Application Performance Improvement By Exploiting Process Variability On FPGA Devices

Konstantinos Maragos, George Lentaris, Dimitrios Soudris
School of ECE
National Technical University of Athens
{komaragos, glentaris, ksiop, dsoudris}@microlab.ntua.gr

Kostas Siozios
Department of Physics
Aristotle Univ. of Thessaloniki
ksiop@auth.gr

Vasilis F. Pavlidis
School of Computer Science
The Univ. of Manchester
pavlidis@cs.man.ac.uk

***Abstract*—Process variability is known to be increasing with technology scaling in IC fabrication, thereby degrading the overall performance of the manufactured devices. The current paper focuses on the variability effect in FPGAs and the possibility to boost the performance of each device at run-time, after fabrication, based on the individual characteristics of this device. First, we develop a sensing infrastructure involving a wide network of customized ring oscillators to measure intra-chip and inter-chip variability in 28nm FPGAs, i.e., in eight Xilinx Zynq XC7Z020T-1CSG324 devices. Second, we develop a closed-loop framework based on dynamic reconfiguration of clock tiles, I/O data sniffing, HW/SW communication, and verification with test vectors, to dynamically increase the operating frequency in Zynq while preserving its correctness. Our results show intra-chip variability in the area of 5.2% to 7.7% and inter-chip variability up to 17%. Our framework improves the performance of example FIR designs by up to 90.3% compared to the SW tool reports and shows speed difference among devices by up to 12.4%.**

I. Introduction

The ever-growing advancements in technology, demand computing platforms with increased performance and decreased power consumption. To follow this aggressive trend, researchers and engineers strive to decrease the size of the transistors in the ICs. However, the transistors approach their physical and power-thermal boundaries challenging the Moore's law [1] and lowering the yield during chip fabrication due to process variations [2]. Thus, today, the semiconductor companies increase the manufacturing cost and invest in new methods to cope with down-scaling challenges and, specifically, with process variability [3], [4], [5].

Process variability stems from both stochastic and systematic variations, which are distinguished as *intrinsic* and *extrinsic* [6]. The intrinsic variation refers to the differences among identical chips due to the metal-gate work-function fluctuation, random-dopant fluctuation, line-edge roughness, gate oxide thickness variation, etc. The extrinsic variation is due to various operation shifts of the fabrication equipment, such as the rapid thermal annealing, the photoresist development, the etching, etc. All these effects have a significant impact on the performance and reliability of the manufactured circuits, especially when the fabrication process reaches down to almost atomic-level processing. As a result, supposedly identical chips exhibit significantly different electrical characteristics, e.g., maximum operation frequency and power efficiency. Moreover, variability can occur even within the same device itself (*intra*-chip). For example, the total failure probability is around 3% for SRAM writes at 32nm and grows beyond 10% for 16nm technology [7].

Among all computing platforms affected by variability issues, the modern Field-Programmable-Gate-Arrays (FPGA) are of particular interest for two major reasons. First, most often, an FPGA device is programmed to operate at the maximum frequency as estimated by a SW synthesis/implementation tool, which however considers only the general characteristics of the device family (set of supposedly identical devices) and not the specifics of the device in hand (its variability). This conventional open-loop approach introduces a safety margin in the operating frequency that translates, however, to a considerable loss of performance. Second, the FPGA allows for fine-grain programming/reconfiguration –almost at the gate level– on a homogeneous architecture consisting of a plethora of identical resources uniformly placed over a large silicon area. That is to say, after fabrication, the most efficient resources of an FPGA can be selected according to the intra-chip variability of the device and relocate the implemented function within the FPGA to improve performance. This latter capability makes the FPGA a very appealing platform to investigate variability at the programming/user level, especially when compared to multi-core CPUs and GPUs that support only coarse-grain spatial relocation of the implemented functions (task migration) [8].

This paper proposes an automated framework to explore and exploit both the intra- and inter-chip variability of FPGAs seeking to improve the performance of any given function on any given device. More specifically, we develop a closed-loop series of steps involving the SW synthesis/implementation tool, the device in hand (without any extra equipment), and the target function, to achieve the highest operating frequency possible. To this end, our main contributions are:

- The investigation of process variability in 28nm FPGA devices. We use a set of eight identical Xilinx XC7Z020T-1CSG324 FPGA devices and we develop a custom VHDL sensing infrastructure involving multiple ring oscillators (RO) to evaluate the behavior of more than 400 areas on each device. To the best of our knowledge, this is the first actual investigation of variability in the literature for such an advanced technology node in commercial FPGAs.
- The development of a closed-loop framework that ex-

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

ploits the benefits of SoC FPGAs (can be applied for other FPGAs as well), the existing variability in the chips, and the limitations of the available static timing analysis by SW tools to improve the device performance at run-time while preserving the correctness of its operation.

As a proof-of-concept design, we use highly-parallelized finite impulse response filters (FIR) and Xilinx Zynq boards to demonstrate a speed improvement of up to 90% compared to the max. frequency estimated by the SW tool. Moreover, we show that inter-chip variability can result in up to $12\%-17\%$ performance difference among devices depending on the size of the implemented function (e.g., FIR or Ring Oscillator).

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 describes our sensing infrastructure and evaluation of variability in 28nm FPGA devices. Section 4 describes the proposed framework for performance improvement. Section 5 shows the experimental results and, finally, Section 6 concludes the paper.

II. Related Work

FPGA variability and the related effects have been investigated by the FPGA research community for longer than a decade. In [9], the *within-die* delay variability was studied for 90nm FPGA devices. The authors developed a measurement circuitry based on an array of 34x26 ring oscillators to characterize the systematic and stochastic delay variability in the logic elements of the FPGA. They tested 18 Cyclone II 2C35F672C6 devices. The stochastic variation was measured up to 4.47%, while the variation produced by systematic sources across a single die reached 3.66%. Within-die delay variation was also analyzed in [10] for 65nm Xilinx Virtex-5 5VLX330T FPGAs by placing 6480 ring oscillators. The total measured within-die variation was 22% (including both random and systematic variation). The systematic variation was also modeled and incorporated in the static timing analysis process to achieve 5.4% frequency improvement, on average, for a set of 11 test designs. In [11], the authors used 112 three-stage ring oscillators for online measurement of intra-die variations on delay, leakage, dynamic power, and temperature; their system was built on Microblaze for monitoring the ring oscillators on each of two Xilinx Virtex-5 XC5VLX110T FPGAs. The delay variation was measured at 7%

To mitigate process variations within the FPGA platforms, the community proposes approaches such as variation-aware placement and routing algorithms [12], or customized FPGA architectures that adaptively body-bias their resources (tested with HSPICE simulations at 130nm) [13].

Regarding the performance improvement in FPGAs, the authors in [14] show that applications implemented on FPGAs can actually operate at a higher operating frequency than the rated one, as estimated by the static time analysis. On this basis, they study the behavior of common arithmetic primitives in FPGA and perform a trade-off analysis between overclocking with timing violations and overclocking via internal bit truncation. Furthermore, in [15], the performance improvement in FPGAs has been studied by performing dynamic voltage and frequency scaling based on in-situ detectors for monitoring the critical path violations. However, this approach introduced a considerable resource overhead when the number of the monitored design paths increases (e.g., 5% for 100 paths).

III. Custom Sensing Infrastructure and Evaluation Of Variability in $28nm$ FPGA

To investigate variability in 28nm FPGAs, we select one of the most popular FPGAs in the market, i.e., the Xilinx Zynq-7000, and we test eight identical XC7Z020T-1CSG324 SoC devices. Each SoC device includes the so-called Processing System (PS) with a dual-core ARM Cortex A9 and a variety of peripherals, as well as the so-called Programmable Logic (PL) with the actual FPGA resources under investigation (85K logic cells with 53K LUTs and 106K DFFs and 140 RAMB36 and 220 DSP slices). To analyze the performance of each device with increased granularity, we develop utilizing VHDL a self-deployed infrastructure consisting of a number of sensors placed systematically over the PL and a control function running on PS. The control function initializes the sensors, retrieves the measurement of local speed across the regions of PL, and communicates the results to a host PC for further processing. Our infrastructure is sufficiently generic to be applied to any modern FPGA device, hard/soft-core processor and any granularity (number of RO). For clarity, the following subsections provide the details of our infrastructure using Ring Oscillators as sensors and the Cortex-A9 as controller.

A. Ring Oscillator customized for multi-replication on FPGA

The fundamental building block of our sensing infrastructure is the well-established Ring Oscillator (RO), which we use to sense the speed of the local regions within the FPGA. In general, the RO consists of a chain of inverters forming an asynchronous loop as shown in the middle box of Fig. 1. When the number of gates is odd and the RO is fed with a constant signal, its output oscillates to provide a square wave that can be used as a clock signal. The frequency of this clock depends only on the delay of the RO's asynchronous path, i.e., on the number and electrical characteristics of the transistors composing the loop. Hence, if we guarantee that the gates and routing in this loop (the total path) remain relatively intact while we successively place the RO within the FPGA, then its frequency change depends only on the electrical characteristics of the local region, providing a metric of the intra-chip variability. Moreover, if we repeat the exact same test on distinct XC7Z020T devices, then we can evaluate the inter-chip variability of these devices.

In this work, we utilize this approach to construct the custom RO shown in Fig. 1. This three-stage RO is augmented with a 16-bit up-counter and I/O registers. The goal is to measure the frequency of the signal generated from the RO by using the signal as a clock to drive the up-counter. Upon initialization, the counter resets and the 1-bit input register holds an activation signal for a predefined time period T. During T, the RO oscillates and the counter increments continuously as if it was measuring the number of positive

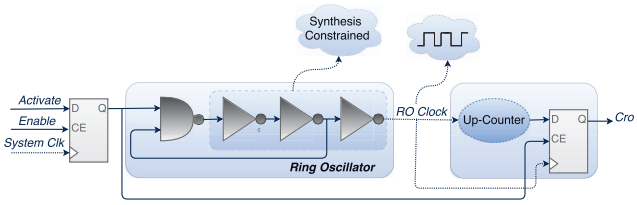
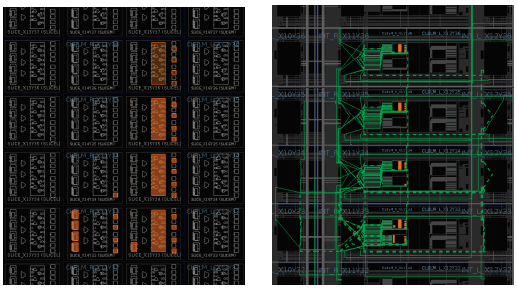


Fig. 1: Ring oscillator design on FPGA.



(a) Constrained logic resources (orange color)

(b) Constrained routing resources (green color)

Fig. 2: Floorplan of our Ring Oscillator on XC7Z020T FPGA.

edges of the generated signal. At the end of T, the 16-bit output register (synchronous to the RO signal) will hold the final outcome c_{ro} of the counter. Thus, the RO's frequency can be calculated externally having access to the I/O registers, as $f_{ro} = c_{ro}/T$.

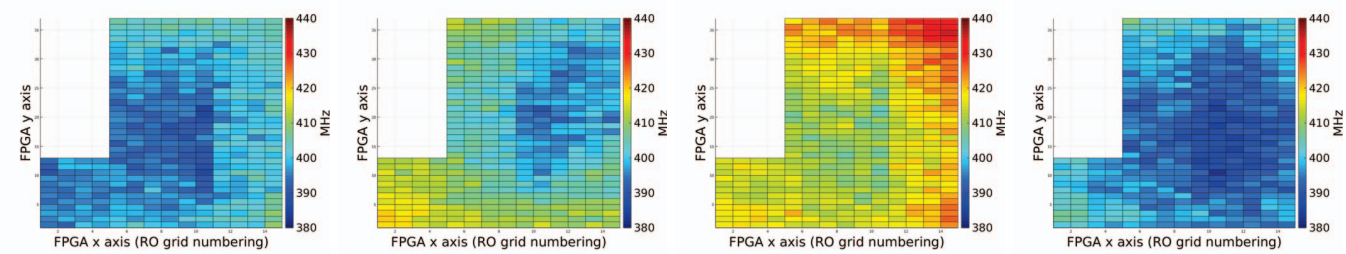
To increase the reliability of our sensing infrastructure, we must ensure that the f_{ro} depends only on the position of the RO in the FPGA and is not affected by the synthesis/implementation tool. Therefore, care should be placed in terms of VHDL, coding attributes, and tool constraints, to produce a robust RO structure replicated with the exact same resources and routing –almost– anywhere within the FPGA. This conclusion is drawn after experimenting with our sensor and the devices/tools, which verified that the operation of RO is highly sensitive to any environmental stimuli or design alteration; overall, according to our tests, we summarize that

- It is very important to isolate the RO from any surrounding logic to eliminate power noise and avoid local temperature increase. We observed a noticeable alteration of our results when mapping extra designs with high activity next to the RO (compared to standalone ROs).
- All ROs must be implemented with identical routing resources (relative to each other). Multiple runs verified that each wire connection can have a considerable impact on the RO timing when its routing is left unconstrained.
- All ROs must utilize identical logic resources (LUTs, Flip Flops and Carry Chains) with fixed placement on the fabric. Unconstrained placement leads to different routing and, hence, different timing (for Xilinx, even the SLICEs must be of the same type, i.e., $SLICEM$ or $SLICEL$, because each type of SLICE uses specific routing connections and type of LUTs).
- The functions implemented in LUTs must use the same LUT pins. Several experiments with isolated ROs and fixed routing/mapping/resources, but with varying LUT pins, resulted in considerably varying RO timing.

Based on these findings, we developed a custom built-in macro block for replicating a robust RO according to various constraints/attributes that prevent all unwanted phenomena/factors. First, as shown in Fig. 1, we use a synthesis constraint to prevent the SW tool from optimizing the chain of inverters, i.e., from removing the functionally redundant gates (for Xilinx tools, attributes $dont\_touch$, $keep$ and constraint $flatten\_hierarchy$). Second, we consider the exact architecture of the underlying FPGA to assign the RO to specific logic resources: each inverter is mapped to a distinct LUT followed by a pass-through latch, whereas the counter is mapped to four specific carry chain units together with 16 flip-flops (for Xilinx tools, constraints LOC, BEL). The four pass-through latches add additional extra delay to the signal transition between the stages of the RO chain (as in [11]). Fig. 2a, shows the resulting mapping of resources with a floorplan view of the FPGA fabric (highlighted in orange, small blocks represent DFF/latches, large blocks represent carry-chains, mid-sized blocks represent LUTs). Notice that the resources are selected to fit within an isolated local area of four adjacent CLBs (for Xilinx tools, constraint $create\_pblock$). Additionally, we use routing constraints to guarantee that the connections among the above resources will be identical, regardless of where the RO is placed on the FPGA, and that the counter's clock will be routed locally (for Xilinx tools, constraints $LOCK\_PIN, FIXED\_ROUTE$). Depending on the tool, we can also use a command (for Xilinx, message $severity$) to force the SW tool to implement the asynchronous combinatorial loop of the RO (otherwise considered as error). Fig. 2b, shows the resulting routing of our RO with nets (green) connecting the fixed RO resources shown in Fig. 2a.

B. Automatic Deployment and Integration of the Infrastructure

The robust RO macro-block described in the previous section is deployed in several copies and placed uniformly across the FPGA to cover the entire fabric. The deployment is automated with parametric VHDL code and a device-specific Python script, which introduces all of the necessary constraints to drive the implementation process (after synthesis). In particular, the script includes an RO template and information regarding the architectural details of the device (e.g., the size of the grid at slice level). The script inputs the necessary net names from the synthesizer output. Then, for each net in each RO, it will generate various constraints by replicating the RO template and increasing the X-Y location on the FPGA's grid. Specifically, for each RO, the script defines its eight SLICEs (using X-Y) and, subsequently, defines the routing and logic resources within and among the eight SLICEs (using fixed identifiers and following the RO template). For Xilinx Vivado tool, the script outputs a constraint file (*.xdc* of 30K lines) including numerous *LOC* for SLICEs, *BEL* for



(a) D764698. (b) D764187. (c) D657343. (d) D764694.

(e) D657830. (f) D764693. (g) D764710. (h) D447252.

Fig. 3: Variability maps of 8 distinct Zynq XC7Z020T-1CSG324 devices (common scale, represents max. frequency per cell).

LUTs and DFFs and latches and carry_chains, *LOCK_PIN* and *FIXED_ROUTE*. In total, for XC7Z020T, we deploy 408 ROs.

To initialize the network of ROs and retrieve their results, we connect all registers to a large multiplexing structure controlled by an AXI-lite port. The AXI-lite port communicates with the ARM CPU of Zynq, which sends a 32-bit command (data) to trigger the ROs (reset and activate), to stop the ROs after T, and finally, to read their results sequentially (the ARM processor sends 408 distinct words over AXI-lite, each one representing an address of the multiplexer and allowing a distinct RO to forward its result over AXI). To determine the measurement period T as accurately as possible, we use the ARM processor's *private* timer operating at 333 MHz; right before sending the activation command to the PL, ARM sets the timer, whereas after 10,000 cycles of the timer, ARM receives an interrupt and sends the deactivation command to the PL. We have estimated that the timing error with this procedure tends to be systematic and less than 0.7%. ARM executes the process bare-metal (without an OS) and communicates with a host PC via the serial port of Zynq's PS. The overall infrastructure is illustrated in Fig. 4.

C. Results: Variability Maps for 8 Zynq XC7Z020T Devices

The developed system was used to evaluate eight distinct Zedboards (seven PB200-248 Rev.D, one PB200-248 Rev.C) with Zynq XC7Z020T-1CSG324. The ambient temperature was 26°C during all tests and the junction temperature was measured at 35°C-38°C using the integrated system monitor (XADC block) of the chip. The results are illustrated in Fig. 3 as eight distinct variability maps, one per device, distinguished by their serial number given in the caption; each map represents the area of the chip and is divided in 408 small cells (top-left is the location of PS, not considered in this work), with each cell being colored according to the frequency of its

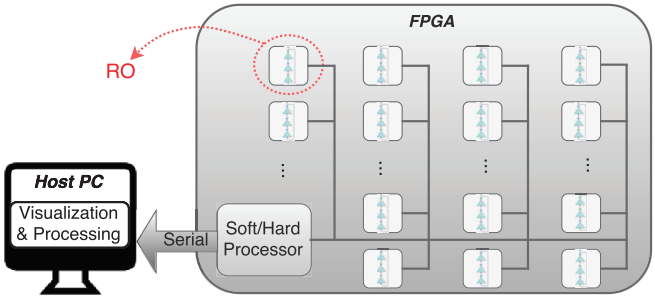


Fig. 4: Developed network of Ring Oscillators (RO) and CPU.

corresponding RO. The frequency of each RO is averaged over 10 runs (the standard deviation is negligible, i.e., 0.1 MHz). The color scale is common for all maps and ranges from 380 MHz to 440 MHz (device D447252 reaches up to 476 MHz on its Rev.C board). Overall, the frequencies vary rather smoothly in each map and reveal some homogeneous regions with respect to variability. Also, even with a limited sample of eight devices, a considerable variation in performance is observed among the devices, both on mean value and on the direction/distribution of their intra-chip variation.

Quantitatively, as in [10], let us define the intra-chip variability (%) of a device as $(max\{f_i\} - min\{f_i\})\ /min\{f_i\}$, where $\{f_i\}$ denotes the set of the 408 frequencies measured by the 408 ROs. Moreover, we define the inter-chip variability among all devices as $(max\{F_j\} - min\{F_j\})\ /min\{F_j\}$, where F_j denotes the maximum frequency measured in the j-th device, $j \in [1, 8]$. According to our results, the intra-chip variability ranges from 5.2% for device D447252 to 7.7% for device D764187. The inter-chip variability is 7.4% and increases to 17% when we include D447252. Moreover, when we compare $max\{F_j\}$ to the minimum f_i of all $408{\cdot}8$ ROs, we

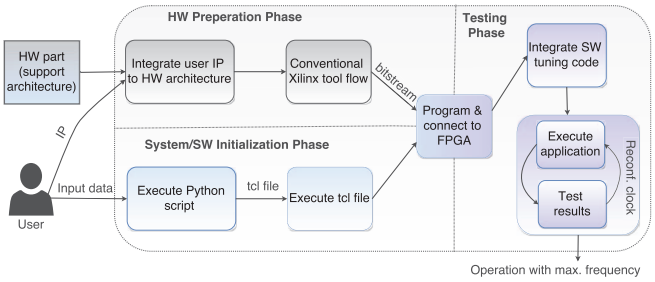


Fig. 5: The proposed framework: flow with HW and SW parts.

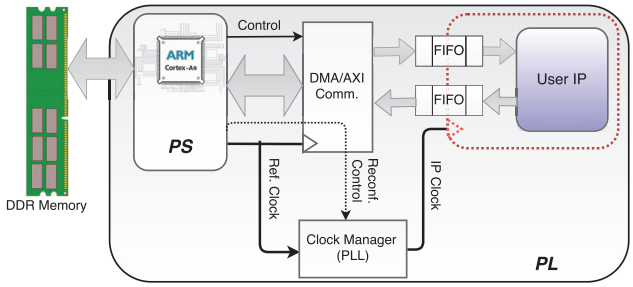


Fig. 6: HW architecture supporting the proposed framework.

get a variability of 24.4%. Such results provide the motivation to develop a framework exploiting the variability to improve the performance of a real-world design.

IV. Proposed Framework for Performance Boost

To exploit the chip variability described in the previous section, we develop a closed-loop framework that considers actual measurements from the device in hand while running a real-world HW netlist/component of interest. The overall idea is to monitor the given HW component running at its *rated* speed (with the frequency reported by the static timing analysis SW tool) to record a correct input/output dataset, and then, in repetitive steps, to increase the operating frequency of the HW component while re-processing this recorded dataset. The procedure completes when errors are detected at the output of the HW component (compared to the recorded data) and reports the maximum frequency for which the FPGA operates correctly. To implement the above idea, our framework is based on dynamic reconfiguration of clock tiles, on I/O data sniffing (e.g., from the PL pins, or inside the processor's memory), communication to external memory for storing/comparing datasets, and AXI communication between the PS and PL of Zynq (or other hard/soft-core processor). Next, we describe the specifics of our framework assuming HW/SW co-processing scenarios with Zynq, where the I/O of the application is managed by the processor (not the PL).

The flow of our framework is depicted in Fig. 5. It begins with HW *preparation* (top-left) and system *initialization* (bottom-left) to complete with the *testing* phase (right). First, as input to HW preparation, we developed the HW architecture shown in Fig. 6. Our component of interest (IP, top right) is placed within its own clock domain (dashed line), which can be configured dynamically by the Phase-Locked-Loop unit (PLL, bottom). The PLL is located in a clock tile of the FPGA and is controlled at run-time by the processor (PS, left) via Xilinx-specific commands sent through the AXI-lite port: at any time, the processor can send a sequence of Xil_out32 commands carrying 32-bit arguments to set designated PLL registers determining frequency factors and configuration flags. The reconfiguration time of this method was measured at 27-30 μsec. Besides PLL, the IP is connected to 2 dual-clock FIFO memories to cross the clock domains and transfer correctly the I/O data of the IP to the DMA controller (DMA, center). The DMA controller communicates with the PS, which includes the ARM processor executing the SW part of the application and the SW part of our framework. The DMA is used to implement AXI-stream PS-PL communication, however, the framework can adapt to any PS-PL communication used by the original HW/SW application. The DMA and PLL operate in their own clock domain at 100 MHz (fixed), whereas the ARM processor operates at 667 MHz. The PS is connected to the external world (e.g., to a host-PC) and to the external DDR memory which, among others, is assumed to hold the I/O data of the application.

After HW *preparation*, we have *initialization* (Fig. 5): in a host-PC, we process the input parameters (e.g., path to the application's data) and board-specific information (e.g., RAM addresses) with a Python script to output a Xilinx-specific TCL file, which includes commands to connect to the board (via JTAG), to initialize the system, and to load the input data to Zynq's external DDR (arbitrary choice, for demonstration purposes). Subsequently, the framework continues with *testing* (Fig. 5, right). A custom portable code loads to the processor, together with the application, and performs the following loop:

1) Initialize IP's clock freq. to f=f_r and f_{step}=10 MHz
2) Run the HW/SW application and store the output D_c to the external DDR memory (assumed correct data)
3) Increase the IP's f by f_{step} MHz (reconfigure the PLL)
4) Run the application and store the new output D
5) Fetch D and D_c from DDR and compare the data
6) If $D = D_c$ go to 3, else set $f = f - f_{step}$ and continue
7) If $f_{step} = 10$ then $f_{step} = 1$ and go to 3, else output f

With the above combination of coarse- and fine-grain search, the framework will detect the maximum possible frequency of the HW IP, and hence, will allow the system to continue its normal operation at maximum throughput. Note that the framework could be further simplified if the application uses an OS instead of the bare-metal approach described here.

V. Experimental Results and Evaluation

To test our framework and demonstrate its benefits, we use the eight Zedboards of Section III and two distinct HW components that both implement an FIR filter, which is a very popular DSP function. Each benchmark implements a different FIR in terms of complexity so that we can test the framework for HW components of dissimilar size (resource utilization). The first component, *IP1*, is a Direct-Form-I 64-tap FIR, highly-parallel (all coefficients are multiplied in

TABLE I: Initial frequency and resource utilization analysis

Resource	Resource Utilization (Zynq XC7Z020T-1CSG324)		
	Component IP1	*Component IP2*	*Framework+DMA*
LUTs	4907 (9.2%)	1859 (3.5%)	6020 (11.3%)
Register	11199 (10.5%)	4044 (3.8%)	7907 (7.4%)
RAMB36	0 (0%)	0 (0%)	3 (2.1%)
Rated freq.	140 MHz	176 MHz	100 MHz

TABLE II: Boost results after applying the proposed solution

Device	Inter-Chip Variability	Max. Achieved Frequency		Mean Boost (vs. rated f)
		Comp. IP1	*Comp. IP2*	
D764694	0 %	234 MHz	298 MHz	68.23%
D764698	0.18 %	232 MHz	298 MHz	67.52%
D764187	2.85 %	235 MHz	304 MHz	70.29%
D657830	3.06 %	233 MHz	301 MHz	68.73%
D764710	5.15 %	233 MHz	300 MHz	68.44%
D657343	6.3 %	238 MHz	304 MHz	71.36%
D764693	7.36 %	241 MHz	312 MHz	74.71%
D447252	17%	258 MHz	335 MHz	87.31%

parallel), with fixed-point accuracy (10 bits I/O, 26 internal bits). The second, *IP2*, is an FIR with similar architecture, but with less taps and lower precision (32-taps, 7 bits I/O, 19 internal bits). Both IPs have streaming I/O with throughput up to 1 datum/cycle. To sustain continuous operation, the framework's FIFO memories store 16 words (10- and 26-bit width) and the PS-PL communication is realized with handshakes over AXI4 protocol (sufficient for our test purposes).

The synthesis/implementation results are shown in Table I, together with the maximum frequency reported by the static time analysis SW tool (*rated* frequency is the baseline value). The size of the benchmark IP is in the area of 10K LUTs or less (reasonable for many applications), whereas the size of our HW framework is 6K LUTs (1.2K for clock management and 4.7K for our example communication). Note that, for a fair comparison, the user constraints and the strategy of the synthesis and implementation tools are calibrated to achieve maximum IP performance: for *IP1*, the reported speed was pushed to 140 MHz and, for *IP2*, to 176 MHz. Even so, our framework provides significant gains.

The speed gain after applying our framework is shown in Table II. We achieve up to 159 MHz higher frequency compared to the *rated* 176 MHz, i.e., 90.3% improvement. Overall, the improvement ranges from 66% to 90% (being $3\%-6\%$ higher for the smaller *IP2*) and almost doubles the speed in our FPGAs. Also, it increases among devices in accordance to the results of section III-C (column 2). For the 3 fastest devices (w.r.t. RO speed), the inter-chip variability and the speed difference against D764698 are correlated, while for the slowest devices we obtain relatively lower IP speeds. In particular, the speed difference among devices increases by up to 24 MHz for *IP1*, i.e., +10.2%, and up to 37 MHz for the smaller *IP2*, i.e., +12.4%. We stress that, for each IP, we placed the exact same bitstream on all eight devices and that each verification procedure used 1 Mega-sample test vectors. The junction temperature was increased from around 49^oC to 52^oC after clock boosting (for sustained operation). The framework's testing phase runs for only $3-6$sec per device.

VI. Conclusions

In the current work, we developed a ring-oscillator sensing infrastructure and a closed-loop framework to evaluate chip variability in 28nm FPGAs and improve their performance at run-time. Our results verified intra-chip variability in Zynq XC7Z020T in the area of 5.2% to 7.7% and inter-chip variability up to 17%. In accordance to this variability, our framework achieved a performance improvement of FIR filters by up to 90.3% compared to the SW reports and showed differences among the devices by up to 12.4%. These results demonstrate an additional performance margin in today's FPGA devices, which is worth exploiting via specialized HW/SW methods. In the future, we will enhance our framework to further improve performance by relocating the netlist within the FPGA.

Acknowledgment

The authors would like to thank Xilinx for the Zedboard donation and HiPEAC for the collaboration grant that supported the UoM and the NTUA in this work. Also, this work was partially supported by the FP7-612069-HARPA EU project.

References

[1] N. Z. Haron and S. Hamdioui, "Why is CMOS scaling coming to an end?" in *3rd Int'l Design & Test Workshop*. IEEE, 2008, pp. 98–103.

[2] S. Saxena, C. Hess, H. Karbasi, A. Rossoni, S. Tonello, P. McNamara, S. Lucherini, S. Minehane, C. Dolainsky, and M. Quarantelli, "Variation in transistor performance and leakage in nanometer-scale technologies," *IEEE Trans. on Electron Devices*, vol. 55, no. 1, pp. 131–144, 2008.

[3] T.-B. Chan, J. Sartori, P. Gupta, and R. Kumar, "On the efficacy of NBTI mitigation techniques," in *DATE 2011*. IEEE, 2011, pp. 1–6.

[4] D. Rodopoulos, P. Weckx, M. Noltsis, F. Catthoor, and D. Soudris, "Atomistic pseudo-transient BTI simulation with inherent workload memory," *IEEE Tr. Dev. & Mat. Rel.*, vol. 14, no. 2, pp. 704–714, 2014.

[5] M. Orshansky, S. Nassif, and D. Boning, *Design for manufacturability and statistical design: a constructive approach*. Springer, 2007.

[6] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM journal of research and development*, vol. 50, no. 4.5, pp. 433–449, 2006.

[7] V. J. Reddi, D. Z. Pan, S. R. Nassif, and K. A. Bowman, "Robust and resilient designs from the bottom-up: Technology, CAD, circuit, and system issues," in *Proc. of the 17th ASP-DAC*. IEEE, 2012, pp. 7–16.

[8] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali, "Supporting task migration in multi-processor systems-on-chip: a feasibility study," in *Proc. of Conference DATE*. EDAA, 2006, pp. 15–20.

[9] P. Sedcole and P. Y. Cheung, "Within-die delay variability in 90nm FPGAs and beyond," in *2006 IEEE International Conference on Field Programmable Technology*. IEEE, 2006, pp. 97–104.

[10] T. Tuan, A. Lesea, C. Kingsley, and S. Trimberger, "Analysis of within-die process variation in 65nm FPGAs," in *Quality Electronic Design (ISQED), 2011 12th Int'l Symposium on*. IEEE, 2011, pp. 1–5.

[11] K. M. Zick and J. P. Hayes, "On-line sensing for healthier FPGA systems," in *Proceedings of the 18th annual ACM/SIGDA int'l symposium on Field programmable gate arrays*. ACM, 2010, pp. 239–248.

[12] A. A. Bsoul, N. Manjikian, and L. Shang, "Reliability and process variation-aware placement for FPGAs," in *Proc. of Conf. DATE*. European Design and Automation Association, 2010, pp. 1809–1814.

[13] G. Nabaa, N. Azizi, and F. N. Najm, "An adaptive FPGA architecture with process variation compensation and reduced leakage," in *Proc. of the 43rd annual Design Automation Conf.* ACM, 2006, pp. 624–629.

[14] K. Shi, D. Boland, and G. A. Constantinides, "Accuracy-performance tradeoffs on an FPGA through overclocking," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*. IEEE, 2013, pp. 29–36.

[15] J. L. Nunez-Yanez, M. Hosseinabady, and A. Beldachi, "Energy optimization in commercial FPGAs with voltage, frequency and logic scaling," *IEEE Trans. Computers*, vol. 65, no. 5, pp. 1484–1493, 2016.