ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	3
EXPERIMENT DATE	:	07.04.2023
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G3

GROUP MEMBERS:

150200054 : ASLI YEL

150190028 : SEVİM EFTAL AKŞEHİRLİ

SPRING 2023

Contents

1 INTRODUCTION

Implementing arithmetic operations on binary numbers is one of the basic components of digital logic design. The design and implementation of logic circuits for arithmetic operations on both signed and unsigned binary numbers are being discussed in this experiment. We'll analyze the basic concepts behind these operations, use some implementation methods and algorithms, understand more about the importance of the 2's complement notation for providing a common method for these operations.

This experiment will be performed on a cumulative basis. First we will build a Half Adder, and then by using the Half Adder we will build a Full Adder, and finally we will work on a structure that is actually made up of four Full Adders and some extra components.

2 MATERIALS AND METHODS

2.1 PRELIMINARY

The following topics are reviewed for binary numbers, in 2's complement notation.

2.1.1 Signed and Unsigned Addition

Integer addition operation is performed out for unsigned and signed numbers in the same way thanks to the 2's complement representation. Signed and unsigned numbers, however, have different interpretations of their results.

If the numbers to be added are of different lengths, before the addition operation a sign extension is needed to be done for the number of shorter length, or it can be done for both numbers if a specific n-bit number system is desired. It should be noted that sign extension is performed in a different way for signed and unsigned numbers.

One more thing to be careful about is that when two n-bit signed numbers are added, the most significant bit of the result is always the (n)th bit and the sign bit of the result is always the most significant bit, so the (n)th bit. If a (n+1)st bit shows up in the result, that bit is ignored.

2.1.2 Signed and Unsigned Subtraction

To perform subtraction operations, computers generally choose the method of complements (1's complement and 2's complement). Similar to the addition, subtraction also operates on signed and unsigned numbers in the same way by taking the advantage of

2's complement system. However, for signed and unsigned numbers, the interpretation of the result differs, again similar to the addition.

Since the subtraction operation can be done by adding the 2's complement of the second operand to the first operand, only an addition circuit is enough to perform both addition and subtraction operations.

Just as for the addition operation, a sign extension may be necessary before the subtraction operation too.

2.1.3 Carry, Borrow and Overflow

Carry:

When there is an addition operation on unsigned numbers, a carry may occur.

This is when the result of the addition of two n-bit unsigned numbers is a (n+1) bit number. When there is an (n+1)st bit, Carry = 1; when the result is an n bit number, Carry = 0.

Presence of Carry means that the result cannot be represented with n bits only and that the (n+1)st bit is required. In this case, carry can be utilized to interpret the result.

Borrow:

When there is an subtraction operation on unsigned numbers, a borrow may occur.

If subtraction performed by utilizing 2's complement notation produces an (n+1) bit number (Carry = 1), the result is proper and there is no borrow.

On the other hand, if (n+1)st bit is zero (No Carry), there is a borrow and the result can not be represented with unsigned numbers. This also implies that the the first operand is smaller than the second one, the one being subtracted.

Overflow:

In the addition and subtraction operations on signed numbers an overflow may occur. It indicates the result cannot be represented with n bits.

By examining signs of the operands and the result, an overflow can be identified. There is four overflow cases, two for addition and two for subtraction. Conditions for these cases are as follow:

pos. + pos. $\rightarrow$ neg.

neg. + neg. $\rightarrow$ pos.

pos. - neg. $\rightarrow$ neg.

neg. - pos. $\rightarrow$ pos.

2.2 EXPERIMENTS

2.2.1 Part1 - Half Adder

In this part of the experiment we implemented a Half Adder.

The logical expressions for Sum and Carry outputs of a Half Adder with inputs A and B are:

$\text{Sum} = A \oplus B$

$\text{Carry} = A \cdot B$

A circuit for the Half Adder is designed accordingly.

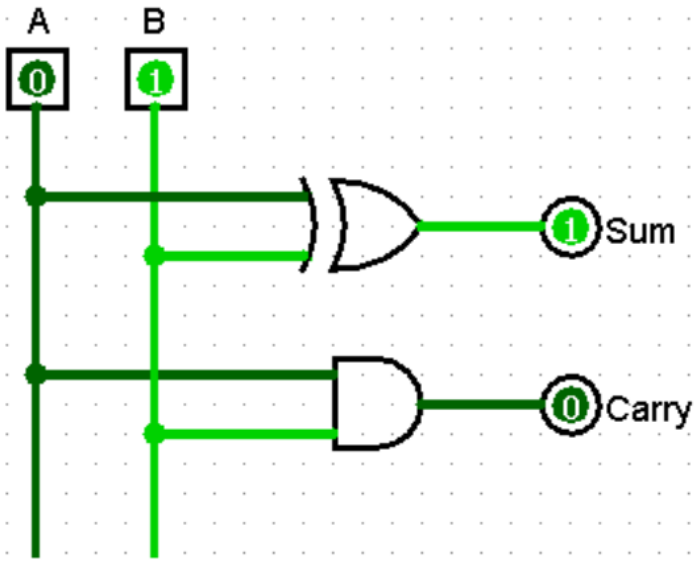

Figure 1: Design of a Half Adder

For the implementation of this circuit,

- C.A.D.E.T. (Complete Analogue Digital Electronic Trainer)
- 74000 series ICs
  - 74xx08 - Quadruple 2-input Positive AND Gates
  - 74xx86 - Quadruple 2-input Positive Exclusive Or (XOR) Gates

are used.

2.2.2 Part2 - Full Adder

In the second part of the experiment, we attached one more input (Carry Input) to the Half Adder, added some extra gates and rearranged the connections of outputs in order to obtain a Full Adder.

The logical expressions for Sum and Carry Out (Cout) outputs of a Full Adder with inputs A, B and Carry In (Cin) are:

$\text{Sum} = A \oplus B \oplus Cin$

$\text{Cout} = A \cdot B + A \cdot Cin + B \cdot Cin$

A circuit for the Full Adder is designed accordingly.

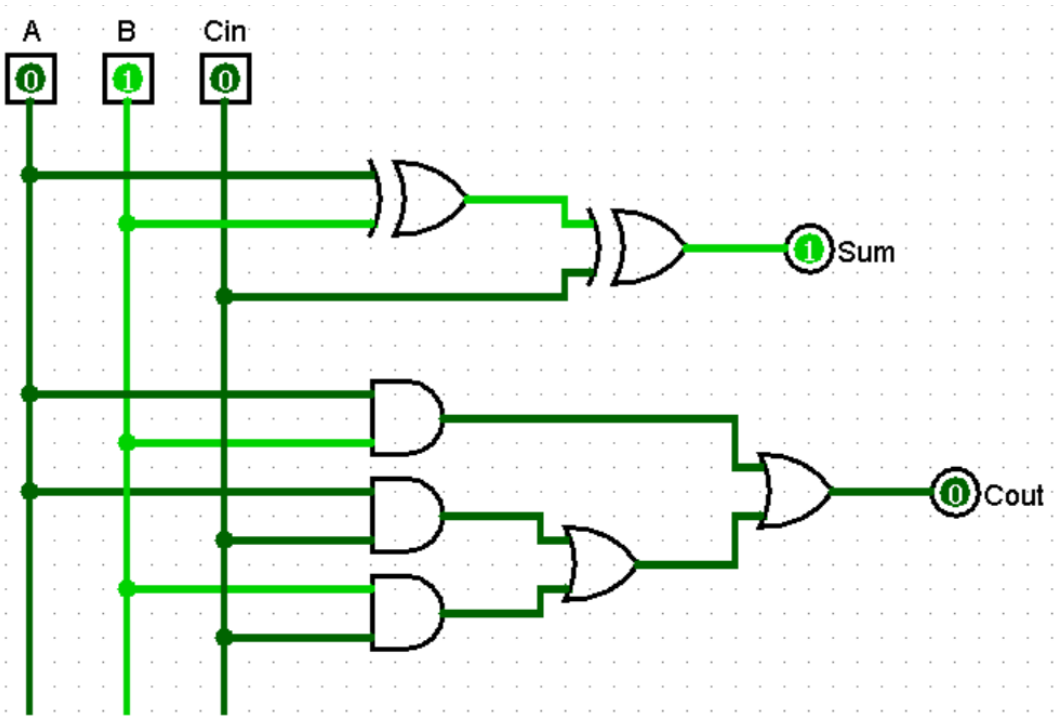

Figure 2: Design of a Full Adder

For the implementation of this circuit,

- C.A.D.E.T. (Complete Analogue Digital Electronic Trainer)
- 74000 series ICs
  - 74xx08 - Quadruple 2-input Positive AND Gates
  - 74xx32 - Quadruple 2-input Positive OR Gates
  - 74xx86 - Quadruple 2-input Positive Exclusive Or (XOR) Gates

are used.

2.2.3 Part3 - 4-Bit Adder/Subtractor

In the last part of the experiment we implemented a 4-Bit Adder/Subtractor.

A circuit that can be either a 4-Bit Adder or a 4-Bit Subtractor can be designed by using a 4-Bit Parallel Adder and four XOR Gates.

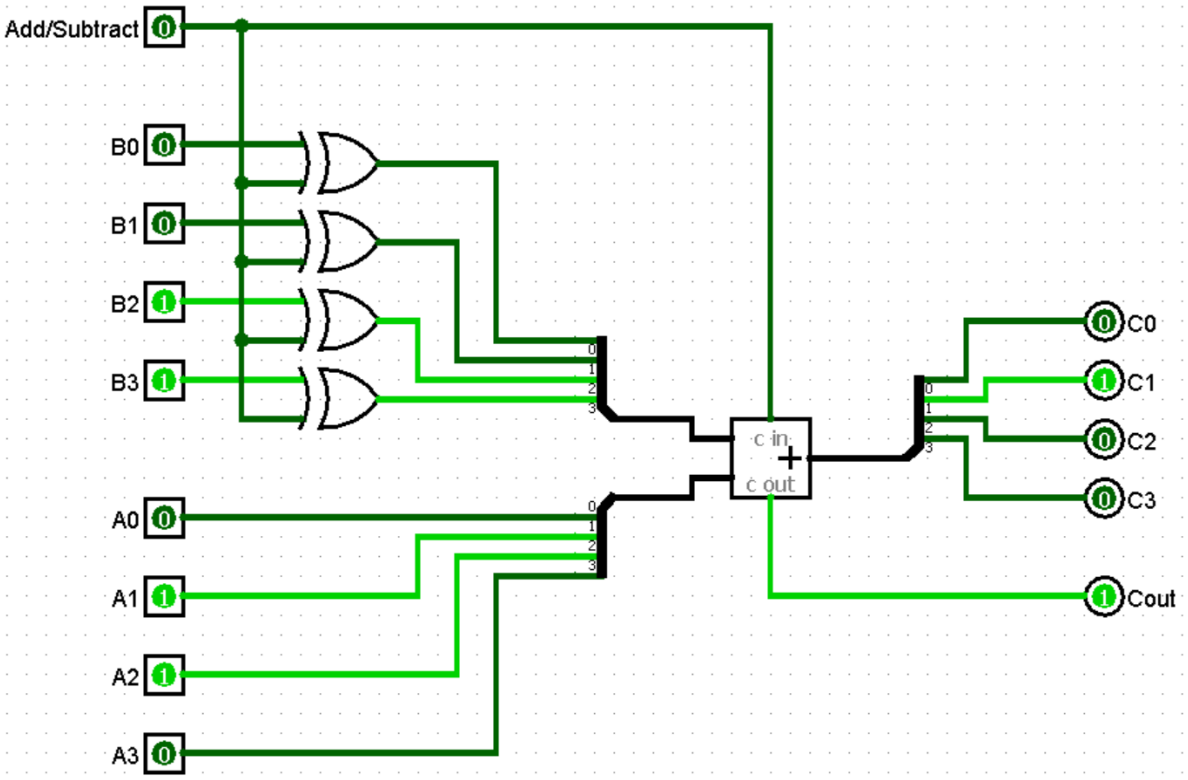

Figure 3: Design of 4-Bit Adder/Subtractor

If one input of a XOR Gate is "0", it functions as a buffer: $0 \oplus x = x$

If one input of a XOR Gate is "1", it functions as an inverter: $1 \oplus x = \bar{x}$

In our case,

When Add/Subtract = 0: Add/Subtract $\oplus$ Bn = Bn for n $\in \{0, 1, 2, 3\}$

$\Rightarrow$ B is forwarded to the parallel adder.

When Add/Subtract = 1: Add/Subtract $\oplus$ Bn = $\bar{B}$n for n $\in \{0, 1, 2, 3\}$

$\Rightarrow$ $\bar{B}$ is forwarded to the parallel adder.

Since Add/Subtract also feeds the Carry Input of the Parallel Adder;

When Add/Subtract = 0: C = A + B + 0 = A + B

$\Rightarrow$ The circuit behaves as an Adder.

When Add/Subtract = 1: C = A + $\bar{B}$ + 1 = A + (2's complement of B) = A - B

$\Rightarrow$ The circuit behaves as a Subtractor.

In conclusion, we can construst following function table for this circuit:

Add/Subt	B XOR Add/Subt	Operand 1	Operand 2	Cin	Result
0	B	A	B	0	A+B
1	$\bar{B}$	A	$\bar{B}$	1	A-B

Table 1: Function Table for a 4-Bit Adder/Subtractor

For the implementation of this circuit,

- C.A.D.E.T. (Complete Analogue Digital Electronic Trainer)
- 74000 series ICs
  - 74xx83 - 4-bit Binary Full Adder
  - 74xx86 - Quadruple 2-input Positive Exclusive Or (XOR) Gates

are used.

3 RESULTS

3.1 Part1 - Half Adder

Here is the Half Adder circuit that we implemented on CADET:

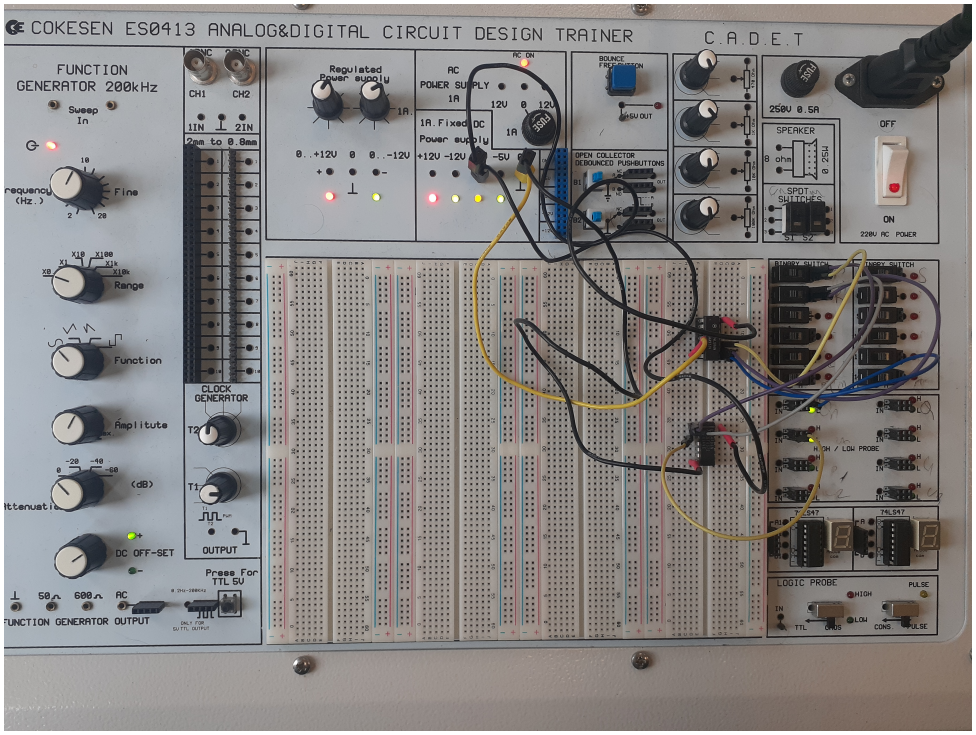

Figure 4: Implementation of a Half Adder

After implementation we tested our circuit and filled the truth table of Half Adder as follows:

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

Table 2: Truth Table for a Half Adder

3.2 Part2 - Full Adder

Here is the Full Adder circuit that we implemented on CADET:

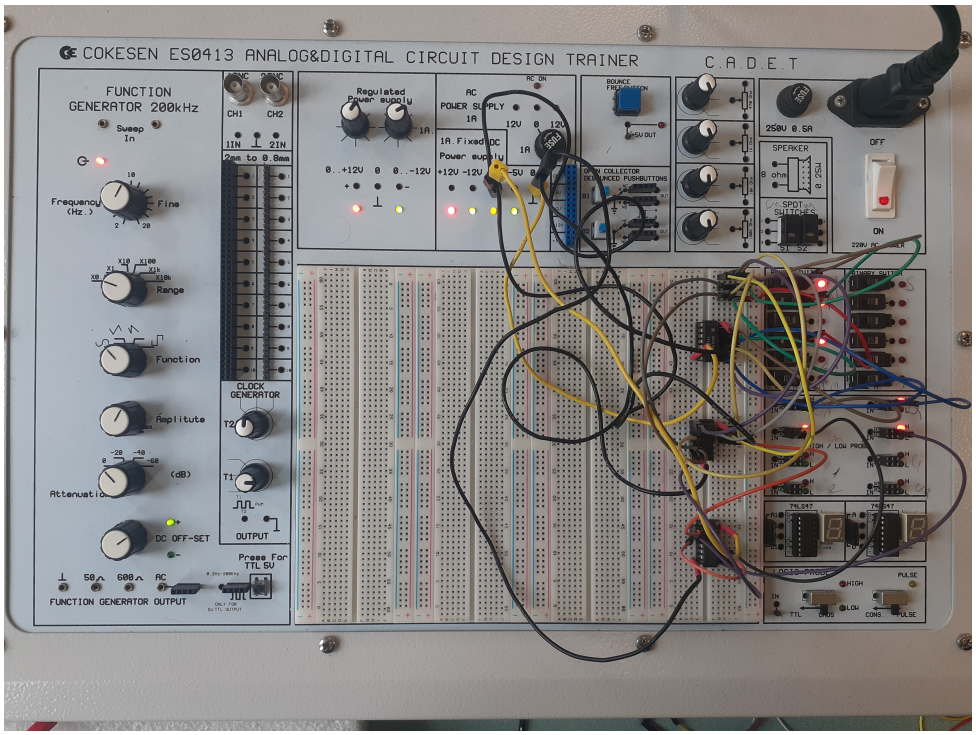

Figure 5: Implementation of a Full Adder

After implementation we tested our circuit and filled the truth table of Full Adder as follows:

Cin	A	B	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 3: Truth Table for a Full Adder

3.3 Part3 - 4-Bit Adder/Subtractor

Here is the 4-Bit Adder/Subtracter circuit that we implemented on CADET:

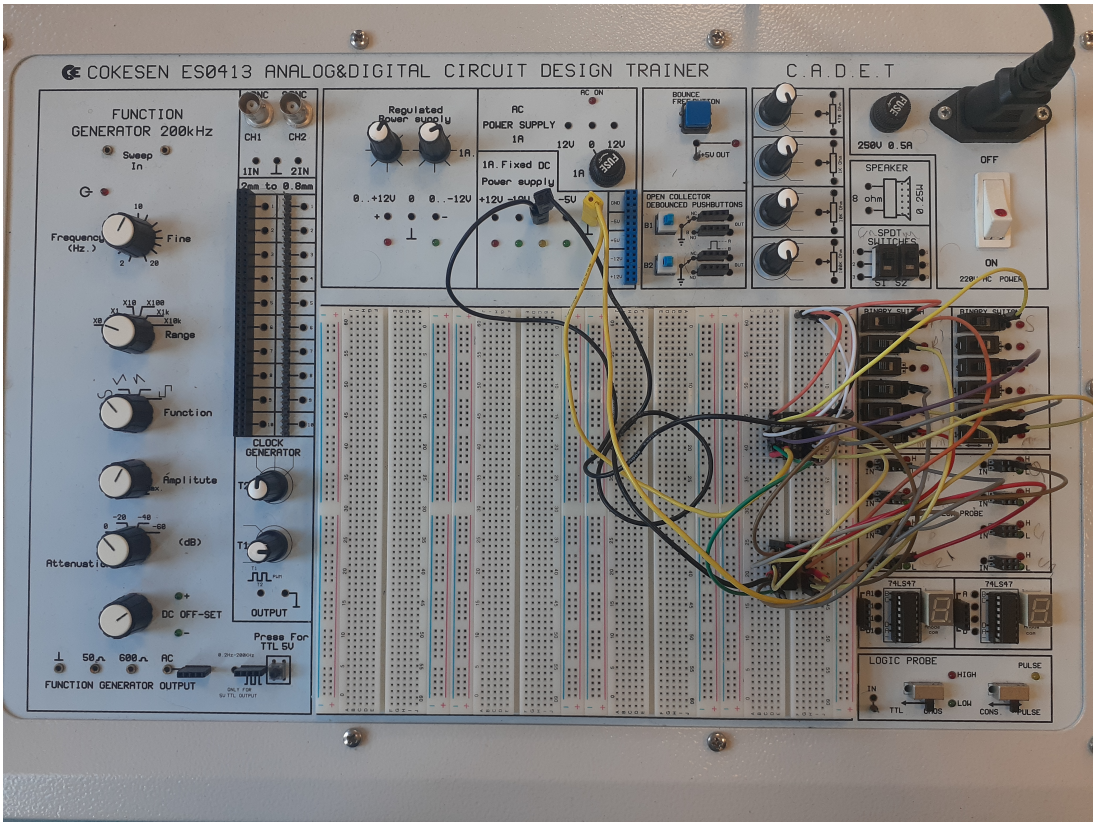

Figure 6: Implementation of 4-Bit Adder/Subtracter

After implementation we tested our circuit for some test cases with both addition and subtraction operations and for each operation we interpreted the obtained results as both signed and unsigned, and filled a table for each specific test.

Test Case	A	B
1	0101	0111
2	1101	1001
3	1111	1111
4	0110	1101

Table 4: Test Cases for 4-Bit Adder/Subtracter

Test Case 1: A = 0101, B = 0111

- Calculate: A + B, Interpret as: Unsigned

A	B	Carry	Result in Binary	Result in Decimal
0101	0111	0	1100	12

Table 5: First Test for Test Case 1

- Calculate: A + B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
0101	0111	occurs	negative	1100	-4

Table 6: Second Test for Test Case 1

- Calculate: A - B, Interpret as: Unsigned

A	B	Borrow	Result in Binary	Result in Decimal
0101	0111	1	1110	14

Table 7: Third Test for Test Case 1

- Calculate: A - B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
0101	0111	does not occur	negative	1110	-2

Table 8: Fourth Test for Test Case 1

Test Case 2: A = 1101, B = 1001

- Calculate: A + B, Interpret as: Unsigned

A	B	Carry	Result in Binary	Result in Decimal
1101	1001	1	0110	6

Table 9: First Test for Test Case 2

- Calculate: A + B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
1101	1001	occurs	positive	0110	+6

Table 10: Second Test for Test Case 2

- Calculate: A - B, Interpret as: Unsigned

A	B	Borrow	Result in Binary	Result in Decimal
1101	1001	0	0100	4

Table 11: Third Test for Test Case 2

- Calculate: A - B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
1101	1001	does not occur	positive	0100	+4

Table 12: Fourth Test for Test Case 2

Test Case 3: A = 1111, B = 1111

- Calculate: A + B, Interpret as: Unsigned

A	B	Carry	Result in Binary	Result in Decimal
1111	1111	1	1110	14

Table 13: First Test for Test Case 3

- Calculate: A + B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
1111	1111	does not occur	negative	1110	-2

Table 14: Second Test for Test Case 3

- Calculate: A - B, Interpret as: Unsigned

A	B	Borrow	Result in Binary	Result in Decimal
1111	1111	0	0000	0

Table 15: Third Test for Test Case 3

- Calculate: A - B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
1111	1111	does not occur	positive	0000	0

Table 16: Fourth Test for Test Case 3

Test Case 4: A = 0110, B = 1101

- Calculate: A + B, Interpret as: Unsigned

A	B	Carry	Result in Binary	Result in Decimal
0110	1101	1	0011	3

Table 17: First Test for Test Case 4

- Calculate: A + B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
0110	1101	does not occur	positive	0011	+3

Table 18: Second Test for Test Case 4

- Calculate: A - B, Interpret as: Unsigned

A	B	Borrow	Result in Binary	Result in Decimal
0110	1101	1	1001	9

Table 19: Third Test for Test Case 4

- Calculate: A - B, Interpret as: Signed

A	B	Overflow	Result Sign	Result in Binary	Result in Decimal
0110	1101	occurs	negative	1001	-7

Table 20: Fourth Test for Test Case 4

4 DISCUSSION

In the first part of the experiment where we implement a half adder, we first created our A and B inputs with switches on the CADET. We have obtained our Sum output by using a 2-input XOR Gate that takes A and B as its inputs. Then we have had our Carry output using an 2-input AND Gate that takes A and B as its inputs. In order to display them, we connected this outputs to the LEDs on the CADET. We tested our circuit for different values of A and B, and filled up a truth table accordingly.

When we moved on to the second part of the experiment, our goal was to design a Full-Adder and we implemented it over the Half Adder that we implemented earlier. So, we added one more input for the Cin (Carry In) in addition to the inputs of the previous part. To obtain the Sum output of the Full Adder we used one more XOR Gate that takes the Sum output of the Half Adder and Cin input. Basically, we performed Sum = (A $\oplus$ B) $\oplus$ Cin operation and got the Sum value as the final result. To obtain the Cout (Carry Output) value, we first obtained the values of A $\cdot$ Cin and B $\cdot$ Cin with two 2-input AND Gates, in addition to the A $\cdot$ B that we obtained in the previous part. To perform the operation Cout = A $\cdot$ B + A $\cdot$ Cin + B $\cdot$ Cin, we wanted to use a 3-input OR Gate, but we did not have one. Therefore we performed two consecutive OR operations such that first OR is over two of these three products and the second OR is over the result of the first OR and the third product. The result of the second OR gave us the value of Cout. Again, we tested our circuit for different values of A, B and Cin and filled up a truth table accordingly.

In the last part of the experiment, we determined 4-bit A and 4-bit B inputs for the binary numbers on which operations will be performed, paying attention to the order of the digits from MSB to LSB. We also created another input for the operation change between addition and subtraction. While connecting the digit values of A directly to the 4-bit full adder as the first operand's digits, on the other hand, we first performed XOR operations on the each digits of B and the operation selection input (Add/Subtract) by using four 2-input XOR Gates and forwarded the outputs of these XOR Gates to the full adder as the second operand's digits. This was to fulfill Adder/Subtractor's working principle that takes the advantages of special properties of a XOR Gate when one of its input is 0 or 1, 2's complement notation of binary numbers, and Cin input of full adder and combines them in a perfect way. In the end, when the value of operation selection input is 0, we do the addition with the Full Adder in an ordinary way by keeping the B as it is; and when it is 1, we first get the 1's complement of B by taking XOR of its digits with 1 (value of the operation selection input), then send the value 1 to the full adder

as Cin, get the 2's complement of B by adding this 1 to the 1's complement of B and perform subtraction by adding 2's complement of B to A. After completing our circuit without encountering a serious problem, we testes our circuit with given test cases. For each test case we have performed both addition and subtraction operation, interpreted the numbers as both signed and unsigned, and observed the results for carry, borrow, and overflow states.

5 CONCLUSION

In this experiment, we worked with the CADET interface to create half adders, full adders, and 4-bit adder/subtractors. We also practiced implementing the "74xx83 - 4-bit Binary Full Adder" component. We analyzed it with the given test case values. The results of carry, borrow, and overflow cases were examined. We learned more about the methods used by computers to deal with both signed and unsigned numbers for both addition and subtraction operations in the same way. Other than a little confusion about the order of digits from MSB to LSB and therefore the misinterpretation of the inputs A and B in the last part of the experiment, we did not face a difficulty and completed the experiment successfully.

REFERENCES