

ENCS2110

DIGITAL ELECTRONICS AND COMPUTER ORGANIZATION LABORATORY

Experiment 11: Arithmetic Elements

Name: Sara Totah

Student ID: 1181779

Instructor: Khader Mohammed

T.A: Enas Jawabreh

Date: 01/04/2021

# Data, Results and Discussion

## Arithmetic Logic Unit (ALU) Circuit

1. Set Cn to “0” and ignore the previous carry.
2. A3A2A1A0=0000 and B3B2B1B0=1111



The output: F3F2F1F0 = 1111

Cn+4 = 0

1. A3A2A1A0=1001 and B3B2B1B0=0100?



The output: F3F2F1F0 = 0110

Cn+4 = 1

As it can be seen, it happens bitwise (bit by bit), in these example, since A = 0000, and B = 1111, and the selection S = 0000, and M =1 then the output F should be equal to the complement of A F = 1111, also it can be seen that the same things happened in the second example.

1. Set Cn to “1” and add the previous carry.
2. A3A2A1A0=0000 and B3B2B1B0=1111



The output: F3F2F1F0 = 1111

Cn+4 = 1

1. A3A2A1A0 = 1001 and B3B2B1B0 = 0100



The output: F3F2F1F0 = 0010

Cn+4 = 1

It can be seen here that Cn value doesn’t affect the output when M is high (M = 1), the output is still equal to the complement of A.

1. Set Cn to “0”. When S3S2S1S0=0001 perform the subtraction.
2. A3A2A1A0 = 0000 and B3B2B1B0 = 1111



The output: F3F2F1F0 = 0000

Cn+4 = 0

1. A3A2A1A0=1001 and B3B2B1B0=0100



The output: F3F2F1F0 = 0010

Cn+4 = 1

When setting the selection S to 0001, the ALU will implement a subtraction between B and A.

1001 – 0100 = 00101

Assuming A = 0011 and B = 0001

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SELECTION | | | | M = 1  Logic Functions | M = 0  ARITHMATIC OPERATIONS | |
| Cn = 0 | Cn = 1 |
| S3 | S2 | S1 | S0 | F3F2F1F0 CN+4 | F3F2F1F0 CN+4 | F3F2F1F0 CN+4 |
| 0 | 0 | 0 | 0 | 1100 1 | 0100 1 | 0011 1 |
| 0 | 0 | 0 | 1 | 1100 1 | 0100 1 | 0011 1 |
| 0 | 0 | 1 | 0 | 0000 0 | 0000 0 | 1111 1 |
| 0 | 0 | 1 | 1 | 0000 0 | 0000 0 | 1111 1 |
| 0 | 1 | 0 | 0 | 1110 1 | 0110 1 | 0101 1 |
| 0 | 1 | 0 | 1 | 1110 1 | 0110 1 | 0101 1 |
| 0 | 1 | 1 | 0 | 0010 0 | 0010 0 | 0001 0 |
| 0 | 1 | 1 | 1 | 0010 0 | 0010 0 | 0001 0 |
| 1 | 0 | 0 | 0 | 0101 1 | 0101 1 | 0100 1 |
| 1 | 0 | 0 | 1 | 0101 1 | 0101 1 | 0100 1 |
| 1 | 0 | 1 | 0 | 0001 0 | 0001 0 | 0000 0 |
| 1 | 0 | 1 | 1 | 0001 0 | 0001 0 | 0000 0 |
| 1 | 1 | 0 | 0 | 0111 1 | 0111 1 | 0110 1 |
| 1 | 1 | 0 | 1 | 0111 1 | 0111 1 | 0110 1 |
| 1 | 1 | 1 | 0 | 0011 0 | 0011 0 | 0010 0 |
| 1 | 1 | 1 | 1 | 0011 0 | 0011 0 | 0010 0 |

## Some Random Cases for the ALU

1. F = A:



1. F = ZERO



1. F = (A + B’) PLUS 1



## Bit Parity Generator Circuits

A parity bit is the leftmost bit of a binary number, which is usually added to indicate whether the number of ones in it is even or odd, first in an odd parity generator, when the bit parity is added, the number of ones should become an odd number.

Second, in the even bit parity generator, a one is added it a binary number with odd number of 1 to make the total number of ones = even number, but if the number of ones was even in the binary number, the bit parity will be 0.

### Even Bit Parity Generator Circuit



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | | | Output |
| E | D | C | B | A | F6 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

In the even parity generator, if the number of ones in the input binary number was an even number then the output should be zero, (there is no need to add another 1 to make it even since it’s already even), but if the number of ones in the input was an odd number then the output should be 1, to make the total amount of ones even number.

In this Example, the input binary number was 11100 containing 3 ones (odd number), so the output was 1 at the even parity generator.

### Bit Parity Generator IC



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I | H | G | F | E | D | C | B | A | Y0  (even) | Y1  (odd) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

**Question:** how to use the chip in the previous figure with inputs more than 9 + less than 9?

More than 9 inputs:



Less than 9 inputs:

