DSP48E Efficient Floating Point Multiplier Architectures on FPGA

Manish Kumar Jaiswal, and Hayden K.-H So
Department of EEE, The University of Hong Kong, Hong Kong
Email: manishkj@eee.hku.hk, hso@eee.hku.hk

***Abstract*—This paper presents FPGA based hardware architectures for floating point (FP) multipliers. The proposed multiplier architectures are aimed for single precision (SP), double precision (DP), double-extended precision (DEP) and quadruple precision (QP) implementation. This paper follows the standard computational flow for FP multiplication. The mantissa multiplications, the most complex unit of the FP multiplication, are built using efficient use of Karatsuba methodology integrated with the optimized used of in-built 25x18 DSP48E blocks available on the Xilinx Virtex-5 onward FPGA devices. It also combined with the other techniques (radix-4 booth encoding for small multipliers, partial products reduction using 4:2, 3:2, 2:2 counters; compression of multi-operands adders) used at places, to improve the design. The proposed architectures out-performs the available state-of-the-art, and used only 1-DSP48, 3 DSP-48, 6 DSP48 and 18 DSP48 for SP, DP, DEP, and QP multipliers respectively.**

***Keywords*-Floating Point Arithmetic, Multipliers, Digital Arithmetic, FPGA, DSP48E.**

I. Introduction

Floating point (FP) multiplication is an essential component required in a large set of FPGA based hardware acceleration application. So, efficient implementation of FPGA based floating point multipliers are highly desirable. To meet this requirement, several literature have focused on the efficient FPGA-based implementation of FP multiplier [1], [2], [3], [4], [5], [6], [7], for various formats of the floating point standard: single precision, double precision and quadruple precision. [6] has shown the large integer implementation using Karatsuba method [8] (around 18x18 multiplier IPs) and using tiling method (around the asymmetric size of DSP48E IP on Xilinx Virtex-5 onward FPGA devices). [1] has also proposed double precision and quadruple precision multiplier using tiling method. [3] has presented a set of FP arithmetic library, and used contemporary block method for FP mantissa multiplication. [2] has presented the architecture of quadruple precision FP multiplier using Karatsuba method around 18x18 multipliers, and has shown better results than [1] implementation. Similarly, [5] has used 3-partition Karatsuba method around 18x18 multiplier IP for a double precision multiplier implementation. All these methods have either used simple block method, or tiling method around 25x18 IP or Karatsuba method around 18x18 IP for the FP multiplier implementation, and tried to improve the multiplier IP usage. However, these methods still need more DSP48 block and have un-balanced DSP48E-LUTs utilization which reduces the scope of parallelism.

This work is aimed towards an efficient utilization of DSP48E for FP multipliers ranging from SP to QP multiplier, with balanced DSP48E-LUTs requirement. The proposed architectures are build around DSP48E IP with combination of Karatsuba method, block method, radix-4 Modified Booth Encoding for small multipliers, tree compression for partial products, and multi-operands-adder reduction using 4:2, 3:2 and 2:2 counters, and all included with balanced pipelining of the architectures.

In summary, the main contributions of present work can be summarized as follows:

- Proposed a set of floating point (SP, DP, DEP, and QP) multiplier architectures on Xilinx FPGA platform.
- DSP48E efficient mantissa multipliers for these architectures are designed using combination of several techniques (Karatsuba, Radix-4 Booth, Tree-Compression using 4:2, 3:2 & 2:2 counters), which results in better hardware utilization, when compared to the prior state-of-the-art.

II. Proposed Architectures

The IEEE-754 [9] defines the standard for the floating point arithmetic. The standard format for the SP, DP, DEP and QP floating point numbers are shown in Table-I.

Table I: Floating Point Format

	Word Size	Sign (1)	Exponent (E)	Mantissa (F)
SP (32-Bit)	[31:0]	[31]	[30:23]	[22:0]
DP (64-Bit)	[63:0]	[63]	[62:52]	[52:0]
DEP (80-bit)	[79:0]	[79]	[78:64]	[63:0]
QP (128-Bit)	[127:0]	[127]	[126:112]	[111:0]

This work is aimed towards the normalized floating point multiplier architectures, however, sub-normal processing can be included with similar benefits. The typical processing of floating point multiplier for normal operands is shown in Fig. 1. The sign and exponent processing as well as rounding and normalization are trivial, however, the mantissa multiplication is most critical component in terms of area and performance. This paper focuses mostly on this unit. "Rounding-to-nearest-place" method is used for rounding, and it consists of ULP (unit at last place) generation (based on rounding position, round-bit, guard-bit and sticky-bit) and

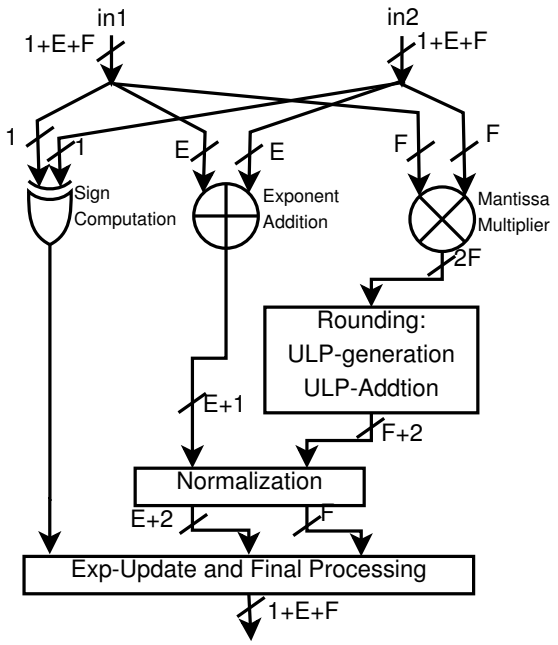

Figure 1: Floating Point Multiplier

ULP addition to the mantissa. The rest of this section will discuss the mantissa multiplier architectures for SP, DP, DEP, and QP multipliers.

A. Single Precision Mantissa Multiplier

The architecture for SP mantissa multiplier is shown in Fig. 2. It is a 24x24 bit multiplier, designed around a DSP48E IP. Its implementation follows the simple block method computation as below:

$$\begin{aligned} y &= a[23:0] * b[23:0] \\ &= a[23:0] * b[16*0] + \{(a[23:0] * b[23:17]) << 17 - bit\} \end{aligned} \quad (1)$$

Here, the DSP48E is used as 24x16 unsigned multiplication and its addition to the 17-bit shifted result of 24x7 unsigned multiplication. The 24x7 multiplier is designed using Radix-4 Modified Booth encoding, which partial products are compressed using 4:2, 3:2 and 2:2 counters, and a pipelined two-operands final adder. These techniques help in improving the critical path. The latency of 24x24 multiplier is 3 cycles.

B. Double Precision Mantissa Multiplier

The double precision multiplier needs a 53x53 bit mantissa multiplier. Its architecture is shown in Fig. 3. The architecture is based on the use of previous 24x24 multiplier, and its underlying computation is based on following:

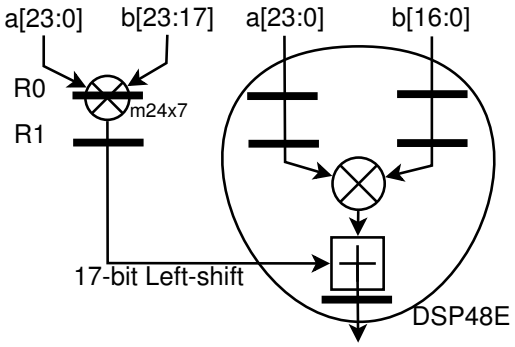

Figure 2: 24x24 Multiplier

$$\begin{aligned} y &= a[52:0] * b[52:0] \\ &= a[45:0] * b[45*0] \\ &\quad + \{(a[45:0] * b[52:46] + a[52:46] * b[45:0]), 46'b0\} \\ &\quad + \{a[52:46] * b[52:46], 96'b0\} \end{aligned} \quad (2)$$

In eq(2), the 46x7 and 7x7 multipliers are designed using radix-4 modified booth multiplication method with 2 pipeline stage, as shown in Fig. 3. However, the 46x46 multiplier is built using two-partition Karatsuba method [8], [1]. Using two-partition Karatsuba method, the multiplication is obtained as follows:

$$\begin{aligned} y &= a[45:0].b[45:0] \\ &= \{a1.b1, a0.b0\} + \{(a10.b10 - (a1.b1 + a0.b0)), 23'b0\} \end{aligned} \quad (3)$$

where, a0,b0,a1 and b1 are as shown in Fig. 3 for m46x46 multiplier. Here, it requires only 3 multiplier blocks (two 23x23 and one 24x24) instead of 4 24x23 using contemporary method, however, needs extra adders/subtractor. The 23x23 and 24x24 are done as in Fig. 2, using a DSP48E. Further, a compressor is used for the combined reduction of various terms from 46x46 multiplier and, the results of 46x7s and 7x7 multipliers. Here, the compressor can also be used at earlier stages, to include the individual terms of 46x7s and 7x7 multiplier, to reduce the latency, however, with slower performance. Here, the use of multi-operands-adder compression helps in overcoming the delay-requirement of extra adders/subtractors in the Karatsuba method, while retaining its benefits of reduced multiplier blocks. Thus, as shown in Fig. 3, the latency of 53x53 is 6 cycles, and it requires only 3 DSP48E blocks.

C. Double Extended Precision Mantissa Multiplier

The mantissa of DEP multiplier is 64 bits and it needs 65x65 mantissa multiplier. Here, it is designed as 66x66 multiplier and shown in Fig. 4. This architecture is based on the 3-partition Karatsuba method, which computes as follows:

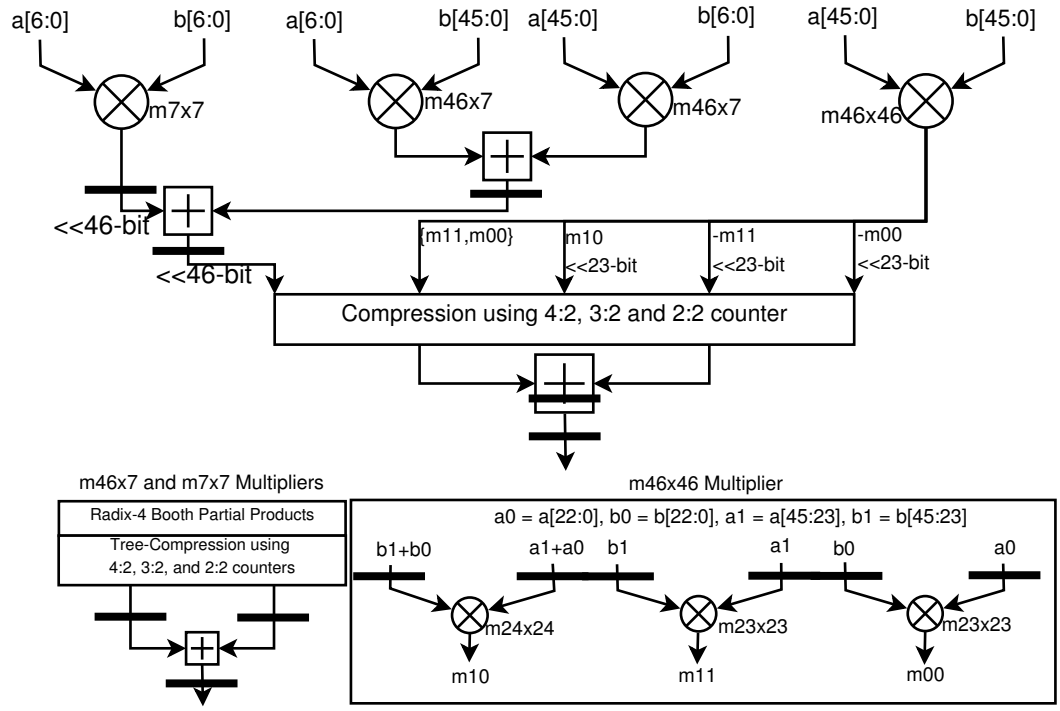

Figure 3: 53x53 Multiplier (m23x23 and m24x24 are 3 stage multipliers designed as in Fig 2)

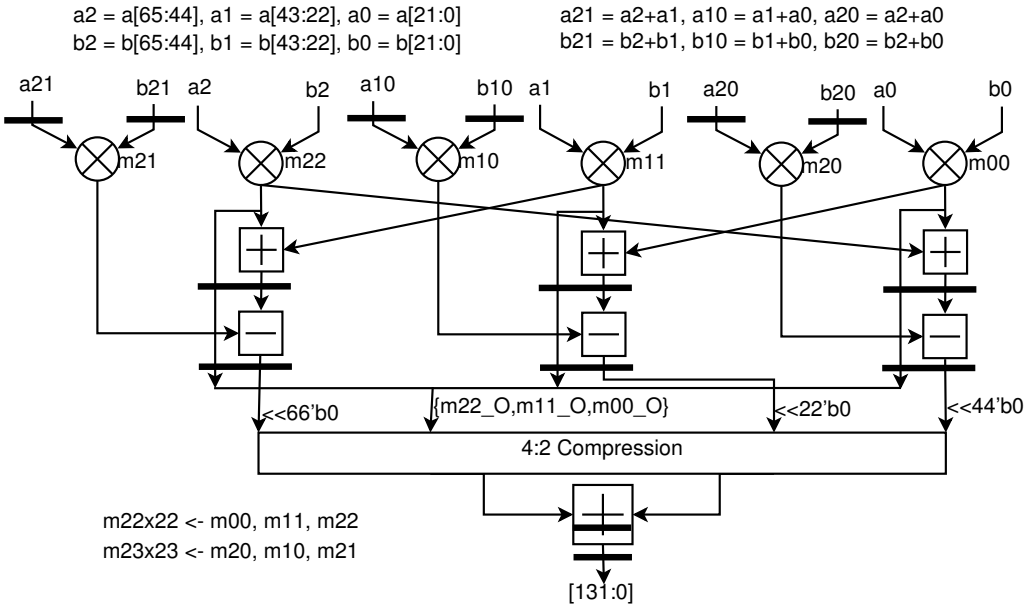

Figure 4: 66x66 Multiplier (m22x22 and m23x23 are 3 stage multipliers designed as m24x24 in Fig 2)

$$
\begin{aligned}
y &= a[65:0].b[65:0] \\
&= \{a2.b2, a1.b1, a0.b0\} \\
&\quad + \{(a21.b21 - (a2.b2 + a1.b1)), 66'b0\} \\
&\quad + \{(a20.b20 - (a2.b2 + a0.b0)), 44'b0\} \\
&\quad + \{(a10.b10 - (a1.b1 + a0.b0)), 22'b0\} \qquad (4)
\end{aligned}
$$

where, a0, a1, a2, b0, b1, b2, a21, a20, a10, b21, b20 and b10 are as shown in Fig. 4. This method needs only 6 multiplier blocks (3 of 22x22, and 3 of 23x23), compared to 9 of 22x22 using general method, while with some extra adders and subtractors. The 22x22 and 23x23 multipliers are implemented as 24x24 multiplier (Fig. 2). Further, the four terms of eq(4) are compressed using a 4:2 compressor

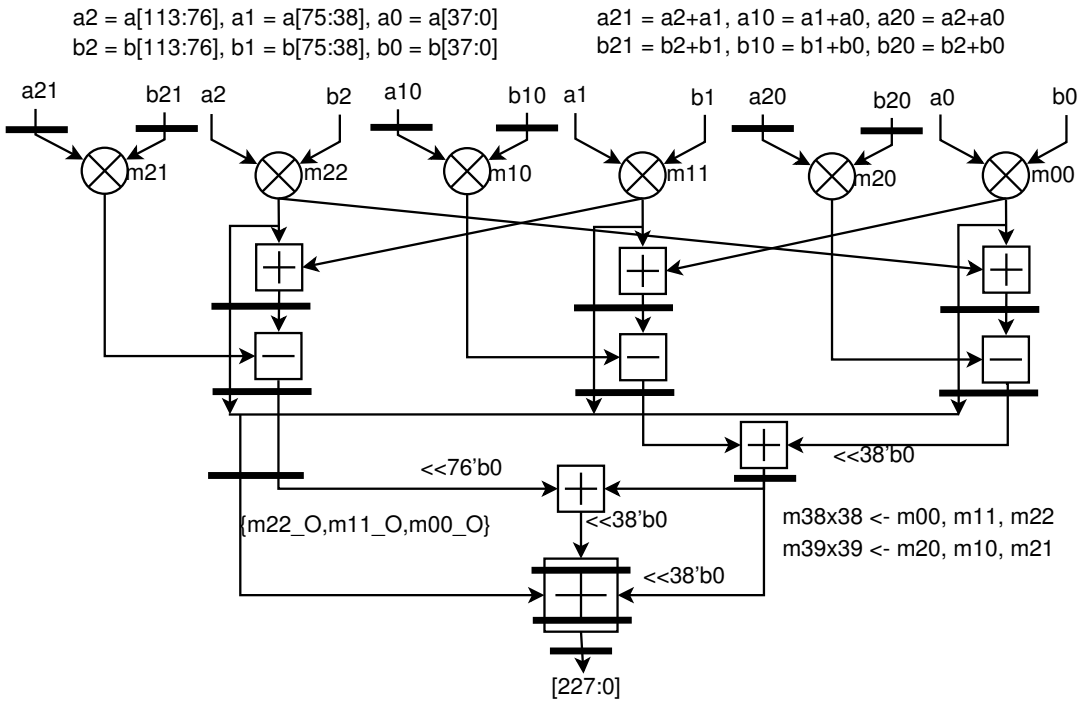

Figure 5: 114x114 Multiplier

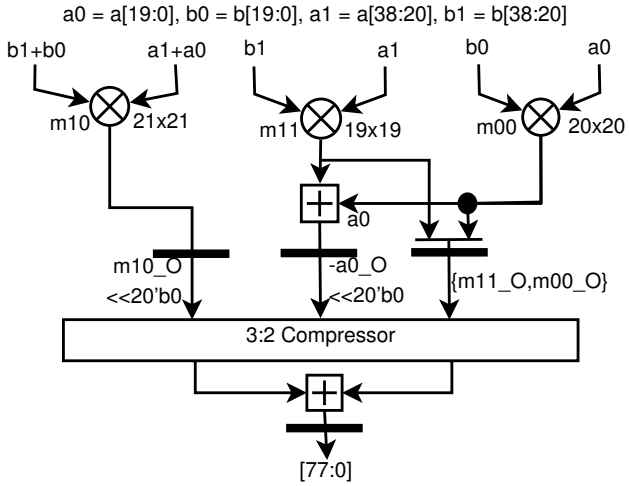

Figure 6: 39x39 Multiplier

to produce two operands, which finally added by a two-stage adder (to meet the critical path). The latency of 66x66 multiplier is 7 cycles and it needs only 6 DSP48E.

D. Quadruple Precision Mantissa Multiplier

The architecture of the quadruple precision mantissa multiplier is shown in Fig. 5. For QP, it needs a 113x113 multiplier and here it is implemented as 114x114. The implementation is based on the combination of 3-partition and 2-partition Karatsuba method with efficient use of DSP48E. The 114-bit operand is first partitioned in to three parts of 38-bit and implemented using eq(4) technique, and as shown in Fig. 5. At this, it requires three 38x38 and three 39x39 multipliers. The 38x38 and 39x39 are further designed using 2-partitioned Karatsuba method. The architecture for 39x39 is shown in Fig. 6, which requires one 20x20, one 19x19, and one 21x21 multipliers. Similarly, it is done for 38x38 multiplier, which needs two 19x19 and one 20x20 multiplier. These 19x19, 20x20 and 21x21 multiplier are implemented as 24x24 (as shown in Fig. 2), but with simple partial product generation method (due to small multiplier-operands). Further, in 114x114 multiplier, all the terms are combined using multi-level adders. Here, compression technique can be beneficial for lower latency implementation, however, with more pipelined implementation of large-size adders are not beneficial with the compression technique. The latency of presented 114x114 multiplier is 11 cycles, and it requires only 18 DSP48E for entire implementation.

III. Results and Comparisons

The proposed architectures of mantissa multipliers and corresponding floating point multipliers are implemented on Xilinx Virtex-5 FPGA device. The design metrics are obtained after PAR (placed and route) step of Xilinx ISE 14.6 tool flow. The design metrics of proposed implementations along with the details on prior state-of-the-art on Virtex-5 FPGA device (Virtex-5 is selected for implementation as most of the prior works are reported on Virtex-5 devices) are shown in Tables-II,III,IV and V, respectively, for single

Table II: Implementation Details and Comparisons for Single Precision Multiplier

	Latency	DSP48E	LUTs	FFs	Freq (MHz)
SP Mantissa Multiplier					
Proposed	3	1	249	121	331
34x34[2]	4	3	118	172	324
SP FP Multiplier					
Proposed	5	1	392	195	331
[3]	-	4	127 Slices		178*
[10]	6	3	99	114	410*
[10]	8	2	112	184	410*
[11]	6	3	79	124	462* (on V7)
[11]	8	2	92	164	462* (on V7)
[11]	8	1	238	363	462* (on V7)

*: Synthesis Results

precision, double precision, double extended precision and quadruple precision architectures. For comparison purpose, we have included the results of Xilinx floating point operator V5.0 [10] for Virtex-5 devices, as well as the latest Xilinx floating point operator V7.1 [11] results for Virtex-7 devices (as V7.1 results are not reported for older devices like Virtex-5). The proposed architectures are functionally validated for an extensive random test cases with round-to-nearest rounding method.

All the proposed architectures are targeted around 300 MHz speed requirements after PAR processing. Literature shows that this speed level is suitable for most of the system level architectures designed around floating point arithmetic. However, speed can be improved with more pipelining stages (as we can observe that the latency of proposed architectures are smaller). To meet critical path, the post mantissa processing requires 2-cycles for SP and, 3-cycles for DP, DEP and QP multipliers. From the comparisons, the proposed architectures are efficient in terms DSP48E requirements, however, requires more LUTs.

The prime benefit of proposed architectures appears more towards the high precision computation, ie double precision onward. For double precision it saves from 3 to 9 DSP48E, and for quadruple precision the saving is 6 to 16 DSP48E. Whereas, for single precision Xilinx latest operator does better than proposed work, but, for double precision and beyond precision proposed architectures are more hardware optimum. Further, Xilinx operators support only up to double precision architectures, and not available for double precision extended and quadruple precision architectures.

On taking account of LUT equivalent of a DSP48E, a simple synthesis of (a[23:0]*b[16:0]+c[47:0], in present context) results in to 644 LUTs. Taking this in to consideration, total equivalent LUTs requirements (LUT + DSP48E*644) of the proposed architectures are better than the prior related arts. Moreover, the presented proposals provide a more balanced utilization of DSP48E and LUTs, and will results in to more number of floating point multiplier units per device, and thus, promote more parallelism per device.

Table III: Implementation Details and Comparisons for Double Precision Multiplier

	Latency	DSP48E	LUTs	FFs	Freq (MHz)
DP Mantissa Multiplier					
Proposed	6	3	2071	1339	310
[1]	9	9	530	506	407*
[1]	8	6	919	872	407*
[4]	0	12	200	-	111*
[6]	4	8	243	400	369*
Logicore[1]	12	10	229	280	450*
DP FP Multiplier					
Proposed	9	3	2219	1301	327
[1]	14	9	804	804	407*
[1]	13	9	1184	1080	407*
[5]	10	6	765	790	390
[3]	-	9	375 Slices		149*
[10]	15	9	402	555	369*
[10]	15	10	342	495	359*
[10]	10	13	191	311	404*
[11]	15	9	276	564	462* (on V7)
[11]	15	10	224	503	462* (on V7)
[11]	10	11	199	492	329* (on V7)

*: Synthesis Results

Table IV: Implementation Details and Comparisons Double Extended Precision Multiplier

	Latency	DSP48E	LUTs	FFs	Freq (MHz)
DEP Mantissa Multiplier					
Proposed	7	6	2273	1320	331
[2]	8	9	796	1126	310
DEP FP Multiplier					
Proposed	10	6	2245	1549	317

*: Synthesis Results

Table V: Implementation Details and Comparisons for Quadruple Precision Multiplier

	Latency	DSP48E	LUTs	FFs	Freq (MHz)
QP Mantissa Multiplier					
Proposed	11	18	4506	4301	305
[2]	14	24	3030	3698	310
[1]	13	34	2070	2062	407*
[4]	0	35	1000	-	90*
QP FP Multiplier					
Proposed	14	18	4779	4392	305
[1]	20	34	2978	2815	355*
[2]	17	24	3211	3961	310

*: Synthesis Results

IV. Conclusions

This paper has presented a set of floating point multiplier architectures on Xilinx FPGA device for Virtex-5 onward series. The architecture for single precision, double precision, double extended precision and quadruple precision multiplier are proposed using efficient use of in-built DSP48E fabric in combination with Karatsuba methodology of multiplication. Various other techniques, like radix-4 booth encoding for small multipliers, reduction of partial products using 4:2, 3:2, 2:2 counters, reduction of multi-

operands adders, etc are also used at places, to improve the design. The proposals lead to a significant improvement in DSP48E usage, while at expense of more LUTs, however, have better overall equivalent LUTs requirements. Also, as shown in results and comparison section, the proposed architectures are better in hardware utilization towards higher precision implementations, ie for double precision and beyond architectures.

V. Acknowledgments

This work is party supported by the "The University of Hong Kong" grant (Project Code. 201409176200), the "Research Grants Council" of Hong Kong (Project ECS 720012E), and the "Croucher Innovation Award" 2013.

References

[1] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for floating-point double precision and beyond on FPGAs," *SIGARCH Comput. Archit. News*, vol. 38, pp. 73–79, Jan 2011.

[2] M. K. Jaiswal and R. C. C. Cheung, "Area-Efficient FPGA Implementation of Quadruple Precision Floating Point Multiplier," in *The IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW 2012)*. Shanghai, China: IEEE Computer Society, May 2012, pp. 369–375.

[3] X. Wang and M. Leeser, "Vfloat: A variable precision fixed- and floating-point library for reconfigurable hardware," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 3, pp. 16:1–16:34, Sep. 2010.

[4] S. Srinath and K. Compton, "Automatic generation of high-performance multipliers for fpgas with asymmetric multiplier blocks," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '10, 2010, pp. 51–58.

[5] M. K. Jaiswal and R. C. C. Cheung, "Area-efficient architectures for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support," *Microelectronics Journal*, vol. 44, no. 5, pp. 421–430, May 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269213000591

[6] F. de Dinechin, "Large multipliers with fewer DSP blocks," in *International Conference on Field Programmable Logic and Applications*, 2009, pp. 250–255.

[7] M. K. Jaiswal and R. C. C. Cheung, "VLSI Implementation of Double-Precision Floating-Point Multiplier Using Karatsuba Technique," *Circuits, Systems, and Signal Processing*, vol. 32, pp. 15–27, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00034-012-9457-3

[8] A. Karatsuba and Y. Ofman, "Multiplication of Many-Digital Numbers by Automatic Computers," in *Proceedings of the USSR Academy of Sciences*, vol. 145, 1962, pp. 293–294.

[9] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[10] Xilinx, "LogiCORE IP Floating-Point Operator v5.0." [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf

[11] ——, "LogiCORE IP Floating-Point Operator v7.1." [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/ru/floating-point.html