Reconfigurable Design with Clock Gating

W.G. Osborne, W. Luk, J.G.F. Coutinho and O. Mencer
Department of Computing
Imperial College, London
{wgo, wl, jgfc, o.mencer}@imperial.ac.uk

Abstract

This paper describes an approach for developing energy-optimized run-time reconfigurable designs which benefit from clock gating. The approach is applied to two techniques: multiplexer-based reconfiguration, and reconfigurable word-length optimization. The conditions under which this approach would be preferable to bit-stream reconfiguration are derived. Various case studies, such as ray tracing, guitar string simulation and matrix–vector multiplication, are used to illustrate this approach.

1 Introduction

The sharp rise in integrated circuit fabrication costs and the need for fast time-to-market and for in-field upgrades have accounted for the increasing popularity of reconfigurable devices such as Field Programmable Gate Arrays (FPGAs). The flexibility of reconfigurable devices, however, comes with overheads in latency, area and power consumption—for instance it has been shown [9] that the dynamic power consumption for FPGAs can be up to 14 times worse than that for Application-Specific Integrated Circuits (ASICs).

Two methods for reducing power consumption for reconfigurable devices have been proposed. The first method involves clock gating: disabling the clock for the inactive regions in a design to minimise signal transitions and hence dynamic power [8, 17]. The second method involves reconfiguring an FPGA with multiple bit-streams, one for each configuration, so that a small device with low power consumption can be used [1]. However, there are delay and energy overheads for reconfiguration with bit-streams.

The aim of this paper is to present a novel approach for reconfiguring designs. The approach involves two run-time reconfiguration techniques: multiplexer-based reconfiguration, and reconfigurable word-length optimization. Clock-gated reconfiguration with physical multiplexers is faster than reconfiguration by adopting a new FPGA bit-stream, but it does not provide the area advantage offered by bit-stream reconfiguration. We also derive the conditions under which the proposed approach involving clock-gated reconfiguration would require less energy than bit-stream reconfiguration.

While much work [14, 21] in this field reports that clock gating reduces power consumption, there is at least one paper [5] which reports the contrary. Moreover, most results from previous work are obtained by vendor's simulators such as XPower. In contrast, all the power consumption results in this paper are obtained by measuring the current and voltage of various applications.

To summarise, the innovative elements of the proposed approach are:

1. Two methods, involving multiplexing and word-length optimization, for developing run-time reconfigurable designs based on clock gating and bit-stream reconfiguration (Section 3).

2. Derivation of the conditions under which clock-gated reconfiguration should be used in preference to bit-stream reconfiguration (Section 4).

3. An implementation of our approach for various case studies: ray tracing, string simulation, matrix–vector multiply and inner product (Section 5 and 6).

2 Related work

Zhang *et al.* [21] analyse the effect of clock gating on power efficiency showing that FPGAs, although not as efficient as ASICs, can achieve significant power reductions. Clock gating may not always be the most energy-efficient solution even if it is the most power-efficient solution in some cases.

Cadenas *et al.* [5] implement a clock gating technique in a pipelined Cordic core with the goal of reducing bit-switching. They do not obtain power improvements using a Cordic pipelined design. We explore optimizations both with clock gating and bit-stream reconfiguration, and use word-length analysis techniques to improve results. In some

cases we set part of the input to zero to reduce dynamic power consumption if clock gating has little effect, to reduce the area overhead.

An alternative to run-time bit-stream reconfiguration is based on using multiplexers and demultiplexers for time multiplexing designs [6, 11, 20]. This method supports fast reconfiguration but requires a large area. However, both this approach and the clock gating approach require configurations known at design time because they need to be installed on the chip at the start, while bit-stream reconfiguration can be used when downloading new configurations.

In control-flow analysis, Styles and Luk use information about branch frequencies to reduce the hardware used for implementing branches that are infrequently taken [18]. Since program executions often change their behaviour based on input data, the circuit needs to evolve at run time in order to keep the error to a minimum.

Bondalapati and Prasanna reconfigure the circuit at run time and show that it can be used to reduce execution time by up to 37% [4].

Our approach not only takes into account accuracy requirements, but also models the difference between reconfiguration techniques to minimize the energy requirements of the system.

3 Reconfigurable design with clock gating

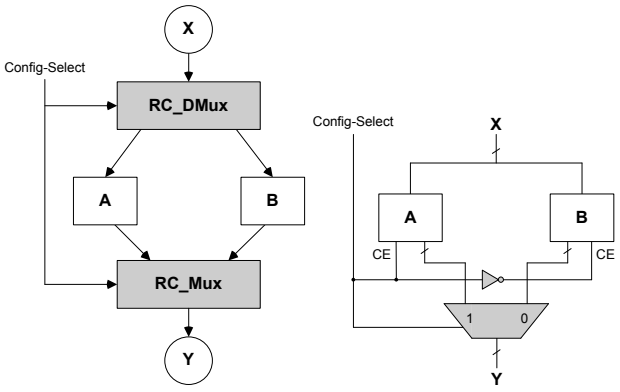
This section begins with a short overview of clock gating in FPGAs (Section 3.1). The application of clock gating to two reconfigurable methods, multiplexer-based reconfiguration (Section 3.2) and reconfigurable word-length optimization (Section 3.3), will then be described.

3.1 Clock gating in FPGAs

Clock gating provides a means of reducing switching activity, and hence dynamic power consumption, by disabling registers from reading external data; so they just keep their contents when they are inactive.

Clock gating is supported by current FPGA devices. For instance, the Xilinx Virtex series of devices contain a clock gating block BUFGCE, which provides a way to turn on or off a global clock net [8], resulting in power saving both from the clock net and from the registers attached to it.

However, the number of such clock gating blocks may be limited—for instance there are only 16 BUFGCE on a Xilinx XC2VP30 FPGA. If we need more, then we have to use the clock-enable input to gate the register in each configurable logic element. To support reconfigurable word-length optimization, for instance, one requires more clock gating resources than the specialised clock gating blocks that current devices provide, so we make use of the clock-enable input of configurable logic elements.



(a) multiplexer-based reconfiguration model (b) clock-gated implementation

Figure 1. A network modelling a design that can behave either as block A or block B, depending on the control blocks RC_DMux and RC_Mux; and its clock-gated implementation.

It is possible that several components can be grouped together so that they share a single clock gating control, such that the number of clock gating elements matches the number of specialised clock gating blocks from a given device. For simplicity, the designs here do not adopt this approach.

3.2 Multiplexer-based reconfiguration

Previously we reported a model [11] and the associated development tools [12] for reconfigurable designs. In this model, a component that can be configured to behave either as A or as B is described by a network with A and B connected between two control blocks. The control blocks, RC_DMux and RC_Mux, route the data and results from the external ports x and y to either A or B at the desired instant, as shown in Figure 1(a).

Each control block can then be mapped either into a real multiplexer or demultiplexer to form a single-cycle reconfigurable design, or into virtual ones which model the control mechanisms for replacing one configuration by another [11].

Here we propose a simple extension to this approach in the case that the control blocks are mapped into real multiplexers and demultiplexers controlling data-flow to the reconfigurable regions. Since only one of the reconfigurable regions will be active at any one time, we can turn off the clock in those reconfigurable regions which are inactive, as shown in Figure 1(b). A reconfiguration controller is used to activate and deactivate successive configurations, and it would also activate and deactivate the clock gating as well.

In case block A and B in Figure 1 are the same apart from different constant coefficients, one can either use constant propagation techniques to optimize A and B, or time-

multiplex between different constant coefficients with a block that supports both A and B. Usually the latter is more area efficient, but at the expense of performance. Some examples will be presented in Section 6.1, comparing the time-multiplexed constant coefficient approach and bit-stream reconfiguration.

The development flow for producing reconfigurable designs with clock gating or with multiple bit-streams can be summarised in the following steps.

1. The reconfigurable regions in the design are identified, with control blocks connecting together the possible configurations for each reconfigurable component, together with the sequence of conditions for activating a particular configuration for each control block.

2. The activation sequence is used to determine the successive configurations which are managed by a reconfiguration controller in hardware or software, adopting a centralised or a distributed implementation.

3. For designs with clock gating or with partial bit-streams, identical regions in successive configurations are extracted to minimise area or to minimise the reconfiguration overhead. An efficient algorithm based on weighted bipartite graphs [16] has been proposed to automate this step.

4. Each configuration is optimized by techniques such as constant propagation, since inputs which stay constant throughout a configuration can result in additional optimization opportunities.

5. For reconfigurable designs with clock gating, there are further opportunities for optimization with the multiplexers and demultiplexers; they can, for instance, be implemented by multiplexer trees or by decoders.

6. For reconfigurable designs with partial bit-streams, there are further opportunities for optimization involving techniques such as support for heterogeneous architectures [3] and for online routing [15].

3.3 Word-length optimization

Another technique that can benefit from the clock gating method described in Section 3.1 is word-length optimization: reducing precision of variables such that power and energy consumption can be minimised. This can be achieved by various approaches; below we present one based on a method that combines word-length optimization with an accuracy-guaranteed technique to reduce power consumption while maintaining error constraints for the outputs [10].

Every variable involved in arithmetic operations has an associated range and precision. Our approach to range and

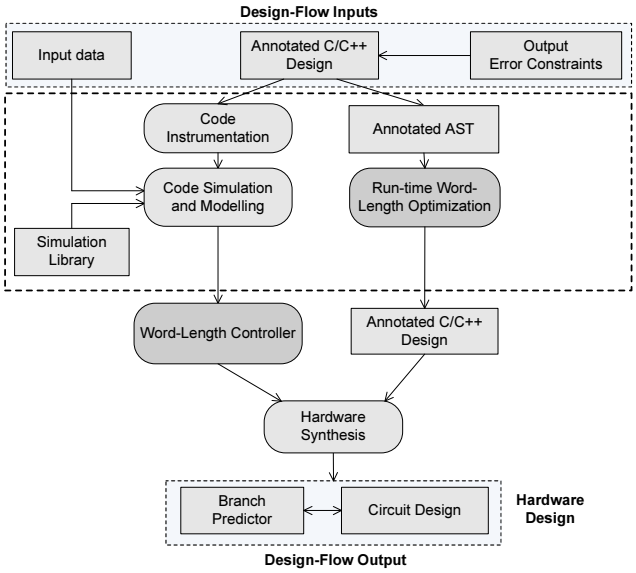


Figure 2. Outline of a word-length optimization tool. AST denotes Abstract Syntax Trees.

precision optimization is accuracy-guaranteed, which means that any operations performed by the static analysis are guaranteed to produce a specified accuracy irrespective of the input data. Since these results will be conservative, dynamic analysis can be used so that the results are guaranteed for a specific set of input data.

Our range analysis uses a combined Interval/Affine analysis. Input ranges are propagated to produce ranges for the intermediate and output variables. Affine arithmetic is used because it maintains correlations between the ranges, resulting in smaller output ranges if a variable is used in several places.

Precision analysis involves performing arithmetic on errors instead of numeric values to calculate the worst-case error on the output from errors on the inputs. Range analysis is performed first because the results are required in these calculations. This ensures that the accuracy of calculations is guaranteed.

To control the range and precision optimization to reduce power consumption, we use a controller, implemented to use as little power as possible. We adopt a technique involving control-flow analysis, although the controller could be replaced for different applications.

Figure 2 shows the outline of a word-length optimization tool. Combined with control-flow analysis, we reduce word-lengths to minimize power consumption, for instance by clock gating part of the registers that remain inactive due to reduced accuracy. Providing constant zero to inputs of the inactive regions can achieve a similar effect.

4 Deriving reconfiguration conditions

In many systems such as ray tracing and feature extraction, the algorithm's control-flow is dependent on input data. This means that the functionality of the system will change as input data changes, so static word-length optimization will no longer have any effect because when the system functionality changes, the word-lengths will need to change.

Based on stimuli which may come from outside the system or may be generated by the system, the word-lengths will adapt in such a way as to reduce the power consumption of the system while keeping the error to a minimum. To change the word-lengths at run time, we can use one of two methods: bit-stream reconfiguration or clock gating.

If clock gating is used, there is a static power overhead because the entire design resides on the chip even if not all of it is active all the time. Bit-stream reconfiguration does not have this overhead, but has a high reconfiguration overhead; for this reason we use a model to determine the most efficient reconfiguration strategy. We will focus on time and power consumption results since area is application-dependent.

4.1 Speed considerations

First we define the following terms:

- t_{eb}: time spent on execution between successive bit-stream configurations.
- t_{eg}: time spent on execution between successive clock gating configurations.
- t_b: time spent on reconfiguring resources between successive execution.
- t_g: time spent on reconfiguring gated clock elements between successive execution.

The total run-time with reconfigurable clock-gating:

$$\sum t_{eg} + \sum t_g$$

The total run-time with bit-stream reconfiguration:

$$\sum t_{eb} + \sum t_b$$

In general $t_g < t_b$ since reconfiguring the clock gating can take only a few cycles, but $t_{eg} > t_{eb}$ since a clock-gated design is larger and usually slower than the corresponding bit-stream configuration. Hence the total run time depends on which term dominates. For instance, if execution time is short compared to reconfiguration time, then designs with clock gating are likely to be faster than the corresponding designs with bit-stream reconfiguration.

In general a design using bit-stream reconfiguration is likely to have a higher clock speed because it has a smaller area as shown in Figure 6(b). It is possible to increase the clock speed of a design with clock gating by using a different clock speed for each block (Figure 1), by having multiple clock domains in the design.

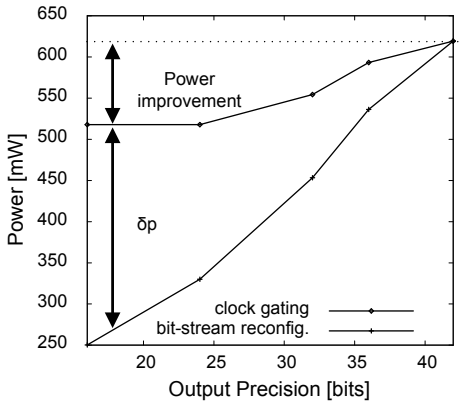


Figure 3. Power consumption for the guitar string simulation benchmark.

4.2 Energy consumption considerations

Figure 3 shows the power savings of a guitar string simulation benchmark with simulated inputs. The *clock gating* graph shows the minimum power consumption achievable of the design at various precisions; overheads such as power dissipation of the controller are not included. Since the entire design, including the inactive parts, is still on-chip, the design is larger than the one with minimum logic resulting from bit-stream reconfiguration; hence the clock-gated design consumes more power during execution than the design with bit-stream reconfiguration for a given precision. This difference is shown in Figure 3 as δp.

However, bit-stream reconfiguration may consume more power during the reconfiguration process than the clock-gated design, so we need to analyse the situation in more detail to derive the conditions under which one method produces better designs than the other. Our analysis is based on energy consumption, which takes into account execution time and reconfiguration time, as well as the power consumption during execution and the power consumption during reconfiguration.

First we define the following terms:

- p_{eb}: power spent on execution between successive configurations with multiple bit-streams, reconfiguring all relevant resources;
- p_{eg}: power spent on execution between successive configurations when reconfiguring just the clock gating elements;

- p_b: power spent on reconfiguring bit-streams between successive execution;
- p_g: power spent on reconfiguring gated clock elements between successive execution.

From Figure 3, we know that total energy with reconfigurable clock gating:

$$\sum(t_{eg} \times p_{eg}) + \sum(t_g \times p_g)$$

Total energy with a reconfigurable design with bit-streams:

$$\sum(t_{eb} \times p_{eb}) + \sum(t_b \times p_b)$$

$\sum(t_g \times p_g)$ is usually small, so we ignore it. Given

$$\begin{aligned} \delta p &= p_{eg} - p_{eb} \\ t_{gb} &= t_{eg} - t_{eb} \end{aligned}$$

(since δp can be obtained from Figure 3) the clock gating designs will require lower energy dissipation than the corresponding designs with bit-stream reconfiguration when:

$$\sum(t_{eg} \times \delta p) + \sum(t_{gb} \times p_{eb}) < \sum(t_b \times p_b)$$

If $\sum(t_{gb} \times p_{eb})$ is small compared to other terms, then we can neglect it.

Assuming that there are n reconfigurations, then:

$$\sum(t_b \times p_b) = n \times t_b \times p_b$$

and we can define t_{ae}, the average time spent on execution between reconfiguration, by:

$$n \times t_{ae} = \sum t_{eg}$$

Hence:

$$t_{ae} < (t_b \times p_b)/\delta p \quad (1)$$

for designs employing clock gating to be used. From Figure 3, given t_{ae}, which depends on the application, we can calculate the average precision required.

In addition to determining the condition that favours clock gating in achieving a low energy design, the above model can be used in various ways. For instance, it enables us to study quantitatively how energy consumption varies with the frequency of reconfiguration: more frequent reconfiguration favours clock-gating, while less frequent reconfiguration favours bit-stream reconfiguration.

5. Related considerations

We now discuss several considerations related to the proposed approach.

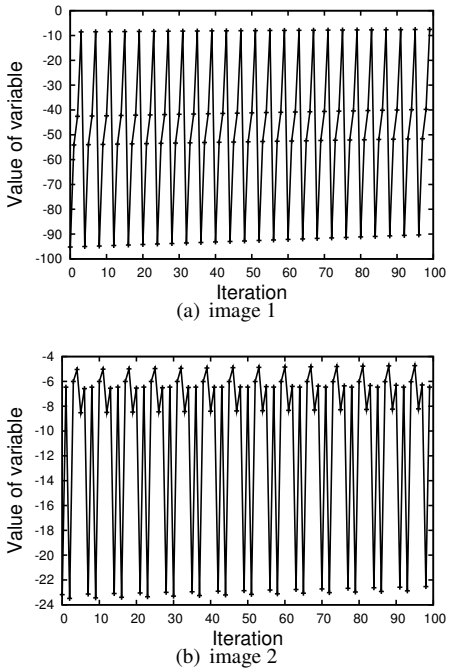


Figure 4. Range variation for different images on the ray tracer.

5.1 Loops and convergence

If the number of iterations of a loop is not known at compile time, a conservative estimate must be used. If the number of iterations varies by a large amount, the accuracy of the computation may be larger than required. For this reason we allow the word-lengths of variables to change at run time which usually reduces energy consumption.

If the solution converges, the accuracy of the variables at the start of the computation can have a lower accuracy than the variables at the end. The variables will gradually need to increase in accuracy. In this case clock gating will be more appropriate since the interval between reconfigurations with different accuracy tends to be small, and bit-stream reconfiguration can involve a large overhead with little gain in area.

5.2 Input range analysis

In a recent paper [13] we show that input range analysis can reduce the word-length of variables without adversely affecting the result. As an example, Figure 4 shows what happens when different scenes are processed by a ray tracer: simpler scenes are likely to have a narrower range. Of the two graphs shown, the variables in the simpler scene (image 2) require 2 fewer bits than those in the more complex scene (image 1). After analysing the input ranges, we can

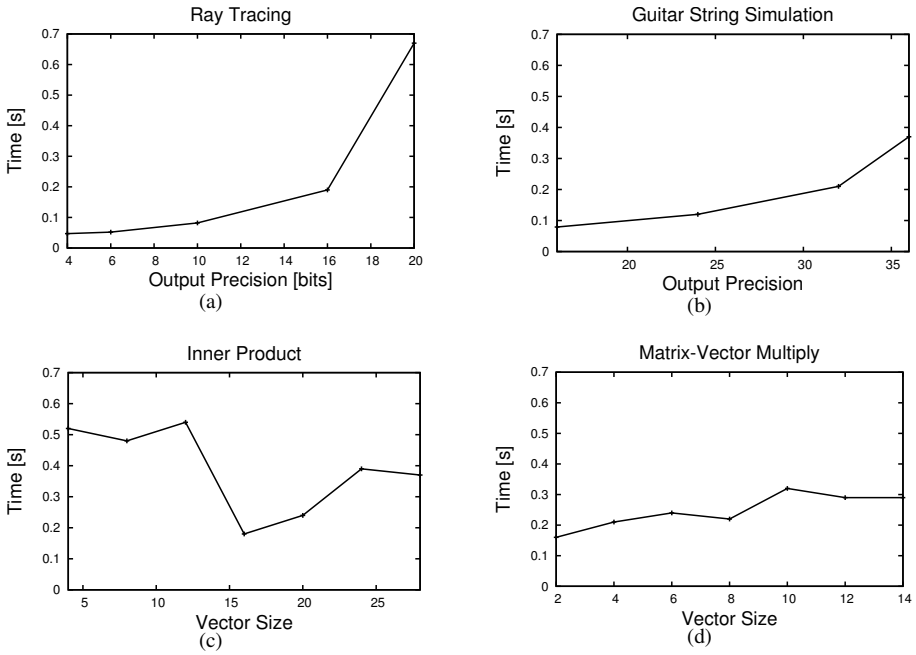


Figure 5. Average execution time above which bit-stream reconfiguration becomes more efficient than clock gating.

see that after fewer than 5 iterations, the maximum and minimum values of variables can be detected.

Similar techniques can be used in analysing word-length of variables in different execution phases, in case they have significantly different accuracy requirements, in addition to different branch probabilities [18]. Hence the proposed approach can provide additional optimisations for phase-optimised reconfigurable systems [19].

5.3 Custom processors

If a custom processor executes several different fixed or floating-point applications and if each application requires a different accuracy, then the unused bits can be set to zero, or clock-gated in order to save power. If a functional unit is not required by the processor, it may be more efficient to reconfigure the chip and remove it, rather than using clock gating and keeping the unit on-chip.

6 Results

To perform the experiments we use the Xilinx XUP board with a Virtex II Pro XC2VP30–7 FPGA. All designs are synthesized using Handel–C 4.1 and Xilinx ISE 9.2. We measure the power consumption by attaching an ammeter to the 1.5V VCCINT jumpers which supply power to the FPGA.

6.1 Arithmetic and simulation designs

When applying our model, we use 14ms for the average reconfiguration time [7] and 1500mW for the average reconfiguration power [2]. The energy required to reconfigure the chip is therefore 21mJ.

Figure 5 shows how the average execution time, given by equation (1), varies with output precision and vector size. In this case when the average execution time exceeds the value indicated on the graphs, bit-stream reconfiguration will be more energy-efficient than clock gating.

Figures 6 and 7 show how area, speed and power vary with vector size for designs with time-multiplexed constant coefficients (see Section 3.2) and designs with bit-stream reconfiguration optimized by constant propagation.

6.2 Ray tracing designs

The bottleneck in most ray-tracers is the ray-sphere intersection. For every ray, it must be determined whether it will intersect with an object. This kernel is executed approximately 70 million times for each image, of which approximately 2 million calls are intersections. We traverse the rays in a breadth-first fashion because this makes the ray intersections more predictable.

If the hardware design were implemented specifically to cater for simpler scenes, the precision could be greatly

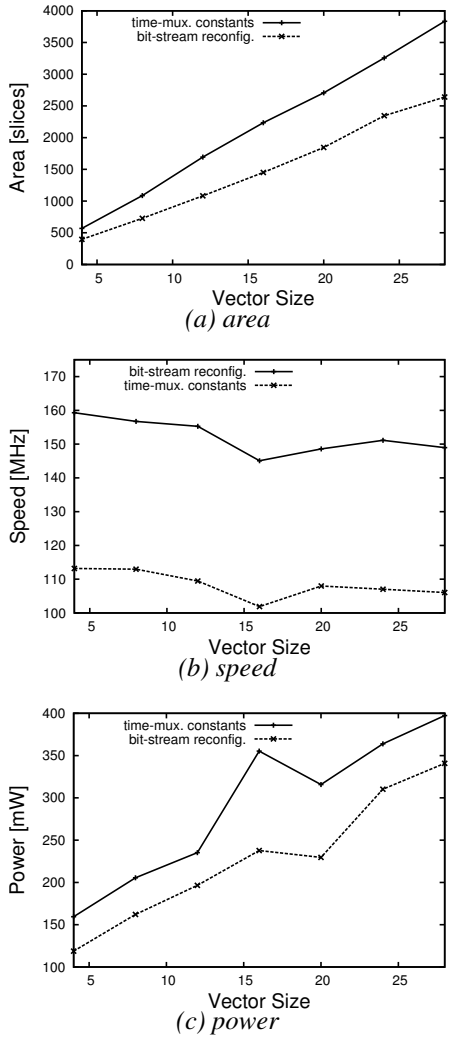


Figure 6. Variation of area, speed and power with vector size for a constant inner product design.

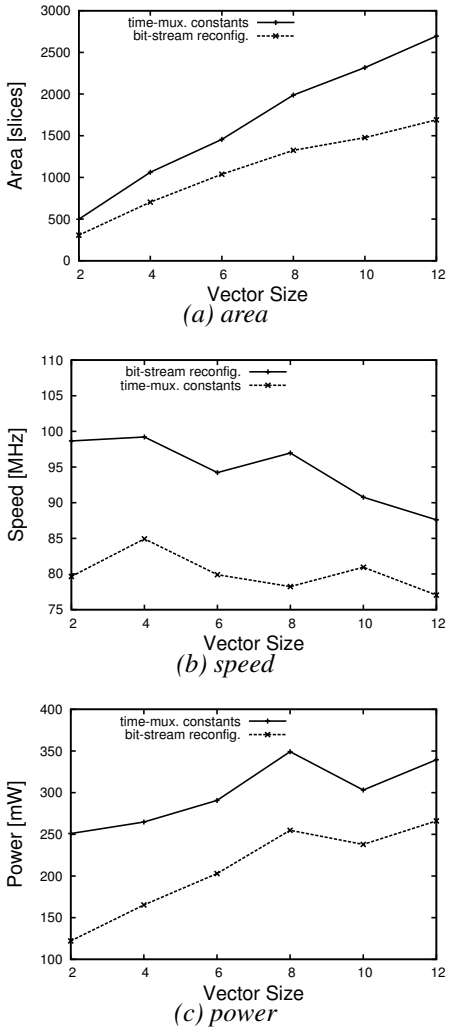


Figure 7. Variation of area, speed and power with vector size for a constant vector multiplier design.

reduced to conserve power. Since the ray tracer should be able to deal with any scene, we reduce the word-lengths at run time to reduce power consumption. Reducing the word-lengths can be achieved by gating the clock or the inputs as explained in Section 3.3.

Figure 8(a) shows the area and speed of the ray tracer at different output precisions. As expected, area increases as output precision increases, while speed decreases since larger designs are slower. Figure 8(b) shows the maximum dynamic power saving of the ray tracer at different output precisions for clock gating and bit-stream reconfiguration. If we adopt bit-stream reconfiguration rather than clock gating with lower precisions, the ray-tracer has shorter execution time and lower power consumption during execution. However, there are performance and energy overheads of reconfiguring the FPGA for designs with bit-stream reconfiguration, as explained in Section 4.

7 Summary

This paper presents an approach for developing reconfigurable designs that can benefit from clock gating, which includes a model that enables comparison of the proposed techniques with bit-stream reconfiguration. Current and future work includes automating the development method in Section 3, extending the proposed approach to cover reconfigurable designs that make use of both bit-stream reconfiguration and clock-gated reconfiguration, and quantifying the trade-offs of these methods for a wide variety of devices.

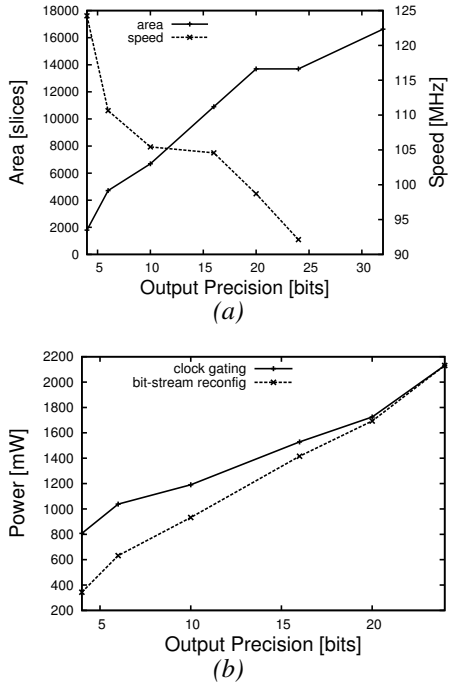


Figure 8. Area, speed and power consumption of the ray tracer at various output precisions.

Acknowledgments

The support of the UK Engineering and Physical Sciences Research Council, European FP6 hArtes (Holistic Approach to Reconfigurable Real Time Embedded Systems) project, Agility, Celoxica and Xilinx is gratefully acknowledged. We thank Tim Todman for providing code of the ray tracer.

References

[1] J. Becker, M. Hübner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka. Dynamic and partial FPGA exploitation. *Proceedings of the IEEE*, 95(2):438–452, July 2007.

[2] J. Becker, M. Hübner, and M. Ullmann. Power estimation and power measurement of Xilinx Virtex FPGAs: Trade-offs and limitations. In *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design*, pages 283–288, September 2003.

[3] T. Becker, W. Luk, and P. Y. K. Cheung. Enhancing relocatability of partial bitstreams for run-time reconfiguration. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 35–44, April 2007.

[4] K. Bondalapati and V. K. Prasanna. Dynamic precision management for loop computations on reconfigurable architectures. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 249–258, April 1999.

[5] O. Cadenas and G. Megson. Power performance with gated clocks of a pipelined Cordic core. In *Proceedings of the International Conference on ASIC*, volume 2, pages 1226–1230, October 2003.

[6] T. Courtney, R. Turner, and R. Woods. Mapping multi-mode circuits to LUT-based FPGA using embedded MUXes. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.

[7] B. Griese, E. Vonnahme, M. Porrmann, and U. Rückert. Hardware support for dynamic reconfiguration in reconfigurable SoC architectures. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 842–846, August 2004.

[8] M. Klein. Power considerations in 90nm FPGA designs. *Xcell Journal*, pages 56–59, Fourth Quarter 2005.

[9] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, February 2007.

[10] D. Lee, A. Abdul Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides. Accuracy-guaranteed bit-width optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10), October 2006.

[11] W. Luk, N. Shirazi, and P. Y. K. Cheung. Modelling and optimising run-time reconfigurable systems. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 167–176, April 1996.

[12] W. Luk, N. Shirazi, and P. Y. K. Cheung. Compilation tools for run-time reconfigurable designs. In *Proceedings IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 56–65, April 1997.

[13] W. G. Osborne, J. G. F. Coutinho, W. Luk, and O. Mencer. Power and branch aware word-length optimization. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2008.

[14] M. Parlak and I. Hamzaoglu. A low power implementation of H.264 adaptive deblocking filter algorithm. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems*, pages 127–133, August 2007.

[15] K. Paulsson, M. Hübner, and J. Becker. On-line optimization of FPGA power-dissipation by exploiting run-time adaption of communication primitives. In *Proceedings of the Symposium on Integrated circuits and systems design*, pages 173–178, 2006.

[16] N. Shirazi, W. Luk, and P. Y. K. Cheung. Automating prouction of run-time reconfigurable designs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 147–156, April 1998.

[17] J. Stephenson. *Design guidelines for optimal results in FPGAs*. Altera, 2005. http://www.altera.com/literature/cp/fpgas-optimal-results-396.pdf.

[18] H. Styles and W. Luk. Exploiting program branch probabilities in hardware compilation. *IEEE Transactions on Compututers*, 53(11):1408–1419, November 2004.

[19] H. Styles and W. Luk. Compilation and management of phase-optimized reconfigurable systems. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 311–316, August 2005.

[20] R. H. Turner and R. F. Woods. Design flow for efficient FPGA reconfiguration. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 972–975, September 2003.

[21] Y. Zhang, J. Roivainen, and A. Mämmelä. Clock-gating in FPGAs: A novel and comparative evaluation. In *Proceedings of the EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools*, pages 584–590, 2006.