International Journal Of Electrical, Electronics And Data Communication, ISSN: 2320-2084 Volume-5, Issue-2, Feb.-2017
http://iraj.in

A NOVEL DESIGN OF POWER AND AREA OPTIMIZATION FOR 4X4 BOOTH MULTIPLIER USING 180NM TECHNOLOGY

[1]SUDHAKAR ALLURI, [2]B.RAJENDRA NAIK, [3]N.S.S.REDDY, [4]RAVINDRA KUMAR NIRANJAN

[1,2]Department of ECE, UCE Osmania University Hyderabad, India
[3]Department of ECE, VCE Osmania University Hyderabad, India, [4]DLRL,Hyderabad, India
E-mail: [1]sudhakaralluri709@gmail.com, [2]rajendranaikb@gmail.com, [3]nssreddy69@gmail.com

Abstract- In Very-large-scale integration (VLSI) application area, delay and power are the important factors for any digital circuits. This paper presents four bit 4x4 Booth Multiplier mapped in Cadence Encounter(R) RTL Compiler Version v14.20-s013_1. By efficiently mapping into cadence tool, area, power and delay are decreased. The results of mapping are viewed using RTL synthesis tool in cadence VIRTUOSO at 180 nm technology, 1.8V. Based on digital signal processing (DSP) architectures, the code for low power is generated using 4x4 Booth Multiplier.

Key words- High-Level Synthesis, 4x4 Booth Multiplier, low power, low area, delay, DSP, VLSI.

I. INTRODUCTION

For the efficient arithmetic and logic operations, the low power and high seed VLSI architectures are need to design for optimizing the parameters like speed and power consumed. In any general purpose microprocessors or digital signal processing (DSP), adders play an important key role. In any Integrated circuit (IC), power dissipation or power consumption is the main design objective after speed and area. In DSP's, the main design part is MAC (Multiplier-Accumulator) unit. For any DSP processor, high speed and simultaneously low power MAC unit is required. Because throughput rate and speed will be in concern for DSP synthesis. Due to the speed growing of the electronic devices like calculator, mobile phones, laptops etc, and mainly low power devices became very fundamental and necessary in today's electronic world. A huge throughput and more speed MAC unit are essential and required for any higher performance DSP's.

The system specifications are processor Intel (R) core (TM) i5-4570 CPU@3.20GHz.,3.20GHz.Installed memory (RAM) 4 GB (usable memory is 3.43GB) and system type: 32-bit operating system (OS).

This paper is formed as follows Section II presents the literature review on 4x4 Booth Multiplier and DSP Section III presents the methodology for 4x4 Booth Multiplier and also discussed the low power analysis and low area. Section IV shows the synthesis and simulation results and they are discussed clearly, finally the paper is concluded with Section V.

II. LITERATURE REVIEW

A binary multiplier performs multiplication of two binary integers. The performance of many computational problems is often dominated by the speed at which a multiplication operation can be executed. The simplest way to perform multiplication is to use a single two input adder. For N inputs that are M and N bits wide, the multiplication takes M cycles using N bit adder. This shift and add algorithm for multiplication adds together M partial products. The partial products are generated by multiplying multiplicand with a bit of the multiplier which is an AND operation and by shifting the result on the basis of the multiplier bit position. The dominating parameter for multiplication is speed. The delay of the multiplication operation depends upon the number of partial products to be used. The number of partial Products to be added is determined by the number of bits that the multiplier or multiplicand have used. In order to get fast performance, one of the methods is to arrange the circuit to generate all the partial products parallel and organizes in an array. The second and widely used algorithm which leads to faster performance is booth algorithm [1]. The booth multiplication algorithm is an algorithm that multiplies two signed binary numbers [2], [3]. In Radix-2 algorithm first will append zero to the multiplier and group the multiplier in such a way that each group consists of 2 bits. So that first pair consists of appended zero and lsb of multiplier bits. And the next pair is the overlapping of the first pair in which msb of the first pair will be the lsb of the second pair. So that the total n partial products will obtain. It requires a large number of full adders and there is no reduction of partial products. Next algorithm is the modified Radix-2 algorithm which also called Radix-4 algorithm. In this algorithm grouping of multiplier is done in such a way that each group consists of 3 bits. After generating all the partial products extend the pact to the produce size [4],[5]. In Radix-8 algorithm the grouping will be 4 bits. In the case of Radix-8 algorithm, it requires complex encoder circuit [6],[7]. The proposed multiplier solves the above problems. The algorithm was invented by Andrew Donald Booth in 1950.The first algorithm is the Radix-2 algorithm [8], [9]. In the scheme of [10], the outputs of the Booth encoder are

used to compensate the error generated by the truncated modified Booth multiplier; this method reduces the error by nearly 50 percent compared with a truncated design without error compensation. Another fixed-width modified Booth multiplier is presented in [11] by utilizing a simplified sorting network as a main part of the error compensation circuit; a high accuracy has been obtained. Moreover, a novel approach has been design to compensate the quantization error in a fixed-width Booth multiplier [12], [13], [14].

III. DESIGN METHODOLOGY

1. BOOTH ALGORITHM

Multiplication of two fixed-point binary numbers will be presented for the following methods: sequential add-shift, Booth algorithm, bit-pair recoding, and array multiplier. The operands are normally *n*-bits and the result is 2*n*-bits, as shown below.

$$\text{Multiplicand:} \quad A = a_{n-1}\, a_{n-2}\, a_{n-3} \cdots a_1\, a_0$$
$$\text{Multiplier:} \quad B = b_{n-1}\, b_{n-2}\, b_{n-3} \cdots b_1\, b_0$$
$$\text{Product:} \quad P = p_{2n-1}\, p_{2n-2}\, p_{2n-3} \cdots p_1\, p_0$$

The algorithm consists of multiplying the multiplicand by the low-order multiplier bit to obtain a partial product. If the multiplier bit is a 1, then the multiplicand

Becomes the partial product; if the multiplier bit is a 0, then zeroes become the partial product. The partial product is then shifted left 1 bit position. The multiplicand is then multiplied by the next higher-order multiplier bit to obtain a second partial product. The process repeats for all remaining multiplier bits, at which time the partial products are added to obtain the product. The sign of the product is positive if the operands have the same sign. If the signs of the operands are different, then the sign of the product is negative. Multiplication of two fixed-point binary numbers is a process of repeated add-shift operations and is best illustrated by examples [15].

Example 1.1

Let the multiplicand and multiplier be two 4-bit operands as shown below. When obtaining the partial products, the sign of the multiplicand is extended left to maintain 2*n* bits in the product. In 2s complement notation, the sign bit is treated the same as any other bit in the operands; that is, the sign bit indicates neither a positive number nor a negative number [15].

Multiplicand:					0	1	1	1	+7
Multiplier:				×)	0	1	1	0	+6
	0	0	0	0	0	0	0	0	
	0	0	0	0	1	1	1		
	0	0	0	1	1	1			
	0	0	0	0	0				
Product:	0	0	1	0	1	0	1	0	+42

Example 1.3

The only restriction in the add-shift technique is that the multiplier must be positive; the multiplicand can be either positive or negative. An example is shown below with a negative multiplicand and a positive multiplier.

Multiplicand:					1	0	1	0	−6
Multiplier:				×)	0	1	0	1	+5
	1	1	1	1	1	0	1	0	
	0	0	0	0	0	0	0		
	1	1	1	0	1	0			
	0	0	0	0	0				
Product:	1	1	1	0	0	0	1	0	−30

Example 1.4 A problem occurs when the multiplier is negative, because the value of the multiplier is not the absolute value. In this case, the multiplier must first be negated through 2s complementation. Then the multiplication operation is performed and the product is 2s complemented. An example is shown below for a positive multiplicand (+7) and a negative multiplier (–3). The product is 91, because the multiplier is an unsigned number of 13.

Multiplicand:					0	1	1	1	+7 (7)
Multiplier:				×)	1	1	0	1	−3 (13)
	0	0	0	0	0	1	1	1	
	0	0	0	0	0	0	0		
	0	0	0	1	1	1			
	0	0	1	1	1				
Product:	0	1	0	1	1	0	1	1	91

The same example will now initially 2s complement the multiplier, perform the multiply operation, then will 2s complement the product to obtain the correct result of
–21.

Multiplicand:					0	1	1	1	+7
Multiplier:				×)	0	0	1	1	+3
	0	0	0	0	0	1	1	1	
	0	0	0	0	1	1	1		
	0	0	0	0	0	0			
	0	0	0	0	0				
	0	0	0	1	0	1	0	1	+21
Product:	1	1	1	0	1	0	1	1	−21

Example 1.5

When both operands are negative, the correct result can be realized by 2s complementing both operands before the operation begins, since a negative multiplicand multiplied by a negative multiplier yields a positive product. The first example below shows the result of not negating both operands, where the multiplicand is –4 and the multiplier is –5; that is, multiplying –4 by 11 to yield –44. The second example 2s complements both operands to obtain a correct product of +20.

Multiplicand:					1	1	0	0	−4
Multiplier:				×)	1	0	1	1	−5 (11)
	1	1	1	1	1	1	0	0	
	1	1	1	1	1	0	0		
	0	0	0	0	0	0			
	1	1	1	0	0				
Product:	1	1	0	1	0	1	0	0	−44

Multiplicand:				0	1	0	0		+4
Multiplier:			×	0	1	0	1		+5
	0	0	0	0	0	1	0	0	
	0	0	0	0	0	0	0		
	0	0	0	1	0	0			
	0	0	0	0	0				
Product:	0	0	0	1	0	1	0	0	+20

In the sequential add-shift technique, the multiplier must be positive. The Booth algorithm treats both positive and negative operands uniformly, including the multiplier. The design of a Booth algorithm multiplier using Verilog HDL. The operands are 4-bit vectors *A[3:0]* and *B[3:0]*; the product is an 8-bit result *RSLT[7:0]*. The following internal wires are defined: *a_ext_pos[7:0]*, which is operand *A* with sign extended, and *a_ext_neg[7:0]*, which is the negation (2s complement) of operand *A* with sign extended.

The following internal registers are defined: *a_neg[3:0]*, which is the negation of operand *A*, and *pp1[7:0]*, *pp2[7:0]*, *pp3[7:0]*, and *pp4[7:0]*, which are the partial
products to be added together to obtain the product *RSLT[7:0]*. The example below illustrates the use of the internal registers. The right-most 4 bits of partial product 1 (*pp1*) are the negation [*a_neg* (2s complement)] of operand *A*, which is generated as a result of the –1 times operand *A* operation. The entire row of partial product 1 (*pp1*) corresponds to *a_neg* with the sign bit extended; that is, *a_ext_neg*.

	Booth algorithm				Bits	3	2	1	0	
	Multiplicand					0	1	1	1	+7
	Recoded multiplier				×	+1	−1	+1	−1	
a_ext_neg →	*pp1*	1	1	1	1	1	0	0	1	← *a_neg*
a_ext_pos →	*pp2*	0	0	0	0	1	1	1	0	
	pp3	1	1	1	0	0	1	0	0	
	pp4	0	0	1	1	1	0	0	0	
		0	0	1	0	0	0	1	1	+35

Assume that the multiplicand is 0111 and the multiplier is 0101 as shown above. This each pair of bits is examined according to the Booth algorithm. Bits *b[1:0]* are examined for all $2^2 = 4$ possibilities. If *b[1:0]* = 00, then partial product 1 (*pp1*) and partial product 2 (*pp2*) are 0000 0000. If *b[1:0]* = 01, then partial product 1 is the 2s complement of operand *A* with sign extended (*a_ext_neg* = 1111 1001) and partial product 2 is +1 times *A*, which is operand *A* aligned to the left of 1 bit position (*{{3{a[3]}}, a[3:0], 1'b0}* = 0000 1110). If *b[1:0]* = 10, then partial product 1 is 0000 0000 and partial product 2 is the 2s complement of operand *A* aligned to the left 1 bit position (*{a_ext_neg[6:0],1'b0}* = 1111 0010). If *b[1:0]* = 11, then partial product 1 is the 2s complement of operand *A* (*a_ext_neg* = 1111 1001) and partial product 2 is 0000 0000. In a similar manner, multiplier bits *b[2:1]* and bits *b[3:2]* are examined.

1.2 Sequential Add-Shift

In this method, the multiplier must be positive. If the multiplier is negative, then the purpose of the multiplier bits is not always the same during the generation of the partial products. Any low-order 0s and the first 1 bit are treated the same as for a positive multiplier; however, the remaining higher-order bits are complemented and have an inverse effect. If the multiplier is negative, then it can be 2s complemented as shown previously, leaving the multiplicand either positive or negative. Negative multipliers are presented in a later section.

An alternative approach is to 2s complement both the multiplicand and multiplier if the multiplier is negative. This is equivalent to multiplying both operands by –1, but does not change the sign of the product. Examples will now be presented to illustrate the hardware add-shift technique for both positive and negative multiplicands. A register is a logic macro device that stores information. The information is retained in the register until changed by setting new information into the register. A shift register is a register that is capable of shifting the contents either left or right a specified number of bits. The *n*-bit multiplicand is stored in an *n*-bit register. The multiplier is placed in the low-order *n* bits of a 2*n*-bit shift register that will ultimately contain the 2*n*-bit product. The high-order *n* bits of this shared register are set to zeroes. The low-order multiplier bit determines the operand to be added to the previous partial product: if the low-order multiplier bit is 0, then no addition takes place and the partial product is shifted right 1 bit position; otherwise, the multiplicand is added to the partial product with sign extended to 2*n* bits and then shifted right 1 bit position [15].

Example 1.6 The add-shift multiply hardware algorithm is shown in Figure 1 for a multiplicand of +5 and a multiplier of +6. There are *n* cycles for *n*-bit operands. During the first cycle, if the low-order multiplier bit is 0, then the shift register is shifted right 1 bit position and the high-order bit of the shift register is propagated to the right. If the low-order multiplier bit is 1, then the multiplicand is added to the high-order *n* bits of the shift register and the register is shifted right 1 bit position with the sign of the multiplicand placed in the high-order bit position of the shift register.

The sign of the partial product is not determined until the first add-shift cycle. Any carry-out during the add operation is shifted into the resulting partial product during the subsequent shift operation of an add-shift cycle.

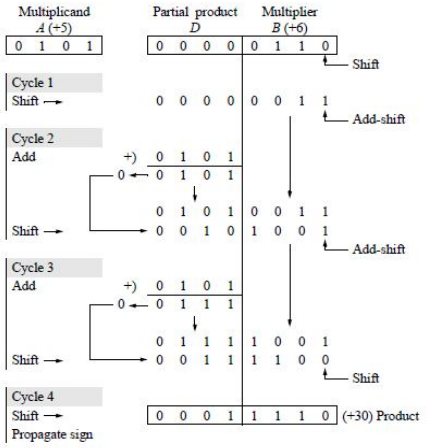


Figure 1 Hardware multiply algorithm for a positive multiplicand and a positive multiplier.

Example 1.7 In Figure 2, the multiplicand is negative (–5) and the multiplier is positive (+7) resulting in a product of –35. The sign of the product is again determined during the first add-shift cycle in which the sign of the multiplicand becomes the sign of the product. The sign can also be determined by the following expression for two *n*-bit operands $A = an – 1\ an – 2\ an – 3 \ldots a1\ a0$ and $B = bn – 1\ bn – 2\ bn – 3 \ldots b1\ b0$:

$an – 1 \oplus bn – 1$

This example has three add-shift cycles. The second and third add-shift cycles shift the carry-out of the add operation into the subsequent partial product during the shift right operation.

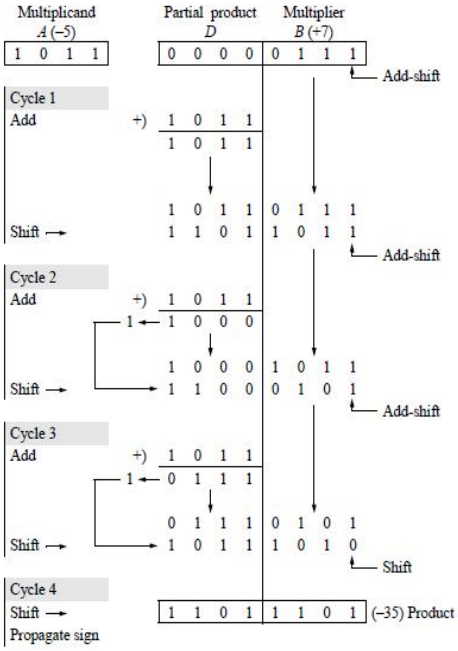


Figure 2 Hardware multiply algorithm for a negative multiplicand and a positive multiplier [15].

During the add operation, a carry look ahead adder of *n* bits is required since only the high-order *n* bits are used for the add operation. The shift register, however, must be 2*n* bits. When multiplication is implemented in a computer, a single adder is used to add the multiplicand to the high-order *n* bits of the shifted partial product and the result is placed in register *D*, which is part of the 2*n*-bit shift register *DB*. A count-down counter may be used to control the number of cycles during the multiply operation. The counter is set to the number of bits that is represented by the value of *n*. When the counter reaches a value of zero, the operation is finished. Alternatively, the operation can be controlled by a micro program sequencer.

2. Low Power analysis

2.1 Dynamic Power

The total power of System on Chip design consists of dynamic power and static power [13].

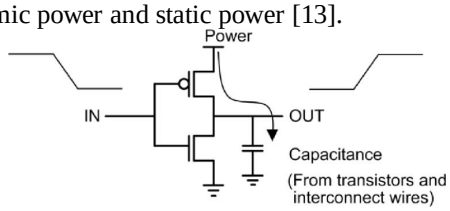


Figure 3: Dynamic Power [16].

Dynamic power is the power consumed when the material is in active mode. Static power is the power dissipated when the device is in active mode but the signal values are unchanged.

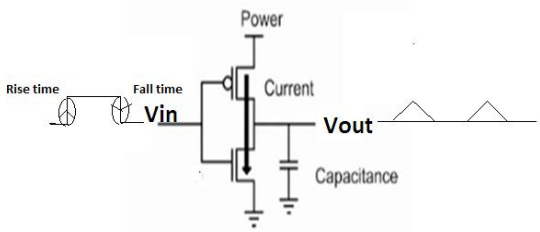


Figure 3: Short Circuit (Crow Bar) Power [16].

As input changes slowly power dissipation takes place even when there is no load or parasitic capacitor. That is known as the short circuit current.

$$P_{SC} = I_{SC} * v \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

2.2 Static Power

Mainly the leakage currents are of four types. (Figure 5)

• Sub-threshold Leakage (ISUB): it is the current flows from drain to source which operates in inverse region.

• Gate Leakage (IGATE): it is the current flows from gate through oxide to the substrate [13].

• Gate Induced Drain Leakage (IGIDL): it is the current flows from drain to substrate caused by high voltage effect in MOSFET due to VDG.

• Reverse Bias Junction Leakage (IREV): it is the current caused because of minor drift and creation of electron hole pairs in the immobile region [16].

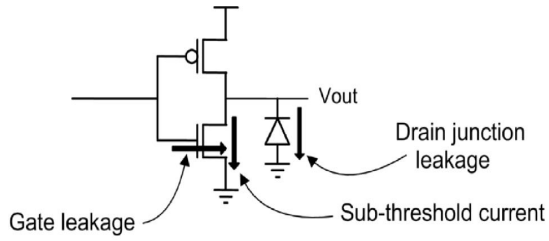


Figure 5: Leakage Currents [16]

$$P_{Static} = I_{Static} * v_{dd} \quad (2)$$

$$P_{Dynamic} = \alpha * c_L * v_{dd}^2 * f \quad (3)$$

$$P_{Shortcircuit} = I_{SC} * v \quad (4)$$

$$P_{Leakage} = V_{dd} * (I_S + I_G + I_D) \quad (5)$$

$$P_{Total} = P_{Dynamic} + P_{Leakage} \quad (6)$$

$$P_{Total} = (\alpha * c_l * v_{dd}^2 * f) + V_{dd} * (I_S + I_G + I_D) \quad (7)$$

Where α is a switching activity factor, c_L is a load capacitance, V_{dd} is a voltage (drain to drain), f is a frequency, I_S (source current), I_G (Gate current) and I_D (Drain current) [16].

IV. SYNTHESIS AND SIMULATION RESULTS

This paper presents 4x4 Booth Multiplier is mapped into Cadence Encounter(R) RTL Compiler Version v14.20-s013_1.By efficiently mapping into cadence tool, area, power and delay are decreased. The results of mapping are viewed using RTL synthesis tool in cadence VIRTUOSO at 180nm technology, 1.8V. Based on DSP architectures, the code for low power is generated using 4x4 Booth Multiplier.

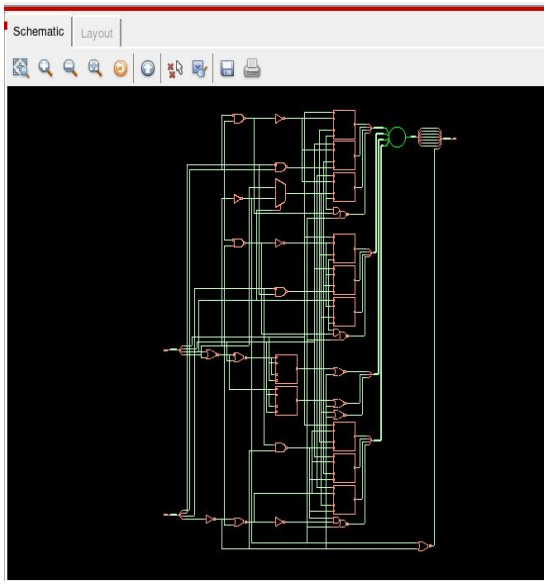


Figure 6 Schematic diagram of 180nm technology.

The proposed 4x4 Booth Multiplier code is mapped into the cadence tool for synthesis and observed the schematic in figure 6.

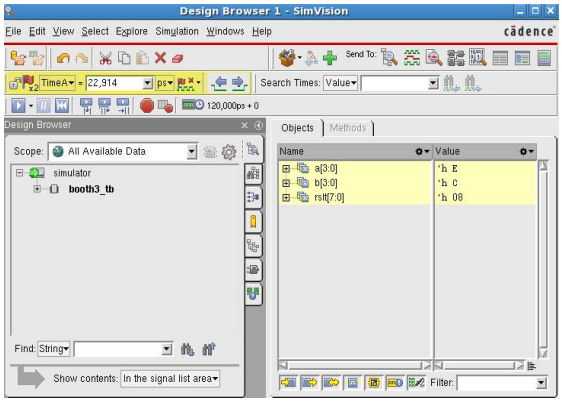


Figure 7 Design browser 1-simvision

The Figure 7 represents 4x4 Booth Multiplier design browser 1-simvision. The 1-simvision gives the analysis of output waveforms respective input.

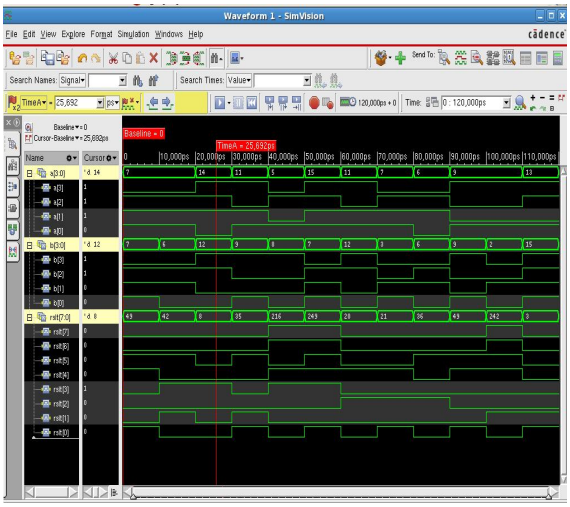
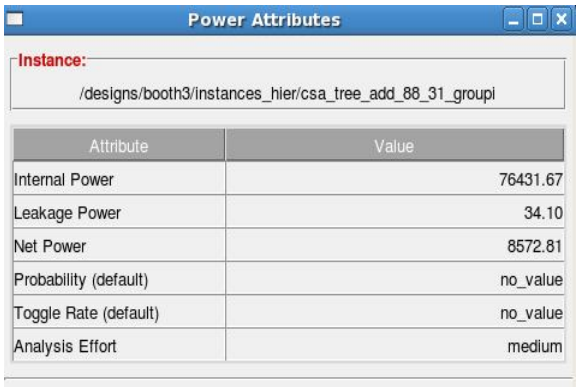


Figure 8 Simulation Result

In Figure 8, the proposed 4x4 Booth Multiplier simulation is observed after mapping into the cadence tool at 180nm technology. The output of the 4x4 Booth Multiplier waveform has the frequency of 8.33 MHz.



Attribute	Value
Internal Power	76431.67
Leakage Power	34.10
Net Power	8572.81
Probability (default)	no_value
Toggle Rate (default)	no_value
Analysis Effort	medium

Figure 9 4x4 Booth Multiplier of power attributes

We proposed mapping style into cadence tool 180nm using 4x4 Booth Multiplier by observing the figure 9 4x4 Booth Multiplier r power attributes.

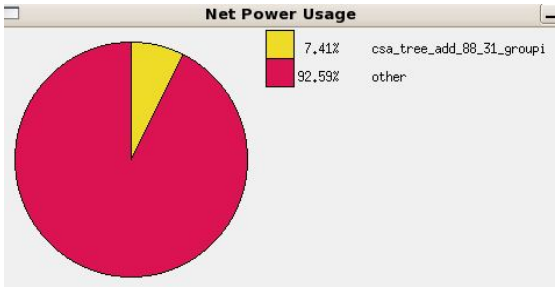


Figure 10 net power usages

We proposed mapping style into cadence tool 180nm using 4x4 Booth Multiplier. By observing the figure 8, 4x4 Booth Multiplier net power usage

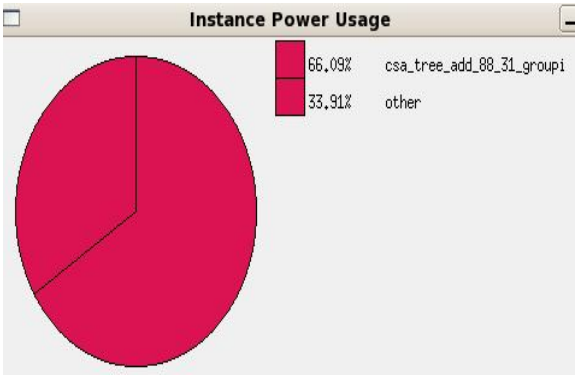


Figure 11 instance power usage

We proposed mapping style into cadence tool 180nm using 4x4 Booth Multiplier. By observing the figure 11, 4x4 Booth Multiplier instance power.

Table 1 4x4 Booth Multiplier area using cadence tool 180nm

Instance	Cells	Cell Area	Net Area	Total Area
booth3	47	1390	0	1390
csa_tree_add_88_31_groupi	15	901	0	901

We design mapping style into cadence tool 180nm using 4x4 Booth Multiplier. By observing the table 1, 4x4 Booth Multiplier area.

Table 2 4x4 Booth Multiplier power dissipation using cadence tool 180nm

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
booth3	47	43.615	115662.546	115706.161
csa_tree_add_88_31_groupi	15	34.097	85004.478	85038.575

We proposed mapping style into cadence tool 180nm using 4x4 Booth Multiplier. By observing the table 2, 4x4 Booth Multiplier power dissipation.

Table 3 4x4 Booth Multiplier delay using cadence tool 180nm

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)
a[0]	in port	3	10.4	0	+0	0 F
g492/B					+0	0
g492/Y	NOR2XL	3	8.5	146	+93	93 R
g476/A0N					+0	93
g476/Y	AOI2BB2XL	4	14.1	241	+136	229 R
g474/A					+0	229
g474/Y	NOR2XL	1	7.2	86	+40	270 F
csa_tree_add_88_31_groupi/in_2[2] g382/A					+0	270
g382/S	ADDFX2	1	6.1	52	+196	465 R
g380/B					+0	465
g380/CO	ADDHXL	1	7.2	86	+88	554 R
g375/A					+0	554
g375/CO	ADDFX2	1	7.2	63	+229	783 R
g374/A					+0	783
g374/CO	ADDFX2	1	2.8	52	+219	1002 R
g2/CI					+0	1002
g2/CO	CMPR42X1	1	6.5	68	+131	1132 R
g372/CI					+0	1132
g372/CO	ADDFXL	1	2.1	59	+107	1239 R
g371/B					+0	1239
g371/Y	XNOR2XL	2	6.0	74	+157	1397 F
g370/A1N					+0	1397
g370/Y	OAI2BB2XL	1	0.0	34	+76	1473 F
csa_tree_add_88_31_groupi/out_0[7] rsk[7]	out port				+0	1473

We proposed mapping style into cadence tool 180nm using 4x4 Booth Multiplier. By observing the table 3 4x4 Booth Multiplier delays.

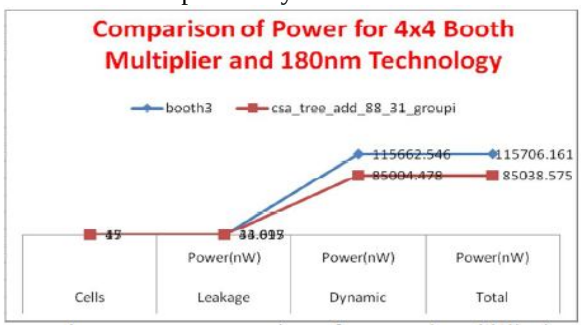


Figure 12 power comparison of 4x4 Booth Multiplier in 180nm technology.

We proposed mapping style into cadence tool. Figure 12 gives the power comparison of 4x4 Booth Multiplier in 180nm technology.

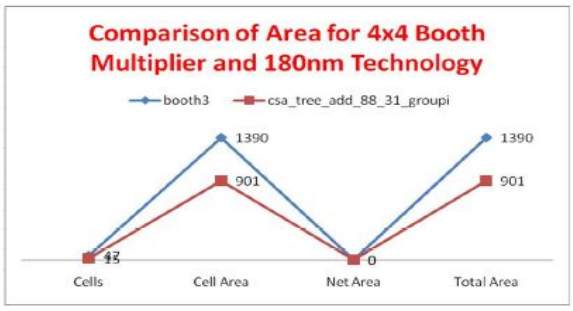


Figure 13 Area comparison of 4x4 Booth Multiplier in 180nm technology.

We proposed mapping style into cadence tool. Figure 13 gives the area comparison of 4x4 Booth Multiplier in 180nm technology.

CONCLUSION

In this paper, we proposed mapping style into cadence tool using 4x4 Booth Multiplier. table 1 gives 4x4 Booth Multiplier of area at 180nm technology, table 2 represents 4x4 Booth Multiplier of power dissipation at 180nm technology, table 3 gives 4x4 Booth Multiplier of delay at 180nm technology, the area, power, and delay obtained are low because the mapping the style in cadence virtuoso at 180nm technology , and power given is 1.8V. With the help of DSP architectures, the code is generated which results in low power.

ACKNOWLEDGEMENT

We acknowledge the University Grants Commission for its economic support in the form of Rajiv Gandhi National Fellowship (RGNF) for the year 2015-16. We thank all the faculties in the Department of (ECE) Electronics and Communications Engineering and the lab staff for their support.

REFERENCES

[1] Kumre, Laxmi, Ajay Somkuwar, and Ganga Agnihotri. "Implementation of radix 4 booth multiplier using MGDI technique." Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy (AICERA/ICMiCR), 2013 Annual International Conference on. IEEE, 2013.

[2] Jose, Bijoy, and Damu Radhakrishnan. "Fast redundant binary partial product generators for Booth multiplication." Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on. IEEE, 2007.

[3] Rao, Pachara V., C. Prasanna Raj P, and S. Ravi. "Vlsi design and analysis of multipliers for low power." Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP'09. Fifth International Conference on. IEEE, 2009.

[4] Cho, Ki-seon, et al. "54x54-bit radix-4 multiplier based on modified booth algorithm." Proceedings of the 13th ACM Great Lakes symposium on VLSI. ACM, 2003.

[5] Venkatraman, S., et al. "Low Power Area Ecient Multiplier Using Shannon Based Multiplexing Logic." International Journal of Embedded Systems Applications 2.2 (2012).

[6] C.Senthilpari, Ajay Kumar Singh and K.Diwakar "Design of a low power, high performance, 8x8 bit multiplier using a Shannonbased adder cell." Microelectronics Journal 39 (2008) 812821.

[7] Rajput, Ravindra P., and MN Shanmukha Swamy. "High speed Modified Booth Encoder multiplier for signed and unsigned numbers." Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on. IEEE, 2012.

[8] Kaur, Lakhvinder, and Parminder Singh Jassal. "Synthesis And Simulation Of 8x8-Bit Modified Booth's Multiplier." Signal processing 1: 2.

[9] N. H E Weste,Principle of CMOS VLSI design, Adision-Wesley 1998.

[10] K.-J. Cho, K.-C. Lee, J.-G. Chung, and K. K. Parhi, "Design of low-error fixed-width modified Booth multiplier," IEEE Trans. VLSI Syst., vol. 12, no. 5, pp. 522–531, May 2004.

[11] J.-P.Wang, S.-R. Kuang, and S.-C. Liang, "High-accuracy fixed-width modified Booth multipliers for lossy applications," IEEE Trans. VLSI Syst., vol. 19, no. 1, pp. 52–60, Jan. 2011.

[12] C.-Y. Li, Y.-H. Chen, T.-Y. Chang, and J.-N. Chen, "A probabilistic estimation bias circuit for fixed-width Booth multiplier and its DCT applications," IEEE Trans. Circuits Syst. II, vol. 58, no. 4, pp. 215–219, Apr. 2011.

[13] Y.-H. Chen, C.-Y. Li, and T.-Y. Chang, "Area-effective and power-efficient fixed-width Booth multipliers using generalized probabilistic estimation bias," IEEE J. Emerging Select. Topics Circuits Syst., vol. 1, no. 3, pp. 277–288, Sep. 2011.

[14] Y.-H. Chen and T.-Y. Chang, "A high-accuracy adaptive conditional-probability estimator for fixed-width Booth multipliers," IEEE Trans. Circuits Syst. I, vol. 59, no. 3, pp. 594–603, Mar. 2012.

[15] Joseph Cavanagh, "Digital Design Verilog HDL and Fundamentals", Santa Clara University California, USA, © 2008 by Taylor and Francis Group, LLC CRC Press is an imprint of Taylor & Francis Group, an Informa business,, http://www.crcpress.com.

[16] Michael Keating, David Flynn., Robert Aitkin Alan G , ibbons, Kaijian Shi., "Low Power Methodology Manual For System-on-Chip Design., Library of Congress Control Number: 2007928355 .,ISBN 978-0-387- 71818-7 e-ISBN 978-0-387-71819-4., springer.com.

★★★