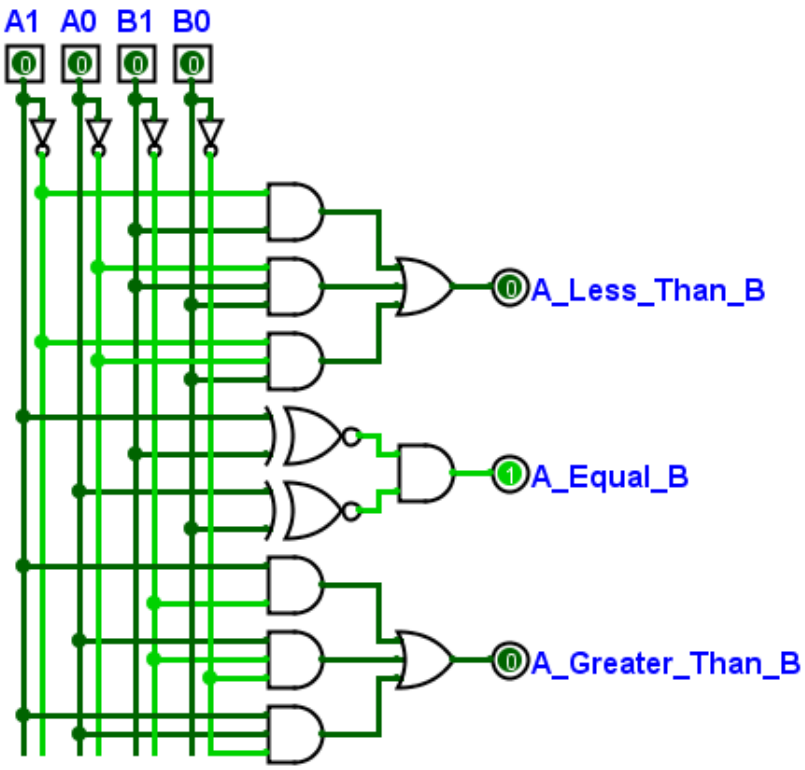

Figure 8.16: Two-Bit Comparator