Measuring Timing Errors in FPGA-based Circuits

Joshua Levine, Edward Stott
Imperial College London
London, United Kingdom
{joshua.levine05,edward.stott07}@imperial.ac.uk

Nachiket Kapre
Nanyang Technological University
Singapore
nachiket@ntu.edu.sg

Abstract

FPGA-based platforms offer a unique and challenging platform for implementation of circuit-level timing-error detection and correction logic. If we operate FPGA designs beyond the conservative margins identified by the CAD tools, we can deliver substantial energy and performance improvements. In this paper we develop a strategy for monitoring timing errors in streaming FPGA circuits based on Razor-like shadow register insertion. Through a combination of careful calibration, hold timing control, critical path sampling and adaptation of the CAD flow, we design a robust, trustworthy error detection methodology for FPGA-based circuits. Our scheme can detect timing errors to deliver 39% energy reductions and 62% throughput improvements for the floating-point single-precision multiplier circuit with negligible overhead.

1 Introduction

FPGA architectures are reconfigurable, fully programmable architectures that allow the designer to implement any feasible circuit by configuring lookup tables (LUTs) and routing between these LUTs. The FPGA architecture community is broadly interested in understanding and exploiting the limits of modern VLSI technologies under the assumption that the resulting logic is reliable and fully deterministic. FPGA CAD tools use timing models that must respect these requirements and this means that substantial margins are built in to guarantee safe operation across all manufacturing, configuration and operating circumstances. This prevent us from using individual chips at their optimum performance and a batch of chips at its mean performance. For example, in our experiments, we implemented a single-precision multiplier on a DE0-Nano board and found that we can run the design error-free until 208MHz on gaussian-distributed input data while the CAD tool only certifies correct operation up to 136MHz.

In software systems (*e.g.* operating system kernels), we can use checksumming techniques or cheap pre-condition/post-condition checks to effectively detect failures. DRAMs have used forward error correction (FEC) schemes with scrubbing support to manage a limited number of upsets and corruption events. Algorithms may even provide cheap application-specific self-testing mechanism *e.g.* correctness of factorizing a number $a = p \cdot q$ can be checked by multiplying the factors, correctness of a solver for linear system of equations $Ax = b$ can be judged through a cheap residue calculation $|b - Ax|$. For circuits implemented on traditional ASIC (or CPU) platforms can implement device-specific error detection using techniques such as Razor [3]. In this paper, we exclusively focus on designing a **timing error** detection strategy for FPGA logic circuits that borrows some ideas and inspiration from the broader reliability systems stack.

What makes timing error detection in arbitrary FPGA circuits hard?
Error detection is only worthwhile if its area, timing and power overhead is outweighed by the benefits of operating beyond normal operating limits. For it to be successful we must address the following challenges:

- How do we build a timing error detector using the resources of the existing FPGA logic fabric?
- Which circuit paths from a given arbitrary circuit should we monitor to achieve the required timing fault coverage?

- How do we implement error detection while preserving the timing of the original design?
- Can we make the system robust to manufacturing and operating variations?
- Is it possible to automate the insertion of error detection logic to the FPGA compilation flow?

What architectural opportunities do FPGAs offer that can be used to design efficient error correction infrastructure?
FPGA architectures are highly configurable and a single chip can be reconfigured to support countless applications. This, plus the need for tractable automated place and route means that virtually all real-world FPGA designs contain significant amounts of unallocated logic, routing and clock resources. Using these spare resources, we can modify a circuit post-placement to insert shadow registers for monitoring critical paths post-compilation. FPGAs also offer superb clock network support with adjustable frequencies and tunable phases. FPGA circuit size, critical path distribution, input data correlations, and parametric variation will dictate the extent and nature of error susceptibility and error recovery opportunities.

The key contributions of this paper include:
- Design of a timing-error detection scheme (CAD flow) for arbitrary streaming FPGA circuits using a Razor-like shadow register insertion.
- Development of an in-situ demonstration platform that allows us to overclock and undervolt the FPGA circuit implemented on a Terasic DE0 Nano FPGA board.
- Empirical model for understanding error susceptibility and potential for recovery using our approach.
- Validation of out technique on the floating-point single-precision multiplier benchmark delivers a potential 39% energy reduction and 62% throughput improvement in the resulting design.

2 Background

2.1 Reducing operating margins

Operating margins of a typical FPGA circuit are affected by a combination of environmental conditions such as temperature, noise, statistical manufacturing variation, lifetime effects such as degradation as well as operational characteristics of input data.

Table 1: Performance optimization methods and the operating margins that they can reduce (✓=strongly, ✓=weakly)

Method	Var.	Deg.	Temp.	Noise	Data
Statistical Timing Analysis (SSTA)	✓				
Offline Characterization	✓	✓			
Tunable Replica Circuits (TRC)	✓	✓	✓		
Online Slack Measure. (OSM)	✓	✓	✓		
Shadow registers[1]	✓	✓	✓	✓	✓
Asynchronous	✓	✓	✓	✓	✓

[1]This work

- **Variation**: Manufacturing inconsistency means that timing and energy characteristics vary between dice and between components on a single die.
- **Degradation**: Phenomena such as NBTI can cause circuits to slow down over time.
- **Temperature**: Circuit performance varies with temperature — typically it deteriorates as temperature rises.
- **Noise**: Several stochastic effects influence circuit timing on a cycle-to-cycle basis, including thermal noise, crosstalk, power supply ripple and clock jitter.
- **Data**: The data arrival time at a particular register depends on the transitions that occur on all its input nodes. Usually, only the slowest case is considered: the critical path. However, the critical path may not be exercised frequently and the average delay may be significantly faster.

We can reduce the margins induced from these physical effects through a combination of a variety of techniques. We summarize the effectiveness and applicability of some of these techniques in Table 1. Our approach in this paper is derived from Razor but adapted to the FPGA fabric. In the Razor approach, shadow registers are inserted into the design and are used to detect timing errors. Multiplexers and stall circuitry are used to correct timing errors in a single clock cycle. By tolerating timing errors, Razor can reduce margins for stochastic timing variations such as noise and data.

2.2 Shadow Registers

We use shadow registers in our FPGA compilation flow to enable timing error detection in arbitrary circuits. The concept of shadow registers was originally introduced to tolerate timing errors in feed-forward processor pipelines due to dynamic voltage-frequency scaling (DVFS using Razor). DVFS for commercial FPGA platforms was studied in [1]. Timing errors are simply detected by inserting a shadow register for a datapath register with a suitably phase-shifted clock and comparing the stabilized values captured in the shadow register with the datapath register. We show this error detection circuit in Figure 1. The error signal generated from this comparison can be used in multiple ways as part of the error correction infrastructure: (1) stall, (2) bubble/counterflow, and (3) rollback. However, in this paper, we focus on the design and engineering of timing-error detection.

3 Vision

Ultimately, we want to design circuits that can operate reliably in presence of sub-micron manufacturing limitations as well as environmental operating conditions. A cross-layer solution [2] that distributes the reliability costs across different levels of the computation stack *e.g.* gates, operators, blocks, runtimes, operating systems, etc allows us to deliver a robust, reliable system with modest overheads. Timing error detection and recovery mechanisms have fundamental engineering constraints that must be considered to deliver efficient reliability solution for FPGA designs.

Detection: Cheap detection of errors in the design is critical to designing scalable reliability frameworks. Circuit errors that manifest as timing faults can be detected using strategic insertion of shadow registers. We can choose to monitor a certain fraction of paths in the circuit based on criticality and activation rates. We can also choose to rely on higher-level application-specific checks as discussed earlier. Other errors and faults such as SEUs in memory could be detected with checksumming and scrubbing techniques.

Correction: While not the focus of this paper, we briefly review correction and mitigation strategies that are available at our disposal. When we detect errors in the shadow registers, we can discard the results in the pipeline and restart/rollback the computation from a checkpoint. Unlike CPU pipelines with inbuilt mechanisms to support instruction reordering (out-of-order), we have to support rollback FIFOs [5] and state checkpointing to enable correct resumption of the circuit. Shadow registers in Razor can be used to hold the correct value and reinserting them in the datapath through a counterflow pipeline. On traditional FPGA fabrics implementing the reinsertion logic may be tricky due to limited availability of spare resources. We intend to investigate these issues in subsequent work.

4 FPGA Timing Error Detection

Our method for detecting timing errors is based on the *strategic* addition of shadow registers. A shadow register is added to the design for every circuit register (RUM) that may experience timing errors — a timing diagram illustrating the principle is given in Fig. 2. The shadow registers latch the same input as their associated RUMs but are controlled by a separate shadow clock, which has a fixed phase offset ϕ with respect to the main clock. During operation, the setup latency at a RUM input may occasionally be so great that the RUM does not correctly latch the data. However, by setting the shadow clock to lag the main clock we ensure that the shadow register always latches the correct data. Following a rising edge of the shadow clock the value stored in the shadow register can be compared to the RUM to establish whether the RUM experienced a timing error.

Shadow register insertion Shadow registers must be inserted to monitor all registers where timing errors may occur. This depends on the arrival time distribution of the registers in the design and the desired overclocking factor.

Shadow register insertion is carried out with an automated flow based on [4]. Here, the placement and routing of the original circuit is preserved while shadow registers are added. This means that the timing characteristics of the application circuit are not significantly impacted by the process. This is important because timing changes could change the criticality ordering of registers with the result that the wrong parts of the circuit are monitored for errors. A side-effect of this is that the data arrival time at the shadow registers can vary significantly from

the RUMs.

The timing slack measurement application of [4] can tolerate large variation in the shadow register timing offsets due to its phase sweeping technique. However, for error detection we require that all the shadow registers have similar delay offsets as they are triggered from a common, fixed-phase clock. We achieve this by adding minimum and maximum timing constraints for the shadow register offset. Fortunately, modern FPGAs are adept at meeting complex timing constraints as they are frequently required in systems with multiple clock domains and off-chip synchronous communication. The flexible and over-provisioned routing resources can be adapted to insert delay into circuit nets as required and the CAD tools have proficient algorithms for automating the process.

Hold timing The first challenge to be resolved in making this scheme practical is associated with hold delays. Since the shadow clock rises after the main clock there is a danger that fast propagation from the RUM reach the shadow register before it latches. This would cause the shadow register to sample early activity from cycle $n+1$ rather than late activity from cycle n as intended. To prevent this, we must enforce a minimum delay between a RUM and all its fan-in registers. As with the constraints for shadow register insertion, the FPGA tool flow can readily meet these requirements and insert delay into fast paths without impacting slow paths.

Calibration To use error detection safely it is important to know the minimum hold delay at each shadow registers — these determine the maximum allowable shadow clock lag ϕ — and the shadow path delay offsets — these, with the selected ϕ, govern the maximum overclocking factor. However, timing models are not accurate enough to control operation of the error detection system. Typically, upper-bound and lower-bound timing models (used respectively by the tool to check setup and hold constraints) differ by a factor of around $2\times$. Fortunately, the shadow register hardware is capable of measuring the necessary timing parameters in an offline calibration process based on frequency and phase sweeps.

5 Experiments

We run our experiments on the Terasic DE0 Nano FPGA board supported by a programmable TTi PL303QMD-P

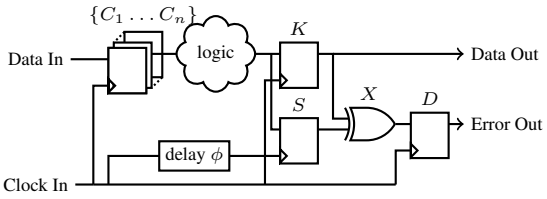

Figure 1: Timing Error Detection using Shadow Registers

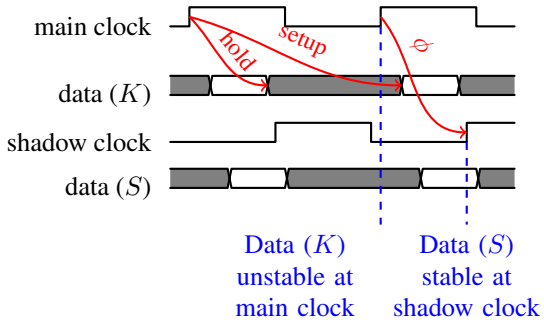

Figure 2: Error detection timing

PSU. We show our experimental setup in Figure 3.

Error detection capability

To test the effectiveness and utility of timing error detection we conducted experiments with widely-used complex logic operators. First, we compared errors detected by shadow registers with traces recorded from the circuit outputs by a block RAM. Analysis of the circuit netlist provides the pipeline latency between each RUM and the output, and this allows us to align detected errors with output discrepancies. A frequency sweep is conducted to overclock the circuit to beyond the point of timing failure.

The results for this in the **fpmult32** (single-precision multiplier) benchmark are shown in Fig. 4. With an 8kword input vector set taken randomly from a gaussian distribution, no errors at all occur below 208MHz — the fmax given by the timing tool is 136MHz. From there, the detected error rate increases almost exponentially with frequency. These are errors detected by the shadow registers which correspond with errors in the output trace. At higher frequencies, missed errors are recorded — these are errors in the output trace that were not picked up by

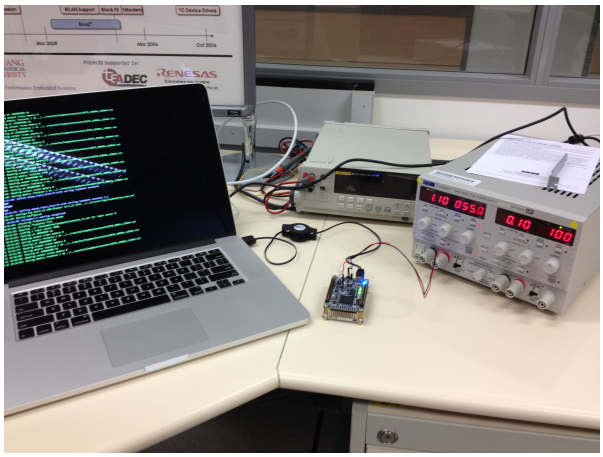

Figure 3: Experimental Platform with the Terasic DE0 Nano FPGA Board and TTi PL303QMD-P PSU controlled using a an Ubuntu 12.04 64-bit VM

the shadow registers. These can occur if timing faults are introduced at registers which are not monitored, or if the clock period becomes so short that timing faults occur at the shadow registers.

Another possibility for the missed errors is metastability. When timing faults occur there is the potential for registers to enter a metastable state. If the metastable state persists long enough then downstream registers can sample non-deterministic data values and errors can be introduced. Such errors may not be observed by the timing error detection circuitry because either (a) the metastable state resolves to a correct data value before the error signal is latched, (b) the metastable state appears as correct data to the error detector due to biasing conditions, or (c) the metastable state propagates to the error detection register. Metastability could be addressed by adding additional shadow registers to downstream logic. These would detect the resulting late transitions. As with synchronizers, once metastability is introduced it is impossible to eliminate entirely — it can only be mitigated to achieve an acceptable failure rate.

The final class of errors recorded are false errors. These are detected timing faults with no corresponding output error. They can occur simply because the fault is masked by downstream logic and does not propagate to the outputs. Other causes include insufficient shadow register hold delay and metastability. False errors do not affect the correctness of the circuit output, but where error correction is employed they will incur an unnecessary overhead.

Performance gains We quantified the potential gains of using error detection by modeling throughput and energy consumption in a error correction system. Fig. 5 shows the throughput relative to the timing model guarantee as clock frequency is increased. We consider two systems, one where error correction is achieved in a single clock cycle (like stall-based Razor) and one where it takes ten clock cycles (for example a rollback system). A substantial improvement, around 50%, could be made without incurring any errors (and 62% with error recovery support). However, that does not mean that it would be safe to operate at this point without error detection. The experiment did not test an exhaustive set of input vectors and even if certain operating parameters are monitored margins would have to be allowed for noise, crosstalk and jitter effects. Error detection and correction offers protection from these effects and allows throughput to be pushed further. A 10-cycle correction penalty soon outweighs the gains of overclocking much beyond the point of timing fault introduction but a single-cycle system could offer significant throughput gains.

We also looked at the implications for computation energy and these are shown in Fig. 6. Here, voltage is reduced and the corresponding computation energy is calculated in a similar way by assuming that errors incur an energy penalty equivalent to either one or ten extra clock cycles. Energy can be reduced by around 35% before the first timing faults occur. Here too a further improvement up to 39% can be made by pushing further and relying on error correction to make a net saving.

6 Conclusions

In this paper, we highlight our preliminary design of a compilation flow for automatically inserting timing error detection logic in arbitrary circuits. As our framework matures, we expect to characterize a wider range of circuits and IP cores. This will be part of a cross-layer reliability stack with a strong desire to integrate with stall/rollback logic. We are currently investigating the feasibility of supporting automated insertion of stall/rollback logic using the FPGA substrate. We estimate 39% en-

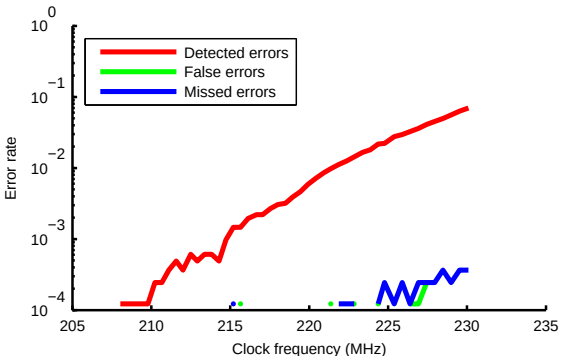

Figure 4: Errors detected, falsely detected and missed, versus clock frequency

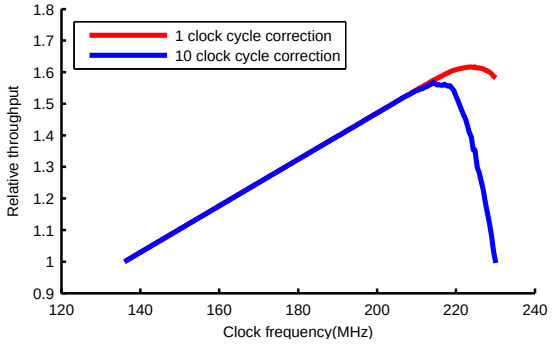

Figure 5: Throughput relative to timing model versus clock frequency

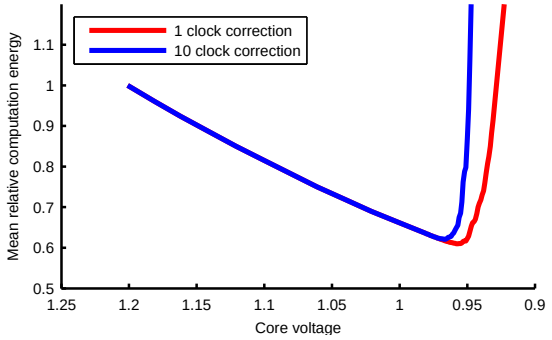

Figure 6: Mean energy per computation versus core voltage

ergy savings and 62% throughput improvements in the FPGA mapped implementation of a floating-point single-precision multiplier when using our approach. These early results show the potential for large improvements in performance and efficiency by responding to timing faults as they occur.

References

[1] C. T. Chow, L. Tsui, and P. Leong. Dynamic voltage scaling for commercial FPGAs. *Proceedings of the International Conference on Field-Programmable Technology (ICFPT)*, 2005.

[2] A. DeHon, H. M. Quinn, and N. P. Carter. Vision for cross-layer optimization to address the dual challenges of energy and reliability. In *Proceedings of Design Automation and Test Europe*, 2010.

[3] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *36th International Symposium on Microarchitecture*, pages 7–18. IEEE Comput. Soc, 2003.

[4] J.M. Levine, E. Stott, G.A. Constantinides, and P.Y.K. Cheung. SMI: Slack Measurement Insertion for Online Timing Monitoring in FPGAs. In *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2013.

[5] H. Naeimi and A. DeHon. Fault-tolerant sublithographic design with rollback recovery. *Nanotechnology*, 2008.