ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	3
EXPERIMENT DATE	:	02.04.2021
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G14

GROUP MEMBERS:

150180112	:	ÖMER MALİK KALEMBAŞI
150190014	:	FEYZA ÖZEN
150190108	:	EKİN TAŞYÜREK

SPRING 2021

Contents

1 INTRODUCTION [10 points]

In this experiment, we implemented AND, OR, NOT and XOR gates by using Vivado. With this experiment, we aimed to implement 1-Bit Half Adder, 1-Bit Full Adder, 4-Bit Full Adder, 16-Bit Full Adder, and Adder-Substractor circuits.

2 MATERIALS AND METHODS [40 points]

2.1 PART 1

In this part, we used Verilog operators such as ‘’ (Bitwise AND), ‘—’ (Bitwise OR), and ‘’ (Bitwise NOT) operations to implement AND, OR, NOT, and XOR Gates.

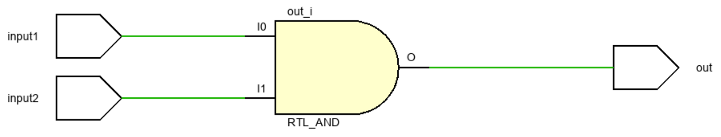

Figure 1: AND Gate RTL Schematic

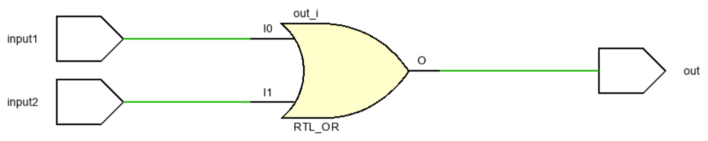

Figure 2: OR Gate RTL Schematic

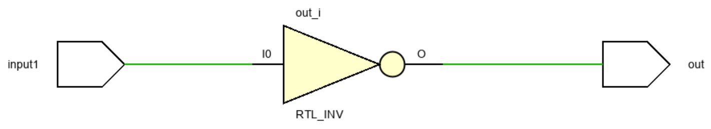

Figure 3: NOT Gate RTL Schematic

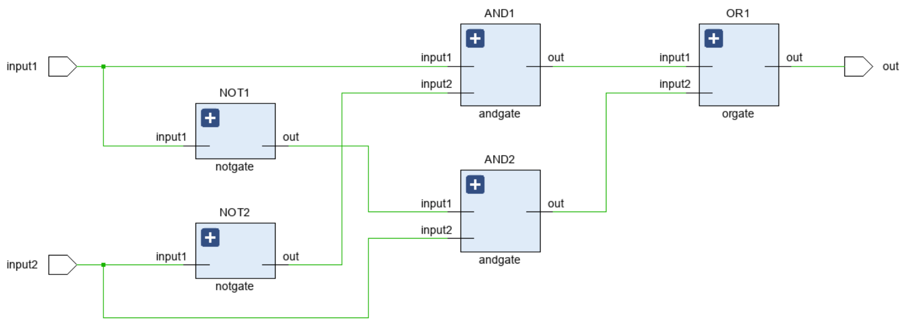

Figure 4: XOR Gate RTL Schematic

2.2 PART 2

In this part, we implemented 1-Bit Half Adder module by using AND, OR, NOT, XOR modules which we designed in the first part. The output of XOR module is the sum of inputs and the output of AND gives the output carry.

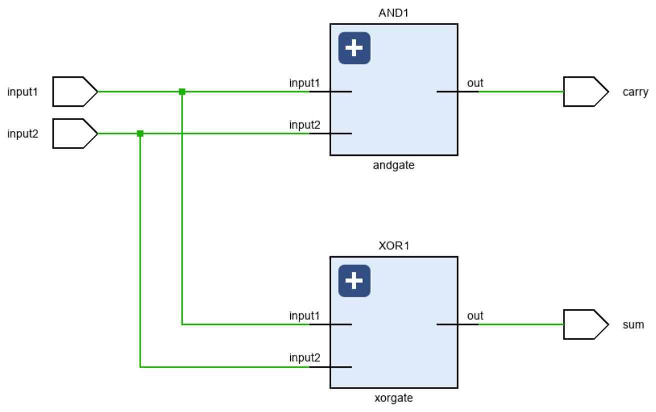

Figure 5: 1-Bit Half Adder RTL Schematic

2.3 PART 3

In this part, we implemented a 1-Bit Full Adder by using two 1-Bit Half Adders and one OR Gate. The first half-adder takes the inputs. Then, it gives the sum and first carry-output. The second half-adder takes the carry input and the first half adder's sum output. Then, it gives the output sum of full-adder and second carry-output. OR gate takes first carry-output and second carry-output Then, it gives carry-output of the full adder.

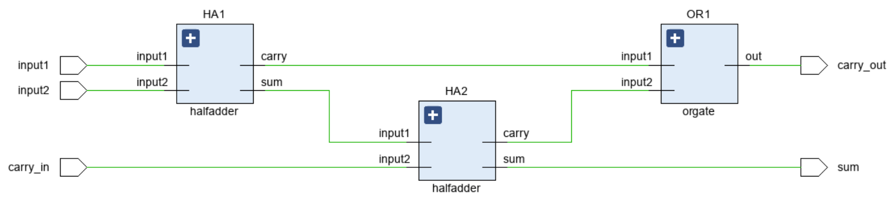

Figure 6: 1-Bit Full Adder RTL Schematic

2.4 PART 4

In this part, we implemented a 4-Bit Full Adder by using 1-Bit Full Adder modules that we designed in the third part. We used 4 parallel 1-Bit Full Adder. The first full adder's carry output becomes the carry-input of the second full adder. The second full adder's carry output becomes the carry-input of the third full adder. The third full adder's carry output becomes the carry-input of the fourth full adder. The fourth full adder's carry output is the carry output of the 4-bit full adder. All sum outputs of 1-bit full adder are digits of 4-bit sum output.

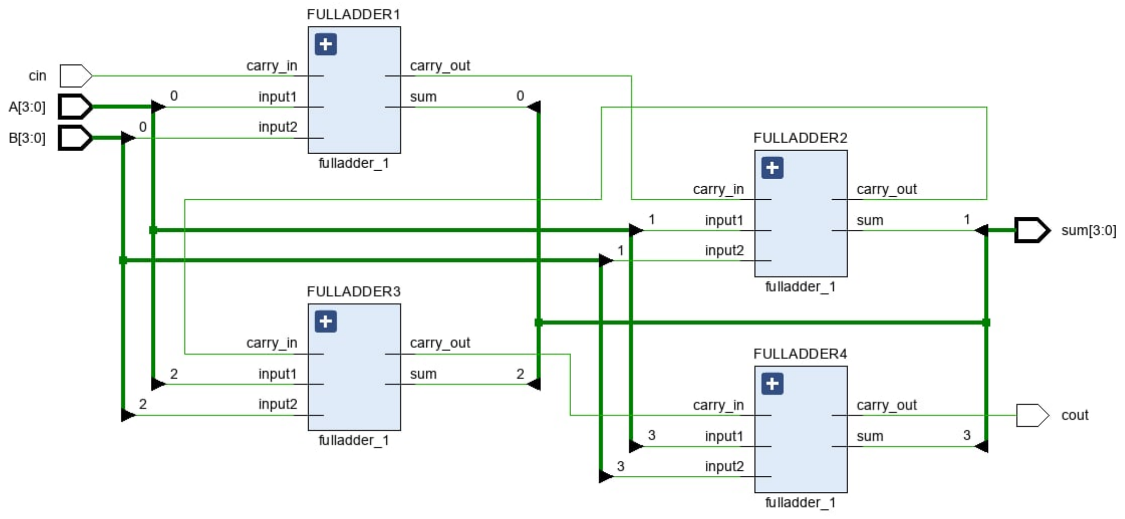

Figure 7: 4-Bit Full Adder RTL Schematic

2.5 PART 5

In this part, we implemented a 16-Bit Full Adder by using 4-Bit Full Adder modules that we designed in the fourth part. We used 4 parallel 4-Bit Full Adder. The first full adder's carry output becomes the carry-input of the second full adder. The second full adder's carry output becomes the carry-input of the third full adder. The third full adder's carry output becomes the carry-input of the fourth full adder. The fourth full adder's carry output is the carry output of the 16-bit full adder. All sum outputs of 4-bit full adder are digits of 16-bit sum output.

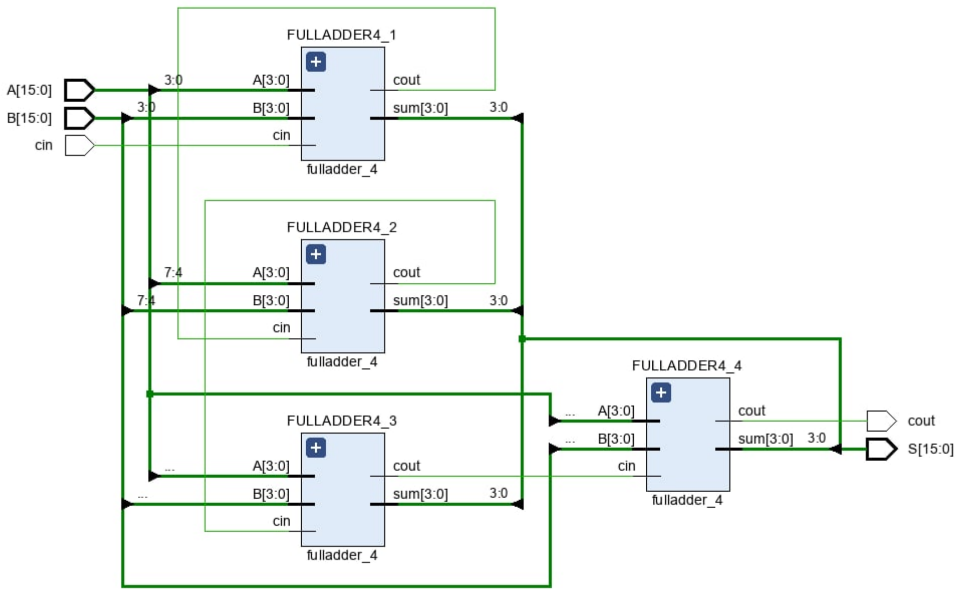

Figure 8: 16-Bit Full Adder RTL Schematic

2.6 PART 6

In this part, we implemented a 16-Bit Adder-Subtractor Circuit by using basic gate modules that we designed before. We had an s input that decides the binary numbers to be unsigned or signed. If numbers are signed, there occur overflows while adding or subtracting. If numbers are unsigned, there may be carry while adding. It shows that the result can not be represented with 16 bits in this case. If numbers are unsigned, there may be borrow while subtracting. It shows that the first number is smaller than the second number.

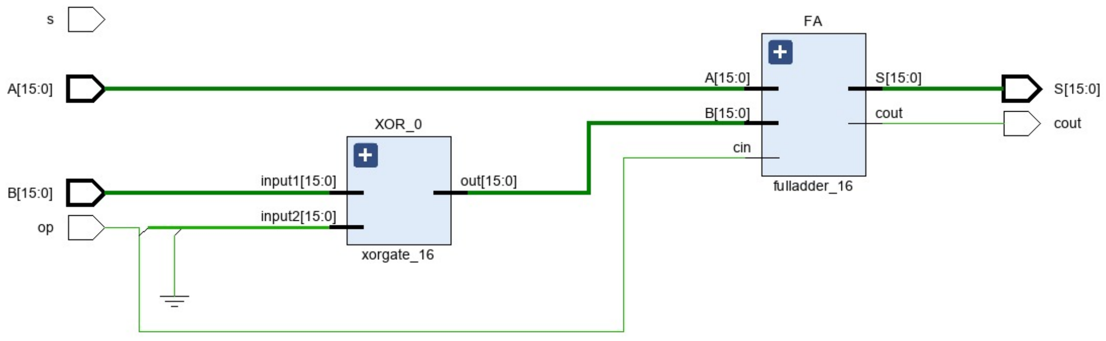

Figure 9: 16-Bit Adder-Substractor RTL Schematic

2.7 PART 7

In this part, we implemented a module to calculate 3A-2B with the 16-bit full adder-substractor that we designed in the previous part. We used 3 different 16-bit full adder-substractor: one for calculating A-B, one for calculating (A-B)+(A-B) which is 2A-2B, and the last one for (2A-2B)+A and we obtained the module for 3A-2B.

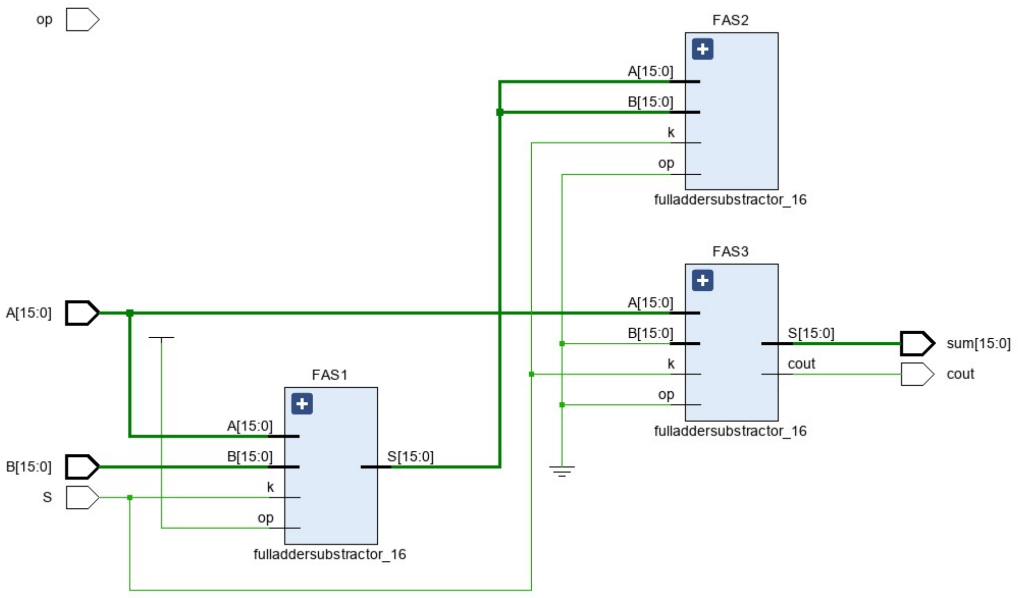

Figure 10: 16-Bit 3A-2B Circuit RTL Schematic

3 RESULTS [15 points]

3.1 PART 1

In this part, we implemented AND, OR, NOT, and XOR gates. The simulation results show that we made them correctly.

3.1.1 AND GATE

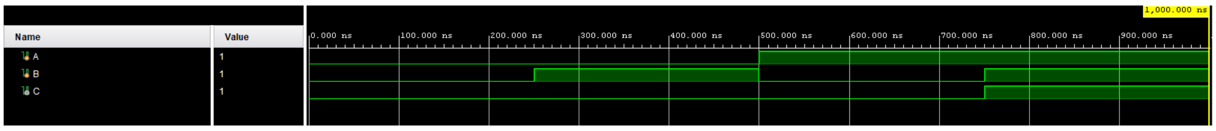

Figure 11: AND gate simulation

3.1.2 OR GATE

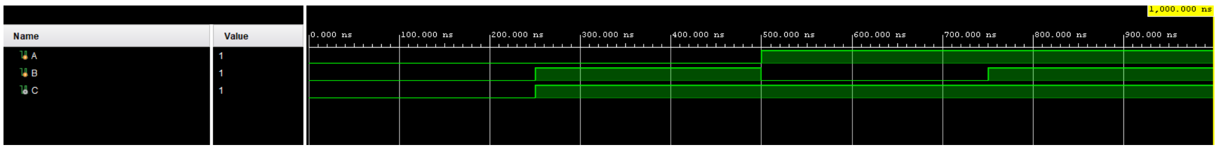

Figure 12: OR gate simulation

3.1.3 NOT GATE

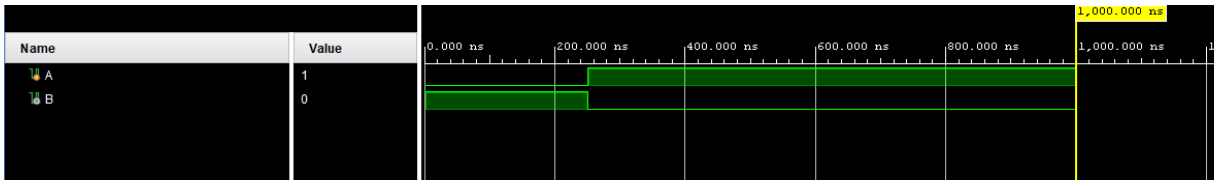

Figure 13: NOT gate simulation

3.1.4 XOR GATE

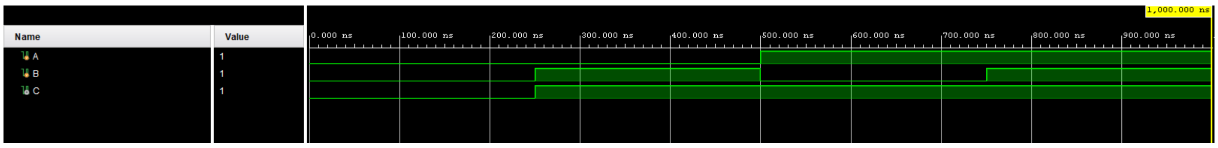

Figure 14: XOR gate simulation

3.2 PART 2

In this part, we implemented a 1-Bit Half-Adder module. We simulated it using different input combinations. The simulation result shows that we did it correctly. A and B corresponds to the inputs, C corresponds to carry bit and D corresponds to sum.

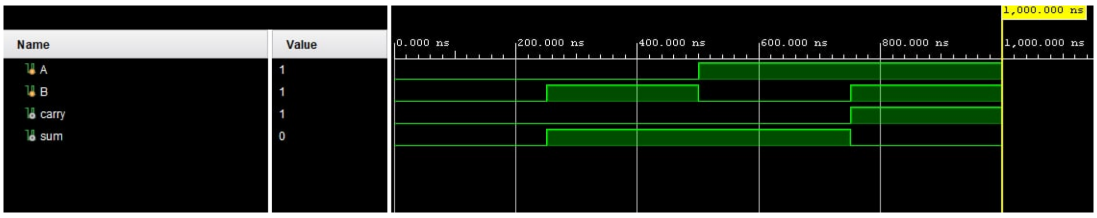

Figure 15: 1-Bit Half Adder Simulation

3.3 PART 3

In this part, we implemented a 1-Bit Full-Adder module. We simulated it using different input combinations. The simulation result shows that we did it correctly. A and B corresponds to the inputs, C corresponds to the input carry bit, D corresponds to the output carry bit and E corresponds to the sum.

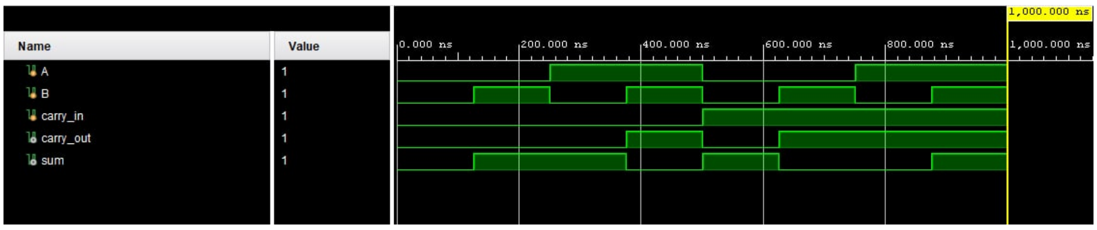

Figure 16: 1-Bit Full Adder Simulation

3.4 PART 4

In this part, we implemented a 4-Bit Full-Adder module. We simulated it with different unsigned number combinations. The simulation result shows that we did it correctly. A and B correspond to the 4-bit inputs, S corresponds to the sum.

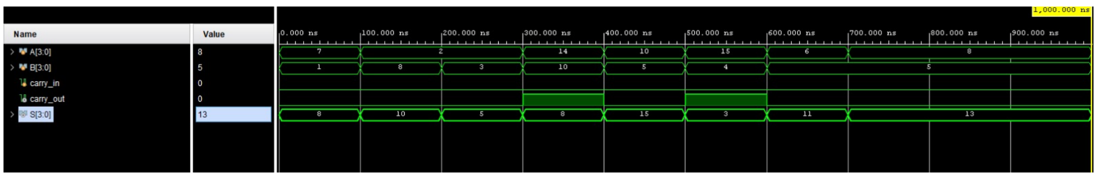

Figure 17: 4-Bit Full Adder Simulation

3.5 PART 5

In this part, we implemented a 16-Bit Full-Adder module. We simulated it with different unsigned number combinations. The simulation result shows that we did it correctly. A and B correspond to the 16-bit inputs, S corresponds to the sum.

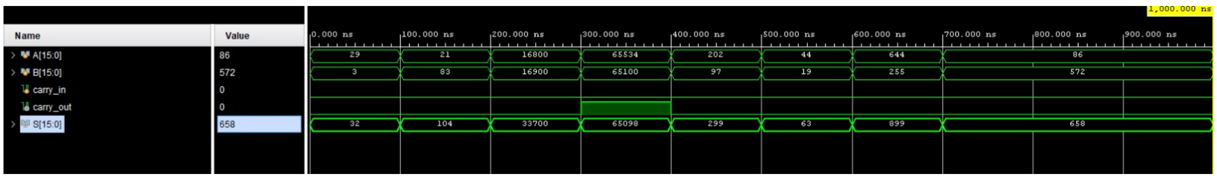

Figure 18: 16-Bit Full Adder Simulation

3.6 PART 6

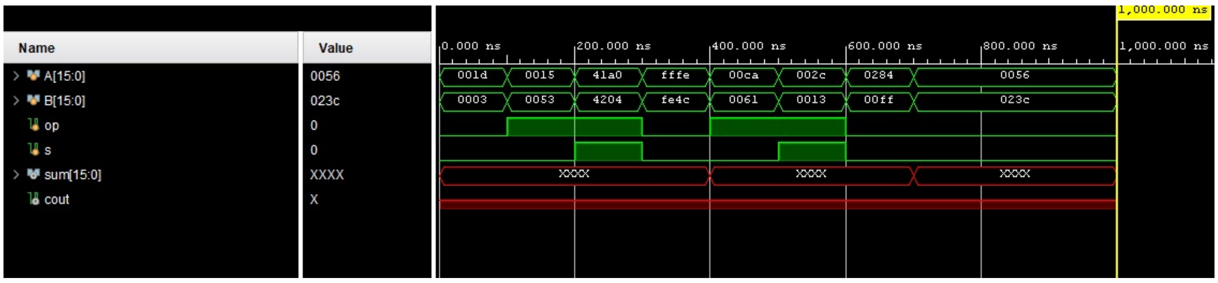

Figure 19: Full Adder-Subtractor Simulation

3.7 PART 7

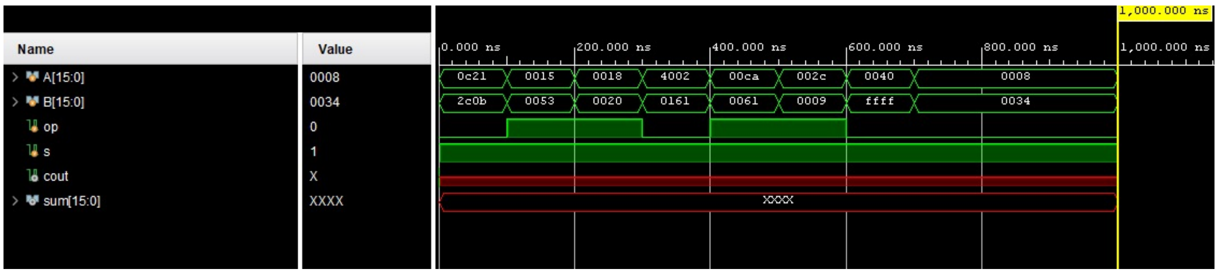

Figure 20: 3A-2B Circuit Simulation

4 DISCUSSION [25 points]

In Part 1, we firstly implemented AND, OR, NOT modules. Then using these modules we also implemented XOR module. XOR gate is a digital logic gate that gives a true output when the number of true inputs is odd. The algebraic expression of an XOR gate is $AB' + A'B$ or simply AB. We took the complements of our inputs, then we multiplied them with the other input: $a'b$ and ab'. Then we added them and finished implementing our XOR gate with the final output: $ab' + a'b$.

In Part 2, we implemented a 1-Bit Half Adder using the gates we designed in the first part.The 1-bit half adder adds two 1-bit numbers without carry input. Carry output is expressed as AB and the sum output is expressed as AB. We had two outputs: sum and carry. We took ab as the carry using the AND gate and $ab' + a'b$ as the sum using the XOR gate. An adder is a digital circuit that performs addition of numbers.

In Part 3, we implemented a 1-Bit Full Adder using the half adder and OR gate we designed before. A 1-bit full adder adds two 1-bit numbers with a carry input(cin). Carry output is expressed as $Acin + Bcin + AB$ and the sum output is expressed as $ABcin$. We used 2 1-Half Full Adders in our code and their names are HA1 and HA2. We obtained two outputs from the first half adder HA1: ab as the carry and $ab' + a'b$ as the sum. In the second half adder HA2, we used the sum that we found in the first half adder as one input, the other input was 0 since there was no carry-in. We again obtained two outputs: 0 as the carry and $ab' + a'b$ as sum. In the OR gate, we multiplied the sum from the first half adder and the carry from the second half adder.

In Part 4, we implemented a 4-Bit Full Adder using the 1-Bit Full Adder that we designed before. A 4-bit full adder adds two 4-bit numbers with a carry input. We used 4 1-Bit Full Adders in our code and their names are FULLADDER1, FULLADDER2, FULLADDER3, FULLADDER4. We obtained two outputs from the first 1-Bit Full Adder FULLADDER1. One of the outputs is araKablo1 which is the carry-out of FULLADDER1 and the carry-in of FULLADDER2. The other output is the first digit of the 4-bit sum. In the second 1-Bit Full Adder FULLADDER2, we took the second digits of 4-bit inputs. We got 2 outputs here. One of them is araKablo2 which is the carry-out of FULLADDER2 and the carry-in of FULLADDER3. The other output is the second digit of the 4-bit sum. The same process is valid for FULLADDER3 and FULLADDER4. The only difference is FULLADDER4's carry-out is the carry of the 4-bit adding operation.

In Part 5, we implemented a 16-Bit Full Adder using the 4-Bit Full Adder that we designed before. 16-bit full adder adds two 16-bit numbers with a carry input. We used 4 4-Bit Full Adders in our code and their names are FULLADDER4_1, FULLADDER4_2, FULLADDER4_3,FULLADDER4_4. We obtained two outputs from the first 4-Bit Full

Adder FULLADDER4_1. One of the outputs is araKablo1 which is the carry-out of FULLADDER41 and the carry-in of FULLADDER4_2. The other output is the first 4 digits of the 16-bit sum. In the second 4-Bit Full Adder FULLADDER4_2, we took the second 4 digits of 16-bit inputs. We got 2 outputs here. One of them is araKablo2 which is the carry-out of FULLADDER4_2 and the carry-in of FULLADDER4_3. The other output is the second 4 digits of the 16-bit sum. The same process is valid for FULLADDER4_3 and FULLADDER4_4. The only difference is FULLADDER4_4's carry-out is the carry of the 16-bit adding operation.

In Part 6, 16-bit full adder-subtractor adds or subtracts two 16-bit numbers with a carry input. An extra input is used to control the function of the circuit. If the input is 0, it's an adder. If the input is 1, it's a subtractor. While the adder simply works as $A + B$, the subtractor uses $A(B' + 1)$. We can design the adder-subtractor circuit using XOR gates.

In Part 7, we implemented a module to calculate 3A-2B with the 16-bit full adder-subtractor that we designed in the part 6. We used 3 different 16-bit full adder-subtractors named FAS1, FAS2 and FAS3. In FAS1, we calculated A-B: our inputs were A and B and our operator was 1 because of the subtraction. Our output was named sum1, and we used it in FAS2. In FAS2, both of our inputs were sum1 to calculate (A-B)+(A-B), because of the addition our operator was 0. Our output was named sum2, and we used it in FAS3. In FAS3, our inputs were sum2 and A to calculate (2A-2B)+A, because of the addition our operator was again 0. The output of FAS3 is our final output, 3A-2B.

5 CONCLUSION [10 points]

In this experience, we used half adders and full adders, strengthening our knowledge about them. We had some difficulties while coding the full adders with more than 1-bit. First, we gave every bit as a new input; then, we learned that how to take more than 1-bits in 1 input. We also wrote our modules with generating blocks before realizing that we're not allowed to use them in this experience. In conclusion, we used AND, OR, NOT, XOR gates to create adders and validated them using the simulation tool. In part 6, we could implement adder-subtractor circuit but we could not do overflow and borrow parts. Thus, Part 6 and Part 7 simulations do not work correctly.