# 4-Bit Wallace Multiplier

Submitted on 24/06/2021
authored by
Student name :- HARSH MISHRA (harsh@iiitmanipur.ac.in),
supervised by
Dr. Nagesh ch. (nagesh@iiitmanipur.ac.in)

***Index Terms*—**

***Abstract*—The aim of this experiment is to study 4x4 Wallace tree multiplier. In high performance processing units computing systems, multiplication of two binary numbers is primitive and most frequently used arithmetic operation. Wallace tree multiplier is area efficient high speed multiplier. through this i am presenting design and verification of Wallace tree multiplier. Design is carried out in xilinx vivado using Verilog hardware descriptive language and board name zynq-7000.**

## I. Introduction

As we know today is the era of power consumption,delay time and also the most important factor is area.more the area greater is the cost.so the major challanges for the chip designers are the area and speed.multipliers plays major roles in various fiels whether it is convolution,discrete fourior transform,fft, etc.
delays plays a major role in the building of a system.delays in circuit is directly related to delay in the multiplier and hence delay in ALU and then delay in microprocessor.
therefore the research is going on to counterpart this problem. so in order to reduce the delay here we have Wallace tree multiplier as one of the most efficient multiplier.
In recent years a lot of research is going on for the improvement of multiplication technique. Researchers are working on various multiplication techniques such as Vedic multiplication, Wallace tree multiplication technique and trying to propose a better algorithm. recently one of the malaysian university has introduced the vedic multiplication based microprocessor.

———FULL ADDER :—-:

following is the truth table and logic diagram of full adder.where F is input low or ’0’. and T is the output high or ’1’.
———HALF ADDER :—-:

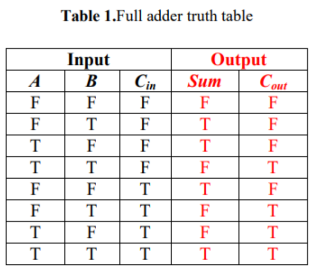


**Table 1.**Full adder truth table

| Input | | | Output | |
|---|---|---|---|---|
| *A* | *B* | $C_{in}$ | *Sum* | $C_{out}$ |
| F | F | F | F | F |
| F | T | F | T | F |
| T | F | F | T | F |
| T | T | F | F | T |
| F | F | T | T | F |
| F | T | T | F | T |
| T | F | T | F | T |
| T | T | T | T | T |

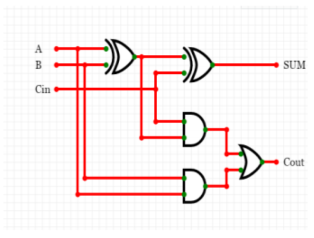


**Fig. 1**. Logic diagram of full adder

Fig. 1. BASIC APPROACH

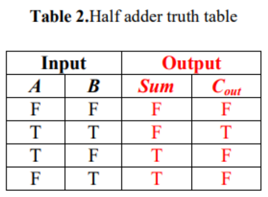


**Table 2.**Half adder truth table

| Input | | Output | |
|---|---|---|---|
| *A* | *B* | *Sum* | $C_{out}$ |
| F | F | F | F |
| T | T | F | T |
| T | F | T | F |
| F | T | T | F |

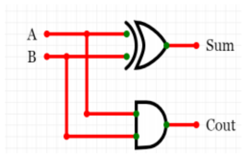


**Fig. 2**. Logic diagram of half adder

).pr

Fig. 2. BASIC APPROACH

following second one is the truth table and logic diagram of half adder.where F is input low or ’0’. and T is the output high or ’1’.

### *A. DESIGN AND IMPLEMENTATION OF WALLACE MULTIPLER*

Wallace tree multiplication algorithm is basically divided into three major steps-
1. Generation of partial products
2. Addition of partial products
3. Generation of final results from the partial products

The initial step is the formation of partial products by multiplying each bit from the multiplier to same bit position of multiplicand. Secondly, groups of three adjacent rows are collected. Each group of three rows is reduced by using half adders and full adders.
Half adders are used in each column where there are two bits whereas full adders are used in each column where

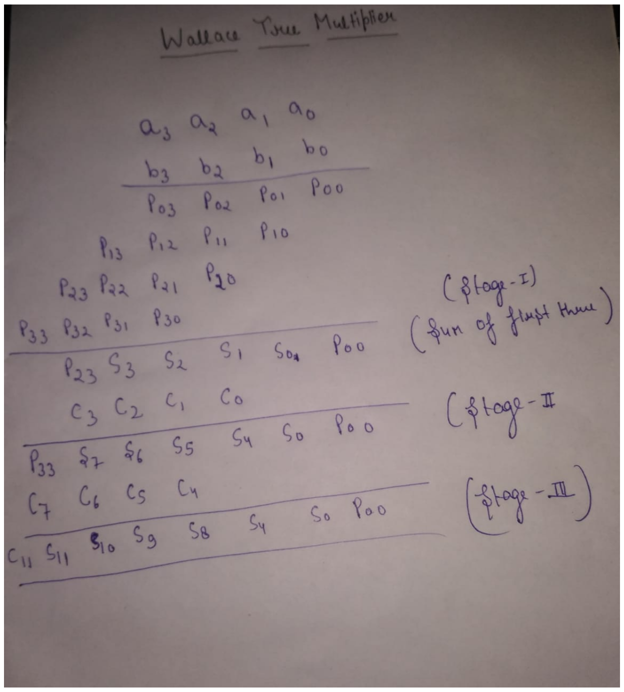


Fig. 3. BASIC APPROACH

there are three bits, any single bit in column is passed to next stage in the same column without any operation. This reduction procedure is repeated in each successive stage until only two rows remain. In the final stage, the remaining two rows are added. After completing all the three stages we get 8 bit of output as shown in figure.

——block diagram————

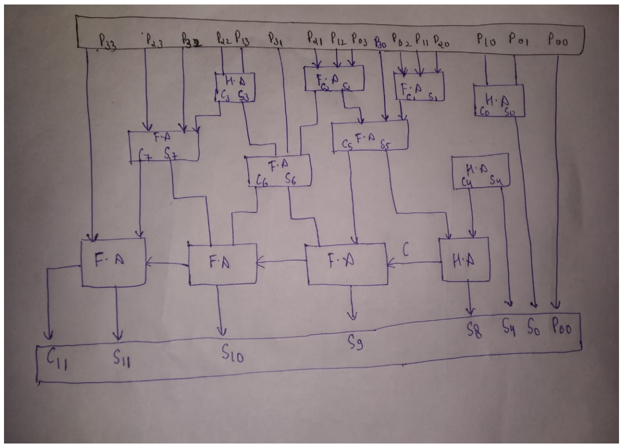

P33 P23 P32 P22 P13 P31 P21 P12 P03 P30 P02 P11 P20 P10 P01 P00
H·A C3 S3
F·A C2 S2
F·A C1 S1
H·A C0 S0
F·A C7 S7
C5 F·A S5
F·A C6 S6
H·A C4 S4
F·A
F·A
F·A
C
H·A
C11 S11 S10 S9 S8 S4 S0 P00

Fig. 4. BASIC APPROACH

## B. VERILOG CODE AND TEST BENCH

module half$_adder(sum, carry, a, b)$;
$inputa, b$;
$outputsum, carry$;
$xor(sum, a, b)$;
$and(carry, a, b)$;
$endmodule$
$modulefull_adder($
$outputS, Cout, inputX1, X2, Cin$
);
reg[1:0] temp;
always @(*)
begin
temp = 1'b0,X1 + 1'b0,X2+1'b0,Cin;
end
assign S = temp[0];
assign Cout = temp[1];
endmodule
module wallace$_multiplier(a, b, prod)$;
//inputs and outputs
input [3:0] a,b;
output [7:0] prod;
//internal variables.
wire s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11;
reg p[3:0][3:0];
integer i,j;
always@(a or b)
begin
for (i=0;i¡=3;i=i+1)
for(j=0;j¡=3;j=j+1)
p[j][i]¡=a[j]  b[i];
end
//final product assignments
assign prod[0] = p[0][0];
assign prod[1] = s0;
assign prod[2] = s4;
assign prod[3] = s8;
assign prod[4] = s9;
assign prod[5] = s10;
assign prod[6] = s11;
assign prod[7] = c11;
//first stage
half$_adderh1(s0, c0, p[0][1], p[1][0])$;
$full_adderf1(s1, c1, p[0][2], p[1][1], p[2][0])$;

$full_adderf2(s2, c2, p[0][3], p[1][2], p[2][1]);$
$full_adderf3(s3, c3, p[1][3], p[2][2], 1'b0);$

//second stage

full$_adderf4(s4, c4, s1, c0, 1'b0);$
$full_adderf5(s5, c5, s2, c1, p[3][0]);$
$full_adderf6(s6, c6, s3, c2, p[3][1]);$
$full_adderf7(s7, c7, p[2][3], c3, p[3][2]);$

//third stage

full$_adderf8(s8, c8, s5, c4, 1'b0);$
$full_adderf9(s9, c9, s6, c8, c5);$
$full_adderf10(s10, c10, s7, c6, c9);$
$full_adderf11(s11, c11, p[3][3], c7, c10);$

endmodule

——-TEST BENCH——–

module wallace$_multiplier_tb;$

// Inputs
reg [3:0] a;
reg [3:0] b;

// Outputs
wire [7:0] prod;
integer i,j,error;

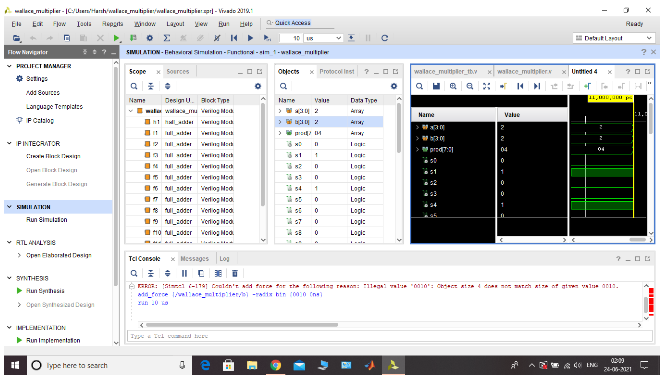
// Instantiate the Unit Under Test (UUT)
wallace uut (
.a(a),
.b(b),
.prod(prod)
);

initial begin
// Apply inputs for the whole range of A and B.
// 16*16 = 256 inputs.
error = 0;
for(i=0;i ¡=15;i = i+1)
for(j=0;j ¡=15;j = j+1)
begin
a ¡= i;
b ¡= j;
1;
if(prod != a*b) //if the result isnt correct
increment "error".
error = error + 1;
end
end

endmodule

## II. RESULTS

Here i have taken input of decimal value 2 by entering 0010 and second value of b is also 2 which is again 0010 and hence the results comes 0100 that is 4.



| Name | Value | Data Type |
|---|---|---|
| a[3:0] | 2 | Array |
| b[3:0] | 2 | Array |
| prod[7 | 04 | Array |
| s0 | 0 | Logic |
| s1 | 1 | Logic |
| s2 | 0 | Logic |
| s3 | 0 | Logic |
| s4 | 1 | Logic |
| s5 | 0 | Logic |
| s6 | 0 | Logic |
| s7 | 0 | Logic |
| s8 | 0 | Logic |

```
ERROR: [Simtcl 6-179] Couldn't add force for the following reason: Illegal value '0010': Object size 4 does not match size of given value 0010.
add_force {/wallace_multiplier/b} -radix bin {0010 0ns}
run 10 us
```

Fig. 5. result

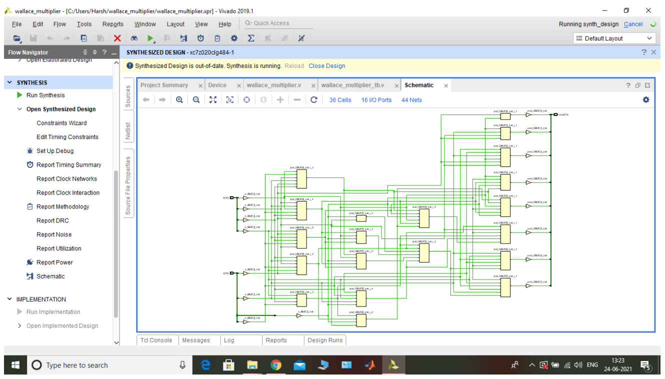


Fig. 6. synthesized circuit

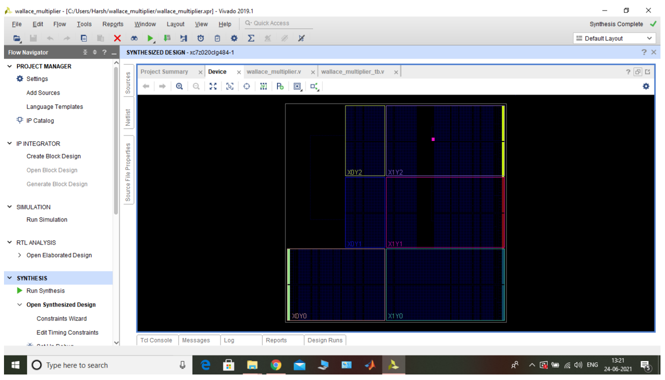


Fig. 7. synthesized diagram

### A. *conclusion*

after all performing the experiment and through some working prototype it was concluded that wallace multiplier is quite fast as compared to the array multiplier or simple multiplier. however it is easy to implement the wallace multiplier as compared other multiplier such as vedic multiplier.

### B. *acknowledgement*

I Would like to express my special thanks of gratitude to my Guide Dr. nagesh ch. who gave me the golden opportunity to do this wonderful experiment on the topic wallace multiplier which also helped me in doing a lot of research and I come to know about so many new things. I really thankful to them.

## III. References

https://core.ac.uk/download/pdf/234643291.pdf
https://en.wikipedia.org/wiki/Wallace$_tree$
$https://ieeexplore.ieee.org/document/6508192$
$http://www.cse.psu.edu/\ kxc104/class/cmpen411/14f/lec/C411L20Multiplier.pdf$
$https://www.ijert.org/design-ff-low-power-$
$multiplier-unit-using-wallace-tree-algorithm$