Al-Azhar University
Faculty of Engineering
Systems and Computers Dep.
Class of 2021/2022

Computer Architecture Project

Booth's Algorithm VHDL Implementation

Presented To:

- Dr. Khaled El-shafei

Presented By:

- Mohamed Salah Sayed [102]
- Ahmed Mohamed Atya [016]
- Mohamed Hosam Bayome [096]

Booth's Algorithm

Problem Description

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation.

Booth's algorithm examines adjacent pairs of bits of the 'N'-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit y_i for i running from 0 to $N - 1$, the bits y_i and y_{i-1} are considered.

Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2^i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2^i is subtracted from P. The final value of P is the signed product.

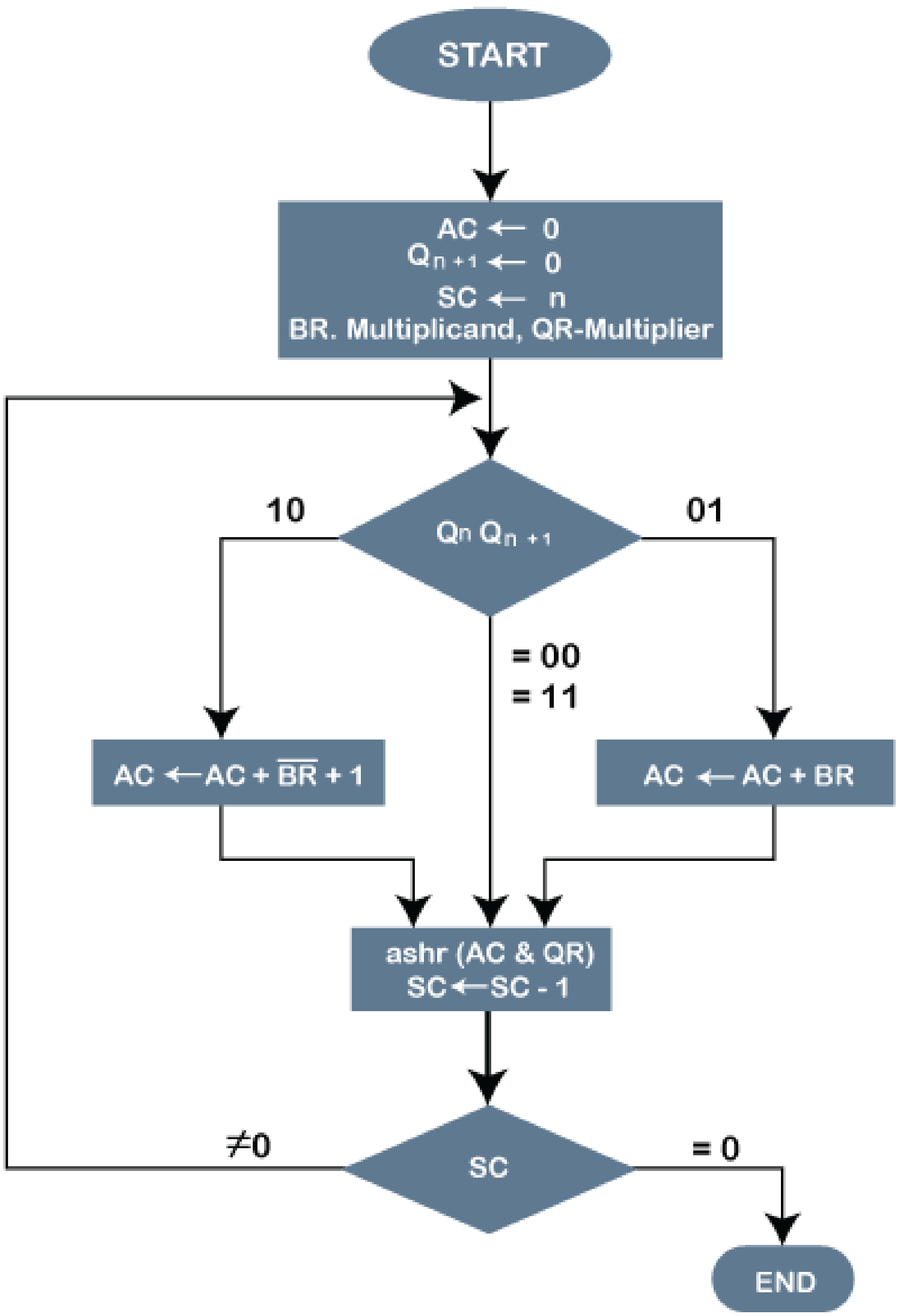
Implementation

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P. Let m and r be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in m and r.

1. Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to (x + y + 1).
    a. A: Fill the most significant (leftmost) bits with the value of m. Fill the remaining (y + 1) bits with zeros.
    b. S: Fill the most significant bits with the value of (−m) in two's complement notation. Fill the remaining (y + 1) bits with zeros.
    c. P: Fill the most significant x bits with zeros. To the right of this, append the value of r. Fill the least significant (rightmost) bit with a zero.
2. Determine the two least significant (rightmost) bits of P.
    a. If they are 01, find the value of P + A. Ignore any overflow.
    b. If they are 10, find the value of P + S. Ignore any overflow.
    c. If they are 00, do nothing. Use P directly in the next step.
    d. If they are 11, do nothing. Use P directly in the next step.
3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.

4. Repeat steps 2 and 3 until they have been done y times.
5. Drop the least significant (rightmost) bit from P.
   This is the product of m and r.

Flowchart Diagram



Example

Let A: 3 and B: 17

Multiplicand -	
Decimal:	3
Binary:	00000011
Multiplier -	
Decimal:	17
Binary:	00010001
Two's Complement:	11101111
Steps -	
Starting Out:	0000000000000011
Subtract:	1110111100000011
Shift:	1111011110000001
Shift:	1111101111000000
Add:	0000110011000000
Shift:	0000011001100000

Shift:	0000001100110000
Shift:	0000000110011000
Shift:	0000000011001100
Shift:	0000000001100110
Shift:	0000000000110011
Final Product (Binary):	0000000000110011
Final Product (Decimal):	51

VHDL Implementation

*C:\Users\Mohamed Salah\Desktop\booth.vhd - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

booth.vhd

```vhdl
-------------------------------------------------------------------------------
-- Project              : Booth's Algorithm
-- Prsented To          : Dr.Khaled AlShafey
-- University / Dep     : Faculty of engineering - Al Azhar University - CS Dep.
-- Prsented By          : 1- Mohamed Salah Sayed [102]
--                        2- Ahmed Mohamed Atya [16]
--                        3- Mohamed Hosam Bayome [96]
-------------------------------------------------------------------------------
--
-- Description :
-- Booth algorithm gives a procedure for multiplying binary integers in signed 2's
-- complement representation in efficient way, i.e., less number of additions/subtractions required.
-- It operates on the fact that strings of 0's in the multiplier require no addition but just shifting
-- and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1 ) to 2^m.
-- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting
-- of the partial product. Prior to the shifting, the multiplicand may be added to the partial product,
-- subtracted from the partial product, or left unchanged according to following rules:

-- 1- The multiplicand is subtracted from the partial product upon encountering
-- the first least significant 1 in a string of 1's in the multiplier
--
-- 2- The multiplicand is added to the partial product upon encountering the first 0
-- (provided that there was a previous '1') in a string of 0's in the multiplier.
--
-- 3- The partial product does not change when the multiplier bit is identical
-- to the previous multiplier bit.
--
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.NUMERIC_STD.all;

entity booth is
     port(
         A : in signed(7 downto 0);
         B : in signed(7 downto 0);
         C : out signed(15 downto 0)
         );
end booth;

architecture Behavior of booth is

begin

    process (A,B)
  variable tmp_B: signed(8 downto 0) ;
  variable tmp_out: signed(15 downto 0) ;
  variable tmp_A: signed(15 downto 0) ;
  variable fixed_A : signed(15 downto 0) ;
    variable i : integer ;

    begin

        tmp_b := "000000000" ;
        fixed_A := "0000000000000000" ;
        tmp_out := "0000000000000000" ;
    tmp_A := "0000000000000000";
    tmp_b(8 downto 1) := b ;
        fixed_A(7 downto 0 ) := A ;
    for i in 0 to 7 loop
        tmp_A := fixed_A ;
        if ((tmp_b(i+1 downto i) = "01")) then
            tmp_A := shift_left(tmp_A , i) ;
            tmp_out := tmp_out + tmp_A ;
        end if;
        if ((tmp_b(i+1 downto i) = "10"  )) then   --000101
            tmp_A := -tmp_A;

            tmp_A := shift_left(tmp_A , i) ;
            tmp_out := tmp_out + tmp_A ;
        end if ;
    end loop;
    c <= tmp_out ;
    end process ;


end Behavior;
```

Testbench

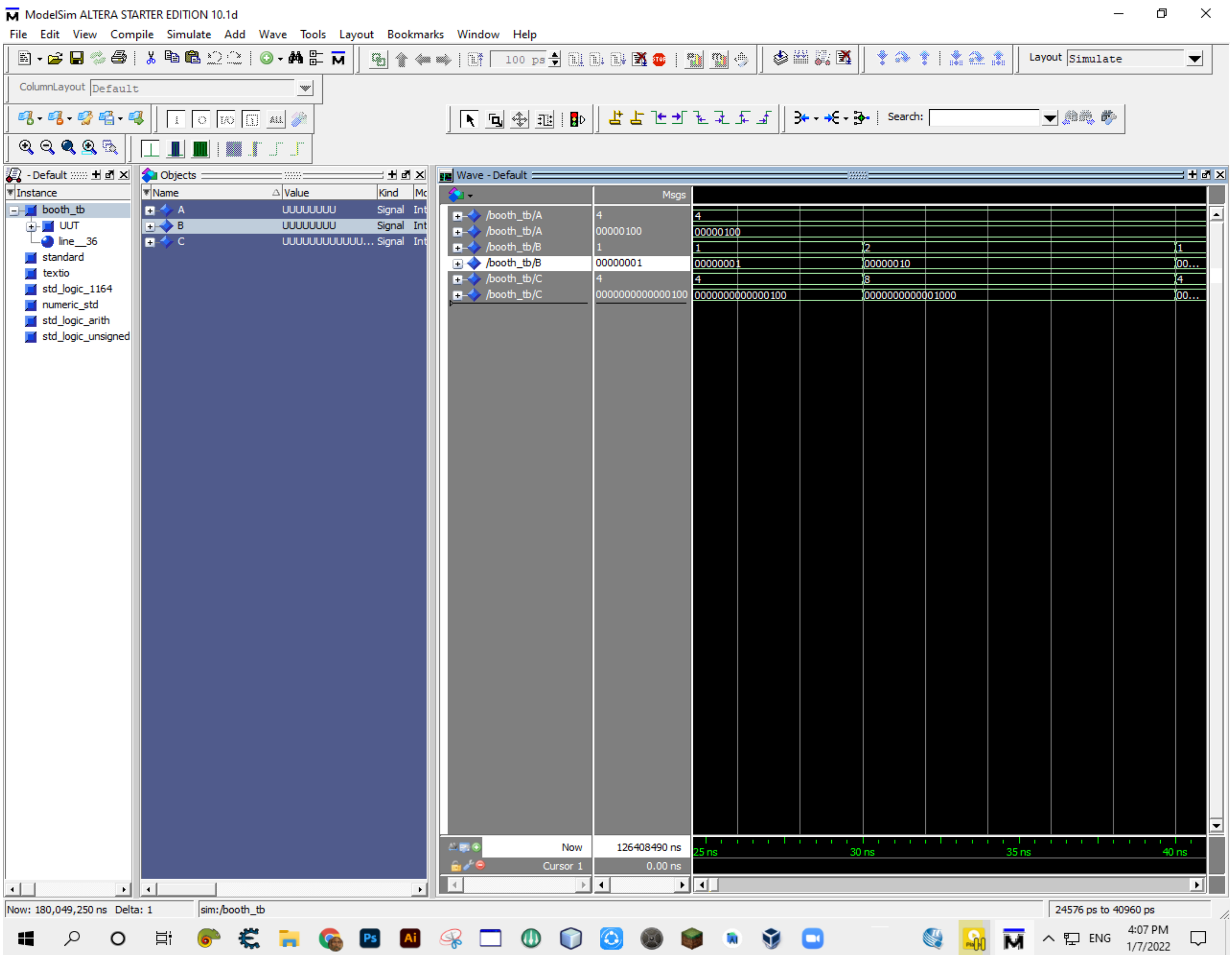
C:\Users\Mohamed Salah\Desktop\booth_tb.vhd - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

booth.vhd | booth_tb.vhd

```vhdl
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity booth_tb is
end booth_tb;

architecture TB_ARCHITECTURE of booth_tb is

    component booth
    port(
        A : in SIGNED(7 downto 0);
        B : in SIGNED(7 downto 0);
        C : out SIGNED(15 downto 0) );
    end component;

    -- Stimulus signals -
    signal A : SIGNED(7 downto 0);
    signal B : SIGNED(7 downto 0);
    -- Observed signals -
    signal C : SIGNED(15 downto 0);


begin

    UUT : booth
        port map (
            A => A,
            B => B,
            C => C
        );

process
    begin
        A <= "00000100";
        B <= "00000001" ;
        WAIt FOR 10 NS ;
        A <= "00000100";
        B <= "00000010" ;
        WAIt FOR 10 NS ;
        A <= "00000101";
        B <= "00000010" ;

    end process ;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_booth of booth_tb is
    for TB_ARCHITECTURE
        for UUT : booth
            use entity work.booth(booth);
        end for;
    end for;
end TESTBENCH_FOR_booth;
```

Simulation Results



Best Case and Worst-Case Occurrence:

Best case is when there is a large block of consecutive 1's and 0's in the multipliers, so that there is minimum number of logical operations taking place, as in addition and subtraction.

Worst case is when there are pairs of alternate 0's and 1's, either 01 or 10 in the multipliers, so that maximum number of additions and subtractions are required

References :

Citation and historical ref:

https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm

Graphics and Samples ref:

https://www.geeksforgeeks.org/computer-organization-booths-algorithm

Graphics and Samples ref:
https://www.javatpoint.com/booths-multiplication-algorithm-in-coa

Useful Tools:

https://www.ecs.umass.edu/ece/koren/arith/simulator/Booth/

https://rndtool.info/booth-algorithm-step-by-step-calculator/