Efficient FPGA Implementation of Digit Parallel Online Arithmetic Operators

Authors removed for blind review

***Abstract*—Online arithmetic has been widely utilized for ASIC implementations with significant performance improvements, as it is designed to perform computations from the most significant digits. Recent research also shows that the digit parallel online operators can fail more gracefully when operating beyond the deterministic region, in comparison to the operators with conventional arithmetic. Unfortunately, the utilization of online arithmetic operators requires large area overhead, and the efficient implementation of general purpose online operators on FPGAs remains under investigation. In this paper, we propose novel approaches to implement the online adder and the online multiplier efficiently on current FPGAs with 6-input LUTs. The FPGA carry resources are intensively used for both area reduction and performance improvements. The performance of the proposed architectures are experimentally demonstrated**

I. Introduction

In the conventional form of computer arithmetic, the computation results are generated either from the least significant digit (LSD), e.g. addition and multiplication, or from the most significant digit (MSD), e.g. division and square root. This inconsistency in computing directions will potentially result in large latency when propagating data among different operations. Online arithmetic was designed to solve this problem [1], [2]. With online arithmetic, both inputs and outputs are processed in a MSD-first manner. This enables parallelism and duplication among various operations, and the overall computation latency can be significantly reduced [xxx]. A brief overview of online arithmetic is given in Section II.

Recent research has demonstrated another key advantage of online arithmetic that it allows gracefully degradation when operating beyond the deterministic region, e.g. under overclocking. This stream of research stems from the inspiration that for certain applications, releasing the absolute accuracy requirements could lead to significant performance improvements. However, traditional form of computer arithmetic does not fail gracefully when timing violation happens, because timing errors initially affect MSDs of the results. In comparison, online arithmetic is more "overclocking friendly", as timing errors only occurs at LSDs of the results. Despite its attractiveness, the usage of online arithmetic on existing FPGAs is still limited due to the huge area overhead.

Generally, it is difficult to efficiently map the arithmetic circuits solely onto the look-up-tables (LUTs), which are fundamental building blocks of FPGAs. Lots of efforts has been done to alleviate this concern. For instance, both main commercial FPGA vendors introduced dedicated carry logic into the logic block to reduce the carry propagation delay by over one order of magnitude [3]. There is a large amount of studies exploring methods that can effectively map circuits using the carry logic for performance improvements [4], [5], or propose alternative carry logic for FPGAs [6], [7] . Besides, hard DSP blocks are also included to boost multiplications and multiply-accumulations [xxx]. However, most of these approaches are designed to accelerate computations with the conventional arithmetic.

For online arithmetic, we notice from existing literatures that there are 2 directions that are worth investigating. First, although there has been previous research about FPGA implementations of online arithmetic, they typically only focus on online adders. A brief review of relevant works in this area is presented in Section III. However, the implementation of other key arithmetic primitives, such as online multipliers, are lack of exploration. Second, we found that existing approaches of building general purpose online adders are specifically for FPGAs with 4-input LUTs, and 2 LUTs within a Slice. However, as will be discussed in Section III, these approaches cannot be directly applied on many current FPGAs with 6-input LUTs and 4 LUTs in a slice, such as the Xilinx Virtex series FPGAs and all Xilinx 7 series FPGAs [8].

In this paper, both issues are addressed. In Section III, we propose a novel approach of mapping the digit parallel online adder onto FPGA with 6-LUTs. The mapping is based on the fast carry logic, and the available resources within a slice can be fully utilized. In addition, in Section IV we optimize the online multiplication algorithm to yield an efficient FPGA implementation of digit parallel online multiplier. The theoretical resource usage of both operators is given, and it is verified experimentally on Xilinx Virtex-6 FPGAs. We demonstrate that the proposed designs achieves significant area reduction and performance gain over their original implementations.

The main contributions of this paper are as follows:

1) An efficient approach of mapping online adders onto modern FPGAs;
2) Area efficient implementation of digit parallel online multiplication algorithm on FPGAs;

II. Background: Online Arithmetic

A. Key Features of Online Arithmetic

Online arithmetic has been widely used in numerous applications such as xxx [xxx]. Online arithmetic was originally designed for digit-serial operation, of which the data flow is illustrated in Fig 1. It can be seen that in order to generate the first output digit, δ digits of inputs are required and δ is called "online delay". Notice that δ is normally a constant, which is

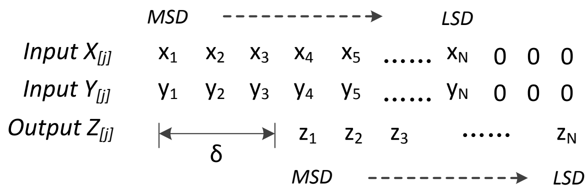


Fig. 1. Dataflow in digit-serial online arithmetic, in which both inputs and outputs are processed from the MSD to the LSD. δ denotes the online delay.

independent of the precision in a given operation. For ease of discussion, in the following of this paper the input data are normalized to fixed point numbers in the range $(-1, 1)$. Based on this premise, the online representation of N-digit operands and result at iteration j are given by (1), where $j \in [-\delta, N-1]$ and r denotes the radix [2].

$$X_{[j]} = \sum_{i=1}^{j+\delta} x_i r^{-i},\ Y_{[j]} = \sum_{i=1}^{j+\delta} y_i r^{-i},\ Z_{[j]} = \sum_{i=1}^{j} z_i r^{-i} \tag{1}$$

MSD-first operation is possible with the employment of the redundant number system . Normally there are 2 most commonly used redundant number representation: carry-save (CS) [9] and signed-digit (SD) [10]. For instance with SD representation, each digit is represented with a redundant digit set $\{-a, \cdots, -1, 0, 1, \cdots, a\}$, where $a \in [r/2, r-1]$. In comparison, the standard non-redundant representation only uses a digit set $\{0, r-1\}$. Thus a standard number corresponds to several possible redundant representations. For example, the two's complement number 0.111 can be represented in the online form as $0.10\bar{1}$, $0.0\bar{1}1$ and 0.111 among many other possible representations.

Due to the redundancy, the MSDs of the result can be calculated only based on partial information of the inputs, which is required by digit-serial online arithmetic. Then the value of the number can be revised by the following digits, because each number embodies multiple representations.

B. Binary Online Addition

Adders serve as a critical building block for arithmetic operations. To perform digit-parallel online addition, a redundant adder can be directly utilized. The adder structure diagram is shown in Fig. 2. An major advantage of the redundant number system over the standard ripple-carry based arithmetic is that the propagation of carry is eliminated, resulting in a precision-independent computation time for addition. As seen in Fig. 2, ideally the computation delay of this adder is only 2 full adder (FA) delays for any operand word-length, with the cost of one extra FA for each digit of operands. This makes the online adder suitable for building up more complex arithmetic operators such as multipliers to accelerate the sum of partial products [11].

(truncate precision)

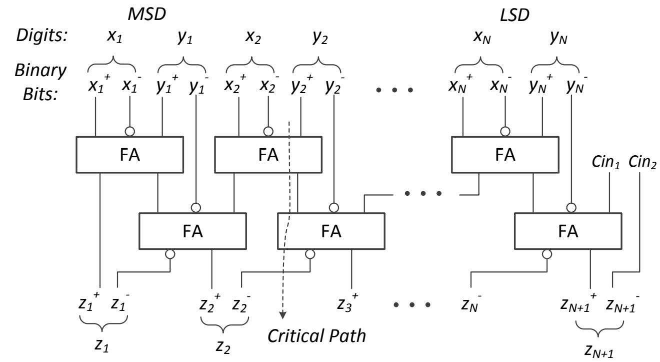


Fig. 2. An N-digit binary digit-parallel online adder with $(N+1)$-digit outputs. Both inputs and outputs are represented using SD representation.

Algorithm 1 Online Multiplication

Initialization: $X_{[-\delta]} = Y_{[-\delta]} = P_{[-\delta]} = 0$
Recurrence: $for\ \ j = -\delta,\ -\delta+1,\ \cdots,\ N-1\ \ do$

$$\begin{array}{rcl} H_{[j]} & = & r^{-\delta}\left(x_{j+\delta+1} \cdot Y_{[j+1]} + y_{j+\delta+1} \cdot X_{[j]}\right) \\ W_{[j]} & = & P_{[j]} + H_{[j]} \\ z_j & = & sel(W_{[j]}) \\ P_{[j+1]} & = & r\left(W_{[j]} - z_j\right) \end{array} \tag{2}$$

C. Binary Online Multiplication

Multiplication is another key arithmetic operator. Typically the online multiplication is performed in a recursive digit-serial manner, as illustrated in Algorithm 1 [12] where both inputs and outputs are N-digit number as given in (1). For a given iteration j, the product digit z_j is generated through a selection function $sel()$. For the radix r and a chosen digit set, there exits an appropriate selection method and a value of δ which ensure convergence [12]. As radix-2 is used most commonly in computer arithmetic, we keep $r = 2$ throughout this paper with the corresponding redundant digit set $\{\bar{1}, 0, 1\}$. In this case $sel()$ is given by (3) [13]. Notice that the election is made only based on 1 integer bit and 1 fractional bit of $W_{[j]}$.

$$sel(W_{[j]}) = \begin{cases} 1 & \text{if } W_{[j]} \geqslant \frac{1}{2} \\ 0 & \text{if } -\frac{1}{2} \leqslant W_{[j]} < \frac{1}{2} \\ \bar{1} & \text{if } W_{[j]} < -\frac{1}{2} \end{cases} \tag{3}$$

In Section IV, we will describe a modified online multiplication algorithm and its FPGA implementation, which are designed specifically targeting on digit parallel operations.

III. Digit Parallel Online Adder on FPGAs

A. Related Works

There has been previous works about FPGA implementation of digit parallel online adders. From the literature, the existing approaches can be classified into three types:

1) efficient mapping of the digit-parallel online adder onto sophisticated FPGAs resources [14], [15];

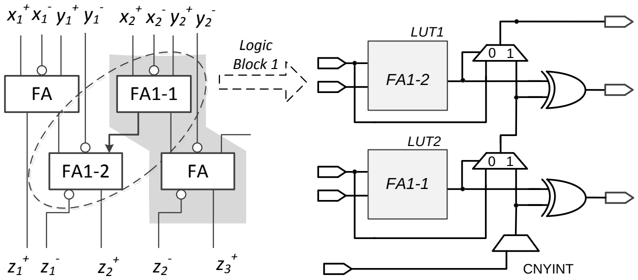


Fig. 3. Map the online adder onto Spartan FPGAs using the fast-carry resources. The grey background highlights the 4:2 compressor. Dotted circle indicates the logic block (LB) which can be mapped to the FPGA carry resources.

2) multiple operands addition by designing compressor trees based on bit counters [16], [17];
3) modifying existing FPGA architecture for more efficient digit-parallel online addition [18] and for specific applications [19].

Type 2 and type 3 are beyond the scope of this paper, as our interests are in the field of two-operand online addition that performs $Z = X + Y$, as shown in Fig. 2, and we are using the main-stream FPGAs for implementation.

Specifically in type 1, both works took advantage of the built-in carry resources in FPGAs. Conventionally the ASIC implementation of online adders is based on the 4:2 compressors, as shown within the gray background in Fig. 3. However, directly applying this approach in the FPGAs could be less efficient. This is because there is no carry propagation between the 2 full adders within a 4:2 compressor, and the net delay between them can be large. Instead, Kamp et al and Ortiz et al described very similar mapping techniques for online adders with 2 different data representations, respectively. The main idea is to map the logic block within the dotted circle in Fig. 3. In this case, the fast-carry logic in the FPGA can be employed, and the delay between the 2 FAs is largely reduced.

However, we notice the major limitations of both approaches that they only target on FPGAs with 4-input LUTs and 2 LUTs within a logic slice, such as the Xilinx Spartan series and the Altera Cyclone series. This is naturally reasonable because one LB can be mapped to a single slice. Nevertheless, for FPGAs with 6-input LUTs (6-LUT) and 4 LUTs in a slice, such as the Xilinx Virtex series and all Xilinx 7 series FPGAs, directly applying the approaches in [14] or [15] will result in either resource waste or logic fault. For instance, if two logic blocks are mapped to a slice with 4 LUTs as seen in Fig. 4, the outputs of `LB1` will be faulty because the its carry input cannot be explicitly initialized.

B. Proposed Mapping Method

To tackle this problem, we first modify the structure of the online adder to enable an efficient FPGA mapping. The structure of a 4-digit online adder is given as an example in Fig. 5. In this equivalent structure, the first FA in each 4:2 compressor is split into 2 parts, which only generate carry and sum respectively. In this case they can be mapped individually

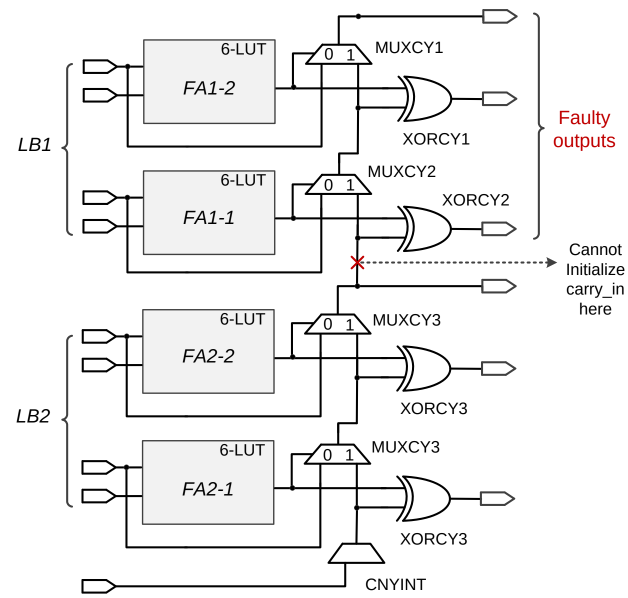


Fig. 4. An example illustrating that direct applying approaches in [14] on a Virtex-6 FPGA will results in faulty outputs, because the carry input of `LB1` cannot be explicitly initialized.

on 2 LUTs. The second FA, which generates the outputs, is unchanged and can be implemented using the fast-carry logic.

The detailed slice mapping of the 2 LBs is shown in Fig. 6. The I/O signals are identical to the previous example in Fig. 5. It can be seen that a 6-LUT can be configured with two different output ports O6 and O5. For LB1, the carry input can be initialized by setting the O6 of LUT2 equal to 0 constantly. In this case, the output of MUXCY2 is O5 of LUT2, and the carry from LB2 will not affect the results of LB1. Using this mapping method the resources within 1 slice can be fully utilized, potentially leads to significant area reduction.

In this mapping approach, the theoretical area figures of the online adder in terms of the number of LUTs and Slices with respect to the operand word-lengths (N) are given in (4) and (5), respectively.

$$OAarea\_LUT = 2N \tag{4}$$

$$OAarea\_Slice = 1 + \left\lceil \frac{N-2}{2} \right\rceil \tag{5}$$

C. Performance Analysis

In our experiments we compare 3 types of adders using Virtex-6 FPGAs: ripple carry adder (`RCA`), online adder which is implemented using behavioural description based on 4:2 compressors (`OA_behv`) and online adder which is built using our proposed approach (`OA_new`). Notice that in order to utilize the carry resources and to avoid logic optimizations by the synthesis tool, the Xilinx primitive component "`Carry4`" is used to create `OA_new`.

The rated frequencies of all designs for a variety of operand word-lengths are shown in Fig. 7. The frequency values

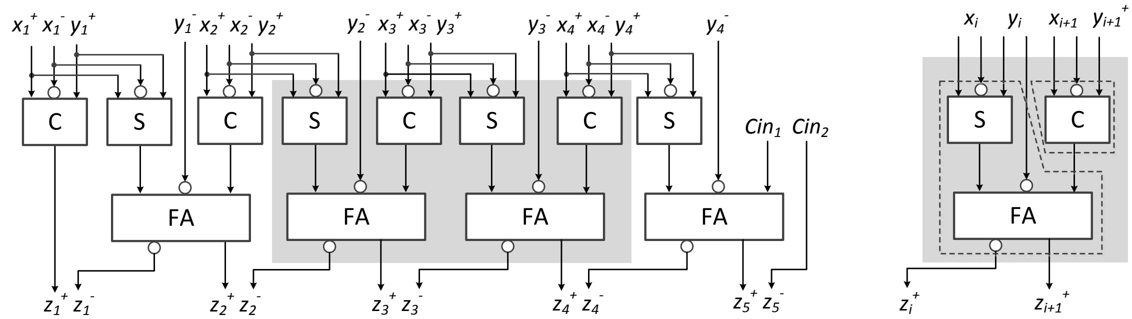


Fig. 5. Modified structure of online adder. Left: an example of 4-digit online adder. The shaded part refers to the 2 logic blocks (LBs) that can be mapped onto 1 slice. Right: one LB. the dotted box outlines the logic that can be mapped onto 1 LUT and the corresponding fast-carry logic.

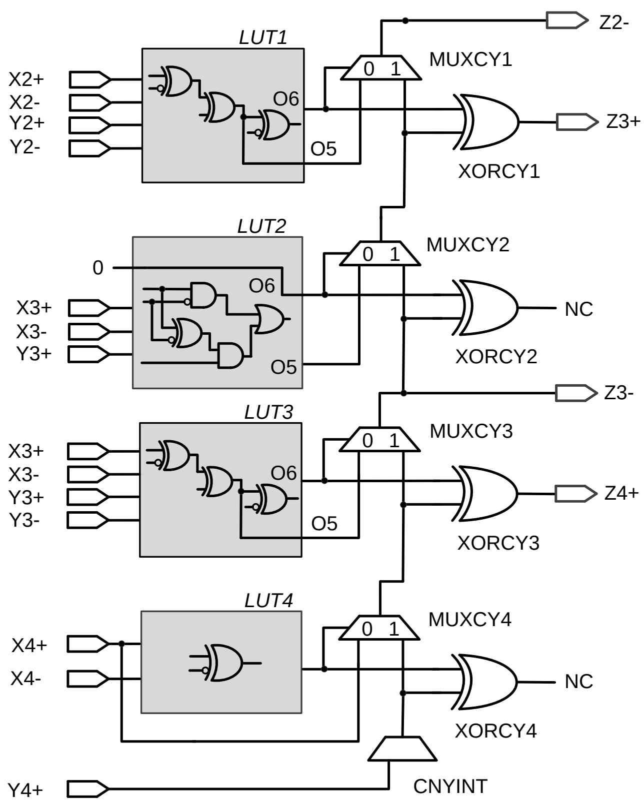


Fig. 6. Implementation of 2 logic blocks (LBs) in 1 FPGA slice which contains four 6-LUTs. NC stands for "Not Care".

are obtained through Xilinx Timing Analyzer after placing and routing in ISE 14.1. It can be seen that our approach outperforms RCA when $N > 8$, and that our approach is always faster than the online adder based on 4:2 compressors. Also we can see that the frequency of RCA decreases drastically with respect to the increment of the operand word-length, whereas the online adder maintains a relatively high frequency even for large designs. This is because the critical path of the online adder is irrelevant to the operand word-lengths.

The area comparisons are demonstrated in Fig.8. The

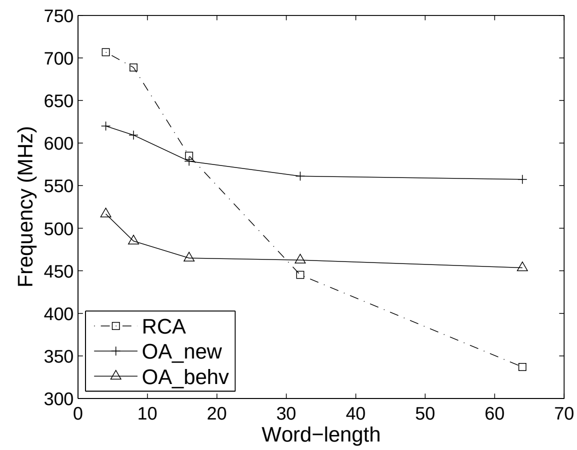


Fig. 7. Rated frequencies of the RCA and the online adder with different implementation methods. The results are obtained from post place and route timing reports in ISE 14.1.

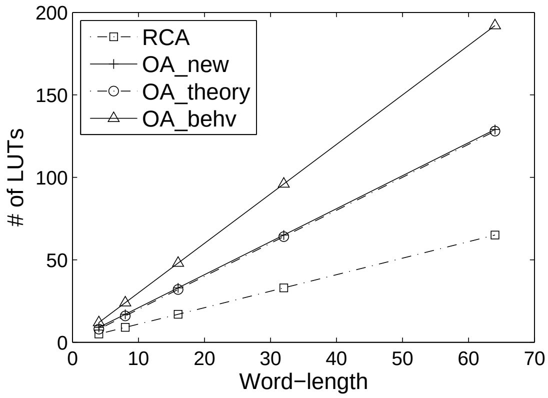
theoretical area of the online adder (`OA_theory`) based on (4) and (5) are also included. It can be seen that `OA_behv` is not area efficient for FPGAs, as the dedicated FPGA resources are not fully utilized. In comparison, our implementation achieves significant area savings: $25\% \sim 33\%$ in LUTs and $67\% \sim 77\%$ in Slices. The area overhead of our approach against RCA is $1.80\times \sim 1.98\times$ in LUTs and $1.00\times \sim 1.88\times$ in Slices. However this is close to the theoretical minimum area overhead for online adders.

IV. Digit Parallel Online Multiplier on FPGAs

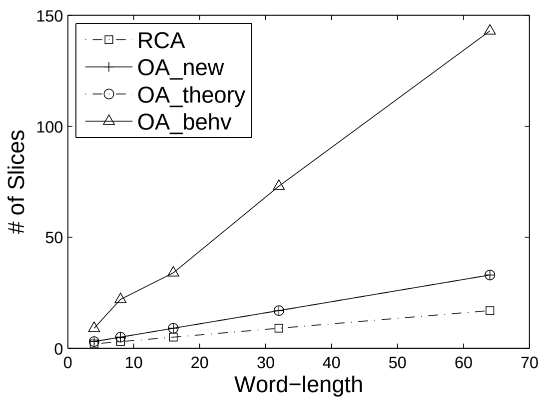
A. Algorithm of Digit Parallel Online Multiplication

Algorithm xxx as described in Section.xxx can be synthesized into a unrolled digit parallel structure, which has been demonstrated to be "overclocking friendly" because timing errors initially affect the least significant digits [xxx]. This synthesis method is straightforward and can be utilized to design other digit parallel online operators.

However, in order to implement the digit parallel online multiplier (OM), large area budget is required because of the large area overhead of a single iteration stage and the overall stage numbers required. For instance in an N-digit OM, totally



(a) Usage of LUTs.



(b) Usage of Slices

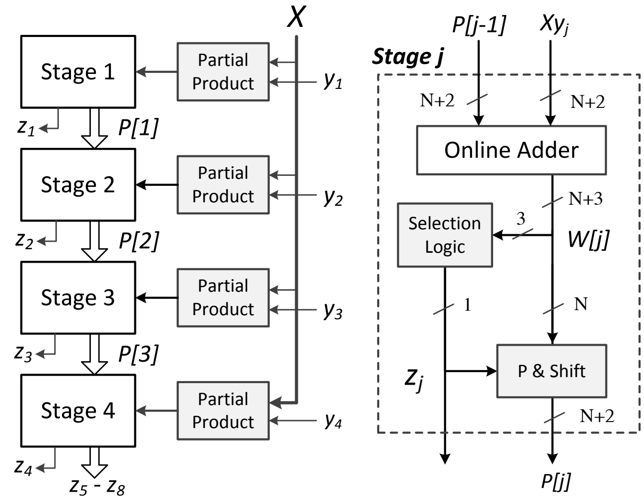
Fig. 8. Area comparisons of different binary adder implementations.

$(2N+\delta)$ iterations are required to generate $2N$ digits outputs, and $(N+\delta)$ iterations for the most significant N digits result. Each iteration stage consists of all operations in Eq.xxx. In comparison, in an N-digit array multiplier there are only $(N-1)$ rows, each of which is basically an N-digit ripple-carry adder.

Instead of simply implementing the unrolled format of Algorithm xxx, it can be optimized given that all inputs are digit parallel. Initially the online delay δ is employed to generate the MSD of the results only based on partial information of the inputs. Nevertheless if all digits of inputs are given simultaneously, δ is no longer necessary. Due to the same reason, $H[j]$, which takes 1 digit of input per stage as shown in Algorithm xxx, can be optimized to take one partial product Xy_j or Yx_j per stage. The optimized digit parallel online multiplication algorithm is described in Algorithm xxx.

B. FPGA Implementation

The structure diagram of a 4-digit `OM` using the proposed algorithm is shown in Fig. 9(a). Generally in an N-digit `OM`, there are only N stages to generate results with $2N$ digits. Each stage can be efficiently implemented using FPGAs, and the structure diagram of stage j is shown in Fig. 9(b). In each stage, an online adder is used to derive $W_{[j]}$. The selection logic, as described in Section xxx and Eq.xxx, takes 3 digits input (6 bits) and generate 1 digit output (2 bits). Hence it



(a) (b)

Fig. 9. (a) Structural diagram of a 4-digit online multiplier using the proposed algorithm. (b) Structure of one stage. The word-length of all signals are labelled in terms of the number of digits. N denotes the word-length of the input signal.

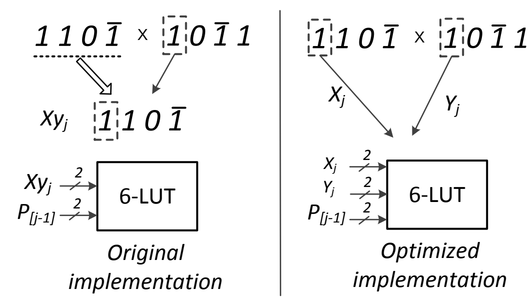
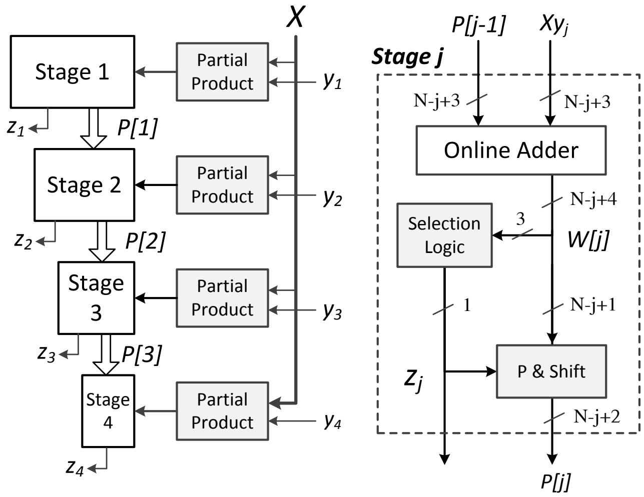


Fig. 10. An example of achieving area reduction by combining the logic blocks that generate partial products into the Online Adder by fully utilizing the 6-LUTs.

can be implemented using two 6-LUTs for each output bit. Similarly the generation of $P_{[j]}$ can also be implemented using one 6-LUT, since only the integer parts of $W_{[j]}$ might change due to the selection of z_j. In addition, the structure of Stage 1 can be further optimized by removing the online adder, because $P_{[0]} = 0$.

Notice that the blocks that generate partial products Xy_j can be incorporated into the online adder for further area reduction. In the online adder as seen in Fig. 6, maximally only 4 inputs per LUT are used. This leaves 2 available inputs per LUT. However in order to generate 1 digit partial product, 1 digit of inputs X and Y are required respectively, as indicated in Fig. 10. Therefore instead of using extra logic to generate Xy_j, it can be incorported into the corresponding LUTs in the online adder by fully utilizing all 6 LUT inputs.

The theoretical minimum resource usages of the proposed `OM` can be calculated as follows. In an N-digit `OM`, overall $N-1$ online adders are needed. According to Algorithm xxx, the word-length of each online adder is $N+2$-digit (2 integer digits and N fractional digits). Therefore based on (4) and (5),



(a) (b)

Fig. 11. (a) Modified structure of a 4-digit online multiplier which only generates the most significant 4-digit result.

the number of LUTs and the number of Slices used by the OM is given in (6) and (7), respectively. Note that L_{S1} and S_{S1} denote the LUT and Slice usages of Stage 1, which is purely built with combinational logic without online adders.

$$OMarea\_LUT = 2(N+2)(N-1) + L_{S1} \tag{6}$$

$$OMarea\_Slice = (1 + \left\lceil \frac{N}{2} \right\rceil)(N-1) + S_{S1} \tag{7}$$

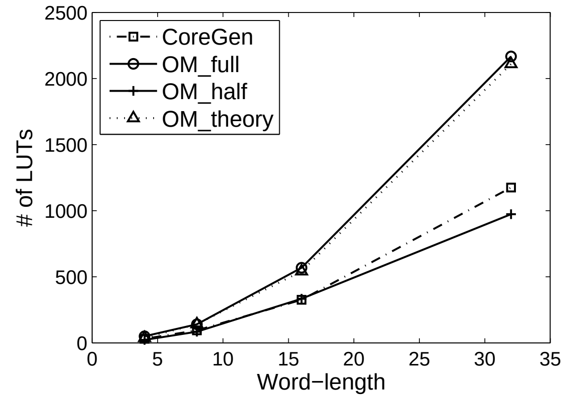
C. Structure Optimization for the MSD Half of the Results

Normally the OM is connected with other arithmetic operators in real applications. If the outputs of an OM is utilized for subsequent operations, only the most significant half of the products are required to maintain the consistency of word-length between inputs and outputs. In the conventional multiplier with standard binary arithmetic, this is achieved by either truncating or rounding the least significant half of the products. However both the computation time and the structure remains unchanged, because the results are generated from LSDs.

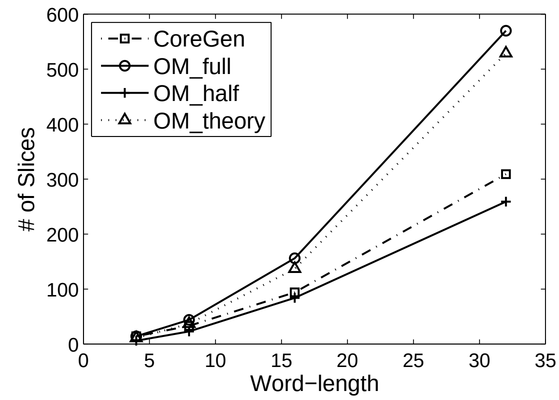
In comparison, the online multiplier offers the possibility to optimize the structure to a further extent if only the MSD half of the results are required, as the results are generated initially from the MSD. The modified structure diagram is illustrated in Fig. 11(a). The word-length of the online adder in a given Stage j can be reduced, as shown in Fig. 11(b). This is because there is no carry propagation in the online adder, and all digits of the online adder are obtained in parallel.

D. Performance Analysis

In the experiments we compare 3 types of binary multipliers. One is created from the Xilinx Core Generator with standard arithmetic. The CoreGen multiplier is implemented



(a) Usage of LUTs.



(b) Usage of Slices

Fig. 12. Area comparisons of different binary multipliers.

based on LUTs, and it is configured with speed optimization [20] . The other two multipliers are both built with online arithmetic, except that the products are generated with full precision (OM_full) and half precision (OM_half), respectively.

We first compare the maximum frequencies of the multipliers for a variety of operand word-lengths. Similar to the experiments in Section xxx, we record both the frequencies from the Xilinx Timing Analyzer and the maximum error-free frequencies acquired through tests, in which the input patterns are sampled from the UI data. The results are presented in Fig.xxx.

The comparison of area in terms of the number of LUTs and Slices with respect to different operand word-lengths are illustrated in Fig. 12. It can be seen that the area of OM_full is close to the theoretical values predicted by (6) and (7). In comparison to the CoreGen multiplier, the area overheads of OM_full are $48\% \sim 84\%$ for LUTs, and $0\% \sim 83\%$ for Slices. On the other hand, the OM_half achieves area reductions for all operand word-lengths when compared to the CoreGen multiplier. The area saving varies from 40% to 55% for LUTs, and $46\% \sim 55\%$ for Slices.

Although our proposed new online multiplier demonstrates a better operating frequency in the practice, we can push this one step further by operating the circuits beyond their deterministic region and allowing timing violations to happen. In this case, we examine the overclocking behavior of all 3 types of multipliers in terms of the mean relative error (MRE), which is given in (xxx). For instance, Fig.xxx demonstrates the MRE of the 8-digit multipliers. [21]

V. Conclusion

The conclusion goes here.

Acknowledgment

References

[1] M. D. Ercegovac, "On-line arithmetic: An overview," in *Proc. Annual Technical Symp. Real time signal processing VII*, 1984, pp. 86–93.

[2] M. D. Ercegovac and T. Lang, *Digital arithmetic*. Morgan Kaufmann, 2003.

[3] Xilinx Inc., "Virtex-6 FPGA configurable logic block user guide."

[4] T. Preusser and R. Spallek, "Enhancing FPGA device capabilities by the automatic logic mapping to additive carry chains," in *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, Aug 2010, pp. 318–325.

[5] ——, "Mapping basic prefix computations to fast carry-chain structures," in *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, Aug 2009, pp. 604–608.

[6] S. Hauck, M. Hosler, and T. Fry, "High-performance carry chains for FPGA's," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 2, pp. 138–147, April 2000.

[7] M. Frederick and A. Somani, "Multi-bit carry chains for high-performance reconfigurable fabrics," in *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, Aug 2006, pp. 1–6.

[8] Xilinx Inc., "7 series FPGAs configurable logic block user guide," 2013.

[9] T. Kim, W. Jao, and S. Tjiang, "Circuit optimization using carry-save-adder cells," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 974–984, 1998.

[10] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, 1961.

[11] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi, "A high-speed multiplier using a redundant binary adder tree," *IEEE Jourl. of Solid-State Circuits*, vol. 22, no. 1, pp. 28–34, 1987.

[12] K. S. Trivedi and M. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. on Computers*, vol. 26, pp. 161–167, 1975.

[13] R. Galli and A. Tenca, "A design methodology for networks of online modules and its application to the levinson-durbin algorithm," *IEEE Trans. on VLSI*, vol. 12, no. 1, pp. 52–66, 2004.

[14] W. Kamp, A. Bainbridge-Smith, and M. Hayes, "Efficient implementation of fast redundant number adders for long word-lengths in FPGAs," in *Proc. Int. Conf. Field-Programmable Technology (FPT)*, Dec 2009, pp. 239–246.

[15] J. Hormigo, M. Ortiz, F. Quiles, F. J. Jaime, J. Villalba, and E. L. Zapata, "Efficient implementation of carry-save adders in FPGAs," in *Proc. Int. Conf. Application-specific Systems, Architectures and Processors (ASAP)*, 2009, pp. 207–210.

[16] J. Hormigo, J. Villalba, and E. L. Zapata, "Multioperand redundant adders on FPGAs," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2013–2025, 2013.

[17] T. Matsunaga, S. Kimura, and Y. Matsunaga, "Multi-operand adder synthesis on FPGAs using generalized parallel counters," in *Proc. Asia and South Pacific Design Automation Conference (ASPDAC)*, 2010, pp. 337–342.

[18] P. Brisk, A. Verma, P. Ienne, and H. Parandeh-Afshar, "Enhancing FPGA performance for arithmetic circuits," in *Proc. Design Automation Conference (DAC)*, June 2007, pp. 334–337.

[19] R. Gutierrez, J. Valls, and A. Perez-Pascual, "FPGA-implementation of time-multiplexed multiple constant multiplication based on carry-save arithmetic," in *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, Aug 2009, pp. 609–612.

[20] Xilinx Inc., "LogiCORE IP multiplier v11.2," 2011.

[21] M. Ercegovac and T. Lang, "On-line arithmetic for DSP applications," in *Proc. Midwest Symp. Circuits and Systems*, 1989, pp. 365–368 vol.1.