Variation-Aware Pipelined Cores through Path Shaping and Dynamic Cycle Adjustment: Case Study on a Floating-Point Unit

Tsiokanos, I., Mukhanov, L., Nikolopoulos, D., & Karakonstantis, G. (2018). Variation-Aware Pipelined Cores through Path Shaping and Dynamic Cycle Adjustment: Case Study on a Floating-Point Unit. In *2018 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED): Proceedings*

Published in:
2018 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED): Proceedings

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
© 2018 IEEE.
This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Open Access
This research has been made openly available by Queen's academics and its Open Research team. We would love to hear how access to this research benefits you. – Share your feedback with us: http://go.qub.ac.uk/oa-feedback

Download date:26. Apr. 2024

Variation-Aware Pipelined Cores through Path Shaping and Dynamic Cycle Adjustment: Case Study on a Floating-Point Unit

Ioannis Tsiokanos, Lev Mukhanov, Dimitrios S. Nikolopoulos and Georgios Karakonstantis
Institute of Electronics, Communications and Information Technology (ECIT), Queens University Belfast
Email: {itsiokanos01, l.mukhanov, d.nikolopoulos, g.karakonstantis}@qub.ac.uk

ABSTRACT

In this paper, we propose a framework for minimizing variation-induced timing failures in pipelined designs, while limiting any overhead incurred by conventional guardband based schemes. Our approach initially limits the long latency paths ($LLPs$) and isolates them in as few pipeline stages as possible by shaping the path distribution. Such a strategy, facilitates the adoption of a special unit that predicts the excitation of the isolated $LLPs$ and dynamically allows an extra cycle for the completion of only these error-prone paths. Moreover, our framework performs post-layout dynamic timing analysis based on real operands that we extract from a variety of applications. This allows us to estimate the bit error rates under potential delay variations, while considering the dynamic data dependent path excitation. When applied to the implementation of an IEEE-754 compatible double precision floating-point unit (FPU) in a 45nm process technology, the path shaping helps to reduce the bit error rates on average by 2.71× compared to the reference design under 8% delay variations. The integrated $LLPs$ prediction unit and the dynamic cycle adjustment avoid such failures and any quality loss at a cost of up-to 0.61% throughput and 0.3% area overheads, while saving 37.95% power on average compared to an FPU with pessimistic margins.

CCS CONCEPTS

•Hardware → Arithmetic and datapath circuits; Very large scale integration design; Robustness;

KEYWORDS

Variation-aware FPU, error-resilience, path shaping, DTA

1. INTRODUCTION

The aggressive shrinking of transistor sizes has led to a 25% delay increase [9] and 20× higher leakage variation [5] in advanced nanometer technologies. These trends render circuits prone to failures and prevent them from meeting power and performance specifications [9]. Operation under scaled voltage, which is considered as the most effective method for saving power, worsens variations and makes circuits more prone to static or dynamic timing failures [6].

State-of-the-Art. Conventional wisdom dictates the adoption of timing guardbands by up-scaling the supply voltage or the clock period to avoid failures [13], [6]. However, such margins are overly pessimistic. They are selected under the assumptions of few worst-case critical paths, which are in turn estimated statically under worst-case input patterns and operating conditions that may be extremely rare. This approach ultimately forces all manufactured chips, even "good" chips (at typical or fast corner) that are not affected by variations, to operate at a much slower speed or at a higher power than what they can potentially achieve [9].

In an attempt to trim down the introduced guardbands, Statistical Static Timing Analysis (SSTA) tools [4] have been proposed. These tools still focus on improving the analysis rather than the design itself. Design-centric techniques try to avoid the timing margins by introducing special circuits to either detect and correct any error in-situ [16] or predict the activation of the long latency paths ($LLPs$) and change the cycle time [8] or replay the failing instructions [6]. Although effective, such schemes impose difficult to meet design constraints and may lead to large recovery overheads especially if the activation probability of the error-prone $LLPs$ is high [16]. An approach proposed in [10] may help to reduce the overall number of the $LLPs$, but it has never been considered jointly with any of the above design-centric techniques and it has not been applied to pipelined designs. Recent works have tried to exploit the instruction and operand dependent dynamic path excitation [7], [15] in pipelined cores. However, the proposed clock frequency adjustment per instruction may be very challenging to be applied in practice. Furthermore, none of the above schemes were applied in pipelined FPUs, which are among the most time consuming and complex processing units within any modern processor. The few existing works on variation-aware FPUs, rely on dual-VDD techniques [11] or optimizations at software/architecture level [14], but since they did not reduce the number of the $LLPs$ within the overall pipeline, they still left unexploited a lot of room for improvements.

Contributions. The primary aim of this paper is to limit the potential timing failures and avoid any costly guardband in pipelined designs. The essential design strategy lies on the minimization of the $LLPs$ before utilizing any other scheme for exploiting the rare dynamic activation of the few remaining $LLPs$, which was not attempted by existing variation-aware schemes. We chose an IEEE-754 compliant FPU [1] to demonstrate the efficacy of our approach, since FPUs are excellent representatives of complex pipelined designs. The contributions of this work can be summarized as follows:

- We develop a framework to redesign a pipelined circuit by carefully shaping the path distribution such that the $LLPs$ are significantly reduced and isolated to as few pipeline stages as possible. The advantages of such an approach are two-fold. First, it reduces the critical $LLPs$ and thus significantly reduces the failure probability. Second, it limits the number of registers/stages, where any error correction or prediction circuit would need to be integrated for addressing potential timing violations, thus limiting any resulting overheads.
- We implement a Long Latency Prediction Unit ($LLPU$) that exploits the inherent rare activation of the $LLPs$ within the FPU core. The unit predicts the $LLPs$ excitation and dynamically provides one extra cycle for the completion of such paths in case they are triggered. Note that since the $LLPs$ are made rare by design (i.e. through path-shaping) and their excitation is also rare as it depends only on few input patterns, the incurred throughput overhead (due to cycle adjustment) is limited by design.
- We develop a post-layout dynamic timing analysis (DTA)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ISLPED '18, July 23–25, 2018, Seattle, WA, USA

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-5704-3/18/07…$15.00

DOI: https://doi.org/10.1145/3218603.3218617

tool that estimates the dynamic excitement of timing paths and potential timing failures considering the operands of executed instructions. The operands used as input to the DTA tool are extracted from applications, which are executed on a compatible RISC processor.

- We estimate the bit error rate (BER) introduced by the variation-induced timing failures under worst-case delay increase for the original/unmodified and the proposed FPU designs. Based on this rate, we evaluate the quality loss quantified in terms of Signal-to-noise ratio (SNR) for various applications, which cover a wide range of domains including Computer Graphics, Fluid Dynamics, Data Mining, Medical Imaging and Computer Vision.
- Our results for the considered benchmarks indicate that the proposed FPU can lead to 37.95% power savings on average at a cost of 0.3% area and up-to 0.61% throughput overheads compared to a guardband based FPU design, while avoiding any quality-loss.

The rest of the paper is organized as follows. Section 2 describes the proposed approach and the implemented design flow using state-of-the-art tools. In Section 3, we apply the proposed framework to the design of an IEEE-754 compatible FPU of a RISC processor; and Section 4 presents the experimental results. Conclusions are drawn in Section 5.

2. PROPOSED APPROACH AND DESIGN FLOW

Typically, a pipelined design consists of a set of N unique combinational paths $P = \{p_1, p_2 ... p_N\}$, which are characterized by their delays $D(p_i)$ for $i = 1, 2 ... N$. As in any synchronous design, the longest path across all S pipeline stages determines the clock period, such as:

$$CP_{STA} = \max_{\mathbf{s=1....S}} \left\{ \max_{\mathbf{p \in P^s}} \{D(p)\} \right\} = \max_{\mathbf{p \in P}} \{D(p)\} \qquad (1)$$

where P^s is the set of unique path-groups in any of the S pipeline stages for $s = 1, 2..., S$. The overall path set can be distinguished into a set of K Short Latency Paths ($SLPs$), which we define as: $P_{SLP} = p_1^{SLP}, p_2^{SLP} ... p_K^{SLP} \subset P$ and a set of M $LLPs$: $P_{LLP} = p_1^{LLP}, p_2^{LLP} ... p_M^{LLP} \subset P$. Assume that P_{SLP} is activated by a set of operands Op_{SLP} and can be completed within a time $T_{SLP} = \max\{D(P_{SLP})\}$, whereas P_{LLP} is activated by a set of operands Op_{LLP} and requires more time than T_{SLP}: $T_{SLP} < D(P_{LLP}) \leq CP_{STA}$. The delay distribution of such paths typically looks like the one shown in Figure 1a. Such a distribution is characterized by a so-called "timing wall", which is a consequence of how modern designs are optimized for power and area, subject to a frequency constraint: the $LLPs$ are optimized by gate up-sizing, while the inherently $SLPs$ are allowed to become near-critical for recovering any area or power costs. This "timing wall" has no negative impact on the adopted CP_{STA}. However, it critically affects the probability of failures since under any, even small, delay variation many paths can fail.

2.1 Path Shaping (PS) and Critical Stage Constraining

The first step of our approach is to eliminate such a "timing wall" by moving away from a path distribution with many $LLPs$ close to the CP_{STA}. Essentially, this sets as a goal the minimization of $|P_{LLP}|$ subject to some design constraints such as the target clock period, as well as the area and/or power. To achieve such a goal, we define appropriate timing constraints for different path-groups and impose them on the design during synthesis. By introducing such path constraints, we ensure that the inherently fast paths are not made slower as opposed to the conventional approach. The end goal is to obtain a path distribution similar to the one depicted in Figure 1b, where $|P_{SLP}| >> |P_{LLP}|$.

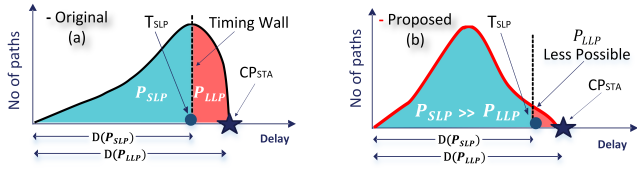

Figure 1: Original vs proposed path distribution

By doing so, we essentially make the $LLPs$ rare and ensure that at most cycles and in most stages only the $SLPs$ are being triggered, which can be completed within much less time than the CP_{STA}. In fact, if at time t the executed instruction that is in the pipeline stage s and the relevant input operands activate only paths from P_{SLP}, then a positive timing slack ($CP_{STA} - D(p_i^{SLP})$) can be observed. This slack can be used as a safety margin against potential delay variations, thus minimizing the probability of a timing failure at that stage. Note that in order to facilitate the isolation of discerned path-groups, we also make modifications at the micro-architectural or register-transfer level (RTL). These modifications do not only help to better control the path-groups, but also enable the isolation of P_{LLP} to as few pipeline stages as possible. This sets the ground for limiting the use of any error detection or correction mechanism only in the particular stage(s) where the $LLPs$ are isolated.

2.2 Dynamic Cycle Adjustment (DCA)

Conventional wisdom dictates that adding a delay margin to the CP_{STA} is necessary to avoid any timing failures induced by potential delay variations. However, such margins are overly pessimistic, since they ignore the fact that in each pipeline stage the $LLPs$ are not always excited. The aim of the second step in our framework is to take advantage of the dynamic sensitization of combinational paths and try to avoid timing failures by identifying in advance the operands that activate the few remaining $LLPs$. Upon detection of such operands, we provide an extra cycle to allow the activated $LLPs$ to be completed correctly.

To elucidate the basic concept, let us provide a simple example of a 6-bit ripple carry adder (RCA), as shown in Figure 2. In such a design, the most timing critical path (LLP_1) will be activated when the carry propagates all the way from $Ci, 0$ to $Co, 5$. By monitoring the carry propagate signal at the middle of such a datapath, i.e. at the 3^{rd} Full-Adder (FA2), which is given by $(A0 \oplus B0) \cdot (A1 \oplus B1) \cdot (A2 \oplus B2)$, we can identify if any LLP is going to be activated. In case that a LLP is excited we could provide an extra cycle for allowing such path to be completed even under any potential delay increase. In any other case, only the off-critical $SLPs$ will be excited, which have sufficient timing slack to address potential delay variations. The implementation of such a scheme allows us to avoid failures under any potential variation-incurred delay increase up-to a magnitude of: $\Delta T \leqslant (CP_{STA} - T_{SLP})$ and prevent the use of timing guardbands. The occasional two-cycle operations in case of $LLPs$ activation are expected to lead to a throughput loss. However, this can be kept small by design as we explain below.

To better understand the involved trade-offs, we define the execution time of any program, consisting of several operands, in a conventional design running at CP_{STA}, as:

$$ExT_{or} = \#cycles \times CP_{STA} \qquad (2)$$

In our design, the execution time considers also the two-cycle operations (i.e #2cycles) required when $LLPs$ are excited:

$$ExT_{pr} = (\#cycles(Op_{SLP}) + \#2cycles(Op_{LLP})) \times CP_{STA} \qquad (3)$$

To lower the overhead incurred by two-cycle operations, it is essential to limit the number of such operations. This could

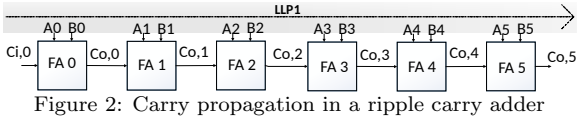

Figure 2: Carry propagation in a ripple carry adder

be achieved by reducing $|P_{LLP}|$, which is targeted by the applied PS, as well as the activation probability of the $LLPs$. This probability can be decreased by intelligently selecting the number of monitored bits in the $LLPU$. For instance, the carry propagation probability across the 1-bit FA2 in the previous example is $2(1-p')p'$, where p' denotes the input signal probability [8]. However, if we select to monitor the carry propagate signal over three FAs (e.g. from FA1 to FA3), then the carry propagation probability and consequently the activation probability of a two-cycle operation can be lowered down to $[2(1-p')p']^3$.

Finally, it is important to note that our approach does not change the CP_{STA} and thus any timing slack can be used either to address potential delay variations or to save power by scaling down the supply voltage in case of a "good" manufactured chip (i.e unaffected by variations).

2.3 Design and Analysis Phase

The proposed approach described above is realized by utilizing state-of-the-art electronic design automation (EDA) tools and enhances the conventional EDA flow as depicted in Figure 3. Initially, we set path-groups constraints during synthesis in the Synopsys Design Constraints (SDC) file for reducing $|P_{LLP}|$. Such constraints are explained in Section 3.1. After the synthesis step, we place and route (PnR) the design using the Innovus tool from Cadence. The design is then verified using STA based on Synopsys PrimeTime.

DTA: As we discussed our approach relies on the dynamic activation of paths to reduce the incurred overheads and minimize the timing failures. However, STA that is used by most existing approaches cannot capture the dynamic activation of paths due to the missing notion of path activation probabilities. Hence, to enable characterization of the data dependent path activation, we enhance the EDA flow with a DTA tool, which aims primarily at uncovering the unused timing margins of the pipeline design that are available at runtime. For the sake of this analysis, we use the post-layout gate-level simulation supported by ModelSim, which monitors the inputs and the outputs of all flip-flops in the design and generates a corresponding event log. To obtain this information and perform full back annotated simulation, we extract a standard delay format (SDF) file which describes the cell and interconnect delay as well as the RTL netlist and a testbench with relevant operands and provide them as input to ModelSim. Provided that every set of operands under nominal conditions produces an error-free output, we define this output as D_{gold}. Additionally, with this analysis, we evaluate how often the $LLPs$ are excited and the throughput overhead incurred by our approach as estimated by Eq. 3. Such an analysis phase also helps to extract instruction aware BERs, which depend on the dynamic excitation of critical paths triggered by specific operands. Finally,

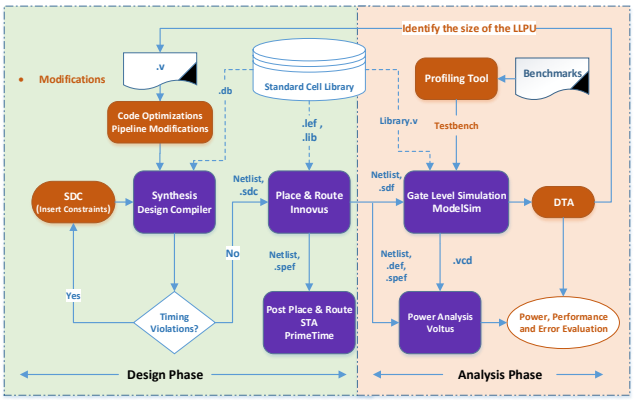

Figure 3: Workflow of the proposed approach. The introduced modifications highlighted in orange

this tool can extract an essential for power analysis value change dump (VCD) file, which contains information about the switching activity and value changes occurred during the simulation for nets and registers.

3. CASE STUDY: APPLICATION TO AN FPU

We apply the proposed approach to a multicycle, IEEE-754 compatible double precision FPU, following the representation: $-1^S \times M \times 2^E$, where S : sign, E : exponent and M : mantissa. In a double precision FP number, the most significant bit (MSB) indicates the sign, the next 11 bits represent the exponent and the mantissa consists of the last 52 bits. The applied FPU is a part of the latest Out-of-Order mor1kx MAROCCHINO pipeline, a 5-stage pipeline microprocessor based on the OpenRISC 1000 Instruction Set Architecture [3]. The following FP instructions are implemented: addition/subtraction, integer to FP and FP to integer conversions and comparison between FP numbers.

Figure 4a illustrates the micro-architecture of the targeted FPU, highlighting the FP addition/subtraction related stages. At Stage 1, an Order Control Buffer and a Pre-Normalize block are implemented, which detect any data dependencies in the instructions and adjust the exponent and mantissa, respectively. Stage 2 is responsible for the pre-addition/subtraction alignment, while Stage 3 performs the necessary multiplexing and shifting of the operands. Mantissa addition and exponent update are performed at Stage 4. Finally, rounding is implemented in the last two stages.

3.1 Redesigned FPU using PS

Initially, we apply the conventional EDA flow (see Figure 3) to the original unmodified design. This design is implemented using the typical corner of the Composite Current Source NanGate 45 nm cell library [2] (@1.1V). After following the synthesis and PnR steps, as well as performing STA, we built the path distribution that is shown in Figure 4b. The obtained distribution implies that the original performance-centric flow incurs large $|P_{LLP}|$, in which many paths are close to the worst-case delay (i.e CP_{STA}). Such a path distribution creates the "timing wall". Figure 5a depicts the path distribution within each pipeline stage, revealing that the "timing wall" exists in 4 out of the 6 stages. These findings indicate that there is an increased likelihood of timing failures in the stages where the $LLPs$ exist.

To circumvent this, we impose various constraints during the synthesis step by grouping paths at different stages based on their latency. As we explained in Section 2, we separate the paths into two different sets; the P_{SLP} and the P_{LLP} based on their computational delays. We define the delay target of the $SLPs$ as T_{SLP}. If the path delay is less than T_{SLP}, then this path is assigned to the P_{SLP}, otherwise it is assigned to the P_{LLP}. These constraints may have an impact on the area and power consumption, but the resulting overheads, which depend on the targeted path distribution, can be kept small. Initially, we apply strict constraints to shift the "timing wall" away from the target CP_{STA}. If there are timing violations after synthesis, we relax the design constraints and re-run our iterative method until the timing target is met. After many iterations we set the T_{SLP} in the particular FPU to 1.68ns. The above iterations are being implemented using tool command language (tcl) scripts, which after every synthesis round sort the paths into the two different sets and update the T_{SLP} based on the worst-case timing within the resulted groups. As a result, this automated procedure reduces $|P_{LLP}|$ (see Figure 4b), while ensuring all other paths are fast enough (at least 9.1% faster than the CP_{STA}) to tolerate variation-induced timing failures. Note that the timing constraints imposed by our design does not exceed the CP_{STA}, which is determined by the original unmodified design.

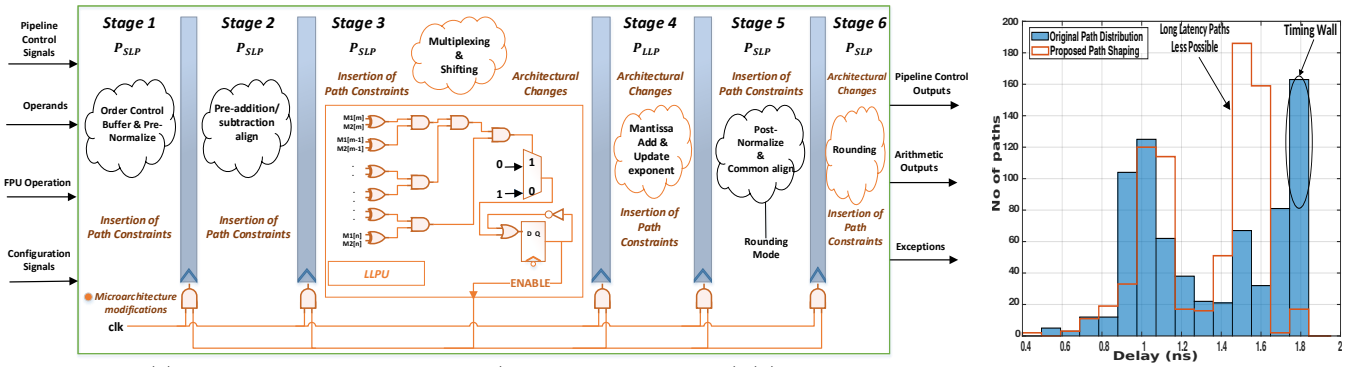

Figure 4: (a) Customized micro-architecture (modifications in orange) (b) Path distribution of the original and proposed FPU

After the above procedure, we identify the pipeline stages which have most of the few remaining $LLPs$ and we make some micro-architectural changes in order to restrict P_{LLP} to as few stages as possible. At this step, we noticed that some simple manual modifications of the RTL could help to optimize the design, and isolate all the $LLPs$ in one stage, while ensuring that the target timing constraints of the design are still met. In particular, rounding, which occurs at Stages 5 and 6 poses a bottleneck as it is applied to the result of all FP operations and thus many paths in these stages are long latency ones. To this end, we re-write part of the RTL code for Stages 5 and 6. Specifically, we modify some necessary conversions to negative numbers such that large conversions for rarely covered cases are eliminated. After the application of these micro-architectural changes, we isolate P_{LLP} to Stage 4, as depicted in Figure 5b. Note that in our design the few remaining $LLPs$ are isolated in such a way that they can be triggered only by FP addition/subtraction instructions at Stage 4.

3.2 Implementation and Integration of LLPU

Apart from the changes facilitating the desired PS, in the redesigned FPU, we also have to implement and integrate the $LLPU$. To achieve this, we alter the combinational logic in which FPU generates some pipeline control signals, especially a valid signal, at Stage 6. This valid signal indicates that the arithmetic output result is valid and ready to be read. We modify its logic considering a stall for one cycle when the $LLPU$ predicts excitation of the $LLPs$, rendering the arithmetic result invalid for this cycle (and waiting for one more cycle to get the valid result). Since all the $LLPs$ are restricted to Stage 4, we deploy the $LLPU$ only at Stage 3. To predict the activation of $LLPs$ in clock cycle i (CC_i), the $LLPU$ should indicate from the previous clock cycle (CC_{i-1}) if the input set for Stage 3 will trigger paths from

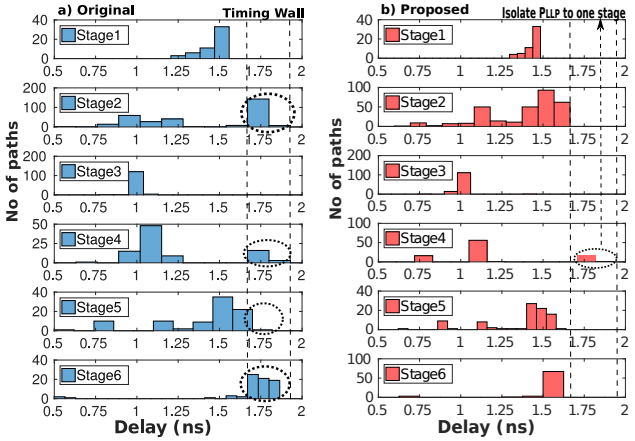

Figure 5: Original and proposed path distribution across stages

P_{LLP} at Stage 4. Specifically, the $LLPU$ produces an enable signal at Stage 3 which is handled at Stage 4. If this signal is 1, then the in-flight instruction and related operands activate paths from P_{SLP}. In this case, we use the normal one cycle operation with clock period CP_{STA}. Otherwise, a LLP is activated and we stall the pipeline by blocking the sampling of pipeline registers for one cycle. Given that Stage 4 implements the 52-bit mantissa addition, we deploy a $LLPs$ prediction scheme similar to [8], which has been limited to integer arithmetic units and has never been applied in conjunction with path shaping in pipelined cores before.

The circuit for the $LLPU$, depicted in Figure 4a, monitors $(m-n)$ bits of the two 52-bit addition operands $(M1, M2)$ and detects if carry propagates across the m^{th} bit, implementing the following logic (as explained in Section 2.2):

$$F(m,n) = (M1_m \oplus M2_m) \cdot (M1_{m-1} \oplus M2_{m-1}) \cdot ..(M1_n \oplus M2_n)$$

Only when F evaluates a value of 1 the carry bit propagates from the n^{th} to the m^{th} bit into the addition result. The probability of a carry propagation across the m^{th} bit essentially is equivalent to the possibility that operands of an executed instruction will activate the $LLPs$ at the 4^{th} stage, where mantissa addition is performed. In case of F equals to 0, paths form P_{SLP} are activated as there is no carry propagation across the m^{th} bit and the effective computation time is maximum of the two delays: one from the 0 to m^{th} bit and the other from m^{th} to 51^{st} bit. In the above equation, there is a trade-off between m and n, area/power and accuracy of $F(m,n)$. More bits are being monitored, with a smaller excitation probability of the $LLPs$ (see RCA example in Section 2.2). However, a higher number of monitored bits is accompanied by an increase of the $LLPU$ area and higher power and throughput overheads. Based on DTA with different sets of m, n and the extracted operands from various applications, we found that the monitoring of 7 bits (i.e. bit 34 to 40) for each operand provides a balanced choice between prediction accuracy and overheads. Finally, one of the aspects of the $LLPU$ that requires special mention is the use of a negative edge triggered D flip-flop, as the $LLPU$ needs to remember the nature of input latency in the CC_{i-1} in order to generate the correct enable value in the CC_i. Additionally, this negative edge flip-flop along with a multiplexer allow computation of the enable signal before the next set of inputs (at the next positive edge of the clock), ensuring that in case of the $LLPs$ sensitization at Stage 4 the stall will occur at the expected next arising edge of the clock.

4. EVALUATION RESULTS

In this Section, we evaluate the efficacy of the redesigned FPU in addressing delay variations and estimate the area, power and throughput overheads compared to the original (unmodified) FPU design without any guardband. We also

estimate any possible gains by exploiting the available timing slacks in our design compared to an FPU with a guardband based scheme under different process corners.

4.1 Application Profiling

To estimate the efficacy of our approach with realistic FP operands, we modified a profiling tool [12] to extract program traces from various compute intensive applications. Such a tool essentially interrupts the execution of a running program with a defined period and collects the values held in registers. In our analysis, we use the `Kmeans`, `CFD` and `Heartwall` benchmarks from the Rodinia suite; the `Raytrace` benchmark from the Parsec suite; a face detection application based on the OpenCV library and a synthetic benchmark that generates random double-precision FP operands. This set of benchmarks represents a variety of algorithms that have many floating-point operations and covers a wide range of domains, i.e. Data Mining, Fluid Dynamics, Medical Imaging, Computer Graphics and Computer Vision. To obtain the program traces, we profile all benchmarks on an `ARM A7` based system, Odroid-Xu3. Figure 7a depicts the percentage of time spent on execution of different types of instructions averaged over all profiled benchmarks. We see that benchmarks spent 31.2% of the total time on execution of FP instructions on average, which indicates their importance. Using such a tool, we extract 10.000 operands from the most frequently executed FP instructions for each application, which we feed to the DTA tool to estimate the BER and the consumed power.

4.2 Evaluation of BER and Quality

BER: We quantify the efficacy of our approach in minimizing the timing failures by estimating the output BER, which depends on the excitation of paths by input operands and an assumed worst-case delay variation. In our experiments, we execute the proposed and the original FPU, implemented in the typical corner (@1.1V), under a delay variation (i.e 8%) imposed by the slow corner cell library (@0.95V), which is representative of potential worst-case delay increase [9]. Note that the original design is optimized using typical options at the synthesis and PnR phases, while the proposed FPU is implemented using as a constraint the best CP_{STA} achieved by the original FPU in the typical corner.

Figure 6 shows the BER in the sign, mantissa and exponent bits of the the original and the proposed FPUs across the 6 benchmarks. We estimate the BER by comparing the simulated output for specific operands and the error-free output D_{gold}. We make several interesting observations on that Figure. First, distinct bit positions incur different BERs under the same or different input dataset. This occurs because different input operands activate different paths, which eventually enable different output bits. Second, the original design exhibits much higher BERs compared to our design, ranging from 1.15% up to 66.9% in the mantissa and up to 37.6% in the sign and the exponent. The exponent bits and the MSBs of the mantissa in the original design exhibit increased BERs which may result in catastrophic quality loss. In contrast to the original design, the proposed FPU (only with the PS and without the DCA activated) incurs significantly lower BERs across the vast majority of the bits, especially in the exponent and the MSBs of mantissa. Specifically, in our design, the BER of the exponent and the 10 MSBs of mantissa across all the benchmarks ranges from 0% to 16.3%. Overall, under the assumed delay increase, the applied PS in the proposed design reduces BERs by 2.71× on average compared to the original FPU, while the activation of the DCA in our design (shown as PS+DCA) eliminates any error.

Output Quality: To evaluate the quality loss incurred by the timing failures, we estimate the SNR of our design and compare it with the SNR of the original FPU in case of

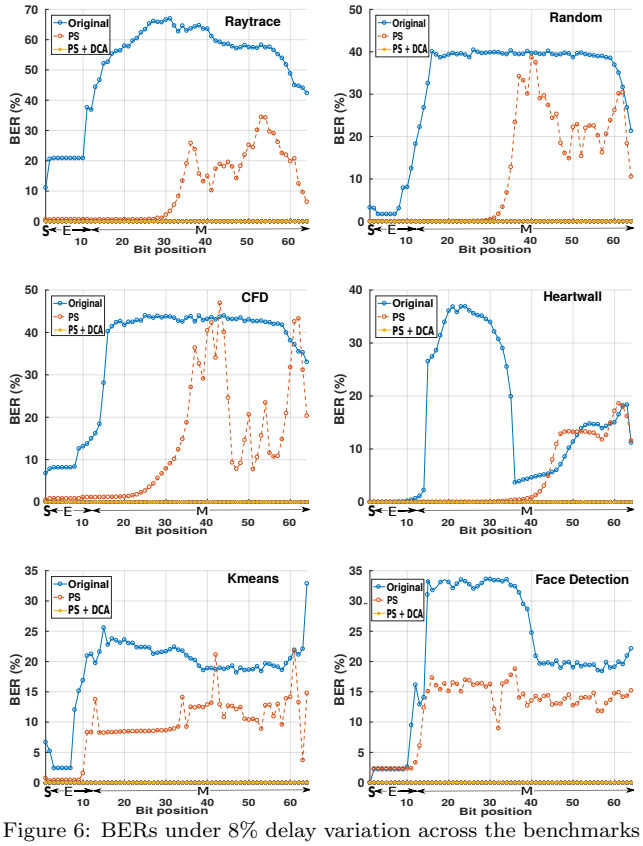

Figure 6: BERs under 8% delay variation across the benchmarks

all applications. SNR compares the level of a desired signal to the level of the noise incurred by timing errors. In these experiments, the desired signal is the D_{gold}, while the noise is estimated by calculating the mean-square error between D_{gold} and the system output of the original and the proposed FPUs under the assumed worst-case delay increase. The output SNR in case of the original design and the proposed design with the DCA activated (PS+DCA) or de-activated (PS) is depicted in Figure 7b. We observe that the proposed FPU with the desired PS results in higher SNR levels compared to the original FPU. As we discussed, the original design exhibits high BERs especially in the exponent and the MSBs of mantissa and as consequence low or even negative SNR levels. Nonetheless, the proposed FPU with the applied PS achieves on average 142.3 db higher SNR than the original FPU, while the proposed approach (PS+DCA) eliminates the noise incurred by the timing errors

4.3 Throughput, Area and Power Penalty

As we discussed the elimination of the timing failures and the limited quality loss achieved by our design comes at

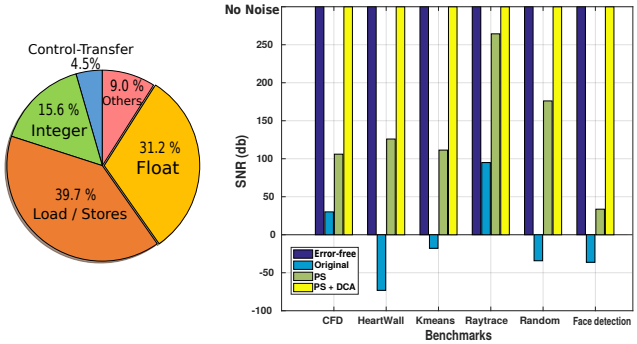

Figure 7: (a) Average instruction distribution and (b) SNR levels across the considered benchmarks

a cost which we estimate in this section. To start with, the $LLPU$ introduces a throughput overhead by stalling the pipeline for an extra cycle when the excitation of the $LLPs$ is detected. To evaluate this overhead, we compare the throughput of our design (PS+DCA) with the throughput of the original FPU under the assumed 8% worst-case delay increase. Initially, we estimated the number of two-cycle operations triggered by 60000 operands (10000 for each application), which are extracted from the profiled applications. As shown in Table 1, the rare excitation of the $LLPs$ and thus the two-cycle operations in the proposed design renders the throughput overhead negligible, ranging from 0.08% up-to 0.61%.

An essential attribute that led to such low overheads in our design is the applied PS, which limited the number of the $LLPs$ and constrained them only to a single stage. Without the PS, the DCA and the $LLPU$ would have to be applied to every stage of the pipeline to monitor and detect any activation of the error-prone $LLPs$, which as shown in Figure 5a are prevalent and are excited very often as captured by the resulted BERs of the original FPU in Figure 6.

Finally, we perform dynamic power analysis utilizing the Voltus tool from Cadence which uses as inputs: the post placed and routed netlist, the VCD file, a design exchange format (DEF) file that represents the physical layout and a standard parasitic exchange format (SPEF) file which corresponds to the parasitic data of wires in a chip. Using the Cadence tools, we also measured the area and the average power overheads of the proposed design (implemented in the typical corner and operated at 1.1V). The results show that the proposed FPU incurs $\sim$ 0.3% area and $\sim$ 7.1% power overheads compared to the original one.

4.4 Comparison with a Guardband based Scheme

In this paragraph we compare the proposed design with the conventional guardband based paradigm. In particular, according to the conventional paradigm, the original FPU adopts enough timing guardbands by up-scaling the voltage to avoid timing failures and obtain an error-free output. Using the available fast corner cell library (@1.25V) and the extracted VCD files (allowing to consider the switching activity and dynamic path activation), we estimated the power consumption across all applications as shown in Table 1. Operation at such a voltage may provide the necessary timing margin to avoid all the dynamic timing failures and lead to an error-free output (shown in Figure 7b) across all applications. However, it comes at a cost of up-to 43.1% power overhead when compared to our design with the PS and DCA activated which as shown in Figure 7b leads also to an error-free output. Finally, under iso-quality, the proposed design can lead to 37.95% power savings on average when compared to the guardband based design.

4.5 Discussion on Potential Power Gains

We propose a variation-aware approach that facilitates the positive timing slacks of the $SLPs$ and exploits the rare activation of the $LLPs$ through occasional two-cycle operations. Conversely, such properties could also be utilized to allow operation at a reduced voltage, when the manufactured chip is unaffected by variations. In this case, our design can tolerate operation at 0.95V, leading to 27.8% average power savings compared to the operation at the nominal supply voltage of 1.1V. In other words, our approach can be used not only for mitigating timing failures at low cost, but also for enabling operation at a reduced voltage.

Table 1: Throughput loss and power savings across all benchmarks in the proposed design

Benchmarks	Throughput loss (%)	Power savings (%)
Kmeans	0.09	38.1
CFD	0.61	39.02
Heartwall	0.19	30.65
Raytrace	0.14	43.1
Face Detection	0.08	42.22
Random	0.17	34.63

5. CONCLUSION

In this paper, we presented a framework for avoiding variation-induced timing failures in pipelined designs by i) lowering $|P_{LLP}|$ and ii) exploiting the rare activation of the $LLPs$. We achieved this by modifying the path distribution to reduce the $LLPs$ and constrain such paths to a single pipeline stage. Such a path distribution facilitates the adoption of a prediction block that detects the $LLPs$ excitation and provides an extra clock cycle for the completion of these paths when they are triggered. The evaluation of our approach with the developed DTA tool, using program traces extracted from a real processor, shows that the proposed design eliminates timing failures/BERs under potential delay variations with very low overheads. When compared to the conventional guardband approach, our design saves 37.95% power on average across the considered benchmarks. Overall, our study indicates that in order to address delay variations, there is a need to redesign the targeted design aiming at reducing the $LLPs$ rather than merely optimizing the performance. Path shaping can help to limit the number of the $LLPs$ which are the main source of overheads in existing schemes and facilitates the use of any other correction or prediction based technique at low cost. Even though we demonstrated the efficacy of the proposed approach by applying it to an FPU, the presented steps can be applied to redesign the stages of any other pipelined core.

ACKNOWLEDGEMENTS

The presented work is partially supported by the European Commission Horizon 2020 programme under grant no. 688540 (UniServer) and grant no. 732631 (OPRECOMP).

REFERENCES

[1] IEEE 754-2008 Standard for Floating-Point Arithmetic.
[2] NanGate FreePDK45, http://nangate.com.
[3] OpenRISC, "OpenRISC 1000 architecture manual".
[4] C. S. Amin *et al.* Statistical static timing analysis: how simple can we get? In *DAC*, pages 652–657, 2005.
[5] S. Borkar *et al.* Parameter variations and impact on circuits and microarchitecture. In *DAC*, 2003.
[6] D. Bull *et al.* A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation. *JSSC*, Jan 2011.
[7] J. Constantin *et al.* Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment. *DATE*, 2015.
[8] S. Ghosh *et al.* Voltage scalable high-speed robust hybrid arithmetic units using adaptive clocking. *TVLSI*, 2010.
[9] P. Gupta *et al.* Underdesigned and opportunistic computing in presence of hardware variability. *TCAD*, 2013.
[10] A. B. Kahng *et al.* Slack redistribution for graceful degradation under voltage overscaling. In *ASP-DAC*, 2010.
[11] X. Liang *et.al.* Revival: A variation-tolerant architecture using voltage interpolation and variable latency. *IEEE Micro*, 2009.
[12] L. Mukhanov *et al.* Alea: Fine-grain energy profiling with basic block sampling. In *PACT*, 2015.
[13] J. M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Second Edition, 2003.
[14] A. Rahimi *et al.* A variability-aware openmp environment for efficient execution of accuracy-configurable computation on shared-fpu processor clusters. In *CODES+ISSS*, 2013.
[15] A. Rahimi *et al.* Application-adaptive guardbanding to mitigate static and dynamic variability. *Trans. on Computer*, 2014.
[16] Y. Zhang *et al.* irazor: Current-based error detection and correction scheme for pvt variation in 40-nm arm cortex-r4 processor. *JSSC*, 2018.