2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines

Timing Fault Detection in FPGA-based Circuits

Edward Stott, Joshua M. Levine and Peter Y.K. Cheung
Imperial College London
London, UK
{*ed.stott;josh.levine05;p.cheung*}@*imperial.ac.uk*

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

***Abstract*—**
The operation of FPGA systems, like most VLSI technology, is traditionally governed by static timing analysis, whereby safety margins for operating and manufacturing uncertainty are factored in at design-time. If we operate FPGA designs beyond these conservative margins we can obtain substantial energy and performance improvements. However, doing this carelessly would cause unacceptable impacts to reliability, lifespan and yield — issues which are growing more severe with continuing process scaling.

Fortunately, the flexibility of FPGA architecture allows us to monitor and control reliability problems with a variety of run-time instrumentation and adaptation techniques. In this paper we develop a system for detecting timing faults in arbitrary FPGA circuits based on Razor-like shadow register insertion. Through a combination of calibration, timing constraint and adaptation of the CAD flow, we deliver low-overhead, trust-worthy fault detection for FPGA-based circuits.

I. Introduction

Traditionally, FPGA designers assume that their circuit behaviour is fully deterministic and highly reliable. To respect this usage model, FPGA CAD tools are based on conservative timing models which guarantee safe operation across all manufacturing, configuration and operating circumstances. This prevents us from using individual chips at their optimum performance and batches of chips at their mean performance.

Unlocking this potential in practice is not straightforward. Some existing techniques attempt to measure performance per-device or even continuously in the field. This can eliminate some margins, especially those that compensate for static or slowly-varying phenomena. However, determining what margins should remain in place is complex as fault-free operation is still required and allowances must be made for any inaccuracy and reliability of the measurement system. Another approach is to permit faults but control their impact. A number of high-level fault detection and correction systems exist but it is challenging to automate their application to arbitrary circuits, especially without incurring large overheads.

Our work follows the approach of Razor [1], where fault detection takes place at the register level and faults are corrected through pipeline stalls or bubble insertion. Razor was designed for use in microprocessors but it has never achieved commercial success due to the difficulty and overhead of integrating fault correction into complex, high frequency instruction pipeline logic. Our approach is to integrate Razor-like fault detection into an FPGA design flow so that in can be applied automatically to arbitrary designs. Since FPGAs operate at moderate clock frequencies and use function-specific pipelines with simple control logic, we hope to achieve performance benefits without the design challenges experienced in CPUs.

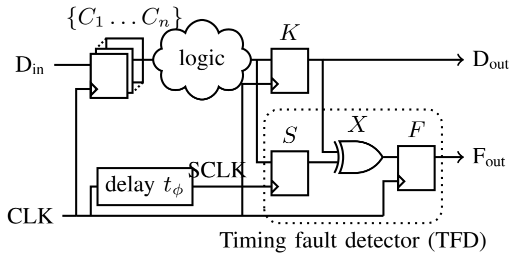

Figure 1: Timing fault detection using a shadow register

Error detection is only worthwhile if its area, timing and power overhead is outweighed by the benefits of operating beyond normal operating limits. Fortunately, virtually all real-world FPGA designs contain significant amounts of unallocated logic, routing and clock resources. Thanks to the flexibility of the FPGA architecture we can modify a circuit post-placement to insert shadow registers for monitoring critical paths post-compilation. Such opportunities are not available in ASICs without significant incremental cost.

The key contributions of this paper include:

- Design of a timing fault detection scheme for FPGA architectures using shadow register insertion.
- Development of a CAD flow and calibration system for automated addition of timing fault detectors to arbritary FPGA applications.
- Development of an in-situ demonstration platform that allows us to explore the fault behaviour of real circuits.

II. FPGA Timing Fault Detection

A. Fault detection cells

Timing fault detection is based on the addition of fault detector cells (TFDs) as illustrated in Fig. 1. A TFD detects

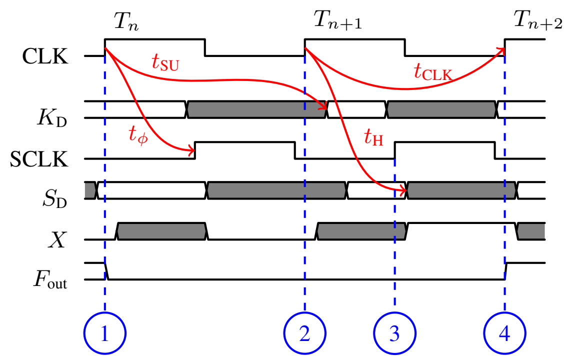

Figure 2: Timing diagram for fault detection

timing faults at a circuit register K that lies at the end of one or more timing paths originating at source registers $\{C_1 \ldots C_n\}$. The TFD consists of a shadow register S, a comparator X and a fault latch F. S samples the same data as K but on a clock SCLK that lags the main system clock CLK by t_ϕ. Any data that arrives after K samples but before S samples will cause a discrepancy between the two registers that is detected by comparator X. F latches the discrepancy signal and resynchronises it with the main clock domain, indicating whether a timing fault has occurred.

Fig. 2 gives an example timing diagram for the system. A transition event, initiated at ①, takes t_{SU} to propagate to the data input of K. In this case, t_{SU} is longer than t_{CLK}, the clock period. Hence the input to K has not stabilised by ②, the following clock edge. ③ is the rising edge of the delayed SCLK. By this time, the data input to register S has stabilised and so this register samples correctly. At this point comparator X detects the discrepancy between the two registers and the fault condition is registered by F at ④.

B. Timing Constraints

For the system to operate correctly a number of timing constraints must be enforced:

Hold timing: The system is dependent on shadow register S sampling correct data on every clock cycle. The shadow clock phase shift t_ϕ ensures that it is unaffected by a long t_{SU}, but this shift also makes S susceptible to hold time violation. The shadow register hold delay t_h is illustrated on Fig. 2 — it is the time for which S_D is stable following a clock edge. t_ϕ must be less than this amount or S will sample activity from clock cycle T_{n+1} while K samples cycle T_n. We must set a minimum t_h constraint for all registers K with attached TFDs. Fortunately, FPGA CAD tools can meet such constraints by inserting delays into routing segments as necessary.

TFD delay offset: As indicated in Fig. 2, data does not arrive simultaneously at K and S. Since TFDs are added after main circuit placement and routing are fixed, data will normally arrive later at S than K. This difference is readily compensated for by adjusting the shadow clock lag t_ϕ. However, if there is too much variation in the TFD delay offsets in a single clock domain then it will not be possible to select a single t_ϕ that simultaneously meets setup and hold constraints at all TFDs. To avoid this problem, timing constraints must be placed on S_D to bound the offset to a fixed range. The upper bound must be large enough for the tool to find a compliant placement for all the TFDs, while the lower bound is met by tuning routing delay.

Fault register timing: The components of the TFD may not be located near to each other due to resource availability or downstream timing constraints. Since t_ϕ is not set at compile time, the CAD tool must be given constraints for the maximum delay between S and F, setting an upper limit for t_ϕ.

C. TFD insertion

TFD insertion is based on an important parameter: the critical delay margin (CDM). The CDM selects which circuit registers have TFDs attached. A register has a TFD attached if, according to timing analysis, $t_{SU} > t_{crit}(1-\text{CDM})$, where t_{SU} is the setup delay for the register under consideration and t_{crit} is the setup delay of the critical path for that clock domain. For timing fault detection, CDM is used to ensure that all the registers which could suffer a timing fault are monitored. This relates directly to the desired operating limit — for example if the circuit is to be clocked at up to 120% of f_{max}, then a CDM of 0.2 is necessary.

There is an interesting discussion about what is meant by f_{max} in this analysis. In the absence of any other information, the figure given by the timing model must be used. If we want to operate the circuit far above this point the CDM, and hence the area overhead of the TFDs, will be large. But if we know that the circuit can operate at a certain empirical f_{max}, which is greater than the timing model predicts, we can operate at the same frequency with a lower CDM. This means that timing fault detection is effectively replacing operating margin with an area overhead margin in the form of a conservative CDM figure.

III. Experimental Method

A. Compile Flow

Error detection is added to FPGA designs with an automated compile flow. The flow has two goals: first, it should add error detection circuitry to arbitrary application circuits without manual intervention, and second, it should not perturb the application circuitry such that its timing is adversely affected.

A block diagram of the process is given in Fig. 3. It is based on a flow developed for adding shadow registers for measuring timing slack[2]. First, the application HDL is compiled as-is to produce a technology-mapped netlist, a timing report and a placement and routing. The netlist is modified to add TFDs, using the information in the timing

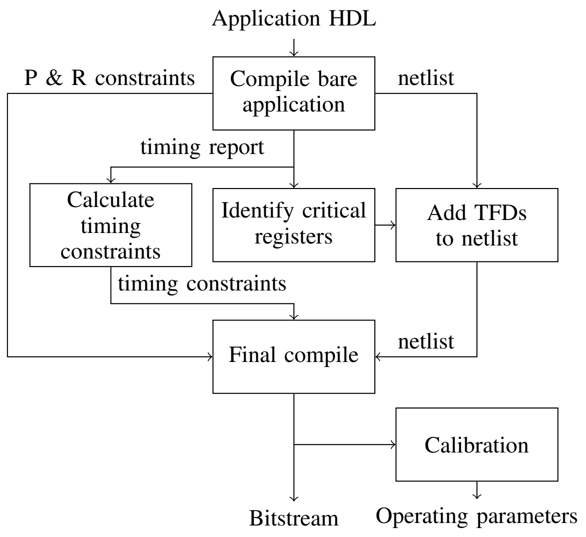

Figure 3: Compile flow for adding timing error detection

report to identify where they should be attached. Then it is recompiled with placement, routing and timing constraints derived from the original compilation and timing constraints as described in the previous section. Once a bitstream is produced, it is loaded onto the target FPGA and calibrated.

B. *Calibration*

To use error detection safely it is important to know certain timing parameters about the design:

1) Shadow register hold slack t_{H}. This determines the maximum shadow clock lag t_ϕ since $t_\phi < t_{\mathrm{H}}$
2) Shadow register setup delay t_{SU} at f_{max}. This governs the maximum overclocking factor by placing a lower limit on t_{clk}, since $t_{\mathrm{clk}} + t_\phi > t_{\mathrm{SU}}$.

Unfortunately, upper-bound and lower-bound timing models differ by a factor of around $2\times$ and they are not accurate enough to produce a tightly bounded delay report. However, the shadow register hardware is capable of measuring the necessary timing parameters in an offline calibration process based on frequency and phase sweeps.

To measure t_{H}, the phase lag of the shadow clock is increased until discrepancies are detected between main and shadow registers and at this point $t_{\mathrm{H}} = t_\phi$. t_{SU} can be measured in a similar way, this time the shadow clock is advanced with respect to the main clock until discrepancies are detected and $t_{\mathrm{SU}} = t_{\mathrm{CLK}} - t_\phi$.

C. *Test Platform*

Experiments are performed on an FPGA test platform based on a 65nm Altera Cyclone IV FPGA, shown in Fig. 4. In the FPGA, the circuit-under-test is connected to block RAMs to provide arbitrary test stimulus and record the primary and error outputs. Clock frequency is precisely adjustable thanks to the flexible clock generator built into the FPGA. Voltage is set with an external power supply and temperature is controlled with a thermoelectric heat pump.

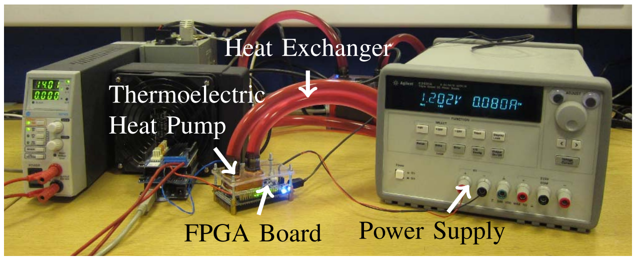

Figure 4: Test Rig with the Terasic DE0 Nano FPGA Board

Table I: Benchmark characteristics after TFD insertion with CDM= 0.1

Name	TFDs	Logic Elements		f_{sta}	
		Δ	%	MHz	%
`fpexp32`	19	38	+5.0%	76.1	-6.0%
`fpexp64`	39	78	+2.6%	84.0	-0.1%
`fplog32`	44	88	+5.1%	95.6	-2.7%
`fpmult32`	24	48	+14%	135	-4.9%
`dct1d`	20	40	+7.7%	169	+1.8%
`filter`	13	26	+4.3%	104	+0.0%

TFD: Timing fault detector

D. *Benchmarks*

We envisage the most powerful implementations of error-managed computing to be based on systems of asynchronous, independently-controlled components. Hence, we have tested our technique on a range of building-block operators varying in size from 300-3000 LEs. Some of the benchmarks use embedded block RAM and multiplier blocks.

IV. Results

A. *Benchmark compilation*

Each of the benchmarks were compiled with shadow register insertion. The results are given in Table I with the area and timing model performance overheads. Overheads are quoted for a CDM of 0.1. All benchmarks were tested with appropriate random input vectors, though changing the input distribution was found to make little difference to the fault behaviour in practice.

B. *Fault detection capability*

To test the effectiveness and utility of timing fault detection we compared faults detected by TFDs 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 faults with output errors. A frequency sweep is

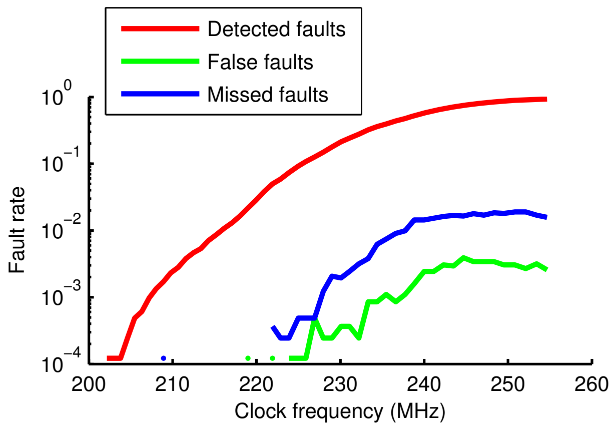

Figure 5: Faults detected, falsely detected and missed, versus clock frequency in `fpmult32`

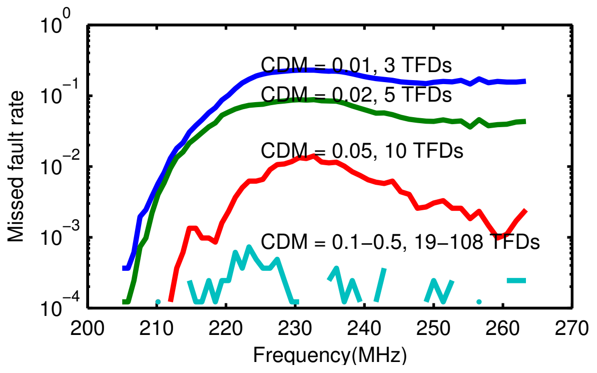

Figure 6: Effect of varying the critical delay margin (CDM) on the rate of missed faults in `dct1d`

conducted to overclock the circuit to beyond the point of timing failure.

The results of this analysis in `fpmult32` (single-precision multiplier) is shown in Fig. 5. With an 8kWord input vector set taken randomly from a Gaussian distribution, no faults at all occur up to around 200MHz — the $f_{\max}$ given by the timing tool is 142MHz. From there, the detected fault rate increases with frequency. These are faults detected by the shadow registers which correspond with errors in the output trace. At higher frequencies, missed faults are recorded — these are errors in the output trace that were not picked up by the TFDs. 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 faults 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. Work is ongoing to establish the severity of metastability in our system. A number of mitigation measures could be investigated if too many missed faults are occurring.

The final class of faults recorded are false faults. 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 faults do not affect the correctness of the circuit output, but where fault correction is employed they will incur an unnecessary overhead.

Fig. 6 shows the result of an experiment to see how the rate of missed faults varies with the CDM. Curves are plotted for four different CDM parameters, which result in TFD counts of between 3 and 108 in the `dct1d` benchmark. Below 0.1, the CDM value has a strong impact on the missed fault rate. However, above 0.1 no significant difference is observed. This pattern is also observed in other benchmarks and for this reason CDM is set to 0.1 elsewhere in this work. Note that the first detected faults in this benchmark occur at 205MHz, meaning that with a CDM below 0.05 faults are missed straight away but when CDM$\geq$ 0.1 peak throughput can be reached with no missed errors.

V. Conclusions

In this paper, we describe a technique for shadow register-based timing fault detection in arbitrary FPGA circuits. We have developed an automated flow for adding the necessary circuitry with low overhead and without disturbing the original design. Our flow is capable of managing the tricky timing constraints and includes a self-calibration process for optimising operating parameters. We tested the system on a range of benchmarks to check its reliability and quantify overheads. Coupled with a suitable fault correction system, we envisage that timing fault detection will be just one component of a cross-layer performance and reliability management stack.

References

[1] 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 *Proc. of the IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2003.

[2] 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. of the Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2013.

This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/K034448/1, EP/I012036/1, EP/H013784/1]. See `www.prime-project.org` for more information about the PRiME programme.