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 operators such as adder and multiplier efficiently on current 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 comparing 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 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

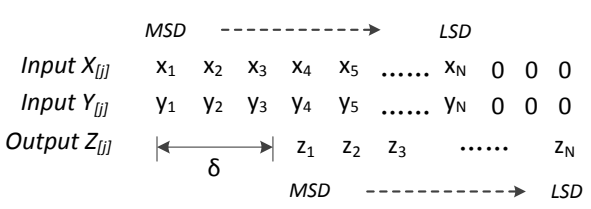


Fig. 1. Dataflow in digit-serial online arithmetic, in which both inputs and outputs are processed from the MSD to the LSD. δ denotes the online delay.

"online delay". Notice that δ is normally a constant, which is 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].

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

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 $0.10\overline{1}$, $0.0\overline{1}1$ 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 structure diagram of an online adder with all signals represented with SD numbers 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 labelled 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].

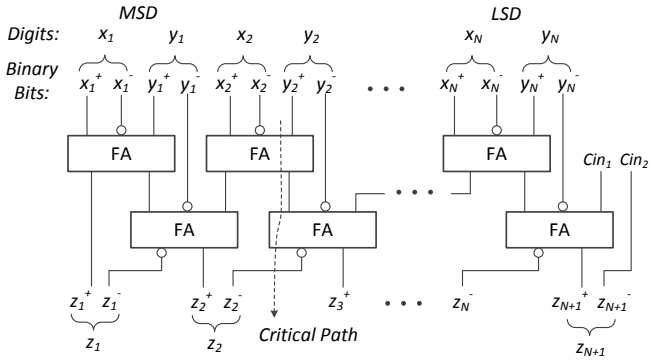


Fig. 2. An N-digit binary digit-parallel online adder with $(N+1)$-digit outputs. Both inputs and outputs are represented using SD representation.

Algorithm 1 Online Multiplication

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

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 with MSD-first 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 $\{\overline{1}, 0, 1\}$. In this case $sel()$ is given by (2) [13]. Notice that the election is made only based on 1 integer bit and 1 fractional bit of $W_{[j]}$.

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

It is worth noting that the precision of the product Z can be determined by directly truncating from the LSDs, as 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 full-precision result is generated. This feature of online arithmetic has inspired research about dynamically control of computation precision in numerous applications.

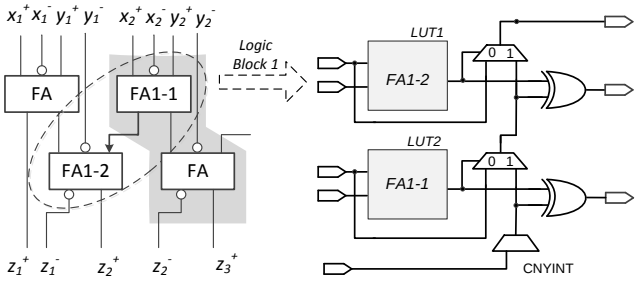


Fig. 3. Map the online adder onto Spartan FPGAs using the fast-carry resources. The grey background highlights the 4:2 compressor. Dotted circle indicates the logic block (LB) which can be mapped to the FPGA carry resources.

In Section IV, we will first describe an optimized online multiplication algorithm and its FPGA implementation, which are designed specifically targeting on digit parallel operations. Then we provide a modified structure which only generates the most significant half the products, 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 [14], [15];
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,

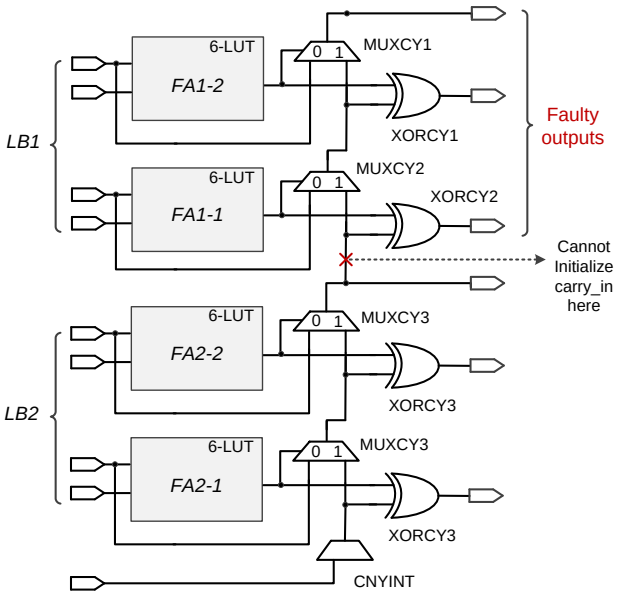


Fig. 4. An example illustrating that direct applying approaches in [14] on a Virtex-6 FPGA will results in faulty outputs, because the carry input of `LB1` cannot be explicitly initialized.

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 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 (3) and (4), respectively.

$$OAarea\_LUT = 2N \tag{3}$$

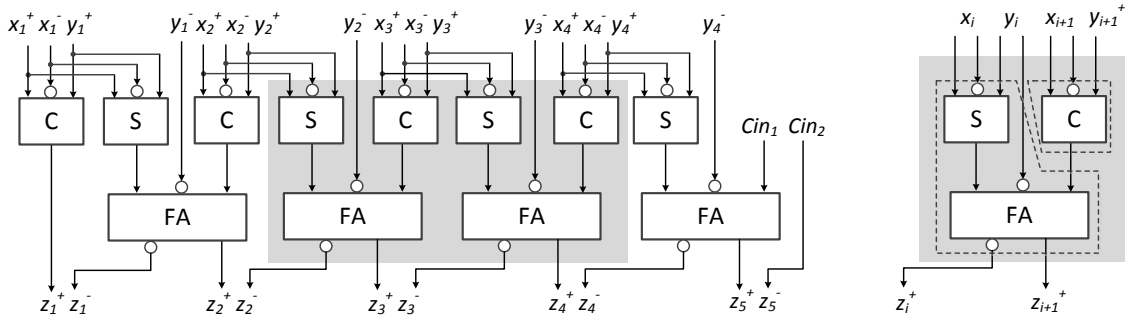


Fig. 5. Modified structure of online adder. Left: an example of 4-digit online adder. The shaded part refers to the 2 logic blocks (LBs) that can be mapped onto 1 slice. Right: one LB. the dotted box outlines the logic that can be mapped onto 1 LUT and the corresponding fast-carry logic.

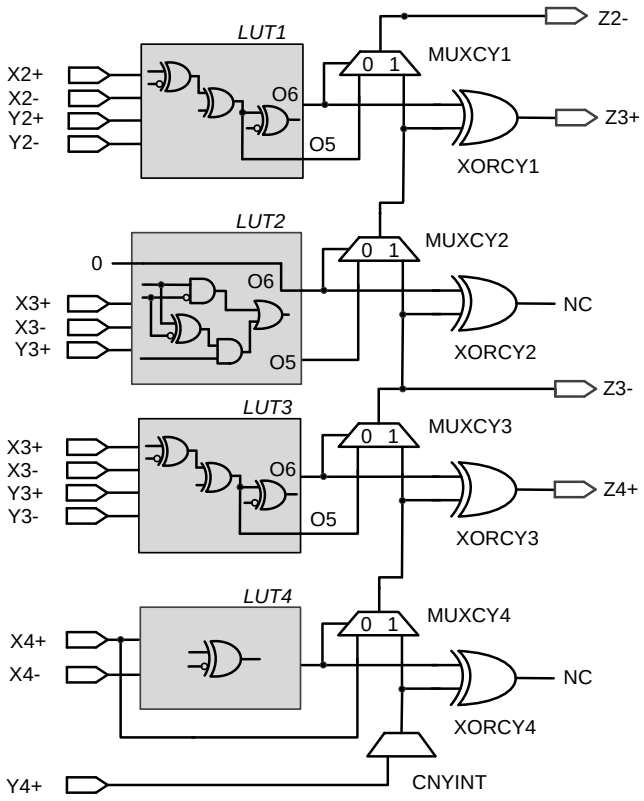


Fig. 6. Implementation of 2 logic blocks (LBs) in 1 FPGA slice which contains four 6-LUTs. NC stands for "Not Care".

$$OAarea\_Slice = 1 + \left\lceil \frac{N-2}{2} \right\rceil \tag{4}$$

C. Performance Analysis

In our experiments, we compare the proposed online adder (`OA_new`) against the original design which is implemented using behavioural descriptions based on 4:2 compressors (`OA_behv`). 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`.

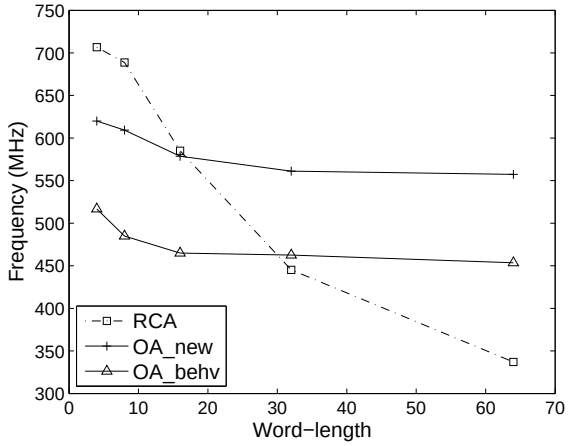
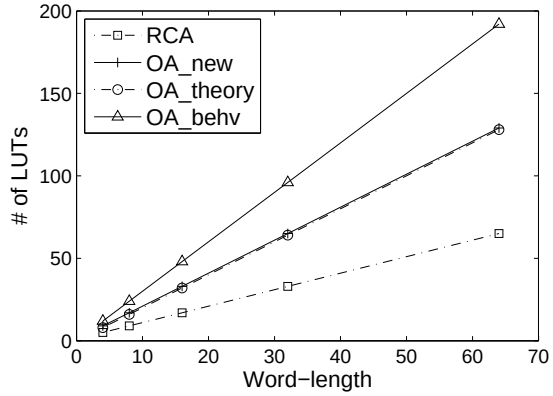


Fig. 7. Rated frequencies of the RCA and the online adder with different implementation methods. The results are obtained from post place and route timing reports in ISE 14.1.

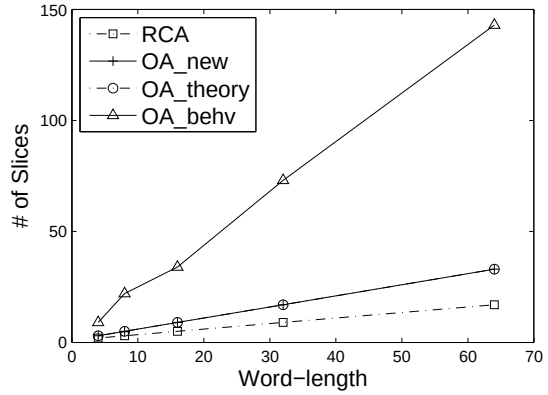
In addition, we also include ripple carry adder (`RCA`) which uses conventional arithmetic, into the comparisons. All designs are targeting on Xilinx Virtex-6 FPGA with speed grade -1.

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 ideally a fixed value and it is irrelevant to the operand precisions. In comparison, the rated frequency of `RCA` drops drastically for larger designs. In this case, both online adders will eventually outperform `RCA`. However our approach operates constantly faster than `OA_behv` for any operand word-length. The speed-ups of `OA_new` over `OA_behv` are from 20% to 26%.

We compare the resource usages of all three adder designs with respect to a variety of operand word-lengths. The results are shown in Fig.8. The theoretical area budgets of the online adder predicted by (3) and (4) are also plotted with dotted lines in Fig.8. Several observations can be made from the results.



(a)



(b)

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.

Firstly the predicted resource usages of both LUTs and Slices match well the results obtained from real implementations. Secondly it can be seen that the behavioural implementation `OA_behv` is not area efficient for FPGAs, because the dedicated FPGA resources are not fully utilized. In comparison, our implementation `OA_new` achieves significant area savings: $25\% \sim 33\%$ in LUTs and $67\% \sim 77\%$ in Slices. Thirdly, our design still costs more area than `RCA`, with the area overhead of $80\% \sim 98\%$ in terms of LUTs, and $0\% \sim 88\%$ in terms of Slices. This is because for a single adder, the area overhead brought by the redundant number representations cannot be removed. However research has shown that this overhead can be reduced for multiple-operand adders or adder trees [xxx].

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 is more tolerant to timing violations in comparison to conventional arithmetic, because timing errors initially affect the least significant digits and this will lead to small error magnitude[xxx]. This implementation method is straightforward and can be utilized to design other digit parallel online operators.

Algorithm 2 Digit Parallel Online Multiplication

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

On the other hand, significant area budget is required in order to implement this type of digit parallel online multiplier (`OM`). Typically 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`, totally $(2N+\delta)$ iterations are required to generate $2N$ digits outputs, and $(N+\delta)$ iterations for the most significant N digits result. Second, in each stage the logic is complex as it involves operations such as online addition, digit-vector multiplication, shifting and other combination logic such as the selection function. Besides, the word-length of signals $H_{[j]}$, $W_{[j]}$ and $P_{[j]}$ is $N+2+\delta$, because the selection function needs 2 integer digits as well as 1 fraction digits to generate z_j, as described in Section II. Whereas in 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 this algorithm because the input data are 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 given 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 the original online multiplication algorithm, 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 connection without using other logic resources.

B. FPGA Implementation

We also explore an area efficient implementation of Algorithm 2 on FPGAs. The general structure diagram of a 4-digit `OM` using the proposed algorithm is shown in Fig. 9(a), and

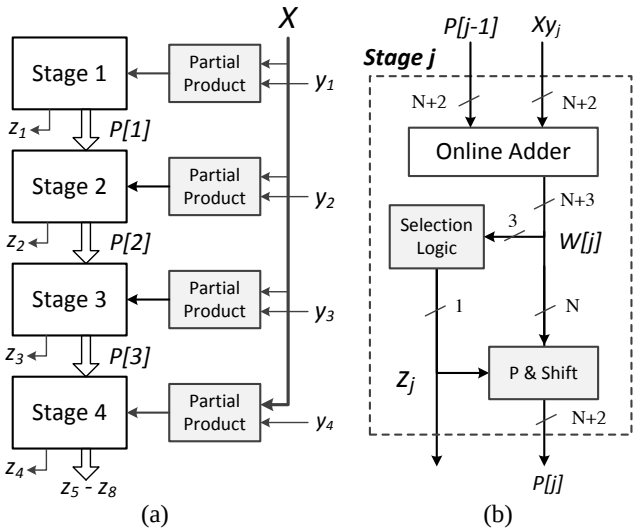


Fig. 9. (a) Structural diagram of a 4-digit online multiplier using the proposed algorithm. (b) Structure of one stage. The word-length of all signals are labelled in terms of the number of digits. N denotes the word-length of the input signals.

the structure diagram of 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 digits input (6 bits) and generate 1 digit output (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 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 with 6-LUT for further area reduction. In the online adder structure as presented in Fig. 6, only 4 inputs per LUT are used to the maximum. This leaves 2 available inputs per LUT. 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 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 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 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 LUT and Slice usages of the first stage, which is purely built with combinational logic without online adders.

$$OMarea\_LUT = 2(N+2)(N-1) + L_{S1} \tag{5}$$

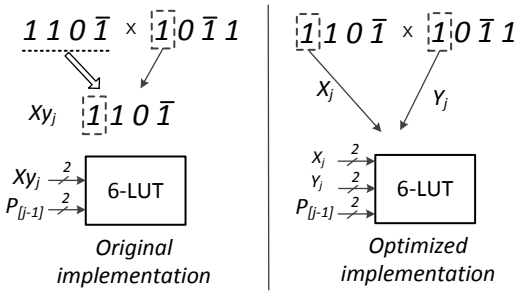


Fig. 10. Left: Direct implementation with extra logic to generate partial products. Right: Combining the logic blocks that generate partial products into the Online Adder by fully utilizing all the inputs of the 6-LUTs. In this case, further area reduction can be achieved.

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

C. Structure Optimization for the MSD Half of the Results

As discussed before in Section II, normally the multiplier is connected with other arithmetic operators in real applications. If the outputs of a multiplier 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 of the system. 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 flexibility to simplify the structure correspondingly without affecting accuracy, if only the MSD half of the results are required. This is possible because in the `OM`, the product digits are generated initially from the MSD, and there is no carry propagation from the LSD to the MSD as described previously. 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 which is implemented to derive full precision results using 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 incorporate multipliers with conventional binary arithmetic, and it is created using the Xilinx Core Generator. The `CoreGen` multiplier is implemented based on LUTs without DSPs, and it is configured with speed optimization [20].

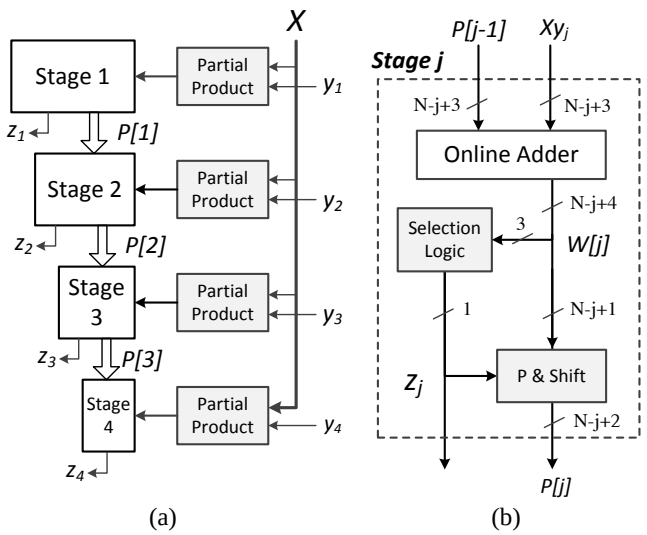


Fig. 11. (a) Modified structure of a 4-digit online multiplier which only generates the most significant 4-digit result. (b) Structure diagram of stage j, with the word-lengths of all internal signals labelled.

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. The area predictions for `OM_full` from (5) and (6) are also included in both figures. Again we can see that the predicted resource usages match well with practical implementations. In comparison to `OM_initial`, clearly significant area savings can be obtained by the optimized design `OM_full`. Specifically, the LUTs usage and Slices usage 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 $48\% \sim 84\%$ 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 against the `CoreGen` multiplier. The area saving varies from 40% to 55% for LUTs, and $46\% \sim 55\%$ 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 Xilinx Timing Analyzer. The results are plotted in Fig.xxx, 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, 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 similar.

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

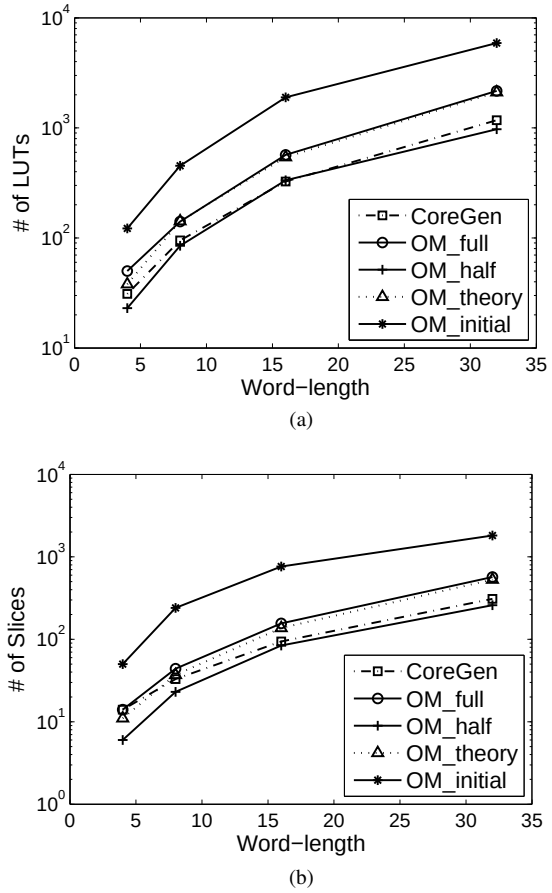


Fig. 12. Area comparisons of different types of binary multipliers. (a) Usage of LUTs. (b) Usage of Slices

happen. In this case, we can 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.

$$MRE = \frac{E_{error}}{E_{out}} \times 100\% \tag{7}$$

For instance, Fig. 13 demonstrates the MRE of the 8-digit multipliers. Notice that the input data to the circuits are randomly sampled from a uniform distribution of N-digit numbers. We see that although the Timing Analyzer predicts slower frequencies for `OM`, it actually operates faster than `CoreGen` without the occurrence of timing errors. Besides,

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.

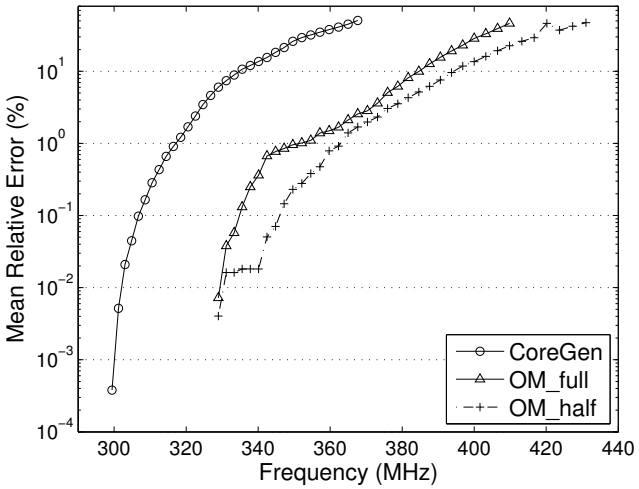


Fig. 13. (a) Modified structure of a 4-digit online multiplier which only generates the most significant 4-digit result. (b) Structure diagram of stage j, with the word-lengths of all internal signals labelled.

[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.