Pragya Singhal, 2019112001

IPA Assignment 1

32-Bit AND

Truth Table -

Input a	Input b	Output y
0	0	0
1	0	0
0	1	0
1	1	1

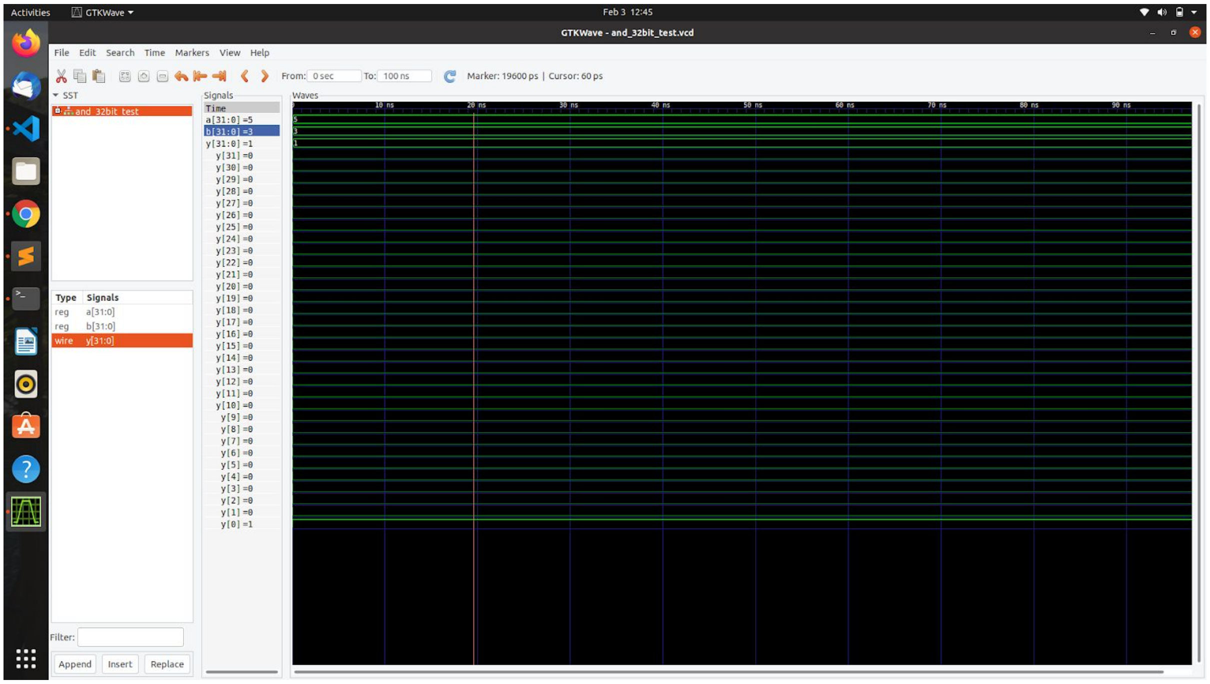
Logic -

module "and_2bit(a,b,y)" - We first implement the 2-Bit AND operator where the output is 1 only when both the inputs are 1 and 0 in all the other cases. Here the inputs are 1 bit each and the output is a register.

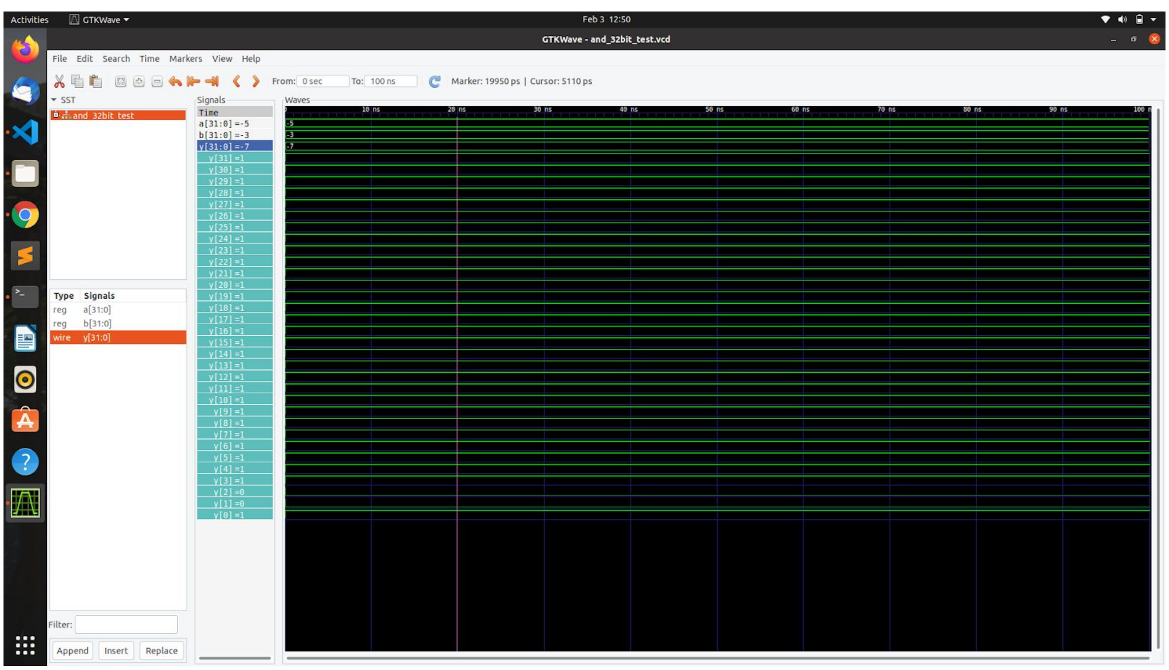
module "and_32bit(a,b,y)" - We then implement the 32-Bit AND operator by calling the 2-Bit and module 32 times for each bit. Here the input and output is an array from 0 to 31 index.

Outputs -

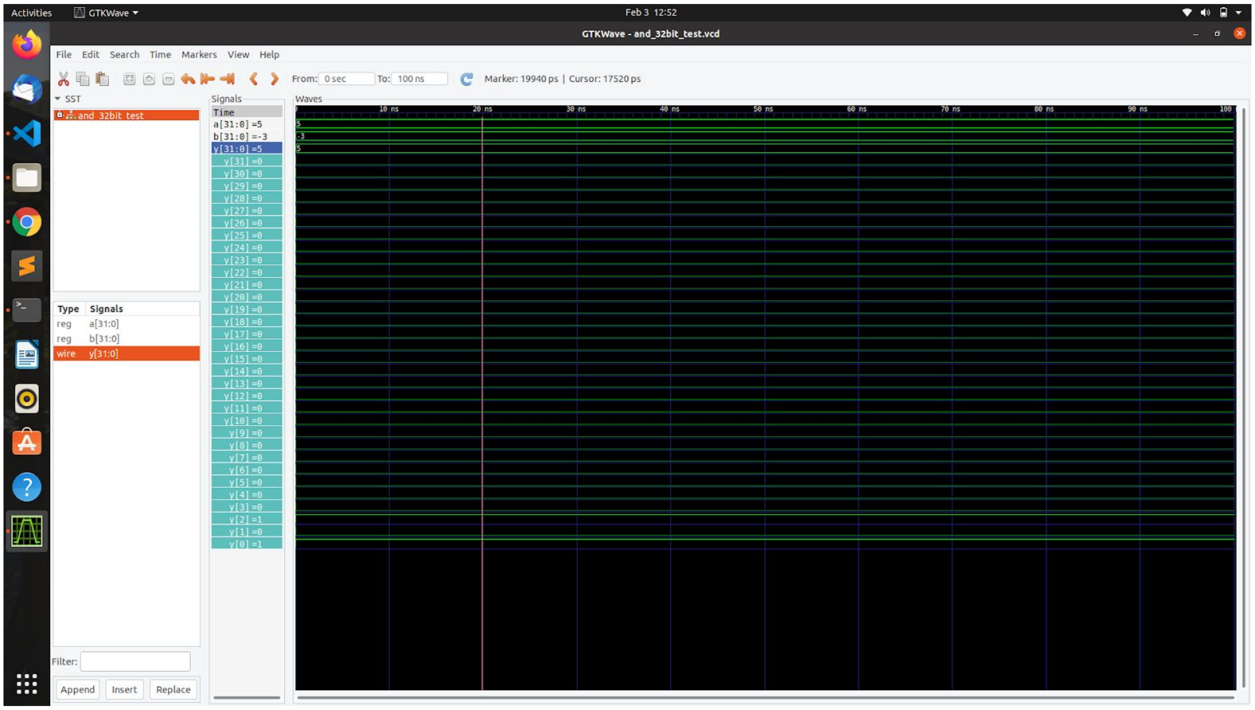
1. For two positive numbers say 5 and 3 : 5 & 3 = 1



2. For two negative numbers say -5 and -3 : -5 & -3 = -7



3. For one positive and one negative number say 5 and -3 : 5 & -3 = 5



32-Bit XOR

Truth Table -

Input a	Input b	Output y
0	0	0
1	0	1
0	1	1
1	1	0

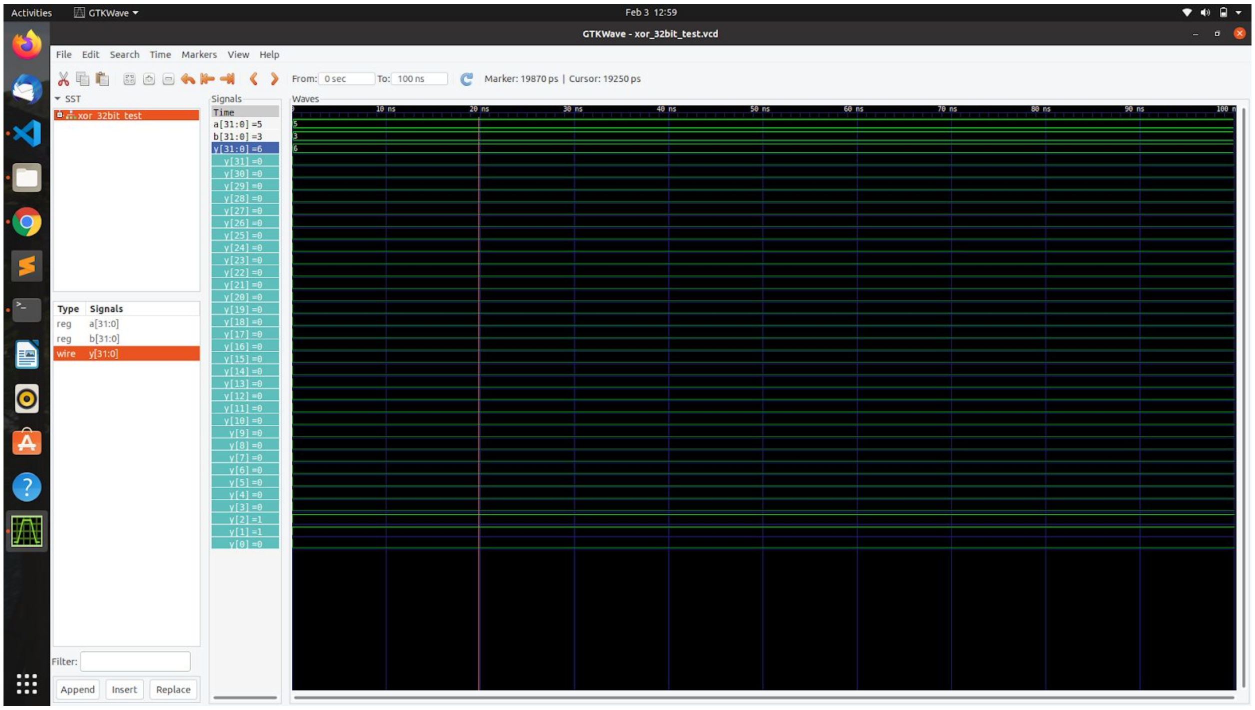
Logic -

module "xor_2bit(a,b,y)" - We first implement the 2-Bit XOR operator where the output is 1 when both the inputs are not equal and 0 when they are equal. Here the inputs are 1 bit each and the output is a register.

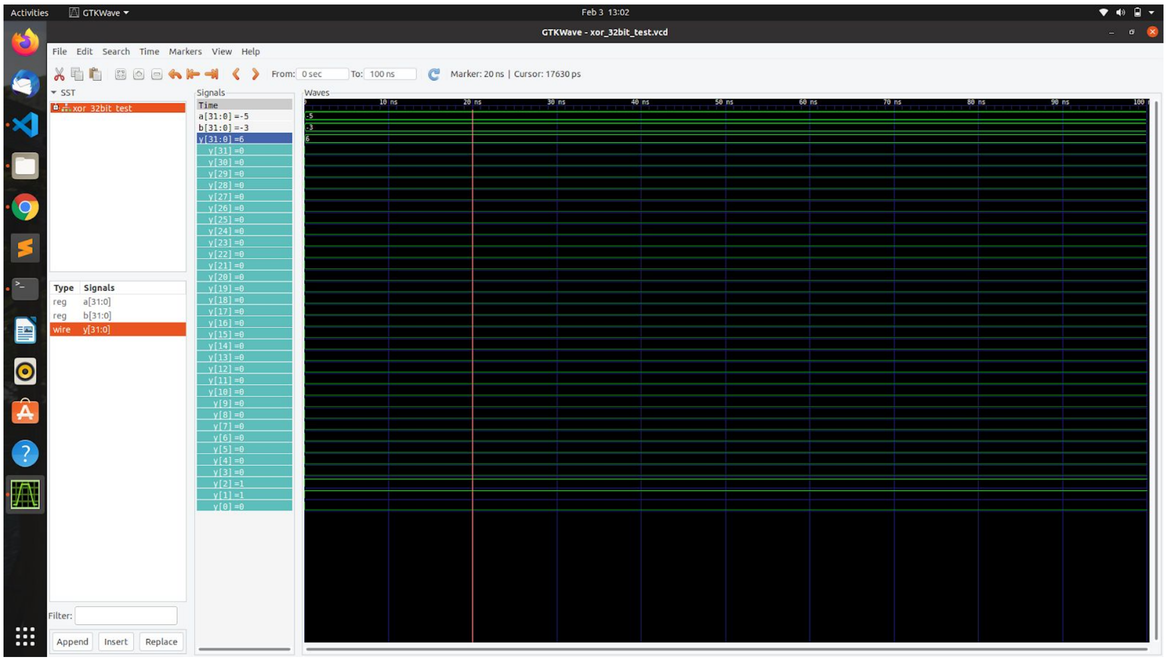
module "xor_32bit(a,b,y)" - We then implement the 32-Bit XOR operator by calling the 2-Bit xor module 32 times for each bit. Here the input and output is an array from 0 to 31 index.

Outputs -

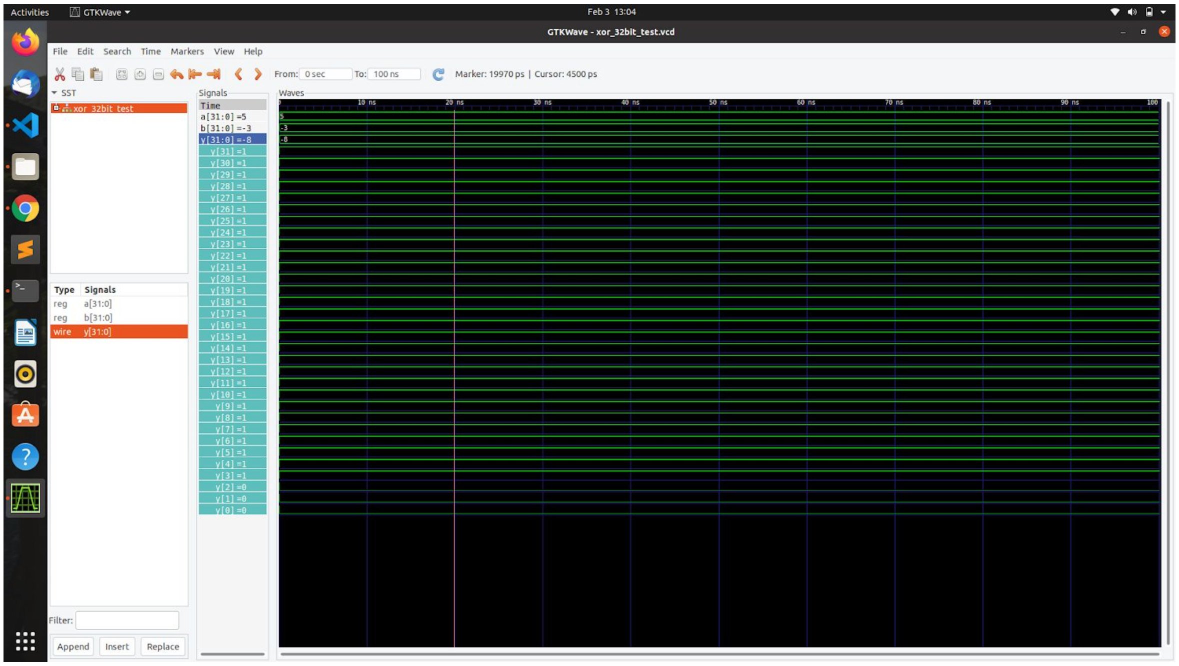
1. For two positive numbers say 5 and 3 : 5 ^ 3 = 6



2. For two negative numbers say -5 and -3 : -5 ^ -3 = 6



3. For one positive and one negative number say 5 and -3 : 5 ^ -3 = -8



32-Bit ADDER

Logic -

module "adder_2bit(a,b,c1,y,c2)" - We have 3 inputs : 'a' and 'b' bits from the main input itself and 'c1' is the carry bit from the addition of previous bits. There are 2 outputs : 'y' which is the main input bit and 'c2' is the carry bit.

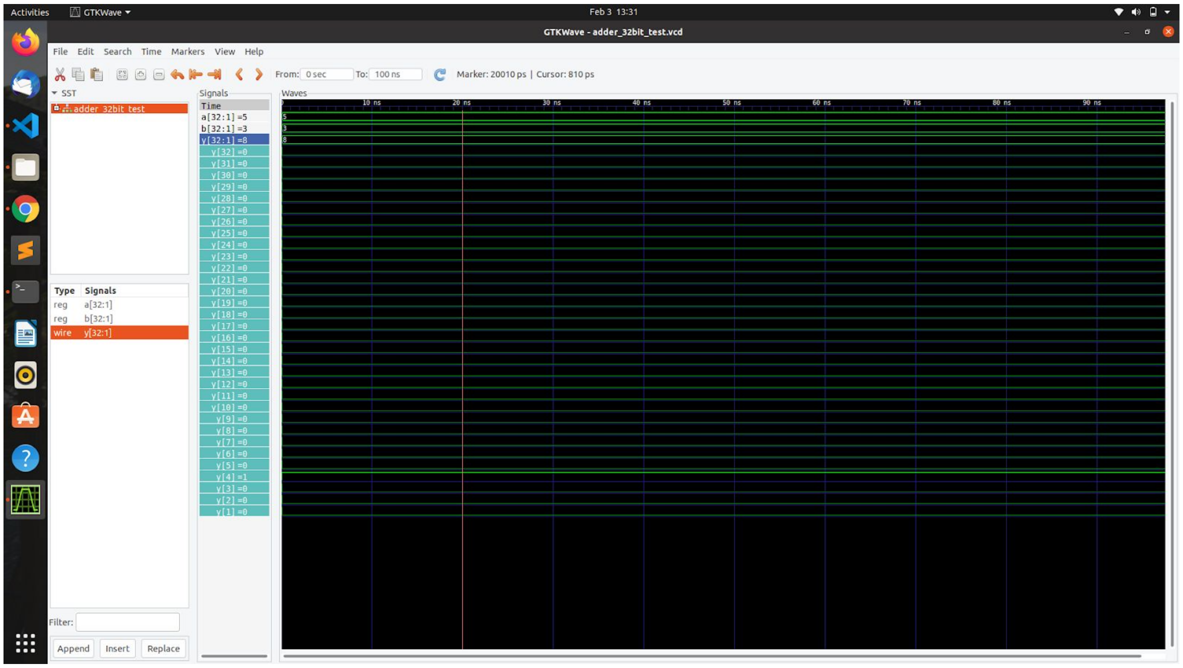
The if else conditions are as follows -

1. a = b = c1 = 1 gives us 3 (11) as the ans and hence y = 1 and c2 = 1
2. a = b = 1 and c1 = 0 gives as 2 (10) as the ans, hence y = 0 and c2 = 1
3. a = b = 0 and c1 = 1 gives us 1 (01), hence y = 1 and c2 = 0
4. a = b = 0 and c1 = 0 gives us 1 (01), hence y = 0 and c2 = 0
5. a and b = (0,1) or (1,0) and c1 = 1 gives us 2 (10), hence y = 0 and c2 = 1
6. a and b = (0,1) or (1,0) and c1 = 0 gives us 1 (01), hence y = 1 and c2 = 0

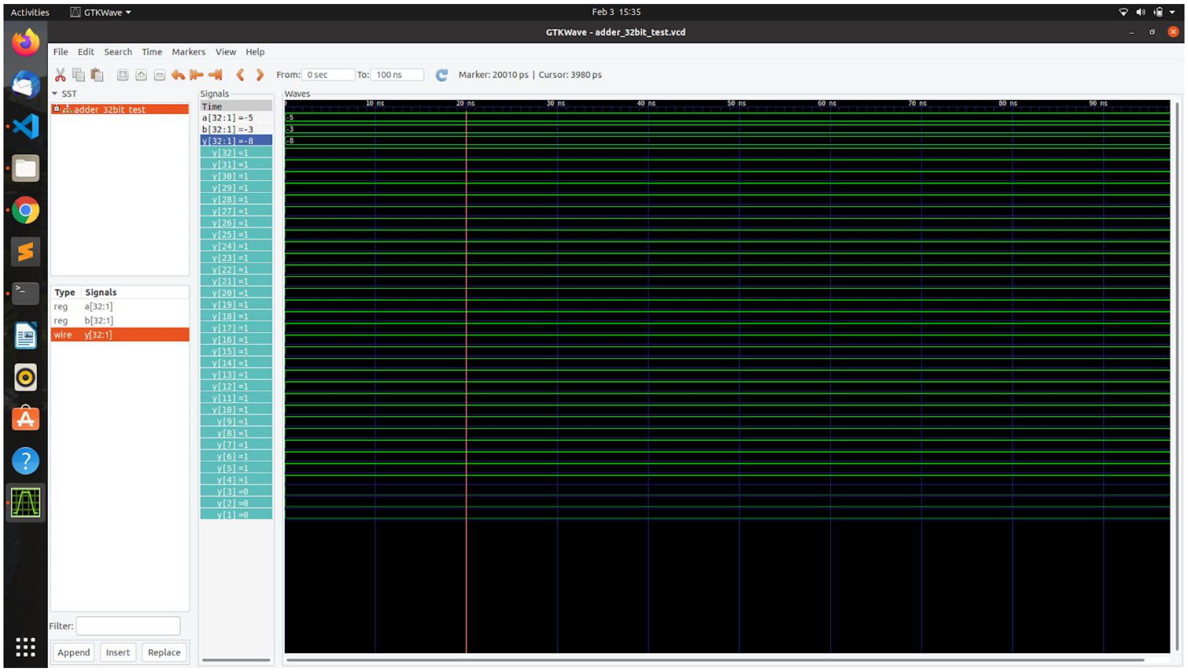
module "adder_32bit(a,b,y)" - We then implement the 32-Bit adder operator by calling the 2-Bit adder module 32 times for each bit. Here the input and output is an array from 0 to 31 index. We initialise the 0 index carry bit as 0.

Outputs -

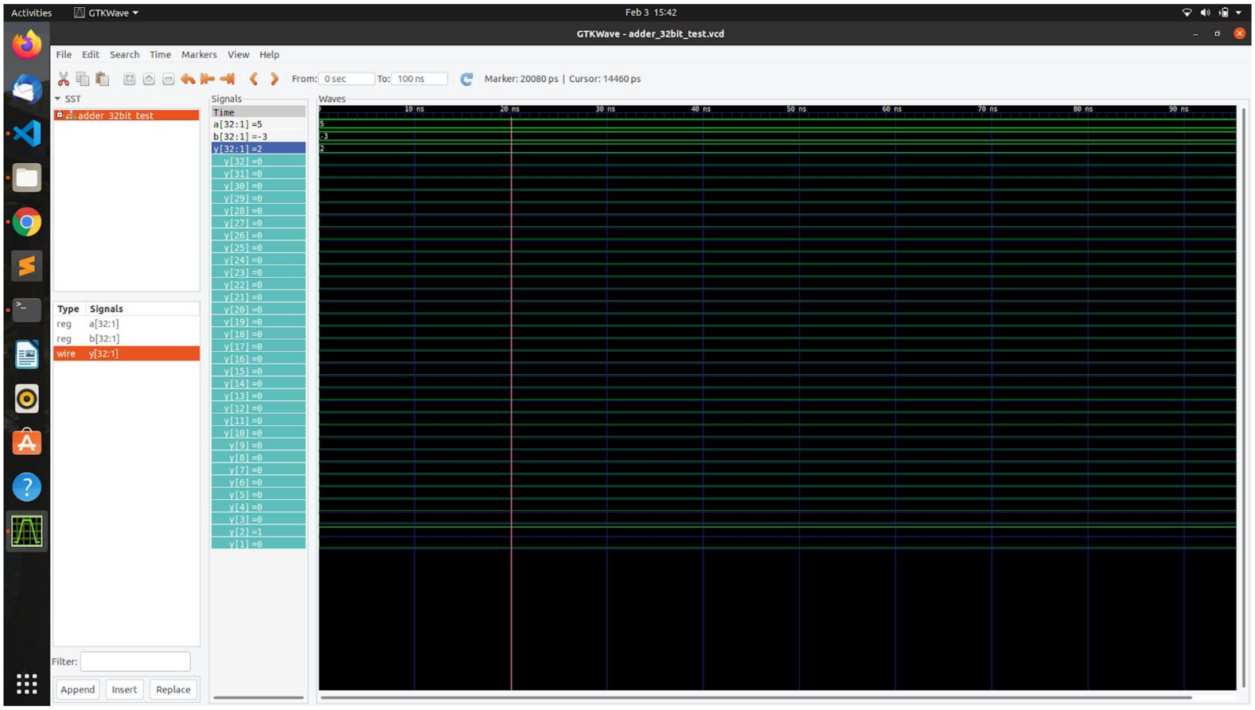
1. For two positive numbers say 5 and 3 : 5 + 3 = 8



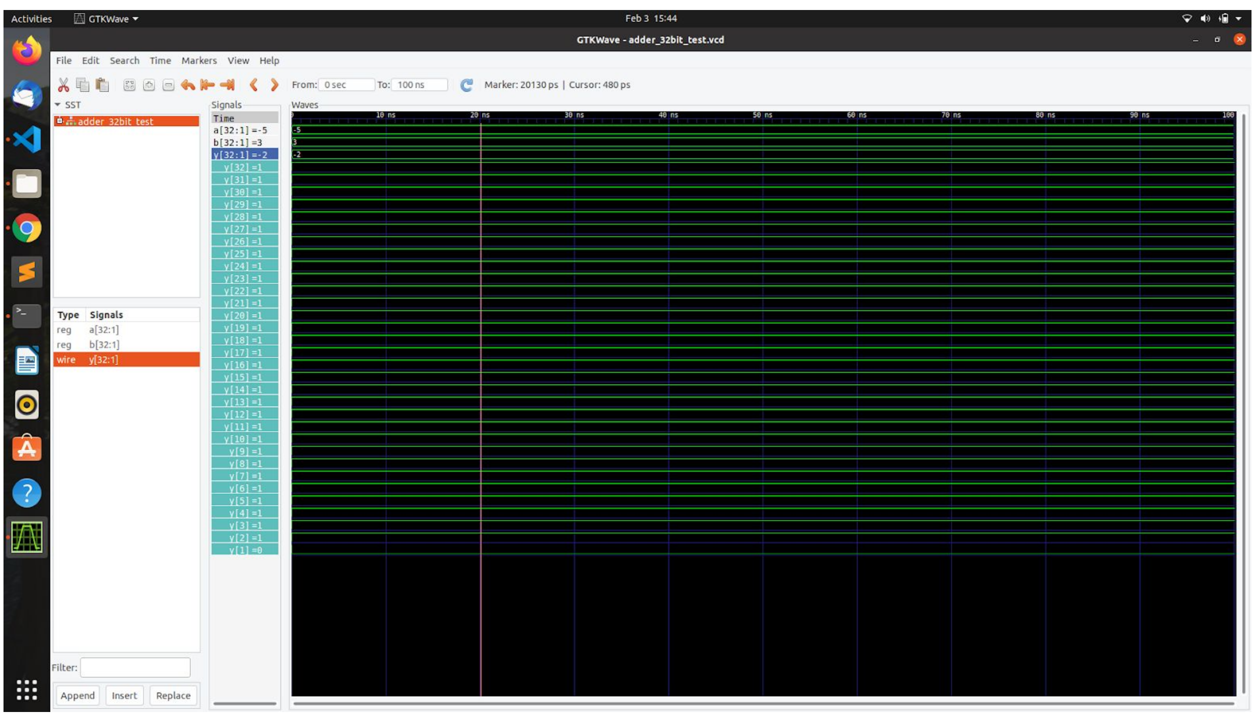
2. For two negative numbers say -5 and -3 : (-5) + (-3) = -8



3. For a positive and a negative number say 5 and -3 : 5 + (-3) = 2



4. For a positive and a negative number say -5 and 3 : -5 + 3 = -2



32-Bit SUBTRACTOR

Logic -

module "not_2bit(p,d)" - This module calculates the not of a bit i.e. if the input 'p' = 1 then output 'd' = 0 and vice versa.

module "subtractor_32bit(a,b,y)" - We then implement the 32-Bit subtractor. We first calculate the 2's complement of the input 'b' by calling the NOT operation for each bit (32 bits) and then adding a '1' using the 32-bit adder implemented above.

Next we call the 32-bit adder for the input 'a' and the 2's complement of b which is actually implementing :

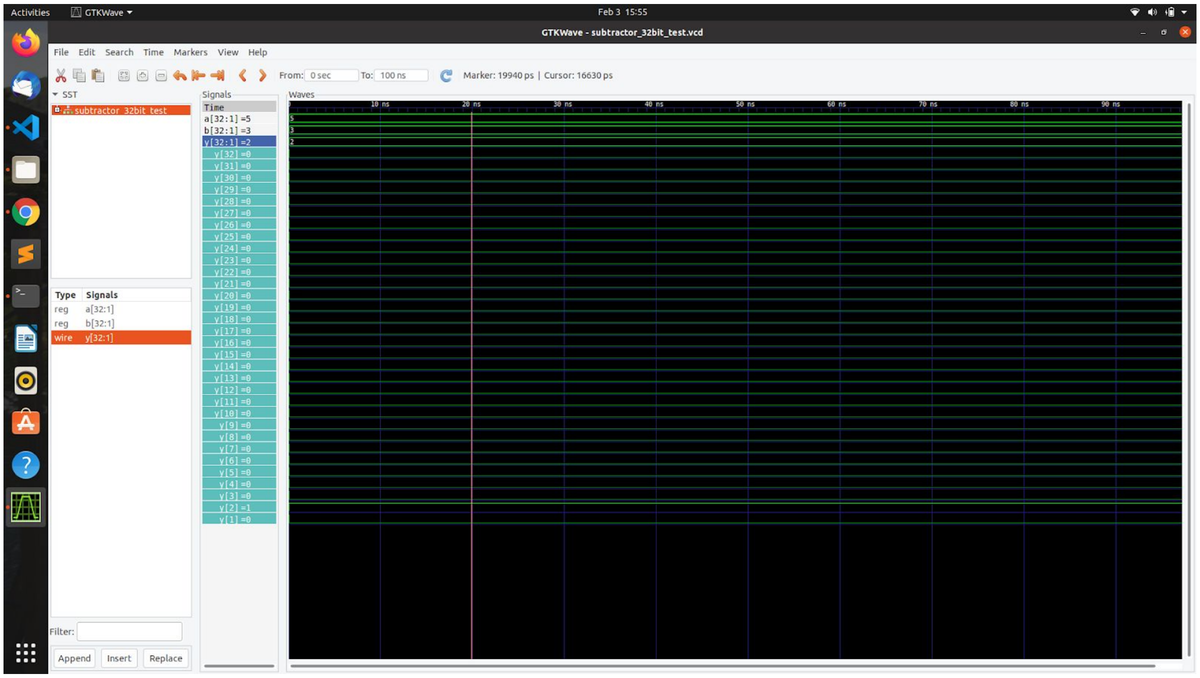
(-b) = 2's complement of (b)

Subtractor = a + (-b)

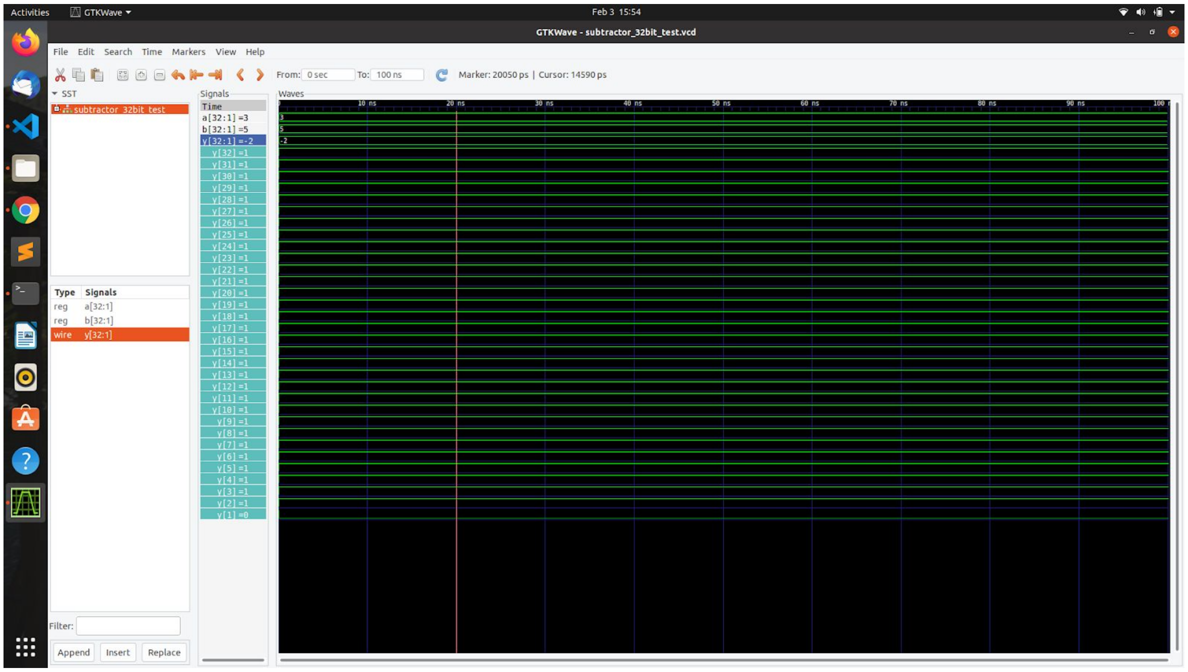
Here the input and output is an array from 0 to 31 index. We initialise the 'e' as number 1 in 32-bits binary form to calculate the 2's complement.

Outputs -

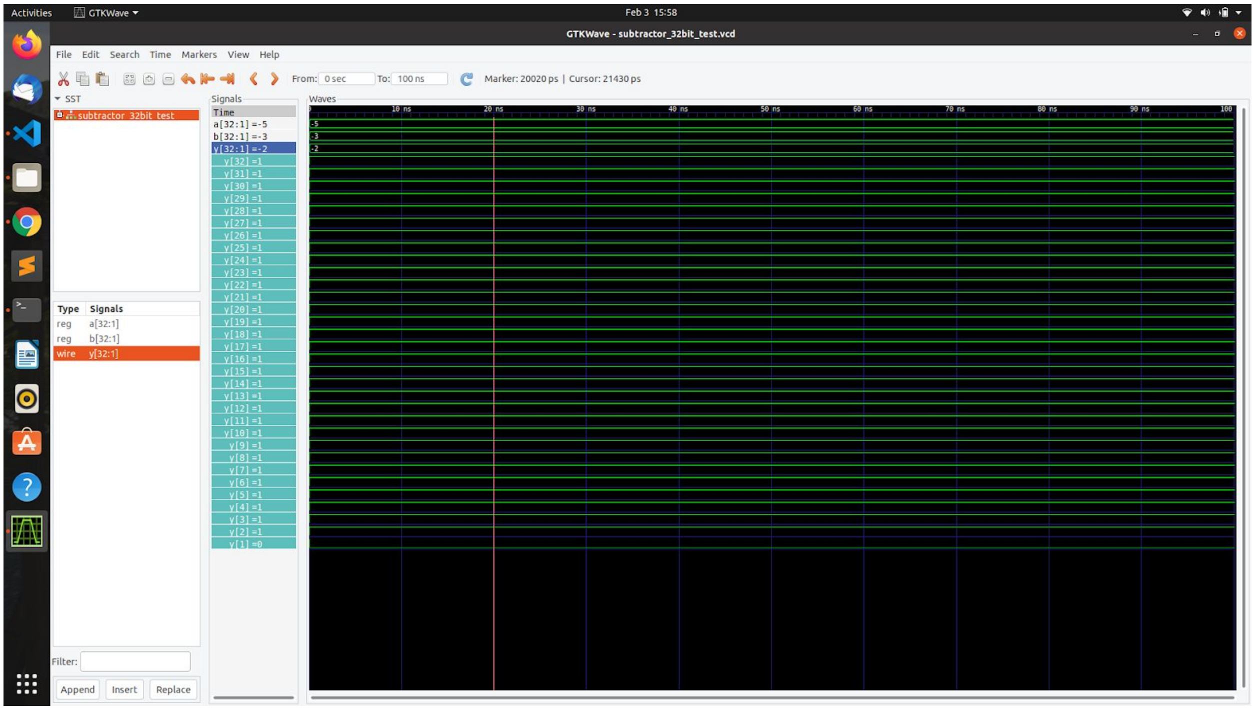
1. For two positive numbers say 5 and 3 : 5 - 3 = 2



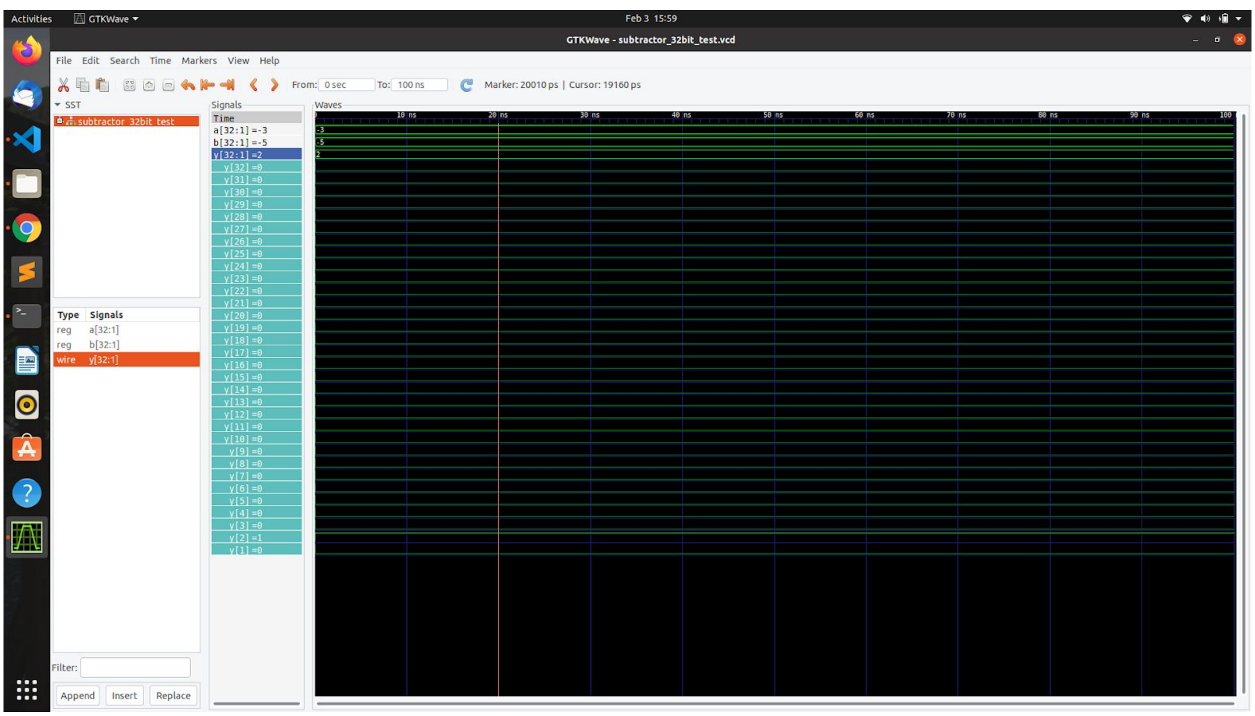
2. For two positive numbers say 3 and 5 : 3 - 5 = -2



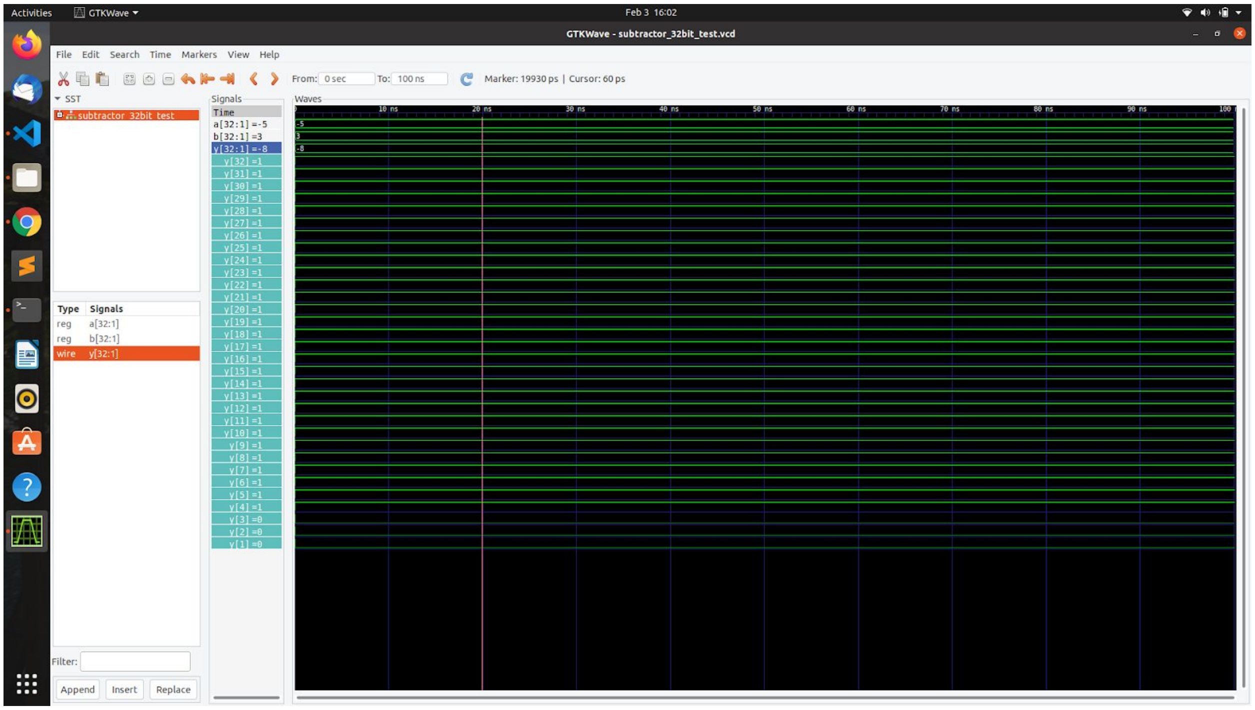
3. For two negative numbers say -5 and -3 : (-5) - (-3) = -2



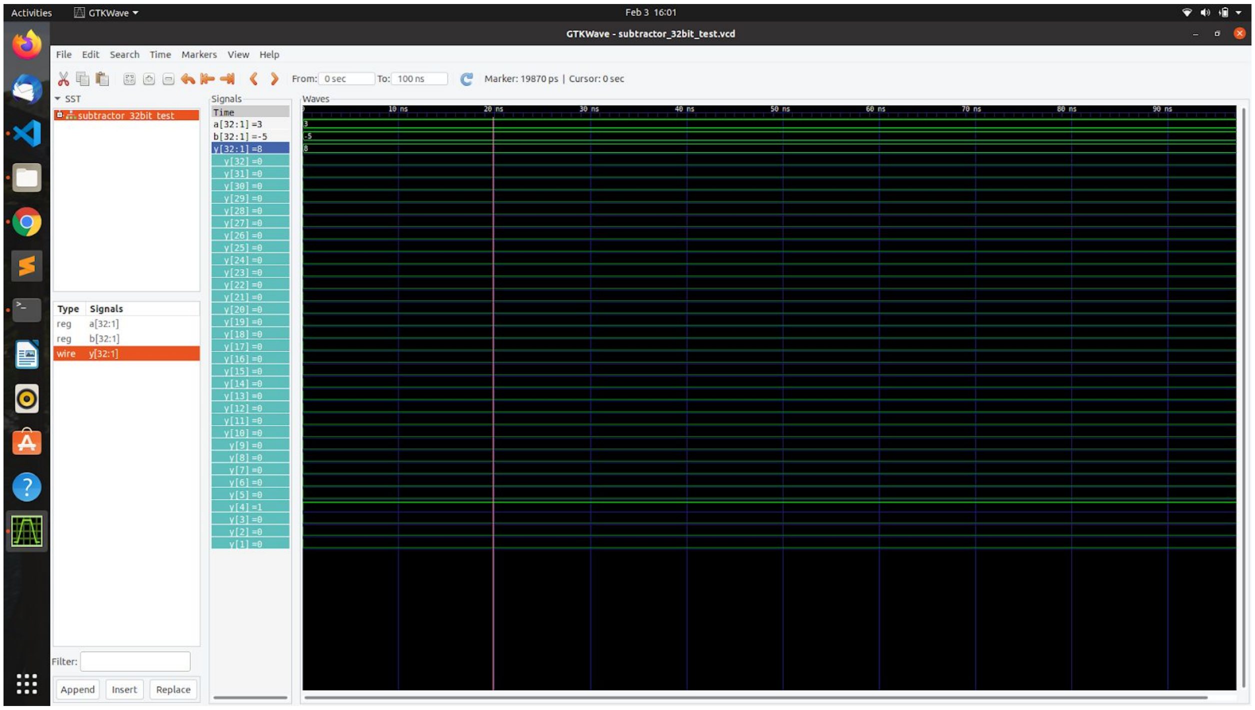
4. For two negative numbers say -3 and -5 : (-3) - (-5) = 2



5. For a negative number and a positive say -5 and 3 : (-5) - (3) = -8



6. For a negative number and a positive say 5 and -3 : (5) - (-3) = 8



32-Bit ALU

Logic -

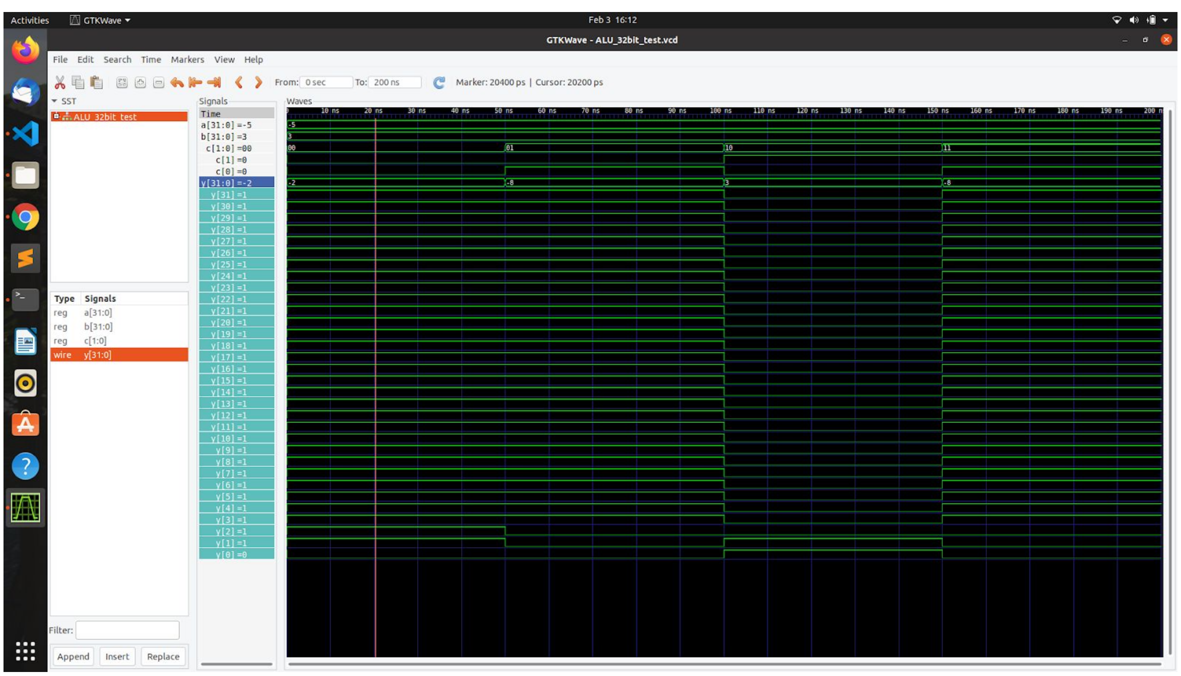
module “ALU_32bit(c,a,b,y)” - This module has 3 inputs where ‘a’ and ‘b’ are 32-bits each and ‘c’ is a 2 bit control input.

1. C = 00 - 32 bit ADDER is implemented
2. C = 01 - 32 bit SUBTRACTOR is implemented
3. C = 10 - 32 bit AND is implemented
4. C = 11 - 32 bit XOR is implemented

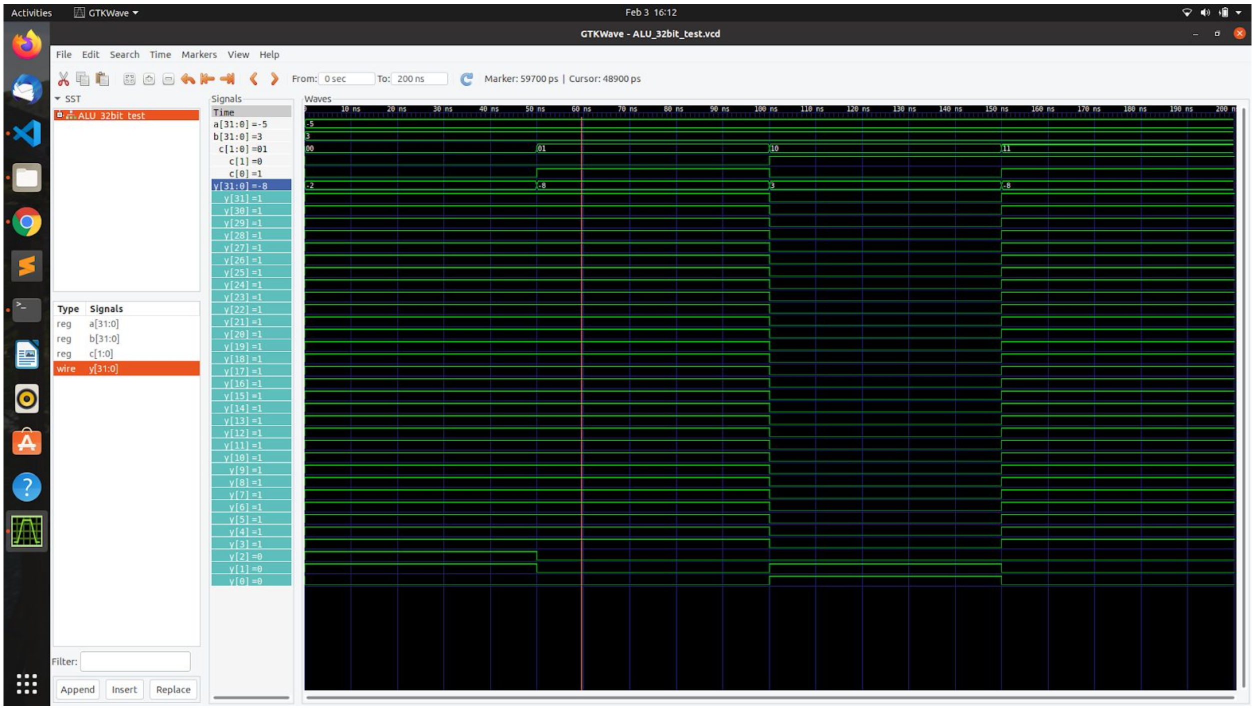
I have implemented all the 4 controls in the same code with a delay of #50 each.

Outputs -

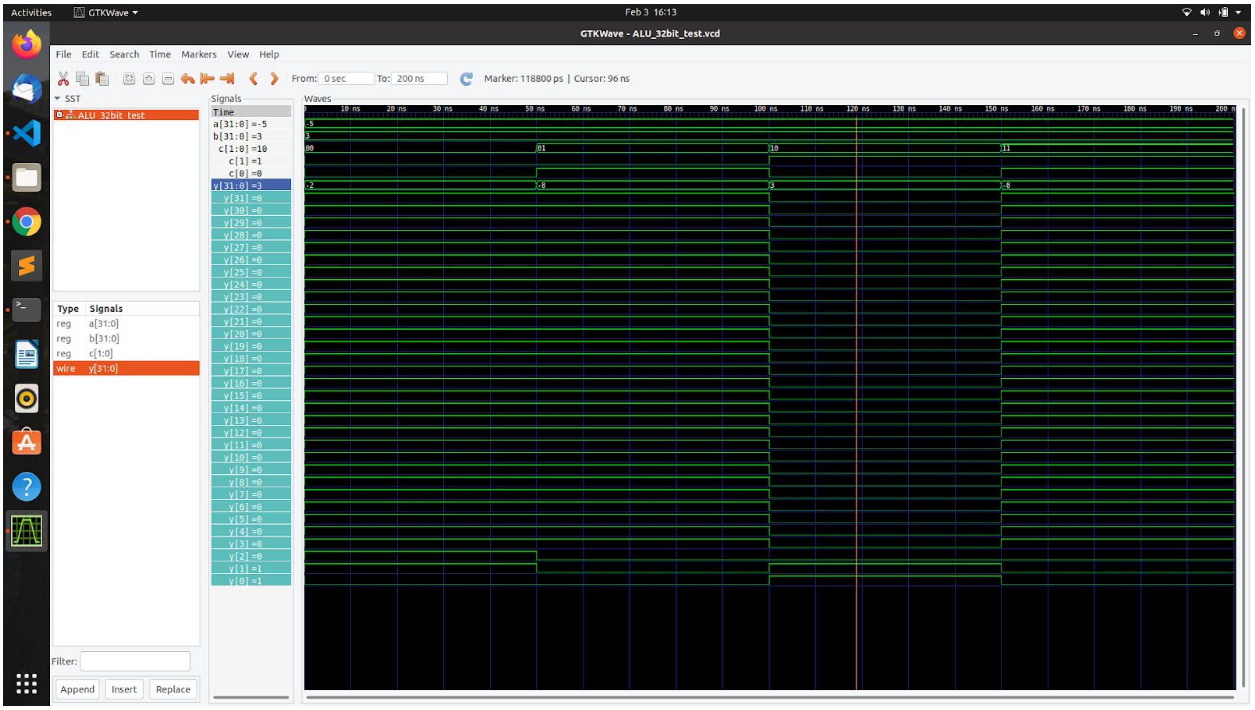
1. Control 1 for adder - (-5) + 3 = -2



2. Control 2 for subtractor - (-5) - 3 = -8



3. Control 3 for and - (-5) & 3 = 3



4. Control 4 for xor - (-5) ^ 3 = -8