

Efficient FPGA Implementation of Digit Parallel Online Arithmetic Operators

Authors removed for blind review

Abstract—Online arithmetic has been widely utilized for ASIC implementations, as it is designed to perform computations from the most significant digits. Recent research also shows that digit parallel online operators can fail more gracefully when operating beyond the deterministic region, in comparison to operators with conventional arithmetic. Unfortunately, online arithmetic operators require a large area overhead, while efficient implementation of general purpose online operators on FPGAs remains under investigation. In this paper, we propose novel approaches to implement key primitives of online arithmetic, such as adders and multipliers, efficiently on modern FPGAs with 6-input LUTs. The FPGA carry resources are used for both area reduction and performance improvements. We demonstrate experimentally that the proposed architectures achieve significant area savings and frequency speed-ups compared to the original implementations.

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 among multiple operations, and the overall computation latency can be significantly reduced. A brief overview of online arithmetic is given in Section II.

Recent research has demonstrated another key advantage of online arithmetic is that it allows gracefully degradation when operating beyond the deterministic region, e.g. under overclocking [3]. 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 forms of computer arithmetic do 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 occur 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 [4]. There is a large amount of studies exploring methods that can effectively map circuits using the carry logic for performance improvements [5], [6], or propose alternative carry logic for FPGAs [7], [8]. Furthermore, hard DSP blocks are also included to boost multiplications and multiply-accumulations [9]. However, most of these approaches are designed to accelerate computations with the conventional form of arithmetic.

For online arithmetic, we notice from existing literature that there are two directions 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 implementations of other key arithmetic primitives, such as online multipliers, lack exploration. Second, we found that existing approaches of building general purpose online adders are specifically for low cost FPGAs with 4-input LUTs and 2 LUTs within a Slice, such as the Altera Cyclone III [10] and Xilinx Spartan 3 series [11]. However, as will be discussed in Section III, these approaches cannot be directly applied on many modern FPGA with 6-input LUTs and 4 LUTs in a Slice, such as the Xilinx Virtex series FPGAs and all Xilinx 7 series FPGAs [12].

In this paper, both issues will be 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 Section IV, we optimize the online multiplication algorithm to yield an efficient FPGA implementation of digit parallel online multiplier. In addition, the theoretical resource usage of both operators is given. We demonstrate experimentally on Xilinx Virtex-6 FPGAs that the proposed designs achieve significant area reduction and performance gain over their original implementations. In summary, 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;
- 3) Theoretical area models of the proposed architectures.

II. BACKGROUND: ONLINE ARITHMETIC

A. Key Features of Online Arithmetic

Online arithmetic has been widely used in numerous applications such as signal processing and control algorithms [13], [14]. Online arithmetic was originally designed for digit-serial

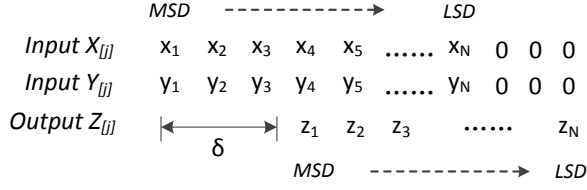


Fig. 1. Dataflow in digit-serial online arithmetic, in which both inputs and outputs are processed from the MSD to the LSD. δ denotes the online delay.

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 small constant, which is independent of the precision in a given operation. For ease of discussion, for the rest of this paper, the input data is 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].

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

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) [15] and signed-digit (SD) [16]. For instance with SD representation, each digit is represented using a redundant digit set $\{-a, \dots, -1, 0, 1, \dots, a\}$, where $a \in [r/2, r-1]$. In comparison, the standard non-redundant representation only uses a digit set $\{0, \dots, r-1\}$. Thus a standard number corresponds to several possible redundant representations. For example, the two’s complement number 0.011 can be represented in the binary online form as 0.111, 0.101 and 0.011 among many other possible representations.

Due to the redundancy, the MSDs of the result can be calculated using partial information from both inputs. Then the value of the number can be revised using the subsequent 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 used directly. The structure diagram of an online adder with all signals represented with SD numbers is shown in Fig. 2. The module 3:2 denotes a 3:2 compressor, which takes 3 inputs and generates 2 outputs. Normally it can be implemented on the basis of a full adder (FA). A 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 labelled in Fig. 2, ideally the computation delay of this adder is only 2 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 [17].

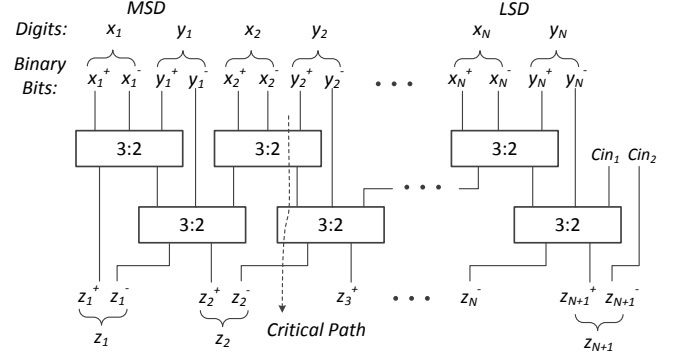


Fig. 2. An N -digit binary digit-parallel online adder. Both inputs and outputs are represented using SD representation. Module 3:2 denotes the 3:2 compressor.

Algorithm 1 Online Multiplication

- 1: **Initialization:** $X_{[-\delta]} = Y_{[-\delta]} = P_{[-\delta]} = 0$
- 2: **for** $j = -\delta, -\delta + 1, \dots, 2N - 1$ **do**
- 3: $H_{[j]} \leftarrow r^{-\delta} (x_{j+\delta+1} \cdot Y_{[j+1]} + y_{j+\delta+1} \cdot X_{[j]})$
- 4: $W_{[j]} \leftarrow P_{[j]} + H_{[j]}$
- 5: $z_j \leftarrow \text{sel}(W_{[j]})$
- 6: $P_{[j+1]} \leftarrow r (W_{[j]} - Z_{[j]})$
- 7: **end for**

C. Binary Online Multiplication

Multiplication is another key arithmetic operator. Typically, online multiplication is performed in a recursive digit-serial manner, as illustrated in Algorithm 1 [18] where both inputs and outputs are N -digit numbers as given in (1). For a given iteration j , the product digit z_j is generated MSD-first through a selection function $\text{sel}()$. For any radix r and chosen digit set, there exists an appropriate selection method and a value of δ which ensure convergence [18]. 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 $\text{sel}()$ is given by (2) [19]. Notice that the selection is based on 2 integer digits and 1 fractional digit of $W_{[j]}$.

$$\text{sel}(W_{[j]}) = \begin{cases} 1 & \text{if } W_{[j]} \geq \frac{1}{2} \\ 0 & \text{if } -\frac{1}{2} \leq W_{[j]} < \frac{1}{2} \\ \bar{1} & \text{if } W_{[j]} < -\frac{1}{2} \end{cases} \quad (2)$$

It is worth noting that the precision of the product Z can be determined by directly truncating from the LSDs, since there is no carry propagations from the LSD to the MSD. This is useful especially when the multiplication product is used by successive computations, where only the first N digits of the results are normally used. In comparison, the product is generated starting from the LSD with conventional arithmetic. Hence the successive operations cannot commence until the result with full-precision is generated. This feature of online arithmetic has inspired research about dynamically control of computation precision in numerous applications [20].

In Section IV, we will first describe an optimized online multiplication algorithm and its FPGA implementation, which

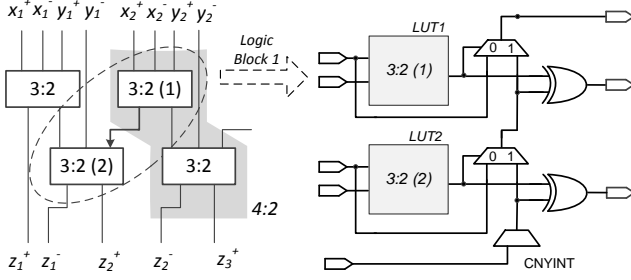


Fig. 3. Map the online adder onto Spartan FPGAs using the fast-carry resources. The grey background highlights the 4:2 compressor. Dotted circle indicates the logic block (LB) which can be mapped to the FPGA using the carry resources.

are designed specifically targeting on digit parallel operations. Then we provide a modified structure which only generates the most significant half of the product in order to achieve extra area savings.

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 [21], [22];
- 2) multiple operands addition by designing compressor trees based on bit counters [23], [24];
- 3) modifying existing FPGA architecture for more efficient digit-parallel online addition [25] and for specific applications [26].

Type 2 examines the use of counters to sum k N -digit numbers where $k > 2$. Instead, we focus on two-operand online addition that performs $Z = X + Y$, as shown in Fig. 2. In comparison to type 3, this paper attempts to obtain the best performance from mainstream FPGAs instead of suggesting ways to modifying the FPGA fabric.

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 outlined in Fig. 3 within the gray background. However, directly applying this approach in the FPGAs could be less efficient. This is because there is no carry propagation between the 2 FAs within a 4:2 compressor, and the net delay between them can be large. Instead, Kamp et al [21] and Hormigo et al [22] described very similar mapping techniques for online adders with SD representations and CS representations, respectively. They share the common idea that maps the logic block (LB) 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 can be largely reduced.

However, we notice the major limitations of both approaches that they only target on FPGAs with 4-input LUTs (4-LUT) and 2 LUTs within a logic Slice, such as the Xilinx Spartan series and the Altera Cyclone series. This is naturally

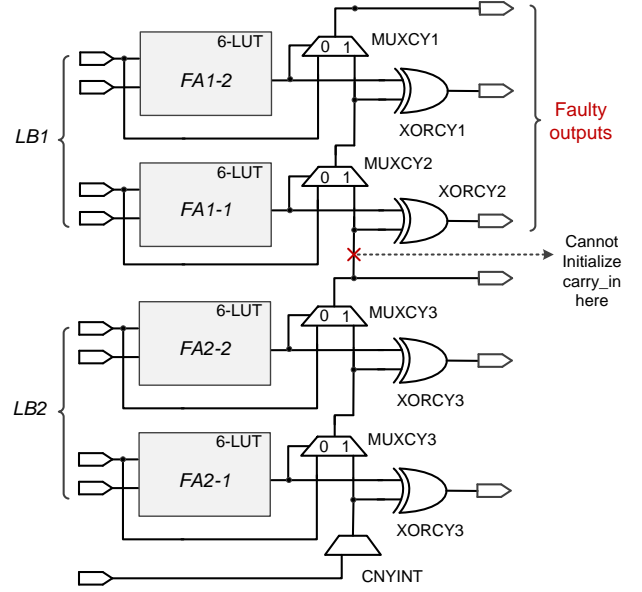


Fig. 4. An example illustrating that direct applying approaches in [21] on a Virtex-6 FPGA will result in faulty outputs, because the carry input of LB1 cannot be explicitly initialized.

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 these approaches will result in either resource waste or logic fault. For instance, if two LBs are mapped to a Slice with 4 LUTs as seen in Fig. 4, the outputs of LB1 will be faulty because its carry input cannot be explicitly initialized. Due to the same reason, the output of XORCY2 should not be used.

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 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 and the logic in each LUT are shown in Fig. 6. The I/O signals are identical to the previous example in Fig. 5. Normally 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 to 0. In this case, the output of MUXCY2 is always O5 of LUT2, and the carry from LB2 will not affect the result of LB1. For those logics to generate the MSD and the LSD, they can be combined into 1 Slice as observed from Fig. 5. Therefore using this mapping method the resources within 1 Slice can be fully utilized, leading to a significant area reduction.

In this mapping approach, the area figures of the online adder in terms of the number of LUTs and Slices with respect

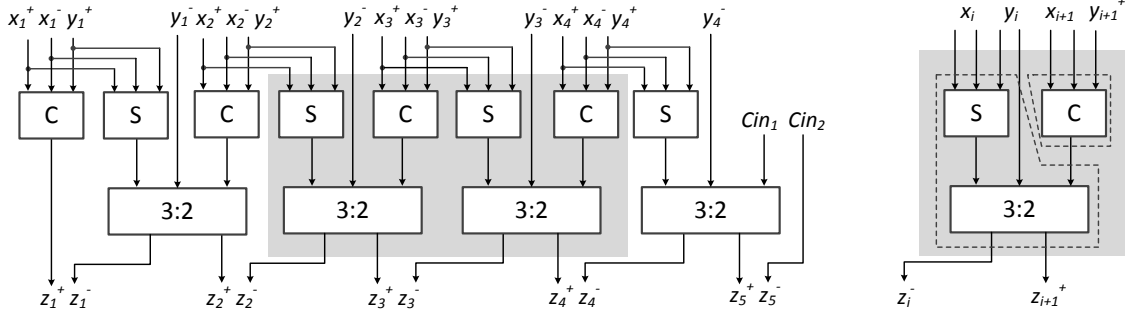


Fig. 5. Modified structure of online adder. Left: an example of 4-digit online adder. The shaded part refers to the 2 logic blocks (LBs) that can be mapped onto 1 Slice. Right: one LB. The dotted box outlines the logic that can be mapped onto 1 LUT and the corresponding fast-carry logic.

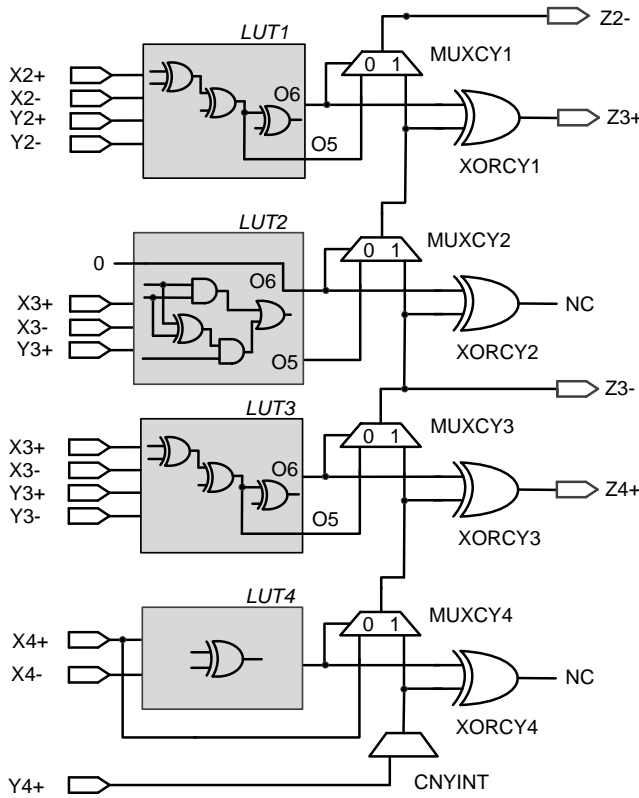


Fig. 6. Implementation of 2 logic blocks (LBs) in 1 FPGA Slice which contains four 6-LUTs. NC stands for "Not Care".

to the operand word-lengths (N) are given in (3) and (4), respectively.

$$OA_{LUT} = 2N \quad (3)$$

$$OA_{Slice} = 1 + \left\lceil \frac{N-2}{2} \right\rceil \quad (4)$$

C. Performance Analysis

In our experiments, we compare the proposed online adder (OA_new) against the original design which is implemented

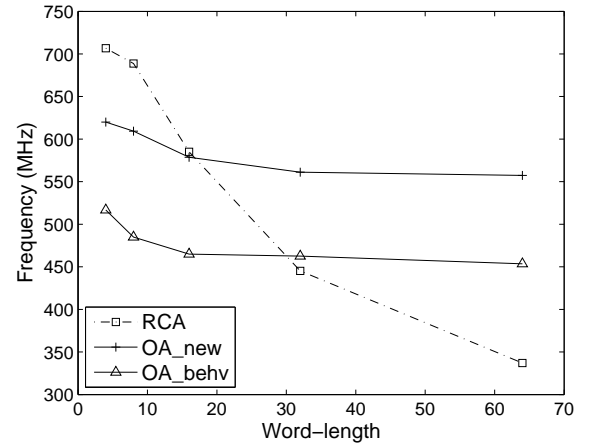


Fig. 7. Rated frequencies of the RCA and the online adder with different implementation methods. The results are obtained from post place and route timing reports in ISE 14.1.

using behavioural descriptions based on 4:2 compressors (OA_behv). In addition, we also include ripple carry adder (RCA) which uses conventional arithmetic, into the comparisons. All designs are targeting on Xilinx Virtex-6 FPGA with speed grade -1. 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.

We record the rated frequencies of all designs when varying the operand word-lengths. The results are shown in Fig. 7. The frequency values are obtained through Xilinx Timing Analyzer after placing and routing the designs in ISE 14.1. It can be seen that both designs with online arithmetic achieves relatively stable operating frequencies across a variety of operand word-lengths. This is as expected, because the critical path delay in the online adder is theoretically a fixed value and it is irrelevant to the operand precisions. In comparison, the rated frequency of the RCA drops drastically for larger word-lengths. It follows that both online adders outperform RCA for large word-lengths. Furthermore, our approach operates faster than OA_behv; the speed-ups of OA_new over OA_behv are from 20% to 26%.

We also compare the resource usages of all three adder designs with respect to a variety of operand word-lengths. The

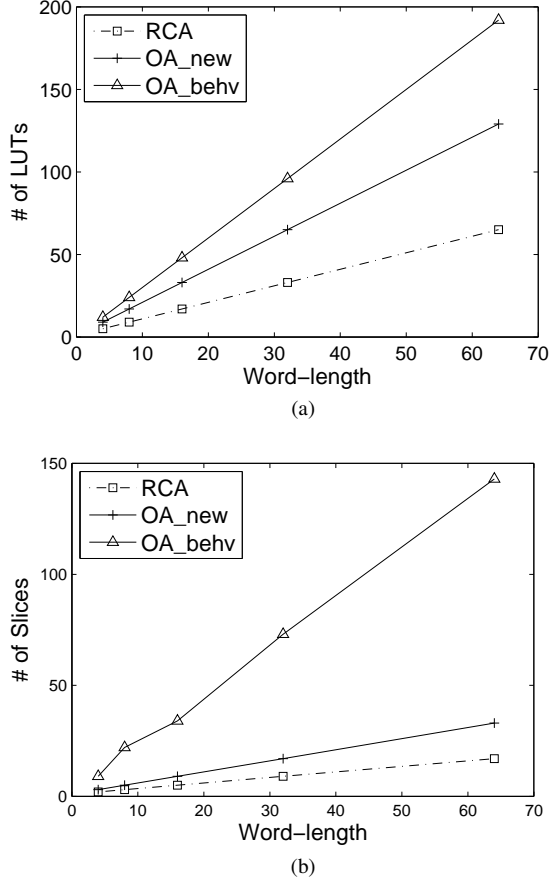


Fig. 8. Area comparisons of different binary adder implementations with respect to a variety of operand word-lengths. (a) Usage of LUTs. (b) Usage of Slices.

results are presented in Fig 8. We can see that in comparison to the original OA_behv, our implementation OA_new achieves significant area savings: 25% ~ 33% in LUTs and 67% ~ 77% in Slices. This is because the dedicated FPGA resources are fully utilized. However, our design still costs more area than an RCA, with an area overhead of 80% ~ 98% in terms of LUTs, and up to 88% in terms of Slices. This is because for a single adder, the area overhead brought by the redundant number representations cannot be removed.

IV. DIGIT PARALLEL ONLINE MULTIPLIER ON FPGAS

A. Algorithm of Digit Parallel Online Multiplication

Algorithm 1 as described in Section II can be synthesized into a unrolled digit parallel structure. Recent study has shown that the digit-parallel online multiplier (OM) is also more tolerant to timing violations in comparison to the conventional arithmetic, because timing errors initially affect the LSDs and this will lead to smaller error magnitude [3]. This implementation method is straightforward and can be directly utilized to design other digit parallel online operators.

On the other hand, significant area overhead is required in order to implement this type of OM. This is due to three main reasons. First, in comparison to the conventional multiplier, more iteration stages are needed. For instance in an N -digit OM, $(2N + \delta)$ iterations are required to generate $2N$ digits

Algorithm 2 Digit Parallel Online Multiplication

```

1: Initialization:  $P_{[0]} = 0$ 
2: for  $j = 1, 2, \dots, N$  do
3:    $Xy_j \leftarrow X \cdot y_j$ 
4:    $W_{[j]} \leftarrow P_{[j-1]} + Xy_j$ 
5:    $Z_{[j]} \leftarrow sel(W_{[j]})$ 
6:    $P_{[j+1]} \leftarrow r(W_{[j]} - Z_{[j]})$ 
7: end for
8:  $Z_{[N+1:2N]} \leftarrow frac(W_{[N]})$ 

```

outputs, or $(N + \delta)$ iterations are required to generate the most significant N digits. Second, the logic of each stage is complex as it involves operations such as online addition, digit-vector multiplication, shifting and other combination logic blocks such as the selection function. Also, the word-length of signals $H_{[j]}$, $W_{[j]}$ and $P_{[j]}$ is $(N + 2 + \delta)$, because the selection function needs 2 integer digits and 1 fractional digits to generate z_j , as described in Section II. In comparison, an N -digit array multiplier, there are only $(N - 1)$ rows, each of which is basically an N -digit ripple-carry adder. Third, this OM architecture is not optimized specifically for FPGA technology.

We address the first 2 design issues by providing an alternative online multiplication algorithm. Instead of simply implementing the unrolled format of Algorithm 1, it can be optimized specifically for digit parallel operations. We notice that the online delay δ is employed in Algorithm 1 because the input data is available in a digit serial fashion. Hence δ is used to accumulate enough input digits to generate the MSD of the result. Nevertheless if all digits of inputs are available simultaneously, δ is no longer necessary. Due to the same reason, $H_{[j]}$, which takes one digit of each input per stage as shown in the original algorithm, can be optimized to take one partial product Xy_j or Yx_j per stage. To summarize, the modified algorithm for digit parallel online multiplication is described in Algorithm 2, where N denotes the operand word-length, function $frac(W_{[N]})$ refers to select the fractional digits of $W_{[N]}$, and the selection function $sel()$ is given previously in (2).

In comparison to Algorithm 1, the maximum stages required to generate results with full precision drops by a factor of 2 using the proposed new algorithm. In each stage, the word-length of all signals also reduces from $(N + 2 + \delta)$ to $(N + 2)$, as δ is not necessary for digit parallel operations. In addition, the $frac()$ function can be implemented only based on wire connections without using other logic resources.

B. FPGA Implementation

We now present an area efficient implementation of Algorithm 2 on FPGAs. For example, the general structure diagram of a 4-digit OM using the proposed algorithm is shown in Fig. 9(a), and the structure diagram of a single stage j is shown in Fig. 9(b) with the word-lengths of all signals labelled. Area optimizations can be performed on all modules. It can be seen that in each stage, an $N + 2$ -digit online adder is used to derive $W_{[j]}$. Our proposed online adder architecture can be employed. The selection logic takes 3 input digits (6 bits) and generate 1

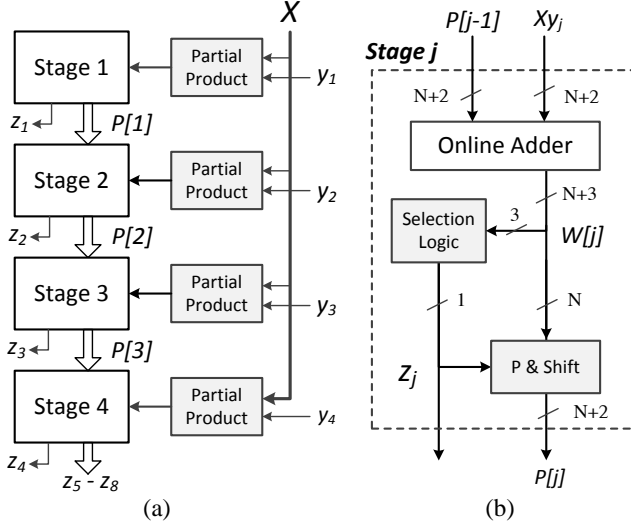


Fig. 9. (a) Structural diagram of a 4-digit online multiplier using the proposed algorithm. (b) Structure of one stage. The word-length of all signals are labelled in terms of the number of digits. N denotes the word-length of the input signals.

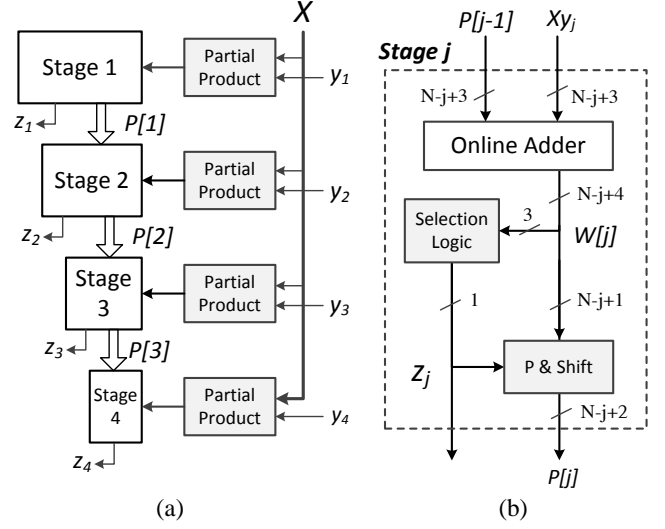


Fig. 11. (a) Modified structure of a 4-digit online multiplier which only generates the most significant 4-digit result. (b) Structure diagram of stage j , with the word-lengths of all internal signals labelled.

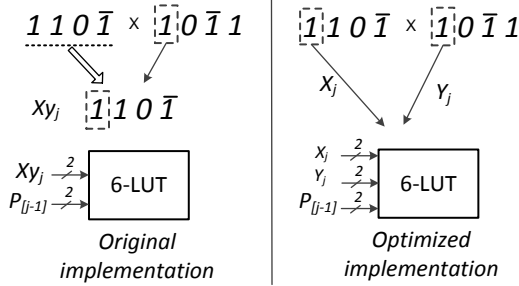


Fig. 10. Left: Direct implementation with extra logic to generate partial products. Right: Combining the logic blocks that generate partial products into the Online Adder by fully utilizing all the inputs of the 6-LUTs. In this case, further area reduction can be achieved.

output digit (2 bits). Hence it 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 digits of $W_{[j]}$ need to be modified due to the selection of $Z_{[j]}$. In addition, the structure of Stage 1 in Fig 9(a) can be further optimized by removing the online adder, because $P_{[0]} = 0$.

The blocks that generate partial products Xy_j can be incorporated into the online adder for further area reduction. In the online adder structure as presented in Fig. 6, we notice that only 4 inputs per LUT are used. This leaves at least 2 available inputs per LUT. Originally in order to generate 1 digit partial product, 1 digit of inputs X and Y are required respectively, as indicated in Fig. 10. Instead of using extra logic to generate Xy_j , it can be merged into the corresponding LUTs in the online adder by fully utilizing all 6 LUT inputs. Notice that this approach is only available for FPGAs with 6-LUTs.

The theoretical minimum resource uses of the proposed OM can be calculated as follows. In an N -digit OM, overall $(N-1)$ online adders are needed. According to Algorithm 2, the word-length of each online adder is $(N+2)$ -digit (2 integer digits

and N fractional digits). Therefore based on (3) and (4), the number of LUTs and the number of Slices used by the OM is given in (5) and (6), respectively. Note that L_{S1} and S_{S1} denote the number of LUTs and Slices used by the first stage, which is built without online adders.

$$OM_LUT = 2(N+2)(N-1) + L_{S1} \quad (5)$$

$$OM_Slice = (1 + \left\lceil \frac{N}{2} \right\rceil)(N-1) + S_{S1} \quad (6)$$

C. Structure Optimization for the MSD Half of the Results

As discussed before, normally the multiplier is connected with other arithmetic operators in real applications. If the outputs of a multiplier is utilized for subsequent operations, and it is chosen to main the consistency of word-length between the I/Os of the system, then only the most significant half of the products are required. In a 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 flexibility to simplify the structure corresponding to the required precision. This is possible because in an OM, the product digits are generated initially from the MSD, and there is no carry propagation from the LSD to the MSD with the employment of the redundant number system. This will potentially lead to a more area efficient design. For example, the modified structure diagram of a 4-digit OM is illustrated in Fig. 11(a). The word-length of all signals within each stage can be correspondingly reduced, as shown in Fig. 11(b). It can be seen that instead of keeping identical word-length throughout all stages, as shown in Fig. 9, in the modified structure the signal word-lengths depend on the stage number j and fewer bits of signals are needed for LSD stages.

D. Performance Analysis

Similar to the experiments as described in Section III-C, we compare the online multiplier based on behavioral descriptions (OM_initial) against the proposed 2 types of multipliers, which are implemented to generate products with full precision (OM_full) and half precision (OM_half), respectively. In addition, we also compare the OM with the conventional binary arithmetic multiplier. The conventional multiplier is created using the Xilinx Core Generator with speed optimization [27], and it is implemented based on LUTs without DSPs.

The comparison of area in terms of the cost of LUTs and the cost of Slices with respect to different operand word-lengths are illustrated in Fig. 12. The results are obtained from ISE 14.1 after mapping the design to the Virtex-6 FPGA. In comparison to OM_initial, significant area savings have been obtained by the optimized design OM_full. Specifically, the LUTs and Slice utilization drops $2.44\times \sim 3.34\times$ and $3.19\times \sim 5.45\times$, respectively.

In comparison to the CoreGen multiplier, the area overheads of OM_full are between 48% and 84% for LUTs and up to 83% for Slices. On the other hand, the OM_half achieves area reductions for all operand word-lengths when compared against the CoreGen multiplier. The area saving varies from 40% to 55% for LUTs, and 46% to 55% for Slices.

We also check the rated frequencies of OM_full, OM_half and CoreGen for a variety of operand word-lengths. The frequency values are still obtained from the post place and route timing reports. Notice that input stimulus to the circuits are randomly sampled from a uniform distribution of N -digit numbers. The results are plotted in Fig.13, from which we can see that there is an increasing gap between the online multipliers and the CoreGen multiplier, especially for large operand word-lengths. This might because in the online multiplier, although the carry resources in the FPGA are efficiently used for area reduction, each 4-digit carry logic within a Slice is divided into two parts, as shown previously in Section III. Therefore the critical path delay of an OM is determined by both the carry logic delays, and the net delay among different stages, which is huge for large operand word-lengths. This also explains the fact that OM_full and OM_half run at similar frequencies, as the critical path of both designs are identical in theory.

However, we can take advantage of the “overclocking friendly” feature of the online arithmetic, and push the previous tests one step further by operating the circuits beyond their deterministic region while allowing timing violations to happen. We examine the overclocking behavior of all 3 types of multipliers in terms of the mean relative error (MRE), which is given in (7), where E_{error} and E_{out} refer to the mean value of error and the mean value of correct output, respectively.

$$MRE = \frac{E_{error}}{E_{out}} \times 100\% \quad (7)$$

For instance, Fig. 14 demonstrates the MRE values of the 8-digit multipliers. We see that although the Timing Analyzer predicts slower frequencies for the OMs, they actually operate faster than CoreGen without the occurrence of timing errors. This is in accordance with the results seen in the previous work [3].

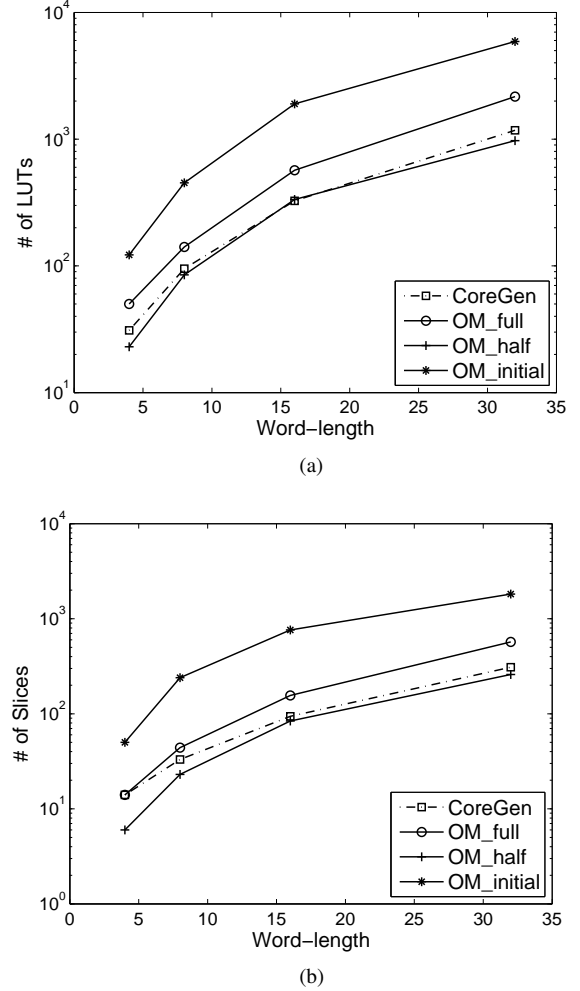


Fig. 12. Area comparisons of different types of binary multipliers. (a) Usage of LUTs. (b) Usage of Slices

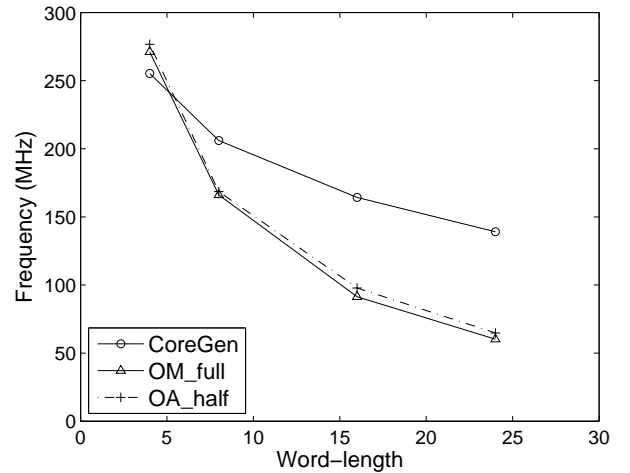


Fig. 13. Rated frequencies of different types of binary multipliers for a variety of operand word-lengths. The results are obtained from the Xilinx Timing Analyzer in ISE 14.1 after placing and routing.

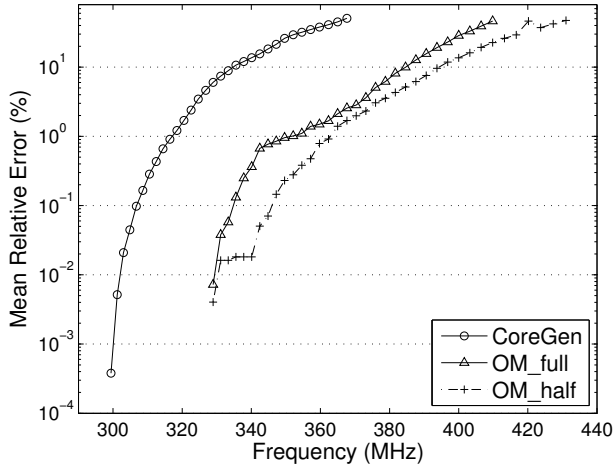


Fig. 14. Mean relative errors seen at the outputs of different types of binary multipliers, when clocked with faster-than-rated frequencies.

V. CONCLUSION

In this paper, we initially describe a novel methodology that can efficiently map digit parallel online adders onto FPGAs. The proposed mapping method targets modern FPGAs, which have 6-input LUTs and 4 LUTs in a Slice, and fully utilizes the built-in carry resources. We have demonstrated analytically and experimentally that the proposed architecture is both area and performance efficient in comparison to the original design. We then propose an area efficient FPGA implementation of the online multiplier, which is specifically optimized for digit parallel operations. We have shown that this also obtains significant area reduction and frequency speed-ups.

In the future we wish to explore alternative mapping methods for the online multiplier that can effectively use carry resources with long word-lengths to improve its performance further. We would also like to combine the proposed online primitives into real-world applications.

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] K. Shi, D. Boland, E. Stott, S. Bayliss, and G. A. Constantinides, "Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs," in *Proc. Design Automation Conference (DAC)*, June 2014.
- [4] Xilinx Inc., "Virtex-6 FPGA configurable logic block user guide."
- [5] 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.
- [6] —, "Mapping basic prefix computations to fast carry-chain structures," in *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, Aug 2009, pp. 604–608.
- [7] S. Hauck, M. Hosler, and T. Fry, "High-performance carry chains for FPGAs," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 2, pp. 138–147, April 2000.
- [8] 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.
- [9] Xilinx Inc., "Virtex-6 FPGA DSP48E1 slice user guide (UG369)," Feb. 2011.
- [10] Altera Co., "Cyclone device handbook," 2008.
- [11] Xilinx Inc., "Spartan-3 generation FPGA user guide (UG331)," Feb. 2008.
- [12] —, "7 series FPGAs configurable logic block user guide," 2013.
- [13] R. Galli and A. F. Tenca, "Design and evaluation of online arithmetic for signal processing applications on FPGAs," in *Proc. SPIE Advanced Signal Processing Algorithms, Architectures and Implementations*, Aug 2001, pp. 134–144.
- [14] M. Dimmler, A. Tisserand, U. Holmbeg, and R. Longchamp, "On-line arithmetic for real-time control of microsystems," *IEEE/ASME Trans. on Mechatronics*, vol. 4, no. 2, pp. 213–217, Jun 1999.
- [15] 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.
- [16] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, 1961.
- [17] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi, "A high-speed multiplier using a redundant binary adder tree," *IEEE J. of Solid-State Circuits*, vol. 22, no. 1, pp. 28–34, 1987.
- [18] K. S. Trivedi and M. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. on Computers*, vol. 26, pp. 161–167, 1975.
- [19] 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.
- [20] S. Rajagopal and J. Cavallaro, "Truncated online arithmetic with applications to communication systems," *IEEE Trans. on Computers*, vol. 55, no. 10, pp. 1240–1252, Oct 2006.
- [21] 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.
- [22] 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.
- [23] 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.
- [24] H. Parandeh-Afshar, A. Verma, P. Brisk, and P. Ienne, "Improving FPGA performance for carry-save arithmetic," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 4, pp. 578–590, April 2010.
- [25] 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.
- [26] 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.
- [27] Xilinx Inc., "LogiCORE IP multiplier v11.2," 2011.