The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays

Steven J.E. Wilton[1], Su-Shin Ang[2], and Wayne Luk[2]

[1]Dept. of Electrical and Computer Eng.
University of British Columbia
Vancouver, B.C., Canada
[2]Department of Computing
Imperial College
London, England

Abstract. This paper investigates experimentally the quantitative impact of pipelining on energy per operation for two representative FPGA devices: a 0.13µm CMOS high density/high speed FPGA (Altera Stratix EP1S40), and a 0.18µm CMOS low-cost FPGA (Xilinx XC2S200). The results are obtained by both measurements and execution of vendor-supplied tools for power estimation. It is found that pipelining can reduce the amount of energy per operation by between 40% and 90%. Further reduction in energy consumption can be achieved by power-aware clustering, although the effect becomes less pronounced for circuits with a large number of pipeline stages.

1 Introduction

Energy consumption has become a critical concern for Field-Programmable Gate Arrays (FPGAs). The programmability of FPGAs is afforded through the use of long routing tracks and programmable switches laden with parasitic capacitance. During high-speed operation, the switching of these tracks causes significant power dissipation. Hence an FPGA can consume up to two orders of magnitude more power than an Application-specific Integrated Circuit (ASIC) in the same technology [22].

FPGA power consumption can be reduced by optimizing the architecture of the programmable fabric or the Computer-Aided Design (CAD) algorithms used to map circuits onto the FPGA. Previous work has proposed CAD algorithms that optimize circuit implementations in an attempt to minimize energy, and achieve energy reductions of approximately 23% [10]. This improvement is unlikely to be sufficient to enable FPGA-based hand-held devices, or to significantly reduce the cost of expensive packaging. Significant gains are only possible at the algorithm or system design level. During algorithm or system design, the designer has considerable freedom regarding resource allocation and scheduling; correct decisions here are likely to result in a much larger impact on energy and power consumption than optimizations performed by logic optimization or physical design CAD tools.

One of the simplest but effective ways of reducing the energy per operation of a circuit is pipelining. A highly pipelined circuit suffers fewer glitches than an unpipelined circuit, since it typically has fewer logic levels between registers. Fewer

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 719–728, 2004.

glitches means that less dynamic power is dissipated during each cycle, which reduces the energy per operation.

The ability of pipelining to reduce glitches in ASICs is well-known [15]. In an FPGA, pipelining may be even more effective. Unlike an ASIC, in which signals can be routed using any available silicon, FPGAs implement interconnects using fixed metal tracks and programmable switches. The relative scarcity of programmable switches often forces signals to take longer routes than would be seen in an ASIC or custom integrated circuit. As a result, the potential for unequal delays among signals, and hence the creation of glitches, is likely more than that in an ASIC. Thus, we would expect pipelining to be an effective energy reduction technique for FPGAs.

Another reason that pipelining should work well in FPGAs is that commercial FPGAs contain one or more flip-flops in every logic block. These flip-flops often go unused. Thus, the additional flip-flops required for pipelining are usually "free" in an FPGA. On the other hand, FPGA clock trees are large and consume significant power; pipelining places further demands on the clock tree.

In this paper, we investigate the effectiveness of pipelining on the energy of FPGA circuit implementations, and compare it to the energy improvements that can be obtained by lower-level, power-aware synthesis algorithms. Specifically,

1. we present quantitative measurements of the impact of pipelining on the energy per operation consumed by datapath circuits in both a 0.13µm CMOS high density/speed FPGA and a 0.18µm CMOS low-cost FPGA;
2. we approximate the best possible gains that can be obtained by pipelining/retiming by considering circuits with registers after every logic element;
3. we investigate the interaction between pipelining (a system-level design optimization) and clustering (a lower-level design optimization), and determine how the degree of pipelining affects the effectiveness of the lower-level CAD algorithms in reducing energy.

The results of this study are important for four reasons. First, they provide guidance to a system designer designing FPGA circuits for a low-power application. As we will show, pipelining can reduce the energy per operation by 40% to 90%; this is the kind of reduction needed if FPGAs are to be used in handheld applications. Second, the results provide guidance to FPGA CAD designers by quantifying the effectiveness of high-level and low-level energy optimizations. As we will show in the next section, there has been significant research into power-aware retiming and other related algorithms; our results give an indication of how useful these optimizations can be at reducing energy. Third, pipelining also reduces the time for design tools to estimate power consumption; current vendor tools can take an excessive amount of time for power estimation when dealing with designs with a low degree of pipelining. Finally, there has been work on placing pipeline registers within the interconnect fabric of an FPGA [17]. Our results suggest that, in addition to increasing the clock rate, these registers will also be effective at reducing energy.

2 Preliminaries

Power consumption for circuits in CMOS technology has a static component and a dynamic component. The static power component is mainly due to leakage current. The dynamic power component is mainly due to switching activities for charging and discharging load capacitance. Although static power is becoming increasingly significant, dynamic power still dominates, even in a 0.13μm technology.

It is well-known that undesirable switching activities are usually caused by glitches: spurious pulses at the output of a combinational component due to input signals arriving at different times because of unequal input propagation delays. Techniques have been proposed to reduce such glitches, for instance by restructuring multiplexer networks and inserting selective delays [15], by logic decomposition based on glitch counting and location [4], and by selective gate freezing [1].

Several researchers have applied retiming for power and energy optimization. For example, different pipelined designs can be obtained by adding flip-flops to circuit inputs; such flip-flops can then be relocated by the retiming algorithm to reduce combinational path length and the associated switching activities [13]. Retiming can be combined with supply voltage scaling [3] and with register disabling [6].

Other techniques for reducing switching activities, including loop folding [8] and finite state machine decomposition [14], have also been studied. In addition, the effect of high-level compiler optimizations on system power has been investigated [7].

The power and energy optimizations described above are not developed specifically for FPGAs. FPGA-specific optimization schemes have also been reported. These include boolean optimization of multiple lookup tables [9], perturbation-based word-length optimization [5], implementation of critical loops in configurable logic [19], empirical energy modeling based on surface-curve fitting [16], and combination of power-aware CAD algorithms such as technology mapping and clustering [10].

Many of these optimization techniques rely on the reduction of switching activity by the insertion of registers. Yet the effectiveness of doing this in a modern FPGA has not been quantified. Our work, therefore, is orthogonal to these previous studies, since most of the previous studies rely on a significant improvement in power as a result of glitch reduction. In our work, we quantify what sort of reduction is possible.

The effect of pipeline granularity on reducing power consumption has been examined for Xilinx XC3000 [2], XC4000 and Virtex devices [20]; it is shown that glitch power reduction compensates the synchronization overheads due to pipelining in these small circuits. Our work is different from these previous studies in several ways. First, we consider FPGAs fabricated in modern including a 0.13μm CMOS technology; the power characteristics of integrated circuits implemented using these technologies are very different from those of integrated circuits implemented using older technologies. Second, we consider the pipeline-aware reductions in the context of an entire CAD flow, and examine the interactions between pipelining (a system-level optimization) and clustering, a lower-level power-optimization stage that has been shown to be effective at reducing power.

3 Impact of Pipelining

In this section, we quantify the impact of pipelining on FPGA implementations, first for a high speed and high density 0.13µm CMOS FPGA, and then for a low-cost 0.18µm CMOS FPGA.

3.1 Experimental Methodology

We employ an experimental methodology to investigate pipelining in FPGAs. We use four benchmark circuits; for each circuit, we create several versions, each with a different degree of pipelining. For some circuits, we add pipeline stages by modifying the original hardware description code by hand; for other circuits, we use an automatic synthesis tool [11] that generates circuits with differing degrees of pipelining. In all cases, the function of all versions of each circuit is the same, except for the additional latency imposed by pipeline stages. The first three columns of Table 1 list respectively our benchmark circuits, the number of registers, and the logic depth in each version of the circuit.

For each design, we also create a version with a pipeline stage after every logic element. Unlike all other versions of the circuit, this version is likely to have a different behaviour from the original circuit, since paths containing different numbers of logic elements will have different numbers of registers. The results from this version of each circuit will give an estimate of the best possible optimization achievable using pipelining. To generate these circuits, we use facilities provided through the Quartus University Interface Program (QUIP) which allow us to insert registers after logic synthesis but before place and route. The rows labeled "Max" in Table 1 show the statistics for these circuits. For all but the multiplier circuit, the depth of these "maximally pipelined" circuits is larger than one LE (Logic Element); this is because we do not insert pipeline registers along the carry and cascade chains.

The circuits are implemented on an Altera Nios Development Kit (Stratix Professional Edition) which contains a 0.13µm CMOS Stratix EP1S40F780C5 device. The input pins are not driven externally, since this could consume significant power, possibly dwarfing the on-chip power we want to measure. Instead, we generate our test vectors on-chip using a linear-feedback shift register. For the same reason, we do not drive a pin with each circuit output. Instead, we combine all outputs using a multi-input exclusive-or gate and feed the result to an output pin. We use an exclusive-or gate rather than leaving the outputs floating, to ensure that Quartus does not "optimize away" parts of the circuit not used to drive output pins. The outputs are registered before the exclusive-or gate, preventing glitches appearing at the inputs to the exclusive-or gate. This ensures that the power consumed by the exclusive-or gate is constant across all versions of a circuit. Finally, we register the output of the exclusive-or gate to ensure that glitches on the output pin do not overwhelm the glitch power internal to the FPGA. The clock of each circuit is driven by a scaled version of the 50Mhz on-board oscillator.

3.2 Effect of Pipelining on a 0.13μm FPGA

The final two columns of Table 1 show our measured power results. These are obtained by measuring the current entering the board from the power supply, and multiplying by the power supply voltage. The first of these columns shows the overall board (system) power. This system power includes the power of the board's transformer. In this paper, we are interested in the dynamic power of the FPGA itself. Therefore, we have subtracted the quiescent power when the board is idle, and recorded the results in the final column of Table 1 – we have found that the power dissipated by an idle FPGA and board is independent of the configuration of the FPGA on that board. This difference between the quiescent power and the total power represents the dynamic power of both the FPGA and the board.

In our experimental set-up, we have no way of isolating the dynamic power of the FPGA itself. To estimate this quantity, we have simulated our circuits using the Quartus simulator and power estimator. These simulation results are shown in the sixth column of Table 1. We also record the component of the FPGA power that is due to dynamic switching inside the logic block array; the fifth column of Table 1 shows these measurements.

In gathering the results in Table 1, we use the same clock speed for all versions of a given circuit. By holding the clock rate constant while varying the amount of pipelining, we obtain measurements proportional to the total energy per operation for each circuit. Normally, pipelining is used to increase the clock frequency, thereby increasing the number of operations per second. However, pipelining can also be used to reduce power. As we will show, by pipelining a circuit without changing the clock frequency, power reductions can be achieved without a reduction in operations per second, provided that the additional latency can be tolerated at the system level.

The results from Table 1 are startling. For the 64-bit unsigned multiplier, the difference in dynamic system energy between our most pipelined variant and our least pipelined variant is 81%. For the other benchmark circuits, this difference ranges from 40% to 82%. When we focus on just the dynamic logic block energy, the difference is as high as 98%. In contrast, lower-level physical design optimizations can typically reduce energy by up to 23% [10]. The results in Table 1 show that, indeed, system-level optimizations such as pipelining can have a far more significant impact on the overall energy dissipation.

Table 1 also shows the results for the "maximally pipelined" variant of each benchmark circuit, in which a register is used at the output of every logic block. As the table shows, for all of the benchmark circuits, the energy dissipated by the maximally pipelined variant is smaller than the energy dissipated by all the other versions. However, in three of the circuits, the difference in energy between the maximally pipelined variant and the next-most pipelined variant is small. This implies that there is little opportunity of reducing glitch energy further by increasing the number of pipeline stages in these circuits.

Table 1. Pipelining results for 0.13µm FPGA.

Bench mark Circuit	Number of Pipeline Stages	Number of Registers	Max. Stage Depth (LE's)	FPGA Power Estimate by Quartus		Measured System Power	
				Logic Block Power (mW)	Total FPGA Power (mW)	Total Power (mW)	Dynamic Power (mW)
64-bit integer array mult.	2	361	105	2 289	2 748	9 657	7 659
	4	555	73	1 977	2 436	7 263	5 265
	8	943	57	173	631	5 427	3 429
	16	1 719	49	59.3	512	4 563	2 565
	32	3 271	45	38.5	498	4 041	2 043
	64	6 374	43	38.7	497	3 654	1 656
	Max	15 730	1	86.4	545	3 402	1 404
Triple-DES encrypt. circuit	6	3 823	33	1 225	1 684	4 204	2 206
	12	4 542	17	2 008	2 467	4 041	2 043
	24	5 983	9	365	824	3 015	1 017
	48	9 023	5	144	603	2 718	720
	Max	20 579	2	109	568	2 664	666
8-tap FP FIR filter	1	100	132	Could not estimate		12 645	10 647
	2	353	100	3 951	4 420	7 866	5 868
	4	545	43	1 987	2 468	5 580	3 582
	Max	6 738	31	219	776	3 834	1 836
Cordic circuit	2	2 147	160	Could not estimate		6 507	4 509
	4	3 811	80	512	971	5 139	3 141
	8	6 835	40	152	611	4 437	2 439
	16	13 171	20	105	565	4 716	2 718
	Max	25 191	20	107	567	4 140	2 142

3.3 Effect of Pipelining on a 0.18µm FPGA

The results in Table 1 are obtained using a 0.13µm FPGA. Intuitively, in an FPGA implemented using a less aggressive technology, we would expect a larger component of the overall energy to be dynamic energy, and hence glitch energy. Thus, pipelining may even be more effective. On the other hand, an FPGA implemented in a 0.18µm technology will likely contain fewer logic elements than one in a 0.13µm technology, hence the user circuits will likely be smaller, and thus the potential for glitches may be reduced.

Table 2 shows measured energy results obtained using a 0.18µm FPGA, a Xilinx Spartan XC2S200 device on a Celoxica RC-100 board. Since the chip is smaller than that employed in the previous section, we have scaled down our benchmark circuits by reducing the bit-width in the multiplier, the number of parallel Cordic modules in the Cordic circuit, and the number of taps in the FIR filter. We do not attempt to reduce the size of the triple-DES circuit. As the results show, the impact of pipelining on power – and hence energy per operation – is similar to that for the 0.13µm chip, although less pronounced.

Table 2. Pipelining Results for 0.18µm FPGA.

Benchmark Circuit	Number of Pipeline Stages	Measured System Power	
		Total Power (mW)	Dynamic Power (mW)
16-bit unsigned integer array multiplier	1	5 124	4 116
	2	3 924	2 916
	4	3 312	1 304
	8	3 192	2 184
	16	3 168	2 160
4-tap Floating Point FIR filter	1	10 476	9 468
	2	9 048	8 040
	4	7 800	6 792
Cordic circuit to compute sine and cosine of angle	2	5 777	4 408
	4	4 094	2 727
	8	3 364	1 995
	16	2 888	1 519

4 Interaction Between Pipelining and Clustering

Energy optimization can be performed during high-level system design by techniques such as pipelining, in addition to optimization during low-level synthesis/physical design. Related work has shown that reductions of up to 23% are achievable during synthesis and physical design [10]. In the previous section, we have shown that larger reductions are achievable during system design by pipelining the circuit.

A true power-aware CAD flow would likely attempt to optimize power at both the system design level and the synthesis/physical design level. However, it is conceivable that fewer power reduction opportunities will exist for the lower-level tools, if the higher-level tools are power-aware. In our case, pipelining will tend to reduce the number of nodes with very high switching activities due to glitches, so that there are fewer of these nodes for the physical design tools to optimize. At the extreme, in a very heavily-pipelined circuit, there are no glitches, meaning all nets will have roughly the same activity. This means that the physical design tools will not be able to effectively optimize for power consumption.

In this section, we investigate whether the effectiveness of lower-level tools is affected by pipelining at the system level. We focus on clustering, since it has been shown that clustering is more effective at reducing power than other low-level CAD stages [10]. Commercial FPGAs contain logic elements arranged in clusters; these are known as Logic Array Blocks in Altera parts and Configurable Logic Blocks in Xilinx parts. Each of these clusters contains between two and ten logic elements implemented as lookup-tables. Clustering attempts to pack tightly-connected logic elements together into clusters.

The power-aware cluster algorithm described in [10] attempts to minimize energy by encapsulating high-activity nets within a cluster, so that they can be implemented using low-capacitance intra-cluster connections. It avoids separating the pins of a high-activity net among several clusters such that the net must be implemented on

high-capacitance inter-cluster connections. Intuitively, this will be less effective if most nets have similar activities; in this section, we investigate whether this intuition holds.

4.1 Experimental Methodology

We illustrate our methodology by considering the Cordic circuit and the 64-bit integer multiplier, since these two circuits have the largest number of variants. Each circuit is first optimized and mapped to lookup-tables by Quartus. Then, using facilities provided in the Quartus University Interface Program (QUIP), we feed each technology-mapped netlist into both the power-aware cluster algorithm described in [10] and the non-power-aware cluster algorithm described in [12]. Next, each of these clustered circuits is read back into Quartus, placed and routed, and implemented on the FPGA. The power consumption, which is proportional to energy per operation, is then measured as in Section 3.

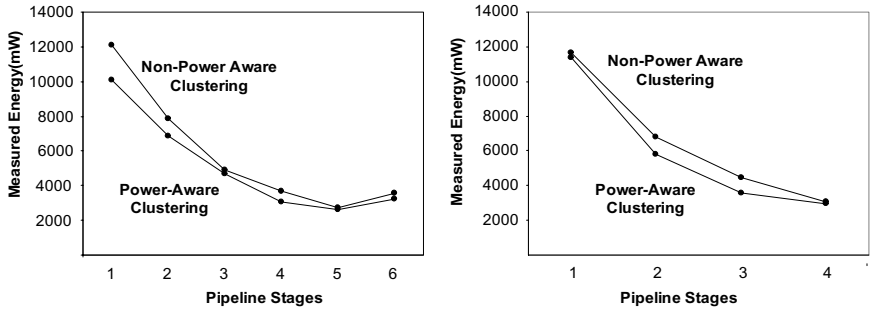
Note that neither of the clustering algorithms in [10] and [12] uses carry chains or cascade chains. As a result, it is not meaningful to compare these results with the results in Table 1 which are obtained using the Quartus clusterer, since those results do employ carry and cascade chains where appropriate. Nonetheless, the trends observed in this section will likely hold in a carry/cascade chain-capable clusterer.

4.2 Results

Figure 1 shows the results for the array multiplier and the Cordic circuits. The horizontal axis on each graph is the number of pipeline stages, and the vertical axis is the measured system (board) dynamic power, which is proportional to the energy per operation of the circuit. The top line in each graph represents the energy obtained when using the non-power-aware cluster algorithm, while the lower line represents the energy obtained when using the power-aware cluster algorithm. As the graphs show, the improvements obtained by pipelining are much more significant than those obtained by making the cluster algorithm power-aware.

The graph also illustrates the interaction between the two optimization schemes: the reduction achieved by the cluster algorithm varies as the degree of pipelining changes. In general, for the array multiplier, the reduction achieved by the cluster algorithm decreases as the degree of pipelining increases. For both circuits, the power-aware cluster algorithm is ineffective at reducing power for the most heavily-pipelined variants, since it has fewer high activity nets to work with.

These results are significant. They indicate that for most circuits, it does make sense to optimize for power both at the system level as well as during low-level synthesis and physical design. The results also show, however, that it is not reasonable for a designer to rely on pipeline-aware synthesis and physical design. Incorrect system-level design decisions, such as a bad choice for pipelining depth, can not be “made up for” by low-level CAD tools.



a) 64-bit integer multiplier b) Cordic circuit

Fig. 1. Power-aware and non-power aware clustering results.

5 Conclusions

In this paper, we have shown that pipelining has a significant impact on the energy dissipation in an FPGA. Pipelining reduces the number of spurious glitches which, in turn, reduces dynamic power. The primary contribution of this paper is the quantification of just how effective this technique can be in modern FPGAs. Among our four benchmark circuits, we have found that pipelining can reduce the amount of energy per operation required by an algorithm by between 40% and 90%.

These results are important for a number of reasons. For system designers, it illustrates the need for careful planning of a datapath and pipelining during system design. No matter how good the subsequent power-aware CAD tools are, it is unlikely that they will be able to "make up for" a bad decision during system design. Although designers have been aware of this before, our results suggest that correct pipelining is especially critical in modern FPGAs. For the CAD research community, these results suggest that more attention needs to be paid to high-level system-level optimizations, such as pipelining. Traditional power-aware optimization studies focus on lower-level technology mapping, clustering, or physical design. The gains achievable at these low level are important, but they will not be enough to make handheld FPGA devices a reality. On the other hand, the significant energy improvements shown in this paper can make FPGAs appropriate for a much larger class of low-power applications than ever before.

References

1. L. Benini et al, Glitch power minimization by selective gate freezing, IEEE Trans. VLSI Systems, 8(3): 287-298, 2000.
2. E. I. Boemo et al, Some notes on power management on FPGA based systems, Field Programmable Logic and Applciations, LNCS 975, Springer, 1995, pp. 149-157.
3. N. Chabini et al, Unification of basic retiming and supply voltage scaling to minimize dynamic power consumption for synchronous digital designs, Proc. ACM Great Lakes Symposium on VLSI, 2003.

4. K. S. Chung, T. Kim and C. L. Liu, A complete model for glitch analysis in logic circuits. Journal of Circuits, Systems, and Computers, 11(2): 137-154, 2002.
5. G. A. Constantinides, Perturbation analysis for word-length optimization, Proc. Int. Symp. field-Programmable Custom Computing Machines, 2003, pp. 81-90.
6. Y. L. Hsu and S. J. Wang, Retiming-based logic synthesis for low power, Proc. Int. Symp. Low Power Electronics and Design, ACM Press, 2002, pp. 275-278.
7. M. Kandemir et al, Influence of compiler optimizations on system power, IEEE Trans. VLSI, 9(6):801-804, 2001.
8. D. Kim and K. Choi, Power conscious high level synthesis using loop folding, Proc. 34th Design Automation Conference, 1997.
9. B. Kumthekar et al, Power optimization of FPGA-based designs without rewiring, IEE Proc., 147(3): 167-174, 2002.
10. J. Lamoureux and S. Wilton, On the interaction between power-aware FPGA CAD algorithms, Proc. ICCAD, 2003.
11. W. Luk et al, Parameterized hardware libraries for configurable system-on-chip technology, Canadian Journal of Elect. and Computer Engineering, 26(3/4):125-129, 2001.
12. A. Marquardt, V. Betz and J. Rose, Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density, ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays, Feb. 1999, pp. 37 - 46.
13. J. C. Monteiro, S. Devadas and A. Ghosh, Retiming sequential circuits for low power, Proc. ICCAD, pp. 398-402, 1993.
14. J. C. Monteiro and A. L. Oliveira, Finite state machine decomposition for low power, Proc. 35th Design Automation Conference, 1998.
15. A. Raghunathan, S. Dey and N. K. Jia, Register transfer level power optimization with emphasis on glitch analysis and reduction, IEEE Trans. CAD, 18(8):114-1131, 1999.
16. L. M. Reyneri et al, A hardware/software co-design flow and IP library based on Simulink, Proc. 38th Design Automation Conference, 2001.
17. A. Singh et al, Interconnect pipelining in a throughput-intensive FPGA architecture, ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays, 2001, pp 153-160.
18. D. Singh and S. Brown, The case for registered routing switches in Field Programmable Gate Arrays, ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays, 2001, pp. 161-169.
19. G. Stitt et al, Using on-chip configurable logic to reduce embedded system software energy, Proc. Int. Symp. Field-Programmable Custom Computing Machines, IEEE Computer Society Press, 2002, pp. 143-151.
20. G. Sutter et al, Logic depth, power, and pipeline granularity: updated results on XC4K and Virtex FPGAs, Computacion Reconfigurable & FPGAs, Publicaciones Digitales S.A., 2003, pp. 201-207.
21. W. Tsu, et al, HSRA: High-speed, hierarchical synchronous reconfigurable array, ACM Seventh International Symposium on Field-Programmable Gate Arrays, Feb. 1999.
22. P. Zuchowski et al, A hybrid ASIC and FPGA architecture, Proc. ICCAD, 2002, pp. 187-194.