FLOATING-POINT MULTIPLICATION IMPLEMENTED USING BINARY MULTIPLIER ARCHITECTURE

PANIKATLLA MAHEEN [1], M. IMTHIAZ BASHA[2]
[1]PG SCHOLAR.DEPT OF ECE, GATES INSTITUTE OF TECHNOLOGY, AP, INDIA
[2] ASSOCIATE PROFESSOR, DEPT OF ECE, GATES INSTITUTE OF TECHNOLOGY, AP, INDIA

Arithmetic logic units (ALUs) are core components of processing devices that perform required arithmetic and logical operations such as multiplication, division, addition, subtraction, and squaring. The multiplication operation is frequently used in ALUs in engineering applications such as signal processing, video processing and image processing for which floating-point multiplication is an important component. The dynamic range of numbers represented by floating-point arithmetic is very large compared with that of fixed-point numbers of the same bit width. A mantissa similarity investigator (MSI)– interfaced multi-precision binary multiplier architecture is developed and can be used in data-intensive applications that require variable precision, high throughput and low delay. This architecture can be configured to operate in single-, double-, quadruple- and octuple-precision modes for mantissa multiplication according to the IEEE 754 standard for floating-point numbers. The system produces increased throughput and utilizes mantissa similarity to reduce system delay.

INTRODUCTION:

Arithmetic logic units (ALUs) are the units that determine processor performance, as ALUs are responsible for executing all arithmetic, logical and other system operations. Among the arithmetic functions, multiplication is the one most frequently used. Multiplication forms the basis of many other complex arithmetic functions such as cubing, squaring and convolution. These operations depend on the binary multiplication method, which is just like the functions used in the decimal system. The methodology proceeds one bit at a time, with a partial product generated after each execution. As multiplicand input bits are consumed with multipliers, partial products are generated, and
all these partial products undergo summation to produce the complete product . In the case of two fractional binary numbers given as inputs to be multiplied, the binary point in
the final product is decided as in the case of the fractional form of decimal numbers. When multiplication is performed by 1 bit, bits are shifted towards the left by one bit, and in such a way, the entire multiplication is completed. Finally, by adding the intermediate results, the result is obtained.

Although computer arithmetic is sometimes viewed as a specialized part of CPU design, still the discrete component designing is also a very important aspect. A tremendous variety of algorithms have been proposed for use in floating-point systems. Actual implementations are usually based on refinements and variations of the few basic algorithms presented here. In addition to choosing algorithms for addition, subtraction, multiplication, and division, the computer architect must make other choices. Our discussion of floating point will focus almost exclusively on the IEEE floating-point standard (IEEE 754) because of its rapidly increasing acceptance. Although floating-point arithmetic involves manipulating exponents and shifting fractions, the bulk of the time in floating-point operations is spent operating on fractions using integer algorithms. Thus, after our discussion of floating point, we will take a more detailed look at efficient algorithms and architectures.

LITERATURE SURVEY

Design and implementation of fast floating point multiplier unit by Sunesh, N.V., Sathishkumar, P.:

Floating point numbers are the quantities that cannot be represented by integers, either because they contain fractional values or because they lie outside the range re presentable within the system's bit width. Multiplication of two floating point numbers is very important for processors. Architecture for a fast floating point multiplier yielding with the single precision IEEE 754-2008 standard has been used in this project. The floating point representation can preserve the resolution and accuracy compared to fixed point. Pipeline is a technique where multiple instructions are overlapped in execution. Multiple operations performed at the same time by pipeline will increase the instruction throughput. In several high performance computing systems such as digital signal processors, FIR filters, microprocessors, etc multipliers are key components. The most important aim of the design is to make the multiplier quicker by decreasing delay. Decrease of delay can be caused by propagation of carry in the adders having smallest amount power delay constant.

Modified binary multiplier architecture to achieve reduced latency and hardware utilisaiton. By Tomar, G.S., George, M.:

Arithmetic Logic Units (ALUs) are very important components of the processor, which performs various arithmetic and logical operations such as multiplication, division, addition, subtraction, cubing, squaring, etc. Of these all operations, multiplication is most elementary and most frequently used operation in the ALUs. The operation of multiplication also forms the basis of many other complex arithmetic operations such as cubing, squaring, convolution, etc. This paper presents the modified novel multi-precision binary multiplier architecture to achieve a reduced latency/delay and area/hardware utilization along with existing implementations of binary multiplication. This system will function as second stage of the of a novel multi-precision binary multiplier system. The system was implemented using Xilinx 14.2 ISE and simulated with ISIM which was available from Xilinx 14.2 ISE. The results of simulation indicate that the latency of the proposed novel binary multiplier systems (8-bit, 16-bit and 24-bit) with significantly shorter than existing implementations

Study of various high speed multipliers. By Abraham, S., Kaur, S., Singh, S

Multiplication or repeated addition is the basic operation used in both Mathematics and Science. The speed of multiplier determines the speed of all Digital Signal Processors. This paper describes four multipliers that is Modified Booth Multiplier, Vedic Multiplier (Urdhva Tiryakbhyam Sutra), Wallace Multiplier and Dadda Multiplier. In Modified Booth Multiplier Algorithm `0' is appended to the right of LSB and then three bits starting from the LSB are grouped as a set to decide the partial product. So as to increase its speed, desk calculators are used that perform the operation of shifting faster. Urdhva Tiryakbhyam Sutra is performed by two multiplication techniques that is straight above multiplication and diagonal multiplication. Then finally, their sum is taken. Here reduction of multi bit multiplication to single bit multiplication takes place followed by the process of addition. Since there is only one step generation of partial product, carry propagation from LSB to MSB is reduced. There are three steps in which both Wallace and Dadda Multiplier work. They are partial product formation, reduction of partial products formed to two rows and addition of these two rows using carry propagation adder. Wallace Multiplier reduces all possible partial products and so has a smaller carry propagating adder whereas Dadda Multiplier performs only minimum necessary reduction and thus has a larger carry propagating adder. But as far as speed is concerned Dadda Multiplier is faster than Wallace Multiplier.

FPGA implementation of vedic floating point multiplier. By Kodali, R.K., Boppana, L., Sai Sourabh Yenamachintala

Most of the scientific operation involve floating point computations. It is necessary to implement faster multipliers occupying less area and consuming less power. Multipliers play a critical role in any digital design. Even though various multiplication algorithms have been in use, the performance of Vedic multipliers has not drawn a wider attention. Vedic mathematics involves application of 16 sutras or algorithms.

One among these, the Urdhva tiryakbhyam sutra for multiplication has been considered in this work. An IEEE-754 based Vedic multiplier has been developed to carry out both single precision and double precision format floating point operations and its performance has been compared with Booth and Karatsuba based floating point multipliers. Xilinx FPGA has been made use of while implementing these algorithms and a resource utilization and timing performance based comparison has also been made.

Analysis of an efficient partial product reduction technique by Vyas, K., et al.

Most of the scientific operation involve floating point computations. It is necessary to implement faster multipliers occupying less area and consuming less power. Multipliers play a critical role in any digital design. Even though various multiplication algorithms have been in use, the performance of Vedic multipliers has not drawn a wider attention. Vedic mathematics involves application of 16 sutras or algorithms.

One among these, the Urdhva tiryakbhyam sutra for multiplication has been considered in this work. An IEEE-754 based Vedic multiplier has been developed to carry out both single precision and double precision format floating point operations and its performance has been compared with Booth and Karatsuba based floating point multipliers. Xilinx FPGA has been made use of while implementing these algorithms and a resource utilization and timing performance based comparison has also been made.

3. EXISTING SYSTEM

Need of fast multipliers:

Multiplication is one of the most important arithmetic operation. A multiplier is one of the key hardware in most digital processing systems. Speed of any digital system operation depends upon the speed of multiplier. With advancement in technology many researchers have tried to design multipliers which offers high speed.

Array multiplier: Array multiplier is well known due to it"s regular structure. Multiplier circuit is based on add and shift algorithms. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added. The addition can be performed with normal carry propagate adder. N-1 adders are required when N is the multiplier length.

Vedic mathematics: Vedic mathematics is an ancient Indian technique developed by shankaracharya shri bharti Krishna tirthaji maharaj. The word Vedic is derived from Sanskrit word „Veda" which means store house of all knowledge. Vedic mathematics is based on sixteen sutras (formulae), out of the sixteen sutra "urdhwa-triyagbhyam" sutra is used for multiplication.

Urdhwa-triyagbhyam sutra: sutra " urdhwa-triyagbhyam" sutra is used for multiplication. It is a general formula of multiplication. It is applicable to all the cases of multiplication. Urdhwa means vertical and triyagbhyam means crosswise. Hence it is also called as vertical and crosswise method

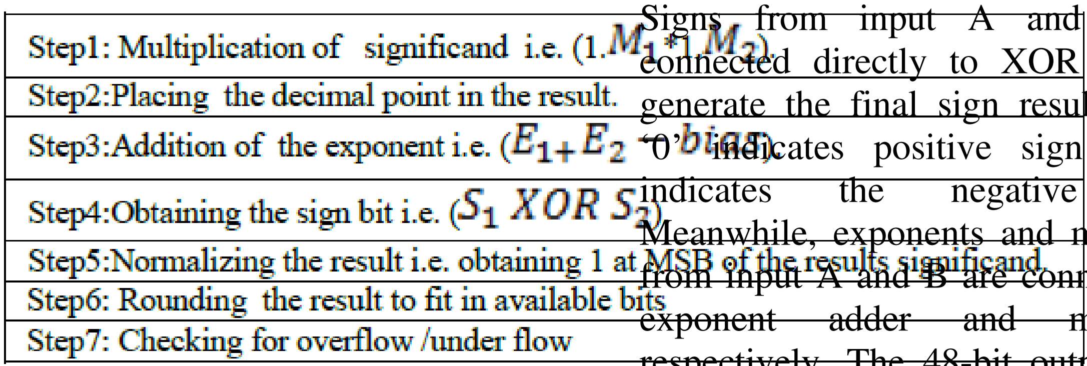
Floating Point Multiplication

Floating point multiplication algorithm: Floating point multiplication process can be divided in to four units; mantissa calculation unit, exponent calculation unit, sign calculation unit and normaliser unit . Normalised floating point number have the form of

$$Z = (-1^{s}) * 2^{(E-bias)} * (1.M).$$

.

To perform multiplication of two single precision floating point numbers following steps are followed:



Step1: Multiplication of significand i.e. $(1.M_1 * 1.M_2)$.
Step2:Placing the decimal point in the result.
Step3:Addition of the exponent i.e. $(E_1 + E_2 - bias)$
Step4:Obtaining the sign bit i.e. $(S_1\ XOR\ S_2)$
Step5:Normalizing the result i.e. obtaining 1 at MSB of the results significand.
Step6: Rounding the result to fit in available bits
Step7: Checking for overflow /under flow

Mantissa Multiplication : Overall performance of designed multiplier depends upon the performance of mantissa multiplier unit faster the multiplication unit faster is the overall calculation. Mantissa multiplication unit is designed using Vedic multiplication technique. "Urdhwa-triyagbhyam" sutra is used for multiplication. 3by3 multiplier is used as basic multiplier. Vedic mathematics technique improves the performance of multiplier unit in terms of speed and power

5.DESIGN AND IMPLEMENTATION

A multiplication of two floating-point numbers is
done in the following 5 steps:
Step 1: Multiplication of mantissas
Step 2: Normalization
Step 3: Addition of the exponents
Step 4: Calculation of the sign
Step 5: Composition of all results
described the 32-bit floating point multiplier data process flow. Each input is split into
three modules (sign, exponent, and mantissa) so that can be easily to route into corresponding components. Signs from input A and B are connected directly to XOR gate to generate the final sign result, either '0' indicates positive sign or '1' indicates the negative sign. Meanwhile, exponents and mantissas from input A and B are connected to exponent adder and multiplier respectively. The 48-bit output from multiplier must pass through to the normalizer to perform rounding to nearest 23-bit of mantissa. In

exponent adder, both exponents from A and B are added before subtract to bias value which is 127. The carry signal from normalizer is also connected to exponent adder to adjust the exponent value, which will be the final 8-bit exponent result. All the output from signer (1-bit), exponent adder (8-bit), and normalizer (23-bit) are then combined to form 32-bit floating point multiplication product as the final results.

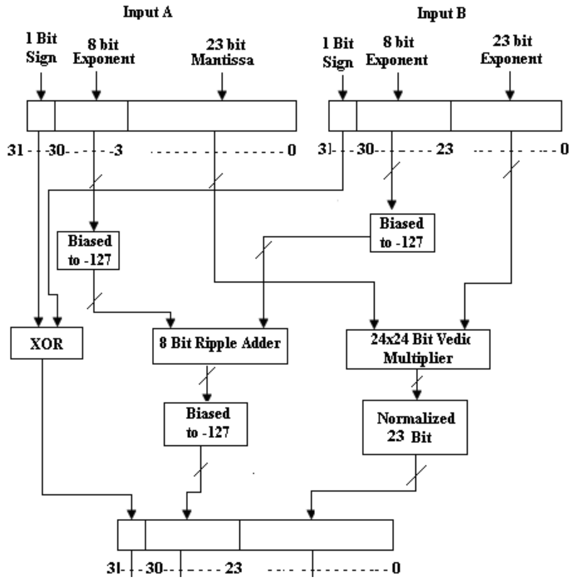


Figure 2: Block diagram of Floating Point multiplier

Multiplication is an important fundamental function in arithmetic operation.Signed multiplication is a careful process. With unsigned multiplication there is no need to take the sign of the number into consideration. However in signed multiplication the same process cannot be applied because the signed number is in a 2's compliment form which would yield an incorrect result if multiplied in a similar fashion to unsignedmultiplication.That's where Booth's algorithm comes in.Booth's algorithm preservesthe sign of the result.

Booth's algorithm is a well known method for 2's complement multiplication. It speeds up the process by analyzing multiple bits of multiplier at a time. This widely used scheme for two's complement multiplication was designed by Andrew D. Booth in 1951. Booth algorithm is an elegant way for this type of multiplication which treats both positive and negative operands uniformly. It allows nbit multiplication to be done using fewer than n additions or subtractions, thereby making possible faster multiplication. Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation.

There are 2 methods that you should know before attempting Booth's algorithm. Rightshiftcirculant and right-shift arithmetic.

Right-shift circulant (**RSC**), is simply shifting the bit, in a binary string, tothe right 1 bit position and take the last bit in the string and append it to the beginning ofthe string.

Example:

10110

after right-shift circulant now equals – 01011

Right-shift arithmetic (RSA), is where you add 2 binary number togetherand shift the result to the right 1 bit position.
Example:
0100
+0110
result = 1010
Now shift all bits right and put the first bit of the result at the beginning of the new
string:
result 1010
shift 11010
According to Booth's multiplication algorithm among the two input binary numbers the one with minimum number of bit changes is considered as multiplier and the other as a multiplicand in order to reduce the time taken for calculating the multiplication product.

The steps for performing booth multiplication are as follows:

Let the multiplicand be 'B' and multiplier be 'Q'.
Assume initially value of 'A' and 'Q-1' is zero.
The main step is to check last two bits.
There will be iterations according to the number of multiplier.
For example, if the multiplier is of 2-bit then 2 Iterations will be done, for 4-bit multiplier 4 iterations are done, and so on.
Now, the algorithm starts, first the last two digits are checked and if the two bits are "00" or "11" the only Arithmetic Right Shift is done.
And if the last two bits are "01", then A is added with B, and result is stored into A.
If the last two bits are "10", then A is subtracted from B, and result is stored into A.
Finally, the result obtained is coded in binary form which gives the desired output.
In this way multiplication of any two numbers is performed using booth algorithm.

Mantissa Calculation Unit

The performance of Mantissa calculation Unit dominates overall performance of the Floating Point Multiplier. The Vedic Multiplication technique is chosen for the implementation of this unit. This technique gives result in terms of speed and power and for single precision multiplier 3*3 bit multiplier designed as a basic multiplier.

Exponent Subtractor

The exponent subtractor is used for exponent comparison and can be implemented as an adder. An fixed-point adder has been extensively studied and can be used in the exponent adder. Adders such as lower-part-OR adders (LOA), approximate mirror adders, approximate XOR/XNOR-based adders, and equal segmentation adders can be found in the literature. For a fast FP adder, a revised LOA adder is used, because it significantly reduces the critical path by ignoring the lower carry bits.

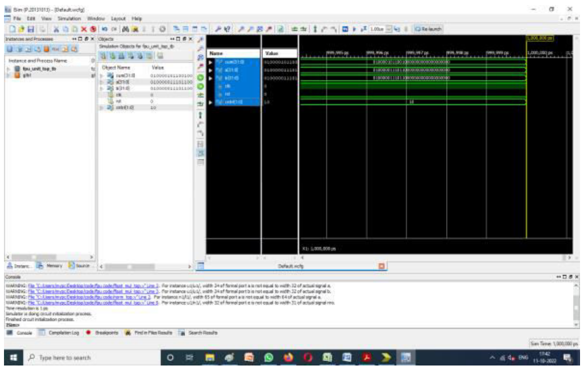
Normalizer The result of the

significand multiplication (intermediate product) must be normalized to have a leading „1" just to the left of the decimal point (i.e. in the bit 46 in the intermediate product). Since the inputs are normalized numbers then the intermediate product has the leading one at bit 46 or 47. If the leading one is at bit 46 (i.e. to the left of the decimal point) then the intermediate product is already a normalized number and no shift is needed. If the leading one is at bit 47 then the intermediate product is shifted to the right and the exponent is incremented by 1.

Overflow/underflow Detection

Overflow/underflow means that the result"s exponent is too large/small to be represented in the exponent field. The exponent of the result must be 8 bits in size, and must be between 1 and 254 otherwise the value is not a normalized one .An overflow may occur while adding the two exponents or during normalization. Overflow due to exponent addition maybe compensated during subtraction of the bias; resulting in a normal output value (normal operation). An underflow may occur while subtracting the bias to form the intermediate exponent. If the intermediate exponent < 0 then it"s an underflow that can never be compensated; if the intermediate exponent = 0 then it"s an underflow that may be compensated during normalization by adding 1 to it .When an overflow occurs an overflow flag signal goes high and the result turns to ±Infinity (sign determined according to the sign of the floating point multiplier inputs). When an underflow occurs an underflow flag signal goes high and the result turns to ±Zero (sign determined according to the sign of the floating point multiplier inputs). Denormalized numbers are signaled to zero with the appropriate sign calculated from the inputs and an underflow flag is raised.

6. RESULT



```
WARNING: File "C:/Users/mypc/Desktop/code/fpu code/float_mul_top.v" Line 2. For instance u1/u1/, width 24 of formal port a is not equal to width 32 of actual signal a.
WARNING: File "C:/Users/mypc/Desktop/code/fpu code/float_mul_top.v" Line 3. For instance u1/u1/, width 24 of formal port b is not equal to width 32 of actual signal b.
WARNING: File "C:/Users/mypc/Desktop/code/fpu code/norm_top.v" Line 2. For instance n1/i1/, width 65 of formal port a is not equal to width 64 of actual signal a.
WARNING: File "C:/Users/mypc/Desktop/code/fpu code/float_mul_top.v" Line 5. For instance u1/n1/, width 32 of formal port c is not equal to width 31 of actual signal mo.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

7. CONCLUSION & FUTURE SCOPE

This project presented the implementation, verification and validation of a novel MSI-interfaced multi-precision binary multiplier architecture for use in implementing a multi-precision floating-point multiplier. No existing system that has previously been reviewed can be configured to operate in single-, double-, quadruple- and octuple-precision modes. The architecture developed in this chapter can operate in all four precision modes. This makes the system very useful and applicable to high-bandwidth and data-intensive operations with strict time constraints. Finally, the system developed in this chapter without the use of MSI performs better than other existing systems where latency is concerned. However, the development of the MSI unit and its incorporation into the novel MSI-interfaced multi-precision binary multiplier architecture has resulted in a system with latency that is significantly shorter than that of other existing binary multiplier systems. As such, the system caters to multi-precision, has shorter path delay and has higher throughput than existing systems. This novel MSI-interfaced multi-precision binary multiplier architecture can especially benefit computer systems in increasing the execution speed of arithmetic operations. The proposed system was compared with existing implementations of 24-, 53-, 113- and 237-bit binary multipliers, which represents the mantissa multiplication at various precision levels. The comparisons were made in terms of path delay throughput and the precision range for the computations performed. The novel MSI-interfaced multi-precision binary multiplier architecture achieved shorter path delay than its counterparts. It consists of 4 modules, i.e. Booth Radix-4 mantissa multiplier, normalizer, exponent adder and the signer. Booth radix-4 multiplication is one of the suitable algorithm to be used to design the high speed 24-bit mantissas multiplier because this algorithm is much simpler than the complex Wallace Tree multiplier, thus less gate delay and able to perform such complex multiplication faster. In addition, Booth radix-4 performance is doubled compared to Booth radix-2 that allows high speed multiplication can be achieved.

8. FUTURESCOPE:

Therefore the improved fused floating-point add–subtract unit will contribute to the next generation floating-point arithmetic and DSP application development. The proposed fused floating-point add–subtract unit takes two normalized floating-point operands and generates their sum and difference simultaneously. This design unit is also used for development of multiplexer. This is also useful for high speed design units.

REFERENCES

1. Sunesh, N.V., Sathishkumar, P.: Design and implementation of fast floating point multiplier unit. Proc. International Conference on VLSI Systems, Architecture, Technology and Applications, PLACE, pp. 1–5. (2015)

2. Tomar, G.S., George, M.: Modified binary multiplier architecture to achieve reduced latency and hardware utilisaiton. Wireless Personal

Communication. 98(4), 3549–3561 (2018)

3. Abraham, S., Kaur, S., Singh, S.: Study of various high speed multipliers. 2015 International Conference on Computer Communication and Informatics (ICCCI), pp. 1–5 (2015). https://doi.org/10.1109/ICCCI. 2015.7218139

4. Kodali, R.K., Boppana, L., Sai Sourabh Yenamachintala: FPGA implementation of vedic floating point multiplier. 2015 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES), pp. 1–4 (2015). https://doi.org/10.1109/ SPICES.2015.7091534

5. Vyas, K., et al.: Analysis of an efficient partial product reduction technique. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), pp. 1–6 (2015). https://doi.org/10.1109/ICGCIoT.201 5.7380417

6. Anitha, P., Ramanathan, P.: A new hybrid Multiplieusing Dadda and Wallace method. 2014 International Conference on Electronics and Communication Systems (ICECS), pp. 1–4 (2014). https://doi.org/ 10.1109/ECS.2014.6892623

7. Chopade, S.S., Mehta, R.: Performance analysis of vedic multiplication technique using FPGA. 2015 IEEE Bombay Section Symposium (IBSS), September 2015, Mumbai, pp. 1–6 (2015). https://doi.org/10.1109/IBSS.2015.74 56657

8. Bisoyi, A., Baral, M., Kumar Senapati, M.: Comparison of a 32-bit vedic multiplier with a conventional binary multiplier. 2014 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), pp. 1757–1760. (2014). https://doi.org/ 10.1109/ICACCCT.2014.7019410

9. Mhaidat, K.M., Abdulmughni, Y.H.: A new efficient reduction scheme to implement tree multipliers on FPGAs. 2014 9th International Design and Test Symposium, pp. 180–184 (2014). https://doi.org/10.1109/IDT.2014.703 8609

10. Bathija, R.K., et al.: Low power high speed 16x16 bit multiplier using vedic mathematics. Int. J. Compu. Appl. 59(6), pp. 41-44 (2012)

11. Rao, J.M., Dubey, S.: A high speed and area efficient Booth Recoded Wallace tree multiplier for fast arithmetic circuits. 2012 Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PRIMEASIA), pp. 220–223 (2012)

12. Jain, A., et al.: FPGA design of a fast 32-bit floating point multiplier unit. International Conference on Devices, Circuits and Systems (ICDCS), pp. 545–547 (2012)

13. Su, A., Sharma, R.K.: An efficient binary multiplier design for high

speed applications using Karatsuba algorithm and Urdhva-Tiryagbhyam algorithm. 2015 Global Conference on Communication Technologies (GCCT), pp. 192–196. (2015). https://doi.org/10.1109/GCCT.2015. 7342650

14. Gokhale, G.R., Bahirgonde, P.D.: Design of vedic-multiplier using areaefficient carry select adder. 2015 International conference on Advances in computing, Communications and Informatics (ICACCI), pp. 576–581 (2015). https://doi.org/10.1109/ICACCI.2015. 7275671