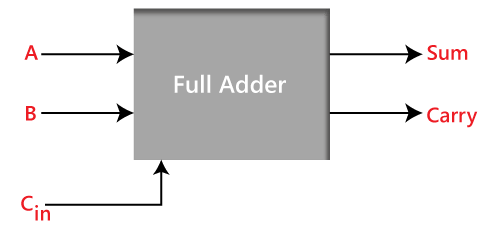
# Full Adder

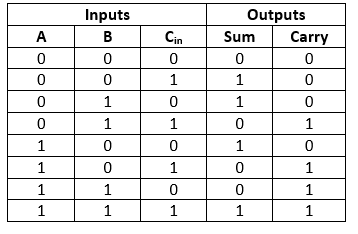
## Theory

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

### Block diagram



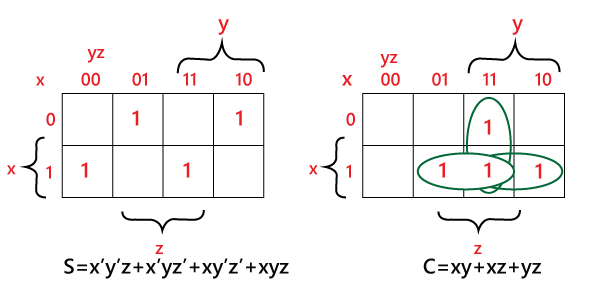
### Truth Table



In the above table,

1. 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added
2. 'Cin' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
3. The 'Sum' and 'Carry' are the output variables that define the output values.
4. 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:



Sum = x' y' z+x' yz+xy' z'+xyz  
Carry = xy+xz+yz

## 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 fa is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

carry : out STD\_LOGIC;

sum : out STD\_LOGIC);

end fa;

architecture Behavioral of fa is

begin

sum <= a xor (b xor c);

carry <= (a and b) or (b and c) or (a and c);

end Behavioral;

## Testbench Code

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

entity fa\_tb is

end;

architecture bench of fa\_tb is

component fa

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

carry : out STD\_LOGIC;

sum : out STD\_LOGIC);

end component;

signal a: STD\_LOGIC;

signal b: STD\_LOGIC;

signal c: STD\_LOGIC;

signal carry: STD\_LOGIC;

signal sum: STD\_LOGIC;

begin

uut: fa port map ( a => a,

b => b,

c => c,

carry => carry,

sum => sum );

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;

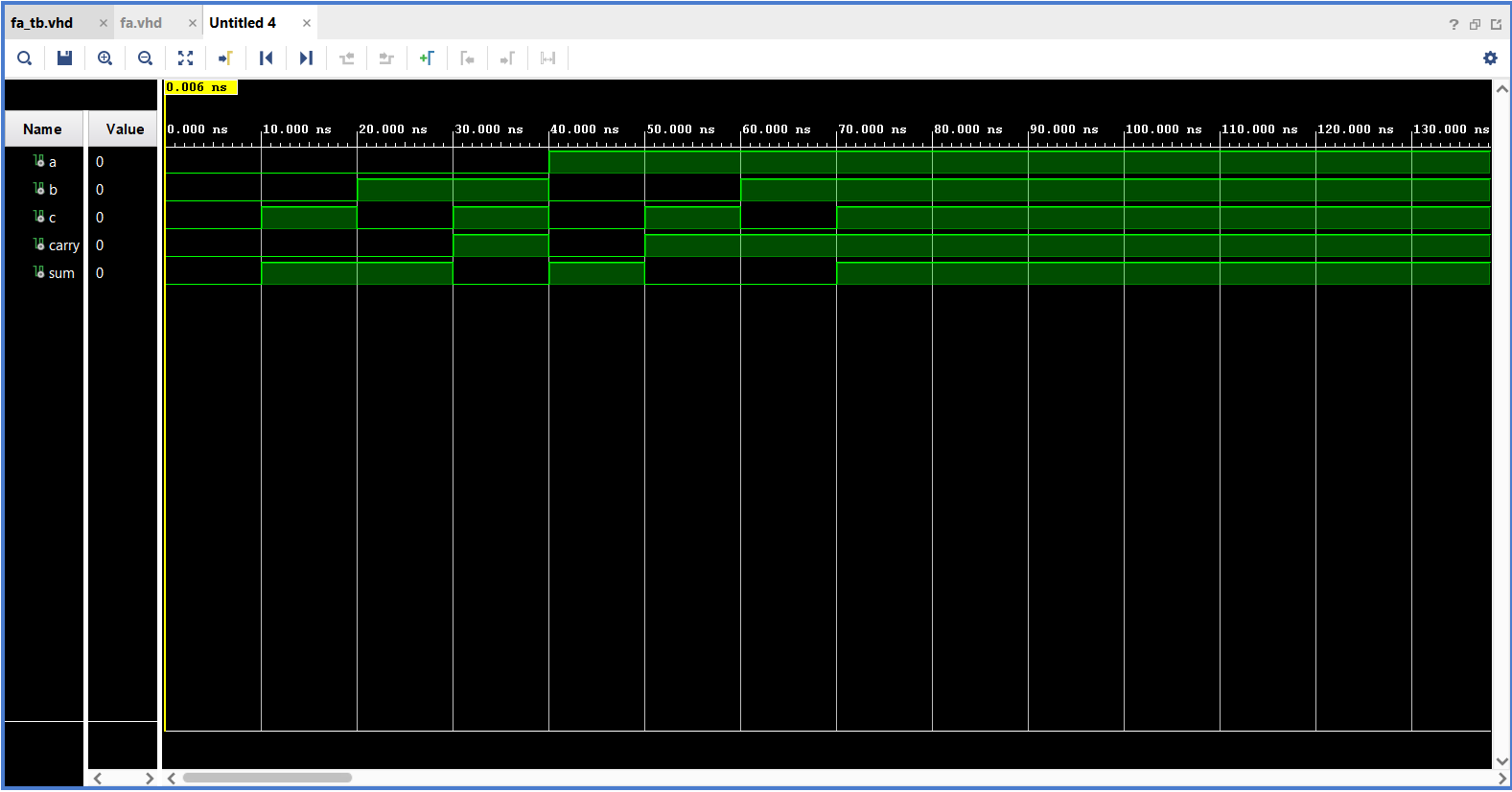
-- Put test bench stimulus code here

wait;

end process;

end;

## Observation



## Output

