Assignment 1: Booth Multiplier

Aaron John Sabu: 170070050

April 19, 2019

1 Overview of the assignment

- The aim of this assignment is to design a **VHDL** code to simulate the functionality of a **booth multiplier**.
- Furthermore, this experiment involves the testing of the code using some predefined test cases generated using a C++ script.

2 Components

The construction of the booth multiplier involves the direct use of a sixteen-bit adder and a 3-bit-controlled 8-channel multiplexer.
The implementation of booth multiplier involved obtaining the proper output for every three bits of B through A. The custom multiplexer took inputs of three bits from B and the whole of A, and gives the output of the below given table:

$B_2B_1B_0$	Output
000	0
001	+A
010	+A
011	+2A
100	-2A
101	-A
110	-A
111	0

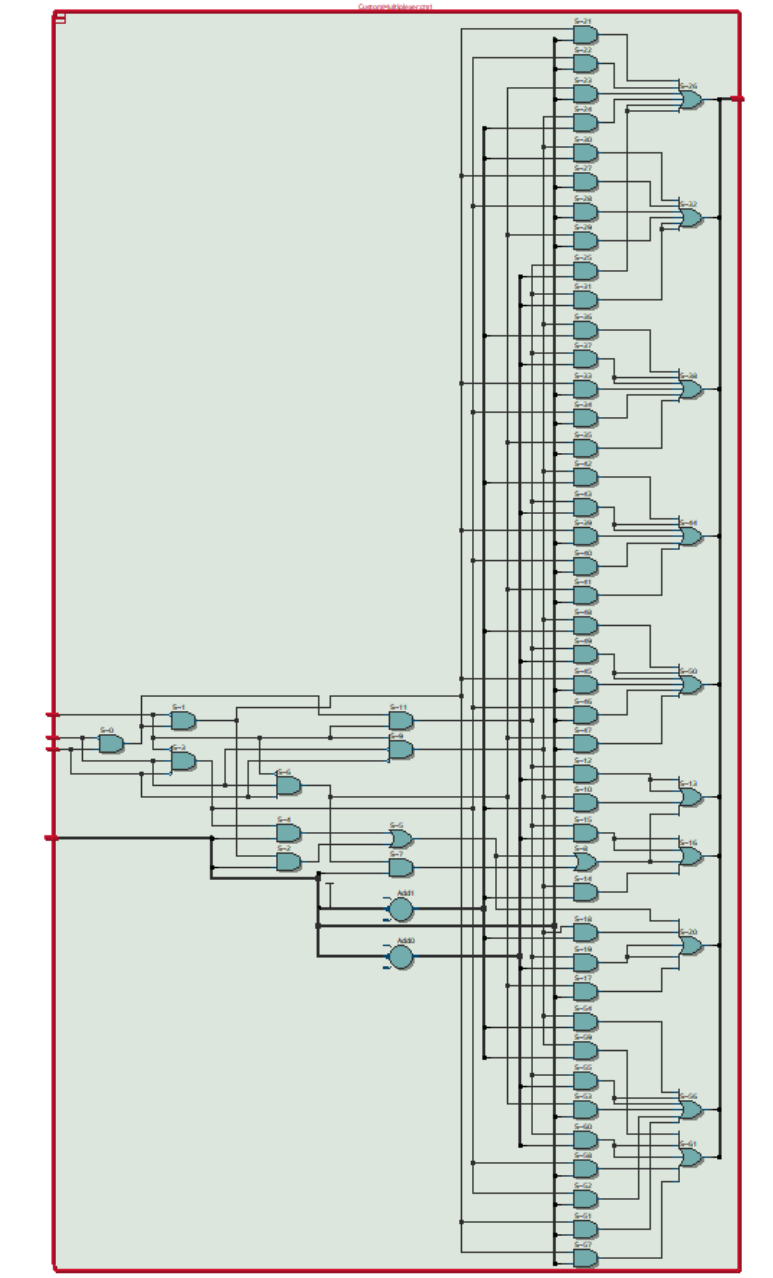


Figure 1: Custom Multiplexer

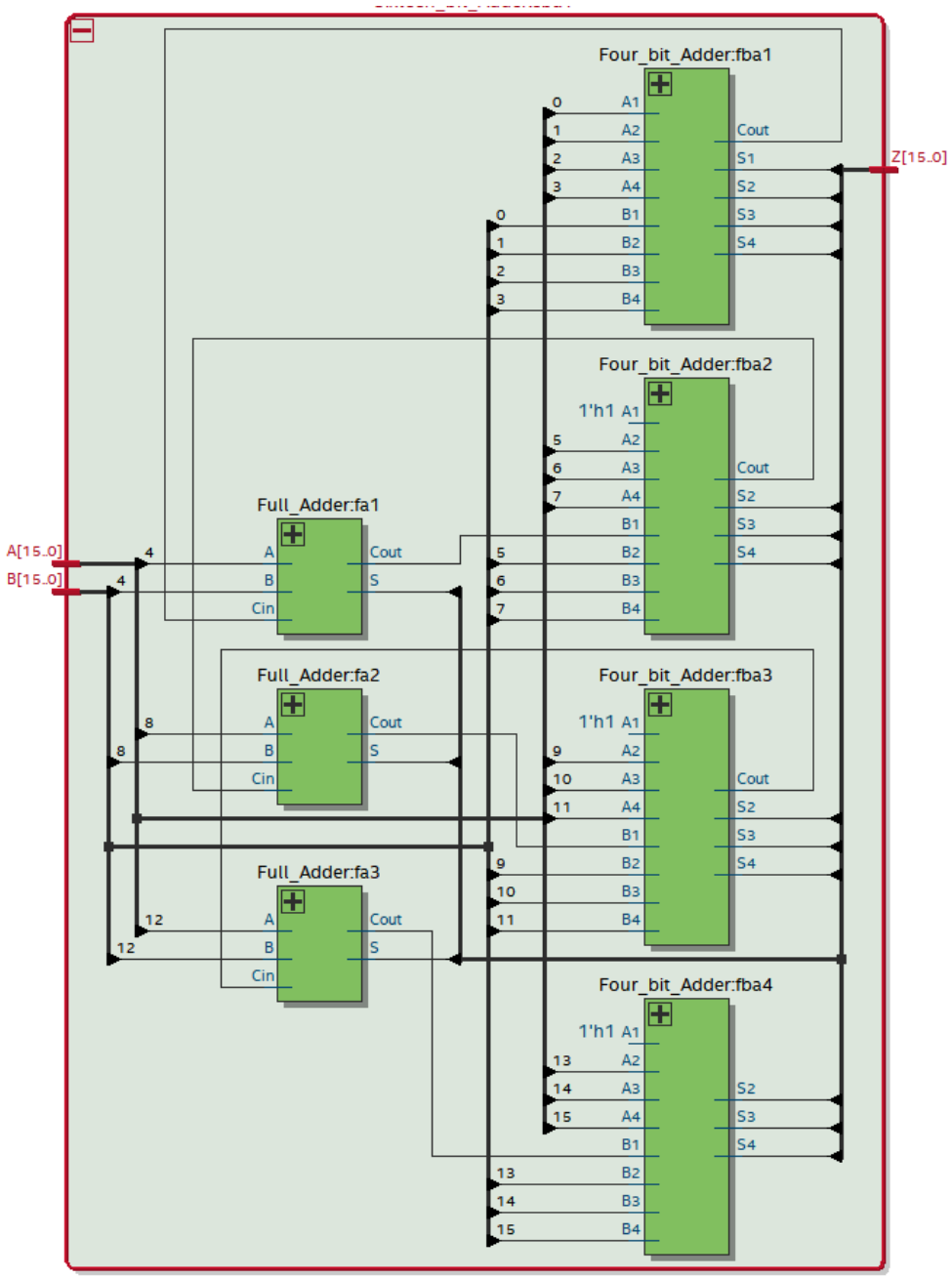


Figure 2: Sixteen Bit Adder

3 Design

The input A of 8 bits is passed into the custom multiplexer with the control pins following the given pattern:

Multiplexer	Control Pins
1	$(B(1), B(0), 0)$
2	$(B(3), B(2), B(1))$
3	$(B(5), B(4), B(3))$
4	$(B(7), B(6), B(5))$

Following this, the output is shifted left by double the number of computation; i.e. for the first computation $((B_2, B_1, B_0) = (B(1), B(0), 0))$, it is not shifted, for the next $(((B_2, B_1, B_0) = (B(4), B(3), B(2))))$, it is shifted left by two positions and so on.
The hence calculated (shifted) four values are summed up to give the final output. The overall circuit behaves as shown below:

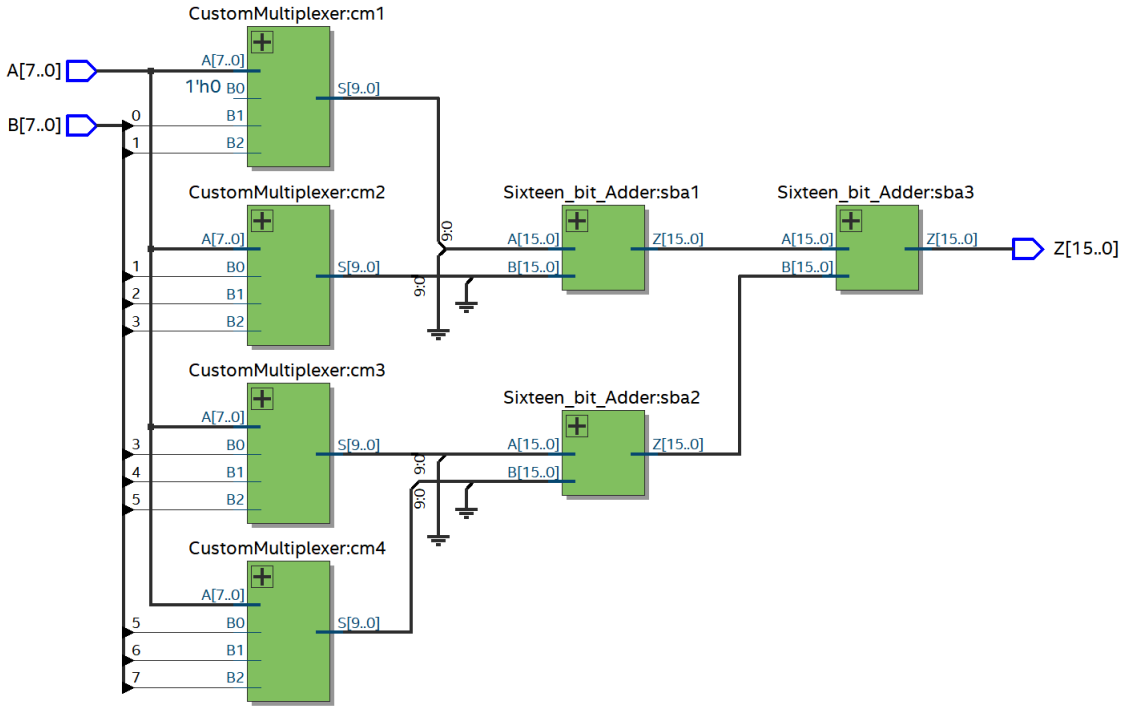


Figure 3: Netlist for the design

Thus Booth encoding reduces the number of partial products to half (multiplying 2 bits at a time). It makes addition in columns of partial products fast because carry propagation during addition will be reduced.

4 VHDL Codes

Gates.vhdl, **Four_bit_Adder.vhdl** and **DUT.vhdl** have been used as basic codes to the project. Moreover, **tracefile_generator.cpp** has been used to generate the TRACEFILE.txt file. The codes for the main code, the sixteen-bit-adder, the custom multiplexer and the C++ code to generate the tracefile are given below:

4.1 Main.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
entity Main  is
  port (A, B: in std_logic_vector(7 downto 0);
    Z: out std_logic_vector(15 downto 0));
end entity Main;
architecture Struct of Main is
  signal C1, C2, C3, C4, C5, C6 : std_logic_vector(15 downto 0);
  signal temp1, temp2, temp3, temp4: std_logic_vector(9  downto 0);
  component CustomMultiplexer is
    port(B2, B1, B0: in std_logic; A: in std_logic_vector(7 downto 0);
      S: out std_logic_vector(9 downto 0));
  end component;
  component Sixteen_Bit_Adder  is
    port (A, B: in std_logic_vector(15 downto 0);
      Z: out std_logic_vector(15 downto 0));
  end component Sixteen_Bit_Adder;
  begin
    cm1 :  CustomMultiplexer port map(B2 => B(1), B1 => B(0), B0 => '0',
      A => A, S => temp1);
    C1 <=  "000000" & temp1;
    cm2 :  CustomMultiplexer port map(B2 => B(3), B1 => B(2), B0 => B(1),
      A => A, S => temp2);
```

```
    C2 <=  ("0000" & temp2) & "00";
    cm3 :  CustomMultiplexer port map(B2 => B(5), B1 => B(4), B0 => B(3),
      A => A, S => temp3);
    C3 <=  ("00" & temp3) & "0000";
    cm4 :  CustomMultiplexer port map(B2 => B(7), B1 => B(6), B0 => B(5),
      A => A, S => temp4);
    C4 <=  temp4 & "000000";
    sba1:  Sixteen_Bit_Adder port map(A => C1, B => C2, Z => C5);
    sba2:  Sixteen_Bit_Adder port map(A => C3, B => C4, Z => C6);
    sba3:  Sixteen_Bit_Adder port map(A => C5, B => C6,Z => Z);
end Struct;
```

4.2 Sixteen_bit_Adder.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.Gates.all;
entity Sixteen_bit_Adder  is
  port (A, B: in std_logic_vector(15 downto 0); Z: out std_logic_vector(15 downto
end entity Sixteen_bit_Adder;
architecture Struct of Sixteen_bit_Adder is
  signal C1, C2, C3, C4, C5, C6, C7, W1, W2, W3: std_logic;
  component Four_bit_Adder  is
    port (A1, A2, A3, A4, B1, B2, B3, B4: in std_logic;
      S1, S2, S3, S4, Cout: out std_logic);
  end component Four_bit_Adder;
begin
  fba1: FOUR_BIT_ADDER  port map (A1 => A(0), A2 => A(1), A3  => A(2),
    A4 => A(3), B1   => B(0), B2 => B(1), B3 => B(2), B4 => B(3), S1 => Z(0),
    S2 => Z(1), S3 => Z(2), S4 => Z(3), Cout => C1);
  fa1:  FULL_ADDER port map (A  => A(4), B  => B(4), Cin => C1, S => Z(4),
    Cout => C2);
  fba2: FOUR_BIT_ADDER  port map (A1 => '1',  A2 => A(5), A3  => A(6),
    A4 => A(7), B1   => C2, B2 => B(5), B3 => B(6), B4 => B(7), S1 => W1,
    S2 => Z(5), S3 => Z(6), S4 => Z(7), Cout => C3);
  fa2:  FULL_ADDER  port map (A  => A(8), B  => B(8), Cin => C3, S  => Z(8),
```

```
    Cout => C4);
  fba3: FOUR_BIT_ADDER  port map (A1 => '1',  A2 => A(9), A3  => A(10),
    A4 => A(11), B1   => C4, B2 => B(9), B3 => B(10), B4 => B(11), S1 => W2,
    S2 => Z(9), S3 => Z(10), S4 => Z(11), Cout => C5);
  fa3:  FULL_ADDER port map (A  => A(12), B  => B(12), Cin => C5, S  => Z(12),
    Cout => C6);
  fba4: FOUR_BIT_ADDER  port map (A1 => '1',  A2 => A(13), A3  => A(14),
    A4 => A(15), B1   => C6, B2 => B(13), B3 => B(14), B4 => B(15), S1 => W3,
    S2 => Z(13), S3 => Z(14), S4 => Z(15), Cout => C7);
end Struct;
```

4.3 Custom_Multiplexer.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library work;
use work.Gates.all;
entity CustomMultiplexer is
  port(B2, B1, B0: in std_logic; A: in std_logic_vector(7 downto 0);
    S: out std_logic_vector(9 downto 0));
end entity;
architecture Struct of CustomMultiplexer is
  signal A10, minA10, plus2A, min2A: std_logic_vector(9 downto 0);
begin
  -- component instances
  A10   <= ((A(7) & A(7)) &   A);
  plus2A <= ((A(7) & A)  & '0');
  minA10 <= std_logic_vector(unsigned(not(A10))    +1);
  min2A  <= std_logic_vector(unsigned(not(plus2A)) +1);

  S(9)  <=  ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
    or (A10(9) and (not(B2) and (not(B1) and   (B0) ) ) )
    or (A10(9) and  (not(B2) and (  (B1) and not(B0) ) ) )
    or (plus2A(9) and (not(B2) and (  (B1) and   (B0) ) ) )
    or (min2A(9) and  ((B2)  and (not(B1) and not(B0) ) ) )
    or (minA10(9) and  ((B2)  and (not(B1) and   (B0) ) ) )
    or (minA10(9) and  ((B2)  and (not(B1) and   (B0) ) ) )
```

```
  or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
S(8)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
  or (A10(8) and (not(B2) and (not(B1) and   (B0) ) ) )
  or (A10(8) and  (not(B2) and (  (B1) and not(B0) ) ) )
  or (plus2A(8) and (not(B2) and (  (B1) and   (B0) ) ) )
  or (min2A(8) and  ((B2)   and (not(B1) and not(B0) ) ) )
  or (minA10(8) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or (minA10(8) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
S(7)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
  or (A10(7) and (not(B2) and (not(B1) and   (B0) ) ) )
  or (A10(7) and  (not(B2) and (  (B1) and not(B0) ) ) )
  or (plus2A(7) and (not(B2) and (  (B1) and   (B0) ) ) )
  or (min2A(7) and  ((B2)   and (not(B1) and not(B0) ) ) )
  or (minA10(7) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or (minA10(7) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
S(6)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
  or (A10(6) and (not(B2) and (not(B1) and   (B0) ) ) )
  or (A10(6) and  (not(B2) and (  (B1) and not(B0) ) ) )
  or (plus2A(6) and (not(B2) and (  (B1) and   (B0) ) ) )
  or (min2A(6) and  ((B2)   and (not(B1) and not(B0) ) ) )
  or (minA10(6) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or (minA10(6) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
S(5)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
  or (A10(5) and (not(B2) and (not(B1) and   (B0) ) ) )
  or (A10(5) and  (not(B2) and (  (B1) and not(B0) ) ) )
  or (plus2A(5) and (not(B2) and (  (B1) and   (B0) ) ) )
  or (min2A(5) and  ((B2)   and (not(B1) and not(B0) ) ) )
  or (minA10(5) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or (minA10(5) and  ((B2)   and (not(B1) and   (B0) ) ) )
  or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
S(4)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
  or (A10(4) and (not(B2) and (not(B1) and   (B0) ) ) )
  or (A10(4) and  (not(B2) and (  (B1) and not(B0) ) ) )
  or (plus2A(4) and (not(B2) and (  (B1) and   (B0) ) ) )
  or (min2A(4) and  ((B2)   and (not(B1) and not(B0) ) ) )
```

```
    or (minA10(4) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or (minA10(4) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
  S(3)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
    or (A10(3) and (not(B2) and (not(B1) and   (B0) ) ) )
    or (A10(3) and  (not(B2) and (  (B1) and not(B0) ) ) )
    or (plus2A(3) and (not(B2) and (  (B1) and   (B0) ) ) )
    or (min2A(3) and  ((B2)   and (not(B1) and not(B0) ) ) )
    or (minA10(3) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or (minA10(3) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
  S(2)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
    or (A10(2) and (not(B2) and (not(B1) and   (B0) ) ) )
    or (A10(2) and  (not(B2) and (  (B1) and not(B0) ) ) )
    or (plus2A(2) and (not(B2) and (  (B1) and   (B0) ) ) )
    or (min2A(2) and  ((B2)   and (not(B1) and not(B0) ) ) )
    or (minA10(2) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or (minA10(2) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
  S(1)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
    or (A10(1) and (not(B2) and (not(B1) and   (B0) ) ) )
    or (A10(1) and  (not(B2) and (  (B1) and not(B0) ) ) )
    or (plus2A(1) and (not(B2) and (  (B1) and   (B0) ) ) )
    or (min2A(1) and  ((B2)   and (not(B1) and not(B0) ) ) )
    or (minA10(1) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or (minA10(1) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
  S(0)  <=     ('0'  and  (not(B2) and (not(B1) and not(B0) ) ) )
    or (A10(0) and (not(B2) and (not(B1) and   (B0) ) ) )
    or (A10(0) and  (not(B2) and (  (B1) and not(B0) ) ) )
    or (plus2A(0) and (not(B2) and (  (B1) and   (B0) ) ) )
    or (min2A(0) and  ((B2)   and (not(B1) and not(B0) ) ) )
    or (minA10(0) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or (minA10(0) and  ((B2)   and (not(B1) and   (B0) ) ) )
    or ('0'  and  ((B2)   and (  (B1) and   (B0) ) ) );
end Struct;
```

4.4 tracefile_generator.cpp

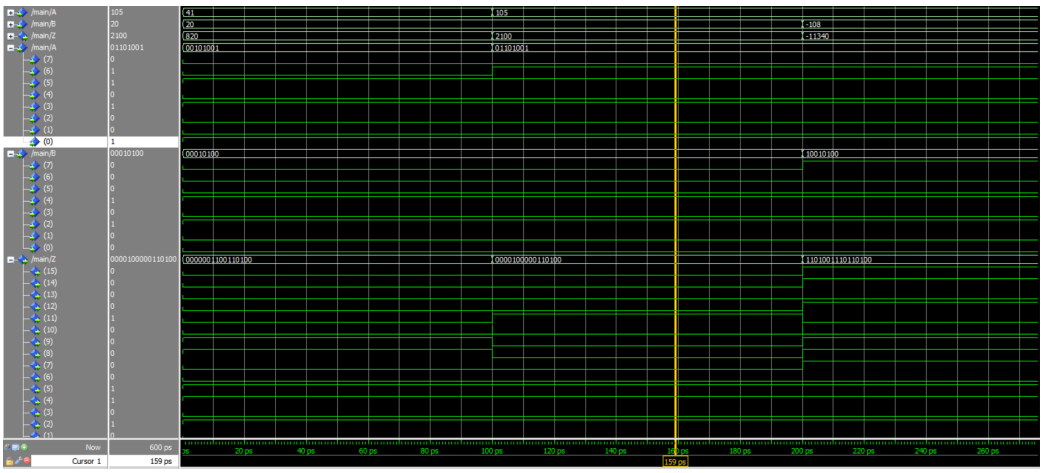
```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <vector>
#include <cmath>
#include <fstream>
using namespace std;

void multiplier_generator(int a, int b){
    ofstream outfile("TRACEFILE.txt");
    int c = a*b;
    vector<int> array_1;
    vector<int> array_2;
    vector<int> output;
    for(int i=0; i<8; i++){
        array_1.push_back(a%2);
        array_2.push_back(b%2);
        a/=2;
        b/=2;
    }
    for(int i=0; i<16; i++){
        output.push_back(c%2);
        c/=2;
    }
    for(int j = 0; j<8; j++)
        cout<<array_1[7-j];
    for(int k = 0; k<8; k++)
        cout<<array_2[7-k];
    cout<<" ";
    for(int l= 0; l<16; l++)
        cout<<output[15-l];
    cout<<" 1111111111111111"<<endl;
    outfile.close();
    return;
}
```

```
int main(){
    for(int x = 0; x<256; x++)
        for(int y =0; y<256; y++)
            multiplier_generator(x,y);
    return 0;
}
```

5 Output

Given below are snapshots of some portions of the digital waveform obtained from the four-bit adder simulation in ModelSim-Altera:

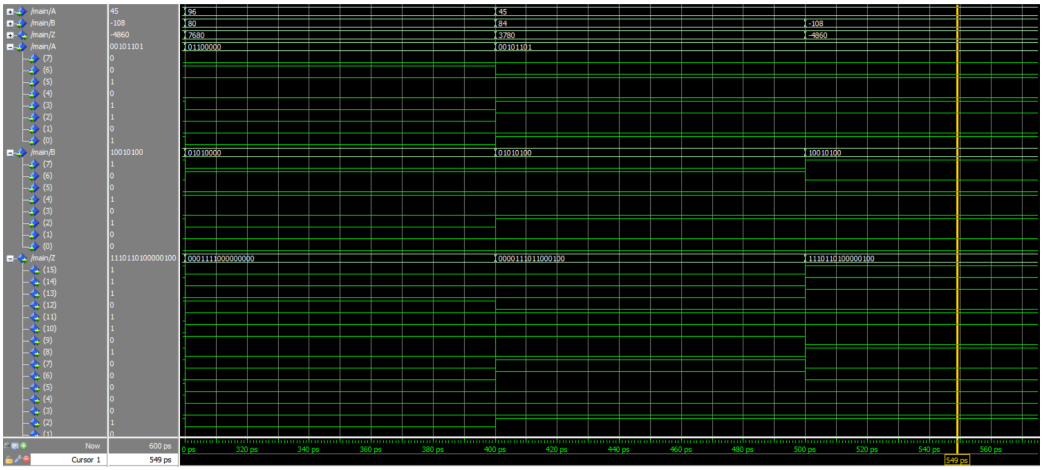


(a) Wave 1

$$41 \times 20 = 820;\ i.e.\ 00101001 \times 00010100 = 0000001100110100$$

$$105 \times 20 = 2100;\ i.e.\ 01101001 \times 00010100 = 0000100000110100$$

$$105 \times -108 = -11340;\ i.e.\ 01101001 \times 10010100 = 1101001110110100$$



(b) Wave 2

Figure 4: Digital Waveform

$$96 \times 80 = 7680;\ \textit{i.e.}\ 01100000 \times 01010000 = 0001111000000000$$

$$45 \times 84 = 3780;\ \textit{i.e.}\ 00101101 \times 01010100 = 0000111011000100$$

$$45 \times -108 = -4860;\ \textit{i.e.}\ 00101101 \times 10010100 = 1110110100000100$$

Also given below is a portion of the TRACEFILE generated from the code given above:

```
0000000000000000000000000000000000 1111111111111111
0000000000000001000000000000000000 1111111111111111
0000000000000010000000000000000000 1111111111111111
0000000000000011000000000000000000 1111111111111111
0000000000000100000000000000000000 1111111111111111
0000000000000101000000000000000000 1111111111111111
0000000000000110000000000000000000 1111111111111111
...
0001100110101110 0001000011111110 1111111111111111
0001100110101111 0001000100010111 1111111111111111
0001100110110000 0001000100110000 1111111111111111
0001100110110001 0001000101001001 1111111111111111
```

```
0001100110110010 0001000101100010 1111111111111111
0001100110110011 0001000101111011 1111111111111111
...

01110111010101100010011111111010 1111111111111111
01110111010101110010100001110001 1111111111111111
01110111010110000010100011101000 1111111111111111
01110111010110010010100101011111 1111111111111111
01110111010110100010100111010110 1111111111111111
01110111010110110010101001001101 1111111111111111
01110111010111000010101011000100 1111111111111111
...
11111111111110011111100000000111 1111111111111111
11111111111110101111100100000110 1111111111111111
11111111111110111111101000000101 1111111111111111
11111111111111001111101100000100 1111111111111111
11111111111111011111110000000011 1111111111111111
11111111111111101111110100000010 1111111111111111
11111111111111111111111000000001 1111111111111111
```