Available online at www.sciencedirect.com
SCIENCE DIRECT®
Microprocessors and Microsystems 30 (2006) 388–397
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

Dynamic clock-frequencies for FPGAs

J.A. Bower [a,*], W. Luk [a], O. Mencer [a], M.J. Flynn [b], M. Morf [b]

[a] Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK
[b] Computer Systems Laboratory, Department of Electrical Engineering Stanford, CA 94305, USA

Available online 28 February 2006

Abstract

Most FPGA designs run at a fixed clock-frequency determined through static analysis in FPGA vendor supplied tools. Such a clocking strategy cannot take advantage of the full run-time potential of an application running on a specific device and in a specific operating environment. This paper describes methods for using dynamic clock-frequencies to overcome this limitation. We begin by describing a methodology for designing systems which allow dynamic clock-frequencies in FPGAs. We then present a framework for exploring the dynamic behaviour of suitable clock-frequencies for a number of FPGA applications in varied operational environments. Finally we introduce our AutoTEA system, which automatically adds circuitry to arbitrary FPGA designs for dynamically adjusting clock-frequency to a safe limit given current operating conditions. Our results show that dynamically clocking designs can lead to a speed improvement of 33–86% compared to using a fixed, statically estimated clock.

Keywords: Timing analysis; Better than worst-case performance; Over-clocking; Power-saving; High-performance computation

1. Introduction

FPGAs enable the implementation of adaptive high-performance applications. Such custom hardware designs require a custom clock-frequency which balances performance with reliable circuit operation.

In a traditional FPGA design flow, clock-frequency is determined through static analysis of netlists for a device performed by software. After place-and-route a timing analyser program locates the longest combinatorial path between RAMs, I/Os and flip-flops [1]. This path, termed the 'critical-path', is the main bottleneck of the system and as such a design's clock-frequency can be derived from this.

We identify two sources of wasted performance with a statically determined clock-frequency. First, the manufacturing process for FPGAs is not perfectly uniform, and different physical devices have different characteristics. Second, the actual propagation delay through any path in a device changes during operation. Such changes are caused either by varying environmental conditions or even internal temperature changes due to different inputs varying power consumption.

Static timing tools deal with the above issues by employing worst-case models for estimating delays in hardware designs – leading to conservative clock frequencies. Another problem with using a fixed, statically determined clock-frequency is that all possible operating environments must be supported, however in some situations the environment may exceed conservative error margins. For example, a battery powered device running low on power and suddenly placed into a hot environment may fail although sufficient power would have been available to continue processing at a lower rate.

Dynamic clock-frequency schemes are alternatives to static estimates and have the potential to improve both high-performance and low-power applications. High-performance applications can run at the maximum physically attainable speed, while low-power applications can in general manage power-consumption by optimally balancing

* Corresponding author.
E-mail addresses: Jacob.bower@imperial.ac.uk (J.A. Bower), w.luk@imperial.ac.uk (W. Luk), o.mencer@imperial.ac.uk (O. Mencer), flynn@ee.stanford.edu (M.J. Flynn), morf@snow.stanford.edu (M. Morf).

0141-9331/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2006.02.006

clock-frequency, voltage supply, and demand for computation.

This paper contributes towards realising practical FPGA-based systems with dynamic clock-frequency. In particular, we provide:

- A methodology for creating dynamic clock-frequency systems which provide user confidence in correct operation.
- LIMIT: A hardware framework and experiments for exploring the behaviour of maximal safe dynamic clock-frequencies in FPGA designs.
- AutoTEA: An automated implementation of a technique for dynamically adjusting clock-frequencies to their optimal value for arbitrary FPGA designs.
- Experimental results from our LIMIT and AutoTEA systems applied to a diverse set of FPGA applications under varied environmental conditions.

The remainder of this paper is organised as follows. In Section 2 we review related work. We present our methodology for designing systems which implement dynamic clock frequencies in Section 3. In Section 4 we present our LIMIT and AutoTEA systems for implementing and evaluating dynamic clock-frequencies. Finally, in Sections 5 and 6 we present results of our experiments with AutoTEA and LIMIT, and our conclusions.

2. Previous work

Dynamic clock-frequency is already common in modern microprocessors and high-end ASICs [2]. ''Over-clocking'' in such systems pushes clock-frequency beyond vendor specifications [3]. While such a manual and brute-force approach is not suitable for serious computing systems, research in this area aims to develop solutions that reliably and dynamically adapt clock-frequency to optimal limits. In this paper, we limit ourselves to a discussion of dynamic clock-frequency systems for FPGAs.

We categorise some examples of this work into error tolerating and error avoiding systems and compare them to our own work in Table 1. From the table, three systems use error detection and correction techniques to tolerate errors in over clocked logic: TIMERTOL [4], Razor [5,12] and DIVA [6]. In the TIMERTOL and Razor systems, errors are detected by sampling inputs to pipeline register stages at two different times. The idea is that early samples can be used to continue processing and later outputs, which are more stable, can be used to detect if the early samples are erroneous. DIVA describes a microprocessor with a combined system of simple checker logic and a complex processing core. The simplicity of the checker logic allows it to be aggressively optimised for high-speed operation. In this system the complex processing logic is over-clocked using the high-speed checker to catch all errors. The common idea in all these schemes is to allow logic to run over-clocked, and check for errors to prevent committing erroneous outputs. Clock-frequency changes dynamically to minimise error rates.

Two systems which avoid errors by continuously re-evaluating and adapting clock-frequency to maintain correct functionality are: a self-timed PIC16C57 compatible microprocessor [7] and the TEAtime system [8]. In the PIC16C57 microprocessor, execution is paused while worst-case inputs exercise the system critical-path and the results are checked for errors. Clock-frequency is adapted to eliminate errors in the critical-path. TEAtime applies a similar idea, except that it allows continuous operation of a microprocessor design by creating a duplicate critical-path for checking. This duplicate (or false) critical-path is a one-bit wide version of the longest flip-flop-to-flip-flop path in the main design with additional delay. The idea is that the false critical-path, with its extra delay, will fail before the main design so clock-frequency can be adjusted based on observing errors in the false critical-path.

Other related research areas include: implementing low temperature designs, designing for ''average case performance'', dynamic voltage scaling (DVS) and adapting clock-frequency to computation. The idea of designing for average case performance is to create hardware which can achieve higher clock-frequencies when inputs are ''average case'', and scaling the clock during worst-case inputs [9]. Designing for low temperature enables higher clock frequencies. A novel method for lowering temperatures in FPGAs is to use dynamic reconfiguration to prevent single areas of a chip from getting too hot [10]. Schemes for adapting clock-frequency to specific computation have also been developed with clock period altered each cycle depending on which units are currently active [11]. Dynamic clock-frequencies are also applicable to systems implementing DVS [14] as they allow frequency to be optimally tailored to match the dynamic voltage.

Table 1
Comparison of our AutoTEA system to related efforts

	AutoTEA (ours)	TIMERTOL [4]	TEATime [8]	DIVA [6]	Razor [5]	PIC16C57 [7]
Error avoidance/tolerance	Avoidance	Tolerance	Avoidance	Tolerance	Tolerance	Avoidance
Overhead scales with design size	No	Yes	No	Yes	Yes	No
Automated implementation	Yes	No	No	No	No	No
Potential for automation	High	Medium	High	Low	Medium	Low
Prototype technology	FPGA	FPGA	FPGA	ASIC	ASIC [12]	ASIC

3. Our methodology for dynamic clock-frequency

Our objective is to create automated, dynamic, and adaptive clock-frequency system. It is critical to the success of such a system that the user has confidence that there are no erroneous outputs due to timing errors. To achieve this, we identify three elements of dynamic clock-frequency systems: (1) error analysis, (2) error handling, and (3) handling of dynamic clock-frequency.

3.1. Error analysis

With dynamic clock-frequency an error analysis method appropriately adjusts the clock. This error analysis can come in one of two forms:

- *Error detection* – discover when errors have already occurred.
- *Error prediction* – determine the likelihood of errors occurring in the future.

For error detection, we modify a hardware design in either an architecture dependent or independent manner to include points where we test results for error. An example of an architecture dependent error checking scheme is manually creating a simplified version of a complex system which can be used to validate results. This is the method employed in the DIVA system [6]. In contrast, architecture independent error detection systems do not require detailed understanding of a design's operation to check for errors. Instead some general form of redundancy either in space or time is used to validate computations. For example in the Razor system results are used early, but compared with later outputs which have had more time to settle on a correct value.

Varying the level of architecture dependence in an error detection method reflects a trade-off in design complexity versus efficiency. In a highly architecture dependent system, complexity arises as a designer must create tailored validation mechanisms. However, this complexity allows a designer to minimise the points at which errors need to be checked. Architecture independent systems have greater potential for automation, but require more overheads as they are applied at the low-levels where features common to more architectures arise.

In an error prediction system there is no explicit check for errors. Instead parameters indicative of system failure are monitored. Suitable parameters include: physical condition e.g., temperature and power consumption, fuse-circuits designed to fail before the main design, and data inputs which indicate when stress on the system is likely to increase. Clock-frequency is dynamically adapted to pre-emptively avoid failure. Our AutoTEA system described later is an example of a fuse-circuit based error predictor.

Advantages of error predictors over error detectors include less interaction with a design thus allowing them to be architecture independent without adding significant overhead. The advantage of error detectors includes stronger guarantees of catching errors.

3.2. Error handling

While error analysis is needed to adjust clock-frequency, error handling is a strategy used for preventing erroneous results that may arise from being committed. Methods for handling error broadly fall into two classes: error avoidance, and error tolerance.

Error avoidance systems detect the potential for failure in either future or past results. Systems using error prediction include implicit error avoidance as they keep changing clock-frequency to avoid errors. In error detection systems where error checking is expensive, error avoidance can also be useful. For example, checkpoints can be used where samples of outputs are checked and in the case of an error we roll-back to the last checkpoint.

Error tolerant systems allow errors to occur but ensure no erroneous results are ever committed by using detection and correction techniques. These types of system rely on error detection with all relevant outputs checked. If an error is detected, the offending computation is either corrected or recomputed. The correction or re-computation of results leads to some kind of time penalty, so clock-frequency is adjusted to balance error rate with performance.

Error tolerant systems have potential to achieve higher clock-frequencies than error avoidance systems. This arises as they can more precisely hone-in on optimal frequencies by recovering from errors caused by going over the optimal limit. Error avoidance systems will inherently be more conservative to ensure no erroneous results ever arise. Despite this conservative approach error avoidance still have the advantage over static systems of adapting to a dynamic environment. The choice between tolerance and avoidance is mediated by a trade-off in complexity of checking all errors versus performance benefits of doing so.

Our AutoTEA system described later is an example of an error avoidance system.

3.3. Handling of dynamic clock-frequency

A method for varying clock input is essential to a system for dynamic clock-frequency. Handling dynamic clock-frequency requires:

- Variable clock-signal generation hardware.
- A system for controlling the clock-generator using information from error analysis.
- An algorithm to maximise the rate of convergence on a suitable clock-frequency.

Clock-generation hardware in its simplest form requires a system to stop operating while clock-frequency is adjusted and allowed to stabilise. A more advanced implementation such as the one used in the self-timed PIC16C57 mentioned earlier, allows clock-frequency to scale rapidly.

Clock-control logic can either be integrated for an embedded system, or in an external controller such as a host computer in a cooperating hardware/software system. The advantage of using an external controller is that this reduces the possibility of errors arising in the clock control logic itself due to environmental conditions, and can also allow for more complex clock-frequency selection systems.

Finally, we employ a strategy for choosing clock-frequency. Two examples of strategies for finding suitable clock-frequency are: binary search and linear increasing/decreasing clock-frequency. Binary search allows fast convergence on a suitable clock-frequency, but a running system may need to halt in order to perform the search to mitigate the times during the search where the clock-frequency is too high. In contrast, a linearly adjusting clock-frequency facilitates less interruptions during operation, but may take longer to reach an ideal frequency. More complex models based on preliminary stress testing of a circuit can also be used.

3.4. Other considerations

In addition to our three main identified features of dynamic clock-frequency systems, other factors to consider include:

- Ability to deal with meta-stability, which arises as physical signal propagation times become comparable to the clock period.
- Overheads when incorporating the scheme in: design time, size and power consumption.
- Applicability to a variety of architectures.

For FPGAs, applicability to a wide range of architectures and low design time overheads are particularly important as designs are often limited only by what is synthesisable and can evolve rapidly.

4. Implementations

We show two implementations of systems with dynamic clock-frequencies in FPGAs: AutoTEA and LIMIT. LIMIT explores the limits of dynamic clock-frequency behaviour in FPGA applications. AutoTEA is motivated by results from LIMIT, and provides what we believe is the first implementation of an automated approach to adding dynamic clock-frequency features to arbitrary FPGA designs.

In both LIMIT and AutoTEA we focus on finding "optimal" clock-frequencies, which we define as clock-frequencies at which there are no errors given current operating conditions.

4.1. LIMIT

LIMIT consists of a framework for discovering the maximum clock-frequency at which an FPGA application can run without error. We use LIMIT to conduct experiments exploring how the optimal clock-frequency of FPGA circuits varies over time in different operating conditions.

Fig. 1 gives an overview of our LIMIT system. LIMIT consists of an FPGA design which feeds pre-computed random data into an arbitrary target 'Test Circuit'. All the outputs of the test circuit are compared with pre-computed results to check for the presence of errors. A host computer adjusts clock-frequency of the test circuit and monitors errors. Using this set-up, we can pragmatically determine optimal clock-frequencies which balance error occurrences with performance given the input samples. Using random inputs effectively gives us an 'average-case' processing load. To enable us to maximise the rate of processing, we store pre-computed inputs and outputs for our test circuits directly into our FPGA designs. This allows us to eliminate data transfer bottlenecks which would arise if the Host PC sent/received data for validation.

We use LIMIT to examine the behaviour of optimal operating clock-frequency for test circuits using two classes of experiments: continuously adaptive and long-term stability. Our continuously adapting experiments explore how optimal clock-frequency behaves over time by continuously re-evaluating the optimal clock-frequency. In our long-term stability experiments we look for optimal clock-frequencies which allow error free operation for an

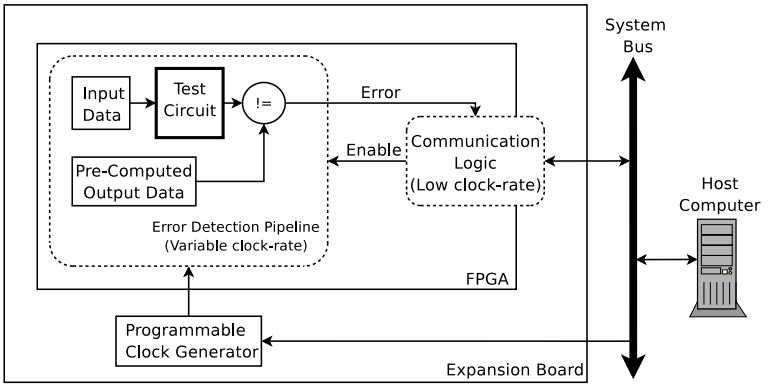

Fig. 1. Overview of our LIMIT system used to determine dynamic clock frequencies which allow error-free operation of test circuits.

extended period of time. For both classes of experiment, we use a common set-up described in Section 4.3.

4.1.1. Continuously adapting experiments

In our continuously adapting LIMIT experiments we attempt to determine:

- How different areas of an FPGA affect optimal clock-frequency,
- How optimal clock-frequency varies over short periods of time,
- The effect of different environments on the optimal clock-frequency.

These aspects are monitored by repeatedly finding the highest clock-frequency of a circuit.

Our algorithm for finding current error-free frequencies is a binary search algorithm. This is implemented by the host PC indicated in Fig. 1 and operates as follows:

(1) Set variable clock-frequency input to the mid-point of our binary search range.
(2) Run the test circuit continuously for 10 s, checking all outputs for error.
(3) If no errors are found, we make the current clock-frequency our binary search lower bound, otherwise in case of an error the current clock-frequency becomes our upper bound.
(4) Repeat steps 1–3, ten times to give us a clock-frequency result of comparable accuracy to the programmable clock-generator in our FPGA board.

We re-evaluate optimal clock-frequency using this method repeatedly for an hour to examine the behaviour of optimal clock-frequency over time. We perform this experiment for every circuit in each of the four quadrants of the FPGA and in all our different environmental conditions.

4.1.2. Long-term stability experiments

In our long-term stability LIMIT experiments we explore optimal clock-frequencies for test circuits that are usable for an extended period of time. We do this differently from our continuously adapting experiments to mitigate the effects of repeatedly changing the clock-frequency during execution due to continuous binary searching.

Our goal in these experiments is to see how long it takes to converge on an optimal clock-frequency which allows 3 h of error free operation. We determine this using the following procedure:

(1) Perform a binary search, as in our continuously adapting experiments, to find an initial optimal clock-frequency.
(2) Run the test-circuit continuously at this optimal clock-frequency for up to 3 h, stopping if an error occurs.
(3) If an error is detected, we set the upper bound of our binary search to the current "unsafe" frequency.
(4) Repeat steps 1–3, until we find an optimal clock-frequency which allowed an error free, 3 h run.

We repeat this experiment for each of our test circuits without varying the environment or circuit location in the FPGA.

4.2. AutoTEA

We demonstrate AutoTEA, which to our knowledge is the first system for automatically adding dynamic clock-frequency features to arbitrary FPGA designs. For FPGAs, automated design processes are particularly important as the zero cost of re-programming a device encourages rapidly evolving designs.

Our AutoTEA system is based on the TEAtime methodology developed at the University of Rhode Island. In TEAtime a "false critical-path" is inserted into a design and continuously checked for errors in parallel with running the main design. This false critical-path is a one-bit wide version of the longest flip-flop-to-flip-flop path in the main design with additional delay. The idea is that the false critical-path, with its extra delay, will fail before the main design so clock-frequency can be adjusted based on observing errors in the false critical-path.

We use TEAtime as a base for our automated system as it does not require detailed interpretation or modification of a design to which it is applied. All an automated version of TEAtime needs to know is the construction of the longest flip-flop-to-flip-flop path. Modification of the design is not necessary as the false critical-path and checker logic just needs to be attached to a system which can adjust the clock-frequency.

To facilitate automated operation, our AutoTEA system diverges from the original TEAtime in the way it constructs a false critical-path. In the original TEAtime system a false critical-path is created manually by constructing a one-bit wide version of the real critical-path in a design. This path is then adjusted to be longer than the real critical-path using results from detailed structural simulation.

In AutoTEA we construct our false critical-path using a chain of inverters, the length of which is based on results from automated timing analysis and optionally from results of running on real hardware with sample inputs. Using a chain of inverters both simplifies automated construction of the false critical-path, and allows easy stress testing in checker logic by simply alternating 1/0 inputs.

Our AutoTEA process for determining the number of inverters necessary to achieve a delay which is as close as possible to the real critical-path delay plus a configurable margin is illustrated in Fig. 2. Our current implementation uses a feed-back loop to converge onto a suitable false critical-path length. We use a loop as adding inverters to the false critical-path and placing them using the Xilinx place-and-route tools can cause unpredictable variations

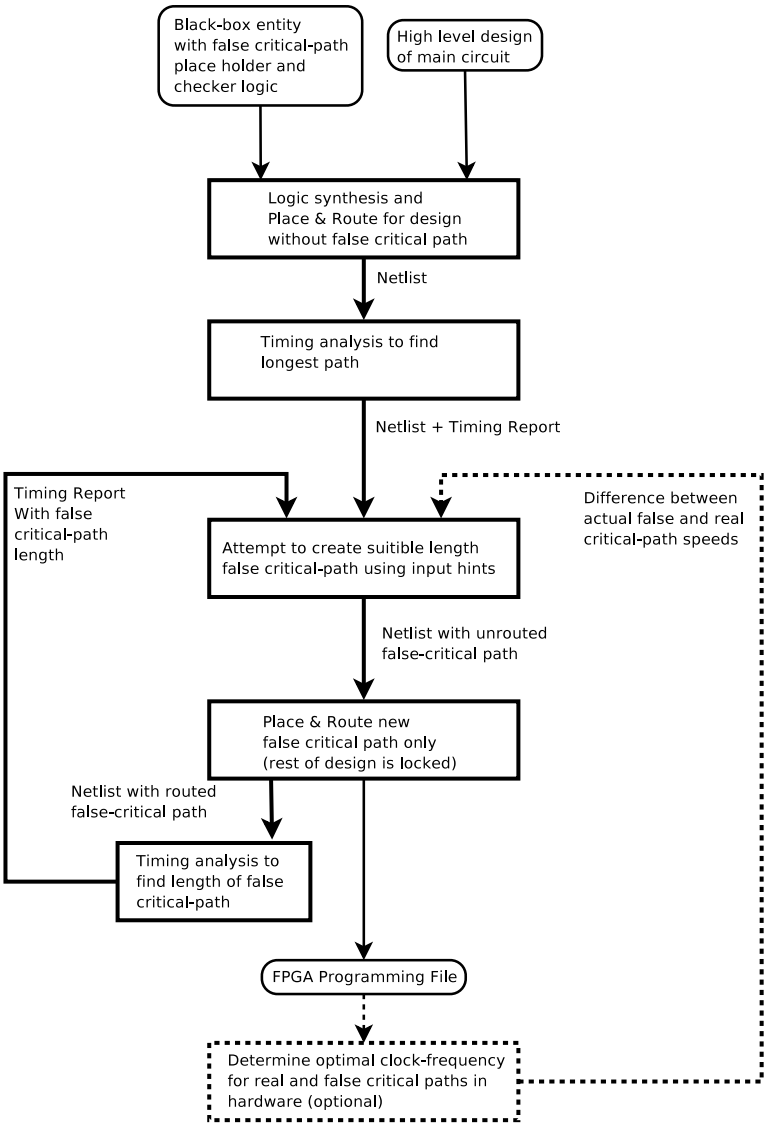

Fig. 2. AutoTEA process for creating a false-critical path suitable for predicting error in arbitrary FPGA designs.

due to other elements in the design. The termination condition for this loop is when changing the number of inverters or timing constraints on the path does not yield an improvement over previously generated paths. Optionally the path can further be tuned using results from testing the resulting bitfile in real hardware using methods similar to LIMIT.

To incorporate our AutoTEA system into an FPGA application all we need to do at the design level is to include a reference to a black-box entity. This entity provides a connection point to the AutoTEA error detection logic and a place holder for the false critical-path that is expanded after the main design has been place-and-routed. Inputs to the black box consist of clock pins, reset and error signals. The error signal is asserted if any failures are detected in the false critical-path and remains high until reset is asserted.

To test our AutoTEA system, we run experiments to determine if our generated false critical-paths are suitable for adapting clock-frequency without allowing errors in test circuits. We show an overview of our experimental set-up in Fig. 3. As with our LIMIT system, we check all outputs from our test circuit for errors. This allows us both to discover if AutoTEA has been successful and to use our AutoTEA facility of fine tuning the false critical-path based on running in actual hardware.

We test that our false-critical path reacts appropriately to environmental changes by actively changing the environment during execution over a 1 h period. We alter the environment manually, based on our chosen environmental conditions described in Section 4.3. We change conditions once every 20 min and expect to see, for example, that moving from warm to cold environments causes an increase in clock-speed without introducing any errors.

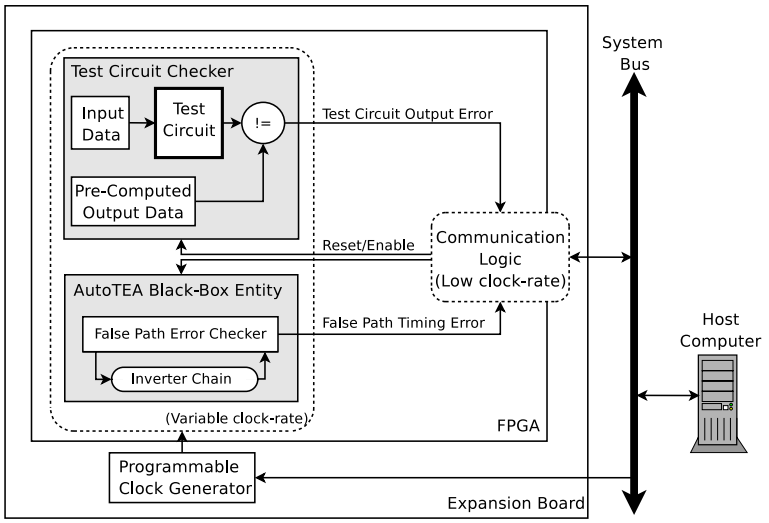

Fig. 3. Overview of AutoTEA experimental set-up to compare AutoTEA predicted and true optimal clock-frequencies for various test circuits.

4.3. Common experimental set-up

For both our LIMIT and AutoTEA experiments we use a common set of test circuits, environment conditions and the same hardware platform.

Our hardware platform consisted of an RC1000 PCI board with a Xilinx Virtex 1000 FPGA hosted in a standard desktop PC running the Linux operating system.

In our experiments FPGA clock-frequency is controlled exclusively by a host-software-programmable oscillator on the RC1000 board. While precise data on the accuracy of the programmable clock generator were not available to us, we were able to examine the board's software driver source code. From this we concluded that the oscillator was accurate to at least 1 MHz, and in practise it would normally be closer to 0.1 MHz with a settable range of 400 KHz–100 MHz.

By running the PC with its case removed we can vary our environmental conditions for the FPGA between externally lamp heated to approximately 70 °C, fan cooled and room temperature. We use fan cooled and room temperature conditions as these are the most common operating conditions in practical use for FPGA devices. We also use lamp heating to push the FPGA to its operating limits.

We use four carefully chosen designs for our test circuits, to give us a wide range of design characteristics:

- *DCT* – A 1D, 8-point DCT generated using the Xilinx CoreGen utility. This circuit is tailored to the target FPGA and thus achieves high performance.
- *DES* – An implementation of DES encryption from the www.opencores.com website. Encryption oriented circuits are of interest as they by their nature have a high amount of chaotic switching.
- *MULT0* – A non-pipelined, single-cycle ($32 \times 32 \rightarrow 64$ bit) multiplier generated with the JHDL hardware description language. This low speed circuit demonstrates the effects on dynamic frequency in circuits with very long combinatorial paths. Some of our past work suggests that long combinatorial paths may cause a large amount of glitching which could severely hinder the maximum safe clock-frequency [13].
- *MULT1* – As MULT0 but with pipelining.

5. Results

5.1. LIMIT results

Figs. 4 and 5 show results from our continuously adapting LIMIT experiments, and Table 2 provides results from our long-term stability experiments. Fig. 4 shows a summary of optimal clock-frequencies sampled in our continuously adapting experiments for different circuits, FPGA locations, and environmental conditions. Fig. 5 shows a detailed view of optimal clock-frequency over time for

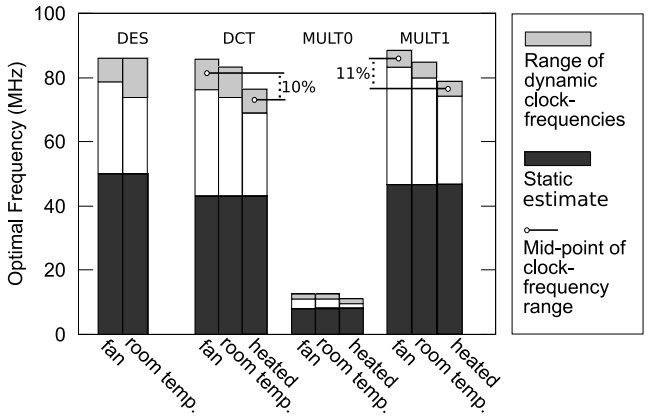

Fig. 4. Comparison of experimentally determined and vendor estimated optimal clock frequencies for test circuits under varied environmental conditions. Note that heated DES exceeds the operating conditions of the FPGA.

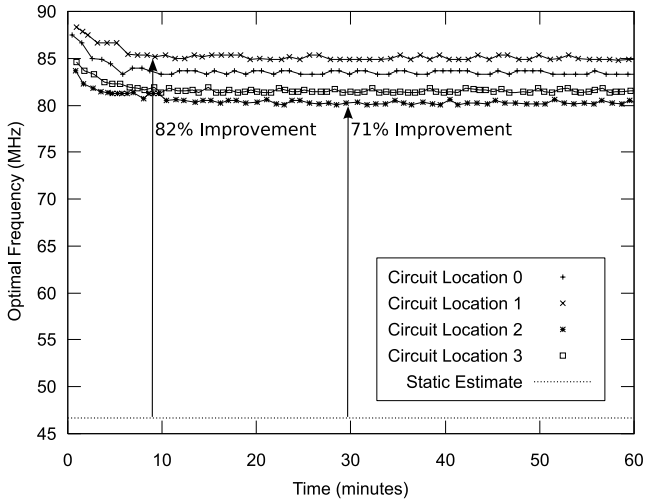

Fig. 5. Optimal clock-frequency over time for a pipelined multiplier running in four different locations on an FPGA.

one of our test circuits, and is typical of results for the other circuits in all environmental conditions. Table 2 shows the maximum clock-frequencies suitable for a 3 h run we were able to achieve with a single instance of each circuit and how long it took to find.

In addition to collecting data on optimal clock-frequency in our continuously adapting experiments, we also capture failed outputs. We use these failed outputs to construct histograms of how often bits in the hardware output are failing. Fig. 6 shows two typical failure patterns for DES and MULT0 instances. For the DES hardware, failures are fairly uniform reflecting the uniform nature of a DES hardware implementation, and the MULT0 errors are skewed, again reflecting the path lengths in a hardware multiplier. In future we could use this information to optimise these circuits for use in dynamic clock-frequency system for example by putting extra error detection logic on bits more likely to fail, or by applying typical-case optimisations suitable for better than worst-case design [9].

From our LIMIT experiments we make the following observations:

- The benefits of a dynamic versus static clock-frequencies for arbitrary FPGA designs are large; from 33.25 to 86.31%. This is particularly significant as these results were achieved with no effort made to optimise our test circuits.
- Environmental conditions do play a role in determining the maximum safe operating frequency of the FPGA, but the difference between a fan cooled and lamp heated circuit is small at around 10%.
- Our continuously adapting experiments indicate that location in an FPGA does not seem to make much of a difference. While we do see a range in performance for our different circuit instances, these do not appear to correlate to FPGA location. They can instead be attributed to random deviations in each instance caused by place and root algorithms.

Table 2
Results of using LIMIT to dynamically determine clock-frequencies allowing test circuits to operate for 3 h without error

Circuit	Static frequency estimate (MHz)	Final stable dynamic frequency (MHz)	Performance improvement (%)	Time to find stable frequency (min)	First dynamic frequency (MHz)	Drop in frequency (%)
DES	42.6	72.45	70.07	20.88	76.56	5.37
DCT	42.95	80.02	86.31	4.55	81.27	1.53
MULT0	8.21	10.94	33.25	1.53	10.94	0.00
MULT1	46.68	81.13	73.80	30.03	84.38	3.85

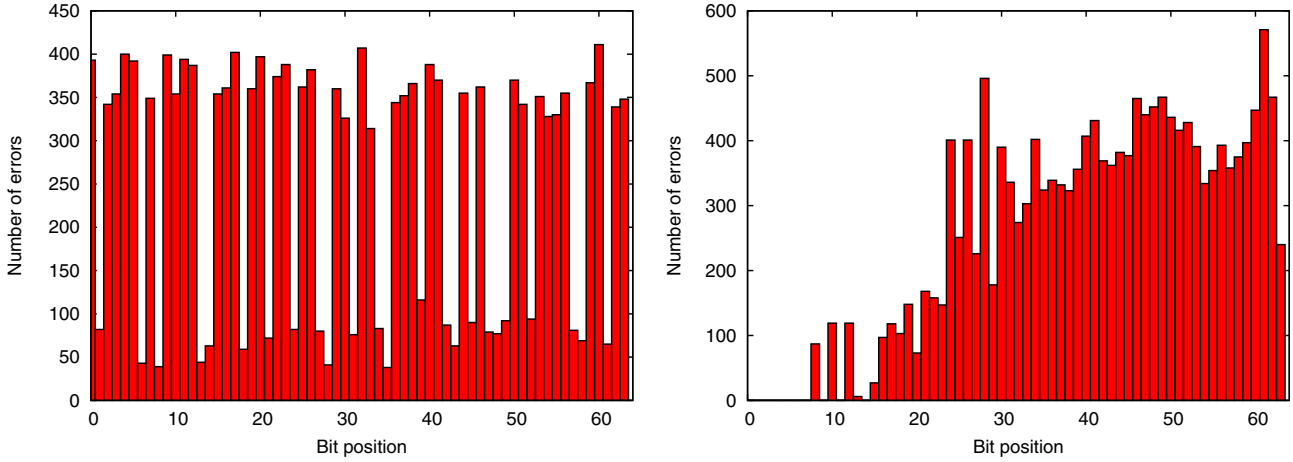

Fig. 6. Histograms showing bit errors in computation outputs due to over-clocking reflect circuit structure in DES (left) and MULT0 (right).

- Our long-term stability experiments show that the major changes in environment, which lower the maximum optimal clock-frequency, seem to occur fairly quickly, at most within 30 min.
- Both our long-term stability and continuously adapting experiments show that optimal clock-frequencies are fairly constant over time. Our long-term stability experiments showed that even after several hours of continuous operation drops of less than 6% in optimal clock-frequency are seen.

5.2. AutoTEA results

Fig. 7 shows graphs of optimal clock-frequency found by binary search for the false and real critical-paths in circuits to which AutoTEA is applied. For these experiments, we enable AutoTEA to fine tune the path length using feedback from both timing analysis and by automatically testing the design on real hardware. AutoTEA is configured to target creating false critical-paths 25% slower than the real critical path for each test circuit. We empirically determined this margin to provide a suitable safety margin for all test circuits given the lack of resolution for controlling critical path lengths using the Xilinx place-and-route tools.

From our results we make the following observations:

- AutoTEA optimal false-path frequencies are always below the optimal frequencies determined through testing of the test circuits. In other words AutoTEA frequencies are below those at which errors occur in the test circuits.
- AutoTEA can react well to changes in environment. Particularly in the DCT and MULT1 examples, we can see that moving from normal to cool and cool to hot environments causes safe increases and decreases in clock-frequency, respectively.
- In most cases the AutoTEA false-path frequencies are considerably above statically estimated frequencies for the test circuits.

From these observations we can see that AutoTEA shows promise for determining dynamic clock-frequencies for FPGA applications to run at without error by taking advantage of run-time conditions.

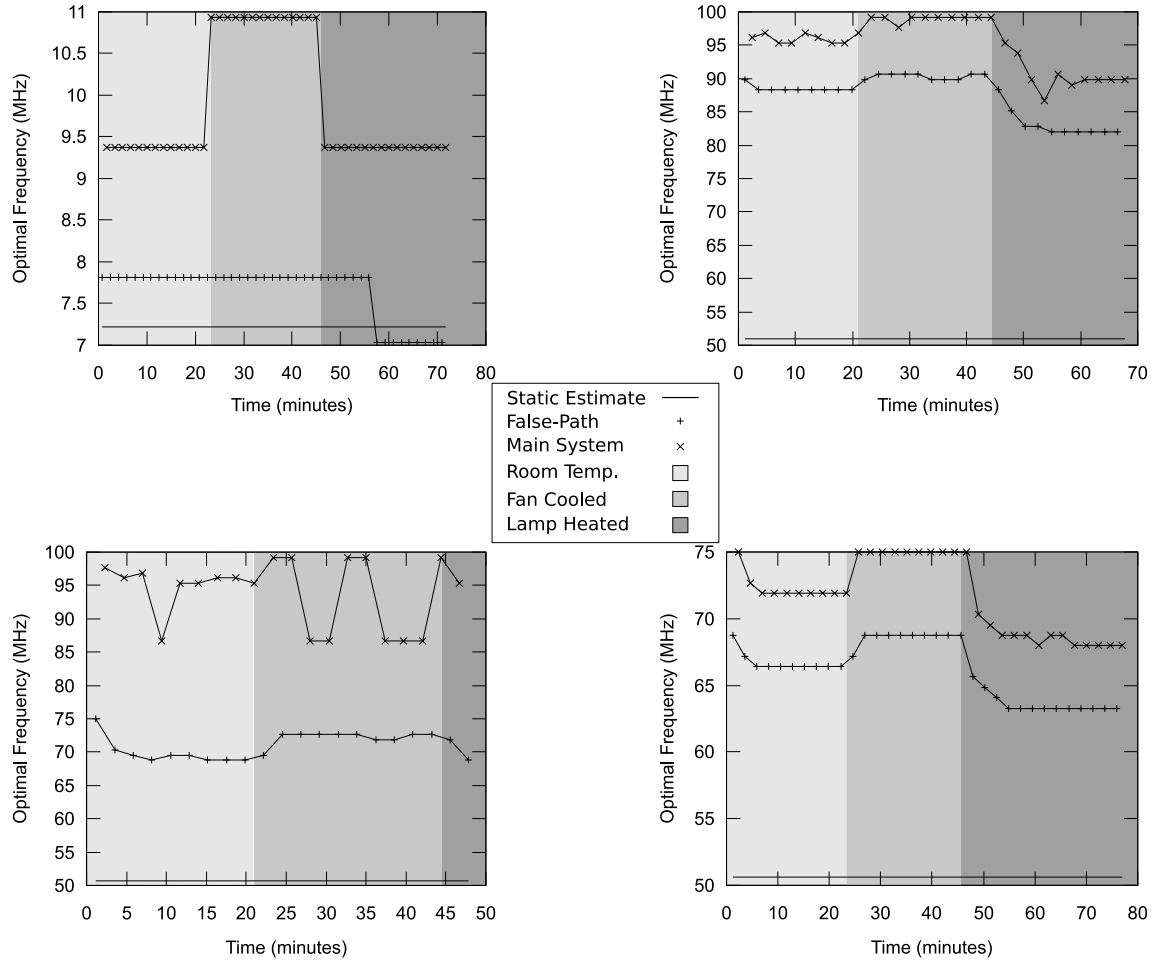

Fig. 7. AutoTEA optimal clock frequencies compared with true optimal frequencies for MULT0 (top-left), DCT (top-right), DES (bottom-left), MULT1(bottom-right). Note that heated DES exceeds the operating conditions of the FPGA.

6. Conclusion

In this paper, we show a methodology and implementations of FPGA systems with dynamic clock-frequencies. AutoTEA enables automatic addition of dynamic clock-frequency to arbitrary FPGA designs. As a consequence, we exploit the performance benefits of dynamic clock-frequencies over a static clock.

Future work includes extensions of our AutoTEA and LIMIT systems to support voltage scaling [15] and ways of further improving performance of applications using dynamic clock-frequencies. Furthermore, dynamic reconfiguration [16] can provide improvements of dynamic clock-frequency systems in FPGAs. For example we can use dynamic reconfiguration to move a circuit around FPGA fabric to reduce the maximum temperature it reaches allowing a higher clock-frequency, and/or we could have a circuit which reconfigures itself depending on data inputs to optimise its maximum clock-frequency. Our AutoTEA and LIMIT systems also provide a starting point for generating tighter static clock-frequency estimates for FPGA designs by incorporating information from real run-time performance of a device and/or application.

References

[1] V. Betz, J. Rose, A. Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers, 1999, Section 2.2.5.

[2] D. Grunwald, P. Levis, K.I. Farkas, C.B. Morrey III, M. Neufeld, Policies for Dynamic Clock Scheduling, USENIX Symposium on Operating System Design and Implementation, October 2000, pp.73–86.

[3] B. Colwell, The zen of overclocking, IEEE Computer vol. 37 (2004) 9–12.

[4] A.K. Uht, Achieving Typical Delays in Synchronous Systems via Timing Error Toleration, University of Rhode Island, Tech. Rep. 032000-0100, March 2000.

[5] T. Austin, D. Blaauw, T. Mudge, K. Flautner, Making typical silicon matter with Razor, IEEE Computer vol. 37 (2004) 57–65.

[6] T. Austin, DIVA: a dynamic approach to microprocessor verification, Journal of Instruction Level Parallelism vol. 2 (2000).

[7] M. Olivieri, A. Trifiletti, A.D. Gloria, A low-power microcontroller with on-chip self-tuning digital clock-generator for variable-load applications, IEEE International Conference on Computer Design (1999) 476.

[8] A.K. Uht, Going beyond worst-case specs with TEAtime, IEEE Computer 37 (2004) 9–12.

[9] T. Austin, V. Bertacco, D. Blaauw, T. Mudge, Opportunities and challenges for better than worst case design, in: Proceedings of the ASP-DAC, Shanghai, China, January 2005.

[10] G.M. Link, N. Vijaykrishnan, Hotspot Prevention Through Runtime Reconfiguration in Network-On-Chip, DATE '05, Munich, March 2005.

[11] S.P. Mohanty, N. Ranganathan, Energy-efficient datapath scheduling using multiple voltages and dynamic clocking, ACM Transactions on Design Automation of Electronic Systems Vol. 10 (No. 2) (2005) 330–353.

[12] C. Weaver, F. Gebara, T. Austin, R. Brown, Remora: A Dynamic Self-Tuning Processor, University of Michigan, Tech. Rep. CSE-TR-460-02, July 2002.

[13] S.J.E. Wilton, S.S. Ang, W. Luk, The Impact of Pipelining on Energy Per Operation in Field-Programmable Gate Arrays, Field Programmable Logic and Applications, LNCS 3203, Springer, 2004, pp. 719–728.

[14] T.D. Burd, T.A. Pering, A.J. Stratakos, R.W. Brodersen, A dynamic voltage scaled microprocessor system, IEEE Journal of Solid-State Circuits 35 (11) (2000) 1571–1580.

[15] C.T. Chow, L.S.M. Tsui, P.H.W. Leong, W. Luk, S.J.E. Wilton, Dynamic voltage scaling for commercial FPGAs, Proceedings of the IEEE International Conference on Field-Programmable Technology (2005) 173–180.

[16] J. Liang, R. Tesier, D. Goeckel, A dynamically-reconfigurable, power-efficient turbo decoder, Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (2004).