Adaptive Operation-Based ALU and FPU Clocking

Athanassios Tziouvaras
Dept. of ECE
University of Thessaly
Volos, Greece
attziouv@uth.gr

Georgios Dimitriou
Dept. of CS
University of Thessaly
Lamia, Greece
dimitriu@uth.gr

Michael Dossis
Dept. of CS
University of W.Macedonia
Kastoria, Greece
mdossis@uowm.gr

Georgios Stamoulis
Dept. of ECE
University of Thessaly
Volos, Greece
georges@uth.gr

***Abstract*—The operating clock period in modern circuits is designated under worst-case operating conditions, in order to ensure error free functionality. To accomplish this goal, designers take into account the timing of the circuit critical path that provides an upper limit for the clock rate. However, that limit imposes heavy performance penalties on the design, since the critical path is not frequently excited during runtime. In contrast with the prevailing methodology, the better-than-worst-case (BTWC) paradigm treats the circuit timing requirements in a more flexible way, as it does not commit to serve the demands of the worst-case scenario.**

In this work we develop a novel timing analysis methodology inspired by the BTWC approach. Instead of performing the standard critical path analysis, we focus on analyzing the timing requirements of each operation separately. In the sequel we specify an adaptive clock period that is derived by the timing requirements of each supported circuit operation. As a result, we are guaranteed error-free circuit functionality, as no timing violations of individual operations are ever allowed to occur. To this end no error correction mechanisms are required.

We verify the proposed methodology by applying it on four different post-layout implementations of ALUs and FPUs. The results we obtain display an average 2.05x throughput increase compared to the implementations operating under the worst-case clock period. We also demonstrate that the area requirements of our methodology are trivial as the overhead is less than 2% of the original design.

Keywords— BTWC, adaptive clock scaling, timing analysis, post-layout implementation.

I. Introduction

Modern integrated circuits are designed to operate under the worst-case scenario as far as timing is considered. In order to cover all possible timing violations designers employ the Static Timing Analysis (STA) methodology, which calculates the circuit's critical path but also induces pessimism into the calculation. As a result, the circuit's critical path acts as a threshold for the specification of the clock period, thus hampering the performance of the design.

On the other hand, the better-than-worst-case (BTWC) design paradigm employs a different approach. BTWC designs often operate at lower-than-critical clock periods, allowing timing errors to occur. Such errors are then detected and either corrected or the circuit is restored to a previously valid state. This approach has proven to be much more efficient in terms of performance, as the circuit operates under higher clock frequencies.

In this work we propose a novel timing analysis approach based on the BTWC design paradigm. Our approach differs from standard BTWC techniques, in that we do not allow timing errors to occur. More specifically, we utilize a technique that obtains the necessary timing information for each individual circuit operation, instead of obtaining the timing information about the critical path for all operations. As a result, we are able to adaptively scale the clock frequency for each corresponding operation, while also guaranteeing error-free execution.

The rest of this paper is organized as follows. Section II contains a short research review on the topics of our research, whereas Section III discusses the proposed timing analysis technique. The experiments conducted, as well as the hardware implementation of the proposed methodology, are described in Section IV. Finally, Section V gives the conclusion of our work.

II. Related Work

Previous research on the BTWC design paradigm has led to the development of the Razor microprocessor [1]. Razor is a fully speculative general purpose processor, which allows timing errors to occur, as its pipeline is clocked at higher-than-critical frequencies. Razor utilizes an error detection mechanism, which detects timing errors in real-time and corrects them. Another work in [2] focuses on both clock scaling and guard banding on specific instruction sequences. In that study, researchers manage to analyze the timing requirements of upcoming instruction sequences and properly scale the voltage levels and clock frequency, in order to speed up the execution time. Both studies in [1] and [2] employ a BTWC design technique known as timing speculation (TS). TS designs efficiently deal with performance/energy tradeoffs, as they operate in lower voltage thresholds compared to traditional processor designs [3][4]. As previous work shows in [5], TS techniques can be applied on larger systems, such as superscalar processors, as long as error correction mechanisms are properly employed. Research in [6] concludes that such mechanisms are frequently the source of metastable behavior, a phenomenon that produces errors on the designs, hampering their functionality. Although researchers are employing various strategies to eliminate metastable behavior from TS designs [7], some argue that these issues are yet to be properly addressed [6]. An earlier work of ours on timing analysis for BTWC design is only focused on timing analysis methodologies for the processor instruction pipeline and is not addressing the clocking issue for any part of the processor [8].

III. BTWC Timing Methodology and Validation

The BTWC methodology we propose aims at eliminating any error correction penalties imposed by false timing prediction events. To this end, we utilize a proactive mechanism, which predicts any timing errors and properly adjusts the core clock frequency before such errors occur. In order to successfully accomplish the aforementioned task, we need to extract the circuit's timing information. Figure 1 displays a diagram of the timing analysis approach we adopt, in order to obtain the necessary timing requirements of the design. As our methodology is heavily dependent on the system's architecture, a post-layout netlist along with any

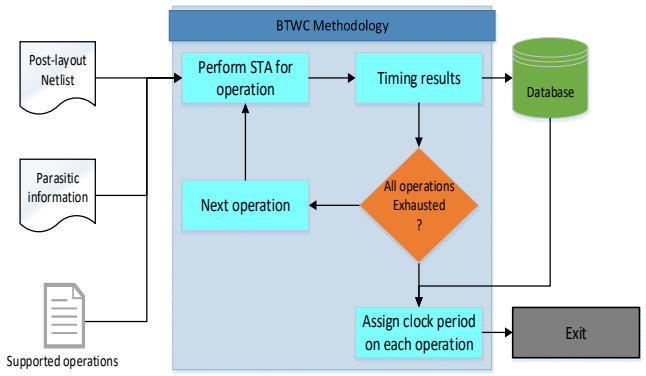

Fig. 1. The proposed timing analysis technique for BTWC designs.

corresponding parasitic information is required. In this work we have implemented four different designs of ALUs and FPUs, and thus we require the knowledge of the operations supported by those designs.

Instead of opting for an STA approach that would calculate the circuit's critical path and would designate the worst-case clock period, we utilize and extend the approach of our previous work presented in [8]. As such we perform STA for each individual operation supported by the design. Below we present the BTWC timing analysis methodology steps we propose:

1) Acquire the post-layout netlist and parasitic information of the circuit as well as the operations supported by the design.

2) Set the operation's input bits at fixed voltage values in order to represent the next operation code.

3) Perform STA in respect to the fixed voltage values set in the previous step.

4) Get the timing results of the current STA and save them for later use.

5) If every supported operation is analyzed and the corresponding timing information is collected, go to step 6. Otherwise go to step 2.

6) Utilize the timing information obtained by step 4 and assign a minimum clock period to each supported operation. The assigned clock period should satisfy the timing requirements of the corresponding operation, according to the STA analysis, so that no timing violations should occur.

7) Finish the analysis.

By performing the analysis described above, we obtain a minimum clock period for each operation's timing path, instead of obtaining a minimum clock period for the whole integrated circuit. We are also guaranteed no timing errors will occur, as the STA criteria are satisfied for each operation separately. As a result, we can utilize such information to adaptively scale the clock frequency according to the particular operation the FPU/ALU is executing. We do not care if the STA criteria are violated for other parts of the circuit, as long as those parts are not used for, i.e., are not in the path of that operation.

We validate the BTWC methodology proposed above on four different 32-bit post-layout implementations of an ALU and FPU. More specifically, we design two single cycle execution units, which we name Baseline designs and two multi-cycle pipelined units with stricter timing, which we name Pipelined implementations. Table I below displays the configuration options of both ALU and FPU designs, as well as the timing requirements of each supported operation. We classify the supported operations into the following categories:

- *Arithmetic class,* which includes the addition and subtraction operation.
- *Multiplication class,* which includes the multiplication operation.
- *Division class,* which includes the division operation.
- *Shift class,* which includes any shift operation such as shift left logical, shift right arithmetic, etc.
- *Logical class,* which includes any logical operation such as AND, NOR, XOR etc.
- *Comparison class,* which includes any comparison operation such as equality, less than, etc.
- *Conversion class,* which includes FP-to-INT and INT-to-FP conversion operations.

Each physical implementation is assigned a typical clock period that depends on the timing requirements of the operation with the highest slack.

IV. Implementation and Experimental Results

Each one of the four physical designs introduced in the previous section is implemented with Synopsis IC and NaNGate 45 nm standard cell library, while the post-layout timing analysis is performed via the Synopsis Primetime tool. Figure 2 below depicts the architecture diagram of the proposed BTWC methodology on the post-layout implementation of a functional unit. The pipelined ALU/FPU utilizes the following inputs:

- *Operands A and B*, each of 32-bit length.

TABLE I. Timing Requirements of the Post-Layout Implementations.

Supported Operation Class	**ALU implementation**	
	Baseline	***Pipelined***
Arithmetic	3 ns	3 ns
Multiplication	5 ns	2.5 ns (2 stages)
Division	15 ns	2.5 ns (6 stages)
Shift	2 ns	2 ns
Logical	1.2 ns	1,2 ns
Comparison	1.7 ns	1.7 ns
Typical clock period	**15 ns**	**3 ns**
Supported Operation Class	**FPU implementation**	
	Baseline	***Pipelined***
Arithmetic	4.6 ns	4.6 ns
Multiplication	6.8 ns	3.5 ns (2 stages)
Division	18 ns	3 ns (6 stages)
Comparison	4.2 ns	4.2 ns
Conversion	3.8 ns	3.8 ns
Typical clock period	**18 ns**	**4.6 ns**

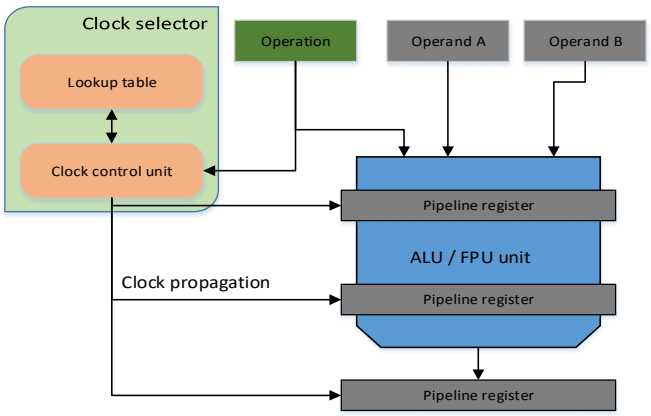

Fig. 2. Architecture diagram of the BTWC design implementation.

- *Operation,* which designates the current ALU/FPU function.
- *Clock signal,* which synchronizes the pipeline registers.

We design a clock selector module capable of scaling the clock frequency, according to the timing requirements of the current ALU/FPU operation. To accomplish this, we utilize lookup tables, which contain the critical timing requirements of each supported operation. Thus, when the functional unit activates, the clock control unit utilizes the operation code input and accesses the lookup table structure. In the sequel, the lookup table returns the timing requirements of the current operation and the clock selector designates the corresponding clock period. As a result, the clock frequency of the unit is adapted according to the timing needs of the current operation.

In order to efficiently change the clock frequency during runtime we develop and implement a clock control unit.This unit takes input from multiple PLLs that operate under different periods and utilizes the lookup table information to select the proper PLL according to the current ALU/FPU operation. We adopt such an approach in order to avoid the adaptive frequency scaling of one single PLL due to the limitations it imposes on the design. As a result, we utilize a set of predefined PLLs and we proceed in dynamically selecting one of them according to each operation's timing requirements.

To evaluate our methodology, we execute 6 million ALU/FPU operations on each design. Figure 3 below depicts a post-layout execution instance using the ModelSim tool on the ALU. We observe the core clock signal, which is the output of the clock selector module (located at the the upper side of the waveform), changing its operating frequency according to the operation that is currently executed. We also measure the throughput improvement we obtain, compared to the typical clock execution paradigm. In Figure 4, we depict the normalized throughput gain over the original implementations. We make two observations based on the acquired results:

- The average throughput improvement over all implementations is 105% (or 2.05x the original throughput). If we exclude the division operation, improvement falls to 60%, which is still significant.
- Designs with looser timing, such as the Baseline ALU, display larger throughput increase, while designs with stricter timing display smaller gains. Nonetheless, even when stricter timing is considered as in pipelined FPU, we managed to obtain a 30% speedup compared to the original implementation.

As we dynamically scale the clock frequency in order to meet each operation's timing requirements, we expect a power consumption increase, compared to designs that operate on typical clock period. In Figure 5, we display the power increase observed when applying the BTWC methodology. The power results obtained indicate the following:

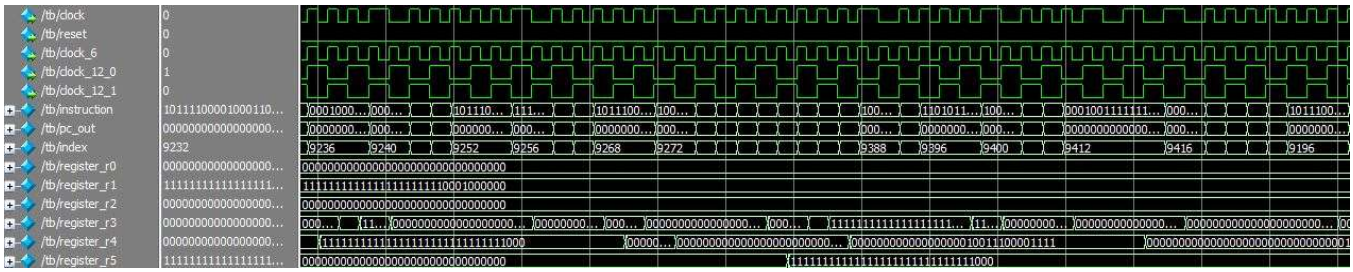

Fig. 3. A post-layout execution instance of the ALU using ModelSim.

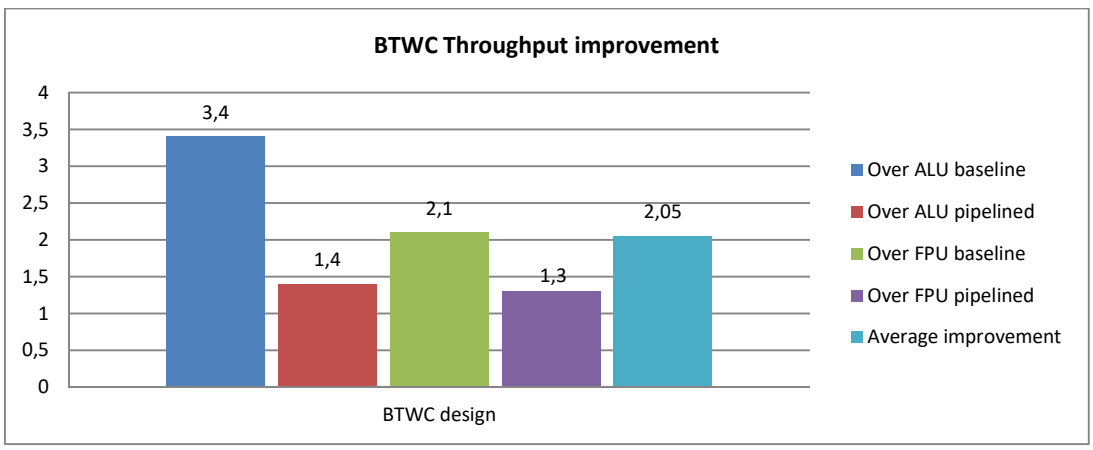

Fig. 4. Normalized throughput improvement of the BTWC design.

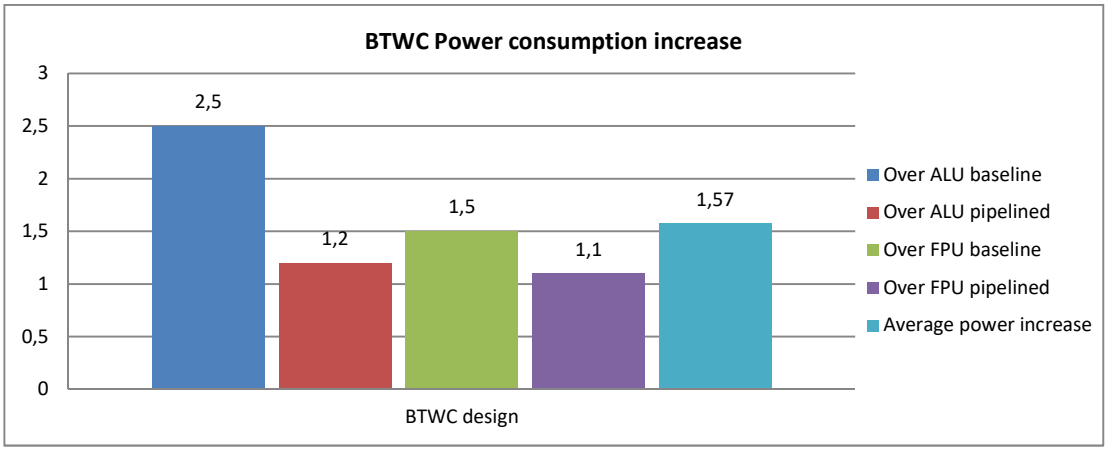

Fig. 5. Normalized power consumption of the BTWC design.

- The average power consumption increase over all implementations is 57% (1.57x), which becomes 32% if we exclude the division operation. Such an increase is unavoidable, since higher clock frequencies significantly affect the power requirements of the design.
- The more opportunities a design offers for clock scaling, the more its power consumption will increase. As a result, the ALU baseline's power increase is significantly higher when compared to the pipelined FPU implementation.

In order to calculate the area requirements of the clock selector unit, we utilize the IC compiler to extract the area requirements of the integrated circuit. Table II below displays such information of each individual implementation. We observe that the area overhead for our methodology is less than 2% of the total area required for each ALU/FPU design. Finally regarding the power consumption of the clock selector module we measure less than 1% contribution to the total power dissipation of the design.

V. Conclusion

In this paper we propose a novel timing analysis approach based on the BTWC premise. We apply our methodology to an adaptive operation-based clocking on post-layout netlists of ALU and FPU designs. We consider the following contributions of our work:

- *Timing analysis of supported operations.* By analyzing the timing requirements of each operation separately, we obtain knowledge on the clock period that should be assigned on each individual function.
- *Error-free execution.* As the timing requirements of every operation are proactively met, we scale the clock frequency according to such constrains. As a result, we make sure that no timing errors will occur during runtime.
- *No error correction is required.* The absence of runtime timing errors relieves the need for complex error correction mechanisms that would increase the design complexity and would impose performance penalties.

TABLE II. Area Requirements of Each Implementation.

	Integrated circuit area	Clock selector unit Area overhead
ALU Baseline	46 um^2	1 um^2
ALU Pipelined	50 um^2	
FPU Baseline	71 um^2	1 um^2
FPU Pipelined	78 um^2	

The results we obtain indicate a significant speedup along with a power consumption increase due to the clock functioning at higher frequencies. Nonetheless, we have managed to achieve an average 1.3x performance-to-power ratio improvement that demonstrates a beneficial performance/power tradeoff. The overall area requirements of the methodology are also proven to be trivial, requiring only about 2% of the total design area.

References

[1] D. Ernst et al., "Razor: a low-power pipeline based on circuit-level timing speculation," Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36., San Diego, CA, USA, 2003, pp. 7-18.

[2] A. Rahimi, L. Benini and R. K. Gupta, "Application-Adaptive Guardbanding to Mitigate Static and Dynamic Variability," in IEEE Trans. on Computers, vol. 63, no. 9, pp. 2160-2173, Sept. 2014.

[3] R. Ye, F. Yuan, J. Zhang and Q. Xu, "On the premises and prospects of timing speculation," 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, 2015, pp. 605-608.

[4] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, "Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis," in Proc. of the 37th Annual International Symposium on Computer Architecture (ISCA), June 2010.

[5] V. Subramanian, M. Bezdek, N. D. Avirneni, and A. Somani, "Superscalar processor performance enhancement through reliable dynamic clock frequency tuning," in Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 2007.

[6] S. Beer, M. Cannizzaro, J. Cortadella, R. Ginosar, and L. Lavagno, "Metastability in better-than-worst-case designs," in Proc. of the 20th IEEE International Symposium on Asynchronous Circuits and Systems, May 2014.

[7] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S.-L. L. Lu, T. Karnik, and V. K. De, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," IEEE Journal of Solid-State Circuits, vol. 44, pp. 49–63, Jan. 2009.

[8] A. Tziouvaras, G. Dimitriou, M. Dossis and G. Stamoulis, "Instruction-Based Timing Analysis in Pipelined Processors," 2019 4th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Piraeus, Greece, 2019, pp. 1-6.