A Project Report

On

Power Efficient Approximate Booth Multiplier

BY

RISHI S PHAYE
(H20201400230)
JEEVARAAM KUMAR
(H20201400216)
SUKRUTH S
(H20201400236)

Under the supervision of

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF VLSI ARCHITECTURE (BITS G540)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN), HYDERABAD CAMPUS

ABSTRACT

An arithmetic processor's speed and power dissipation are both affected by the multiplier. Many algorithms, such as classification and recognition in data processing, do not necessarily need precise results. Furthermore, due to human perceptual limits, many mistakes do not make a visible impact in applications such as image processing. Approximate multipliers have been advocated by error-tolerant algorithms and applications to trading off accuracy for speed, implementation area, and power efficiency.

For the booth multiplier algorithm, we used four different ways in this research. The results of a behavioural and structural model of the precise and approximate booth multiplier have been cross-checked. In the exact booth multiplier, a new feature has been added: we have not utilised the 4-2 compressor in our implementation.

Contents

List of Figures

List of tables

1. Introduction

We are on the verge of a massive data explosion, which will be fuelled not just by massive, powerful scientific and corporate computers, but also by billions of different sorts of low-power gadgets. As conventional workloads such as transactional processing and databases mature, the computing footprint of a variety of applications is constantly expanding in order to get deep insight into vast volumes of organised and unstructured data. Traditional computing requires a level of precision that is not required for the majority of data processing types. Nonetheless, these cognitive applications continue to be very accurate and dependable systems for everyday use (and accelerators). Since major applications like scientific computing, social media, and financial analysis are gaining importance, modern systems' processing and storage need far surpass the capacity available. In the next decade, the volume of data managed by worldwide data centres is expected to increase by 50 times, but the number of CPUs will increase by ten times. Approximate computation (and storage) seem to be a viable solution to this problem.

For several applications, including image and multimedia data processing, artificial intelligence and machine learning, the need for high speed and low power in nano-scale integrated circuits (IC) has stimulated the development of approximation computing. Approximation circuits have been widely investigated and influence the energy efficiency of these systems, particularly approximation arithmetic units. It is based on the intuitive observation that, even if exact computations are performed or high-level demand is maintained, selective approximation or infrequent violations may result in disproportionate efficiency improvements.

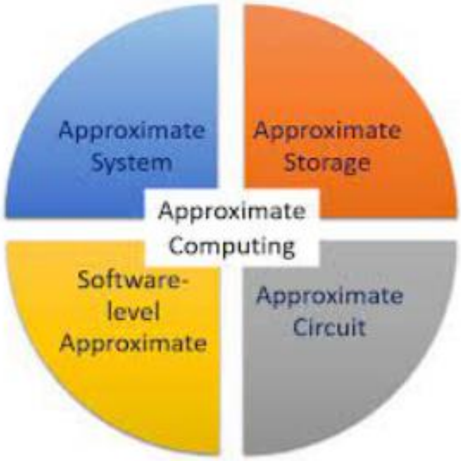


Figure 1.Approximate computing strategies

As previously stated, research on different methods for the utilisation of tolerated errors in about The use of arithmetic, software, and architectures that tolerate errors is growing. Signal and image processing are two domains where approximation arithmetic has found a home. The core of digital speculate circuits is a speculative functional unit, an arithmetic unit employed for signal transmission rather than waiting for it to propagate as a predictor. The speculative unit hypothesises the movement of one or more cells on a digital circuit without true propagation. It is analogous to a microprocessor predictor.

All of them may be related to design inefficiencies, availability, and dependability, hard standardised formats, stringent software portability requirements, and accurate outcomes duplication. These specifications are not usually strictly required. The portability/reproducibility criteria can be employed with a narrower ("approximate") range. Variable precision was developed as an on-the-fly solution to the inefficiencies of fixed formats, and it is now extensively utilised in software and reconfigurable hardware.

Approximate multipliers are typically recommended for energy-efficient calculation in applications with an inherent tolerance for errors. However, in addition to performance, scope and power, the added preciseness as an essential design parameter makes it challenging to select the optimum approximator.

The Booth multiplier is commonly used for high-performance signed multiplication that uses encryption to reduce partial products. The radix 4 (or modified Booth) multiplier is relatively efficient due to the ease of the partial product creation, while a radix-8 booth multiplier is sluggish due to the complexity of creating the odd multiplicand. Booth multipliers are extensively employed and reduce the number of incomplete products by roughly half. Although truncation of fixed point booth multipliers has gotten much attention, approximation of booth multipliers has gotten a lot less attention. In the creation of the radix-8 portion, an approximation is used.

This study describes a new approximation approach for creating radix-4 recoded partial products. OR gates are utilised to efficiently approximate the created exact and approximate recorded partial products after the production of partial products and accompanying correction terms. Although OR gates are a standard way of approximation, variable input OR gates are used to increase design and accuracy in well-fitting areas.

1. Related work

Several publications have been written about the implementation of an approximate booth multiplier. We have covered a handful and compared them to the best of our ability. To the best of our knowledge, no article has employed the method described in this research.

S.No	Publication and Year	Paper and Author	Content	Limitation
1	2013	Approximate computing: An emerging the paradigm for energy-efficient design J. Han and M. Orshansky	The design of approximation arithmetic blocks, relevant error and quality measurements, and algorithm-level strategies for approximation computing are all covered in this study.	Recent advances in approximation computing are discussed in this work, emphasising approximation circuit design, relevant error measures, and algorithm-level approaches. As a new paradigm, approximate computing has much potential for developing energy-efficient and error-tolerant systems. No precise method is explored.
2	2016	Design-Efficient Approximate Multiplication Circuits Through Partial Product Perforation G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris and K. Pekmestzi	In this study, the author introduces the partial product perforation methodology for building approximate multiplication circuits, which focuses on hardware-level approximation. In partial product perforation, the author shows that the imposed errors are finite and predictable, based only on the input distribution, in a mathematically rigorous manner.	The suggested method eliminates several incomplete products, resulting in significant space and power savings while maintaining excellent accuracy. The omitted partial products may lead to improper results.
3	2017	Energy-efficient approximate multiplier design using bit significance driven logic compression I. Qiqieh, R. Shafik, G. Tarawneh, D.	The use of significance-driven logic compression offers a unique approximation multiplier design in this article (SDLC). This design solution reduces the number of partial product terabytes using an algorithmic and adjustable lossy	The suggested methodology may not be applied with low-power computation units that currently exist to extract several advantages with minimum output quality degradation.

		Sokolov and A. Yakovlev	compression based on bit significance.	
4	2020	Design Of Area And Power Efficient Booth Multipliers Using Modified Booth Encoding Sodadasi.Queen Victoria and Kottu.Veeranna Babu	For n = 64-bit unsigned operands, we present an optimisation for binary radix-4 modified Booth recoded multipliers that reduces the maximum height of the partial product columns to [n/4]. In contrast to the standard maximum height of [(n + 1)/4]. As a result, the maximum height has been reduced by one unit.	For 64-bit and 128-bit radix-4 Booth recoded magnitude multipliers, a solution to lower the maximum height of the partial product array by one has already been implemented.

2. Proposed work

According to the presented document, the suggested work may be divided into four portions described below. These are the ones.

- Encoding and multiplication of radix-4 booths
- Booth multipliers approximation LP
- LP with modest approximation
- Minor equivalence

Booth multiplier encoding

A Booth multiplier consists of three components: a Booth encoder for partial product generation, compressors for partial product accumulation, and a fast adder for final product generation. Booth encoding was developed to increase two-s complement binary number multiplication; it was subsequently enhanced by MBE or radix-4 Booth encoding. The Booth encoder is a crucial component of the Booth multiplier, which cuts the number of incomplete product rows in half.

Consider the two-s complement multiplication of two N-bit integers, that is, a multiplicand A and a multiplier B:

$$A = -a_{N-1}2^{N-1} + \sum_{n=0}^{N-2}\{a_n\}2^n$$

$$B = -b_{N-1}2^{N-1} + \sum_{n=0}^{N-2}\{b_n\}2^n$$

$$Pout = -P_{2n-1}2^{2n-1} + \sum_{n=0}^{2n-2}\{P_n\}2^n$$

Figure 2. Refers 2s complement notation of inputs

We may now go on to Booth's multiplication, which aims to decrease addition and subtraction operations. The basic concept behind Booth's multiplication is that we are transforming our multiplier from a two's complement representation to a balanced ternary representation. The most crucial point is that 0111 may also be written as 1001', as we saw before. As previously stated, a multiplier of 0111 will require three addition/subtraction operations, but a multiplier of 1001' will only require two.

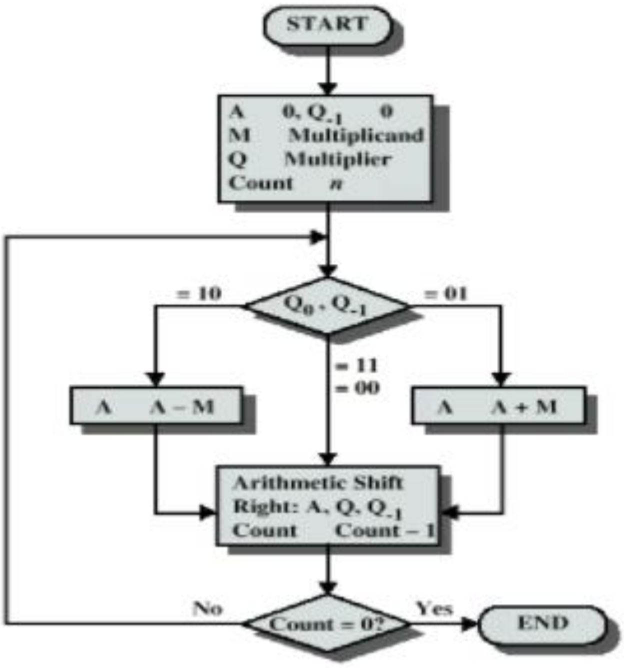


Figure 3. Basic booth flowchart

The algorithm for converting this is relatively simple. In theory, all we want to do is find the start of a string of ones. When we do, we convert it to 1' and then add 0's until the string of 1's is complete, at which point we add a 1. As a result, the algorithm :

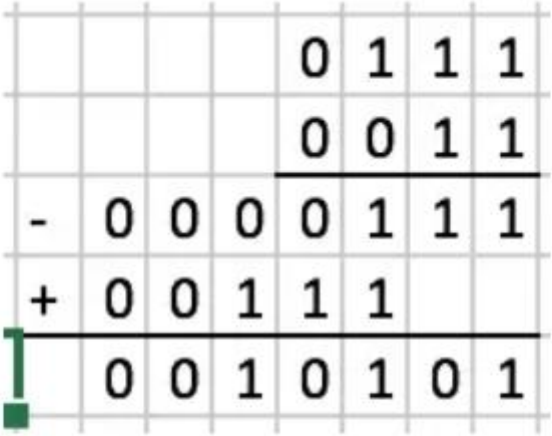
1. Set the partial product to zero.
2. Add a fake 0 bit to the multiplier's least significant bit.
3. Starting from the least significant two bits, check the following:
    a. If 00, do nothing
    b. If 01, add the multiplicand to the partial product
    c. if 10, subtract the multiplicand from the partial product
    d. if 11, do nothing
4. By multiplying the multiplicand by two, we may get the multiplicand.
5. Repeat steps 3 and 4 with the following two bits of the multiplier.
6. Return the partial product, which will now be the entire product, when we run out of multiplier bits.

So, for the 0111 multiplicands and 0011 multiplier example above, you would execute the following: :

1. The partial product is zero.
2. Add a dummy 0: 0011 -> 0011|0
3. Look at the least significant 2 bits, which are 10
    a. Ten means we do a subtraction, so we will subtract our multiplicand and get a partial product of 1001
4. We now increase our multiplicand by two, yielding 01110.
5. Look at the next 2 bits, which are 11

   a. We do nothing
6. Multiply our multiplicand by two more times to get 011100.
7. Look at the next 2 bits, which are 01
   a. Add our multiplicand, so we have 011100 + 111001 = 010101
8. Multiply our multiplicand by two more times to get 011000.
9. Because our final two bits are 00, we do nothing and return 010101 as our final response.

The multiplication looks somewhat like this when written out.



				0	1	1	1
				0	0	1	1
-	0	0	0	0	1	1	1
+	0	0	1	1	1		
	0	0	1	0	1	0	1

Figure 4. Example 1 representation

When executing multiplication, this approach aims to decrease the amount of addition/subtraction operations. However, there are two apparent exceptions. If our multiplier is 010101, we end up converting it to 11'11'11'. We now have six addition/subtraction operations when we would only have had three if we had simply left the multiplier alone.

When our multiplier is 101011, we have another instance. Our algorithm converts this to 1'11'101'. This provides us with five operations when we only had four before.
By looking at the multiplier three digits at a time, Modified Booth's solves these two instances. When we encounter the number 010, we want to conduct a simple addition. Similarly, we may rephrase 101 as 01'0 and perform a single subtraction as a result. So for the 010 cases, after we do the addition, we need to replace the 1 with a 0 to ensure that we do nothing on the next step, and likewise, with the 101 cases, we replace the 0 with a 1.

The Modified Booth's algorithm is thus:

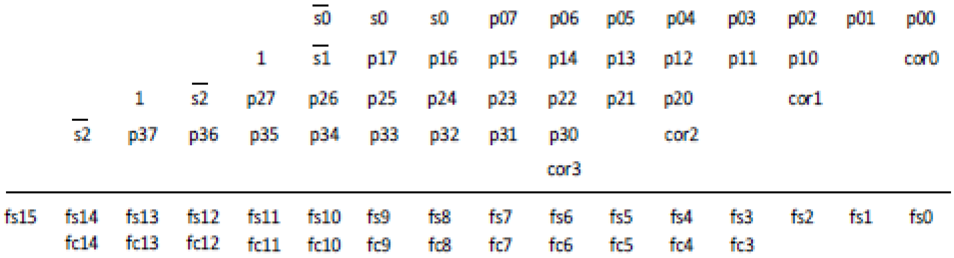
1. Sign extend the most significant bit and add a fake zero to the least significant bit.
2. The partial product is zero.
3. Take a look at the three least significant bits.
   a. If 000, do nothing
   b. If 001, add the multiplicand
   c. If 010, do addition and replace the 1 with a 0
   d. If 011, do nothing
   e. If 100, do nothing
   f. If 101, do subtraction and replace the 0 with a 1
   g. If 110, do a subtraction

   h. If 111, do nothing
4. Two times the multiplicand
5. Repeat steps 3 and 4 with the following three bits of the multiplier.
6. Return the partial product, which will now be the full product, when we run out of bits.

A table representation of the given logic is shown

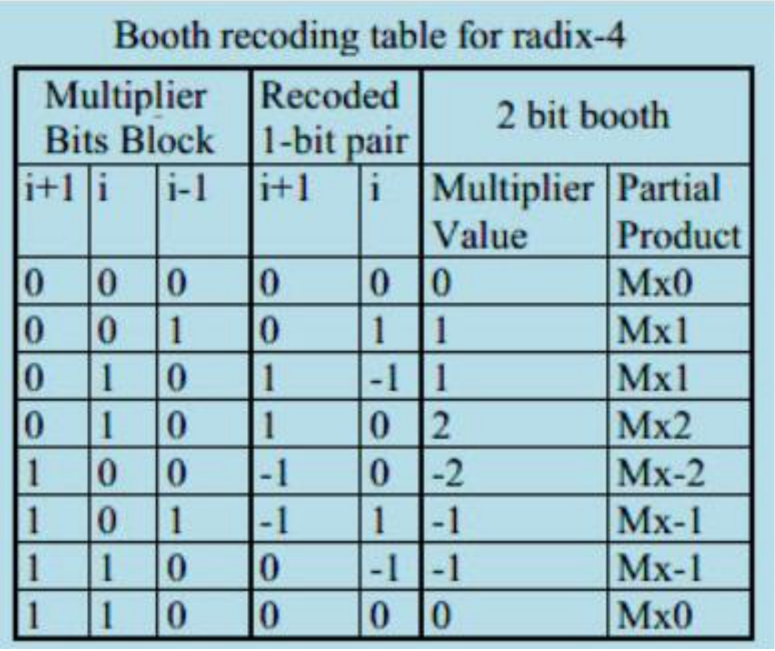
b_{2i+1}	b_{2i}	b_{2i-1}	neg_i	two_i	$zero_i$	*Partial product* P_{ij} *for* $a_j a_{j-1}$			
						00	01	10	11
0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	0	0	1	1
0	1	1	0	1	0	0	1	0	1
1	0	0	1	1	0	1	0	1	0
1	0	1	1	0	0	1	1	0	0
1	1	0	1	0	0	1	1	0	0
1	1	1	0	0	1	0	0	0	0

Table 1. Table representing the conversion of the multiplicand



					$\overline{s0}$	s0	s0	p07	p06	p05	p04	p03	p02	p01	p00
				1	$\overline{s1}$	p17	p16	p15	p14	p13	p12	p11	p10		cor0
		1	$\overline{s2}$	p27	p26	p25	p24	p23	p22	p21	p20		cor1		
	$\overline{s2}$	p37	p36	p35	p34	p33	p32	p31	p30		cor2				
									cor3						
fs15	fs14	fs13	fs12	fs11	fs10	fs9	fs8	fs7	fs6	fs5	fs4	fs3	fs2	fs1	fs0
	fc14	fc13	fc12	fc11	fc10	fc9	fc8	fc7	fc6	fc5	fc4	fc3			

Figure 5. Partial products generation in exact booth multiplier

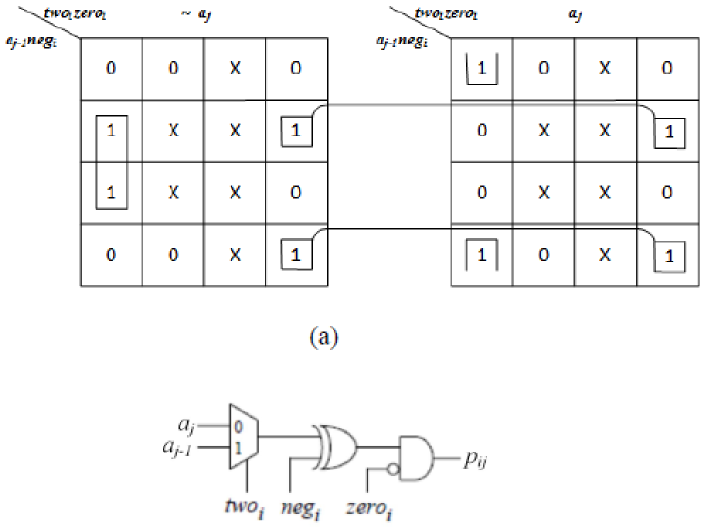


Booth recoding table for radix-4

Multiplier Bits Block			Recoded 1-bit pair		2 bit booth	
i+1	i	i-1	i+1	i	Multiplier Value	Partial Product
0	0	0	0	0	0	Mx0
0	0	1	0	1	1	Mx1
0	1	0	1	-1	1	Mx1
0	1	0	1	0	2	Mx2
1	0	0	-1	0	-2	Mx-2
1	0	1	-1	1	-1	Mx-1
1	1	0	0	-1	-1	Mx-1
1	1	0	0	0	0	Mx0

Table 2. Booth recoding table

A circuit using the input A and its subsequent Kmap is also shown and derived.



0	0	X	0
1	X	X	1
1	X	X	0
0	0	X	1

1	0	X	0
0	X	X	1
0	X	X	0
1	0	X	1

(a)

Figure 6. Modified Booth multiplier circuit and k map

Approximation in modified booth multiplier

This paper develops and tests an 8-bit approximation booth multiplier. The proposed approximation approach can also benefit large multipliers. Figure 5 shows how the bottom half of the partial product matrix is further divided into lower part major (LPmajor) and lower part minor (LPminor) using variable approximations. The sections that follow go through the approximation methodologies utilised for LPmajor and LPminor in detail.

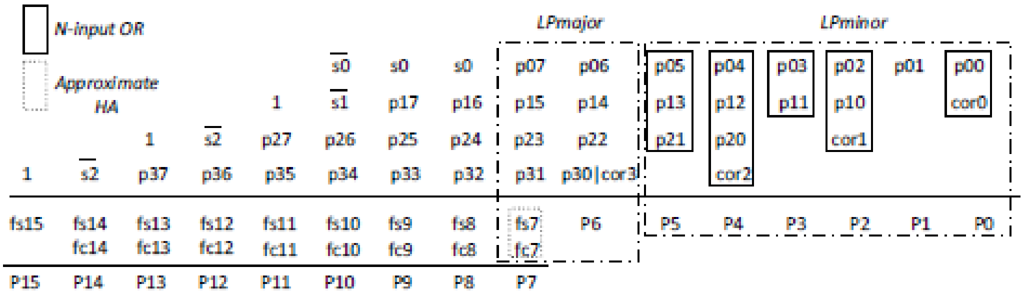
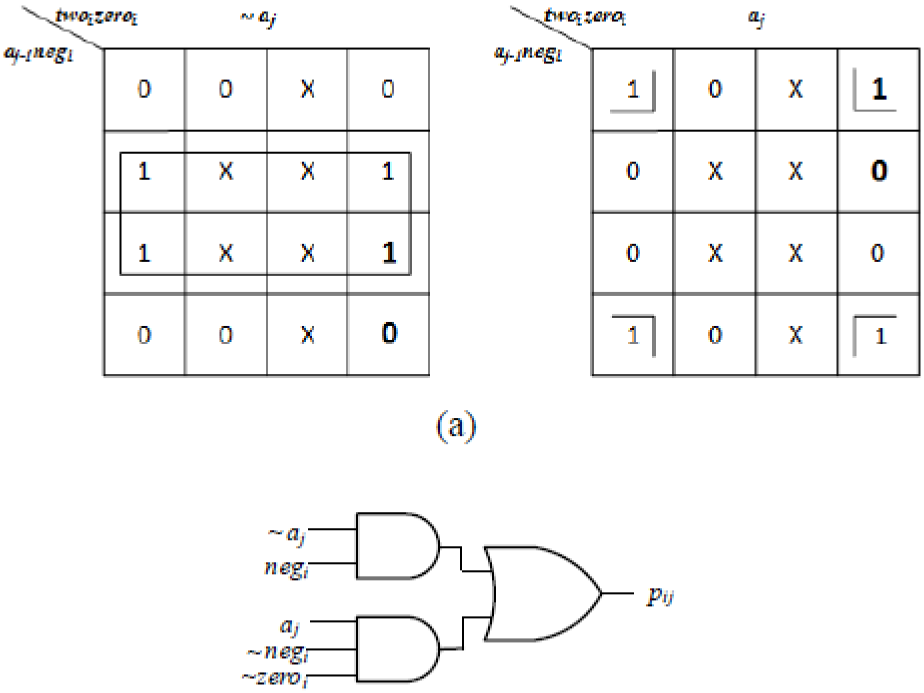

N-input OR
Approximate HA
LPmajor
LPminor
s0 s0 s0 p07 p06 p05 p04 p03 p02 p01 p00
1 s1 p17 p16 p15 p14 p13 p12 p11 p10 cor0
1 s2 p27 p26 p25 p24 p23 p22 p21 p20 cor1
1 s2 p37 p36 p35 p34 p33 p32 p31 p30|cor3 cor2
fs15 fs14 fs13 fs12 fs11 fs10 fs9 fs8 fs7 P6 P5 P4 P3 P2 P1 P0
fc14 fc13 fc12 fc11 fc10 fc9 fc8 fc7
P15 P14 P13 P12 P11 P10 P9 P8 P7

Figure 7. Partial products of Approximate booth multiplier

At both the partial product generation and partial product accumulation stages, an approximation is used. This paper proposes a unique method for simulating partial product manufacture. Figure 5 ((a)K-map) has been modified, as seen in Figure 7. (a). Four of the 32 occurrences were modified and highlighted, resulting in a considerable decrease in logic circuits. Making a mistake has a 1/8 chance of occurring.



0	0	X	0
1	X	X	1
1	X	X	**1**
0	0	X	**0**

1	0	X	**1**
0	X	X	**0**
0	X	X	0
1	0	X	1

(a)

Figure 8.Approximate circuit and k map

The exact partial product generating logic equation is as follows:

$$P_{ij} = \sim zero_i \left(neg_i(\sim a_j . \sim two_i + \sim a_{j-1} . two_i) \right. \\ \left. + \sim neg_i(a_j . \sim two_i + a_{j-1} . two_i) \right)$$

The logic equation is decreased after approximation, as seen in

$$P_{ij} = \sim a_j . neg_i + a_j . \sim neg_i . zero_i$$

The resulting values are gathered using OR gates after a fresh approximation in a generation, as seen in Figure 6. A similar strategy is used to collect the unfinished products. After generating partial approximation products in LPminor, the approximate partial products are aggregated column-wise using a variable input OR gate whose input is dictated by the number of components in each column.

Figure 6 displays an approximation in which the correction term is ORed with a minor partial product of the row to which it corresponds. The primary goal of this approximation is to maintain the alignment, which considerably improves the LPmajor's delay performance. Sum and carry rows are gathered and constructed using exact half-adders, full-adders, and 4-2 compressors. In LPmajor, LOA is used to add sum and carry.

3. Results

We completed four implementations in this report. In both behavioural and structural implementation, the same booth multiplier has been realised. Similarly, in behavioural and structural implementation, an approximate booth multiplier has been realised.

Exact Booth multiplier (Behavioural)

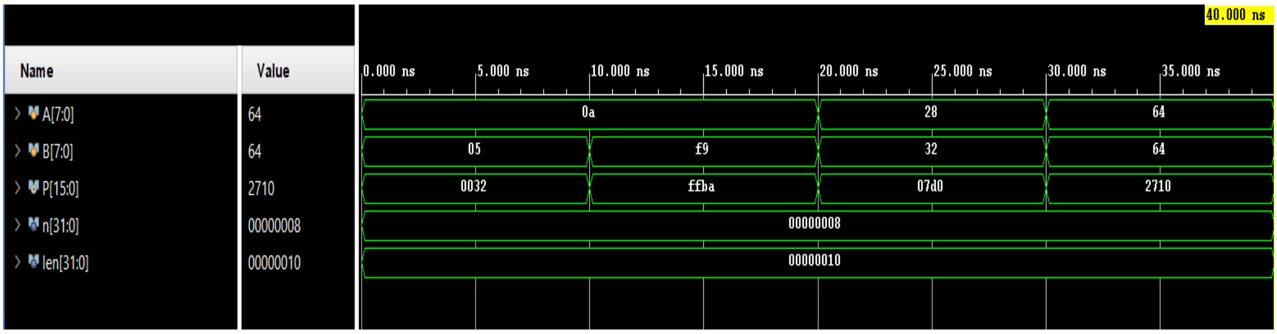


Figure 9.Exact booth multiplier test bench in the behavioural model.

Exact Booth multiplier(structured)

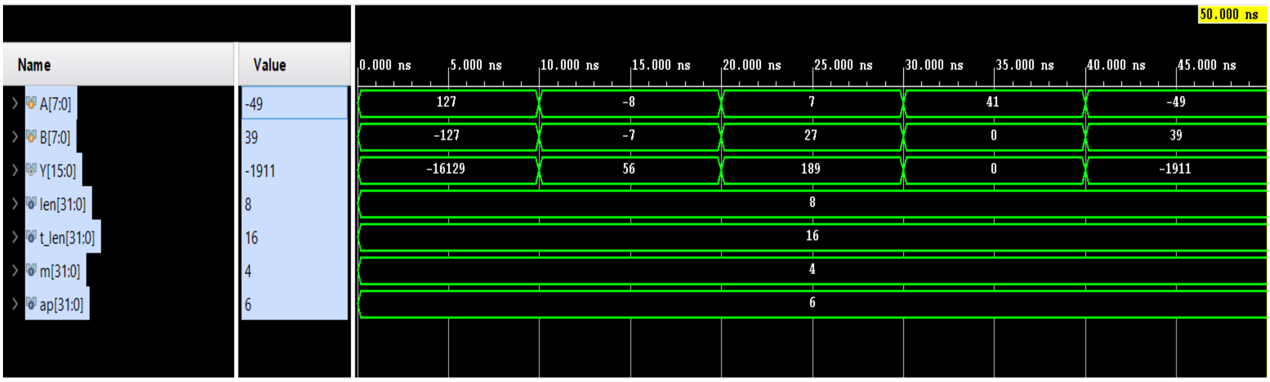


Figure 10. Exact booth multiplier simulation in the structured model.

Approximate Booth multiplier (Behavioural)

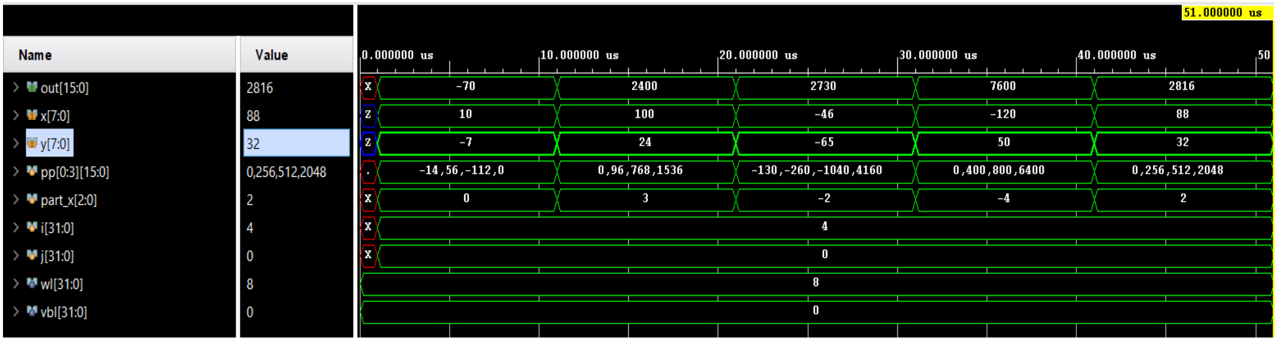


Figure 11. Approximate booth multiplier using Behavioral model

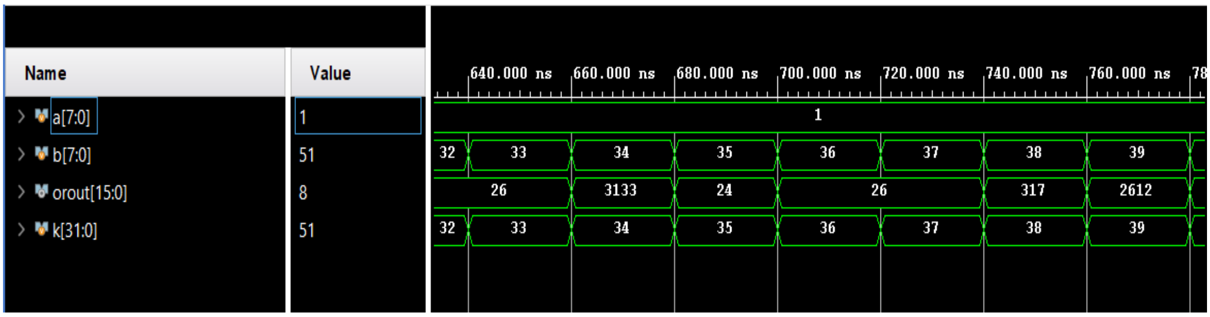


Figure 12. Approximate booth multiplier sing structural model

RTL Schematics

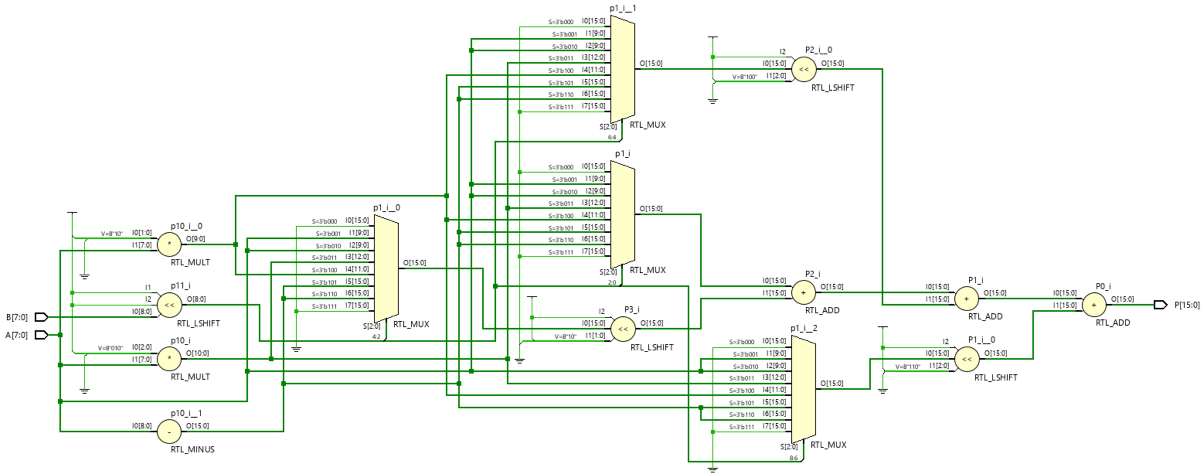


Figure 13. RTL of exact booth multiplier(behavioural)

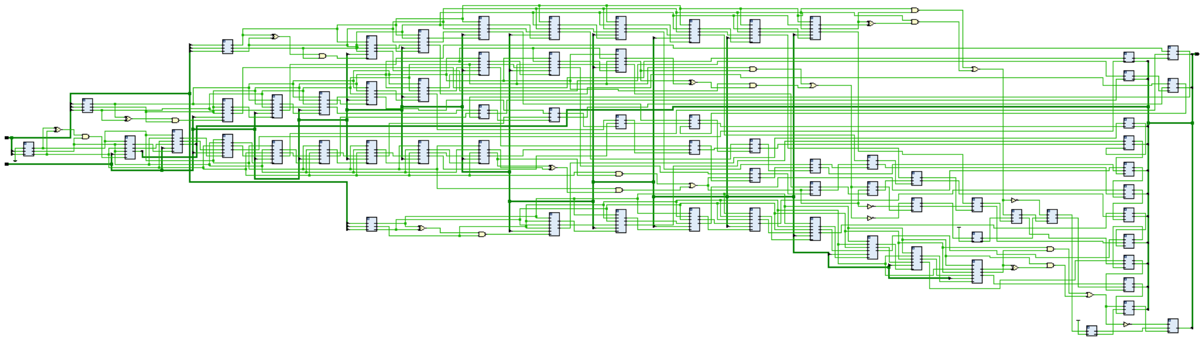


Figure 14.RTL of exact booth multiplier(structured)

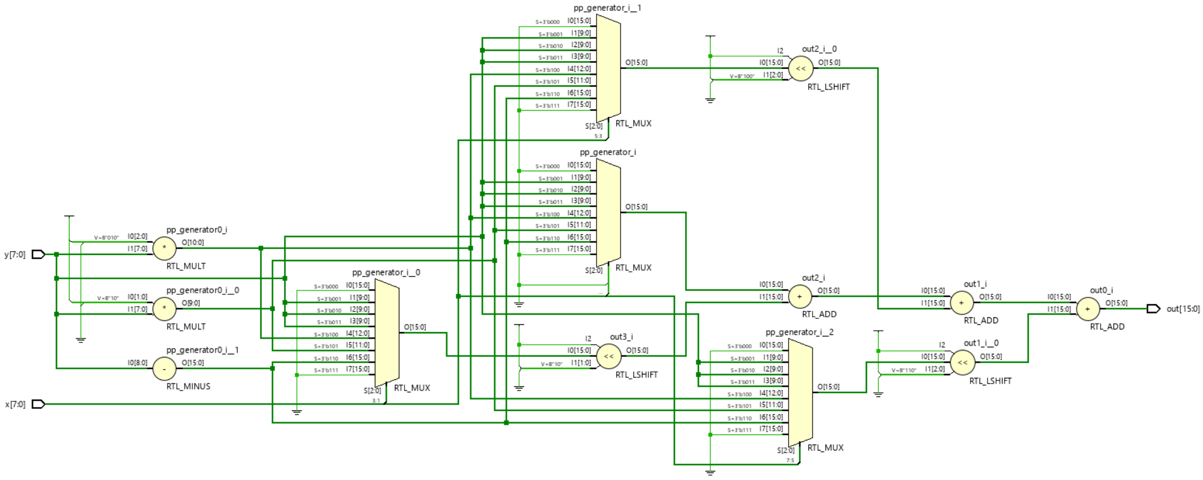


Figure 15. RTL of approximate booth multiplier using a behavioural model

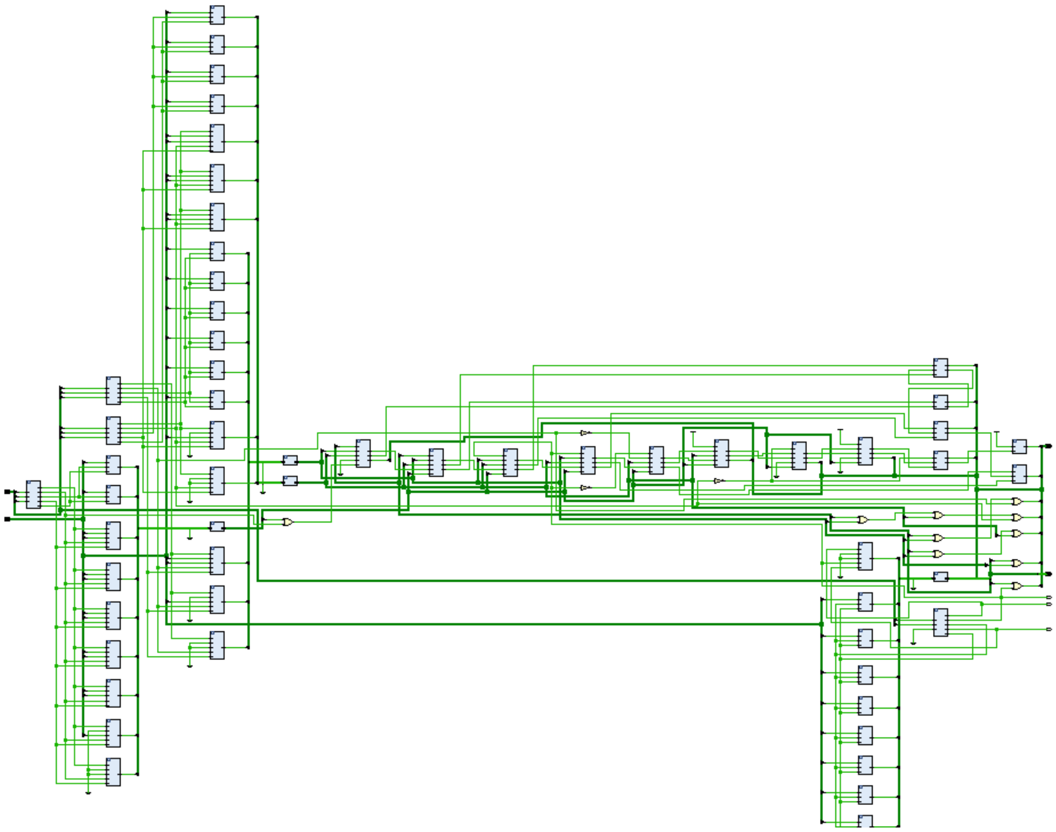
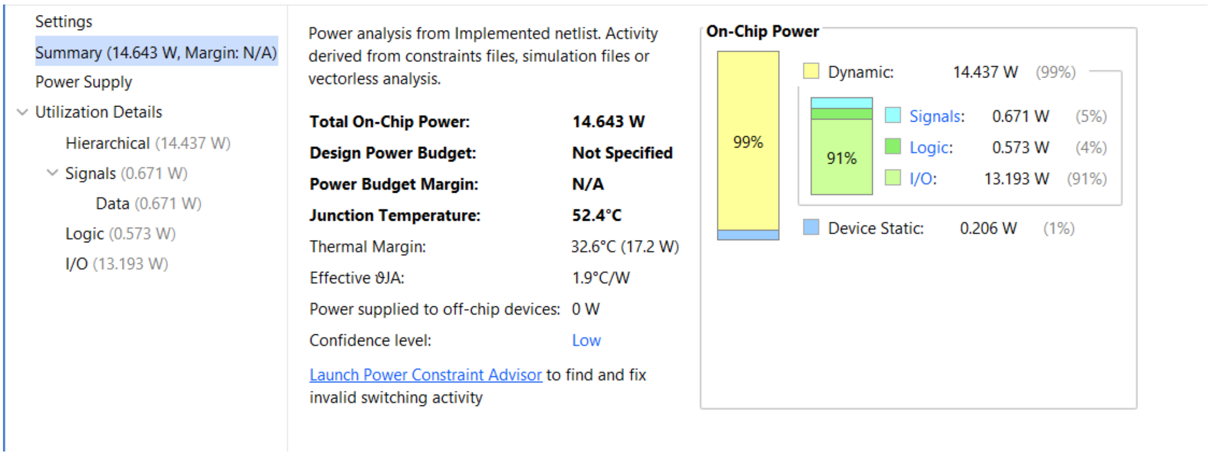


Figure 16.RTL of approximate booth multiplier using Structural

Power readings



Settings
Summary (14.643 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (14.437 W)
Signals (0.671 W)
Data (0.671 W)
Logic (0.573 W)
I/O (13.193 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

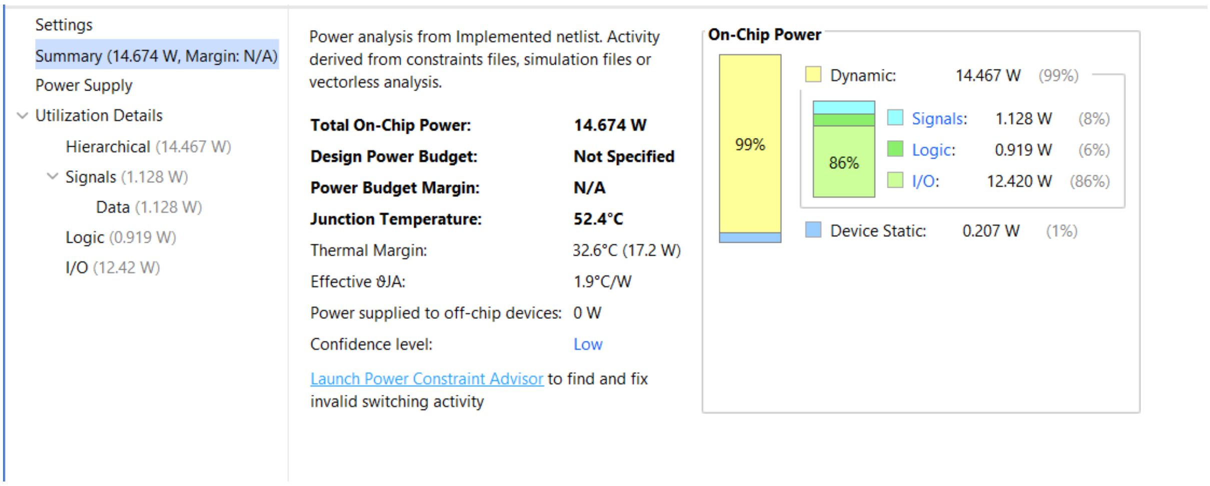
Total On-Chip Power:	**14.643 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**52.4°C**
Thermal Margin:	32.6°C (17.2 W)
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	14.437 W	(99%)
Signals:	0.671 W	(5%)
Logic:	0.573 W	(4%)
I/O:	13.193 W	(91%)
Device Static:	0.206 W	(1%)

Figure 17.Power of exact booth multiplier(behavioural)



Settings
Summary (14.674 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (14.467 W)
Signals (1.128 W)
Data (1.128 W)
Logic (0.919 W)
I/O (12.42 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

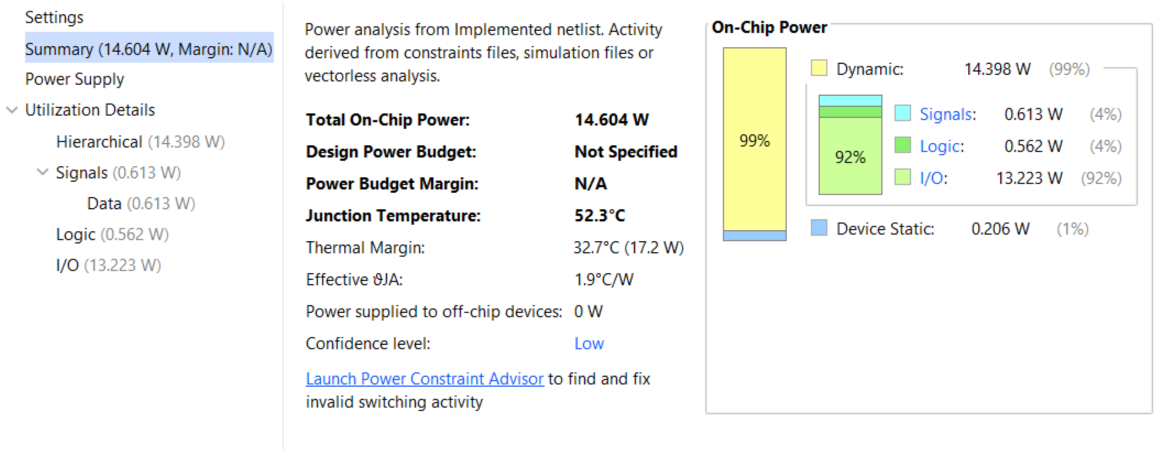
Total On-Chip Power:	**14.674 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**52.4°C**
Thermal Margin:	32.6°C (17.2 W)
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	14.467 W	(99%)
Signals:	1.128 W	(8%)
Logic:	0.919 W	(6%)
I/O:	12.420 W	(86%)
Device Static:	0.207 W	(1%)

Figure 18 Power of exact booth multiplier(structural)



Settings
Summary (14.604 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (14.398 W)
Signals (0.613 W)
Data (0.613 W)
Logic (0.562 W)
I/O (13.223 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

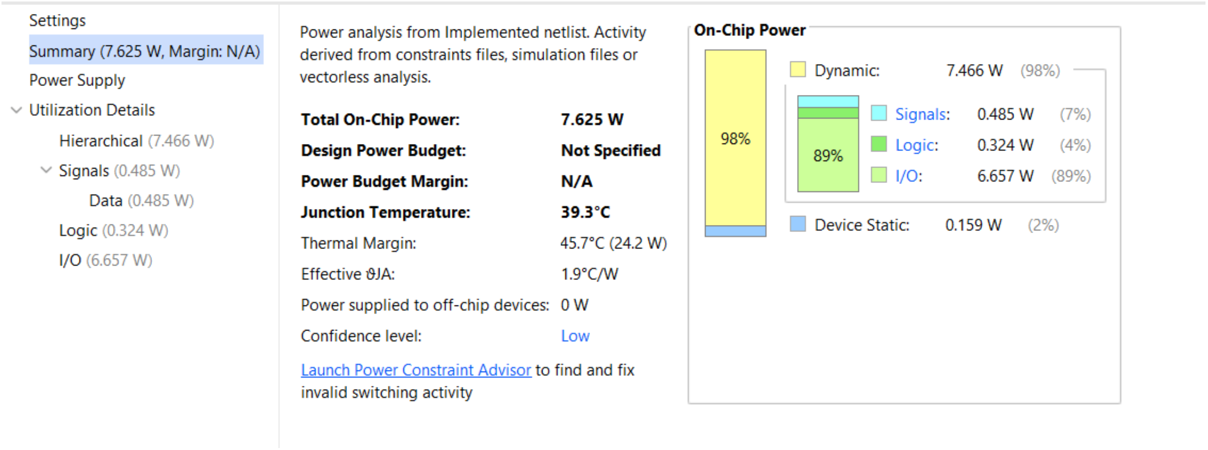
Total On-Chip Power:	**14.604 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**52.3°C**
Thermal Margin:	32.7°C (17.2 W)
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	14.398 W	(99%)
Signals:	0.613 W	(4%)
Logic:	0.562 W	(4%)
I/O:	13.223 W	(92%)
Device Static:	0.206 W	(1%)

Figure 19.Power of approximate booth multiplier using a behavioural model



Settings

Summary (7.625 W, Margin: N/A)

Power Supply

Utilization Details

- Hierarchical (7.466 W)
- Signals (0.485 W)
  - Data (0.485 W)
- Logic (0.324 W)
- I/O (6.657 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**7.625 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**39.3°C**
Thermal Margin:	45.7°C (24.2 W)
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		7.466 W	(98%)
	Signals:	0.485 W	(7%)
	Logic:	0.324 W	(4%)
	I/O:	6.657 W	(89%)
Device Static:		0.159 W	(2%)

Figure 20. Power of approximate booth multiplier using (structural)

Circuit	Power(mW)	Area (%)	Improvement
Exact booth multiplier (Behavioural)	671	0.18	-
Exact booth multiplier (Structural)	1128	0.07	61.11
Approximate booth multiplier (Behavioural)	613	0.05	72.22
Approximate booth multiplier(structural)	411	0.04	77.77

Table 3.Results

4. Conclusion

A least error approximation booth multiplier with a considerable area and power improvement over an exact equivalent multiplier was studied. Estimated circuits for partial product creation and accumulation are presented as an unique radix-4 approximation. The partial product generating circuit is a critical and energy-intensive phase in the booth multiplier. The precise partial product generating circuit is improved by reducing logic complexity and introducing an error probability of 0.125, yielding a high-performance circuit. OR gates may be utilised efficiently in partial product accumulation to conserve space and power while retaining accuracy.

When compared to an exact multiplier, the proposed model offers a substantial improvement in area and power. The design takes up less space and power than comparable cutting-edge products and has superior error metrics. To investigate the impact of approximation, a real-world application is chosen. In the future, the approximate multipliers will be tested in additional real-world application scenarios, and an error compensation block will be built to increase the proposed multiplier's accuracy even more.

REFERENCES

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," *2013 18th IEEE European Test* Symposium (ETS), Avignon, 2013, pp. 1-6.

[2] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie and C. Lucas, "BioInspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850-862, April 2010.

[3] K. Du, P. Varman and K. Mohanram, "High-performance, reliable variable latency carry select addition," *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2012, pp. 1257-1262.

[4] Ning Zhu, W. L. Goh and K. S. Yeo, "An enhanced low-power highspeed Adder For error-tolerant application," *Proceedings of the 2009 12th International Symposium on IC*, Singapore, 2009, pp. 69-72.

[5] A. Momeni, J. Han, P. Montuschi and F. Lombardi, "Design and Analysis of Approximate Compressors for Multiplication," in *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984-994, April 2015.

[6] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris and K. Pekmestzi, "Design-Efficient Approximate Multiplication Circuits Through Partial Product Perforation," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3105-3117, Oct. 2016.

[7] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance driven logic compression," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, Lausanne, 2017, pp. 7-12.

[8] R. Venkatesan, A. Agarwal, K. Roy and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, 2011, pp. 667-673.

[9] M. J. Schulte and E. E. Swartzlander, "Truncated multiplication with correction constant [for DSP]," *Proceedings of IEEE Workshop on VLSI Signal Processing*, Veldhoven, 1993, pp. 388-396 .

[10] Hong-An Huang, Yen-Chin Liao and Hsie-Chia Chang, "A self-compensation fixed-width booth multiplier and its 128-point FFT applications," *2006 IEEE International Symposium on Circuits and Systems*, Island of Kos, 2006, pp. 4 pp.-3541.

[11] 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," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 4, pp. 215-219, April 2011.

[12] H. Jiang, J. Han, F. Qiao and F. Lombardi, "Approximate Radix-8 Booth Multipliers for Low-Power and High-Performance Operation," in *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2638-2644, Aug. 1 2016.

[13] J. Liang, J. Han and F. Lombardi, "New Metrics for the Reliability of Approximate and Probabilistic Adders," in *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760-1771, Sept. 2013.

[14] E. de Angel and E. E. Swartzlander, "Low power parallel multipliers," *VLSI Signal Processing, IX*, San Francisco, CA, 1996, pp. 199-208.