4 Bit Binary Arithmetic Circuits with BCD Conversion

Timothy Rodgerson

Department of Engineering
The Division of Engineering and Applied Technologies
Gateway Community College

Email: timrodgerson2020@gmail.com

***Abstract* --- Mathematical arithmetic circuits are a foundational component to the Arithmetic Logic Unit (ALU) of modern microprocessors. A 4bit full adder/subtractor was built with displays to demonstrate an understanding of the operations and processing of binary data. Using Multisim, the addition circuit was built then tweaked for combining subtraction into a single circuit. Then, a second multiplication circuit was built implementing temporary memory and a Double-Dabble conversion method for the binary to BCD conversion. The adder and subtractor circuit will output correctly for only some of the input combinations, while the multiplier output without errors. Indeed, more circuitry can be incorporated to handle larger BCD values along with negatives. Overall, these circuits exhibit the necessities to perform elementary math.**

I. INTRODUCTION

Addition for computers is largely based on arithmetic from the decimal system. Typically, when doing this procedure people will write the numbers that are to be added in rows. Consequently, this puts the place holders in columns starting from 10^0 and exceeding to 10^n. The first column is to the extreme right and is the least significant column [X * 10^0] and pertains to the LSB in a binary bite. On the other hand, the last column to the left is the MSB. To start, the first step in the addition of any two numbers is combining them together into a group consisting of two separate parts. Those parts are the sum and carry. The sum is the least significant bit of the values being added and is noted within the same column. Next, the overflow value is carried to the next place holder (or the more significant column) for the next step of arithmetic. If there is no overflow the process remains the same and the two addends in the specific place holder are summed. However, if there is overflow, three numbers require addition at once. For example, (25+35=60): first both 5s are added because they are in the least significant column and the result is 10. Now, the 1 is the carry and the 0 is the sum. Therefore, the sum (the 0 of the 60) is noted in the first column and the 1 is carried to the next column to be added along with the other two values in that column originally (2 and 3). Again, this

will result in a sum (the 6 of 60) for this specific column and carry value (0 of 060, or [0 * 10^2]) for the next column. Essentially, the answer is a series of sums that end with a carry being the most significant bit. Conversely, as one might have noticed, the beginning of the process always starts with adding 2 values, then each consecutive step has the potential to require the addition of a 3rd value. This process can be applied to binary and is scalable, which makes it perfect for the basis of creating circuitry to perform mathematical functions.

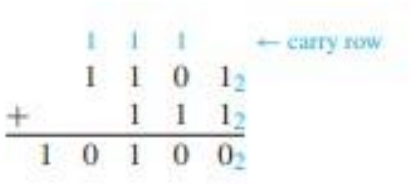


Figure 1: Binary addition demonstrating the sum and carry

One special fact about binary is that each place holder only can have one of two different values, 1 or 0. This means a truth table can be constructed and then translated to a circuit using logic gates for the task of performing the basic mathematic operations automatically. For example, when adding the number 0 to 0 in binary the answer is 0. Alternatively, when adding one to zero (0+1, 1+0) the answer is one. Consequently, an EXOR gate can be used to achieve the desired output of these three scenarios and is accordingly labelled the sum. Although both answers only occupy one place holder in a similar fashion and are the LSB of the value being summed. However, when adding 1+1 the answer is 2, but is written as (00000010 - with leading zeros for context) in binary. As a result, the LSB is a 0 and the remaining 1 must be carried to the next placeholder. Yet, the XOR gate still satisfies this scenario as it is returns LOW when both inputs are HIGH.

A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

(Exclusive-OR)

Figure 2: Truth Table for Exclusive OR

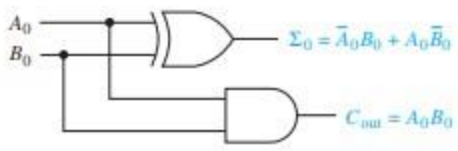
but in the three other instances (0+1, 0+0, 1+0) there is no value being carried to the next column and thus returns a 0. As mentioned before, the only time 1 is carried over is when both the addend and augend are 1. In other words, an AND gate is sufficient to handle the logic of a carry output.

AND

Input		Output
P	*Q*	*R*
1	1	1
1	0	0
0	1	0
0	0	0

Figure 3: Truth Table for AND

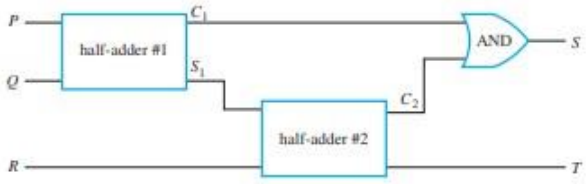
Together, a half-adder achieved by incorporating both concepts into a single circuit. Overall, a half-adder has two inputs A and B with two outputs sum and carry.



2 inputs | 2 outputs

A_0	B_0	Σ_0	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 4: Truth table and circuit diagram for a half adder



3 inputs | 2 outputs

A_1	B_1	C_{in}	Σ_1	C_{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: Circuit diagram and truth table for a full adder

In order to perform addition of greater values more logic is required. For example, a scenario when there is overflow from the previous arithmetic and the value is carried over to be added to the addend and augend of the next placeholder. The first step in doing this arithmetic is to sum original values in the column together and then add the carry to the result. As one might have figured, two half-adders should be able to handle this task, but there is one more piece. The solution is inputting the carry from the first half adder into an AND gate with the sum inputted into the next half adder. Next, the carry from the second half adder is inputted into the AND gate and the output of this is the sum of both and half adder while the sum of the second half adder becomes the carry value. Essentially, "The operation of the full-adder is based on the fact that addition is a binary operation: Only two numbers can be added at one time" (discrete math). Therefore, no matter how long the bit length or amount of values, the process remains the same. However, the necessary amount of arithmetic increases at a fixed rate of n-1.

Certainly, a circuit for adding two inputs with a bit length of 3 or greater is needed in modern electronics and the easiest way is to use a half-adder that is connected it to a full adder. This creates a parallel adder and can be scaled up indefinitely by adding a full adder for every subsequent bit. Another name for this design is a complete adder because it can be used to add an infinite bit length. Of course, that is not practical, and most computers are 32 or 64 bits, with 8 and 16 being used in the past. Interestingly, this means a half adder will always be the first component wired because there is never going to be a carry-in value and all succeeding components will be full adders. For example, an 8-bit complete adder would be composed of one half-adder and seven full-adders. The final carry out is the overflow.

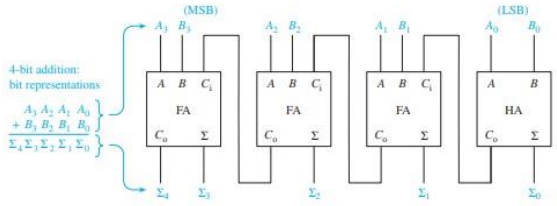


Figure 6: Complete parallel adder for 4 bits

II. METHODOLOGY

A. Addition

Normally, to add bits only one full adder is needed, but in order to display the output values correctly in decimal a conversion from binary to BCD had to be made. Since the LED displays have four inputs, hexadecimal values are displayed. For this reason, there will need to be a way for adding six to value if it is ten or greater. When this occurs, six will be added again to the sum. Because hexadecimal has sixteen values per placeholder and adding six to the sum will shift a 1 to the ten's column.

```
  0111
+ 0110
  1101   ←— invalid BCD
+ 0110   ←— add 6 to correct
1 0011
  ↖ carry to next BCD digit
```

Figure 7: BCD addition

B. Subtraction

Subtraction is done using the same circuitry. In other words, digital systems perform subtraction by addition. Therefore, the difference is found by converting the subtrahend to a negative and then adding it back to the minuend. This works because adding a negative value to another is the same as subtracting. As follows, the easiest method of changing binary values to a negative is by using the twos complement. This technique starts with raising the bit length as an exponent to the base of two and then subtracting one from that. Specifically, 2^8 - 1 = 256 - 1 = 255. In binary this can be written as 11111111. Next, the value being subtracted is converted to ordinary binary and is subtracted from the two's complement and then one is added back to the difference for the final answer. This may appear trivial, but "It turns out that there is a convenient way to compute two's complements that involves less arithmetic than direct application of the definition" (discrete math text). Simply flip each bit and then add 1. Find the ones complement is even easier and is found by just changing every bit. This can be achieved with an inverter. Commonly in computers, a byte is used to represent numbers from -127 to 128. This is because a truth table with 8 inputs can produce 256 different combinations, thus reserving half of these combinations for negative integers and the other half for the positive integers, including 0. Consequently, the leading bit, or the most significant bit, is the value sign for a number. 1 is negative and 0 is positive. Similarly, finding the one's complement is even easier as the only requirement is to change all the bits.

```
Two's complement of 18 = 0001 0010
+ Two's complement of −9 = 1111 0111
          Sum = 0000 1001 = +9₁₀   Answer
```

Figure 8: Two's Complement addition

C. Multiplying

Multiplying binary bits is the same process as it is for decimals. The process can be a bit long and tedious as it requires multiplying bit by bit and then adding all the products together. Interestingly, the act of

multiplying is actually very easy and can be done with a single AND gate. As mentioned earlier the gate only goes HIGH when both inputs are 1 and only time a product is acquired is when 1*1=1. Thus, cascading through each combination of multiplier and multiplicand several products are formed. As one might already be aware when multiplying, the length bit, or decimal answer, has the potential to double in size.

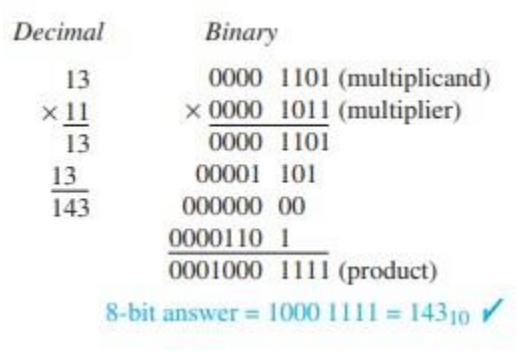


Figure 9: Binary multiplication process

Also, the answer will be converted to BCD. This will require a couple more steps than before and involve temporary memory and many more correction adders, which are slightly different. The solution is not going to be as simple of a process as it was with the adder/subtractor circuit because there is potential for the answer to exceed into the hundreds place. However, to solve this issue the same notion is needed. That is, for each place value, shift to the next placeholder once the number ten is met. There are many different solutions to this problem, but for sake of comprehensive purposes the double dabble method was chosen. Essentially, the input data gets shifted to left 4 times into the first BCD nibble and then 6 is added if the value is 10 or greater to prevent hexadecimals being displayed. Adding 6 is required because this is the amount of undesired hexadecimal digits (A-F). Finally, one last bit shift is required to obtain an accurate BCD value. Furthermore, since every place holder shift doubles the value, the requirements for adding and checking should be cut in half for the circuit to work properly. For this reason, 3 should be added when the nibble is 5 or greater.

Operation	Tens	Ones	Input
Binary Input			10110
Shift Left (and check if ≥ 5?)		1	0110
Shift Left		10	110
Shift Left (Ones >5: add 3)		**101**	10
Add 3=0011$_2$		**1000**	10
Shift Left	1	0001	0
Shift Left	10	0010	
BCD Output	2	2	

```
 101
+ 11
1000
```

Figure 10: Table showing double dabble steps

Normally, processing starts with the last three bits and ignores the first bit, leading to seven total double dabbles being used to convert 8bits to BCD. If the first bit were passed through the double dabble to be processed, unnecessary addition would take place and resulting in an incorrect display of output. For example, if 7 (0111) was inputted 3 would add to the nibble as it is greater than 5. This results in an output a of a hexadecimal digit "A" and this is obviously unwanted.

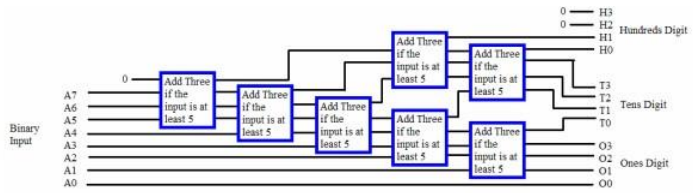


Figure 11: Double dabble being applied to a byte

However, this is technique is limiting since there is carry addition (addition of 3) within the circuit. Therefore, any input can only be used once. The reason is that extra data from the first process has the potential to be remaining and skew the next result. In

order to clear this unwanted data, a memory mechanism is required. To avoid this trouble, a parallel in and parallel out bit register was created using D flip flops with the set and resets both negated. Importantly, this allows the input value to be stored and then reset by a duty clock. Essentially, data is retrieved and cleared from the input all in one clock cycle. Meaning the moment, the input changes, the previous data has already been cleared thus eliminating lingering data. Overall, each double dabble needs its own memory mechanism for the desired output to be achieved.

III. Experimental Setup

A. Addition Circuit

A two-input addition circuit was built with two 74LS238's. In between is the modified carry out of the first chip. This solution requires only three logic gates with the output to be inputted into the correction adder to achieve a binary value of six. Since this is essentially a hijacked carry circuit, there is no connections to the carry in or out on the correction adder.

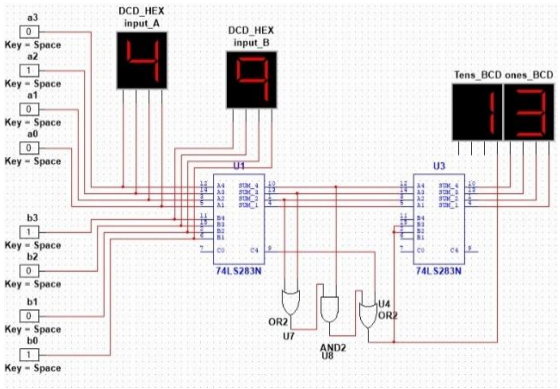


Figure 12: Addition circuit

B. Subtraction Circuit

Constructing the subtractor circuit is relatively simple. It was built by including XOR gates to the input, carry in, and the modified carry out of the first adder connected by a dipswitch. When the switch is on, subtraction is active and when it is open addition active. This implementation uses the one's complement method and simply flips the bits of the second input.

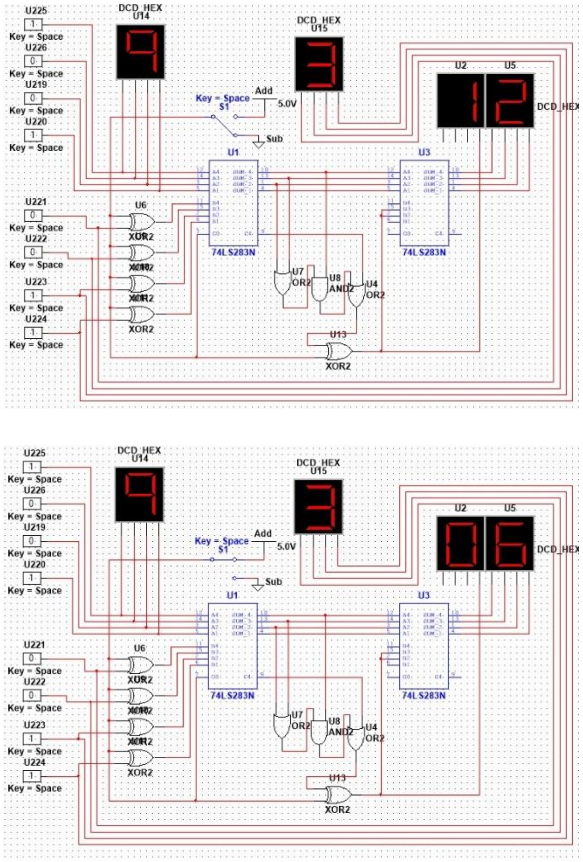


Figure 13: Addition circuit that includes subtraction. Displaying addition and subtraction

C. Multiplication Circuit

Since multiplying single binary bits results in the same output as an AND gate, all partial products can be obtained in this fashion. A two input 4-bit multiplier was built using three 4-bit full adders and sixteen AND

gates. Then, to add each partial product together, a full adder is required. However, the very first output bit derives from the first AND gate while all others are fed into full adders. Also, the first sum of the full adder is outputted as part of the final product. This leaves an empty space for the carry in to be wired. Next, the output from the multiplier is outputted into a binary to BCD converter that uses temporary memory, and lastly to the display. Also, there is a parallel wire connection at the input into a single double dabble to power a visual of the input.

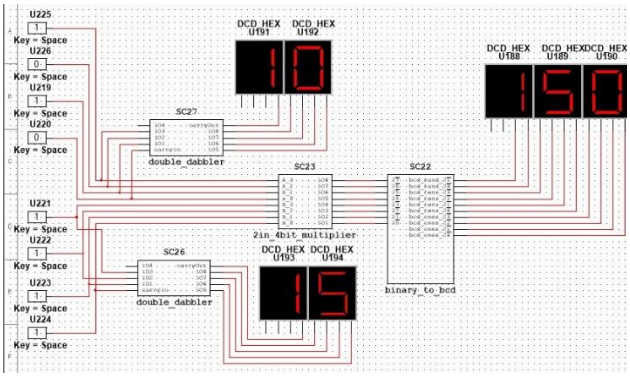


Figure 14: Complete multiplication circuit

The double dabble circuit is like the addition circuit show previously, except the double dabbler has slightly different connections for the modified carry and only has one full adder.

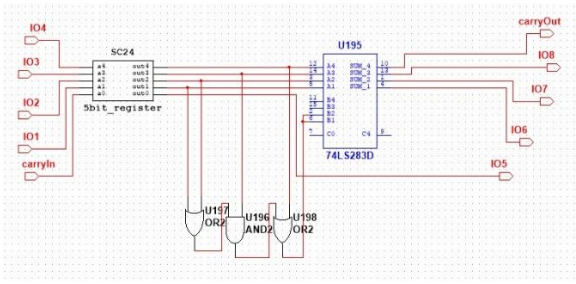


Figure 15: Double Dabble circuit

Setting up the 5-bit register uses five D flip flops in which the inputs are connected to the D pin. The output is connected to Q while all the flops are connected to the same duty clock cycle set to 1kHz.

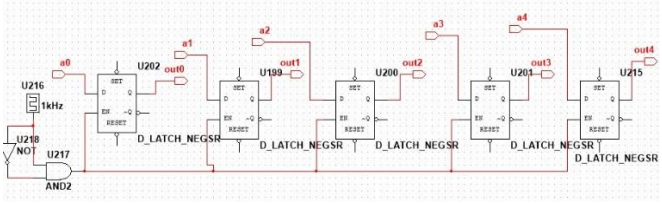


Figure 16: Circuit for 5-bit temporary register

V. RESULTS

The adder and subtractor has some outputs which are undesirable. Since there are only 5 bits of output, when using addition, the answer is limited to 19 even though the inputs have potential to reach 32. Also, there is no circuitry to handle negative numbers when subtracting. Perhaps adding circuitry that uses the twos complement to handle sign changes and negatives would be an improvement. Conversely, the multiplication circuit was successful in displaying all the correct results for the input combinations. Fundamentally, complexity will increase with larger input size and implementing other arithmetic such as division or more advanced trigonometry functions.

CONCLUSIONS

Even though these circuits are limited in their application, they provide the foundation for building more advanced ALUs that can power useful electronics such as calculators or error detection. On the other hand, the use of bit registers can be applied to timing circuits for other digital items like an alarm clock. Ultimately, knowledge and experience were the greatest gain from this research as many of these parts can apply to a plethora of other digital applications.

REFRENCES

[1] Kleitz, William. Digital electronics: a practical approach with VHDL. —9th ed.

[2] Discrete Mathematics with Applications, Fourth Edition Susanna S. Epp

[3] University of Pennsylvania Department of Electrical and Systems Engineering - https://www.seas.upenn.edu/~ese171/labise/Lab03_CombMultiplier.pdf

[4] C.B. Falconer. An Explanation of the Double-Dabble Bin-BCD Conversion Algorithm

[5] The D Type Flip Flop - https://www.electronics-tutorials.ws/sequential/seq_4.html