

# Datapath Synthesis for Overclocking: Online Arithmetic for Latency-Accuracy Trade-offs

Authors

## ABSTRACT

In this paper we take a fresh look at Online Arithmetic, originally proposed for digit-serial operation, and synthesize unrolled digit-parallel online operators to allow for graceful degradation under overclocking. We quantify the impact of timing violation on key arithmetic primitives, and show that substantial performance benefits can be obtained in comparison to binary arithmetic. Since timing errors are generated in least significant digits with online arithmetic, less impact is caused than conventional implementations. Using analytical models and empirical FPGA results, we demonstrate an error reduction over 89% and an improvement in SNR of over 20dB for the same clock rate.

## Categories and Subject Descriptors

B.2.4 [Arithmetic and Logic Structures]: High-Speed Arithmetic—*Algorithms, Cost/performance*

## General Terms

Design, Performance, Reliability

## Keywords

Online Arithmetic, Overclocking, Imprecise Design

## 1. INTRODUCTION

Circuit performance has increased tremendously over the past decades with the continuous scaling of CMOS technology such that they occupy less area and operate faster. Typically static-timing analysis incorporates safety margins for timing closure. However, continuing with this approach could become very expensive in the short term future. As circuit dimensions scale down, it is increasingly difficult to maintain low relative manufacturing-variation. Because timing margins in static-analysis tools increase with process variability, designing for worst-case delay in circuits becomes increasingly costly as relative process variation increases.

While datapath can be pipelined to increase clock frequency, this does not reduce overall computational latency. In em-

bedded applications, which often have strict latency requirements, or in any datapath containing feedback, where C-slow retiming is inappropriate, pipelining does not help meet performance targets. To overcome these problems, a large volume of studies have demonstrated that relaxing the absolute datapath-accuracy requirements can provide the freedom to create designs with better performance or energy efficiency [1, 7, 9, 10].

However, we notice that most research in this area focuses on standard binary arithmetic [7, 9, 10], which potentially suffers from large magnitude of timing errors. This is because, in standard arithmetic operators such as ripple-carry adders, the most significant digit (MSD) is updated *last* due to carry propagation, so timing violation initially affects the MSD of the result. In this paper, for the first time we attempt to solve this problem by adopting an alternative form of MSD-first arithmetic known as online arithmetic. We evaluate the probabilistic behavior of basic arithmetic primitives with different types of arithmetic when operating beyond the deterministic clocking region. We suggest that in comparison to conventional arithmetic, operators employing this arithmetic are less sensitive to overclocking, because timing errors only affect the least significant digits. To support this hypothesis, we propose probabilistic models of error for an online multiplier. We back up the models with experimental results on an image processing application. We demonstrate that our novel design methodology can lead to substantial performance benefits compared to the design method using conventional arithmetic. A summary of the main contributions of this paper are as follows:

- to our knowledge, the first “overclocking friendly” computer arithmetic,
- probabilistic models of overclocking errors for a digit-parallel online multiplier,
- analytical and empirical results which demonstrate that the impact of timing violations is ameliorated with online arithmetic and that large performance benefits can be achieved.

## 2. BACKGROUND

### 2.1 Online Arithmetic

In conventional arithmetic, the computation results are generated either from the least significant digit (LSD), e.g. addition and multiplication, or the MSD, e.g. division and square root. This inconsistency in computing directions can

result in large latency when propagating data among different operations. To tackle this problem, online arithmetic has been proposed and studied over the past few decades [4, 13, 11]. Both the inputs and outputs are processed in a MSD-first manner using online arithmetic, enabling parallelism among multiple operations. Therefore the overall latency can be greatly reduced.

Originally online arithmetic was designed for digit-serial operation. In order to generate the first output digit,  $\delta$  digits of inputs are required, as illustrated in Figure 1.  $\delta$  is known as the online delay, which is a constant, independent of the precision in a given operation. For ease of discussion, the input to our circuit is normalized to a fixed point number in the range  $(-1, 1)$ . Based on this premise, the online representations of  $N$ -digit operands and result at iteration  $j$  are given by (1) for  $j \in [-\delta, N-1]$  and  $r$  denotes the radix [5].

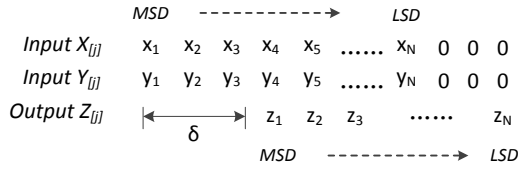


Figure 1: Dataflow in digit-serial Online Arithmetic.

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

MSD-first operation in online arithmetic is possible because they adopt a redundant number system [2], in which each digit is represented with a redundant digit set  $\{-a, \dots, -1, 0, 1, \dots, a\}$ , where  $a \in [r/2, r-1]$ . Due to the redundancy, the MSDs of the result can be estimated only based on partial information of the inputs and then the value of the number can be revised by the following digits.

## 2.2 Radix-2 Digit-parallel Online Adder

Adders serve as a critical building block for arithmetic operations. To perform digit-parallel online addition, a redundant adder can be directly utilized. The adder structure diagram is shown in Figure 2. A major advantage with the redundant number system over the standard ripple-carry based arithmetic is that the propagation of carry is eliminated, resulting in a precision-independent computation time for addition. As seen in Figure 2, 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 online adder suitable for building up more complex arithmetic operators such as multipliers to accelerate the sum of partial products [8, 12].

## 2.3 Radix-2 Digit-parallel Online Multiplier

Multiplication is another key primitive of arithmetic operations. Typically the online multiplication is performed in a recursive digit-serial manner, as illustrated in Algorithm 1 where both inputs and outputs are  $N$ -digit number as represented in (1). For a given iteration  $j$ , the product digit  $z_j$  is generated through a selection function  $sel()$ . For the radix  $r$  and a chosen digit set, there exists an appropriate selection method and a value of  $\delta$  which ensure convergence. As radix-2 is used most commonly in computer arithmetic,

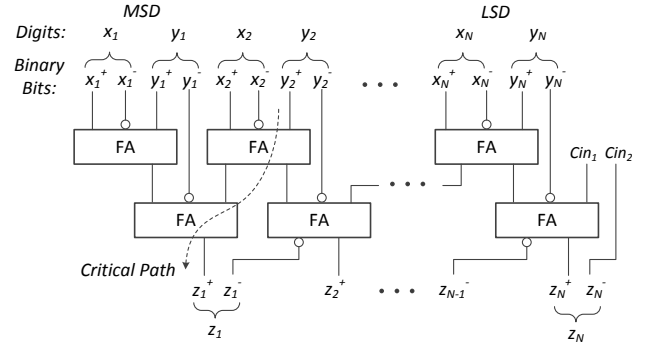


Figure 2: An  $N$ -digit radix-2 unrolled online adder.

### Algorithm 1 Online Multiplication

**Initialization:**  $X_{[-\delta]} = Y_{[-\delta]} = P_{[-\delta]} = 0$

**Recurrence:** for  $j = -\delta, -\delta+1, \dots, N-1$  do

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

we keep  $r = 2$  throughout this paper with the corresponding redundant digit set  $\{\bar{1}, 0, 1\}$ . In this case we have  $\delta = 3$ , and  $sel()$  is given by (3) [6], from which it can be seen that the decision is made only based on 1 integer bit and 1 fractional bit of  $W_{[j]}$ .

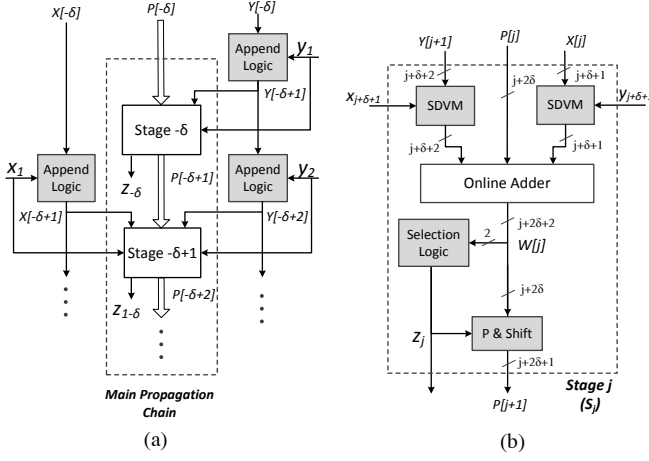
$$sel(W_{[j]}) = \begin{cases} 1 & \text{if } W_{[j]} \geq \frac{1}{2} \\ 0 & \text{if } -\frac{1}{2} \leq W_{[j]} < \frac{1}{2} \\ \bar{1} & \text{if } W_{[j]} < -\frac{1}{2} \end{cases} \quad (3)$$

Algorithm 1 can be synthesized into a unrolled digit parallel structure as shown in Figure 3(a). In an  $N$ -digit OM, there are totally  $N + \delta$  stages, of which the online inputs are generated from the appending logic according to (1). For each stage, an online adder is used as shown in Figure 3(b). The SDVM module performs the signed-digit vector multiplication. When  $r = 2$  and the digit set  $\{\bar{1}, 0, 1\}$  is used, the implementation of SDVM is straight forward, because for instance the output of  $(y_{j+\delta+1} \cdot X_{[j]})$  is 0,  $X_{[j]}$  or  $-X_{[j]}$  when  $y$  is equal to 0, 1 and  $-1$  respectively.

Instead of duplicating the digit-serial implementation  $N + \delta$  times, each stage in the digit-parallel architecture can be optimized for area reduction. For instance, the SDVM modules and the appending logic are not required in the last  $\delta$  stages because the inputs are 0. This leads to a smaller online adder in these stages. Similarly for the first  $\delta$  stages the selection logic can be removed, as the first digit of the result is generated at stage 0 ( $S_0$ ).

## 3. PROBABILISTIC MODEL OF OVERCLOCKING ERROR

As it is unlikely that timing violation happens in the online adder, in this section we model the overclocking error in the radix-2 online multiplier (OM). From Figure 3 we observe two types of delay chains. One is caused by generation and propagation of  $P_{[j]}$  among different stages. The other is the



**Figure 3: (a) Synthesis of Algorithm 1 into a digit-parallel online multiplier (b) Structure of one stage, in which the maximum digit widths of signals are labeled.**

generation of online inputs  $X_{[j]}$  and  $Y_{[j]}$  from the appending logic. Since the appending logic is basically wires and simple combinational logic [3], the overall latency will eventually be determined by the delay of the  $P_{[j]}$  path, especially with increasing operand word-lengths. As such, we initially model the delay of each stage within an OM to be a constant value  $\mu$ , as shown in Figure 3(b). We also assume that the generation of online inputs costs no delay.

Let  $\mu_{OM}$  denote the worst-case delay of an OM. It follows that if the clock period  $T_S$  is greater than  $\mu_{OM}$ , correct results will be sampled. If, however, faster-than-rated sampling frequencies are applied such that  $T_S < \mu_{OM}$ , timing violations might happen and intermediate results will be sampled, potentially generating errors. For a given  $T_S$ , the maximum length of error-free propagation is described by (4) where  $f_s$  denotes the sampling frequency. To avoid too large frequencies, we always ensure that the first digit of the result will be generated correctly, i.e.  $b > \delta$ .

$$b := \left\lceil \frac{T_S}{\mu} \right\rceil = \left\lceil \frac{1}{\mu \cdot f_s} \right\rceil \quad (4)$$

We then model the errors assuming that every digit of each input to our circuit is uniformly and independently generated with the digit set  $\{\bar{1}, 0, 1\}$ . This assumption will be relaxed in Section 4 where real image data are used.

### 3.1 Probability of Timing Violations

For an  $N$ -digit OM, let a propagation chain be generated at stage  $S_\tau$  with the length of  $d(\tau)$  digits, then the stage number  $\tau$  is bounded by (5). The presence of timing violation requires  $d(\tau) > b$ . Besides, the chain cannot propagate over stage  $S_{N-1}$ . Therefore the bound of parameter  $d(\tau)$  is given by (6).

$$-\delta \leq \tau \leq N - 1 - b = \tau_{max} \quad (5)$$

$$b < d(\tau) \leq N - 1 - \tau \quad (6)$$

However, the actual length of a propagation chain is dependent upon input patterns. For  $S_\tau$ , let the specific input pattern of the 1-digit input be represented by  $C(\tau)$ , which can be classified into four types, as listed in (7). Under the

assumption that all digits of input signal are mutually independent, the probability of  $C_1 \sim C_4$  equal to 1/9, 4/9, 2/9 and 2/9 respectively.

$$C(\tau) = \begin{cases} \text{Case 1 } (C_1) : & x_{\tau+\delta+1} = 0, y_{\tau+\delta+1} = 0 \\ \text{Case 2 } (C_2) : & x_{\tau+\delta+1} \neq 0, y_{\tau+\delta+1} \neq 0 \\ \text{Case 3 } (C_3) : & x_{\tau+\delta+1} \neq 0, y_{\tau+\delta+1} = 0 \\ \text{Case 4 } (C_4) : & x_{\tau+\delta+1} = 0, y_{\tau+\delta+1} \neq 0 \end{cases} \quad (7)$$

The classification based on whether a digit is zero, as all internal signals are reset to zero initially. In this case, (2) can be modified to give (8) for  $\tau > -\delta$ . If  $C(\tau) = C_1$ , no “carry” chain will be generated at  $S_\tau$  because  $P_{[\tau+1]} = 0$ .

$$\begin{aligned} P_{[\tau+1]} &= r \cdot W_{[\tau]} \\ &= r^{-\delta+1} (x_{\tau+\delta+1} Y_{[\tau+1]} + y_{\tau+\delta+1} X_{[\tau]}) \end{aligned} \quad (8)$$

In other cases,  $P_{[\tau+1]} \neq 0$  and a new chain will be generated at  $S_\tau$ . We denote the number of leading non-zero digits of  $P_{[\tau+1]}$  to be  $D$ . That is,  $P_{[\tau+1]}$  changes  $D$  digits for chain generation. From (2) we notice that  $D = D - 1$  for each propagation because of the 1-digit shifting when computing  $P_{[\tau+1]} = r(W_{[\tau]} - z_\tau)$ , and this is independent of the inputs. Finally, this chain will be annihilated when  $D = 1$ . As a summary, the actual chain length  $d(\tau)$  corresponds to the value of  $D$ , which we consider separately for different  $C(\tau)$ .

If  $C(\tau) = C_2$ ,  $D$  equals to the maximum word-length of  $P_{[\tau+1]}$  as shown in Figure 3(b), i.e.  $D = \tau + 2\delta + 1$ . Combining  $D$  and (6) gives the longest chain length in (9).

$$\begin{aligned} d(\tau)_{max} &= \text{Min}(D, N - 1 - \tau) \\ &= \text{Min}(\tau + 2\delta + 1, N - 1 - \tau) \end{aligned} \quad (9)$$

For  $C_3$  and  $C_4$ , the carry generation, propagation and annihilation behavior of both cases are identical. Hence we only discuss  $C(\tau) = C_3$ . In this case, (8) is modified to give (10). Thus  $D$  only depends on the number of leading non-zero digits of  $Y_{[\tau+1]}$ . Besides, we have  $Y_{[\tau+1]} = Y_{[\tau]}$  because the appending digit  $y_{\tau+\delta+1} = 0$ . It means  $d(\tau)$  becomes dependent of the inputs of previous stage.

$$P_{[\tau+1]} = r^{-\delta+1} (x_{\tau+\delta+1} Y_{[\tau+1]}) \quad (10)$$

For the first stage where  $\tau = -\delta$ , (8) can be modified further yielding  $P_{[-\delta+1]} = r^{\delta+1} (x_1 Y_{[-\delta+1]})$ . Therefore the value in (9) can only be obtained when  $C(-\delta) = C_1$ , otherwise  $d(-\delta) = 0$ .

For  $\tau > -\delta$ , there are 2 possible situations as stated below:

- For the previous stage if  $y_{\tau+\delta} \neq 0$ , then the word-length of  $Y_{[\tau]}$  is maximized. Thus  $d(\tau) = d(\tau)_{max} - 1$ .
- If  $y_{\tau+\delta} = 0$ , then  $Y_{[\tau]} = Y_{[\tau-1]}$ . This is a recursive process and these two judgements will be performed again for  $\tau = \tau - 1$ . It terminates when  $\tau = -\delta$  or the first judgement is satisfied.

In summary, Algorithm 2 is derived to examine all possible stages that timing violations may occur, and all possible input patterns for each stage. This algorithm returns  $\text{Prob}(T_S)$ , which denotes the probability that timing violations happen in an  $N$ -digit online multiplier as a function of  $T_S$ . In this algorithm,  $\tau_{max}$  and  $b$  are obtained from (5) and (4), respectively. The probability of the input pattern for stage  $i$  is denoted by  $\text{Prob}(C(i))$ , which can be determined through (7).

**Algorithm 2** Probability of Timing Violations

---

```

1:  $Prob(T_S) \leftarrow 0$ 
2: for all stages  $\tau \in \{-\delta, -\delta + 1, \dots, \tau_{max}\}$  and all input
   cases  $C(\tau) \in \{C_1, \dots, C_4\}$  do
3:   Determine  $d(\tau)$  based on  $C(\tau)$ 
4:   if  $d(\tau) > b$  then
5:      $Prob(T_S) \leftarrow Prob(T_S) + \prod_{i=-\delta}^{\tau} Prob(C(i))$ 
6:   end if
7: end for
8: return  $Prob(T_S)$ 

```

---

### 3.2 Magnitude of Overclocking Error

In the presence of timing violation, multiple chains might not be correctly propagated. Let us consider the first chain that timing violation may occur, i.e.  $d(\tau)_{max} > b$ , denote that it annihilates at  $S_\lambda$  where  $\lambda = \tau + d_{max}(\tau) - 1$ . Therefore overclocking errors may happen from digit  $\lambda$  to digit  $N - 1$  of the result. In general, the magnitude of overclocking error is given by (11), where  $z_i$  and  $z_i'$  denote the correct value and the actual value of the output digit of  $S_i$ , separately, and  $\varepsilon_i \in \{\pm 2, \pm 1\}$ .

$$|\varepsilon| = \left| \sum_{i=\lambda}^{N-1} 2^{-i-1} (z_i - z_i') \right| = \left| \sum_{i=\lambda}^{N-1} \varepsilon_i \right| \quad (11)$$

However, the change of output digits may not result in a different number with online arithmetic, because the number is represented in a redundant form. For instance, the two's complement number 0.111 can be represented in the online form as 0.10 $\bar{1}$ , 0.1 $\bar{1}$ 1 and 0.111. In contrast, errors will always be generated in this case with conventional arithmetic.

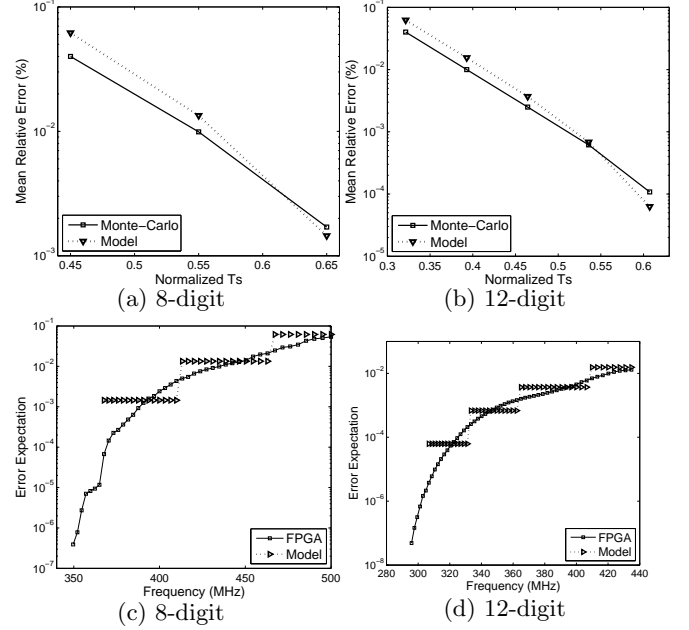
### 3.3 Expectation of Overclocking Error

For a given  $T_S$ , the combination of  $Prob(T_S)$  and  $|\varepsilon|$  in Algorithm 2 and (11) yields the overall expectation of overclocking error, as presented in (12). The verification of the proposed model is illustrated in the top row of Figure 4 against the results from Monte-Carlo simulations based on the aforementioned timing model.  $T_S$  is normalized with respect to the value from structural timing analysis, i.e.  $(N + \delta)\mu$ . Note that both results are obtained when assuming the input data are randomly generated following uniform distribution. It can be seen that the modeled value match well with the simulation results. We also verify the models against the FPGA results without timing assumptions, as illustrated in the bottom row in Figure 4. The general behavior of the model is similar to the FPGA implementation, however it does not capture the small increments in errors because we only model the coarse delay which is in units of  $\mu$ . Whereas the other sources of delay are not modeled, such as the generation of the online inputs and result digits.

$$E_{ovc} = Prob(T_S) \cdot |\varepsilon| \quad (12)$$

For all possible  $\tau$  as bounded by (5), the probability of different chain lengths within an  $N$ -digit OM and the corresponding magnitude of overclocking error can be obtained, as shown in Figure 5 with  $N = 8, 12, 16, 32$ , respectively. For a given  $T_S$ , the overall expectation of overclocking error in (12) can also be computed by (13).

$$E_{ovc} = \sum_{d>b} P_d \cdot \varepsilon_d \quad (13)$$



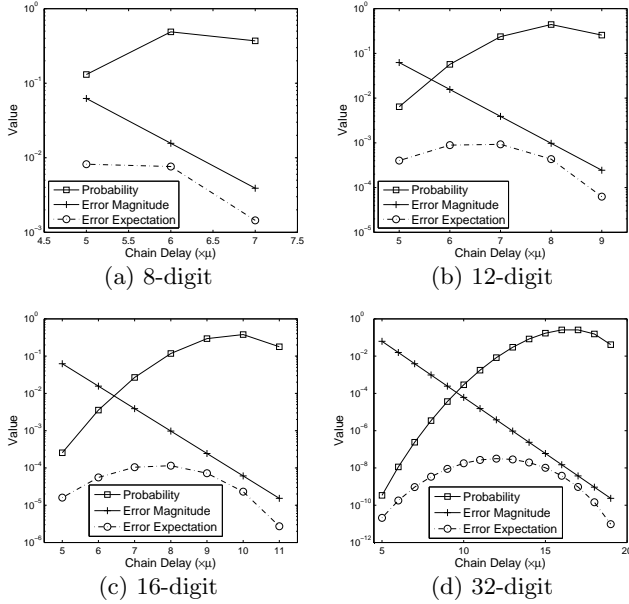
**Figure 4: Expectation of overclocking error for online multipliers: verification of the proposed model against Monte-Carlo simulations with timing assumptions (top row) and FPGA results with real timing information (bottom row).**

From Figure 5 several observations can be made. First, the error magnitude decreases exponentially with longer chain lengths, because timing violation firstly affects LSDs with online arithmetic. Second, we see that chains with longer delay would happen with greater probabilities in an OM. This is because for the unrolled radix-2 OM,  $d(\tau)$  is only dependent upon the inputs for chain generation instead of propagation and annihilation. Thus long chain is generated as long as both input digits are not zeros with the probability of 4/9. Compared to the traditional arithmetic, carry generation, propagation and annihilation are all decided by specific input patterns. This limits the overall probability of long chains. However, we also notice that for chains with long delays, the increasing in likelihood of error is outweighed by the decrease in magnitude of error. Therefore the combination of both results in a decline in error expectation. In contrast, one would expect error expectation to grow faster as the amount of overclocking is increased when using traditional arithmetic because errors occur in MSBs.

## 4. CASE STUDY: IMAGE FILTER

### 4.1 Experimental Setup

The benefits of the proposed methodology are demonstrated by using a  $3 \times 3$  Gaussian image filter, which contains 9 multipliers and 8 adders, with two types of computer arithmetic. One is the radix-2 traditional arithmetic with data represented using 2's complement form. In this case, speed-optimized adders and multipliers are created using Xilinx Core Generator [15]. The other is the radix-2 online arithmetic, with the basic building blocks as described in Section 2.2 and Section 2.3. In this case, all inputs and outputs are represented in the online format. In order to achieve the desired latency between input and output, both designs are overclocked and the errors seen at the output are



**Figure 5: Probabilities of different chain delay, the corresponding magnitude of overclocking error and the combination of both in terms of error expectation in a radix-2 OM.**

recorded. The results are obtained from a Xilinx Virtex-6 FPGA through post place-and-route simulations. The results are evaluated in terms of mean relative error (MRE), which represents the percentage of error at outputs, as given by (14) where  $E_{error}$  and  $E_{out}$  refer to the mean value of error and correct output, respectively.

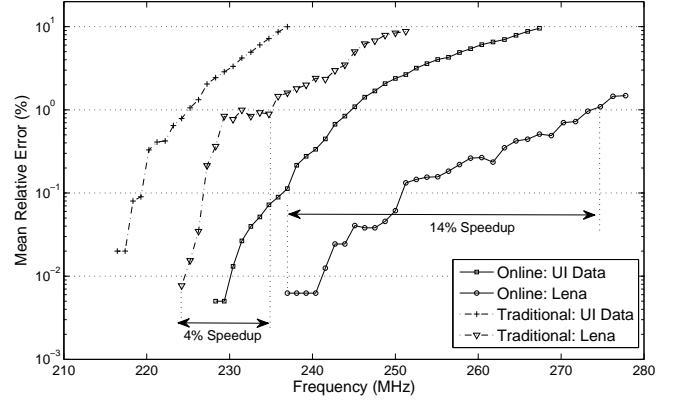
$$MRE = \left| \frac{E_{error}}{E_{out}} \right| \times 100\% \quad (14)$$

In our experiments, two types of input data are utilized. One is randomly sampled from a uniform distribution of  $N$ -digit numbers. This type is referred to as “Uniform Independent (UI) inputs”. The other is called “real inputs”, which are the pixel values of several  $512 \times 512$  benchmark images.

## 4.2 Quantify the Impact of Overclocking

The MRE values of the image filter with traditional arithmetic (dotted lines) and online arithmetic (solid lines) when  $N = 8$  are illustrated in Figure 6. According to the timing analysis tool, the rated operating frequencies of the two designs are 168.7MHz and 148.3MHz, respectively. However as seen in Figure 6, for the UI inputs, design with online arithmetic actually operates at a higher frequency without timing violations in comparison to the design using traditional arithmetic. This error-free frequency is even larger when using the real image data as inputs, since the real data do not exactly follow the uniform distribution or the independent assumption. In Figure 6 the “Lena” benchmark image is used as the real inputs.

If errors can be tolerated, we may allow timing violations to happen for better performance. The sensitivity of overclocking error for a given arithmetic can be evaluated by the data slope in Figure 6. For instance under an error budget of 1% MRE, using the UI inputs the frequency of the traditional design can be improved by 3.89% with respect to the maximum frequency without errors, whereas the design with



**Figure 6: Overclocking error in an image filter with two types of computer arithmetic: online arithmetic and standard binary arithmetic, of which the rated frequencies are 148.3MHz and 168.7MHz, respectively, according to the timing analysis tool.**

online arithmetic can be overclocked by 6.85%. This indicates that online arithmetic is less sensitive to overclocking, as illustrated in Section 3.3. The difference is even greater using real image data: 13.74% frequency speedup with online arithmetic against 4.04% with traditional arithmetic, because fewer long chains are generated with real inputs.

The output images for both design scenarios are presented in Figure 7. Since the overclocking errors are in the LSDs of the results with online arithmetic, the degradation on the image can be hardly observed. In contrast, timing violations cause error in the MSDs with traditional arithmetic. This leads to “salt and pepper noise” and severe quality loss as shown on the images in the right column of Figure 7. Meanwhile, errors in the MSDs would result in large noise power and therefore the signal-to-noise ratio (SNR) for the traditional design is small.

We also perform experiments using other benchmark images. The results are summarized in Table 1 in terms of the relative reduction of MRE as given by (15) where  $MRE_{OL}$  and  $MRE_{Trad}$  denote the value obtained with online arithmetic and with traditional arithmetic, respectively, and in Table 2 for the differences of SNR. In both tables the frequency is normalized to the maximum error-free frequency for each arithmetic. A significant reduction of MRE can be observed using online arithmetic for all input types. The geometric mean reduction of MRE is 89.2% using UI data. Even larger differences of MRE can be achieved when testing with real image data, varying from 97.3% to 98.2%, as expected given the results shown in Figure 6. Similarly from Table 2, the differences in SNR are 21.4dB ~ 43.9dB.

$$\frac{MRE_{Trad} - MRE_{OL}}{MRE_{Trad}} \times 100\% \quad (15)$$

The area comparison between two designs is illustrated in Table 3. While we notice that our approach comes at some increase in area, the FPGA architecture is optimized for conventional arithmetic. For instance, the Virtex series employs dedicated multiplexers and encoders for very fast ripple carry addition [14]. Besides, at least 2 orders of magnitude error reduction is obtained for a given frequency in our design, as shown in Figure 6. Although the differences in



(a)  $1.05f_0$ , SNR=38.3dB



(b)  $1.05f_0'$ , SNR:21.5dB



(c)  $1.15f_0$ , SNR:36.2dB



(d)  $1.15f_0'$ , SNR:9.0dB



(e)  $1.25f_0$ , SNR:26.7dB



(f)  $1.25f_0'$ , SNR:8.5dB

**Figure 7: Output images of image filter using online arithmetic (left column) and traditional arithmetic (right column), where  $f_0$  and  $f_0'$  denote the maximum error-free frequencies for each design.**

both error and area can be compensated by using more digits with traditional arithmetic, this will result in longer delay and an even larger gap in frequency between two designs.

**Table 1: Relative Reduction of MRE with Online Arithmetic for Various Normalized Frequencies.**

Inputs	Normalized Frequency					Geo. Mean
	1.05	1.10	1.15	1.20	1.25	
Uniform	94.5%	89.1%	90.1%	88.3%	84.3%	89.2%
Lena	99.3%	99.2%	98.9%	97.7%	94.9%	97.9%
Pepper	99.7%	98.3%	98.1%	97.2%	95.1%	97.7%
Sailboat	99.5%	97.9%	97.3%	96.8%	95.1%	97.3%
Tiffany	99.9%	97.6%	98.4%	97.8%	97.2%	98.2%

## 5. CONCLUSION

In this paper, we have studied the probabilistic behavior of key arithmetic primitives with different computer arithmetic when operating beyond the deterministic region. Through the usage of analytical models and empirical FPGA results, we have demonstrated that significant error reduction and performance improvement can be achieved by using the “overclocking friendly” online arithmetic.

## 6. REFERENCES

- [1] T. Austin, D. Blaauw, T. Mudge, and K. Flautner. Making typical silicon matter with razor. *IEEE Trans.*

**Table 2: Improvement of SNR (dB) with Online Arithmetic for Various Normalized Frequencies.**

Inputs	Normalized Frequency				
	1.05	1.10	1.15	1.20	1.25
Lena	44.6	36.3	33.2	29.1	22.9
Pepper	35.7	28.3	28.7	25.9	24.1
Sailboat	33.7	27.5	26.3	25.0	21.7
Tiffany	43.9	25.9	29.5	25.6	24.5

**Table 3: Area Comparison between Two Designs.**

Metric	Arithmetic Type		Overhead
	Traditional	Online	
LUTs	912	1896	2.08
Slices	324	525	1.62

on Computer, 37(3):57–65, 2004.

- [2] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Trans. on Electronic Computers*, EC-10(3):389–400, 1961.
- [3] M. Ercegovac and T. Lang. On-the-fly conversion of redundant into conventional representations. *IEEE Trans. on Computer*, C-36(7):895–897, 1987.
- [4] M. D. Ercegovac. On-line arithmetic: An overview. In *Proc. Annual Technical Symp. Real time signal processing VII*, pages 86–93, 1984.
- [5] M. D. Ercegovac and T. Lang. *Digital arithmetic*. Morgan Kaufmann, 2003.
- [6] 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*, 12(1):52–66, 2004.
- [7] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(1):124–137, 2013.
- [8] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi. A high-speed multiplier using a redundant binary adder tree. *IEEE Jounl. of Solid-State Circuits*, 22(1):28–34, 1987.
- [9] Z. M. Kedem, V. J. Mooney, K. K. Muntimadugu, and K. V. Palem. An approach to energy-error tradeoffs in approximate ripple carry adders. In *Proc. Int. Symp. on Low Power Elec. and Design*, pages 211–216, 2011.
- [10] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *Proc. Int. Conf. on VLSI Design*, pages 346–351, 2011.
- [11] S. Rajagopal and J. Cavallaro. Truncated online arithmetic with applications to communication systems. *IEEE Trans. on Computers*, 55(10):1240–1252, 2006.
- [12] N. Takagi, H. Yasuura, and S. Yajima. High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Trans. on Computers*, 100(9):789–796, 1985.
- [13] K. S. Trivedi and M. Ercegovac. On-line algorithms for division and multiplication. *IEEE Trans. on Computers*, 26:161–167, 1975.
- [14] Xilinx. Virtex-6 FPGA configurable logic block user guide.

[15] Xilinx. LogiCORE IP multiplier v11.2, 2011.