Title

***Abstract*—**

***Index Terms*—Keywords**

I. Introduction

This paper is collocated into the reconfigurable computing context. An important problem into the reconfigurable computing is to analyse the error propagation when the circuit "under test" is setted with values of the frequency too high or in disagree with the values of the delay. The paper investigates the propagation of the error caused by a timing violation into a digital online multiplier.

.....

This work starts from the results obtained in a previus work on the Radix-2 Digit-parallel Online Multiplier (see [1]).
In [1] the autors have provided methodologies to implement a parallel version of the online multiplier and they have proposed a probabilistic model describing the propagations of the error caused by timing violation conditions. They also backed up the models with experimental results from an image processing application demostrating that this novel design methodology can lead to substantial performance benefits compared to the design method using conventional arithmetic. The purpose of this paper is to provide a numerical model estimating the error on the online multiplier and a model to estimate the probability of timing violation in order to provide an "a priori" estimation for the error expectation for different values of the circuit parameters (frequency, number of digits, online delay, etc).
In this paper we provide a formal model of the Overclocking Error for the radix-2 digit-parallel Online Multiplier. We also proved a mathematical '"support" to the experimental results obtained in [1] and, in particular, we estimate the expectation of the overclocking error for these results. We provide a model describing the probablity of timing violation as "chain start probability" and "chain rule probability" representing the probability that a chain starts during the operation and the probability that a chain starts & perpetuals for a fixed number of digits, respectively.

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 [**?**], [**?**]. 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". Normally δ a small

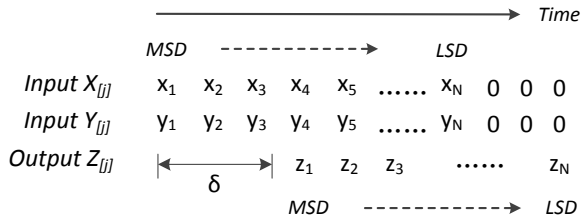


Fig. 1. Dataflow in digit-serial online arithmetic, in which both inputs and outputs are processed from the MSD to the LSD. δ denotes the online delay.

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 [**?**].

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

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) [**?**] and signed-digit (SD) [**?**]. 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.

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

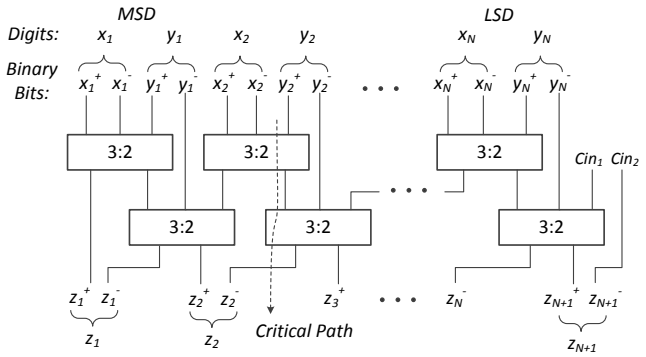


Fig. 2. An N-digit binary digit-parallel online adder. Both inputs and outputs are represented using SD representation. "3:2" denotes a 3:2 compressor.

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**

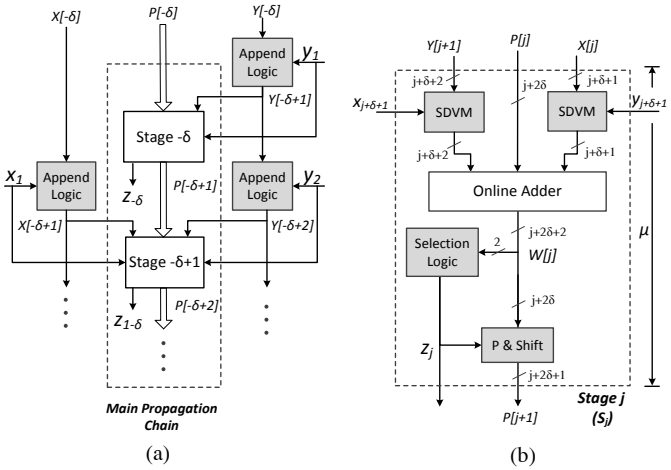
for building up more complex arithmetic operators such as multipliers to accelerate the sum of partial products [**?**].

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 [**?**] 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 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 $W_{[j]}$ [**?**].

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

Algorithm 1 can be synthesized into a unrolled digit parallel structure as shown in Figure 3(a). In an N-digit online multiplier (OM), there are totally $N + \delta$ stages, of which the online inputs are generated from the appending logic according to (1). In 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 $\{\overline{1}, 0, 1\}$ 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.



(a) (b)

Fig. 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.

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 δ stages because the inputs are 0. This leads to a smaller online adder in these stages. Similarly for the first δ stages the selection logic can be removed, as the first digit of the result is generated at stage 0 (S_0).

III. Probability 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 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 generation of online inputs $X_{[j]}$ and $Y_{[j]}$ from the appending logic. Since the appending logic is basically wires and simple combinational logic [**?**], 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 μ, as shown in Figure 3(b). We also assume that the generation of online inputs costs no delay.

Let μ_{OM} denote the worst-case delay of an OM. It follows that if the clock period T_S is greater than μ_{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 (3) where f_S denotes the sampling frequency. We always ensure that the first digit of the product will be generated correctly, i.e. $b > \delta$.

$$b := \left\lceil \frac{T_S}{\mu} \right\rceil = \left\lceil \frac{1}{\mu \cdot f_S} \right\rceil \tag{3}$$

We now determine when this timing constraint is not met and the size of error in this case.

A. Probability of Timing Violations

For an N-digit OM, let a propagation chain be generated at stage S_τ with the length of $d(\tau)$ digits, then the stage number τ is bounded by (4). The presence of timing violation requires $d(\tau) > b$. Besides, the chain cannot propagate over S_{N-1}. Thus the bound of parameter $d(\tau)$ is given by (5).

$$-\delta \leq \tau \leq N-1-b = \tau_{max} \tag{4}$$

$$b < d(\tau) \leq N-1-\tau \tag{5}$$

However, the actual length of a "carry" chain is dependent upon input patterns. We then examine the relationship between $d(\tau)$ and the inputs that corresponds to the generation, propagation and annihilation of this carry chain. Let the specific input pattern of stage S_τ be represented by $C(\tau)$, which can be classified into four types, as listed in (6).

$$C(\tau) = \begin{cases} Case1 & (C_1(\tau)): \quad x_{\tau+\delta+1} = 0, \quad y_{\tau+\delta+1} = 0 \\ Case2 & (C_2(\tau)): \quad x_{\tau+\delta+1} \neq 0, \quad y_{\tau+\delta+1} \neq 0 \\ Case3 & (C_3(\tau)): \quad x_{\tau+\delta+1} \neq 0, \quad y_{\tau+\delta+1} = 0 \\ Case4 & (C_4(\tau)): \quad x_{\tau+\delta+1} = 0, \quad y_{\tau+\delta+1} \neq 0 \end{cases} \tag{6}$$

The classification is based on whether a digit is zero, as all internal signals are reset to zero initially. Under the assumption that all digits are mutually independent and uniformly sampled from the digit set $\{\overline{1}, 0, 1\}$ as given in Section II, the probabilities of $C_1, ..., C_4$ are given in (7).

$$\begin{array}{ll} C_1(\tau): \frac{1}{(2a+1)^2}, & C_2(\tau): \frac{(2a+1)^2-2(a+1)-1}{(2a+1)^2} \\ C_3(\tau): \frac{a+1}{(2a+1)^2}, & C_4(\tau): \frac{a+1}{(2a+1)^2} \end{array} \tag{7}$$

Notice that no carry chain will be generated under Case 1, because $P_{[\tau+1]} = 0$. In other words, a chain could be generated if one of the cases $C_2(\tau)$,...,$C_4(\tau)$ occur. In this case we provide the following propositions which describe the probabilities of timing violations.

Proposition 3.1: (Chain rule probability)
Let $Pr_{\tau,d(\tau)}$ denote the probability that a carry chain start at stage S_τ, and it have lenght $d(\tau)$, $\forall \tau \in [-\delta, N-1-b]$, $\forall d(\tau) \in [b, N-1-\tau]$, then it is given by (8)

$$Pr_{\tau,d(\tau)} = \left(\frac{1}{(2a+1)^2}\right)^{\tau} \left(\frac{(2a+1)^2-1}{(2a+1)^2}\right)^{d(\tau)} \tag{8}$$

Proof:

Let E_2 be the event "the carry chain is generated at stage S_τ and it have length $d(\tau)$", we have

$$Pr_{\tau,d(\tau)} = Pr(E_2) = \underbrace{Pr(C_1)\cdots Pr(C_1)}_{\tau-1 \ times} \cdot \tag{9}$$

$$\cdot \underbrace{Pr(C_2 \ or \ C_3 \ or \ C_4)\cdots Pr(C_2 \ or \ C_3 \ or \ C_4)}_{d(\tau) \ times} Pr(C_1)$$

that is

$$Pr_{\tau,d(\tau)} = Pr(C_1)^{\tau-1} Pr(C_2 \cup C_3 \cup C_4)^{d(\tau)} Pr(C_1) \tag{10}$$

Substituting (7) into (10) yields (11).

$$Pr_{\tau,d(\tau)} = \left(\frac{1}{(2a+1)^2}\right)^{\tau-1} \left(\frac{(2a+1)^2-1}{(2a+1)^2}\right)^{d(\tau)} \frac{1}{(2a+1)^2} \tag{11}$$

Simplifying (11) will give (8).

♣

For all possible values of τ and $d(\tau)$, the possibility that timing violations happen is given by Pr in (12). According to (4) and (5), the ranges in which the parameters τ and $d(\tau)$ are defined depend upon the value of the swamping period T_S. Hence Pr is a function of T_S.

$$Pr = \sum_{\tau} \sum_{d(\tau)} Pr_{\tau,d(\tau)} \tag{12}$$

B. Estimation of Timing Errors

In the presence of timing violation, multiple chains might not be correctly propagated in the online multiplication, resulting in overclocking errors generated from LSDs. Let z_i and $z_i{}'$ denote the correct value and the actual value of the output digit at the stage S_i, respectively. Then we have $z_i{}' = z_i + \varepsilon_i$ where ε_i is referred to as the overclocking error. We now locate the first output digit z_λ that contains error. For a given T_S and a minimum value of τ such that $d(\tau)_{max} > b$, we have $\lambda = \tau + d(\tau)_{max} + 1$ for chain annihilation. As such, the overclocking error can be expressed by (13)

$$|\varepsilon| = \left| \sum_{j=\lambda}^{N} r^{-j} \varepsilon_j \right| \tag{13}$$

We would like to investigate the statistical characteristics of the ε. In general, the distribution of ε can be obtained from the convolution of variable ε_j where $j \in [\lambda, N]$. Each variable ε_j is sum of uniform iid variables with mean zero, therefore its mean is zero and the mean of ε is zero as well.

Let $v\varepsilon$ and $v\varepsilon_j$ denote the variances of variable ε and ε_j, respectively. We can form the relationship between them as given in (14) based on (13).

$$v\varepsilon = \sum_{j=\lambda}^{N} r^{-2j} v\varepsilon_j \tag{14}$$

Normally for a given algorithm, in this case the Algorithm 1, the value of ε_j can be determined based on the inputs, which are uniformly distributed with the digit set $\{\overline{a}, \cdots, 0, \cdots, a\}$. Hence for a single digit of inputs $X_{[j]}$ and $Y_{[j]}$, its variance can be denoted by $v\varepsilon^{in}$, which is a fixed value with respect to a given value of a. Let the variances of the input signals $X_{[j]}$ and $Y_{[j]}$ be denoted by $v\varepsilon_j^y$ and $v\varepsilon_j^x$, respectively. Then we have (15) according to (1).

$$v\varepsilon_j^y = v\varepsilon_j^x = \sum_{i=1}^{j+\delta} r^{-2i} v\varepsilon^{in} \tag{15}$$

In Algorithm 1, the variances of the input data propagate through for all iterations. For a single iteration j, we can calculate $v\lambda_j$ based on $v\varepsilon_j^y$ and $v\varepsilon_j^x$. For example initially the variance of $H_{[j]}$ is given by (16).

$$v\varepsilon_j^H = r^{-2\delta}(v\varepsilon^{in} v\varepsilon_{j+1}^y + v\varepsilon^{in} v\varepsilon_j^x) \tag{16}$$

This value propagates throughout the entire iteration and finally we have the value of $v\varepsilon_j$ in (17).

$$v\varepsilon_j = r^2 \left(r^{-2\delta} \cdot v\varepsilon_j^H\right) \tag{17}$$

Combining (16) and (17) yields

$$v\varepsilon_j = r^2 \left(r^{-2\delta} \left(v\varepsilon^{in} v\varepsilon_{j+1}^y + v\varepsilon^{in} v\varepsilon_j^x\right)\right). \tag{18}$$

Finally, (18) can be used to derive the expression of $v\varepsilon$ in (19) according to (14). Notice that this expression forms the relationship between the variance of overclocking error and the given input distribution.

$$\begin{aligned} v\varepsilon &= \mathcal{F}(T_S, \mu, v\varepsilon^{in}, \delta) \\ &= \textstyle\sum_{k=\lambda}^{N} r^{-2k} \left(r^{2(1-\delta)} \left(v\varepsilon^{in} v\varepsilon_{k+1}^y + v\varepsilon^{in} v\varepsilon_k^x\right)\right) \end{aligned} \tag{19}$$

IV. Model Verification with FPGA results

In this Section, we compare the proposed models for both the probability and the magnitude of overclocking errors with the experimental results from FPGAs. In the verification, we choose the redundant digit set $\{\overline{1}, 0, 1\}$, which is most commonly used for binary redundant representations.

A. Verification of Models for Error Probability

We initially verify the models for the probability of timing violations. With the usage of the redundant digit set $\{\overline{1}, 0, 1\}$, we have $a = 1$. Substituting this in (8) yields (20).

$$Pr_{\tau, d(\tau)} = \left(\frac{1}{9}\right)^{\tau} \left(\frac{8}{9}\right)^{d(\tau)} \tag{20}$$

Combining (20) and (12) we obtain the values of the probability of timing violations with respect to a variety of sampling period T_S. The results of an 8-digit OM are shown in Fig. 4. For comparison, we also plot the results obtained from a Virtex-6 FPGA, and the data are obtained from post place-and-route simulations. It can be seen that the modelled probability values match well with the experimental results.

B. Verification of Models for Error Magnitude

We then verify the proposed models for the estimation of timing errors. Fig. 5 shows the distribution of the error ε on the FPGA results of the Algorithm 1 with input data uniformly distributed into the digit set $\{\overline{1}, 0, 1\}$.

Several slice through Fig. 5 are presented in Fig. 6 for different sampling period values and the corresponding variances of overclocking errors. We observe that the profile of the error distribution changes as the value of the sampling period changes, and the width of the bell changes as well also (see Fig. 6). The value of the mean is zero for all the values of the sampling period, while the variance of the distribution changes as the sampling period changes. In particular, the values of the variance decrease as the sampling period increases in agreement with the information provided by the values of the probability of timing violation.

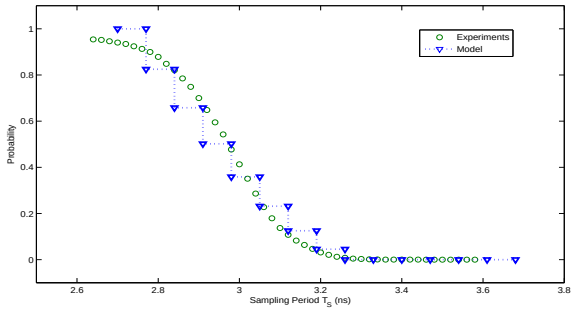


Fig. 4. Probability of timing violation for $N = 8$ and $\delta = 3$

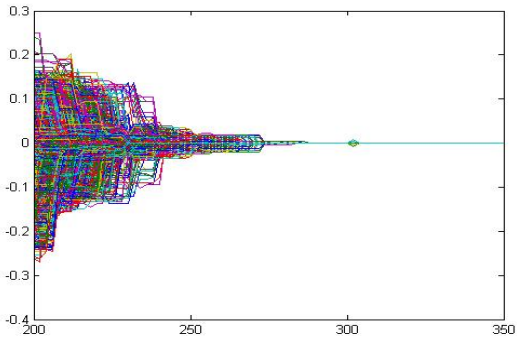


Fig. 5. Distribution of overclocking errors with respect to a variety of sampling period. The Xlable should be "Sampling period", which varies from 2ns to 3.5ns, and the Ylabel is "Error Values".

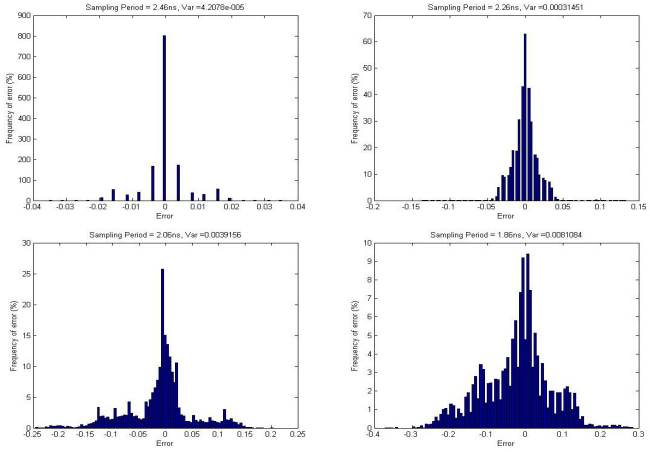


Fig. 6. Several slice through Fig. 5 that shows error distribution with respect to a variety of sampling periods and the corresponding variances of overclocking error. The results are obtained from experiments.

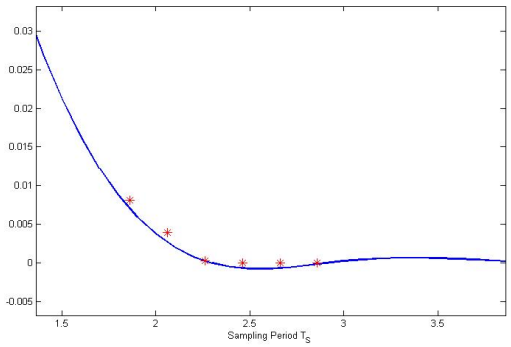


Fig. 7. Variances of overclocking errors. The results are obtained from the proposed models (blue line) and the experiments (red stars). (Legend and xylabels to be added.)

We have compared these experimental results with the results provided by the model defined in (19). The Figure 7 shows the curve provided by the model (blue line) and the experimental results (red stars). It can be seen that the modelled results are close to the experimental results.

V. Case Study: Image Filter

A. Experimental Setup

The benefits of the proposed methodology are demonstrated by using a 3×3 Gaussian image filter, which is implemented with two types of computer arithmetic. One is the binary online arithmetic while the other is the binary traditional arithmetic. For design with online arithmetic, all inputs and outputs are represented in the online format as given in (1). The basic building blocks: adders and multipliers as described in Section II-B and Section II-C are employed. For design with traditional arithmetic, all signals are represented using the 2's complement format. We build this type of image filter by using speed-optimized adders and multipliers which are created using Xilinx Core Generator [**?**].

In order to achieve the desired latency between input and output, both designs are overclocked and the errors seen at the output are recorded. The results are obtained from a Xilinx Virtex-6 FPGA xxx 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 (21) where E_{error} and E_{out} refer to the mean value of error and correct output, respectively.

$$MRE = \left| \frac{E_{error}}{E_{out}} \right| \times 100\% \tag{21}$$

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 512×512 benchmark images.

B. 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 Fig. 8. According to the timing analysis tool, the rated operating frequencies of the two designs are 169MHz and 148MHz, respectively. However as seen in Fig. 8, 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 Fig. 8 the "Lena" benchmark image is used as the real inputs.

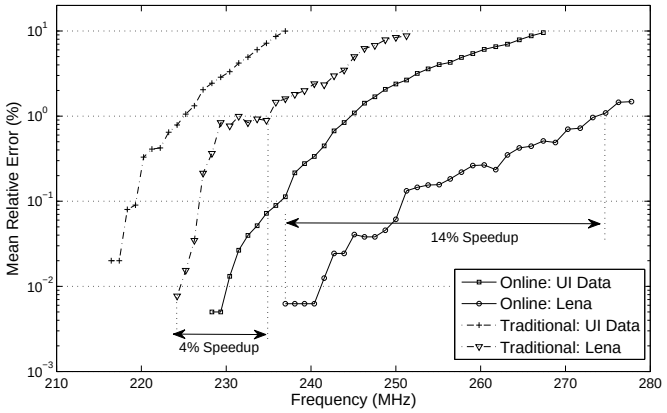


Fig. 8. Overclocking error in an image filter with two types of computer arithmetic: online arithmetic and standard binary arithmetic, of which the rated frequencies are 148.3MHz and 168.7MHz, respectively, according to the timing analysis tool.

If errors can be tolerated, we may allow timing violations to happen for better performance. The sensitivity of overclocking for a given arithmetic can be evaluated by the data slope in Figure 8. For instance under an error budget of 1% MRE, using UI inputs the frequency speedup of the traditional design is 3.89% with respect to the maximum frequency without errors, whereas the design with online arithmetic can be overclocked by 6.85%. This indicates that online arithmetic is less sensitive to overclocking, as discussed in Section **??**. The difference is greater using real image data: 13.74% frequency speedup with online arithmetic against 4.04% with traditional arithmetic, because longer chains happen with a smaller probability with real inputs.

The output images for both design scenarios are presented in Fig. 9. Since the overclocking errors are in the least significant ends 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 Fig. 9. Furthermore, errors in the MSDs result in large noise power. Hence the signal-to-noise ratio (SNR) of the traditional design is small.

C. Potential Benefits in Circuit Design

In general, our results could be of interest to a circuit designers in two ways. By choosing different arithmetic and data representations, either a circuit can be designed to operate at a certain frequency with the minimum possible MRE, or a

(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

Fig. 9. 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.

given error budget specified by the algorithm designer can be met with the fastest achievable frequency. For the first case, the experimental results obtained by UI inputs and 4 benchmark images are summarized in Table I in terms of the relative reduction of MRE as given by (22) where MRE_{OL} and MRE_{Trad} denote the value obtained with online arithmetic and with traditional arithmetic, respectively, and in Table II for the differences of SNR.

We also perform experiments using other benchmark images. The results are summarized in Table I and Table II in terms of the relative reduction of MRE and the improvements of SNR, respectively. In both tables the frequency is normalized to the maximum error-free frequency for each arithmetic. From Table I, a significant reduction of MRE can be observed using online arithmetic. The geometric mean reduction of MRE is 89.2% using UI data. Even larger reductions 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 8. Similarly from Table II, the improvements in SNR are $21.4dB \sim 43.9dB$.

$$\frac{MRE_{Trad} - MRE_{OL}}{MRE_{Trad}} \times 100\% \tag{22}$$

For the second design perspective, Table III illustrates the frequency speedups with different input types when specific error budgets can be tolerated. We see that for all input types using online arithmetic still outperforms the traditional design for each MRE budget in terms of operating frequency. Likewise the geometric mean of frequency speed-ups is larger for real image inputs.

TABLE I
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%

TABLE II
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 III
RELATIVE IMPROVEMENT IN FREQUENCY WITH ONLINE ARITHMETIC FOR VARIOUS ERROR BUDGETS.

Inputs	**Error Budget**				**Geo. Mean**
	0.01%	0.1%	1%	10%	
Uniform	N/A	4.59%	8.78%	12.83%	8.03%
Lena	7.96%	11.50%	16.39%	13.71%	11.88%
Pepper	6.22%	8.87%	18.68%	15.94%	11.32%
Sailboat	5.71%	8.87%	16.93%	15.29%	10.70%
Tiffany	2.35%	6.90%	18.58%	12.22%	7.79%

D. Area Overhead of Our Approach

The area comparison between two designs is illustrated in Table IV. 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 [**?**]. Besides, at least 2 orders of magnitude error reduction is obtained for a given frequency in our design, as shown in Figure 8. Although the differences in 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 IV
FPGA RESOURCE USAGE COMPARISON.

Metric	**Arithmetic Type**		**Overhead**
	Traditional	Online	
Look-Up Tables	912	1896	2.08
Slices	324	525	1.62

REFERENCES

[1] Datapath synthesis for overclocking: Online Arithmetic for Latency-Accuracy Trade-offs.
[2] J. B. Uspensky, Introduction to Mathematical Probability (New York: McGraw-Hill, 1937)
[3] Ross, Introduction to Probability Models, University of Southern California - Los Angeles (CA), 2010 ELSEVIER
[4] C. de Boor, A Practical Guide to Splines, Springer-Verlag, 1978.
[5] Kendall E. Atkinson, On the order of convergence of natural cubic spline interpolation, SIAM Journal on Numerical Analysis, Vol 5 No 1, 1968
[6] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.