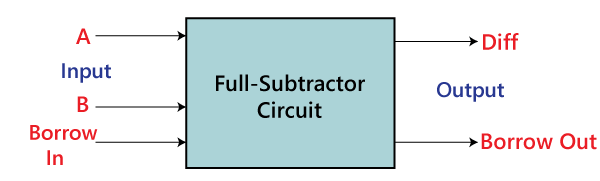
# Full Subtractor

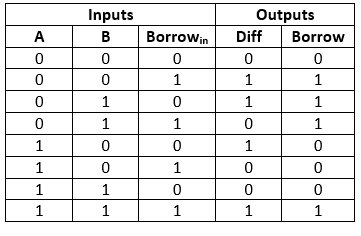
## Theory: -

The Half Subtractor is used to subtract only two numbers. To overcome this problem, a full subtractor was designed. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minuend, subtrahend, and borrow, respectively. The full subtractor has three input states and two output states i.e., diff and borrow.

### Block diagram



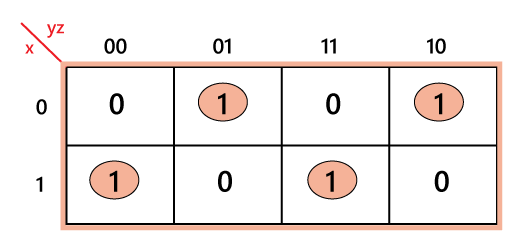
### Truth Table



In the above table,

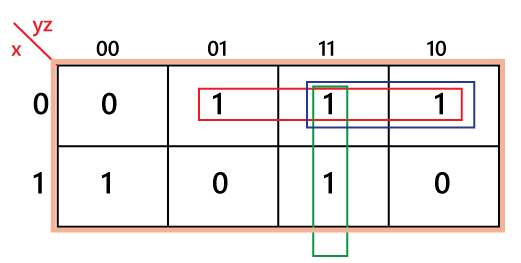
* 'A' and' B' are the input variables. These variables represent the two significant bits that are going to be subtracted.
* 'Borrowin' is the third input which represents borrow.
* The 'Diff' and 'Borrow' are the output variables that define the output values.
* The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

The SOP form can be obtained with the help of K-map as:



Diff=xy'z' + x'y'z + xyz + x'yz' = (x xor y) xor z

Java Try Catch



Borrow=x'z + x'y + yz = z (x xory)’ + x’y

## Source Code

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity full\_sub is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

sum : out STD\_LOGIC;

diff : out STD\_LOGIC);

end full\_sub;

architecture Behavioral of full\_sub is

begin

diff <= (a xor b) xor c;

sum <= ((not a) and c ) or ((not a ) and b) or (b and c);

end Behavioral;

## Testbench Code

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

entity full\_sub\_tb is

end;

architecture bench of full\_sub\_tb is

component full\_sub

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

sum : out STD\_LOGIC;

diff : out STD\_LOGIC);

end component;

signal a: STD\_LOGIC;

signal b: STD\_LOGIC;

signal c: STD\_LOGIC;

signal sum: STD\_LOGIC;

signal diff: STD\_LOGIC;

begin

uut: full\_sub port map ( a => a,

b => b,

c => c,

sum => sum,

diff => diff );

stimulus: process

begin

-- Put initialisation code here

a <= '0';

b <= '0';

c <= '0';

wait for 10ns;

a <= '0';

b <= '0';

c <= '1';

wait for 10ns;

a <= '0';

b <= '1';

c <= '0';

wait for 10ns;

a <= '0';

b <= '1';

c <= '1';

wait for 10ns;

a <= '1';

b <= '0';

c <= '0';

wait for 10ns;

a <= '1';

b <= '0';

c <= '1';

wait for 10ns;

a <= '1';

b <= '1';

c <= '0';

wait for 10ns;

a <= '1';

b <= '1';

c <= '1';

wait for 10ns;

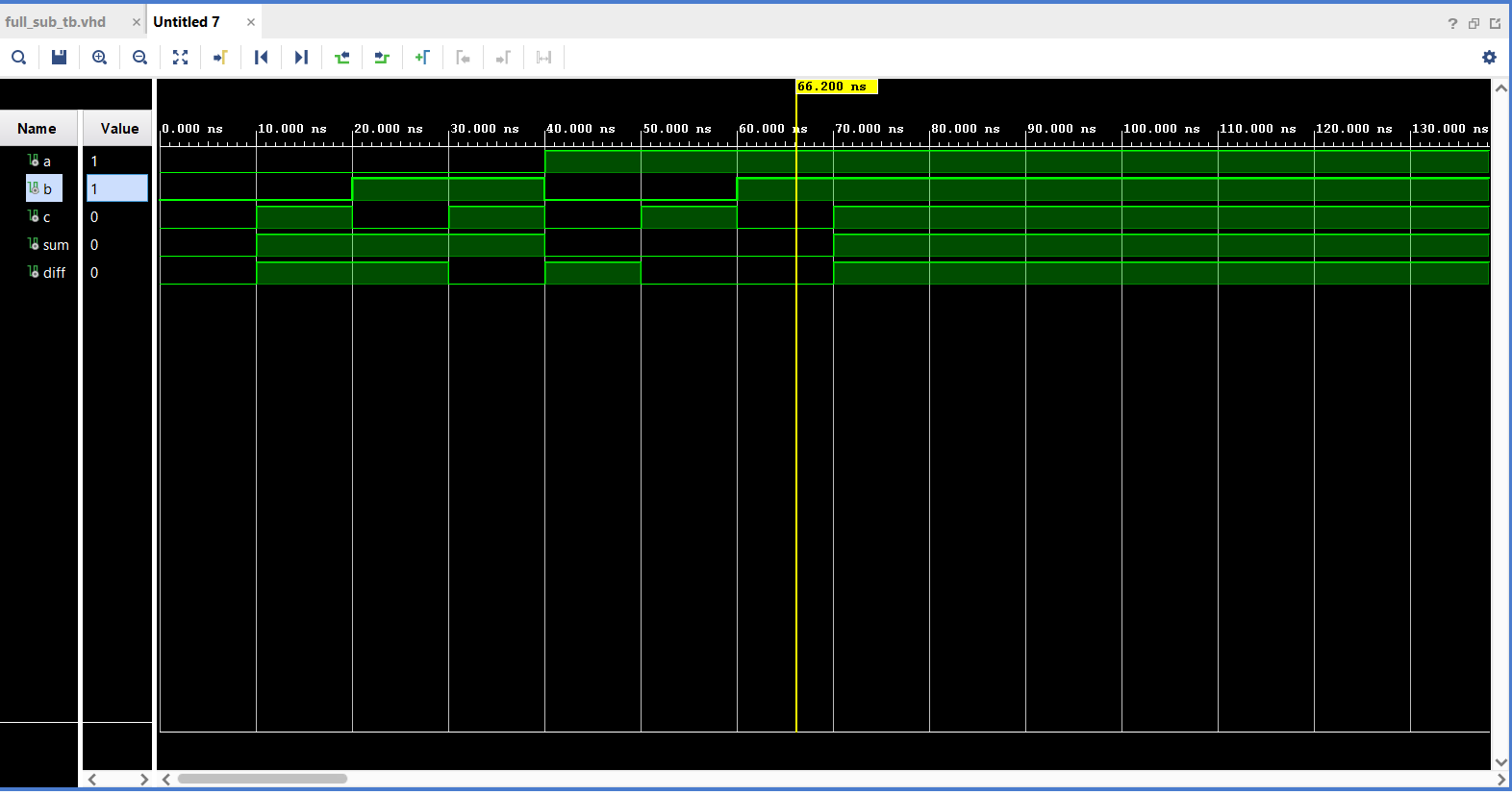
-- Put test bench stimulus code here

wait;

end process;

end;

## Observation



## Output

