**Laboratory Report Cover Sheet**

**18ECE206J ADVANCED DIGITAL SYSTEMS DESIGN**

**Fourth Semester, 2021-22 (Even semester)**

SRM Institute of Science and Technology College of Engineering and Technology

Department of Electronics and Communication Engineering

**Name :**

**Register No. :**

**Day/ Session :**

**Venue :**

**Title of Experiment :**

**Date of Conduction :**

**Date of Submission :**

|  |  |  |
| --- | --- | --- |
| **Particulars** | **Max. Marks** | **Marks**  **Obtained** |
| Pre lab and Post lab | 10 |  |
| Lab Performance | 20 |  |
| Simulation and results | 10 |  |
| Total | 40 |  |

**REPORT VERIFICATION**

**Staff Name : Signature :**

## 8. Design of 4x4 Wallace Tree Multiplier

**Aim**: To design and verify 4x4 Wallace Tree multiplier using VHDL.

**Software Required:** Xilinx ISE & ModelSim

##### Theory

A **Wallace tree** is an efficient hardware implementation of a digital circuit that multiplies two integers, devised by Australian Computer Scientist Chris Wallace in 1964. The Wallace tree has three steps:

1. Multiply (that is – AND) each bit of one of the arguments, by each bit of the other, yielding results. Depending on position of the multiplied bits, the wires carry different weights, for example wire of bit carrying result of is 128
2. Reduce the number of partial products to two by layers of full and half adders.
3. Group the wires in two numbers, and add them with a conventional adder.

The second step works as follows. As long as there are three or more wires with the same weight add a following layer:-

* Take any three wires with the same weights and input them into a full adder. The result will be an output wire of the same weight and an output wire with a higher weight for each three input wires.
* If there are two wires of the same weight left, input them into a half adder.
* If there is just one wire left, connect it to the next layer.

The benefit of the Wallace tree is that there are only reduction layers, and each layer has propagation delay. As making the partial products is and the final addition is the multiplication is only , not much slower than addition (however, much more expensive in the gate count). Naively adding partial products with regular adders would require time. From a complexity theoretic perspective, the Wallace tree algorithm puts multiplication in the class NC1.These computations only consider gate delays and don't deal with wire delays, which can also be very substantial. The Wallace tree can be also represented by a tree of 3/2 or 4/2 adders.

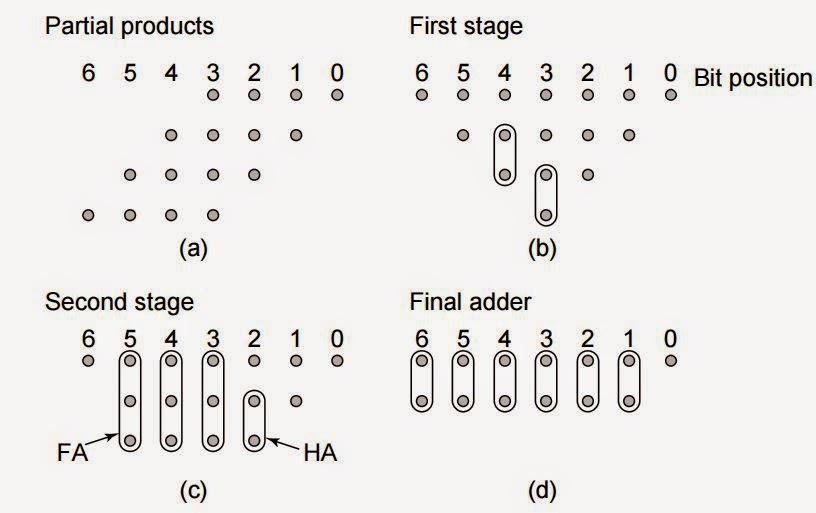
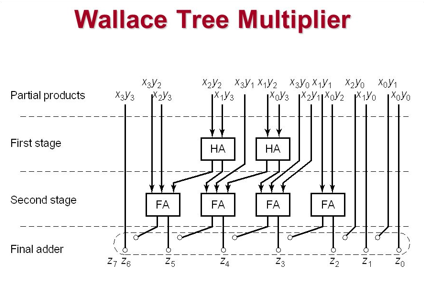


Fig.1 Structural description of 4X4 Wallace tree multiplier



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity wallace4 is

Port ( A : in STD\_LOGIC\_VECTOR (3 downto 0);

B : in STD\_LOGIC\_VECTOR (3 downto 0);

P : out STD\_LOGIC\_VECTOR (7 downto 0)); end wallace4;

architecture Behavioral of wallace4 is component full\_adder is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

sum : out STD\_LOGIC;

carry : out STD\_LOGIC); end component;

component half\_adder is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

sum : out STD\_LOGIC;

carry : out STD\_LOGIC); end component;

signal s11,s12,s13,s14,s15,s22,s23,s24,s25,s26,s32,s34,s35,s36,s37

: std\_logic;

signal c11,c12,c13,c14,c15,c22,c23,c24,c25,c26,c32,c34,c35,c36,c37

: std\_logic;

signal pp0,pp1,pp2,pp3 : std\_logic\_vector(3 downto 0);

begin

process(A,B)

begin

for i in 0 to 3 loop

pp0(i) <= A(i) and B(0);

pp1(i) <= A(i) and B(1);

pp2(i) <= A(i) and B(2);

pp3(i) <= A(i) and B(3);

end loop; end process;

P(0) <= pp0(0); P(1) <= s11; P(2) <= s22; P(3) <= s32; P(4) <= s34; P(5) <= s35; P(6) <= s36; P(7) <= s37;

--first stage

ha11 : half\_adder port map(pp0(1),pp1(0),s11,c11);

fa12 : full\_adder port map(pp0(2),pp1(1),pp2(0),s12,c12); fa13 : full\_adder port map(pp0(3),pp1(2),pp2(1),s13,c13); fa14 : full\_adder port map(pp1(3),pp2(2),pp3(1),s14,c14); ha15 : half\_adder port map(pp2(3),pp3(2),s15,c15);

--second stage

ha22 : half\_adder port map(c11,s12,s22,c22);

fa23 : full\_adder port map(pp3(0),c12,s13,s23,c23); fa24 : full\_adder port map(c13,c23,s14,s24,c24); fa25 : full\_adder port map(c14,c24,s15,s25,c25); fa26 : full\_adder port map(c15,c25,pp3(3),s26,c26);

--third stage

ha32 : half\_adder port map(c22,s23,s32,c32); ha34 : half\_adder port map(c32,s24,s34,c34); ha35 : half\_adder port map(c34,s25,s35,c35); ha36 : half\_adder port map(c35,s26,s36,c36); ha37 : half\_adder port map(c36,c26,s37,c37);

end Behavioral;

**--Half Adder Code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity half\_adder is Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

sum : out STD\_LOGIC;

carry : out STD\_LOGIC); end half\_adder;

architecture Behavioral of half\_adder is begin

sum <= a xor b; carry <= a and b;

end Behavioral;

**--Full Adder Code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity full\_adder is Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

sum : out STD\_LOGIC;

carry : out STD\_LOGIC); end full\_adder;

architecture Behavioral of full\_adder is begin

sum <= (a xor b xor c);

carry <= (a and b) xor (c and (a xor b)); end Behavioral;

Pre-lab questions

1. Discuss the pros and cons of Wallace tree multiplier
2. Write about Booth multiplier
3. Compare the performance of ripple carry and Carry look ahead adder

Post-lab questions

1. Write VHDL Code for 2 X 2 array multiplier.
2. Draw the structure of 4 X 4 array multiplier

Result: