Efficient FPGA Implementation of Digit Parallel Online Arithmetic Operators

***Abstract*—Online arithmetic has been widely studied for ASIC implementation. Online components were originally designed to perform computations in digit serial with most significant digit first, resulting in the ability to chain arithmetic operators together for low latency. More recently, research has shown that digit parallel online operators can fail more gracefully when operating beyond the deterministic clocking region in comparison to operators with conventional arithmetic. Unfortunately, the utilization of online arithmetic operators in the past has required a large area overhead for FPGA implementation. In this paper, we propose novel approaches to implement the key primitives of online arithmetic, adders and multipliers, efficiently on modern Xilinx FPGAs with 6-input LUTs and carry resources. We demonstrate experimentally that in comparison to a direct RTL synthesis, the proposed architectures achieve slice savings of over 67% and 69%, and speed-ups of over $1.2\times$ and $1.5\times$ for adders and multipliers, respectively. Furthermore in comparison to standard arithmetic primitives, this translates to reduction of the area overheads of up to $8.41\times$ and $8.11\times$ for adders and multipliers using direct implementation, to $1.88\times$ and $0.54\times$ using our approaches.**

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 potentially results in large latency when propagating data between 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: that it allows graceful degradation when operating beyond the deterministic clocking region, *i.e.* when overclocking [3]. This theme 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 tend to 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 use of online arithmetic on existing FPGAs is still limited due to a large area overhead. For standard arithmetic, because it is inefficient and slow to map arithmetic circuits solely onto look-up-tables (LUTs), both main commercial FPGA vendors have introduced dedicated carry logic [4]. Significant prior work explores methods that can effectively map circuits to the carry logic for performance improvements [5], or proposes alternative carry logic for FPGAs [6], [7]. Hard DSP blocks are also included in modern architectures to improve the area and performance of multiplications and multiply accumulates [8]. However, most of these approaches are designed to accelerate computations with the conventional form of arithmetic.

For online arithmetic, two major research directions have been investigated. Firstly, there has been some research into FPGA implementations of online adders; a brief review of relevant work in this area is presented in Section III. Secondly, there are existing approaches to build general purpose online adders specifically for FPGAs with 4-input LUTs and with two LUTs within a slice or a logic element (LE), such as the Xilinx Spartan 3 series [9] and the Altera Cyclone III [10]. However, as will be discussed in Section III, these approaches cannot be directly applied on many modern FPGAs, which typically have 6-input LUTs and four LUTs in a slice, such as the Xilinx Virtex series FPGAs and all Xilinx 7 series FPGAs [11]. Moreover, the move to the 6-LUT on the basic architecture opens new opportunities for further optimization.

In this paper, both issues will be addressed. In Section III, we propose a novel approach to map digit parallel online adders onto FPGAs with 6-LUTs. The mapping is based on the fast carry logic and ensures the available logic resources within a slice are fully utilized. In Section IV, we optimize the online multiplication algorithm to yield an efficient digit parallel online multiplier for FPGAs. We demonstrate experimentally on Xilinx Virtex-6 FPGAs that the proposed designs achieve significant area reduction and performance gain over their original implementations. For online adders, our method achieves slice savings of over 67% and frequency speed-ups of over $1.2\times$ in comparison to a direct implementation. In comparison to the standard ripple-carry adder, the area overhead drops from up to $8.41\times$ to $1.88\times$ using the proposed online adder. For online multipliers, the slice utilization of the proposed architecture drops of over 69%, with a frequency speed-up over $1.5\times$ in comparison to a direct implementation. When compared to the multiplier with standard arithmetic, the direct implementation has an area overhead up to $8.11\times$, which drops to $1.84\times$ when using the proposed new architecture.

Our approach has the additional advantage of being able to create a correctly rounded multiplier with a reduced output precision using less hardware than a standard multiplier, which multiplies two N-digit numbers and produces an output of $2N$ digits. However, in many cases the full precision product is not required. In such cases, a reduced precision output is created either by truncating or rounding the LSDs. Although there has been research into saving area by avoiding the creation of unnecessary LSDs [12], these techniques still require extensive

simulation to guarantee correctness. In contract, since online multipliers compute the output from the MSD first, hardware can be saved by not generating the LSDs in the first place. As an example, if we are only interested in the most significant half of the product, we can create a digit parallel online multiplier that saves upto 56% slice usage in comparison to a standard multiplier with truncated output.

In summary, the main contributions of this paper are:

1) Efficient methods to map online adders and multipliers to modern FPGAs;
2) Discussion of how to implement correctly-rounded online multipliers for a chosen output precision;
3) Demonstration of the performance improvements of online operators in terms of silicon area and operating frequency.

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 operation, as illustrated in Fig 1. It can be seen that in order to generate the first output digit, δ digits of inputs are required, where δ is called the "online delay". δ is normally a small constant, which is independent of the precision. For ease of discussion, for the rest of this paper, the input data is assumed to be 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].

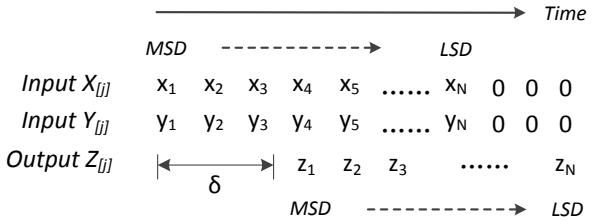


Fig. 1. Dataflow in digit-serial online arithmetic, in which both inputs and outputs are processed from the MSD to the LSD. δ denotes the online delay.

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

MSD-first operation is possible only if a redundant number system is used. Normally there are two most commonly used redundant number representations: carry-save (CS) [15] and signed-digit (SD) [16]. With SD representation, each digit is represented using 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 $\{0, \cdots, r-1\}$. Thus a standard number corresponds to several possible redundant representations. For example, the binary number 0.011 can be represented in SD form as $0.1\overline{1}1$, $0.10\overline{1}$ or 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 has multiple representations.

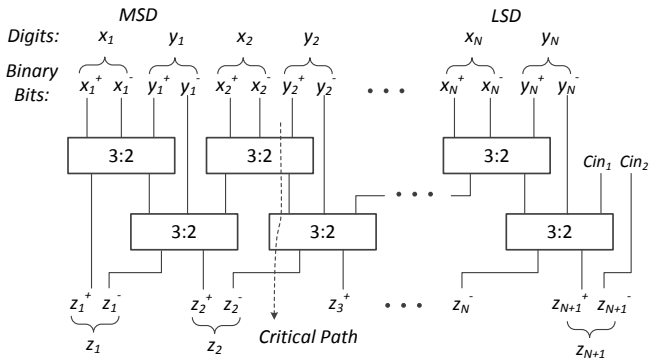


Fig. 2. An N-digit binary digit-parallel online adder. Both inputs and outputs are represented using SD representation. "3:2" denotes a 3:2 compressor.

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 of an online adder where all signals represented with SD numbers of digit set $\{-1, 0, 1\}$ is shown in Fig. 2. The module "3:2" denotes a 3:2 compressor, which takes three inputs and generates two outputs, and is logically equivalent to 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 two 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].

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 $sel()$. For any radix r and chosen digit set, there exits an appropriate selection method and a value of δ which ensure convergence. As the binary radix 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), and the selection is based on two integer digits and one fractional digit of $W_{[j]}$ [18].

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

It is worth noting that the precision of the product Z can be determined by directly truncating from the LSDs, since

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**

there is no carry propagation from the LSD to the MSD. This is useful especially when the multiplication product is used by successive computations, where often only the first N digits of the results are 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 [19].

In Section IV, we will first describe an optimized online multiplication algorithm and its FPGA implementation, which are designed specifically targeting 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 for FPGAs

A. Related Works

There have 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 [20], [21];
2) multiple operand addition by designing compressor trees based on bit counters [22], [23];
3) modifying existing FPGA architecture for more efficient digit-parallel online addition [24] and for specific applications [25].

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 modify 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 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 two FAs within a 4:2 compressor, and the net delay between them can be large. Instead, Kamp et al [20] and Hormigo et al [21] described very similar mapping techniques for online adders with SD representations and CS representations, respectively. They share the common idea of building the online adder based on the logic block (LB) as highlighted

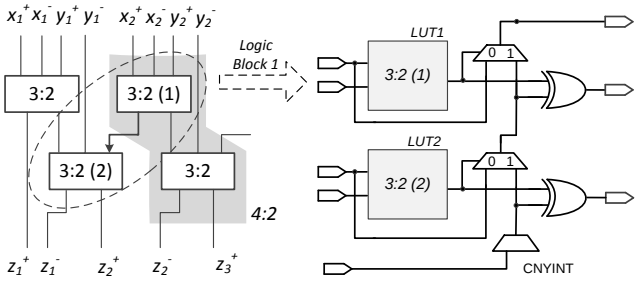


Fig. 3. Map the online adder onto Spartan FPGAs using the fast-carry resources. The gray background highlights the 4:2 compressor. Dotted circle indicates the logic block (LB) which can be mapped to the FPGA using the carry resources.

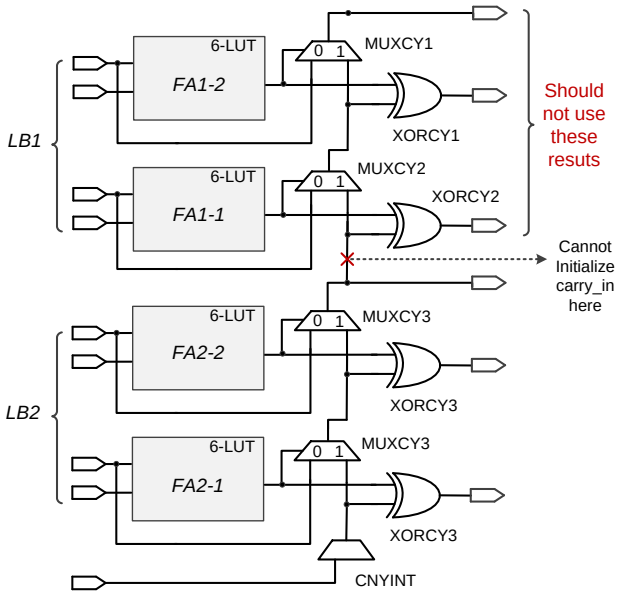


Fig. 4. An example illustrating that direct applying approaches in [20] on a Virtex-6 FPGA will results in faulty outputs, because the carry input of LB1 cannot be explicitly initialized.

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 two 3:2 compressors can be greatly reduced.

However, we notice the major limitations of both approaches that they only target FPGAs with 4-input LUTs (4-LUT) and two 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 four 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 wrong results. For instance, if two LBs are mapped to a slice with four LUTs as seen in Fig. 4, the outputs of LB1 should not be used because the its carry input cannot be explicitly initialized. Due to the same reason, the output of XORCY2 should not be used.

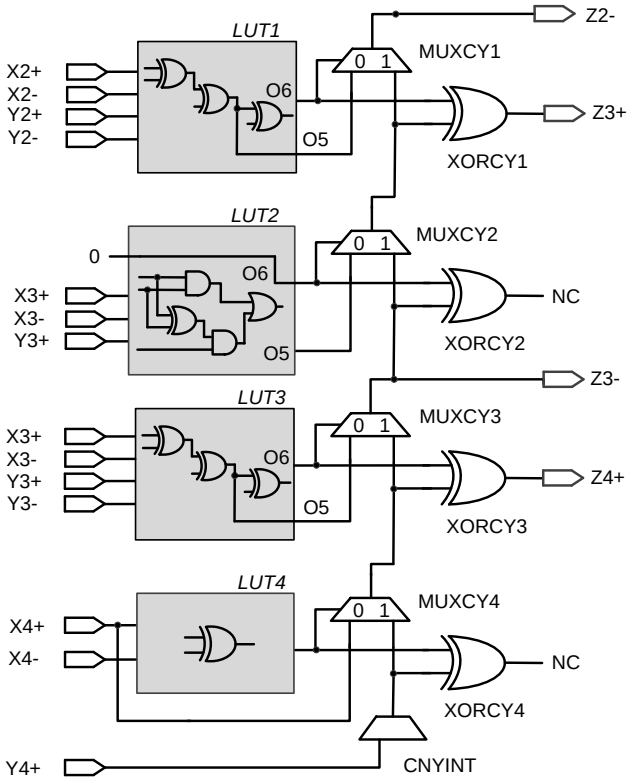


Fig. 6. Implementation of two logic blocks (LBs) in one FPGA slice which contains four 6-LUTs. NC stands for "Not Care".

B. Proposed Mapping Method

To tackle this problem, we first present an alternative 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 two parts, which only generate carry and sum respectively. In this case they can be mapped individually on two 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 two `LB`s and the logic in each LUT are shown in Fig. 6. The I/O signals are identical to the previous example in Fig. 5. The 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`. The logic to generate the MSD and the LSD can be combined into one slice as observed from Fig. 5. Therefore using this mapping method the resources within one 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 to the operand word-lengths (N) can be calculated as given in (3) and (4), respectively.

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

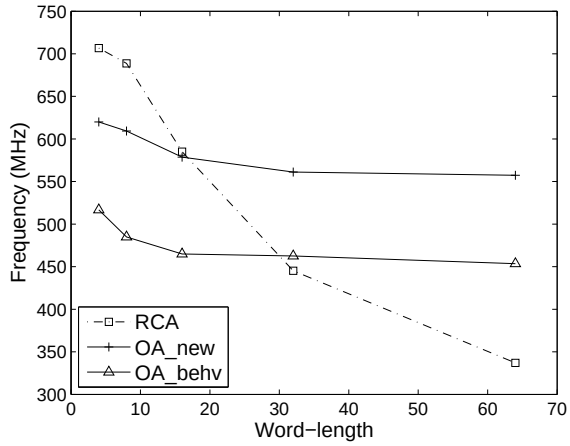


Fig. 7. Rated frequencies of the RCA and the online adder with different implementation methods. The results are obtained from post place and route timing reports in ISE 14.7.

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

C. Performance Analysis

In our experiments, we compare the proposed online adder (`OA_new`) against the original design which is implemented using functional descriptions in Verilog based on 4:2 compressors (`OA_behv`). We also compare these architectures against a ripple carry adder (`RCA`), which uses conventional arithmetic. While our experiments are targeting on Xilinx Virtex-6 FPGAs with speed grade -1, our methods are applicable to any FPGAs with the same slice architecture such as all the Xilinx 7 series FPGAs. Notice that in order to utilize the carry resources and 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 14.7. It can be seen that both designs with online arithmetic achieve relatively stable operating frequencies across a variety of operand word-lengths. This is as expected, because the critical path delay of the online adder is theoretically independent of the operand precision. In comparison, the rated frequency of the `RCA` drops dramatically for larger word-lengths. It follows that both online adders outperform `RCA` for large word-lengths. More specifically, the frequency of `OA_behv` is faster than `RCA` when $N \geqslant 30$, whereas the proposed online adder outperforms `RCA` for $N > 16$. As a result, this makes the `OA_new` more attractive for a wider range of applications. Furthermore, the speed-ups of `OA_new` against `OA_behv` are over $1.2\times$.

We also compare the resource usage of all three adder designs with respect to a variety of operand word-lengths. The 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: $25\% \sim 33\%$ in LUTs and $67\% \sim 77\%$ in slices. This is because the dedicated FPGA resources are fully utilized. These area savings reduce the overhead of an `OA` over an `RCA` from upto $2.95\times$ to $1.98\times$ for LUTs, and

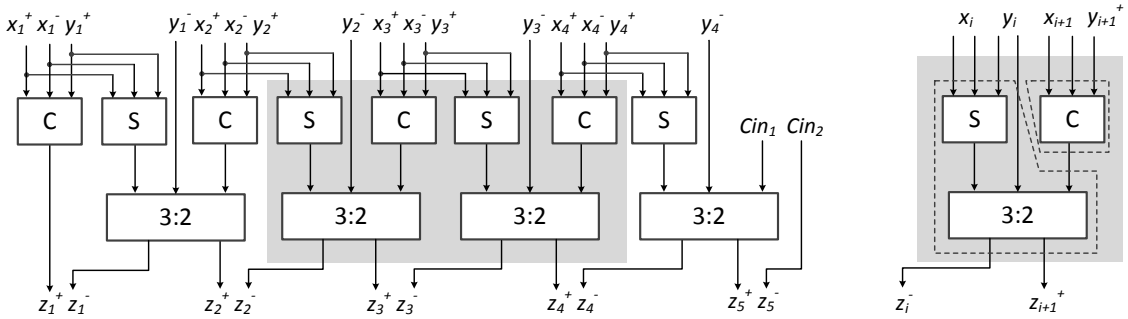
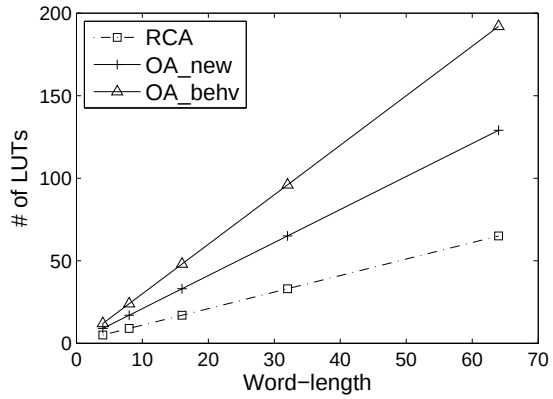
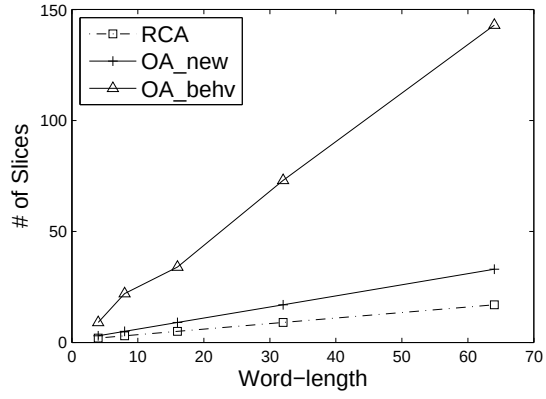


Fig. 5. Alternative structure of online adder. Left: an example of 4-digit online adder. The shaded part refers to the two logic blocks (LBs) that can be mapped onto one slice. Right: one LB. The dotted boxes outline the logic that can be mapped onto each LUT and the corresponding fast carry logic.



(a) Usage of LUTs



(b) Usage of slices

Fig. 8. Area comparisons of different binary adder implementations with respect to a variety of operand word-lengths.

from $8.41\times$ to $1.88\times$ for slices. It is unlikely that the overhead can be completely avoided when using an OA because of the redundant number system. Nevertheless, with the reduced overhead, using OA to take advantage of the graceful degradation when operating beyond the deterministic clocking region may become more attractive for an FPGA developer.

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. A 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].

However, significant area overhead is required in order to implement this type of OM, 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 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 (to generate $x_{j+\delta+1} \cdot Y_{[j+1]}$ and $y_{j+\delta+1} \cdot X_{[j]}$ in Algorithm 1), shifting and other combinational 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 two integer digits and one fractional digit to generate z_j, as described in Section II. Third, this OM architecture is not optimized specifically for FPGA technology.

We address the first two design issues by providing an alternative online multiplication algorithm. Instead of simply implementing the unrolled version of Algorithm 1, it can be optimized specifically for digit parallel operations. 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. However if all digits of the inputs are available simultaneously, then the delay δ is no longer necessary. For 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 number of 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 logic resources.

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: $Z_{[N+1:2N]} \leftarrow frac(W_{[N]})$

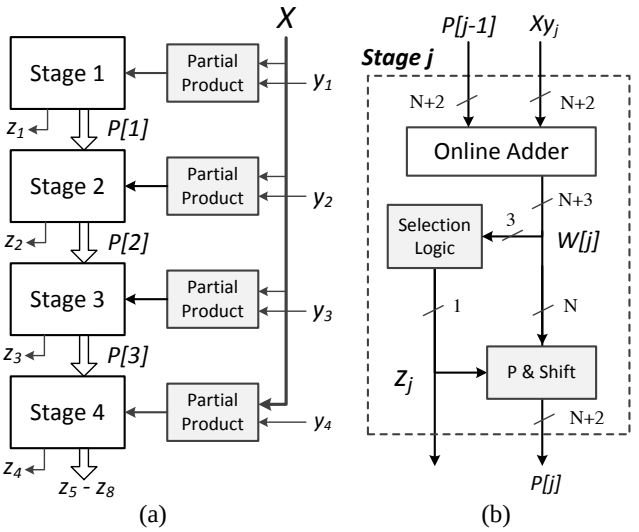


Fig. 9. (a) Structure of a 4-digit online multiplier using the proposed algorithm. (b) Structure of one stage. The word-length of all signals are labelled in terms of the number of digits. N denotes the word-length of the input signals.

B. FPGA Implementation

We now present an area efficient implementation of Algorithm 2 on FPGAs. The general structure of a 4-digit OM using the proposed algorithm is shown in Fig. 9(a), and the structure 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 generates 1 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 four inputs per LUT are used. This leaves at least two available inputs per LUT. Originally in order to generate one digit partial product, one digit of inputs X and Y are required respectively, as indicated in Fig. 10. Alternatively, instead of using extra logic to generate Xy_j, it can be merged into the

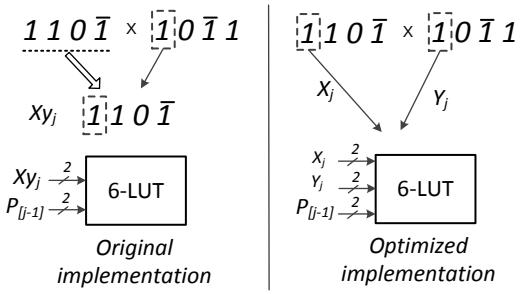


Fig. 10. Left: Direct implementation with extra logic to generate partial products. Right: Combining the logic blocks that generate partial products into the Online Adder by fully utilizing all the inputs of the 6-LUTs. In this case, further area reduction can be achieved.

corresponding LUTs in the online adder by fully utilizing all six LUT inputs, as shown on the right of Fig.10. Notice that this approach is only available for FPGAs with 6-LUTs.

The resource usage of the proposed OM can be calculated as follows. In an N-digit OM, in total $(N-1)$ online adders are needed. According to Algorithm 2, the word-length of each online adder is $(N+2)$-digit (two 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.

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

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

C. Structure Optimization for Half Precision 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 a consistent word-length is used throughout the system, then only the most significant half of the product is 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 OM 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 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 the stages (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.

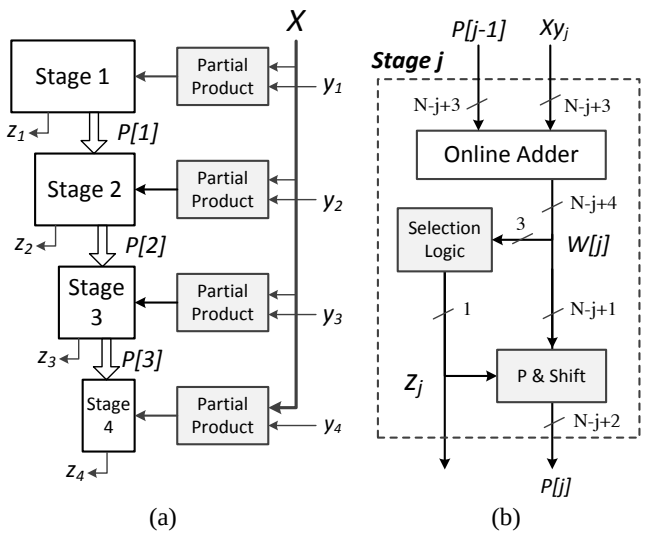


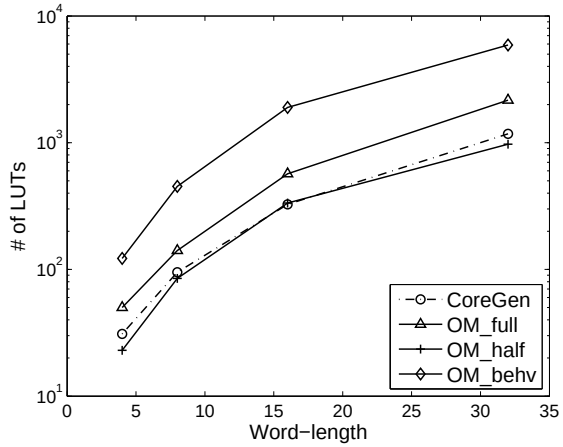
Fig. 11. (a) Modified structure of a 4-digit online multiplier which only generates the most significant 4-digit result. (b) Structure diagram of stage j, with the word-lengths of all internal signals labelled.

D. Performance Analysis

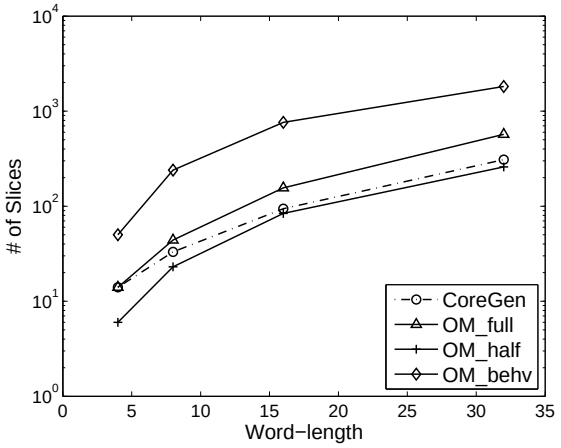
Similar to the experiments as described in Section III-C, we compare the direct RTL online multiplier (`OM_behv`) against the proposed two types of multiplier, which are implemented to generate products with full precision (`OM_full`) and half precision (`OM_half`), respectively. In addition, we also compare the `OM` with the conventional binary arithmetic multiplier. The conventional multiplier is created using the Xilinx Core Generator with speed optimization [26], and it is implemented based on LUTs without DSPs for a fairer comparison.

The comparison of area in terms of the cost of LUTs and slices with respect to different operand word-lengths are shown in Fig. 12. The results are obtained from ISE 14.7 after mapping the design to the Virtex-6 FPGA. In comparison to `OM_behv`, significant area savings have been obtained by the optimized design `OM_full`. Specifically, the LUTs and slice utilization reduces $59.0\% \sim 70.1\%$ and $68.6\% \sim 81.7\%$, respectively. In comparison to the `CoreGen` multiplier, the area overhead of online multipliers drops significantly using the proposed new structure. In order to generate full precision product, the area overhead of `OM_behv` against `CoreGen` are $3.94\times \sim 5.82\times$ for LUTs, and $3.58\times \sim 8.11\times$ for slices. Using the proposed structure `OM_full` the area overheads drop to $1.48\times \sim 1.83\times$ for LUTs and up to $1.84\times$ for slices. Furthermore as mentioned in the introduction, a traditional multiplier is unable to compute this correctly rounded output using less silicon area without extensive simulation. In contrast, using our approach we can trade area for precision and tune output precision to be the minimum desired. As an example, if only half of the output precision is required, `OM_half` can save silicon area over the traditional `CoreGen` multiplier. The area saving varies from 40% to 55% for LUTs, and $46\% \sim 56\%$ for slices.

We also check the rated frequencies of all multipliers for a variety of operand word-lengths. The frequency values are plotted in Fig. 13, which are still obtained from the post place and route timing reports. In Fig. 13 we see that in comparison to `OM_behv`, our proposed architectures achieve frequency speed-ups varying from $1.49\times$ to $1.98\times$.



(a) Usage of LUTs



(b) Usage of slices

Fig. 12. Area comparisons of different types of binary multipliers.

We comment that there remain a large gap in terms of frequency between the proposed online multipliers and the `CoreGen` multiplier, especially for large operand word-lengths. This is because in the online multiplier, even with the carry resources in the FPGA being used efficiently, 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 hyperbole for large operand word-lengths. This also explains why `OM_full` and `OM_half` run at similar frequencies, because 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

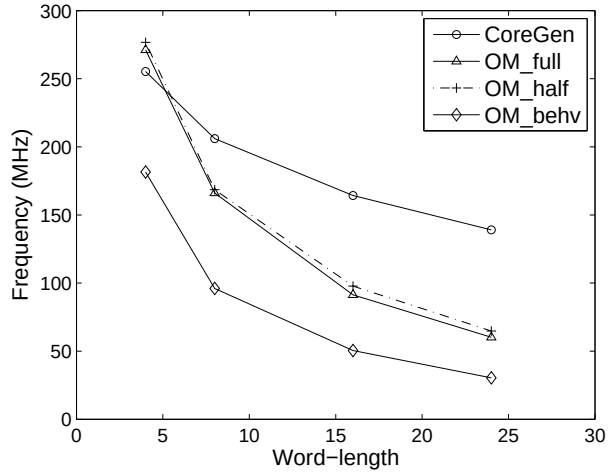


Fig. 13. Rated frequencies of different types of binary multipliers for a variety of operand word-lengths. The results are obtained from the Xilinx Timing Analyzer in ISE 14.7 after placing and routing.

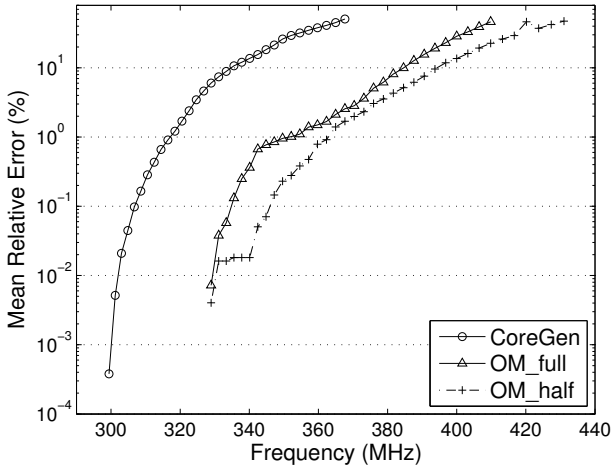


Fig. 14. Mean relative errors seen at the outputs of different types of 8-digit binary multipliers, when clocked with faster-than-rated frequencies.

of error and the mean value of correct output, respectively.

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

For instance, Fig. 14 demonstrates the MRE values of the 8-digit multipliers. The results are from post place and route simulations, and the input stimulus are randomly sampled from a uniform distribution of 8-digit numbers. 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].

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 four LUTs in a slice, and fully utilizes the built-in carry resources. We have demonstrated 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] Removed for blind review.

[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. FPL*, Aug 2010, pp. 318–325.

[6] S. Hauck, M. Hosler, and T. Fry, "High-performance carry chains for FPGA's," *IEEE Trans. VLSI*, 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. FPL*, Aug 2006, pp. 1–6.

[8] Xilinx Inc., "Virtex-6 FPGA DSP48E1 slice user guide," Feb. 2011.

[9] ——, "Spartan-3 generation FPGA user guide," Feb. 2008.

[10] Altera Co., "Cyclone device handbook," 2008.

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

[12] T. Drane, T. Rose, and G. A. Constantinides, "On the systematic creation of faithfully rounded truncated multipliers and arrays," *IEEE Trans. on Computers*, vol. 99, 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 Jourl. 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] S. Rajagopal and J. Cavallaro, "Truncated online arithmetic with applications to communication systems," *IEEE Trans. on Computers*, vol. 55, no. 10, pp. 1240–12 529, Oct 2006.

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

[21] 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. ASAP*, 2009, pp. 207–210.

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

[23] H. Parandeh-Afshar, A. Verma, P. Brisk, and P. Ienne, "Improving FPGA performance for carry-save arithmetic," *IEEE Trans. VLSI*, vol. 18, no. 4, pp. 578–590, April 2010.

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

[25] R. Gutierrez, J. Valls, and A. Perez-Pascual, "FPGA-implementation of time-multiplexed multiple constant multiplication based on carry-save arithmetic," in *Proc. FPL*, Aug 2009, pp. 609–612.

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