Multi-precision binary multiplier architecture for multi-precision floating-point multiplication

Geetam Singh Tomar | Marcus Llyode George | Abhineet Singh Tomar

1Rajkiya Engineering College, Sonbhadra, India
2Department of Electrical and Computer Engineering, University of West Indies, St. Augustine, Trinidad and Tobago
3Volvo Group Trucks Technology, Gothenburg, Sweden

Correspondence
Geetam Singh Tomar, Birla Institute of Applied Sciences, Bhimtal, India.
Email: gstomar@ieee.org

Abstract
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 utilises mantissa similarity to reduce system delay. The system was synthesised for a variety of field-programmable gate array targets using Xilinx ISE Design Suite 14.7, and performance was simulated using that suite’s ISim simulator.

1 | 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 [1, 2]. 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.

2 | LITERATURE SURVEY

In [3, 4], the authors have reported a study on five binary multipliers of high speed: ‘Booth Multiplier, Modified Booth Multiplier, Vedic Multiplier, Wallace Multiplier and Dadda Multiplier’. For most of those, the cost reportedly increases when the number of multiplier bits is increased, and operating speed declines in response to more partial products. However, the gaps in speed and cost have not been analysed in that study. In [5], the authors have reported on an efficient methodology for partial product reduction in binary multipliers and claimed that the same was designed with 16 nm TSMH CMOS technology using the simulation tool Tanner EDA 14.1. The authors have also compared the various available methods of other multipliers. In [6], the authors proposed an 8 x 8 hybrid tree multiplier system to show the speed of operation, but the system was not justified numerically, which remains a gap in the idea and leads to some confusion in conceptual design. The system has reportedly been implemented on the DSCH2 tool and simulated on MICROWIND using 0.25 um technology. In [7], the authors analysed the performance of Vedic
multiplication techniques reported to have been implemented using FPGA. Generation of the technology used was not specified, which makes such a study and its proposals a bit unjustified. However, the implementations suggested were in accordance with the concepts put forth. The authors also reportedly simulated a high-speed multiplier based on Vedic mathematics and further claimed to have compared it with conventional binary multipliers considering 8-, 16- and 32-bit numbers. However, no discussion was offered in support of that assertion. In [8], the authors have reported comparison of 32-bit Vedic multiplication with conventional binary multipliers implemented on Xilinx Nexys3 Spartan-3 FPGA and simulated using the Xilinx simulation platform. In [9], the authors have presented different implementations of a reduction scheme and have implemented tree multipliers by simulation followed by implementation on FPGA platforms. The system implemented was not a binary multiplier system but a reduction scheme for partial product reduction up to a 32-bit multiplier scheme implemented in Verilog in the Xilinx ISE Suite and targeted on the Xilinx Spartan-6 platform as hardware. In [10], the authors have reported implementation of a multiplier with a low power and high speed with a word size of 16 bits. This was reported to have been a binary multiplier using Vedic mathematics. The design started with the construction of a 2 x 2 multiplier block that was used to construct a 4 x 4 multiplier block, after which an 8 x 8 multiplier block was constructed. The required 16 x 16 multiplier block was constructed from the 8 x 8 multiplier blocks. In [11], the authors reportedly implemented a high-speed, area-efficient 16-bit Vedic multiplier and 32-bit Booth-recoded Wallace tree multiplier that is proposed for use in implementing arithmetic circuits. The system was implemented in Verilog HDL and synthesised for the Xilinx Virtex-6 FPGA. In [12], the authors have reported on multiplier systems with calculations of path delay and reported that the systems were implemented and results were displayed as path delays of 13.45 and 11.57 ns. However, the hardware utilisation was not stated. In [13], the authors have reported on the design of a 24-bit binary multiplier that was thereafter used to implement a 32-bit floating-point multiplier system. It was also stated that Vedic mathematics were used to implement this design, though there were some limiting observations in the criteria. The authors have also reported an efficient arithmetic strategy for unsigned binary multiplication that was designed to improve the implementation in terms of path delay and area. The system utilised a combination of the Karatsuba and Urdhva tiryagbyham algorithms to implement the required system. In [14], the authors have reportedly designed an area-efficient multiplier using a design of modified carry select adders, which was based on crosswise and vertical Vedic multiplier algorithms. This modified CSLA design was then reported to have been used in implementing the proposed 8-bit Vedic multiplier. The authors in [15] have proposed to have designed a high-speed 32-bit multiplier architecture based on Vedic mathematics and claim to have implemented this system by adjustment of the partial products using a concatenation approach. The authors in [15] reportedly developed area-efficient 8 x 8 and 16 x 16 multiplier systems using Vedic mathematics to improve performance. The system was reportedly implemented in Verilog HDL and synthesised on Xilinx ISE 12.2 for the target device, Spartan-3E, XC3S500-5FG320.

3 | BACKGROUND OF THE RESEARCH

Most of the multiplier systems reviewed in this paper carried out the processes of partial product generation, partial product storage and partial product reduction. For example, the multipliers developed in [2–4] perform partial product reduction using Wallace or Dadda multipliers, and thereafter, compressors are used to compress the results. Another design in [5] uses a combination of multiplier and compressor techniques to perform the partial product reduction segment. Most of the existing systems reviewed utilised Vedic mathematics for partial product generation. The multiplier systems in [14, 16, 17], for instance, developed Vedic multipliers by utilising smaller multipliers as building blocks to develop bigger multipliers. For instance, the construction of a 2 x 2 multiplier block was used to construct a 4 x 4 multiplier block, after which an 8 x 8 multiplier block was constructed. Some multiplier systems, such as that in [11], concurrently added the partial products during the multiplication operation, hence reducing delay at the expense of hardware utilisation. Others, like [18], added the partial products as they were generated to reduce demands on memory for the storage of partial products. Some systems, such as in [19], broke up the input mantissa bits into 10 parts and partial products were generated, after which the partial products were summed and reduced.

The authors in [20] reported a technique for low-power operation that utilised both sleep and BIVOS techniques. When starting from the columns of least significance, some columns are switched to sleep mode while the remaining columns are supplied with a biased voltage. This method results in a loss in accuracy. Only a few, such as the multiplier of [21] and others, catered to multi-precision, and at best, they processed two batches of input single-precision floating-point numbers or one in double-precision mode. Some systems, such as the multiplier architecture of [22], split the mantissa multiplier into upper and lower components of the number and predicted the sticky bit, carry bit and mantissa product from the upper part. In the event that the prediction was correct, the computation of the lower part was disabled, and the rounding operation was reported to be simplified [23]. In [24], computation-intensive applications were considered. In [25], a NAND-based multiplier was reported to have a compressor function. In [26], a high-speed-squaring and multiplication-based module was proposed.

Finally, it was observed that none of the existing binary multiplication systems analysed past multiplication operations to further reduce the latency of the multiplication operation. Focussing on previous multiplication operations could benefit future multiplications, hence preventing the system from having to undergo lengthy partial product generation operations, especially in the case of quadruple- and octuple-precision
modes, where the number of partial products can become very large. The contribution of this paper is the development of a novel mantissa similarity investigator (MSI)-interfaced multi-precision binary multiplier architecture capable of operating in several precision modes. Hence, this architecture allows for multiplication in single-, double-, quadruple- and octuple-precision multiplication modes and is also capable of increased throughput. Further, this provides for multiplication of eight batches of multiplicand/multiplier pairs for single-precision mantissa multiplication, four batches of multiplicand/multiplier pairs for double-precision mantissa multiplication, two batches of multiplicand/multiplier pairs for quadruple-precision mantissa multiplication, or one batch of multiplicand/multiplier pairs every time the system is initiated. The system has a shorter path delay than all other existing binary multiplier implementations. It is also capable of further reductions in path delay by utilising the novel technique of mantissa similarity investigation. The system has a shorter path delay than all other existing of binary multiplier implementations. This contribution is likely to be extremely useful for implementing arithmetic operations in digital and computer systems now and in the future.

4 | DESIGN OF NOVEL MULTI-PRECISION BINARY MULTIPLIER ARCHITECTURE

This section presents the design of the MSI-interfaced ‘novel multi-precision binary multiplier architecture’ to be used in the development of the novel multi-precision floating-point multiplier architecture. The novel multi-precision binary multiplier architecture was designed using the design approach reported by [22]. The module has been applied on the basis of its features and design as explained in the sections that follow.

4.1 | Mantissa similarity investigator—prepared novel multi-precision binary multiplier architecture

This section presents the design of the MSI-prepared novel multi-precision binary multiplier architecture to be used in the development of the novel MSI-interfaced binary multiplier architecture. The system interface definition of the MSI-prepared novel multi-precision binary multiplier architecture consists of eight inputs and two outputs. The clock input to this system is a periodic waveform—when the reset signal goes to HIGH, the multiplier is initialised, and the predetermined default state is enforced. The precision_mode of the system is used to select the floating point (single, double, quadruple, octuple), and the subjected implementation makes full use of this input to facilitate all four precision modes. When start is set to HIGH logic, the multiplication operations are commenced, and the system performs binary multiplication of multiplicand with multiplier. The system consists of two additional inputs, 8-bit similarity_code and 474-bit similar_products. Because it is possible to compute eight single-precision, four double-precision, two quadruple-precision, or one octuple-precision products every time start is set to HIGH, both 8-bit similarity_code and 474-bit similar_products cater to the range of products generated in all four precision modes. The MSI unit of Section IV(B) of this paper analyses the multiplier inputs A and B and estimates, based on past multiplication operations, the products that those batch inputs should yield. The 474-bit similar_products inputs hold all such estimated products. The 8-bit similarity_code indicates, by the presence of logic 1’s, which batches (or segments) of similar_products hold a previously computed product. The result of this multiplication is then assigned to the output product, and thereafter, the done signal goes to HIGH, which indicates the end of the arithmetic process.

The data path design of the MSI-prepared novel binary multiplier architecture consists of 14 blocks. This system considers the possibility that multiplication operations with particular multiplicand and multiplier inputs have been executed in the past and therefore should not be repeated; as such, those products should be estimated and made available to the multiplier system to avoid the expenditure of system time to report multiplication operations. This system also considers that the 237-bit wide multiplicand and multiplier inputs to the data path may comprise eight single-precision batches of multiplicand/multiplier pairs, or four double-precision batches of multiplicand/multiplier, or two quadruple-precision batches of multiplicand/multiplier pairs, or one batch of multiplicand/multiplier pair for octuple-precision mantissa multiplication. The simultaneous sum previously accumulated from the 48-bit binary adder is stored in the hold register. Then the sum of the new partial product, which is generated by the hold-shift register and stored, and the previously accumulated sum is computed by the 48-bit binary adder. On the other hand, if the least significant bit of the multiplier is zero then the bit monitor is still incremented, and the shifting and checking operations are repeated accordingly. This process is undertaken when all the multiplier bits are being examined. When all multiplier bits have been examined, the bit monitor signals the batch monitor (another up-counter) via the ‘tc_up’ port. The batch monitor keeps track of the number of binary multiplication operations carried out for the stated precision mode since triggering of the ‘start’ port.

4.2 | Mantissa similarity investigator

The MSI consists of eight (8) inputs and three (3) outputs. The input clock is the periodic clock waveform to the proposed system. When reset signal goes to HIGH, the entire system is initialised and the default state is enforced. The precision_mode of the system is used to select the floating-point precision mode, which can be single, double, quadruple, or octuple. When check_similarity is indicated as HIGH,
add_new_products is asserted as LOW, and the system analyses inputs A (multiplicands) with B (multipliers) to determine whether such inputs have been multiplied in the past, and if matches are found, the system estimates such products associated with those inputs and assigns them as batches of the 474-bit similar_products output and also places a logic 1 in the corresponding bit of the 8-bit similarity_code, which generally indicates which of the batches (or segments) of similar_products hold products computed in the past. When check_similarity is asserted as LOW, add_new_products is asserted to be HIGH, and the system adds the new products generated by the novel binary multiplier system (found on port new_products) to the memory storage element of the MSI unit for future similarity checks. Finally, the done signal is asserted as HIGH to indicate the end of the process in both cases (checking similarity and adding new products). The bit of the similarity_code that was recently updated corresponds to the product corresponding to the matching BA batch of inputs. If the batches of BA are analysed by interaction with the RAM block, and it is determined by the MSI unit that such inputs were not multiplied in the past, the comparator output equal is set to LOW, and this is recorded as part of the similarity code similarity_code stored in the load-shift register, and nothing is stored in the load-shift register. The process is repeated until all BA batches have been analysed. In terms of new products, the RAM write monitor is a modulo counter that provides the RAM block. The address in the RAM block is where the next new product can be added. The load-shift register stores and shifts the BA batches of multiplicands and multipliers processed by the data organiser, while the load-shift register stores and shifts 474 bits of products computed by the novel binary adder.

The load-shift register holds the last 8-bit similarity code generated by the MSI. Each bit of this similarity code is compared with zero (0) to determine the locations of the new product batches in the 474 bits of products generated by the novel binary multiplier. If the bits of the similarity code match zero (0), the equal port of the comparator is set to logic HIGH, and this results in the corresponding batch of product being stored in the next available location of RAM specified by the RAM write monitor. This process continues until all new products have successfully been added for future similarity checks.

4.3 | Complete novel mantissa similarity investigator–interfaced multi-precision binary multiplier architecture

The MSI-prepared novel binary multiplier system (prior to MSI interfacing) was combined with the MSI to form the complete modified novel MSI-interfaced binary multiplier system architecture, which was expected to result in significant reductions in path delay in all precision modes.

The data path design (Figure 1) of the complete novel multi-precision binary multiplier with MSI comprises four blocks. Blocks 1A and 1B are 237-bit 4-line to 1-line multiplexers. The purpose of these multiplexers is to accept inputs of 237 bits of data consisting of all batches of mantissa data for both multiplicands and multipliers of floating-point numbers. In cases such as single-, double- and quadruple-precision, zeros are used to pad unused bits. The multiplexer assigns one of the inputs to its output depending on the precision mode selected. The MSI examines the multiplicand and multiplier inputs to determine whether they match any previously multiplied multiplicand and multiplier inputs, and if there are any matches, it outputs values matching the expected product.

**FIGURE 1** Data path design for the novel mantissa similarity investigator–interfaced multi-precision binary multiplier architecture
along with a similarity code that the novel multi-precision binary multiplier uses to generate the resulting batches of products, hence further reducing the system’s latency and path delay.

The data path design of the novel MSI-interfaced multi-precision binary multiplier (Figure 2) was utilised in the construction of the data path interface definition after which the control path interface definition was constructed by connecting all control input and output ports along with data path–control path interface ports. Both data path interface definition and control interface definition were combined to produce the finite state machine with datapath model for the system (Figure 2). The state transition table was then constructed by analysing the operation of the data path design and following the desired sequence.

5 | HARDWARE IMPLEMENTATION OF NOVEL MULTI-PRECISION BINARY MULTIPLIER ARCHITECTURE

Implementation of the novel MSI-interfaced multi-precision binary multiplier in hardware was done using VHDL using Xilinx ISE Design Suite 14.7. Several sub-modules are part of the considered system, where a structural approach has been used for implementation of the proposed system. Sub-modules were port-mapped to implement the data path on the basis of the systems proposed in [23, 27]. The control path has been implemented as a finite state machine. Data path and control path were then port-mapped together to produce the overall system and the results thereon.

6 | VERIFICATION AND VALIDATION OF NOVEL MULTI-PRECISION BINARY MULTIPLIER ARCHITECTURE

The novel MSI-interfaced multi-precision binary multiplier system in single-, double-, quadruple- and octuple-precision modes was verified using a dynamic range of inputs values of multiplicand and multiplier. Simulation of the proposed system was performed on all components of the multiplier system to ascertain whether they were operating as expected out of the design; specifically with timing of the system. The implemented system was implemented for all four precision modes to verify that it correctly multiplied multiplicand and multipliers for a range of values at all four precision modes.

The actual outputs of the multiplier at all four precision modes were compared with the expected outputs to verify that they correctly multiplied the inputs. The comparison confirmed that all indicated that the multiplier’s actual outputs for all four precision modes corresponded to their expected results as shown in Figure 3.

The novel MSI-interfaced multi-precision binary multiplier for use in a multi-precision floating-point multiplier was simulated using ISim, and latency was extracted from the simulation screen. The hardware utilisation was obtained using the synthesis report from Xilinx ISE Design Suite 14.7. The path delay of the system for several FPGA targets was
determined via a post-place and route static timing report. The Xilinx ISE Design Suite 10.1 was utilised to obtain the post-place and route static timing report for the Virtex-2 FPGA target because that target was not supported by Xilinx ISE Design Suite 14.7 (Table 1). Tables 2 and 3 indicate the performance parameters extracted via simulation and synthesis.

When the novel multi-precision binary multiplier architecture is equipped with MSI, the system is arranged such that if it determines that the present inputs were used in binary multiplication operations in the past (referred to as ‘similar inputs’), the system avoids performing the multiplication operation completely and only utilises the previously computed product as the result, hence reducing the path delay. The more input batches that are determined to be similar inputs, the shorter the path delay is expected to be.

To analyse system performance, worst-case and best-case scenarios must be considered. Once the MSI-interfaced multi-precision binary multiplier architecture detects no similar inputs in the input batches, the system no longer benefits from MSI at that instant, but it is likely to benefit from MSI in the future if such inputs were determined to be similar for

### Table 1

| FPGA platform | Nomenclature          |
|---------------|-----------------------|
| Spartan-3     | XC3S1000-4FG256       |
| Spartan-3A    | XC3S1000-5TQ144       |
| Spartan-3E    | XC3S1000E-5TQ144      |
| Spartan-6     | XC6SLX4-3TQG144       |
| Virtex-2      | XC2V1000-5FG456C      |
| Virtex-4      | XC4VFX12-10FF668      |
| Virtex-5      | XC5VFX30T-2FF665      |
| Virtex-6      | XC6VLX75T-3FF484      |

### Table 2

| Precision mode | Platform | Maximum path delay/ns |
|----------------|----------|-----------------------|
| Single (8 batches) | Spartan-3 | 8.163 ns (3.857 ns logic, 4.306 ns route) |
|                 | Spartan-3A| 7.087 ns (3.017 ns logic, 4.070 ns route) |
|                 | Spartan-3E| 6.898 ns (2.367 ns logic, 4.531 ns route) |
|                 | Spartan-6 | 4.806 ns (1.516 ns logic, 3.290 ns route) |
|                 | Virtex-2  | 3.631 ns (0.866 ns logic, 2.765 ns route) |
|                 | Virtex-4  | 3.515 ns (0.837 ns logic, 2.678 ns route) |
|                 | Virtex-5  | 4.622 ns (2.198 ns logic, 2.424 ns route) |
|                 | Virtex-6  | 2.816 ns (1.055 ns logic, 1.761 ns route) |
| Double (4 batches) | Spartan-3 | 8.484 ns (3.946 ns logic, 4.538 ns route) |
| | Spartan-6  | 4.984 ns (1.840 ns logic, 3.144 ns route) |
| | Virtex-2   | 4.712 ns (1.679 ns logic, 3.033 ns route) |
| | Virtex-4   | 4.629 ns (1.869 ns logic, 2.760 ns route) |
| | Virtex-5   | 4.079 ns (2.125 ns logic, 1.954 ns route) |
| | Virtex-6   | 3.374 ns (1.280 ns logic, 2.094 ns route) |
| Quadruple (2 batches) | Spartan-3 | 9.274 ns (4.294 ns logic, 4.980 ns route) |
| | Spartan-6  | 5.546 ns (1.700 ns logic, 3.846 ns route) |
| | Virtex-2   | 4.681 ns (1.911 ns logic, 2.770 ns route) |
| | Virtex-4   | 4.856 ns (0.577 ns logic, 4.279 ns route) |
| | Virtex-5   | 5.189 ns (2.694 ns logic, 2.495 ns route) |
| | Virtex-6   | 3.153 ns (1.247 ns logic, 1.906 ns route) |
| Octuple (1 batch) | Spartan-3 | 10.612 ns (4.857 ns logic, 5.755 ns route) |
| | Spartan-6  | 5.608 ns (0.998 ns logic, 4.610 ns route) |
| | Virtex-2   | 5.488 ns (1.799 ns logic, 3.689 ns route) |
| | Virtex-4   | 5.441 ns (0.449 ns logic, 4.992 ns route) |
| | Virtex-5   | 6.566 ns (4.447 ns logic, 2.119 ns route) |
| | Virtex-6   | 4.269 ns (0.622 ns logic, 3.647 ns route) |
The longest path delay for this system corresponds to scenarios where all the bits of input batches are non-zero. As such, the inputs to the novel architecture were arranged such that all multiplicand and multiplier bits for all batches were non-zero. This was applied to all precision modes.

When no similar inputs were found, the MSI-interfaced multi-precision binary multiplier architecture exhibited path delays shorter than the existing 24-bit binary multiplier for nearly all existing 8-, 16-, and 24-bit binary multipliers at all precision modes. In the case of single-, double- and quadruple-precision modes, the path delays stated in Tables 2 and 3 correspond to the time taken to compute the product of multiple batches. As such, it is likely that the path delay for the computation of products for one batch of inputs in single- and double-precision modes is shorter than for all existing implementations of 8-, 16- and 24-bit binary multipliers.

It must be noted that the novel system in single-, double-, quadruple- and octuple-precision modes carries out 24-, 53-, 113- and 237-bit binary multiplications, respectively. As such, in most cases the novel multi-precision binary multipliers in all precision modes outperform existing implementations of 8-, 16- and 24-bit binary multipliers.

Once the MSI-interfaced multi-precision binary multiplier architecture detects similar inputs in the input batches, the system avoids execution of lengthy multiplication operations. It must be noted that the system operates equally when similar inputs are found, when at least one input is zero (0) and at least one input is one (1). As such, the effect of MSI on the performance of the novel multiplier architecture was determined by first arranging all input bits of the multiplicand and multiplier to be zero (0). The path delays obtained corresponded to the novel multiplier system in which all inputs were determined to be similar to past inputs and in which the novel multiplier

| Precision mode          | Platform | Maximum path delay/ns |
|-------------------------|----------|-----------------------|
| Single (8 batches)      | Spartan-3| 4.841 ns (1.873 ns logic, 2.968 ns route) |
|                         | Spartan-3A| 2.498 ns (0.988 ns logic, 1.510 ns route) |
|                         | Spartan-3E| 2.672 ns (1.107 ns logic, 1.565 ns route) |
|                         | Spartan-6| 2.385 ns (0.963 ns logic, 1.422 ns route) |
|                         | Virtex-2| 2.136 ns (0.879 ns logic, 1.257 ns route) |
|                         | Virtex-4| 2.108 ns (0.836 ns logic, 1.272 ns route) |
|                         | Virtex-5| 1.896 ns (0.655 ns logic, 1.241 ns route) |
|                         | Virtex-6| 1.347 ns (0.565 ns logic, 0.782 ns route) |
| Double (4 batches)      | Spartan-3| 3.476 ns (2.018 ns logic, 1.458 ns route) |
|                         | Spartan-6| 2.180 ns (1.029 ns logic, 1.151 ns route) |
|                         | Virtex-2| 1.689 ns (0.577 ns logic, 1.112 ns route) |
|                         | Virtex-4| 1.610 ns (0.836 ns logic, 0.774 ns route) |
|                         | Virtex-5| 1.674 ns (0.656 ns logic, 1.018 ns route) |
|                         | Virtex-6| 1.377 ns (0.527 ns logic, 0.850 ns route) |
| Quadruple (2 batches)   | Spartan-3| 5.132 ns (0.981 ns logic, 4.151 ns route) |
|                         | Spartan-6| 2.435 ns (0.991 ns logic, 1.444 ns route) |
|                         | Virtex-2| 2.114 ns (0.593 ns logic, 1.521 ns route) |
|                         | Virtex-4| 1.817 ns (0.825 ns logic, 0.992 ns route) |
|                         | Virtex-5| 1.637 ns (0.656 ns logic, 0.981 ns route) |
|                         | Virtex-6| 1.547 ns (0.293 ns logic, 1.254 ns route) |
| Octuple (1 batch)       | Spartan-3| 6.534 ns (1.410 ns logic, 5.124 ns route) |
|                         | Spartan-6| 2.428 ns (0.934 ns logic, 1.494 ns route) |
|                         | Virtex-2| 2.818 ns (1.076 ns logic, 1.742 ns route) |
|                         | Virtex-4| 3.323 ns (1.051 ns logic, 1.272 ns route) |
|                         | Virtex-5| 1.576 ns (0.655 ns logic, 0.921 ns route) |
|                         | Virtex-6| 2.021 ns (0.565 ns logic, 1.456 ns route) |
system bypassed the multiplication of all input batches as a result.

When all inputs to the MSI-interfaced multi-precision binary multiplier architecture were determined to be similar inputs, the path delay for various FPGA targets was further reduced compared with the scenario in which no similar inputs were found. Furthermore, when all inputs were determined to be similar inputs, the resulting path delays were shorter than for all existing 16- and 24-bit binary multiplier systems and for almost all existing 8-bit binary multipliers at all precision modes. In the case of single-, double- and quadruple-precision modes, the path delays stated in Tables 2 and 3 correspond to the time taken to compute the products of multiple batches. As such, it is likely that the path delay for the computation of product for one batch of inputs in all precision modes is shorter than it is for all existing implementations of 8-, 16- and 24-bit binary multipliers.

It should be noted that only a few authors who have documented existing 8-, 16- and 24-bit binary multiplier systems have explicitly stated that the figures stated for the delays of their multiplier systems were actually maximum path delays obtained after post-place and route static timing.

**Table 4** Performance comparison between the novel 24-bit binary multiplier architecture and the various 8-bit binary multipliers reviewed

| Source | Multiplier | Path delay/ns |
|--------|------------|---------------|
| This paper | Novel 24-bit Binary Multiplier Architecture | See Tables 2 and 3 |
| [5] – DSCH2 tool | Regular Dadda Multiplier | 4.40 |
| | Decomposed Dadda Multiplier | 4.10 |
| | Partitioned Dadda Multiplier | 5.50 |
| | Wallace Tree Multiplier based on 3:2, 4:2 & 5:2 compressor | 9.40 |
| | Dadda Multiplier based on Higher-Order Compressors | 6.40 |
| | Proposed Hybrid Multiplier Combination (Dadda/Wallace/Wallace/Dadda) | 7.50 |
| [6] – Spartan-3 XC3S50A-4TQ144 | Conventional Multiplier | 11.00 |
| | Urdhva Vedic Multiplier | 5.50 |
| | Nikhilam Sutra of Vedic Multiplier | 6.25 |
| [12] – Virtex-4 (XC4VFX12-10FF668) | Vedic Multiplier | 9.40 |
| [13] – Spartan-3 (XC3S1000-4FG256) | Vedic Multiplier | 45.68 |
| [14] – Spartan-3E (XC3S100E-5TQ144) | Vedic Multiplier | 13.43 |
| [16] – Spartan-3 (XC3S100-5FG320) | Vedic Multiplier | 23.18 |

**Table 5** Performance comparison between the novel 24-bit binary multiplier architecture and the various 16-bit binary multipliers reviewed

| Source | Multiplier | Path delay/ns |
|--------|------------|---------------|
| This paper | Novel 24-bit Binary Multiplier Architecture | See Tables 2 and 3 |
| [9] | Vedic Multiplier | 27.15 |
| [10] – Virtex-6 (XC6VLX75T-3FF484) | Vedic Multiplier | 13.45 |
| [6] – Spartan-3 (XC3S1000-4FG256) | Conventional Multiplier | 11.00 |
| | Urdhva Vedic Multiplier | 6.00 |
| | Nikhilam Sutra of Vedic Multiplier | 6.00 |
| [12] – Virtex-4 (XC4VFX12-10FF668) | Vedic Multiplier | 11.51 |
| [14] – Spartan-3E (XC3S100E-5TQ144) | Vedic Multiplier | 17.62 |
| [16] – Spartan-3 (XC3S100-5FG320) | Vedic Multiplier | 38.82 |

**Table 6** Performance comparison between the novel 24-bit binary multiplier architecture and the various 24-bit binary multipliers reviewed

| Source | Multiplier | Path delay/ns |
|--------|------------|---------------|
| This paper | Novel 24-bit Binary Multiplier Architecture | See Tables 2 and 3 |
| [11] | Vedic Multiplier | 16.32 |
| [12] – Virtex-4 (XC4VFX12-10FF668) | Vedic Multiplier | 13.00 |
analysis (consisting of logic delays, routing delays and clock skew) and not path delay obtained from the synthesis report (less accurate). Many of the authors have indicated that their path delays were obtained after synthesis, and as a result, the delays stated may not include routing delays and clock skew.

The delays stated for the novel MSI-interfaced multi-precision binary multiplier implemented in this paper are maximum path delays after post-place and route static timing analysis. In this paper it was assumed that the authors of documentation for existing implementations of 8-, 16- and 24-bit binary multiplier systems provided maximum path delay after post-place and route static timing analysis. The novel MSI-interfaced multi-precision binary multiplier implemented in this paper still had shorter delays than those of existing multiplier systems. However, if the delays stated by the authors who have investigated existing multiplier systems were indeed excluding routing delays and clock skew, it would mean that the novel MSI-interfaced multi-precision binary multiplier implemented in this paper has performed much better than its existing counterparts (Tables 4–6).

7  |  CONCLUSIONS

This paper 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. No other existing system allows binary multiplication of eight pairs of 24-bit inputs in single-precision mode, four pairs of 53-bit inputs in double-precision mode, two pairs of 113-bit inputs in quadruple-precision mode, or one pair of 237-bit inputs in octuple-precision mode. The system developed in this chapter, however, can facilitate this and hence increase the throughput of the system every time the system is initiated. 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.

ORCID
Geetam Singh Tomar https://orcid.org/0000-0002-0246-1527

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 utilisation. 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 Sourabhi Yenamachinta: 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.2015.7380417
6. Anitha, P., Ramanathan, P.: A new hybrid Multiplying 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. Choppade, S.S., Mehta, R.: Performance analysis of vedic multiplication technique using FPGAs. 2015 IEEE Bombay Section Symposium (IBSS), September 2015, Mumbai, pp. 1–6 (2015). https://doi.org/10.1109/IBSS.2015.7456657
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 (ICACCT), pp. 1757–1760. (2014). https://doi.org/10.1109/ICACCT.2014.7019410
9. Mhaidat, K.M., Abdelmughni, 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.2014.7038609
10. Buthija, R.K., et al.: Low power high speed 16x16 bit multiplier using vedic mathematics. Int. J. Comp. 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-Tiryagbyam algorithm. 2015 Global Conference on Communication Technologies (GCCT), pp. 192–196. (2015). https://doi.org/10.1109/GCCT.2015.7342650
14. Goldhale, G.R., Bahirongde, P.D.: Design of vedic-multiplier using area-efficient 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
15. Ram, G.C. et al.: Area efficient modified vedic multiplier. 2016 International Conference on Circuit and Computing Technologies (ICCPCT), pp. 276–279. (2016). https://doi.org/10.1109/ICCPCT.2016.7530294

16. Sharma, R., Kaur, M., Singh, G.: Design and FPGA implementation of optimised 32-bit vedic multiplier and square architectures. 2015 International Conference on Industrial Instrumentation and Control (ICIC) College of Engineering, Pune, India, pp. 960–964 (2015). https://doi.org/10.1109/ICIC.2015.7150885

17. Thapliyal, H., Srinavas, M.B.: A novel time-area-power efficient single precision floating multiplier. Proceedings of MAPLD, 1–3 (2005). http://klabs.org/mapld05/papers/1013_thapliyal_paper.pdf

18. Anane, N., et al.: Hardware implementation of variable precision multiplication on FPGA. 4th International Conference Design & Technology of Nanoscale Era, 2009 (DTIS '09), pp. 77–81. (2009). https://doi.org/10.1109/DTIS.2009.4938028

19. Ramesh, A. P., Tilak, A. V. N., Prasad, A. M.: An FPGA based high speed IEEE-754 double precision floating point multiplier using Verilog. 2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), 1–5 (2013). https://doi.org/10.1109/ICEVENT.2013.6496575

20. Gupta, A., et al.: Low power probabilistic floating point multiplier design. 2011 IEEE Computer Society Annual Symposium on VLSI, 24(3), 182–187. (2011). https://doi.org/10.1109/ISVLSI.2011.54

21. Havaldar, S., Gurumurthy, K.S.: Design of Vedic IEEE 754 floating point multiplier. 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), 24(3), 1131–1135 (2016). https://doi.org/10.1109/RTEICT.2016.7808008

22. Kuang, S.-R., Wang, J.-P., Hong, H.-Y.: Variable-latency floating-point multipliers for low-power applications. IEEE Trans Very Large Scale Integr Syst. 18(10), 1493–1497 (2010). https://doi.org/10.1109/TVLSI.2009.2025167

23. Institute of Electrical and Electronic Engineers (IEEE). IEEE Standard VHDL Language Reference Manual. IEEE 1076.3 (1993)

24. Naik, A., Keka, D., Pal, D.: ASIC implementation of high speed Adaptive Recursive Karatsuba multiplier with Square Root carry select adder. Proc 11th Latin American symposium on Circuits and Systems, pp. 1–5 (2020)

25. Satish, T., Pande, K.S.: Multiplier using NAND based compressors. 3rd International conference on Electronics, Materials Engineering and Nano technology, pp. 1–5 (2019)

26. Deepa, A, Marimuthu, C.N., Murugesan, C: An efficient high speed squaring and multiplier architecture using yavadunam sutra and bit reduction technique. J Phys Conf, 1–14 (2019)

27. George, M., Singh Tomar, G.: Hardware design procedure: principles and practices. 2015 Fifth International Conference on Communication Systems and Network Technologies, pp. 834–838 (2015)

How to cite this article: Tomar, G.S., George, M.L., Tomar, A.S.: Multi-precision binary multiplier architecture for multi-precision floating-point multiplication. IET Circuits Devices Syst. 15(5), 455–464 (2021). https://doi.org/10.1049/ cds2.12041