Politecnico di Torino

III Facoltà di Ingegneria

Area, Power and Timing Analysis for Booth-Encoding Multipliers

Integrated system technology

GROUP: 16

Giovanni Amedeo Cirillo
Alessandro Di Paola
Roberto Guerrieri
Ibrahim Shour

April 4, 2017

Contents

1 Introduction

The aim of this project is to create a model of a Modified Booth Encoding (MBE) Multiplier, a combinational implementation of a Booth multiplier. This approach permits to obtain a circuit that computes the product of two numbers in one clock cycle with all the advantages of the Booth multiplier, and in particular the reduction of the number of partial products with respect to a traditional binary multiplier (*e.g.* the one based on Baugh-Wooley algorithm).
In the Theoretical analysis it is discussed the general description of the multiplier above mentioned and then the calculation of Power Consumption, Occupied Area and Timing in function of the type of tree reduction algorithm (Wallace or Dadda) and the type of the adder employed (Ripple Carry Adder, Ladner-Fisher and Brent-Kung Parallel Prefix Adders). The calculations have been executed transforming the logic gates inside the structure in NAND gates. In the Octave implementation the code used to realize the models is shown . Moreover, two tables, which list the legend of the parameters, have been created in order to help the reader of this document. The last section concerns the results for each analysis indicated above. The used technological parameters refer to **Bulk Technology HP 2009 (predictions 2010)**.

2 Theoretical analysis

2.1 Description of Booth Multipier

Booth's encoding was first proposed for speeding up radix-2 multiplication in early digital computers. Booth observed that whenever there are sequences of consecutive 1s in the multiplier, multiplication can be speeded up by replacing the corresponding sequence of additions with a subtraction at the least-significant end and an addition in the position immediately to the left of its most-significant end. In other words

$$\sum_{k=i}^{j} 2^k = 2^{j+1} - 2^i$$

Booth encoding can be viewed as a digit-set conversion, thus the muliplier with digit set [0, 1] is transformed in a new number characterized by binary signed digits with digit set [-1, 1]. The operation is done scanning the number from right to left.

x_i	x_{i-1}	value	Explanation
0	0	0	No string of 1s
0	1	1	End of string of 1s
1	0	-1	Beginning of string of 1s
1	1	0	Continuation of string of 1s

Table 1: Radix-2 Booth's encoding.

An example of multiplication based on Booth's encoding is here reported.

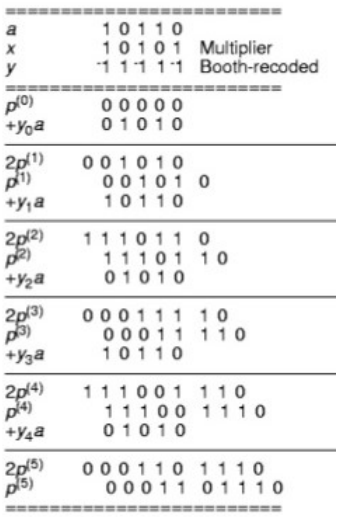


Figure 1: Example of multiplication based on Booth's encoding.

Radix-2 Booth encoding is not directly applied in modern arithmetic circuits and it is replaced by the radix-4 version of this encoding, also known as Modified Booth Encoding (MBE). According to the MBE algorithm [2] for radix-4 multiplication, the product of two numbers in 2's complement representation

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$
$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \quad (1)$$

can be written as

$$P = A \times B = \sum_{i=0}^{\frac{n}{2}-1} A \cdot m_i 2^{2i} = \sum_{i=0}^{\frac{n}{2}-1} A(-2b_{2i+1} + b_{2i} + b_{2i-1})2^{2i} \quad (2)$$

where $b_{-1} = 0$ and $m_i \in \{-2, -1, 0, 1, 2\}$. According to the encoded results from B, the Booth selectors choose $-2A$, $-A$, 0, A or $2A$ to generate the partial product rows. This approach permits to reduce by a factor $\frac{1}{2}$ the number of partial products, thus having significantly better performances in terms of latency.

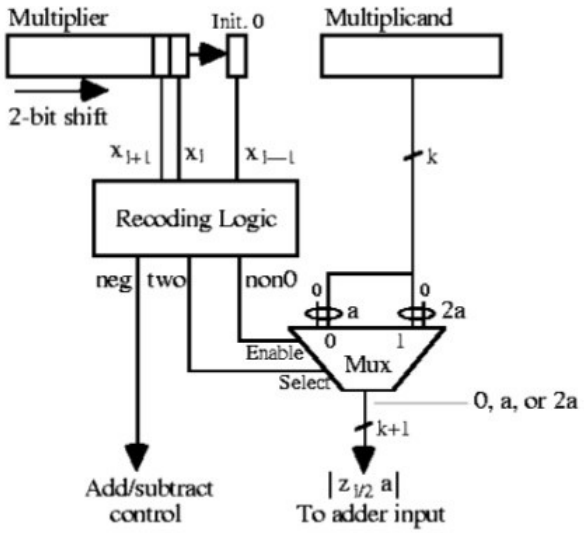


Figure 2: Scheme of a sequential MBE multiplier.

A combinational multiplier consists of three steps: partial product generation, partial product compression and final product generation. The first step is based on the MBE, in particular on its truth table present in [3].

x_{2i+1}	x_{2i}	x_{2i-1}	value	X1_b	X2_b	Neg	Z
0	0	0	0	1	0	0	1
0	0	1	1	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	2	1	0	0	0
1	0	0	-2	1	0	1	0
1	0	1	-1	0	1	1	0
1	1	0	-1	0	1	1	1
1	1	1	0	1	0	1	1

Table 2: Truth Table of MBE scheme.

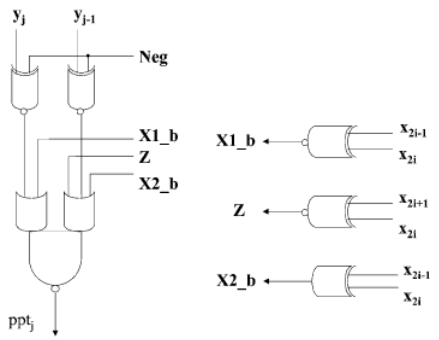


Figure 3: Decoder (left) and Encoder (right) logic circuits.

The only logic equation missing in the cited paper is that related to Neg_cin, *i.e.* the bit required to obtain negative partial products in 2's complement. Its characteristic equation is, according to [4]

$$Neg_{cin} = b_{2i+1} \cdot \overline{b_{2i} \cdot b_{2i-1}} \tag{3}$$

The circuit that generates a partial product (*i.e.* a row of the partial products array) is here reported.

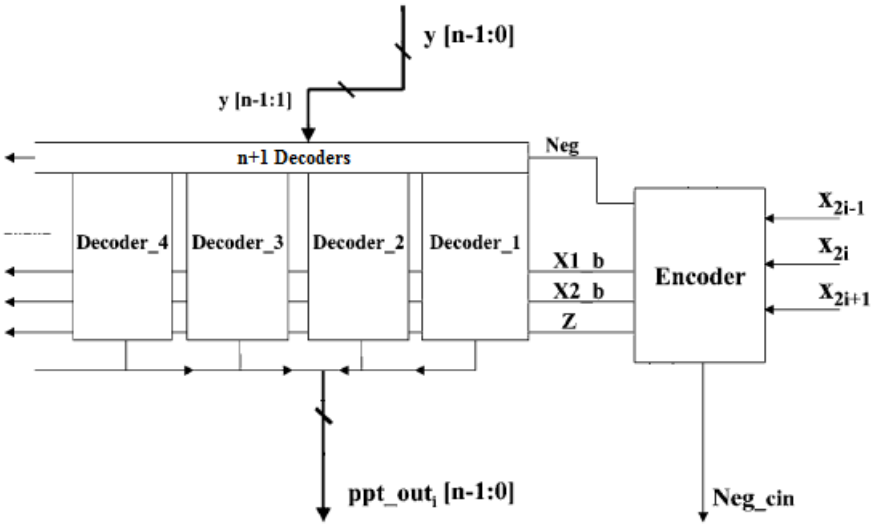


Figure 4: Partial products circuit.

The partial products for a 8×8 multiplication are organized in the following structure.

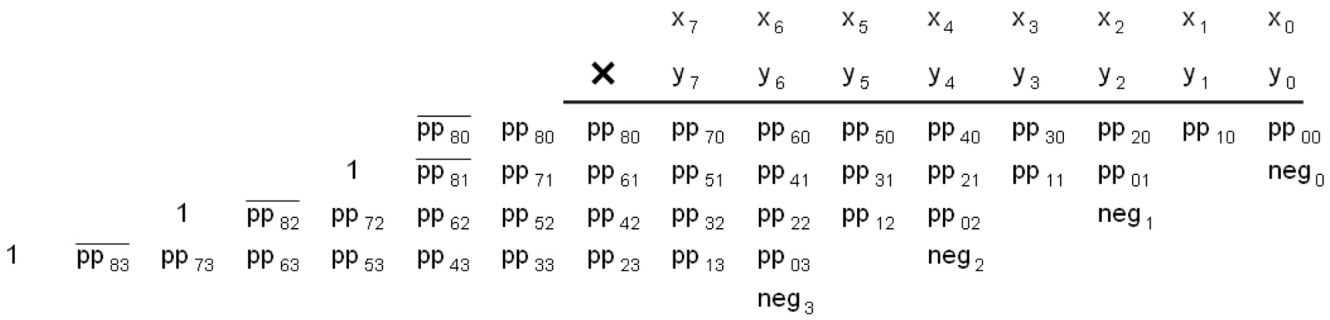
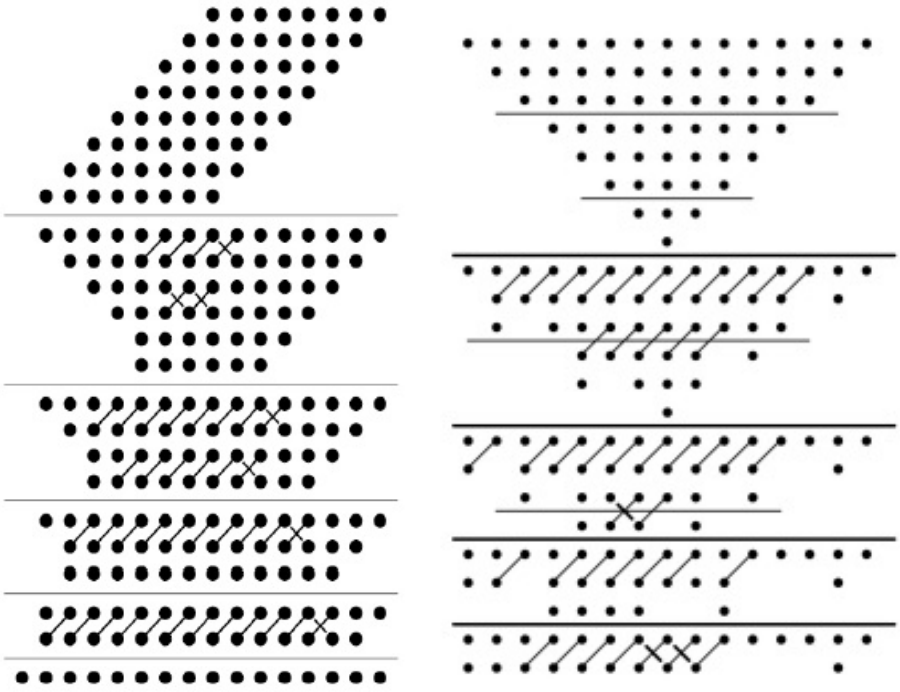


Figure 5: Array of partial products of a 8×8 multiplier.

In general the number of partial products will be $N_{PP} = \lfloor \frac{n+1}{2} \rfloor$ where n is the parallelism of the multiplier and the number of rows will be $N_{PP} + 1$ because of the *Neg* signal of the last partial product.
The partial products are reduced by exploiting Modified Wallace [7] and Dadda [6] trees, in order to have two operands of length $2 \cdot n$.



(a) Dadda (b) Modified Wallace

Figure 6: 8×8 reduction.

These two reduction methods employ the same total number of compressors. However the Dadda tree exploits more Half Adders than the Modified Wallace one.

The addition of the last two operands is done by three different adders: Ripple Carry Adder (RCA) and Parallel Prefix Adders (Ladner-Fisher and Brent-Kung) [1].

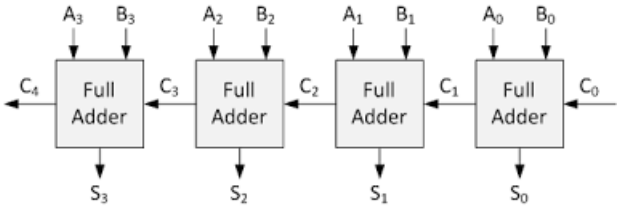


Figure 7: Ripple Carry Adder.

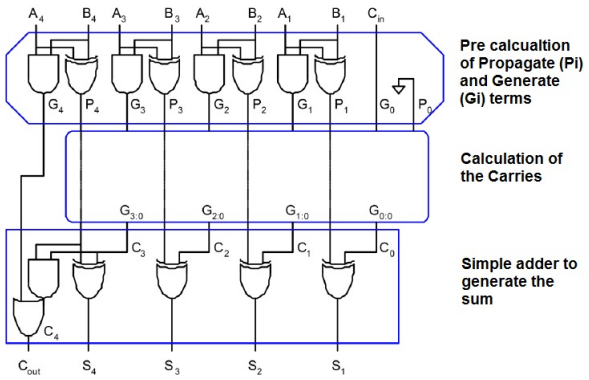


Figure 8: Parallel Prefix general structure.

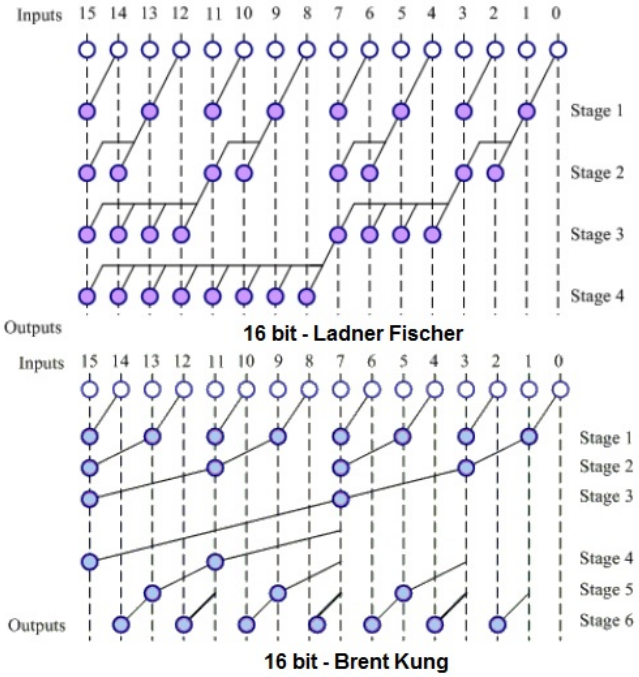


Figure 9: Ladner-Fisher and Brent-Kung PPAs.

2.2 Occupied Area

The area of a single transistor can be estimated by the product between width (W) and length (L). The length considered is the total one, it is calculated adding the effective length (between source and drain) and the length due to the overlapping of the gate in source and drain regions. For the project it was also taken into account the override factor to consider the contribution of interconnects.

$$area = (W \cdot L)\,(1 + interc_{override})\,\left[\mu m^2\right] \tag{4}$$

It was decided to convert each gate in **NAND2** logic to characterize the system. The following figure describes how a NAND2 is designed by using transistors with dimensions related to the minimum inverter width; considering the inverter, $W_n = 10 \cdot L$.

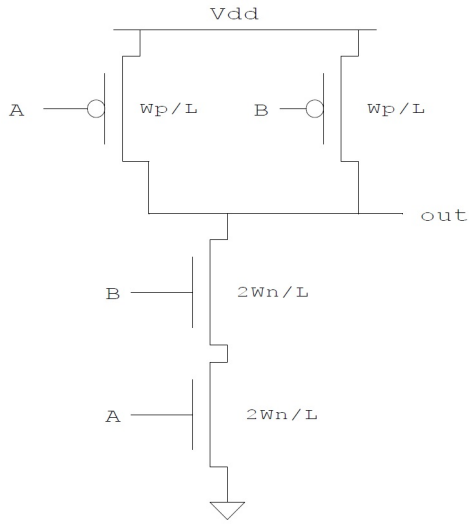


Figure 10: Transistors of NAND2.

It is possible to see that the number of NMOS is equal to the number of PMOS, so the area of a single NAND2 is:

$$A_{NAND} = 2 \cdot (W_p + 2 \cdot W_n) \cdot L\,(1 + interc_{override})\,\left[\mu m^2\right] \tag{5}$$

Where $W_p = \beta W_n$, and in this case $\beta = 1.29$. All the two-inputs logic gates have been converted into NAND2 gates, in this way it is possible to compute the total area by multiplying the area of a single NAND2 by the total number of gates. These conversions are here reported.

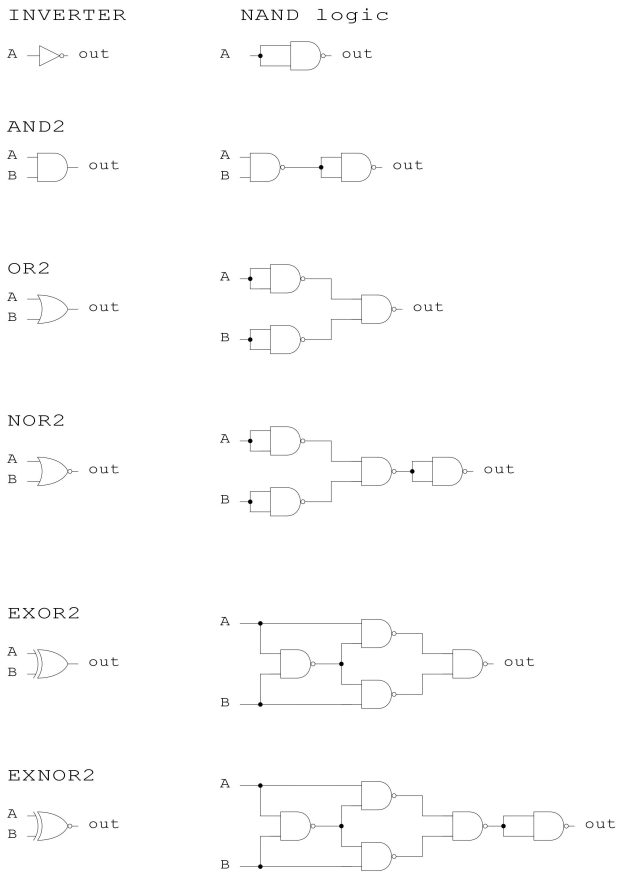


Figure 11: NAND construction of logic gates.

In the design of multipliers, Half Adders and Full Adders are employed. The NAND2 logic description of these components is here reported.

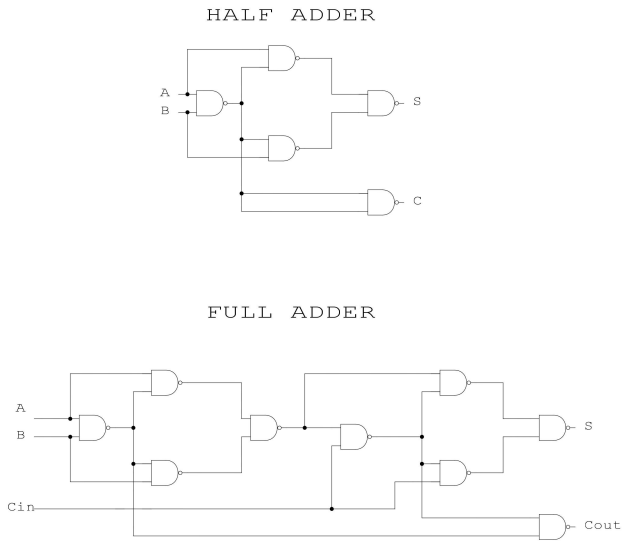


Figure 12: NAND construction of Half Adder and Full Adder.

The system described in this report is formed of three blocks:

- encoder and decoder
- tree
- final adder

The area of encoder and decoder is:

$$A_{enc} = A_{Neg} + 2 \cdot A_{XNOR} + A_{XOR} = 17 \cdot A_{NAND} \tag{6}$$

where $A_{Neg} = A_{AND} + A_{NAND}$.

$$A_{dec} = 2 \cdot A_{XNOR} + 2 \cdot A_{OR} + A_{NAND} = 17 \cdot A_{NAND} \tag{7}$$

The area of trees is related to the area of Full Adder and Half Adder, thus to the number of NAND gates used to synthesize these components.

$$A_{HA} = 5 \cdot A_{NAND} \tag{8}$$

$$A_{FA} = 9 \cdot A_{NAND} \tag{9}$$

$$N_{NAND_{tree_{wallace}}} = 9 \cdot (FA_{tree_{wallace}}) + 5 \cdot (HA_{tree_{wallace}}) \tag{10}$$

$$N_{NAND_{tree_{dadda}}} = 9 \cdot (FA_{tree_{dadda}}) + 5 \cdot (HA_{tree_{dadda}}) \tag{11}$$

Where the *FA* and *HA* is the number of Full Adders and Half Adders used in the tree; these two terms depend on the number of bits of the operands, so the area occupied considering Wallace and Dadda trees is respectively:

$$A_{tree} = A_{NAND} \cdot N_{NAND_{tree_{wallace}}} \tag{12}$$

$$A_{tree} = A_{NAND} \cdot N_{NAND_{tree_{dadda}}} \tag{13}$$

The last block to be considered is the adder. In this report three types of adder are compared:

- Ripple Carry Adder (RCA);
- Brent-Kung Parallel Prefix Adder;
- Ladner-Fisher Parallel Prefix Adder.

For the RCA version, it must be considered the total number of Full Adders:

$$FA_{RCA_{adder}} = 2 \cdot N_{bit} \tag{14}$$

so it is possible to calculate the number of NAND2 associated to the FA:

$$N_{NAND_{RCA_{adder}}} = 9 \cdot FA_{RCA_{adder}} \tag{15}$$

For the Parallel Prefix approach the computation of blocks in the prefix stage is required:

$$num_{precomputation} = 2 \cdot N_{bit} \cdot (4 + 2) \tag{16}$$

$$num_{b_{block\_PPA\_Brent}} = 2 \cdot 2 \cdot N_{bit} - 2 - log_2 (2 \cdot N_{bit}) \tag{17}$$

$$num_{b_{block\_PPA\_Ladner}} = 2 \cdot \frac{N_{bit}}{2} \cdot log_2 (2 \cdot N_{bit}) \tag{18}$$

$$num_{g_{block}} = 2 \cdot N_{bit} \tag{19}$$

$$num_{result} = 2 \cdot N_{bit} \cdot 4 + 5 \tag{20}$$

The last equation represents the generation of sum bits and the carry out. Now it is possible to calculate the number of NAND2 used, considering that each b (black) block has two AND2 and one OR2 gate (7 NAND2) and each g (grey) block has one AND2 and one OR2 gate (5 NAND2); for the definition of black and grey blocks, see [8]. So the number of NAND2 used in prefix stage of Brent-Kung and Ladner-Fisher approach is respectively:

$$N_{NAND_{PPA\_prefix\_stage\_Brent}} = 7 \cdot num_{b_{block\_PPA\_Brent}} + 5 \cdot num_{g_{block}} \tag{21}$$

$$N_{NAND_{PPA\_prefix\_stage\_Ladner}} = 7 \cdot num_{b_{block\_PPA\_Ladner}} + 5 \cdot num_{g_{block}} \tag{22}$$

The number of NAND2 gate in the Parallel Prefix approach is equal to the number of NAND2 in the pre-computation plus the number of NAND2 in the prefix stage plus the number of NAND2 in the final calculation (sum and carry bits):

$$N_{NAND_{PPA\_Brent}} = 2 \cdot (6 \cdot N_{bit}) + N_{NAND_{PPA\_prefix\_stage\_Brent}} + 4 \cdot (2 \cdot N_{bit}) + 3 + 2 \tag{23}$$

$$N_{NAND_{PPA\_Ladner}} = 2 \cdot (6 \cdot N_{bit}) + N_{NAND_{PPA\_prefix\_stage\_Ladner}} + 2 \cdot 4 \cdot N_{bit} + 3 + 2 \tag{24}$$

Therefore, in order to find the area contribution the Parallel Prefix Adders it is sufficient to multiply the number of NAND2 in the adder by the area of a NAND2.
The total area of the system is obtained by the sum of area contributions due to partial product generator, tree and final adder:

$$A_{total} = A_{PP} + A_{tree} + A_{adder} \tag{25}$$

However, the equation employed in the script to compute the total area is given by the product of the total number of NAND2 gates in the multiplier (so the sum of total number of gates in each stage) by the area of the NAND2.

2.3 Power consumption

The power consumption of each multiplier is equal to the sum of its dynamic and static power consumptions.

$$P_{tot} = P_{dyn} + P_{static}\ [W] \tag{26}$$

The dynamic power ([μW]), for a single gate, is obtained as:

$$P_{dyn} = \alpha \cdot V_{dd}^2 \cdot C \cdot f \tag{27}$$

where α is the switching activity factor, C is the capacitance of NAND2, V_{dd} is the supply voltage and f is the clock frequency. The load capacitance C is calculated considering a generic fan-out $n_{fan-out}$ and the junction capacitance between the drain of each MOSFET and the substrate. The analytical expressions of the capacitances are the following:

$$C_l = n_{fan-out} \cdot C_{in_{tot}}\ [\text{pF}] \tag{28}$$

where

$$C_{in_{tot}} = 2 \cdot (C_{in_{NMOS}} + C_{in_{PMOS}}) \tag{29}$$

$$C_{in_{NMOS}} = C_{ox} \cdot L_{eff} \cdot 2 \cdot W_n + 2 \cdot C_{overlap_n} \cdot 2 \cdot W_n \tag{30}$$

$$C_{in_{PMOS}} = C_{ox} \cdot L_{eff} \cdot W_p + 2 \cdot C_{overlap_p} \cdot W_p \tag{31}$$

where $C_{overlap}$ is the capacitance due to the overlapping of the gate oxide with source and drain regions (for this reason it is multiplied by 2.

$$C_{j_{ND2}} = 2 \cdot C_{j_{n_n}} + 2 \cdot C_{j_{n_p}}\ [\text{pF}] \tag{32}$$

where $C_{j_{n_n}} = C_{bottom_n} \cdot 2 \cdot W_n + C_{sidewall_n} \cdot perimeter_n$ and $C_{j_{n_p}} = C_{bottom_p} \cdot W_p + C_{sidewall_p} \cdot perimeter_p$. The final expression of the load capacitance C is given by

$$C = C_{ND2} = C_l + C_{j_{ND2}}\ [\text{fF}] \tag{33}$$

The static power ([μW]) is computed as

$$P_{static} = V_{dd} \cdot I_{static} \tag{34}$$

where I_{static} is the static current, obtained as the sum of two contributions:

- subthreshold leakage, that is the I_{off} current that flows in source node (connected to ground) of the n-MOSFET when the gate is connected to ground and the drain to supply voltage and of the p-MOSFET when the gate is connected to supply voltage and the drain to ground;
- gate leakage: is the gate current that flows from gate to channel when the gate is connected to V_{dd} while the drain and the source are connected to ground (n-MOSFET) or it is the current that flows in the opposite direction when the source and drain are connected to V_{dd} and the gate is connected to ground (p-MOSFET). There are two intermediate cases of gate current, when the drain and source are connected to different voltages and the gate is connected to ground (n-MOSFET) or to the supply voltage (p-MOSFET), *i.e.* in the MOSFET conditions of subthreshold leakage. However, these contributions are negligible with the respect to the subthreshold one, so they have not been taken into account in the considered model.

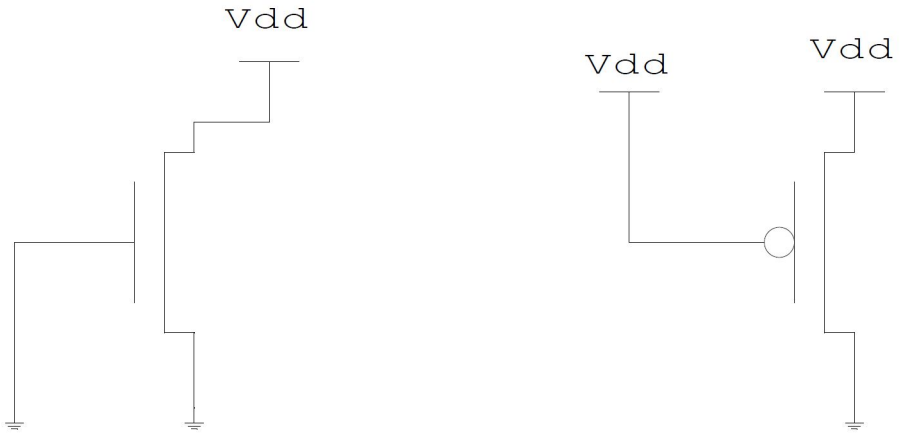


Figure 13: n-MOSFET and p-MOSET configurations that imply subthreshold current.

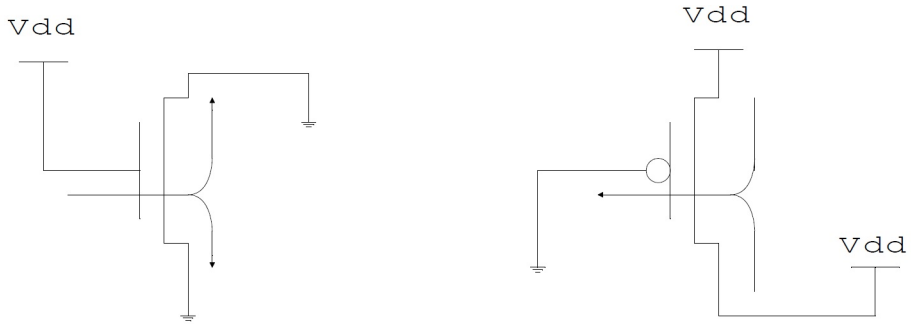


Figure 14: Significant contributions of gate current of n-MOSFET and p-MOSFET.

The final expression for I_{static} is determined considering that all the gates are transformed in NAND gates, as described before.
Taking into account that there are four possible input combinations of NAND2, the table that calculates the static current in all cases is the following:

A	**B**	**OUT**	**Static Current**
0	0	1	$I_{off_n} + 2 \cdot I_{gate_p}$
0	1	1	$2 \cdot I_{off_n} + I_{gate_p}$
1	0	1	$2 \cdot I_{off_n} + 2 \cdot I_{gate_n} + I_{gate_p}$
1	1	0	$2 \cdot I_{off_p} + 2 \cdot 2 \cdot I_{gate_n}$

Table 3: Static current for each combination of inputs of NAND2.

Assuming that the probability for each input combination is $\frac{1}{4}$, the current ([μA]) is

$$I_{static} = \frac{1}{4} \cdot \left[(I_{off_n} + 2 \cdot I_{gate_p}) + (2 \cdot I_{off_n} + I_{gate_p}) + (2 \cdot I_{off_n} + 2 \cdot I_{gate_n} + I_{gate_p}) + (2 \cdot I_{off_p} + 2 \cdot 2 \cdot I_{gate_n}) \right] \quad (35)$$

The static and dynamic contributions of MBE multiplier are defined as the sum of the contributions of each stage of the circuit.

$$P_{static_{mult}} = P_{static_{enc-dec}} + P_{static_{tree}} + P_{static_{adder}} = \left(N_{ND2_{enc-dec}} + N_{ND2_{tree}} + N_{ND2_{adder}} \right) \cdot P_{static_{ND2}} \text{ [W]} \quad (36)$$

$$P_{dyn_{mult}} = P_{dyn_{enc-dec}} + P_{dyn_{tree}} + P_{dyn_{adder}} = \left(N_{ND2_{enc-dec}} + N_{ND2_{tree}} + N_{ND2_{adder}}\right) \cdot P_{dyn_{ND2}} \text{ [W]} \tag{37}$$

For the estimation of the dynamic power of the fundamental logic gate, the switching activity factor α is supposed equal to $\frac{1}{2}$. This value is usually an overestimation of the typical value of α, so that the real power consumption is generally lower than that obtained.

2.4 Timing

According to the Roadmap, the delay of a NAND2 gate can be computed as

$$t_{NAND} = \frac{C_{ND2} \cdot V_{dd}}{I_{ND2}} \tag{38}$$

where C_{ND2} is the capacitance of the NAND2 gate, V_{dd} is the supply voltage and I_{ND2} is the gate current.
The delays of the each logic gate are computed in function of the that of NAND2.

$$t_{XOR} = 3 \cdot t_{NAND} \tag{39}$$

$$t_{XNOR} = 4 \cdot t_{NAND} \tag{40}$$

$$t_{OR} = 2 \cdot t_{NAND} \tag{41}$$

For the encoder circuit the delay of the Neg term is evaluated as

$$t_{Neg} = 3 \cdot t_{NAND} \tag{42}$$

So, taking into account the encoder circuit, the critical path is

$$t_{enc} = max(t_{Neg}, t_{XNOR}, t_{XOR}) = t_{XNOR} = 4 \cdot t_{NAND} \tag{43}$$

The total delay of decoder is equal to

$$t_{dec} = t_{XNOR} + t_{OR} + t_{NAND} = 7 \cdot t_{NAND} \tag{44}$$

Then, in order to evaluate the delay of the tree, the delay of Full Adder is evaluated. In order to make the calculation more precise a distinction has been made between the delays of sum and carry bits.

$$t_{FA_{SUM}} = 6 \cdot t_{NAND} \tag{45}$$

$$t_{FA_{CARRY}} = 5 \cdot t_{NAND} \tag{46}$$

Then the total delay of the tree depends on the height of the tree that is a function of the number of bits ($h[N_{bit}]$). For the Dadda tree it can be considered equal to

$$t_{tree_{DADDA}} = (h[N_{bit}] - 1) \cdot t_{FA_{CARRY}} + t_{FA_{SUM}} \tag{47}$$

This delay is generally studied from an optimistic point of view. However, since the worst-case scenario for adders delay has been taken into account, the tree delay compensates the pessimistic adder delay, thus obtaining a more realistic delay for the whole multiplier.
Then in order to roughly estimate the delay of the Wallace tree, the delay of the Dadda has been multiplied by a proportionality coefficient obtained as the ratio of the number of Full Adders in the two different tree configurations, since the delay (such as the number of Full Adders) of a Wallace tree is tipically slightly greater than that of a Dadda one.

$$t_{tree_{WALLACE}} = t_{tree_{DADDA}} \cdot delay_{ratio} \tag{48}$$

The last step is to evaluate the delay of the final adders in each configuration. For this calculation, it has to be considered that the adder works on $2 \cdot N_{bit}$.
The RCA delay is equal to

$$t_{RCA} = (2 \cdot N_{bit} - 1) \cdot t_{FA_{CARRY}} + t_{FA_{SUM}} \tag{49}$$

For the Parallel Prefix adders, the delay of the different internal blocks has to be evaluated.

$$t_{black_{block}} = 4 \cdot t_{NAND} \tag{50}$$

$$t_{grey_{block}} = t_{black_{block}} \tag{51}$$

Now it is possible to evaluate the delay of the Parallel Prefix adders.
For the Ladner-Fisher configuration the delay can be calculated as

$$t_{PPA_{Ladner}} = log_2(2 \cdot N_{bit}) \cdot t_{black\_block} + t_{grey\_block} \tag{52}$$

In the Brent-Kung configuration

$$t_{PPA_{Brent}} = (2 \cdot log_2(2 \cdot N_{bit}) - 2) \cdot t_{black\_block} + t_{grey\_block} + 2 \cdot t_{XOR} \tag{53}$$

In both Parallel Prefix Adders the last contribution is related to two XOR gates: the first one in pre-computation stage, the other one in the final stage, that computes the final result.
Finally it is possible to evaluate the total delay of the multiplier; in fact it can be obtained as the sum of the delays of the encoder, the decoder, the tree and the final adder

$$t_{Booth} = t_{enc} + t_{dec} + t_{tree} + t_{add} \tag{54}$$

Since three types of adders and two types of trees have been take into account, six different values for the delay have been obtained.

3 Octave implemetation

Below is reported the Matlab code and the tables of input and output parameters to have a simpler reading:

```
close all
clear all
clc

%% PARAMETER
%the default values of parameter are referred to the roadmap 2009 , year
% 2010
lambda =1; % referred to the design rules
L_eff =27*10^(-3); %um %the effective length of the gate
L_s_d = lambda*L_eff ; %the length of the source is equal to the length of the drain
L= L_eff +2*L_s_d ; %the total length
interc_override =0.15; % 15% of interconnetion override
alpha =0.5; % activity factor
vdd =0.97; %V
Wgate =0.27; %um width of NMOS W_n =10* L;
W_n = Wgate ; %um
beta =1.29;
W_p = beta * Wgate ; %um width of PMOS W_p =1.29* W_n ;

Cj0n =2.7* 10^( -3); %pf/um ^2 N- MOS junction capacitance per unit area under zero - bias conditions
Cj0p =3.3*10^( -3); %pF/um ^2 P- MOS junction capacitance per unit area under zero - bias conditions
Cjswn = 9.2*10^( -10); %F/m N-MOS sidewall junction capacitance
Cjswp =8*10^( -10); %F/m P-MOS sidewall junction capacitance
Cgd0n =1.35*10^( -10); %F/m N-MOS overlap capacitance between gate and drain per unit transistor width
Cgd0p =1*10^( -10); %F/m P-MOS overlap capacitance between gate and drain per unit transistor width
Mjn =0.38;
Mjp =0.45;
Mswn =0.22;
Mswp =0.265;
Pbn =0.85;
Pbp= 0.87;
Pbswn =0.67;
Pbswp =0.76;
k =10;
Cox =4.6041e-6; %pf/um ^2
n_fan_out =4; % Nand with two input is loaded with four identical inverter

% Source and drain diffusion length
lungh_diff =2.5* L_eff ; %um
% nMOS source and drain diffusion capacitance
Cbottom_n = lungh_diff * Cj0n *(1+ vdd /(2* Pbn ))^( - Mjn ); %pF/um

% nMOS source and drain sidewall capacitance
Csidewall_n =1e6* Cjswn *(1+ vdd /(2* Pbswn ))^( - Mswn ); %pF/um
% pMOS source and drain diffusion capacitance
Cbottom_p = lungh_diff * Cj0p *(1+ vdd /(2* Pbp ))^( - Mjp ); %pF/um

% pMOS source and drain sidewall capacitance
Csidewall_p =1e6* Cjswp *(1+ vdd /(2* Pbswp ))^( - Mswp ); %pF/um

%NAND2 capacitance
WD=k*L_eff ; %WD is W_n
perim_N =2* lungh_diff + 2*WD; %um (P = 2*Length_diffusion + Width)
perim_P=2* lungh_diff + beta*WD; %um (P = 2*Length_diffusion + Width)
Cjn= Cbottom_n *2*WD+ Csidewall_n * perim_N ; %pF
Cjp= Cbottom_p*WD+ Csidewall_p * perim_P ; %pF
Coverlap_n =1e6* Cgd0n ; %pF/um (initial value is in F/m)
Coverlap_p =1e6* Cgd0p ; %pF/um (initial value is in F/m)
```

```
Cin = 2*( Cox* L_eff *1e-3*2*W_n + Cox * L_eff *1e-3* W_p + Coverlap_n*2*W_n + Coverlap_p*W_p) ;% pF (2 nMOS & 2 pMOS)
Cout = 2*Cjn + 2*Cjp; %pF (2 nMOS & 2 pMOS)
Cl = Cin* n_fan_out ; %pF
cap = ( Cout +Cl )*10^(3);% fF % capacitance of NAND2

% static currents of inverter
ioff_n =1*10^( -1)*2*W_n; %uA
ioff_p =1*10^( -1)* W_p; %uA
igate_n =8.3*L_eff*2*W_n ; %uA
igate_p =8.3*L_eff* W_p ; %uA
i_NAND =1.2*2*W_n; %mA

%height of tree
h=[3 4 6 8 10];
delay_ratio = [1.27 1.07 1.03 1.01];
% delay ratios obtained by the following papers
% http://www.ijarcce.com/upload/2013/march/48-anju-HIGH%20PERFORMANCE%20DADDA%20MULTIPLIER%20IMPLEMENTATION%20USING%20HIGH%20SPEED%20CARRY%20SELECT%20ADDER.pdf
% http://www.enggjournals.com/ijet/docs/IJET13-05-02-306.pdf
FA_tree_wallace = [19 97 448 1917];
FA_tree_dadda = [15 91 435 1891];
HA_tree_wallace = [6 16 33 68];
HA_tree_dadda = [10 22 46 94];
% To simplify the calculation of timing , each logic gate is transformed in a NAND .
A_NAND = 2*( 2*W_n+W_p )*L *(1+ interc_override ) %um^2

% A_enc = A_Neg + 2*A_XNOR + A_XOR
A_enc=17*A_NAND; %um^2

% A_dec = 2*A_XNOR + 2*A_OR + A_NAND
A_dec=17*A_NAND; %um^2

% area of Full Adder
A_FA = 9*A_NAND; %um^2

% number of NAND gates involved for the FAs of tree (and corresponding total area)
N_NAND_tree_wallace = 9*(FA_tree_wallace)+5*(HA_tree_wallace);
N_NAND_tree_dadda = 9*(FA_tree_dadda)+5*(HA_tree_dadda);

% delay of NAND gate
t_NAND = ( cap* vdd )/ i_NAND ; %ps

% delay of gates involved in decoder and encoder
t_XOR = 3*t_NAND;
t_XNOR = 4*t_NAND;
t_OR = 2*t_NAND;

t_dec = t_XNOR + t_OR + t_NAND;

t_Neg = 3*t_NAND;
%t_enc = max(t_Neg, t_XNOR, t_XOR)
t_enc = t_XNOR;

%delay of FA
t_FA_s = 6*t_NAND;
t_FA_c = 5*t_NAND;

%delay of black and grey blocks of PPA (AND + OR)
t_black_block = 4*t_NAND;
t_grey_block = t_black_block;
```

```
%% BODY OF THE MODEL
for var =1:4
N_bit =2^( var+2)

% N_rows is the number of rows of PARTIAL PRODUCTS
N_rows =floor( (N_bit + 1)/2 );

%N_columns is the number of columns of PARTIAL PRODUCTS
N_columns=N_bit+1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COMPUTATION OF AREA IN FUNCTION OF NUMBER OF TRANSISTORS FOR PARTIAL PRODUCTS STAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% A_rows is the area of partial products
%N_NAND(var) is the number of NAND gates in the enc-dec stage
A_PP(var)=0;
N_NAND(var)=0;
for k=1:N_rows
    if k~=1
        A_rows(k)=N_columns*A_dec + A_enc;
        N_NAND(var)=N_NAND(var)+ (1 + N_columns)*17;
    else
        A_rows(k)=N_columns*A_dec + A_enc + A_NAND;
        N_NAND(var)=N_NAND(var) + (1 + N_columns)*17 +1;
    end
    A_PP(var) = A_PP(var) + A_rows(k);
end

% total number of NAND gates = NAND gates of enc/dec + NAND gates of FAs
% involved in tree and RCA and ppa
N_NAND_enc_dec = N_NAND(var)
N_NAND_wallace = N_NAND_tree_wallace(var)
N_NAND_dadda = N_NAND_tree_dadda(var)
%for ppa we need the computation of block in the prefix stage
num_precomputation = 2*N_bit*(4+2);
num_b_block_PPA_Brent = 2*2*N_bit -2 -log2(2*N_bit);
num_b_block_PPA_Ladner = 2*N_bit/2 * log2(2*N_bit);
num_g_block = 2*N_bit;
num_result = 2*N_bit*4+5;
%generation of sum bits + generation of carry_out
%each b block has two AND and one OR gate (7 NAND)
%each g block has one AND and one OR gate (5 NAND)
N_NAND_PPA_prefix_stage_Brent = 7*num_b_block_PPA_Brent + 5*num_g_block;
N_NAND_PPA_prefix_stage_Ladner = 7*num_b_block_PPA_Ladner + 5*num_g_block;
FA_RCA_adder = 2*N_bit;
N_NAND_RCA_adder = 9*(FA_RCA_adder)
%number of NAND gate in ppa adder = number of NAND in pre-coputation
%(12*N_bit) + the number of NAND in the prefix stage + number of NAND in
%the final calculation (8*N_bit for the XORs and 3+2 for AND & OR of carry out)
N_NAND_PPA_Brent = 2*(6*N_bit)+N_NAND_PPA_prefix_stage_Brent + 4*(2*N_bit) + 3+2
N_NAND_PPA_Ladner = 2*(6*N_bit)+N_NAND_PPA_prefix_stage_Ladner + 2*4*N_bit + 3+2

%with final RCA adder
N_NAND_wallace_RCA(var)=N_NAND(var) + N_NAND_tree_wallace(var) + N_NAND_RCA_adder;
N_NAND_dadda_RCA(var)=N_NAND(var) + N_NAND_tree_dadda(var) + N_NAND_RCA_adder;
%with final PPA Brent-Kung adder
N_NAND_wallace_PPA_Brent(var)=N_NAND(var) + N_NAND_tree_wallace(var) + N_NAND_PPA_Brent;
N_NAND_dadda_PPA_Brent(var)=N_NAND(var) + N_NAND_tree_dadda(var) + N_NAND_PPA_Brent;
%with final PPA Brent-Kung adder
N_NAND_wallace_PPA_Ladner(var)=N_NAND(var) + N_NAND_tree_wallace(var) + N_NAND_PPA_Ladner;
N_NAND_dadda_PPA_Ladner(var)=N_NAND(var) + N_NAND_tree_dadda(var) + N_NAND_PPA_Ladner;
```

```
%  total area = A_PP + A_TREE + A_RCA
%  A_TOT_wallace_RCA(var) = A_PP(var) + (FA_tree_wallace(var)+2*N_bit)*A_FA;
%  A_TOT_dadda_RCA(var) = A_PP(var) + (FA_tree_dadda(var)+2*N_bit)*A_FA;
  A_enc_dec = A_rows; %um^2
  A_TOT_wallace_RCA(var) = N_NAND_wallace_RCA(var)*A_NAND; %um^2
  A_TOT_dadda_RCA(var) = N_NAND_dadda_RCA(var)*A_NAND; %um^2
 %total area = A_PP + A_TREE +A_PPA
 A_TOT_wallace_PPA_Brent(var) = N_NAND_wallace_PPA_Brent(var)*A_NAND; %um^2
 A_TOT_dadda_PPA_Brent(var) = N_NAND_dadda_PPA_Brent(var)*A_NAND; %um^2
 A_TOT_wallace_PPA_Ladner(var) = N_NAND_wallace_PPA_Ladner(var)*A_NAND; %um^2
 A_TOT_dadda_PPA_Ladner(var) = N_NAND_dadda_PPA_Ladner(var)*A_NAND; %um^2

 %%
 %%COMPUTATION OF DELAY

 %delay of tree
 t_tree_dadda = (h(var)-1)*t_FA_c + t_FA_s;
 t_tree_wallace = delay_ratio(var)*t_tree_dadda;
 % Adders work on 2*N_bit bits!
 %delay of RCA
 t_RCA = (2*N_bit -1)*t_FA_c + t_FA_s
 %delay of Brent-Kung
 t_PPA_Brent = ((2*log2(2*N_bit)-2)*t_black_block)+t_grey_block+t_XOR*2 %t_XOR*2 is due to precomputation
     and final computation
 %delay of Ladner-Fisher
 t_PPA_Ladner = (log2(2*N_bit))*t_black_block + t_grey_block+t_XOR*2
 %the additional t_NAND is related to the inverter of the MSB of first row
 t_cp_PP1 = max ([t_Neg+t_dec , t_XNOR+t_OR+t_NAND, t_XOR+t_OR+t_NAND]); %ps

 %critical path (approximated to total delay)
 t_cp_dadda_RCA(var) = (t_cp_PP1 + t_tree_dadda + t_RCA)*10^-3; %ns
 t_cp_wallace_RCA(var) = (t_cp_PP1 + t_tree_wallace + t_RCA)*10^-3; %ns
 t_cp_dadda_PPA_Brent(var) = (t_cp_PP1 + t_tree_dadda + t_PPA_Brent)*10^-3; %ns
 t_cp_wallace_PPA_Brent(var) = (t_cp_PP1 + t_tree_wallace + t_PPA_Brent)*10^-3; %ns
 t_cp_dadda_PPA_Ladner(var) = (t_cp_PP1 + t_tree_dadda + t_PPA_Ladner)*10^-3; %ns
 t_cp_wallace_PPA_Ladner(var) = (t_cp_PP1 + t_tree_wallace + t_PPA_Ladner)*10^-3; %ns
 freq_dadda_RCA(var) = (1/t_cp_dadda_RCA(var)); %GHz
 freq_wallace_RCA(var) = (1/t_cp_wallace_RCA(var)); %GHz
 freq_dadda_PPA_Brent(var) = (1/t_cp_dadda_PPA_Brent(var)); %GHz
 freq_wallace_PPA_Brent(var) = (1/t_cp_wallace_PPA_Brent(var)); %GHz
 freq_dadda_PPA_Ladner(var) = (1/t_cp_dadda_PPA_Ladner(var)); %GHz
 freq_wallace_PPA_Ladner(var) = (1/t_cp_wallace_PPA_Ladner(var)); %GHz

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % COMPUTATION FOR POWER CONSUMPTION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% %the total power is the sum of the static power and the dynamic power .
% %the dynamic power is directly proportional to the voltage supply to the
% %square , to the frequency ,to the capacitance and to the activity parameter
%
 p_dynamic_NAND_dadda_RCA = alpha * vdd ^2 * cap* freq_dadda_RCA(var); %uW
 p_dynamic_NAND_dadda_PPA_Ladner = alpha * vdd ^2 * cap* freq_dadda_PPA_Ladner(var); %uW
 p_dynamic_NAND_dadda_PPA_Brent = alpha * vdd ^2 * cap* freq_dadda_PPA_Brent(var); %uW
 p_dynamic_NAND_wallace_RCA = alpha * vdd ^2 * cap* freq_wallace_RCA(var); %uW
 p_dynamic_NAND_wallace_PPA_Ladner = alpha * vdd ^2 * cap* freq_wallace_PPA_Ladner(var); %uW
 p_dynamic_NAND_wallace_PPA_Brent = alpha * vdd ^2 * cap* freq_wallace_PPA_Brent(var); %uW

 p_dynamic_wallace_RCA(var) = N_NAND_wallace_RCA(var)* p_dynamic_NAND_wallace_RCA; %uW
 p_dynamic_dadda_RCA(var) = N_NAND_dadda_RCA(var)* p_dynamic_NAND_dadda_RCA; %uW
 p_dynamic_wallace_PPA_Brent(var) = N_NAND_wallace_PPA_Brent(var)* p_dynamic_NAND_wallace_PPA_Brent; %uW
```

```
 p_dynamic_dadda_PPA_Brent(var) = N_NAND_dadda_PPA_Brent(var)* p_dynamic_NAND_dadda_PPA_Brent; %uW
 p_dynamic_wallace_PPA_Ladner(var) = N_NAND_wallace_PPA_Ladner(var)* p_dynamic_NAND_wallace_PPA_Ladner; %uW
 p_dynamic_dadda_PPA_Ladner(var) = N_NAND_dadda_PPA_Ladner(var)* p_dynamic_NAND_dadda_PPA_Ladner; %uW

% The static power is given by two contributions : the subthreshold
% leakage ( Ioff ) and the gate leakage .
% Assuming that the probability associated to the gate NAND is 25% for each
%case , the static current is given by:
 a =1/4;
 i_static =a*(( ioff_n +2* igate_p )+(2*ioff_n + igate_p )+(2*ioff_n + 2*igate_n + igate_p )+(2* ioff_p +2*2*igate_n )); %uA
 p_static_wallace_RCA(var) =N_NAND_wallace_RCA(var)*vdd* i_static ; %uW
 p_static_dadda_RCA(var) =N_NAND_dadda_RCA(var)*vdd* i_static ; %uW
 p_static_wallace_PPA_Brent(var) =N_NAND_wallace_PPA_Brent(var)*vdd* i_static ; %uW
 p_static_dadda_PPA_Brent(var) =N_NAND_dadda_PPA_Brent(var)*vdd* i_static ; %uW
 p_static_wallace_PPA_Ladner(var) =N_NAND_wallace_PPA_Ladner(var)*vdd* i_static ; %uW
 p_static_dadda_PPA_Ladner(var) =N_NAND_dadda_PPA_Ladner(var)*vdd* i_static ; %uW

% %the total power of a single nand gate is
% %so the total power of the architecture will be
 p_tot_wallace_RCA(var) = (p_dynamic_wallace_RCA(var) + p_static_wallace_RCA(var)) *10^( -6); %W
 p_tot_dadda_RCA(var) = (p_dynamic_dadda_RCA(var) + p_static_dadda_RCA(var)) *10^( -6); %W
 p_tot_wallace_PPA_Brent(var) = (p_dynamic_wallace_PPA_Brent(var) + p_static_wallace_PPA_Brent(var)) *10^( -6); %W
 p_tot_dadda_PPA_Brent(var) = (p_dynamic_dadda_PPA_Brent(var) + p_static_dadda_PPA_Brent(var)) *10^( -6); %W
 p_tot_wallace_PPA_Ladner(var) = (p_dynamic_wallace_PPA_Ladner(var) + p_static_wallace_PPA_Ladner(var)) *10^( -6); %W
 p_tot_dadda_PPA_Ladner(var) = (p_dynamic_dadda_PPA_Ladner(var) + p_static_dadda_PPA_Ladner(var)) *10^( -6); %W
 N_bit_vector (var)= N_bit ;

 freq = [freq_wallace_RCA(var),freq_dadda_RCA(var),freq_wallace_PPA_Brent(var),freq_dadda_PPA_Brent(var), freq_wallace_PPA_Ladner(var),freq_dadda_PPA_Ladner(var)]
 freq_min(var) = min(freq)
 p_dynamic_NAND_fmin(var) = alpha * vdd ^2 * cap* freq_min(var); %uW
 p_dynamic_wallace_RCA_fmin(var) = N_NAND_wallace_RCA(var)* p_dynamic_NAND_fmin(var); %uW
 p_dynamic_dadda_RCA_fmin(var) = N_NAND_dadda_RCA(var)* p_dynamic_NAND_fmin(var); %uW
 p_dynamic_wallace_PPA_Brent_fmin(var) = N_NAND_wallace_PPA_Brent(var)* p_dynamic_NAND_fmin(var); %uW
 p_dynamic_dadda_PPA_Brent_fmin(var) = N_NAND_dadda_PPA_Brent(var)* p_dynamic_NAND_fmin(var); %uW
 p_dynamic_wallace_PPA_Ladner_fmin(var) = N_NAND_wallace_PPA_Ladner(var)* p_dynamic_NAND_fmin(var); %uW
 p_dynamic_dadda_PPA_Ladner_fmin(var) = N_NAND_dadda_PPA_Ladner(var)* p_dynamic_NAND_fmin(var); %uW
end;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % COLLECTING RESULTS
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% NUMERICAL VISUALIZATION
% show the numerical result in the matlab shell in terms of vectors

N_bit_vector

%number of NAND gates
N_NAND_wallace_RCA
N_NAND_dadda_RCA
N_NAND_wallace_PPA_Brent
N_NAND_dadda_PPA_Brent
N_NAND_wallace_PPA_Ladner
N_NAND_dadda_PPA_Ladner
```

```
297 %Area [um^2]
    A_TOT_wallace_RCA
299 A_TOT_dadda_RCA
    A_TOT_wallace_PPA_Brent
301 A_TOT_dadda_PPA_Brent
    A_TOT_wallace_PPA_Ladner
303 A_TOT_dadda_PPA_Ladner

305 %delay [ns]
    t_cp_wallace_RCA
307 t_cp_dadda_RCA
    t_cp_wallace_PPA_Brent
309 t_cp_dadda_PPA_Brent
    t_cp_wallace_PPA_Ladner
311 t_cp_dadda_PPA_Ladner

313 %frequency [GHz]
    freq_wallace_RCA
315 freq_dadda_RCA
    freq_wallace_PPA_Brent
317 freq_dadda_PPA_Brent
    freq_wallace_PPA_Ladner
319 freq_dadda_PPA_Ladner

321 %dynamic power [uW]
    p_dynamic_wallace_RCA
323 p_dynamic_dadda_RCA
    p_dynamic_wallace_PPA_Brent
325 p_dynamic_dadda_PPA_Brent
    p_dynamic_wallace_PPA_Ladner
327 p_dynamic_dadda_PPA_Ladner

329 %static power [uW]
    p_static_wallace_RCA
331 p_static_dadda_RCA
    p_static_wallace_PPA_Brent
333 p_static_dadda_PPA_Brent
    p_static_wallace_PPA_Ladner
335 p_static_dadda_PPA_Ladner

337 %total power [W]
    p_tot_wallace_RCA
339 p_tot_dadda_RCA
    p_tot_wallace_PPA_Brent
341 p_tot_dadda_PPA_Brent
    p_tot_wallace_PPA_Ladner
343 p_tot_dadda_PPA_Ladner

345 freq_min %GHz

347 %dynamic power with fixed frequency [uW]
    p_dynamic_NAND_fmin
349 p_dynamic_wallace_RCA_fmin
    p_dynamic_dadda_RCA_fmin
351 p_dynamic_wallace_PPA_Brent_fmin
    p_dynamic_dadda_PPA_Brent_fmin
353 p_dynamic_wallace_PPA_Ladner_fmin
    p_dynamic_dadda_PPA_Ladner_fmin
355
    %% GRAPHICAL VISUALIZATION
357 % plot area
     figure (1)
```

```
359 plot((N_bit_vector), A_TOT_wallace_RCA, 'b*:', N_bit_vector, A_TOT_dadda_RCA , 'ro--', N_bit_vector,
      A_TOT_wallace_PPA_Brent , 'go-', N_bit_vector, A_TOT_dadda_PPA_Brent , 'v-', N_bit_vector,
      A_TOT_wallace_PPA_Ladner , 'ko-', N_bit_vector, A_TOT_dadda_PPA_Ladner , 'm*-');
    grid on
361 legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
      Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northwest')
    title (' Total Area')
363 xticks([8 16 32 64 128])
    xlabel ('Number of bits ')
365 ylabel (' Area [\mum^2] ')

367 % plot dynamic power consumption
    figure (2)
369 plot ( (N_bit_vector) , p_dynamic_wallace_RCA*10^-3 ,'b*:',(N_bit_vector) , p_dynamic_dadda_RCA*10^-3 ,'
      ro--',(N_bit_vector) , p_dynamic_wallace_PPA_Brent*10^-3 ,'go-',(N_bit_vector) ,
      p_dynamic_dadda_PPA_Brent*10^-3 ,'v-',(N_bit_vector) , p_dynamic_wallace_PPA_Ladner*10^-3 ,'ko-',(
      N_bit_vector) , p_dynamic_dadda_PPA_Ladner*10^-3 ,'m*-');
    grid on
371 title (' Dynamic Power Consumption')
    legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
      Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northwest')
373 xticks([8 16 32 64 128])
    xlabel (' Number of bits ')
375 ylabel (' Power [mW] ')

377 % plot static power consumption
    figure (3)
379 plot ( (N_bit_vector) , p_static_wallace_RCA*10^-3 ,'b*:',(N_bit_vector) , p_static_dadda_RCA*10^-3 ,'ro
      --',(N_bit_vector) , p_static_wallace_PPA_Brent*10^-3 ,'go-',(N_bit_vector) , p_static_dadda_PPA_Brent
      *10^-3 ,'v-',(N_bit_vector) , p_static_wallace_PPA_Ladner*10^-3 ,'ko-',(N_bit_vector) ,
      p_static_dadda_PPA_Ladner*10^-3 ,'m*-');
    grid on
381 title (' Static Power Consumption')
    legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
      Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northwest')
383 xticks([8 16 32 64 128])
    xlabel (' Number of bits ')
385 ylabel (' Power [mW] ')

387 % plot total power
    figure (4)
389 plot ( (N_bit_vector) , p_tot_wallace_RCA*1000 ,'b*:',(N_bit_vector) , p_tot_dadda_RCA*1000 ,'ro--',(
      N_bit_vector) , p_tot_wallace_PPA_Brent*1000 ,'go-',(N_bit_vector) , p_tot_dadda_PPA_Brent*1000 ,'v-'
      ,(N_bit_vector) , p_tot_wallace_PPA_Ladner*1000 ,'ko-',(N_bit_vector) , p_tot_dadda_PPA_Ladner*1000 ,'
      m*-')
    grid on
391 xticks([8 16 32 64 128])
    title (' Total Power Consumption')
393 legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
      Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northwest')
    xticks([8 16 32 64 128])
395 xlabel (' Number of bits ')
    ylabel (' Power [mW] ')
397
    % plot delay
399 figure (5)
    plot ( (N_bit_vector) , t_cp_wallace_RCA ,'b*:',(N_bit_vector) , t_cp_dadda_RCA ,'ro--',(N_bit_vector) ,
       t_cp_wallace_PPA_Brent ,'go-',(N_bit_vector) , t_cp_dadda_PPA_Brent ,'v-',N_bit_vector,
      t_cp_wallace_PPA_Ladner,'ko-',N_bit_vector, t_cp_dadda_PPA_Ladner,'m*-');
401 grid on
    xticks([8 16 32 64 128])
403 legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
```

```
    Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northwest')
 title (' Critical path ')
 xlabel (' Number of bits ')
 ylabel (' Delay [ns]')

% plot frequency
 figure (6)
 plot ( (N_bit_vector) , freq_wallace_RCA ,'b*:',(N_bit_vector) , freq_dadda_RCA ,'ro--',(N_bit_vector) ,
     freq_wallace_PPA_Brent ,'go-',(N_bit_vector) , freq_dadda_PPA_Brent ,'v-',N_bit_vector ,
    freq_wallace_PPA_Ladner ,'ko-',N_bit_vector , freq_dadda_PPA_Ladner ,'m*-');
 grid on
 xticks([8 16 32 64 128])
 legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
    Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northeast')
 title (' Maximum clock frequency ')
 xlabel (' Number of bits ')
 ylabel (' Frequency [GHz]')

% plot dynamic power consumption with fixed frequency
 figure (7)
 plot ( (N_bit_vector) , p_dynamic_wallace_RCA_fmin*10^-3 ,'b*:',(N_bit_vector) ,
    p_dynamic_dadda_RCA_fmin*10^-3 ,'ro--',(N_bit_vector) , p_dynamic_wallace_PPA_Brent_fmin*10^-3 ,'go-'
    ,(N_bit_vector) , p_dynamic_dadda_PPA_Brent_fmin*10^-3 ,'v-',(N_bit_vector) ,
    p_dynamic_wallace_PPA_Ladner_fmin*10^-3 ,'ko-',(N_bit_vector) , p_dynamic_dadda_PPA_Ladner_fmin*10^-3
    ,'m*-');
 grid on
 title (' Dynamic Power Consumption at fixed frequency')
 legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
    Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northwest')
 xticks([8 16 32 64 128])
 xlabel (' Number of bits ')
 ylabel (' Power [mW] ')

% plot dynamic power consumption with fixed frequency
 figure (8)
 plot ( (N_bit_vector) , (p_static_wallace_RCA + p_dynamic_wallace_RCA_fmin)*10^-3 ,'b*:',(N_bit_vector)
    ,(p_static_dadda_RCA + p_dynamic_dadda_RCA_fmin)*10^-3 ,'ro--',(N_bit_vector) , (
    p_static_wallace_PPA_Brent + p_dynamic_wallace_PPA_Brent_fmin)*10^-3 ,'go-',(N_bit_vector) , (
    p_static_dadda_PPA_Brent+p_dynamic_dadda_PPA_Brent_fmin)*10^-3 ,'v-',(N_bit_vector) , (
    p_static_wallace_PPA_Ladner + p_dynamic_wallace_PPA_Ladner_fmin)*10^-3 ,'ko-',(N_bit_vector) , (
    p_static_dadda_PPA_Ladner + p_dynamic_dadda_PPA_Ladner_fmin)*10^-3 ,'m*-');
 grid on
 title (' Total Power Consumption at fixed frequency')
 legend (' Wallace tree  + RCA',' Dadda tree + RCA ',' Wallace tree  + Brent-Kung',' Dadda tree + Brent-
    Kung ',' Wallace tree  + Ladner-Fisher',' Dadda tree + Ladner-Fisher ', 'Location','northwest')
 xticks([8 16 32 64 128])
 xlabel (' Number of bits ')
 ylabel (' Power [mW] ')
```

BoothMultiplier_Group16.m

Quantity name	Description	u.m. (S.I.)	Variable name
λ	referred to the design rules		lambda
T_{ox}	Oxide physical	nm	Tox
L_{eff}	Effective length of the gate	μm	L_eff
$L_{s,d}$	Source and drain length	μm	L_s_d
L_{gate}	Channel physical length	nm	Lgate
$interc_{override}$	Override of area due to interconnection	%	Interc_override
α	activity factor		alfa
V_{DD}	Power supply	V	vdd
W_{gate}	Width of NMOS	μm	Wgate
β	beta factor between NMOS and PMOS		beta
C_{j0n}	NMOS junction capacitance per unit of area	$\frac{pF}{\mu m^2}$	Cj0n
C_{j0p}	PMOS junction capacitance per unit of area	$\frac{pF}{\mu m^2}$	Cj0p
C_{jswn}	NMOS sidewall junction capacitance	$\frac{F}{m}$	Cjswn
C_{jswP}	PMOS sidewall junction capacitance	$\frac{F}{m}$	Cjswp
C_{gd0n}	NMOS overlap capacitance between gate and drain	$\frac{F}{m}$	Cgd0n
C_{gd0p}	PMOS overlap capacitance between gate and drain	$\frac{F}{m}$	Cgd0p
C_{ox}	Oxide capacitance	$\frac{pF}{\mu m^2}$	Cox
L_{diff}	Source and drain diffusion length	μm	lungh_diff
C_{ND2}	Capacitance NAND2	fF	cap
i_{offn}	Subthreshold current for NMOS	mA	ioff_n
i_{offp}	Subthreshold current for PMOS	mA	ioff_p
i_{gaten}	Gate current for NMOS	mA	igate_n
i_{gatep}	Gate current for PMOS	mA	igate_p
i_{ND}	Output current for NAND gate	mA	i_NAND

Table 4: Input data

The outputs are the parameters of each multiplier, that differs from the others for the type of tree (Dadda or Wallace) and final adder (RCA, PPA Ladner-Fisher or PPA Brent-Kung). For sakes of simplicity, the output table contains generic tree and adder types, therefore the reader has to replace the words "tree" and "adder" with a specific label ("dadda" or "wallace" in the first case, "RCA", "PPA_Brent" or "PPA_Ladner" in the second one).

Quantity name	Description	u.m. (S.I.)	Variable name
$N_{NAND_{tree_{adder}}}$	Total number of NAND gates		N_NAND_tree_adder
$A_{TOT_{tree_{adder}}}$	Area of multiplier	μm^2	A_TOT_tree_adder
$t_{cp_{tree_{adder}}}$	Delay of multiplier	ns	t_cp_tree_adder
$freq_{tree_{adder}}$	Maximum clock frequency achievable by multiplier	GHz	freq_tree_adder
$p_{dynamic_{tree_{adder}}}$	Dynamic power of multiplier at $freq_{tree_{adder}}$	μW	p_dynamic_tree_adder
$p_{static_{tree_{adder}}}$	Static power of multiplier	μW	p_static_tree_adder
$p_{tot_{tree_{adder}}}$	Total power of multiplier at $freq_{tree_{adder}}$	W	p_tot_tree_adder
$freq_{min}$	minimum clock frequency (at fixed degree of parallelism)	GHz	freq_min
$p_{dynamic_{NAND_{f_{min}}}}$	Dynamic power of NAND gate at $freq_{min}$	μW	p_dynamic_NAND_fmin
$p_{dynamic_{tree_{adder_{f_{min}}}}}$	Dynamic power of multiplier at $freq_{min}$	μW	p_dynamic_tree_adder_fmin
$p_{total_{tree_{adder_{f_{min}}}}}$	Total power of multiplier at $freq_{min}$	W	p_dynamic_tree_adder_fmin

Table 5: Output data

4 Results

The first graph shows the comparison of the occupied area for each implementation. As it can be observed, the circuit using a parallel prefix Ladner-Fischer adder has the maximum area since it has the maximum number of processing elements, while the area of the circuit that implements a RCA for the final addition is minimal. In all three cases (different final adder), the area of the multiplier based on Dadda tree is slightly less than that of the multiplier based on Wallace tree, since it employs less Full Adders than the other one.

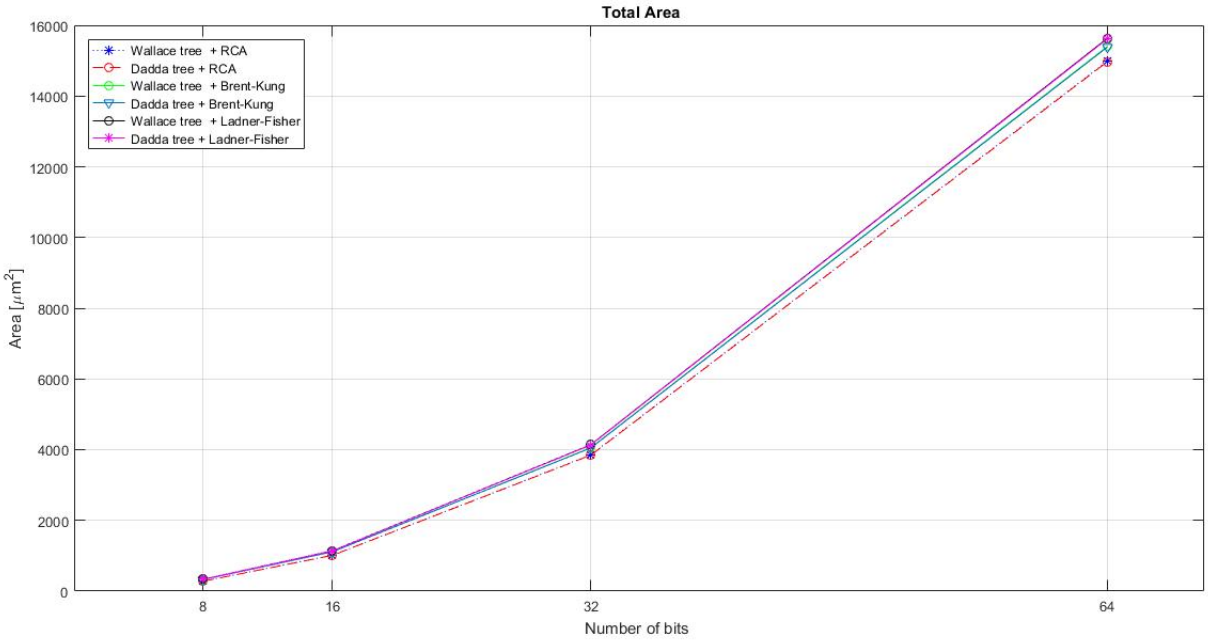


Figure 15: Area

The results concerning the static power consumption are coherent with the results obtained for the area. In fact the implementations with more processing elements; and more cells; have higher static power consumption. For this reason, Ladner-Fischer adder-based multiplier has the highest static power consumption, while the one based on RCA has the lowest static power consumption.

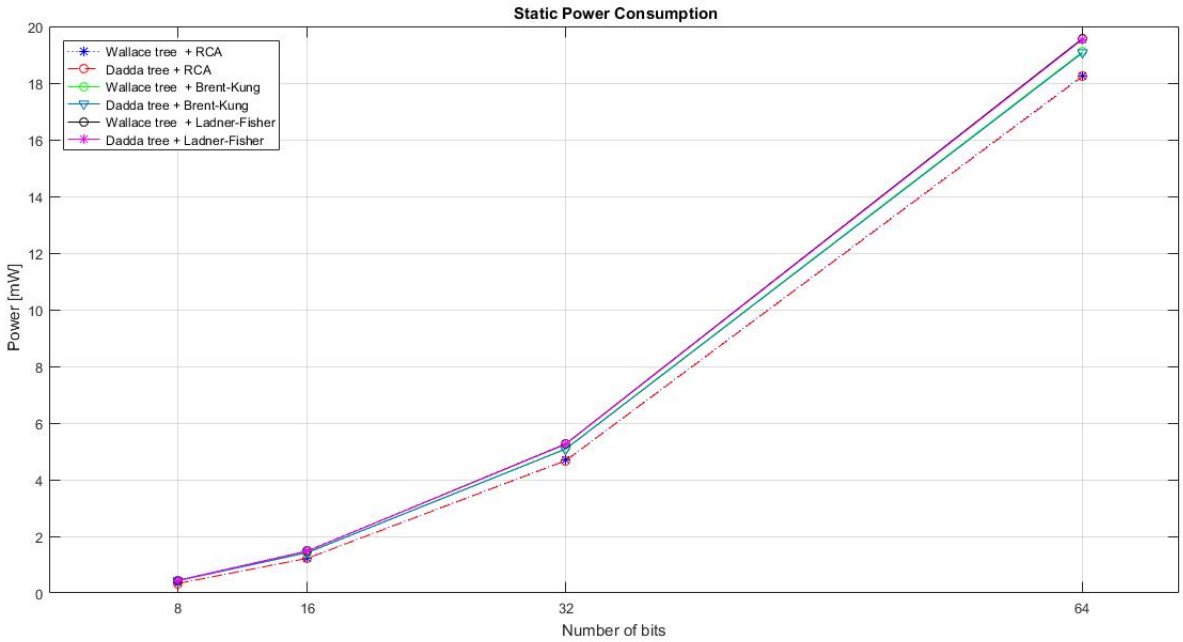


Figure 16: Static Power

Although the RCA shows high advantages in terms of area and static power consumption, it shows on the other hand high delay values due to the chain of Full Adders, so that delay increases linearly with the number of bits. Considering Ladner-Fischer and Brent-Kung adders, the delay's increase with number of bits is highly limited compared to that of the RCA. It is possible to observe that the difference between multipliers involving different trees and the same adder is not significant.

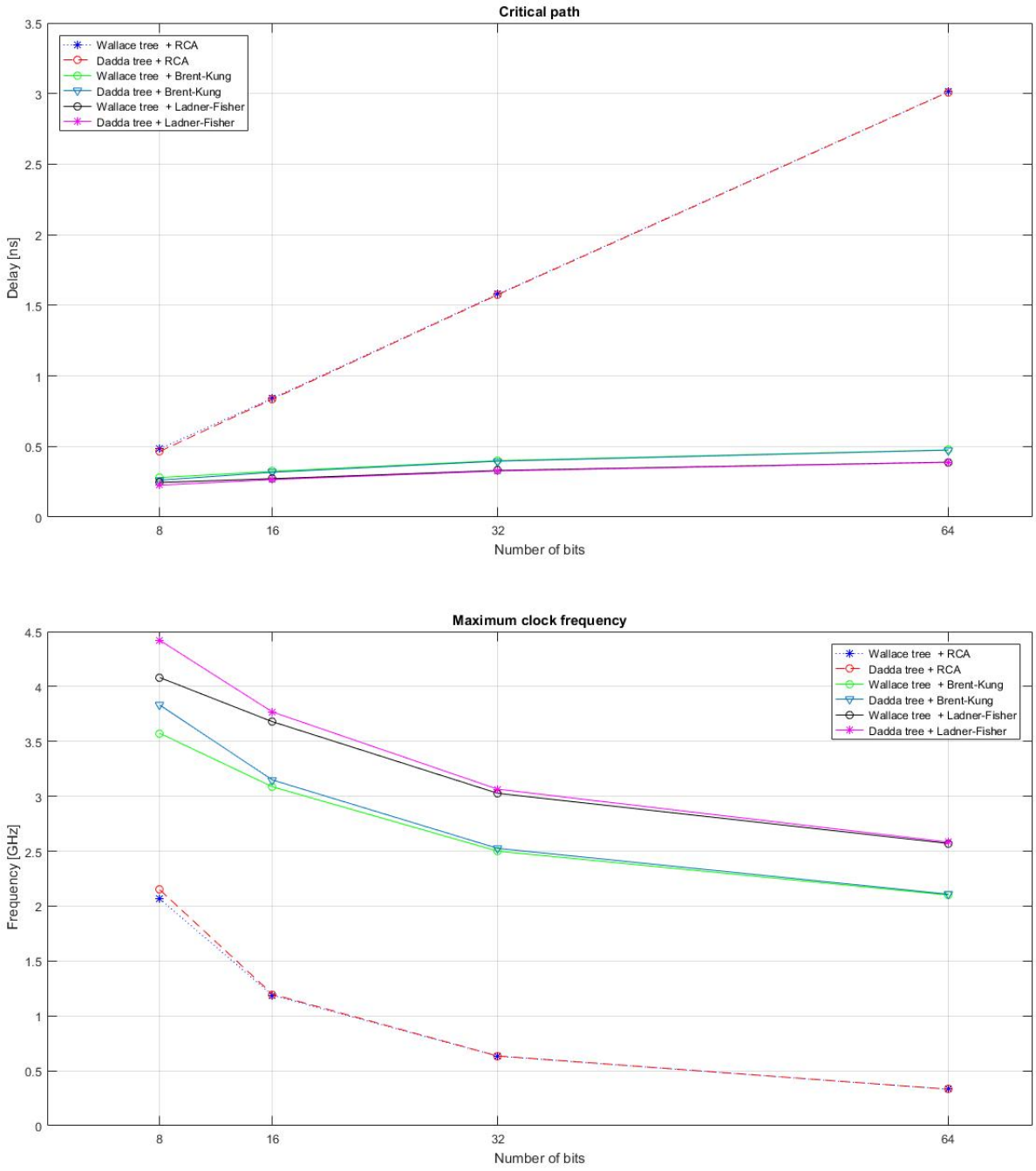


Figure 17: Critical Path Delay and Maximum Clock Frequency.

Finally, a comparison of the dynamic power consumption has been held; the values are much higher than the static power ones. Since the number of NAND gates - hence the total capacitance - does not change significantly in the different configurations (it is possible to ascertain it considering Figure 15), the parameter to which dynamic power has the highest sensitivity is the frequency.
Since the RCA has the highest delay - hence the lowest frequency - it has the lowest dynamic power consumption. Moreover, it can be observed that the dynamic power difference between the Brent-Kung based multiplier and Ladner-Fischer based multiplier becomes more evident as the number of bits increases, with Ladner-Fischer based multiplier having a leading

value. Similiarly to the frequency case, the difference between multipliers involving different trees and the same adder is not significant.

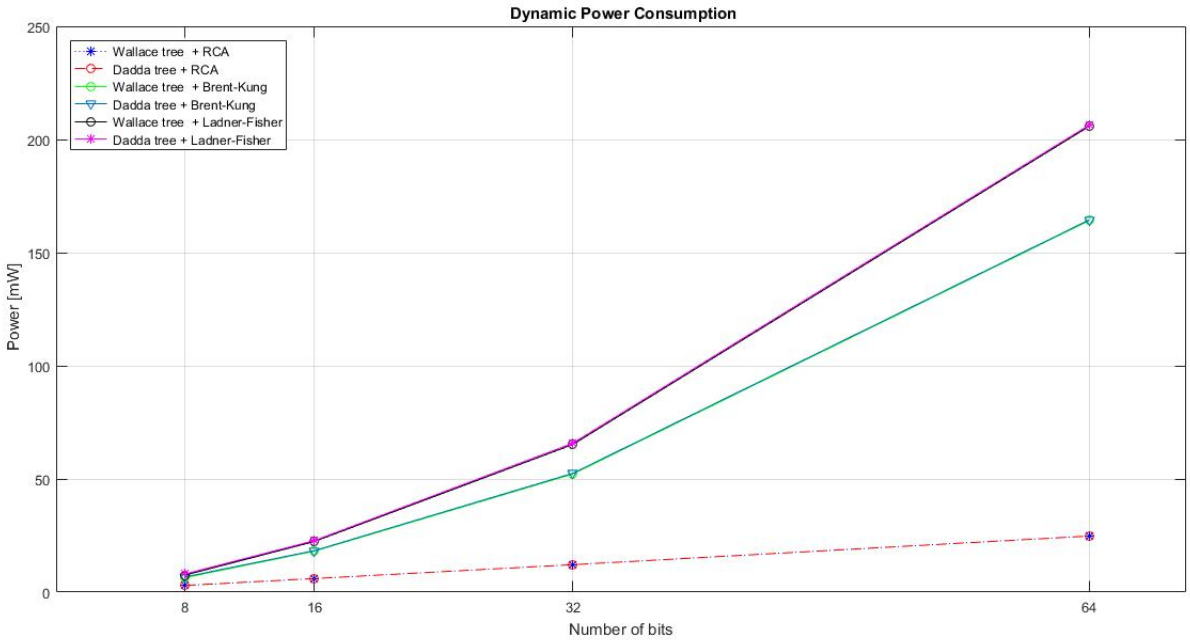


Figure 18: Dynamic Power

As a result, since the dynamic power is dominating in the expression of the total power, the total power of the Ladner-Fischer is the highest one, with the RCA having the lowest total power consumption.

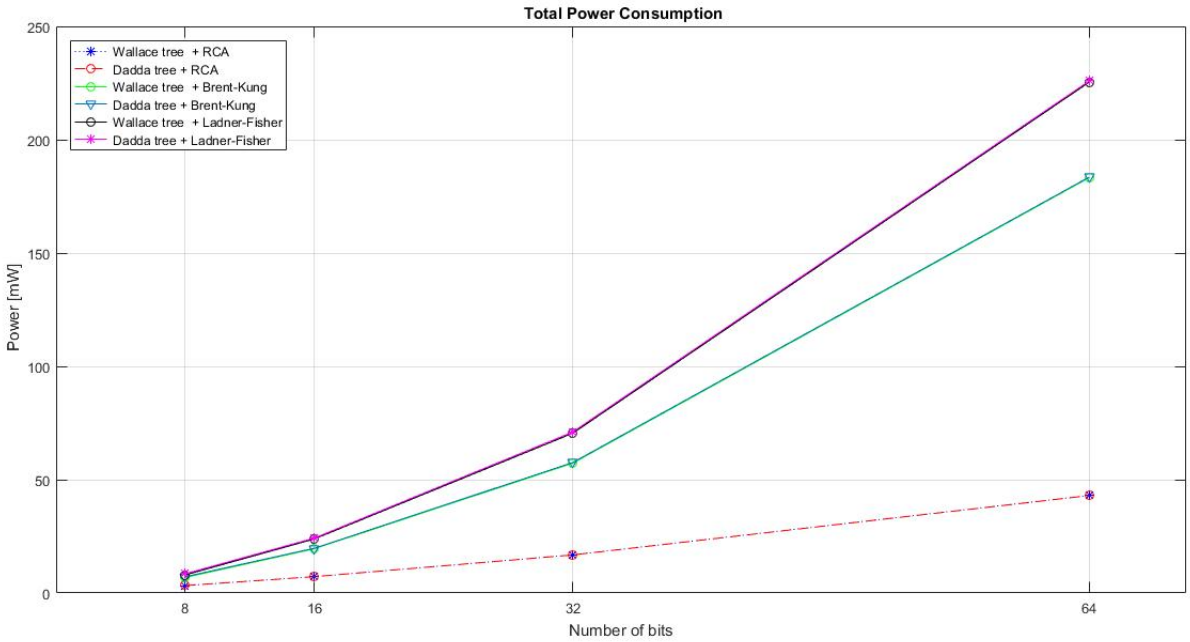


Figure 19: Total Power

The last comparison that has to be considered is that related to the dynamic power of multipliers working at the same frequency, in order to establish which one permits to minimize the power consumption with fixed timing specifications. The minimum frequency among all multipliers with the same degree of parallelism has been taken into account, so that it is ensured that all circuits work properly in terms of timing. The total power in these conditions has been also computed, considering that the static power is the same of Figure 16, since this quantity does not depend on frequency.

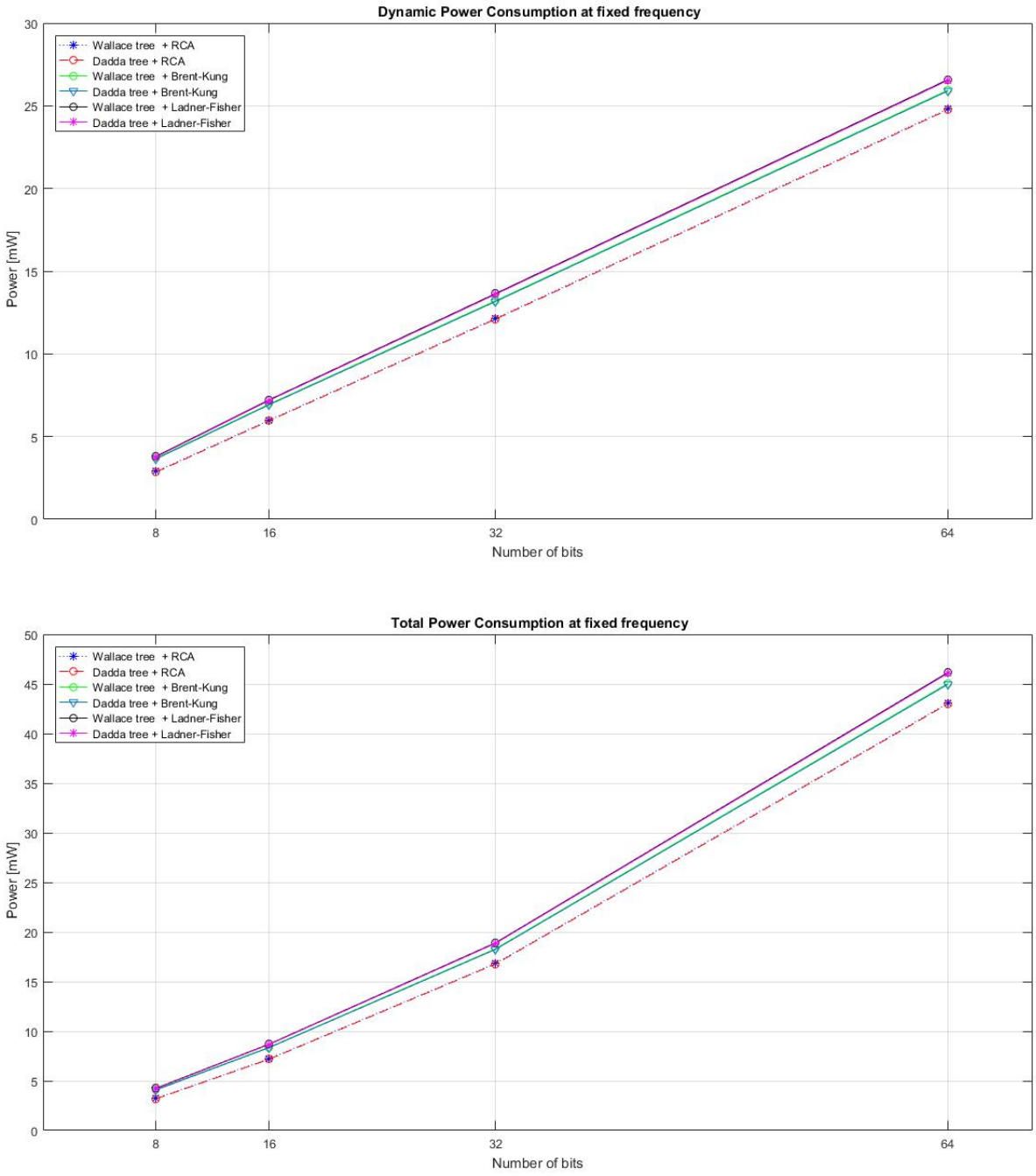


Figure 20: Dynamic and Total power consumptions at fixed frequency

The same behaviour observed in the case of total area and static power of each multiplier has been observed: the multipliers involving the Ripple-Carry Adder present the minimum power consumption, since they have the lowest number of logic gates; in fact - since the dynamic power of a single gate is constant for all multipliers - the only variable parameter that can vary the dynamic power is the number of logic gates.

References

[1] B. Parhami, *Algorithms and Design Methods for Digital Computer Arithmetic*, Oxford University Press, New York, 2012.

[2] O. L. McSorley, "High-speed arithmetic in binary computers", *Proc. Ire*, vol. 49, pp. 67-91, Jan. 1961.

[3] W. C. Yeh and C. W. Jen, "High-speed booth encoded parallel multiplier design", *IEEE Transactions on Computers*, pp. 692-701, 2000.

[4] S. K. Baba and D. Rajaramesh, "Design and implementation of advanced modified Booth encoding multiplier", *International Journal of Engineering Science Invention*, Volume 2, Issue 8, pp. 60-68, August 2013.

[5] C. S. Wallace, "A suggestion for a fast multiplier", *IEEE Transactions on Electronic Computers*, Vol. 13, pp- 14-17, 1964.

[6] L. Dadda, "Some schemes for parallel multipliers", *Alta Frequenza*, Vol. 34, pp. 349356, 1965.

[7] R. S. Waters, and E. E. Swartzlander, "A Reduced Complexity Wallace Multiplier Reduction", *IEEE Transactions on Computers*, vol. 59, no. 8, pp. 1134-1137, August 2010.

[8] R. Monsu‘ and F. Alaimo, "Area, Power and Timing Analysis for Adders", *Integrated Systems Technology Report*, July 2015