# A Modified Logic Circuit of BCD Adder to Overcome the "Carry" Problems.

**Umana Thompson Itaketo, Ph.D.**

Department of Electrical/Electronics and Computer Engineering,
University of Uyo, Uyo, Nigeria.

E-mail: engr1easy@yahoo.com

## ABSTRACT

A Binary-Coded Decimal (BCD) code is defined and presented in this paper. The code is compared with the decimal and binary system. Advantages of the BCD code over the other two are explored. A logic circuit that adds two sets of BCD digits together, in parallel, is designed and its application illustrated. A major problem of the logic circuit (the parallel adder) is mentioned. A logic circuit that overcomes the problem is designed and incorporated into the parallel adder. The operation of the circuit, called the modified parallel adder, is illustrated.

(Keywords: binary, coded, decimal, BCD, digits, full adder, parallel, adding, carry, machine)

## INTRODUCTION

The Binary-Coded Decimal (BCD) code, sometimes referred to as the "8421" code, has each decimal digit expressed by its 4-bit binary equivalent. The "8421" basis for this code is built on the following descending powers of Figure 2: $2^3$ $2^2$ $2^1$ $2^0$. This can be seen to equal "8421". The BCD code is a code that combines the features of decimal and binary base number systems. The decimal base system is $base_{10}$ which we are all used to while the binary base system is $base_2$. Combining the features of these two systems allows a wider latitude to carry out digital mathematical operations.

An example of the decimal, BCD and binary equivalent is shown in Table 1 (Malvino, 2009). In this table, each decimal digit is converted to its 4-bit equivalent in BCD and binary.

It should be noted that 1001 is the largest 4-bit group in the BCD code. In other words, only 10 (0 – 9) of the (0 – 15) possible 4-bit groups are used in BCD. For this reason, the BCD code does not use the numbers 1010, 1011, 1100, 1101, 1110 and 1111. They are all forbidden numbers in the BCD code (Malvino, 2009). If any of these forbidden numbers appears in a machine using the BCD code, then one would know that an error has occurred.

The 8421 code (BCD code), is identical to the binary code up to the number "9". Beyond 9, it differs from the binary code. This may be visualized from table 1. For example, the binary number for 12 is 1100, but the 8421 number of 12 is 0001 0010.

The main advantage of the 8421 code is the ease of converting to and from the decimal numbers. Here, what is needed is only to remember the binary numbers of "0" through "9" because only one decimal digit is encoded at a time. The code (8421 code), however, has a major disadvantage. The disadvantage is that when adding two BCD numbers together, sometimes, true results will not be obtained (Atkin, 2008). The main aim of this paper is to point out that problem and provide a solution to it.

This paper also presents the circuit diagram and the working philosophy of addition using BCD adders.

## ADDITION OF TWO BCD NUMBERS TOGETHER

To add two 4-digit BCD numbers together, a parallel adder is often used. A parallel-adder is a combination of full-adder and half-adder logic circuits (Malvino, 2009). A full-adder uses a combination of two Exclusive – OR (Ex-OR), an OR and two AND logic gates as shown in Figure 1. A half-adder on the other hand is a combination of an Ex-OR and an AND logic gates as shown in Figure 2.

**Table 1:** An Illustration of the Decimal, BCD and Binary Numbers Equivalents.

| Decimal | 8421 (BCD) | Binary |
|---|---|---|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0010 |
| 3 | 0011 | 0011 |
| 4 | 0100 | 0100 |
| 5 | 0101 | 0101 |
| 6 | 0110 | 0110 |
| 7 | 0111 | 0111 |
| 8 | 1000 | 1000 |
| 9 | 1001 | 1001 |
| 10 | 0001 0000 | 1010 |
| 11 | 0001 0001 | 1011 |
| 12 | 0001 0010 | 1100 |
| 13 | 0001 0011 | 1101 |
| 14 | 0001 0100 | 1110 |
| 15 | 0001 0101 | 1111 |
| ⋮ | ⋮ | ⋮ |
| 98 | 1001 1000 | 0110 0010 |
| 99 | 1001 1001 | 0110 0011 |
| 100 | 0001 0000 0000 | 0110 0100 |
| 101 | 0001 0000 0001 | 0110 0101 |
| 102 | 0001 0000 0010 | 0110 0110 |
| ⋮ | ⋮ | ⋮ |
| 578 | 0101 0111 1000 | 0100 0100 0010 |
| ⋮ | ⋮ | ⋮ |

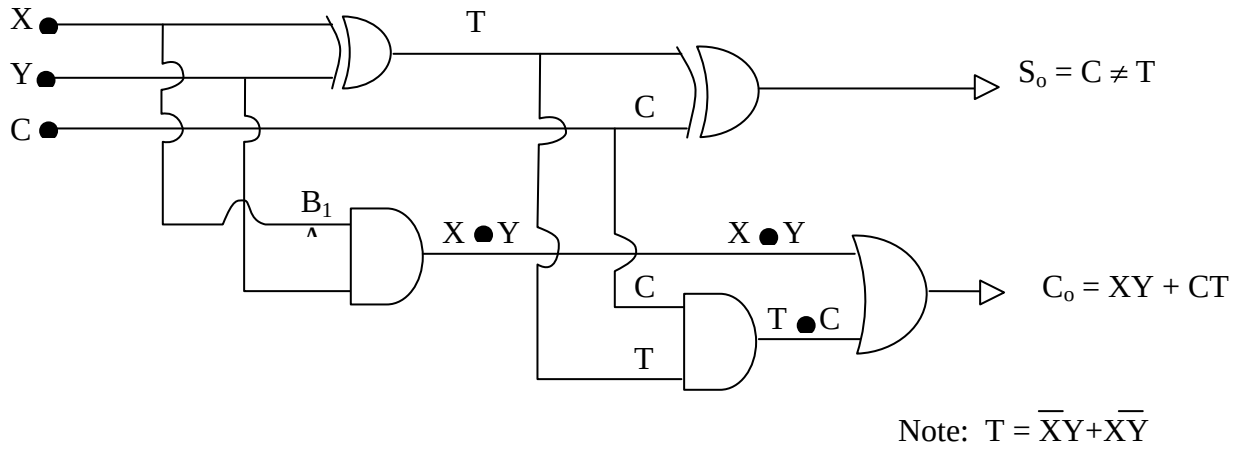


**Figure 1:** A Full-Adder Logic Circuit.

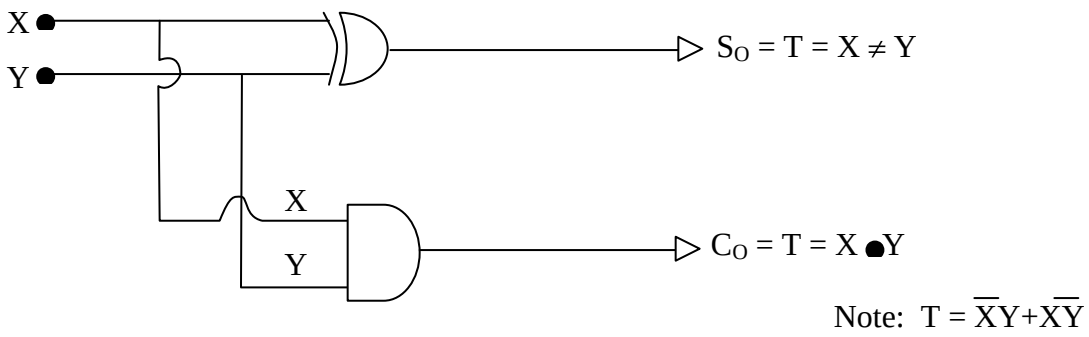


**Figure 2:** A Half-Adder Logic Circuit.

A full-adder is so-called because it has the provision to take an extra bit called a "CARRY" to add to its input digits and produce outputs, which include a possible "CARRY". A half-adder on the other hand, has no provision for any extra bit for a "CARRY" to add to its input digits. However, the two adders – full adder and half-adder, combine beautifully to give a parallel-adder with which two BCD numbers can be added together (Inyiama, 2008). A symbolic illustration of this combination is shown in Figure 3. In Figure 3:

- FA represents a Full-Adder
- HA represents a Half-Adder
- S0 represents the first stage output
- S1, S2 and S3, are also outputs from the respective stages
- C0 represents the first stage "CARRY",
- C1, C2 and C3, are also "CARRYs" from the respective stages.

Now, the addition of the two BCD numbers together goes this way: referring to Figure 3, two 4-digit BCD numbers A3 A2 A1 A0 and B3 B2 B1 B0 are to be added. The circuit to perform the first stage of the addition is the half-adder circuit shown in Figure 3.

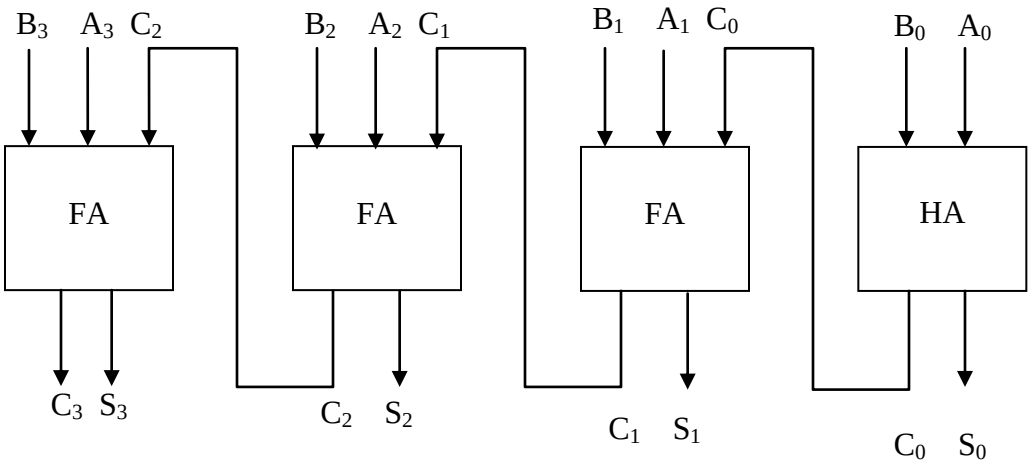


**Figure 3:** A Symbolic Illustration of a Parallel-Adder using Full-Adder and Half-Adder Logic Circuits.

At the beginning of the addition, two inputs, B0 A0, are available. After the addition, the sum S0 appears on the output side, with a possible "CARRY", C0. The "CARRY" from the first stage gets into the input of the second stage. Hence there are now 3 inputs at the second stage: B1 A1 and C0. Since there are 3 inputs, the required circuit is full-adder. The result/output from this addition is S1 with a possible "CARRY" C1. The "CARRY" from the second stage gets into the input of the third stage and are all added. The addition continues in this way with successive "CARRYs" until the final addition A3 + B3 + C2 is reached, all using full-adders. The result/output from the final stage is S3 (output sum) and a "CARRY," C3. This "CARRY" from the final stage is left at the immediate left of S3 hence a 5-bit binary result: C3 S3 S2 S1 S0, is obtained.

From the above illustration, when adding two BCD numbers together, using 4-bit parallel adders, a major problem arises. For example, if it is desired to add 0111 to 0100 (i.e. adding $7_{10}$ to $4_{10}$ using a 4-bit parallel-adder, the answer would be 11, in decimal base. The BCD addition operation would be:

$$\begin{array}{r} 0\,1\,1\,1 \\ +\ 0\,1\,0\,0 \\ \hline 1\,0\,1\,1 \end{array}$$

The result as can be seen, would be 1011. However, it would be recalled that the digits 1011, has no BCD meaning since it belongs to the forbidden BCD set of digits (Malvino, 2009). The desired result, 11 (in decimal), is 0001 0001, in BCD code. This result could not be obtained because the result is up to 10 and has exceeded that number. There is an overflow of a digit when the addition gets to 10. With a parallel adder, there is no way to detect when the addition gets to 10 and exceeds it. This problem of parallel adders has been removed through a proposed solution which is presented below.

## PROPOSED TECHNIQUE

A decimal, binary and BCD code shown in table 2 will be used to illustrate the solution to the problem of adding two BCD numbers when the value of the results is 10 or higher (Atkin, 2008).

**Table 2:** A Decimal, Binary and BCD Code, (up to decimal 15).

| Decimal | Binary Code | | | | "CARRY" | BCD Code | | | | f = A . (B + C) or f = $S_3$ . ($S_2$ + $S_1$) |
|---|---|---|---|---|---|---|---|---|---|---|
| | A $S_3$ | B $S_2$ | C $S_1$ | D $S_0$ | C | 8 | 4 | 2 | 1 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 13 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | |

This problem is overcome by adding the BCD equivalent of 6 (i.e. 0110) to the actual result, 1011.
The operation would be like this:

```
    1 0 1 1
+   0 1 1 0
  1 0 0 0 1
```

With this, the result would be 1 0 0 0 1, which is the same as the desired result, 10001. By breaking the result into 4-bit configuration, it can be expressed as 0001 0001

To implement this logic by circuitry, a logic circuit which detects when a sum is up to "$10_{10}$" and higher, and interprets it as a logic "1," is used to implement the "addition of 6" to an original BCD result to obtain what could have been obtained. The logic circuit is shown in Figure 4.

The logic circuit in Figure 4 is the type which detects any number from decimal 10 and above. It is used to implement the "addition of 6" to a BCD result in a scenario such as the one described. How it does this is that whenever the output, f, is logic "1," this is interpreted as a "10" in the decimal base.

By possessing such property, the circuit is effectively utilized in parallel adders to detect numbers from "10" and above and express them as overflows. Such overflows are entered in the "CARRY, (C)" column in Table 2. Whenever the operation: $S_3 . (S_1 + S_2) = 1$, this indicates an overflow from the 4th bit and to obtain a sensible cross-over from the $4^{th}$ bit to the $5^{th}$ bit, $(0110)_2$, which is $(6_{10})$, is added to the original sum. This logic has proved itself true without any loss of generality and without any distortion of mathematical truth.

For example, to express the number $12_{10}$ $(1100)_2$ in BCD Code, one would be tempted to write it as 1100 as though it were in $base_2$. However, by adding $(0110)_2$ which is $(6_{10})$ to 1100 the result obtained would be 10010. By breaking the result into 4-bits configuration, what is obtained is 0001 0010. By this, the number $12_{10}$ is sensibly expressed in 5 bits, in BCD code.

## THE PROPOSED CIRCUIT

The overflow detector circuit shown in Figure 4 will now be incorporated into the parallel adder circuit, Figure 3, in order to have a comprehensive circuit that would give BCD-based summation outputs, detect any overflow whenever it occurs, process the information and provide sensible results as would be expected. The modified circuit is illustrated in Figure 5.

The logic circuit in Figure 5 is a modified BCD parallel adder logic circuit. The operation of the circuit is as follows: The addition starts initially with a half-adder, having two inputs $B_0$ and $A_0$. The outputs from this sum are the sum, $S_0$ and the "CARRY, $C_0$". The "CARRY, $C_0$," goes into the second stage and constitutes a third input to that stage. Having three inputs $B_1$ $A_1$ and $C_0$, requires a full-adder.

The addition is carried out giving outputs of $S_1$ as sum and $C_1$ as a "CARRY". The "CARRY" $C_1$, from the second stage gets into the third stage and the addition continues in that way up to the final stage where the addition $B_3 + A_3 + C_2$ is performed. The output of this stage is correspondingly $S_3$, as sum and a "CARRY, $C_3$."

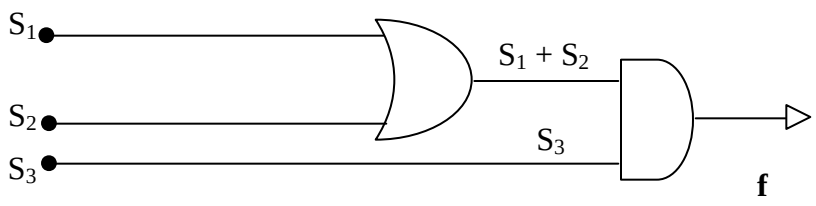


**Figure 4:** A Logic Circuit to Detect an Overflow from Parallel Adders.

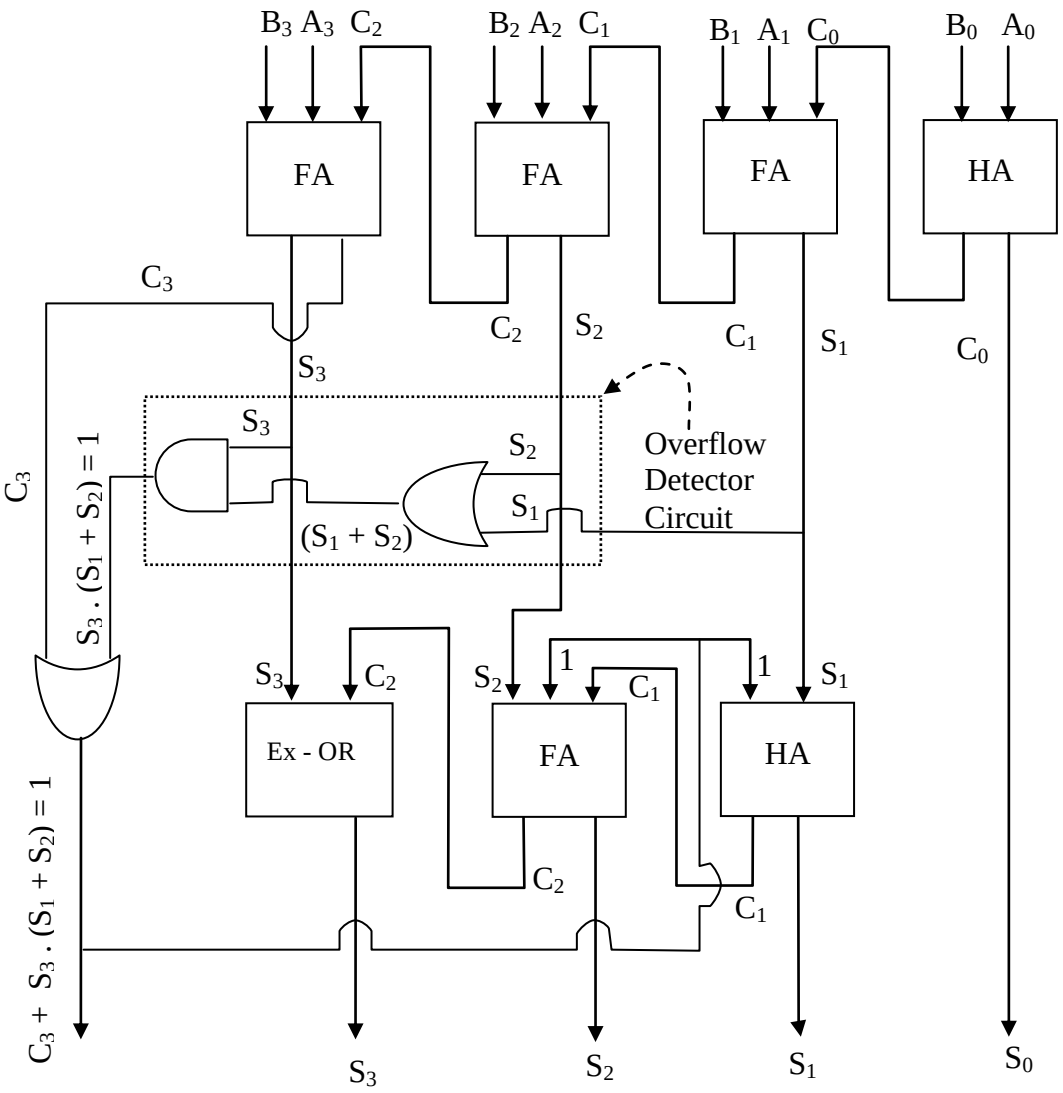


**Figure 5:** A Modified BCD Parallel Adder Logic Circuit for Overflow Detection.

The intermediate circuit, which is for overflow detection, as indicated in the box, will give an output based on the logic $S_3$ . ($S_1$ + $S_2$), to be 1 whenever there is an overflow. When this happens, $C_3$ is automatically brought in and ORed with the logic 1 to give an output of logic 1, recall, $C_3$ + 1 = 1, from Boolean Algebra (Holt, 2007). Now, this output of 1 is tapped to the input stages of the half-adder and full-adder respectively as illustrated in Figure 5. The addition is like this:

| | $S_3$ | $S_2$ | $S_1$ | $S_0$ |
|---|---|---|---|---|
| + | 0 | 1 | 1 | 0 |

In effect, this is adding $(0110)_2$, which is $(6)_{10}$, to the previous sum. The outputs from the half-adder are $S_1$ and $C_1$. $C_1$ is tapped to the input stage of the next adder (full-adder). The outputs from these adders are $S_2$ and $C_2$. $C_2$ is connected to the input stage of the immediate circuit to the left, which is an Exclusive-OR (Ex-OR) circuit. The output from the Ex-OR is $S_3$. Hence at last, the sum 1 $S_3$ $S_2$ $S_1$ $S_0$ is obtained and correctly/sensibly displayed in BCD pattern.

## DISCUSSION

The BCD code combines the features of the decimal number system ($base_{10}$) and the binary number system, ($base_2$). With these features, the BCD code provides a wide range of freedom for digital systems engineers to handle digital computations. With the BCD code, it is possible to carry out digital arithmetic to any extent of the decimal number system without the loss of mathematical truth and free from possible confusion, which would have been the case, if the binary base system were used. It is on this concept that several arithmetic machines, which were based on the binary system, had to be upgraded to have the capability to perform digital computations in the BCD code. However, in the process of executing these arithmetic computations using parallel-adders, sometimes the problem of false results does occur, with attendant confusion, as has been illustrated in the paper.

A digital system has been designed in the paper to correct that problem. As illustrated in Figure 5, the corrective circuit has been incorporated into the parallel-adder logic circuit and a comprehensive parallel-adder logic circuit has been obtained. The comprehensive parallel-adder logic circuit shown in Figure 5, has an infinite latitude to manipulate digital arithmetic in any dimension without room for error or confusion and without any distortion of mathematical truth.

**5.**

## CONCLUSION

The basis for a BCD logic has been presented. How to carry out digital arithmetic on the BCD Code has also been presented. The operation of a logic circuit that is used to implement the arithmetic has been described and illustrated. That logic circuit is known as a parallel-adder. A problem that arises while carrying out digital computation, using the parallel-adder, has been mentioned. A digital circuit that detects and corrects that problem has been designed and its application demonstrated. That circuit has been integrated into the main logic circuit (the parallel-adder) for digital computations and a modified (comprehensive) circuit for implementation of digital computation obtained.

An illustration of the operation of the modified logic circuit has been presented. The modified logic circuit will enable digital arithmetic on the BCD code, up to any number, (far higher than the binary code) to be carried out.

## REFERENCES


1. Atkin, J. K. 2008. *Computer Science*. MacDonald & Evans Ltd., The Chaucer Press: London, UK. 20–38.

2. Holt, C.A. 2007. *Electronic Circuits, Digital & Analog*. John Willey & Sons Inc.: New York, NY. 129–140.

3. Inyiama, H.C. 2008. “Lecture Notes on Digital Systems”. Department of Electrical Engineering & Computer Science, University of Science & Technology: Enugu, Nigeria. 60 – 78.

4. Malvino, A.P., et al. 2009. *Digital Principles & Applications*. Prentice Hall Int. Inc.: London,UK. 40–50.


## SUGGESTED CITATION


Itaketo, U.T. 2011. “A Modified Logic Circuit of BCD Adder to Overcome the “Carry” Problems”. *Pacific Journal of Science and Technology*. 12(2):246-252.

Pacific Journal of Science and Technology