Quantifying the Gap Between FPGA and Custom CMOS to Aid Microarchitectural Design

Henry Wong, *Student Member, IEEE* Vaughn Betz, *Member, IEEE* and Jonathan Rose, *Fellow, IEEE*

***Abstract*— This paper compares the delay and area of a comprehensive set of processor building block circuits when implemented on custom CMOS and FPGA substrates, then uses these results to show how soft processor microarchitectures should be different from those of hard processors. We find that the ratios of the custom CMOS versus FPGA area for different building blocks varies considerably more than the speed ratios, thus, area ratios have more impact on microarchitecture choices. Complete processor cores on an FPGA use 17–27× more area ("area ratio") than the same design implemented in custom CMOS. Building blocks with dedicated hardware support on FPGAs such as SRAMs, adders, and multipliers are particularly area-efficient (2–7×), while multiplexers and content-addressable memories (CAM) are particularly area-inefficient (>100×). Applying these results, we find out-of-order soft processors should use physical register file organizations to minimize CAM size.**

I. Introduction

THE area, speed, and energy consumption of a digital circuit will differ when it is implemented on different substrates such as custom CMOS, standard cell application-specific integrated circuits (ASICs), and field-programmable gate arrays (FPGAs). Those differences will also change based on the nature of the digital circuit itself. Having different cost ratios for different circuit types implies that systems built using a range of different circuit types must be tuned for each substrate. In this paper, we compare the custom CMOS and FPGA substrates with a focus on implementing instruction-set processors—we examine both full processors and subcircuits commonly used by processors, and explore the microarchitecture tradeoff space of soft processors in light of these differences.[1]

We believe this is a timely exercise, as the plausible area budget for soft processors is now much greater than it was when the first successful commercial soft processors were architected and deployed [2], [3]. Those first processors typically used less than a few thousand logic elements and have mostly employed single-issue in-order microarchitectures due to a limited area budget. Since then the size of FPGAs available has grown by one to two orders of magnitude, providing more space for more complex microarchitectures, if the increased complexity can achieve payoffs in performance. The design decisions that will be required to build more complex processors can benefit from a quantitative understanding of the differences between custom CMOS and FPGA substrates.

Previous studies have measured the average delay and area of FPGA, standard cell, and custom CMOS substrates across a large set of benchmark circuits [4], [5]. While these earlier results are useful in determining an estimate of the size and speed of the full system that can be implemented on FPGAs, it is often necessary to compare the relative performance of specific types of building block circuits to have enough detail for guiding microarchitecture design decisions.

This paper makes two contributions.

1) We compare the delay and area of custom CMOS and FPGA implementations of a specific set of building block circuits typically used in processors.
2) With these measured delay and area ratios, and prior custom CMOS processor microarchitecture knowledge, we discuss how processor microarchitecture design tradeoffs should change on an FPGA substrate.

We begin with a survey of prior work in Section II and describe our methodology in Section III. We then present the building block comparisons in Section IV and their impact on microarchitecture in Section V and conclude this paper in Section VI.

II. Background

A. Technology Impact on Microarchitecture

One of the goals in processor microarchitecture design is to make use of circuit structures that are best suited to the underlying implementation technology. Thus, studies on how process technology trends impact microarchitecture are essential for designing effective microarchitectures that best fit the ever-changing process characteristics. Issues currently facing CMOS technology include poor wire delay scaling, high power consumption, and more recently, process variation. Microarchitectural techniques that respond to these challenges include clustered processor microarchitectures and chip multiprocessors [6], [7].

Circuits implemented on an FPGA substrate face a very different set of constraints from custom CMOS. Although power consumption is important, it is not currently the dominant design constraint for FPGA designs. FPGA designs run at lower clock speeds and the architectures of FPGAs are already designed to give reasonable power consumption across the vast majority of FPGA user designs. Interestingly, area is often the primary constraint due to high area overhead of the programmability endemic to FPGAs. This different perspective combined with the fact that different structures

Manuscript received May 27, 2013; revised August 19, 2013; accepted September 25, 2013. Date of publication November 13, 2013; date of current version September 23, 2014. This work was supported in part by NSERC and in part by Altera.

The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: henry@eecg.utoronto.ca; vaughn@eecg.utoronto.ca; jayar@eecg.utoronto.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2284281

[1] An earlier version of this paper appeared in [1], which contained fewer circuit-level comparisons and less microarchitectural discussion. We have also significantly elaborated on the data in our discussions of the results, and added a section related to the effect of routing congestion in FPGAs.

have varying area, delay, and power characteristics between different implementation technologies means that understanding and measuring these differences are required to make good microarchitecture choices to suit the FPGA substrate. Characteristics such as inefficient multiplexers and the need to map RAM structures into FPGA hard static RAM (SRAM) blocks are known and are generally adjusted for by modifying circuit-level, but not microarchitecture-level, design [8]–[11].

B. Measurement of FPGAs

Kuon and Rose [4] have measured the area, delay, and power overheads of FPGAs compared with a standard cell ASIC flow on 90-nm processes. They used a benchmark set of complete circuits to measure the overall impact of using FPGAs compared with ASICs and the effect of FPGA hard blocks. They found that circuits implemented on FPGAs consumed 35× more area than on standard cell ASIC for circuits that did not use hard memory or multiplier blocks, to a low of 18× for those that used both types. The minimum cycle time (their measure of speed) of the FPGA circuits ranged from 3.0 to 3.5× greater than that of the ASIC implementations, and were not significantly affected by hard blocks. Chinnery and Keutzer [5] made similar comparisons between standard cell and custom CMOS and reported a delay ratio of 3–8×. Combined, these reports suggest that the delay of circuits implemented on an FPGA would be 9–28× greater than on custom CMOS. However, data for full circuits are insufficiently detailed to guide microarchitecture-level decisions, which is the focus of this paper.

III. Methodology

We seek to measure the delay and area of FPGA building block circuits and compare them against their custom CMOS counterparts, resulting in area and delay ratios. We define these ratios to be the area or delay of an FPGA circuit divided by the area or delay of the custom CMOS circuit. A higher ratio means the FPGA implementation is worse. We compare several complete processor cores and a set of building block circuits against their custom CMOS implementations, then observe which types of building block circuits have particularly high or low overhead on an FPGA.

As we do not have the expertise to implement highly optimized custom CMOS circuits, most of our building block circuit comparisons use data from custom CMOS implementations found in the literature. We focus mainly on custom CMOS designs built in 65-nm processes, because it is the most recent process where design examples are readily available in the literature. The custom CMOS data are compared with an Altera Stratix III 65-nm FPGA. In most of the cases, the equivalent FPGA circuits were implemented on an FPGA using the standard FPGA CAD flows. Power consumption is not compared due to the scarcity of data in the literature and the difficulty in standardizing testing conditions such as test vectors, voltage, and temperature.

We normalize area measurements to a 65-nm process using an ideal scale factor of 0.5× area between process nodes. We normalize delay using published ring oscillator data, with the understanding that these reflect gate delay scaling more than interconnect scaling. Intel reports 29% fanout-of-one (FO1) delay improvement between 90 and 65 nm, and 23% FO2 delay improvement between 65 and 45 nm [12], [13]. The area and delay scaling factors used are summarized in Table I.

TABLE I
Normalization Factors Between Processes

	90 nm	65 nm	45 nm
Area	0.5	1.0	2.0
Delay	0.78	1.0	1.23

TABLE II
Stratix III FPGA Resource Area Usage

Resource	Relative Area (Equiv. LABs)	Tile Area (mm^2)
LAB	1	0.0221
ALUT (half-ALM)	0.05	0.0011
M9K memory	2.87	0.0635
M144K memory	26.7	0.5897
DSP block	11.9	0.2623
Total core area	18 621	412

Delay is measured as the longest register to register path (sequential) or input to output path (combinational) in a circuit. In papers that describe CMOS circuits embedded in a larger unit (e.g., a shifter inside an arithmetic logic unit (ALU)), we conservatively assume that the subcircuit has the same cycle time as the larger unit. In FPGA circuits, delay is measured using register to register paths, with the register delay subtracted out when comparing subcircuits that do not include a register (e.g., wire delay).

To measure FPGA resource usage, we use the logic utilization metric as reported by Quartus rather than raw lookup table (LUT) count, as it includes an estimate of how often a partially used fracturable logic element can be shared with other logic. We count partially used memory and multiplier blocks as entirely used since it is unlikely another part of the design can use a partially used memory or multiplier block. Table II shows the areas of the Stratix III FPGA resources. The FPGA tile areas include the area used by the FPGA routing network so we do not track routing resource use separately. The core of the largest Stratix III (EP3LS340) FPGA contains 13 500 clusters (logic array block (LAB)) of 10 logic elements (adaptive logic module (ALM)) each, 1040 9-kb (M9K) memories, 48 144-kb (M144K) memories, and 72 DSP blocks, for a total of 18 621 LAB equivalent areas and 412-mm^2 core area.

We implemented FPGA circuits using Altera Quartus II 10.0 SP1 CAD flow and employed the fastest speed grade of the largest Stratix III device. We set timing constraints to maximize clock speed, reflecting the use of these circuits as a part of a larger circuit in the FPGA core, such as a soft processor.

We note that the custom CMOS design effort is likely to be much higher than for FPGA designs because there is more potential gain for optimization and much of the design process is automated for FPGA designs.

TABLE III
COMPLETE PROCESSOR CORES. AREA AND DELAY NORMALIZED TO 65 nm

Processor	Custom CMOS		FPGA		Ratios		FPGA Resource Utilization				
	f_{max} (MHz)	Area (mm^2)	f_{max} (MHz)	Area (mm^2)	f_{max}	Area	Utilization (ALUT)	ALUT	Registers	M9K	DSP
SPARC T1 (90 nm) [14]	1800	6.0	79	100	23	17	86 597	54 745	54 950	66	1
SPARC T2 (65 nm) [15]	1600	11.7	88	294	18	25	250 235	163 524	116 085	275	0
Atom (45 nm) [10], [16]	>1300	12.8	50	350	26	27	—— 85% of Virtex-5 LX330 ——				
Nehalem (45 nm) [17], [18]	3000	51	-	1240	-	24	—— 300% of Virtex-5 LX330 ——				
Geometric Mean					22	23					

IV. CUSTOM CMOS VERSUS FPGA

A. Complete Processor Cores

We begin by comparing complete processor cores implemented on an FPGA versus custom CMOS to provide context for the subsequent building block measurements. Table III shows a comparison of the area and delay of four commercial processors that have both custom CMOS and FPGA implementations, including in-order, multithreaded, and out-of-order processors. The FPGA implementations are synthesized from RTL code for the custom CMOS processors, with some FPGA-specific circuit-level optimizations. However, the FPGA-specific optimization effort is smaller than for custom CMOS designs and could inflate the area and delay ratios slightly.

The OpenSPARC T1 and T2 cores are derived from the in-order multithreaded UltraSPARC T1 and T2, respectively [19]. The OpenSPARC T2 processor core includes a floating-point unit. We synthesized one processor core for the Stratix III FPGA, with some debug features unnecessary in FPGA designs removed, such as register scan chains and SRAM redundancy in the caches.

The Intel Atom is a dual-issue in-order 64-bit x86 processor with two-way multithreading. Our Atom processor comparisons use published FPGA synthesis results by Wang *et al.* [10], which includes only the processor core without L2 cache, and occupies 85% of the largest 65 nm Virtex-5 FPGA (XC5VLX330). They do not publish a detailed breakdown of the FPGA resource utilization, so the FPGA area is estimated assuming the core area of the largest Virtex 5 is the same as the largest Stratix III.

The Intel Nehalem is an out-of-order 64-bit x86 processor with two-way multithreading. The FPGA synthesis by Schelle *et al.* [17] includes the processor core and does not include the per-core L2 cache, with an area utilization of roughly 300% of the largest Virtex-5 FPGA. They partitioned the processor core across five FPGAs and time-multiplexed the communication between FPGAs, so the resulting clock speed (520 kHz) is not useful for estimating delay ratio.

Table III compares the four processors' speed and area. For custom CMOS processors, the highest commercially available speed is listed, scaled to a 65-nm process using linear delay scaling, as described in Section III. The area of a custom CMOS processor is measured from die photos, including only the area of the processor core that the FPGA version implements, again scaled using ideal area scaling to a 65-nm process. The sixth and seventh columns contain the speed and area ratio between custom CMOS and FPGA, with higher ratios meaning that the FPGA is worse.

For the two OpenSPARC processors, FPGA area is measured using the resource usage (ALUT logic utilization, DSP blocks, and RAM) of the design reported by Quartus multiplied by the area of each resource in Table II. The FPGA synthesis of the Intel processors use data from the literature, which only gave approximate area usage, so we list the logic utilization as a fraction of the FPGA chip.

Overall, custom processors have delay ratios of 18–26× and area ratios of 17–27×. We use these processor core area and delay ratios as a reference point for the building block circuit comparisons in the remainder of this paper. For each building block circuit, we compare the FPGA versus custom CMOS area and delay ratios for the building block circuit to the corresponding ratios for processor cores to judge whether a building block circuit's area and delay are better or worse than the overall ratios for a processor core.

Interestingly, there is no obvious area ratio trend with processor complexity—for example, we might expect that an out-of-order processor synthesized for an FPGA, such as the Nehalem, to have a particularly high area ratio, but it does not. We speculate that this is because expensive content-addressable memories (CAMs) are only a small portion of the hardware added by a high-performance microarchitecture. The added hardware includes a considerable amount of RAM and other logic, since modern processor designs already seek to minimize the use of CAMs due to their high power consumption. On the FPGA-synthesized Nehalem, the hardware structures commonly associated with out-of-order execution (reorder buffer (ROB), reservation stations, and register renaming) consume around 45% of the processor core's LUT usage [17].

B. SRAM Blocks (Low Port Count)

SRAM blocks are commonly used in processors for building caches and register files. SRAM performance can be characterized by latency and throughput. Custom CMOS SRAM designs can trade latency and throughput by pipelining, while FPGA designs are limited to the prefabricated SRAM blocks on the FPGA.

TABLE IV
CUSTOM CMOS AND FPGA SRAM BLOCKS

Design	Ports	Size (kbit)	f_{max} (MHz)	Area (mm^2)	Bit Density (kbit/mm^2)	Ratios f_{max}	Ratios Density
IBM 6T 65 nm [20]	2r or 1w	128	5600	0.276	464	9.5	2.1
Intel 6T 65 nm [21]	1rw	256	4200	0.3	853	7.1	3.9
Intel 6T 65 nm [22]	1rw	70 Mb	3430	110 [23]	820	–	–
IBM 8T 65 nm SOI [24]	1r1w	32	5300	–	–	9.0	–
Intel 65 nm Regfile [25]	1r1w	1	8800	0.017	59	15	3.7
Stratix III FPGA							
Registers	1rw	-	-	-	0.76		
MLAB	1rw	0.625	450	0.025	25		
M9K	1rw	9	590	0.064	142		
M144K	1rw	144	590	0.59	244		

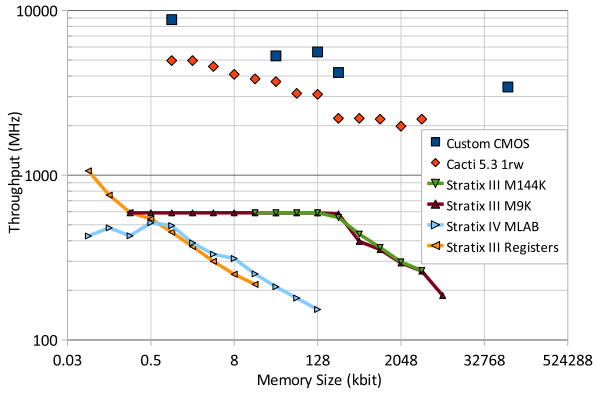


Fig. 1. SRAM throughput.

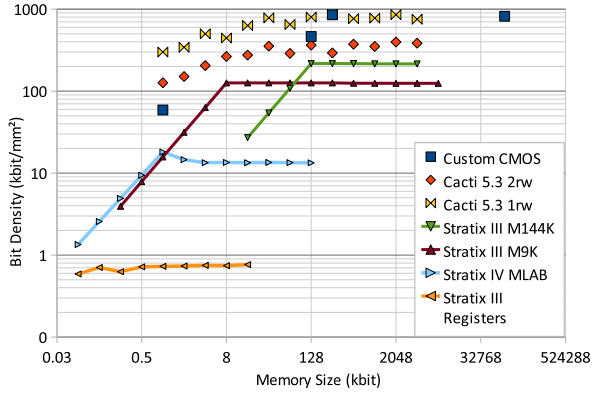


Fig. 2. SRAM density.

Logical SRAMs targeting the Stratix III FPGA can be implemented in four different ways: using one of the two physical sizes of hard block SRAM, using the LUT RAMs in memory LABs (MLABs allow the lookup tables in a LAB to be converted into a small RAM), or in registers and LUTs. The throughput and density of the four methods of implementing RAM storage are compared in Table IV to five high-performance custom SRAMs in 65-nm processes. In this section, we focus on RAMs with one read–write port (which we will refer to as 1rw), as it is a commonly used configuration in larger caches in processors, but some custom CMOS SRAMs have unusual port configurations, such as being able to do two reads or one write [20]. The size column lists the size of the SRAM block. For MLAB (LUT RAM, 640 bit), M9K (block RAM, 9 kbit), and M144K (block RAM, 144 kbit) FPGA memories, memory size shows the capacity of the memory block type. The f_{max} and area columns list the maximum clock speed and area of the SRAM block. Because of the large variety of SRAM block sizes, it is more useful to compare bit density. The last two columns of the table list f_{max} and bit density ratios between custom CMOS SRAM blocks and an FPGA implementation of the same block size on an FPGA. Higher density ratios show worse density on FPGA.

The density and throughput of custom CMOS and FPGA SRAMs listed in Table IV are plotted against memory size in Figs. 1 and 2. The plots include data from CACTI 5.3, a CMOS memory performance, and area model [26]. There is good agreement between the CACTI models and the design examples from the literature, although CACTI appears to be slightly more conservative.

The throughput ratio between FPGA memories and custom is 7–10×, lower than the overall delay ratio of 18–26×, showing that SRAMs are relatively fast on FPGAs. It is surprising that this ratio is not even lower because FPGA SRAM blocks have a little programmability. The 2-kb MLAB (64 × 32) memory has a particularly low delay because its 64-entry depth uses the 64 × 10 mode of the MLAB, allowing both its input and output registers to be packed into the same LAB as the memory itself (each LAB has 20 registers), yet it does not need external multiplexers to stitch together multiple MLABs.

The FPGA data above use 32-bit wide data ports (often the width of a register on 32-bit processors) that slightly underutilize the native FPGA 36-bit ports. The raw density of a fully used FPGA SRAM block is listed in Table IV. Below 9 kb, the bit density of FPGA RAMs falls off nearly linearly with reducing RAM size because M9Ks are underutilized. The MLABs use 20-bit wide ports, so a 32-bit wide memory block always uses at least two MLABs, using 80% of their capacity. The MLAB bit density (25 kb/mm^2) is low, although it is still much better than using registers and LUTs (0.76 kb/mm^2). For larger arrays with good utilization, FPGA SRAM arrays have a density ratio of only 2–5× versus single read–write

TABLE V
MULTIPORTED 8-kb SRAM. LVT DATA FROM [27]

Ports	CACTI 5.3		FPGA		Ratios	
	f_{max} (MHz)	Density ($\frac{kbit}{mm^2}$)	f_{max} (MHz)	Density ($\frac{kbit}{mm^2}$)	f_{max}	Density
2r1w	3750	177	497	63	7.6	2.8
4r2w	3430	45	228	0.25	15	179
6r3w	3270	27	214	0.20	15	140
8r4w	2950	17	178	0.15	17	109
10r5w	2680	11	168	0.11	16	104
12r6w	2450	8.0	140	0.091	18	87
14r7w	2250	6.1	130	0.080	17	75
16r8w	2070	4.8	126	0.064	16	74
Live Value Table (LVT)						
4r2w			375	3.9	9.2	12
8r4w			280	0.98	11	17

port (1rw)[2] CMOS (and CACTI) SRAMs, far below the full processor area ratio of 17–27×.

As FPGA SRAMs use dual-ported (2rw) arrays, we also plotted CACTIs 2rw model for comparison. For arrays of similar size, the bit density of CACTIs 2rw models are 1.9× and 1.5× the raw bit density of fully utilized M9K and M144K memory blocks, respectively. This suggests that half of the bit density gap between custom CMOS and FPGA SRAMs in our single-ported test is due to FPGA memories paying the overhead of dual ports.

For register file use where latency may be more important than memory density, custom processors have the option of trading throughput for area and power using faster and larger storage cells. The 65-nm Pentium 4 register file trades decreased bit density for 9-GHz single-cycle performance [25]. FPGA RAMs lack this flexibility, and the delay ratio is even greater (15×) for this specific use.

C. Multiported SRAM Blocks

FPGA hard SRAM blocks can typically implement up to two read–write ports (2rw). Implementing more read ports on an FPGA can be achieved reasonably efficiently by replicating the memory blocks, but increasing the number of write ports is more difficult. A multiple write port RAM can be implemented using registers for storage and LUTs for multiplexing and address decoding, but is inefficient. A more efficient method using hard RAM blocks for most of the storage replicates memory blocks for each write and read port and uses a live value table (LVT) to show for each word which of the replicated memories holds the most recent copy [27].

We present data for multiported RAMs implemented using registers, LVT-based multiported memories from [27], and CACTI 5.3 models of custom CMOS multiported RAMs. Like for single-ported SRAMs (Section IV-B), we report the random cycle time of a pipelined custom CMOS memory. We focus on a 256 × 32-bit (8 kb) memory block with twice as many read ports as write ports (2*N* read, *N* write) because it is a port configuration often used in register files in processors and the size fits well into an M9K memory block. Table V shows the throughput and density comparisons.

The custom CMOS versus FPGA bit density ratio is 2.8× for 2r1w, and increases to 12× and 179× for 4r2w LVT- and register-based memories, respectively. When only one write port is needed (2r1w), the increased area needed for duplicating the FPGA memory block to provide a second read port is less than the area increase for tripling the number of ports from 1rw to 2r1w of a custom CMOS RAM (445 kb/mm^2 1rw from Section IV-B to 177 kb/mm^2 2r1w). LVT-based memories improve in density on register-based memories, but both are worse than simple replication used for memories with one write port and multiple read ports.

The delay ratio is 7.6× for 2r1w, and increases to 9× and 15× for 4r2w LVT- and register-based memories, respectively, a smaller impact than the area ratio increase. The delay ratios when using registers to implement memories (15–18×) are higher than those for single-ported RAMs using hard RAM blocks, but still slightly lower than the overall processor core delay ratios.

[2]There are three basic types of memory ports: read (r), write (w), and read–write (rw). A read–write port can read or write, but not both, per cycle.

TABLE VI
CAM DESIGNS

	Size (bits)	Search Time (ns)	Bit Density ($\frac{kbit}{mm^2}$)	Ratios vs. Soft Logic Delay	Density
Ternary CAMs (65 nm)					
IBM 64×72 [31]	4 608	0.6	-	5.4	-
IBM 64×240 [31]	15 360	2.2	167	1.8	519
Binary CAMs (65 nm)					
POWER6 8×60 [32]	480	<0.2	-	14	-
Godson-3 64×64 [33]	4 096	0.55	76	5	99
Intel 64×128 [34]	8 192	0.25	167	14	209
FPGA Ternary CAMs					
Soft logic 64×72	4 608	3.2	0.40		
Soft logic 64×240	15 360	4.0	0.32		
FPGA Binary CAMs					
Soft logic 8×60	480	2.1	0.83		
Soft logic 64×64	4 096	2.9	0.77		
Soft logic 64×128	8 192	3.4	0.80		
MLAB-CAM 64×20	1 280	4.5	1.0		
M9K-CAM 64×16	1 024	2.0	2.0		

D. Content-Addressable Memories

A CAM is a logic circuit that allows associative searches of its stored contents. Custom CMOS CAMs are typically implemented as dense arrays of cells using nine-transistor (9T) to 11T cells compared with 6T used in SRAM and are typically 2–3× less dense than custom SRAMs. Ternary CAMs use two storage cells per bit to store three states (0, 1, and don't care). In processors, CAMs are used in tag arrays for high-associativity caches and translation lookaside buffers (TLBs). CAM-like structures are also used in out-of-order instruction

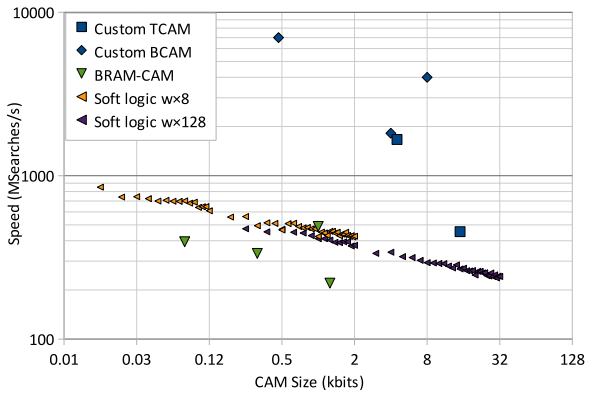


Fig. 3. CAM search speed.

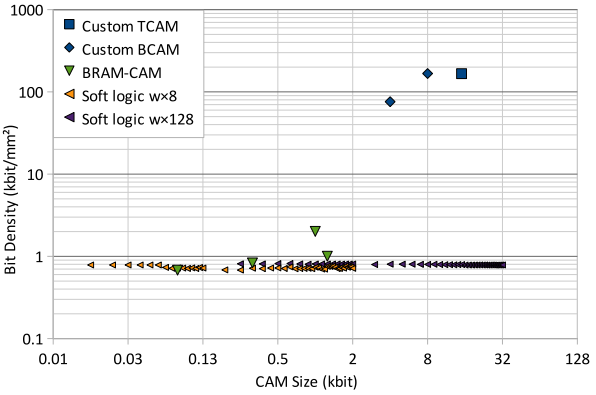


Fig. 4. CAM bit density.

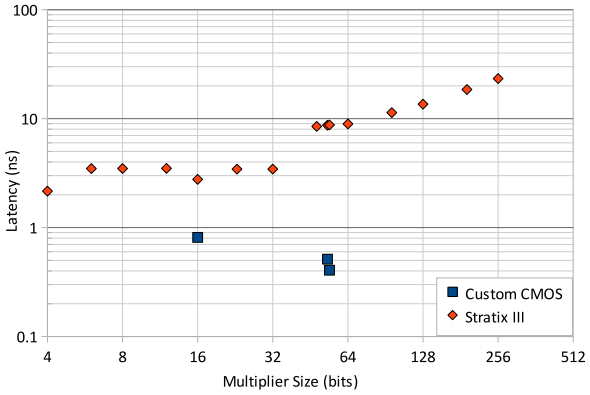


Fig. 5. Multiplier latency.

TABLE VII
MULTIPLIER AREA AND DELAY, NORMALIZED TO 65-nm PROCESS. UNPIPELINED LATENCY IS PIPELINED CYCLE TIME × STAGES

Design	Size	Stages	Latency (ns)	Area (mm^2)	Ratios Latency	Ratios Area
Intel 90 nm 1.3 V [35]	16×16	1	0.81	0.014	3.4	4.7
IBM 90 nm SOI 1.4 V [36]	54×54	4	0.41	0.062	22	7.0
IBM 90 nm SOI 1.3 V [37]	53×53	3	0.51	0.095	17	4.5
Stratix III	16×16	1	2.8	0.066		
Stratix III	54×54	1	8.8	0.43		

schedulers. CAMs in processors require both frequent read and write capability, but not large capacities. Pagiamtzis and Sheikholeslami [28] give a good overview of the CAM design space.

There are several methods of implementing CAM functionality on FPGAs that do not have hard CAM blocks [29]. CAMs implemented in soft logic use registers for storage and LUTs to read, write, and search the stored bits. Another proposal, which we will refer to as BRAM-CAM, stores one-hot encoded match-line values in block RAM to provide the functionality of a $w \times b$-bit CAM using a $2^b \times w$-bit block RAM [30]. The soft logic CAM is the only design that provides one-cycle writes. The BRAM-CAM offers improved bit density but requires two-cycle writes—one cycle each to erase then add an entry. We do not consider FPGA CAM implementations with even longer write times that are only useful in applications where modifying the contents of the CAM is a rare event, such as modifying a network routing table.

Table VI shows a variety of custom CMOS and FPGA CAM designs. Search time shows the time needed to perform an unpipelined CAM lookup operation. The FPGA versus custom CMOS ratios compare the delay (search time) and density between each custom CMOS design example and an FPGA soft logic implementation of a CAM of the same size. Figs. 3 and 4 plot these and also 8- and 128-bit wide soft logic CAMs of varying depth.

CAMs can achieve delay comparable with SRAMs but at a high cost in power. For example, Intel's 64 × 128 BCAM achieves 4 GHz using 13 fJ/bit/search, while IBMs 450-MHz 64 × 240 ternary CAM uses 1 fJ/bit/search.

As shown in Table VI, soft logic binary CAMs have poor bit density ratios versus custom CMOS CAMs—from 100 to 210 times worse. We included ternary CAM examples in the table for completeness, but since they are generally not used inside processors, we do not include them when summarizing CAM density ratios. Despite the poor density of soft logic CAMs, the delay ratio is only 14 times worse. BRAM-CAMs built from M9Ks can offer 2.4× better density than soft logic CAMs but needs two cycles per write. The halved write bandwidth of BRAM-CAMs makes them unsuitable for performance-critical uses, such as tag matching in instruction schedulers and L1 caches.

We observe that the bit density of soft logic CAMs is nearly the same as using registers to implement RAM (Table IV), suggesting that most of the area inefficiency comes from using registers for storage, not the added logic to perform associative searching.

E. Multipliers

Multiplication is an operation performed frequently in signal processing applications, but not used as often in processors. In a processor, only a few multipliers would be found in ALUs to perform multiplication instructions. Multiplier blocks can also be used to inefficiently implement shifters and multiplexers [38].

Fig. 5 shows the latency of multiplier circuits on custom CMOS and on FPGA using hard DSP blocks. Latency is

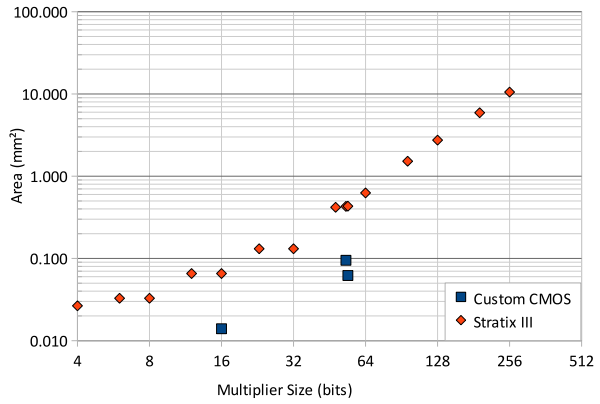


Fig. 6. Multiplier area.

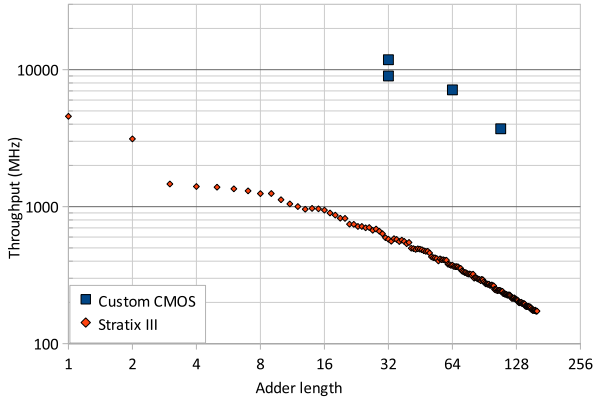


Fig. 7. Adder delay.

the product of the cycle time and the number of pipeline stages, and does not adjust for unbalanced pipeline stages or pipeline latch overheads. Table VII shows details of the design examples.

The two IBM multipliers have latency ratios comparable with full processor cores. Intel's 16-bit multiplier design has much lower latency ratios as it appears to target low power instead of delay. In designs where multiplier throughput is more important than latency, multipliers can be made more deeply pipelined (three and four stages in these examples) than the hard multipliers on FPGAs (two stages), and throughput ratios can be even higher than the latency ratios.

The area of the custom CMOS and FPGA multipliers is plotted in Fig. 6. FPGA multipliers are relatively area efficient. The area ratios for multipliers of 4.5–7.0× are much lower than for full processor cores (17–27×, Section IV-A).

F. Adders

Custom CMOS adder circuit designs can span the area-delay tradeoff space from slow ripple-carry adders to logarithmic-depth fast adders. On an FPGA, adders are usually implemented using hard carry chains that implement variations of the ripple-carry adder, although carry-select adders have been also been used. Although fast adders can be implemented on FPGAs with soft logic and routing, the lack of dedicated circuitry means fast adders are bigger and usually slower than the ripple-carry adder with hard carry chains [43].

Fig. 7 plots a comparison of adder delay, with details in Table VIII. The Pentium 4 delay is conservative as the delay

TABLE VIII
ADDERS. AREA AND DELAY NORMALIZED TO 65-nm PROCESS

Design	Size (bit)	f_{max} (MHz)	Area (mm^2)	Delay Ratio	Area Ratio
Agah [39]	32	12 000 1.3 V	-	20	-
Kao 90 nm [40]	64	7 100 1.3 V	0.016	19	4.5
Pentium 4 [41]	32	9 000 1.3 V	-	16	-
IBM [42]	108	3 700 1.0 V	0.017	15	6.9
Stratix III	32	593	0.035		
	64	374	0.071		
	108	242	0.119		

TABLE IX
ANALYTICAL MODEL OF TRANSMISSION GATE OR PASS TRANSISTOR TREE MULTIPLEXERS [44] NORMALIZED TO 65-nm PROCESS

Mux Inputs	FPGA Area (mm^2)	FPGA Delay (ps)	Custom CMOS Delay (ps)	Delay Ratio
2	0.0011	210	2.8	74
4	0.0011	260	4.9	53
8	0.0022	500	9.1	54
16	0.0055	680	18	37
32	0.0100	940	29	32
64	0.0232	1200	54	21

given is for the full integer ALU. FPGA adders achieve delay ratios of 15–20× and a low area ratio of around 4.5–7×. Despite the use of dedicated carry chains on the FPGA, the delay ratios are fairly high because we compare FPGA adders to high-performance custom CMOS adders. For high-performance applications, such as in processors, FPGAs offer a little flexibility in trading area for even more performance using a faster circuit-level design.

G. Multiplexers

Multiplexers are found in many circuits, yet we have found little literature that provides their area and delay in custom CMOS. Instead, we estimate delays of small multiplexers using a resistor–capacitor analytical model, the delays of the Pentium 4 shifter unit, and the delays of the Stratix III ALM. Our area ratio estimate comes from an indirect measurement using an ALM.

Table IX shows a delay comparison between an FPGA and an analytical model of transmission gate or pass gate tree multiplexers [44]. This unbuffered switch model is pessimistic for larger multiplexers, as active buffer elements can reduce delay. On an FPGA, small multiplexers can often be combined with other logic with minimal extra delay and area, so multiplexers measured in isolation are likely pessimistic. For small multiplexers, the delay ratio is high, roughly 40–75×. Larger multiplexers appear to have decreasing delay ratios, but we believe this is largely due to the unsuitability of the unbuffered designs to which we are comparing.

An estimate of the multiplexer delay ratio can also be made by comparing the delay of larger circuits that are composed

TABLE X
DELAY OF MULTIPLEXER-DOMINATED CIRCUITS

Circuit	FPGA Delay (ps)	Custom CMOS Delay (ps)	Ratio
65 nm Pentium 4 Shifter	2260	111	20
Stratix III ALM			
Long path	2500	350	7.1
Short path	800	68	11.7

mainly of multiplexers. The 65-nm Pentium 4 integer shifter datapath [41] is one such circuit, containing small multiplexers (sizes three, four, and eight). We implemented the same datapath excluding control logic on the Stratix III. A comparison of the critical path delay is shown in Table X. The delay ratio of 20× is smaller than suggested by the isolated multiplexer comparison, but may be optimistic if Intel omitted details from their shifter circuit diagram causing our FPGA equivalent shifter to be oversimplified.

Another delay ratio estimate can be made by examining the Stratix III ALM itself, as its delay consists mainly of multiplexers. We implemented a circuit equivalent to an ALM, as described in the Stratix III Handbook [45], comparing delays of the FPGA implementation to custom CMOS delays of the ALM given by the Quartus timing models. Internal LUTs are modeled as multiplexers that select between static configuration RAM bits. Each ALM input pin is modeled as a 21-to-1 multiplexer, as 21–30 are reasonable sizes according to [46].

We examined one long and one short path, from after the input multiplexers for pins datab and dataf0, respectively, terminating at the LUT register. Table X shows delay ratios of 7.1× and 11.7× for the long and short paths, respectively. These delay ratios are lower compared with previous examples due to the lower power and area budgets preventing custom FPGAs from being as aggressively delay-optimized as custom processors, and to extra circuit complexity not shown in the Stratix III Handbook.

We can also estimate a lower bound on the multiplexer area ratio by implementing only the multiplexers in our FPGA equivalent circuit of an ALM, knowing the original ALM contains more functionality than our equivalent circuit. Our equivalent ALM consumes 104 ALUTs, or roughly 52 ALMs, resulting in an estimated area ratio of 52×. However, the real ALM area ratio is substantially greater, as we implemented only the ALMs input and internal multiplexers and did not include global routing resources or configuration RAM. A rule of thumb is that half of an FPGA's core area is spent in the programmable global routing network, doubling the area ratio estimate to 104× while still neglecting the configuration RAM.

In summary, groups of multiplexers (measured from the Pentium 4 shifter and ALM) have delay ratios below 20×, with small isolated multiplexers being worse (40–75×). However, multiplexers are particularly area intensive with an area ratio greater than 100×. Thus, we find that the intuition that multiplexers are expensive on FPGAs is justified, especially from an area perspective.

TABLE XI
PIPELINE LATCH DELAY

Design	Register Delay (ps)	Delay Ratio
[47]	35 (90 ps in 180 nm)	12
[48]	32 (2.5 FO4)	14
[49]	23 (1.8 FO4)	19
Geometric Mean	29.5	15
Stratix III	436	-

H. Pipeline Latches

In synchronous circuits, the maximum clock speed of a circuit is typically limited by a register-to-register delay path from a pipeline latch,[3] through a pipeline stage's combinational logic, to the next set of pipeline latches. The delay of a pipeline latch (its setup and clock-to-output times) impacts the speed of a circuit and the clock speed improvement when increasing pipeline depth. Note that hold times do not directly impact the speed of a circuit, only correctness.

The effective cost in delay of inserting an extra pipeline register into LUT-based combinational pipeline logic is measured by observing the increase in delay as the number of LUTs between registers increases, then extrapolating the delay to zero LUTs. This method is different from, and more pessimistic than, simply summing the T_{co}, T_{su}, clock skew, and one extra LUT-to-register interconnect delay to reach a register, which is 260 ps. This pessimism occurs because inserting a register also impacts the delay of the combinational portion of the delay path. The measured latch delay in Stratix III is 436 ps.

Table XI shows estimates of the delay of a custom CMOS pipeline latch. The 180-nm Pentium 4 design assumed 90 ps of pipeline latch delays including clock skew [47], which we scaled according to the FO1 ring oscillator delays for Intel's processes (11 ps at 180 nm to 4.25 ps at 65 nm) [22]. Hartstein *et al.* [48] and Hrishikesh *et al.* [49] present estimates expressed in FO4 delays, which were scaled to an estimated FO4 delay of 12.8 ps for Intel's 65-nm process.

Thus, the delay ratio for a pipeline latch ranges from 10 to 15 times. Although we do not have area comparisons, registers are considered to occupy very little FPGA area because more LUTs are used than registers in most FPGA circuits, yet FPGA logic elements include at least one register for every LUT.

I. Interconnect Delays

Interconnect delay comprises a significant portion of the total delay in both FPGAs and modern CMOS processes. In this section, we explore the point-to-point delay of these technologies, and include the effect of congestion on these results.

1) Point-to-Point Routing: In this section, we measure the wire delay of a point-to-point (single fanout) connection. In modern CMOS processes, there are multiple layers of interconnect wires, for dense local and faster global connections.

[3]Latch refers to pipeline storage elements. This can be a latch, flip-flop, or other implementation.

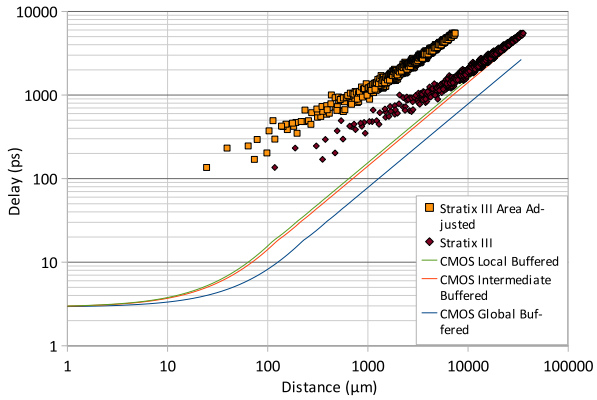


Fig. 8. Point-to-point routing delay.

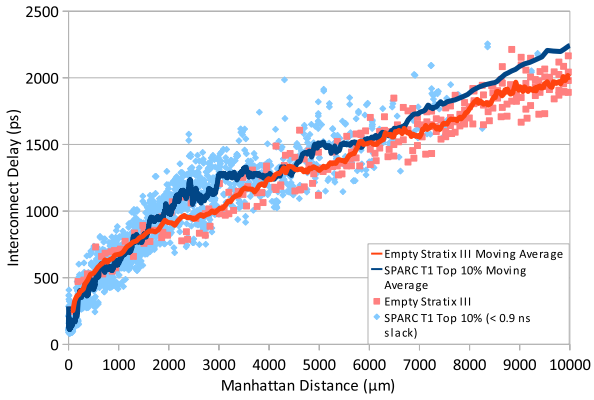
On an FPGA, an automated router chooses a combination of faster long wires or more abundant short wires when making a routing connection.

For custom CMOS, we approximate the delay of a buffered wire using a lumped-capacitance model with interconnect and transistor parameters from the International Technology Roadmap for Semiconductors (ITRS) 2007 report [50]. The ITRS 2007 data could be pessimistic when applied to high-performance CMOS processes used in processors, as Intel's 65-nm process uses larger pitch and wire thicknesses than the ITRS parameters, and thus reports lower wire delays [22]. On the Stratix III FPGA, point-to-point delay is measured using the delay between two manually placed registers with automated routing, with the delay of the register itself subtracted out We assume that LABs on the Stratix III FPGA have an aspect ratio (the vertical/horizontal ratio of delay for each LAB) of 1.6 because it gives a good delay versus Manhattan distance fit.

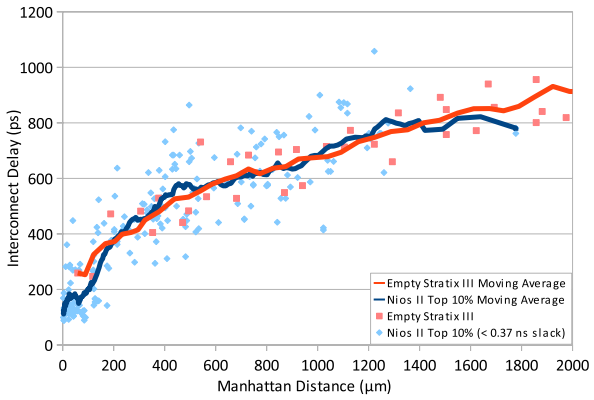
Fig. 8 plots the point-to-point wire delays for custom CMOS and FPGA wires versus the length of the wire. The delay for short wires (under 20 mm) is dominated by the delay of the driver and load buffers (i.e., one FO1 delay). These delays may be optimistic for global wires because we do not include the delay of the vias required to access the top layers of wiring. The FPGA point-to-point wire delays are plotted as Stratix III. FPGA short local wires (100 mm) have a delay ratio around 9× compared with local wires of the same length. Long wire delay (above 10 000 mm) is quite close (2×) to CMOS for the same length of wire.

When trying to measure the impact of wire delays on a circuit, routing delays are more meaningful when distance is normalized to the amount of logic that can be reached. To approximate logic density-normalized routing delays, we adjust the FPGA routing distance by the square root of the FPGAs overall area overhead versus custom CMOS ($\sqrt{23\times} = 4.8\times$). That is, a circuit implemented on an FPGA will need to use wires that are 4.8× longer than the equivalent circuit implemented in custom CMOS.

The logic density-normalized routing delays are plotted as Stratix III area adjusted in Fig. 8. Short local FPGA wires (100 mm) have a logic density-normalized delay ratio of 20×, while long global wires (7 500 mm) have a delay ratio of only 9×. The short wire delay ratio is comparable with the overall



(a)



(b)

Fig. 9. Comparing interconnect delays between an empty FPGA and soft processors. (a) SPARC T1. (b) Nios II/f.

delay ratio for full processors, but the long wire delay ratio is half of that, suggesting that FPGAs are less affected by long wire delays than custom CMOS.

2) FPGA Routing Congestion: Section IV-I.1 compared FPGA versus custom CMOS point-to-point routing delays in an uncongested chip. These delays could be optimistic compared with routing delays in real circuits where congestion causes routes to take suboptimal paths. This section shows how much FPGA routing delay changes from the ideal point-to-point delays due to congestion found in real FPGA designs.

To measure the impact of congestion, we compare the delay of route connections found on near-critical paths in a soft processor to the delay of routes traveling the same distance on an empty FPGA. We synthesized two soft processors for this measurement: The OpenSPARC T1, a large soft processor, and the Nios II/f, a small soft processor specifically designed for FPGA implementation. We extracted register-to-register timing paths that had delay greater than 90% of the critical path delay (i.e., the top 10% of near-critical paths). Timing paths are made up of one or more connections, where each connection is a block driving a net (routing wires) and terminating at another block's input. For each connection in the top 10% of paths, we observed its delay as reported by the Quartus timing analyzer and its Manhattan distance calculated by placement locations of the source and destination blocks.

The resulting delay versus distance plots are shown in Fig. 9(a) for the OpenSPARC T1 and Fig. 9(b) for the

TABLE XII
OFF-CHIP DRAM LATENCY AND THROUGHPUT. LATENCY ASSUMES CLOSED-PAGE RANDOM ACCESSES

	Custom CMOS	FPGA [51]	Ratio
DDR2 Frequency (MHz)	533	400	1.3
DDR3 Frequency (MHz)	800	533	1.5
Read Latency (ns)	55-65	85	1.4

Nios II/f. The empty-chip measurements are the same as those from the preceding section (Fig. 8). The larger size of the OpenSPARC T1 results in many longer distance connections, while the longest connection within the top 10% of paths in the small Nios II/f has a distance of 1800 mm or about the width of 15 LAB columns. We see from these plots that the amount of congestion found in typical soft processors does not appreciably impact the routing delays for near-critical routes, and that routing congestion does not alter our conclusion in the preceding section that FPGA long wire routing delays are relatively low.

J. Off-Chip Large-Scale Memory

Table XII gives a brief overview of off-chip dynamic RAM (DRAM) latency and bandwidth as commonly used in processor systems. Random read latency is measured on Intel DDR2 and DDR3 systems with off-chip (65 ns) and on-die (55 ns) memory controllers. FPGA memory latency is calculated as the sum of the memory controller latency and closed-page DRAM access time [51]. While these estimates do not account for real access patterns, they are enough to show that off-chip latency and throughput ratios between custom CMOS and FPGA are far lower than for any of the in-core circuits discussed above.

K. Summary of Building Block Circuits

A summary of our estimates for the FPGA versus custom CMOS delay and area ratios is given in Table XIII. Note that the range of delay ratios (from 7–75×) is smaller than the range of area ratios (from 2–210×). The multiplexer circuit has the highest delay ratios. Hard blocks used to support specific circuit types have only a small impact on delay ratios, but they considerably impact the area efficiency of SRAM, adders, and multiplier circuits. Multiplexers and CAMs are particularly area inefficient.

Reference [4] reported an average of 3.0–3.5× delay ratio and 18–35× area ratio for FPGA versus standard cell ASIC for a set of complete circuits. Although we expect both ratios to be higher when comparing FPGA against custom CMOS, our processor core delay ratios are higher but area ratios are slightly lower, which is initially surprising. We believe this is likely due to custom processors being optimized more for delay at the expense of area compared with typical standard cell circuits.

V. IMPACT ON PROCESSOR MICROARCHITECTURE

Section IV measured the area and delay differences between different circuit types targeting both custom CMOS and FPGAs. In this section, we relate those differences to the microarchitectural design of circuits in the two technologies. It is important to note that area is often a primary concern in the FPGA space, given the high area cost of programmability, leading to lower logic densities and high relative costs of the devices. In addition, the above results show that the area ratios between different circuit types vary over a larger range (2–200×) than the delay ratios (7–75×). For both of these reasons, we expect that area considerations will have a stronger impact on microarchitecture than delay.

TABLE XIII
DELAY AND AREA RATIO SUMMARY

Design	Delay Ratio	Area Ratio
Processor Cores	18 - 26	17 - 27
SRAM 1rw	7 - 10	2 - 5
SRAM 4r2w LUTs / LVT	15 / 9	179 / 12
CAM	14	100 - 210
Multiplier	17 - 22	4.5 - 7.0
Adder	15 - 20	4.5 - 7.0
Multiplexer	20 - 75	> 100
Pipeline latch	12 - 19	-
Routing	9 - 20	-
Off-Chip Memory	1.3 - 1.5	-

The building blocks we measured cover many of the circuit structures used in microprocessors.

1) SRAMs are very common, but take on different forms. Caches are usually low port count and high-density SRAMs. Register files use high port count, require higher speed, and are lower total capacity. RAM structures are also found in various predictors (branch direction and target, memory load dependence), and in various buffers and queues used in out-of-order microarchitectures (ROB, register rename table, and register free lists).
2) CAMs can be found in high-associativity caches and TLBs. In out-of-order processors, CAMs can also be used for register renaming, memory store queue address matching, and instruction scheduling (in reservation stations). Most of these can be replaced by RAMs, although store queues and instruction scheduling are usually CAM based.
3) Multipliers are typically found only in ALUs (both integer and floating point).
4) Adders are also found in ALUs. Addition is also used for address generation, and in miscellaneous places such as the branch target address computation.
5) Small multiplexers are commonly scattered within random logic in a processor. Larger, wider multiplexers can be found in the bypass networks near the ALUs.
6) Pipeline latches and registers delimit the pipeline stages (which are used to reduce the cycle time) in pipelined processors.

We begin with general suggestions applicable to all processors, then discuss issues specific to out-of-order processors. Our

focus on out-of-order processors is driven by the desire to improve soft processor performance given the increasing logic capacity of new generations of FPGAs, while also preserving the ease of programmability of the familiar single-threaded programming model.

A. Pipeline Depth

Pipeline depth is a fundamental choice in the design of a processor microarchitecture. Increasing pipeline depth results in higher clock speeds, but with diminishing returns due to pipeline latch delays. Hartstein *et al.* [48] show that the optimal processor pipeline depth for performance is proportional to $\sqrt{t_p/t_o}$, where t_p is the total logic delay of the processor pipeline and t_o is the delay overhead of a pipeline latch. Other properties of a processor design, such as branch prediction accuracy, the presence of out-of-order execution, or issue width, also affect the optimal pipeline depth, but these properties depend on microarchitecture, not implementation technology. The implementation technology-dependent parameters t_o and t_p have a similar effect on the optimal pipeline depth for different processor microarchitectures, and these are the only two parameters that change when comparing implementations of the same microarchitecture on two different implementation technologies (custom CMOS versus FPGA).

Section IV-H showed that the delay ratio of registers (which is the t_o of the FPGA versus the t_o custom CMOS, measured as $\sim 15\times$) is lower than the delay ratio of a complete processor (which is roughly[4] the t_p of the processor on the FPGA versus the t_p of a custom CMOS processor, $\sim 22\times$), increasing t_p/t_o on FPGA. The change in t_p/t_o is roughly (22/15), suggesting soft processors should have pipeline depths roughly 20% longer compared with an equivalent microarchitecture implemented in custom CMOS. Today's soft processors prefer short pipelines [52] because soft processors had low complexity and have low t_p, and not due to a property of the FPGA substrate. In addition, pipeline registers are nearly free in area in many FPGA designs because most designs consume more logic cells (LUTs) than registers, further encouraging deeper pipelines in soft processors.

B. Interconnect Delay and Partitioning of Structures

The portion of a chip that can be reached in a single clock cycle is decreasing with each newer process generation, while transistor switching speeds continue to improve. This leads to microarchitectures that partition large structures into smaller ones. This could be dividing the design into clusters (such as grouping a register file with ALUs into a cluster and requiring extra latency to communicate between clusters) or employing multiple cores to avoid global, one-cycle, communication [7].

In Section IV-I, we observed that after adjustment for the reduced logic density of FPGAs, long wires have a delay ratio roughly half that of a full processor core. The relatively faster long wires lessen the impact of global communication, reducing the need for aggressive partitioning of designs for FPGAs. In practice, FPGA processors have less logic complexity than high-performance custom processors, further reducing the need to partition.

[4]The value of t_p is the total propagation delay of a processor with the pipeline latches removed, and is not easily measured. It can be approximated by the product of the number of pipeline stages (N) and cycle time if we assume perfectly balanced stages. The cycle time includes both logic delay (t_p/N) and latch overhead (t_o) components for each pipeline stage, but since we know the custom CMOS versus FPGA t_o ratio is smaller than the cycle time ratio, using the cycle time ratio as an estimate of the t_p ratio results in a slight underestimate of the t_p ratio.

C. ALUs and Bypassing

Multiplexers consume much more area ($>100\times$) on FPGAs than custom CMOS (Section IV-G), making bypass networks that shuffle operands between functional units more expensive on FPGAs. On the other hand, the functional units themselves are often composed of adders and multipliers and have a lower 4.5–7$\times$ area ratio. The high cost of multiplexers reduces the area benefit of using multiplexers to share these functional units.

There are processor microarchitecture techniques that reduce the size of operand-shuffling networks relative to the number of ALUs. Fused ALUs that perform two or more dependent operations at a time increase the amount of computation relative to operand shuffling, such as the common fused multiply-accumulate unit and interlock collapsing ALUs [53], [54]. Other proposals cluster instructions together to reduce the communication of operand values to instructions outside the group [55], [56]. These techniques may benefit soft processors more than hard processors.

D. Cache Organization

Set-associative caches have two common implementation styles. Low-associativity caches replicate the cache tag RAM and access them in parallel, while high-associativity caches store tags in CAMs. High-associativity caches are more expensive on FPGAs because of the high area cost of CAMs (100–210$\times$ bit density ratio). In addition, custom CMOS caches built from tag CAM and data RAM blocks can have the CAMs decoded match lines directly drive the RAMs word lines, while an FPGA CAM must produce encoded outputs that are then decoded by the SRAM, adding a redundant encode–decode operation that was not included in the FPGA circuits in Section IV-D (we assumed CAMs with decoded outputs). In comparison, custom CMOS CAMs have minimal delay and 2–3$\times$ area overhead compared with RAM allowing for high-associativity caches (with a CAM tag array and RAM data array) to have an amortized area overhead of around 10%, with minimal change in delay compared with lower associativity set-associative caches [57].

CAM-based high-associativity caches are not area efficient in FPGA soft processors and hence soft processor caches should have lower associativity than similar hard processors. Soft processor caches should also be of higher capacity than those of similar hard processors because of the good area efficiency of FPGA SRAMs (2–5$\times$ density ratio).

E. Memory System Design

The lower area cost of block RAM encourages the use of larger caches, reducing cache miss rates and lowering the demand for off-chip DRAM bandwidth. The lower clock

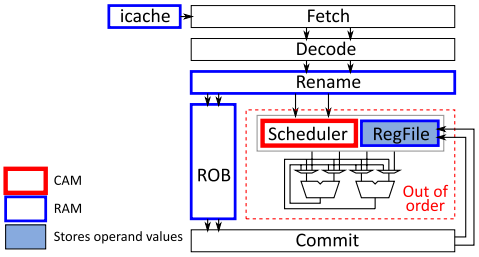


Fig. 10. Typical out-of-order processor microarchitecture.

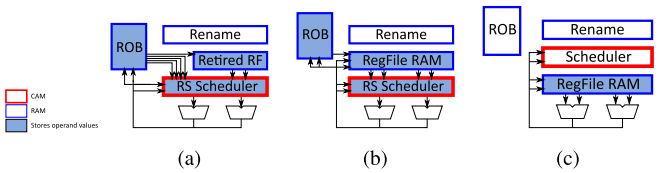


Fig. 11. Out-of-order processor microarchitecture variants. (a) Intel P6. (b) AMD K7. (c) PRF.

speeds of FPGA circuits further reduce off-chip bandwidth demand. The latency and bandwidth of off-chip memory are only slightly worse on FPGAs than custom CMOS processors as they use essentially the same commodity DRAMs.

Hard processors use many techniques to improve memory system performance, such as DRAM access scheduling, non-blocking caches, prefetching, memory dependence speculation, and out-of-order memory accesses. The lower off-chip memory system demands on FPGA soft processors suggest that more resources should be dedicated to improving the performance of the processor core than memory bandwidth or tolerating latency.

F. Out-of-Order Microarchitecture

Superscalar out-of-order processors are more complex than single-issue in-order processors. The larger number of instructions and operands in flight increase multiplexer and CAM use, leading to the common expectation that out-of-order processors would be disproportionately expensive on FPGAs, and therefore not a suitable choice for use in soft processors. However, Section IV-A suggests that processor complexity does not have a strong correlation with FPGA versus custom CMOS area ratio: even when not specifically FPGA optimized, the multiple-issue out-of-order Nehalem processor has an area ratio similar to the three in-order designs, suggesting that out-of-order and in-order processor designs appear equally suited for FPGA implementation. One possible explanation is that, for issue widths found in current processors, most of the area in a complex out-of-order processor is not spent on the CAM-like schedulers and multiplexer-like bypass networks, even though these structures are often high power, timing critical, and scale poorly to very wide issue widths. The small size of the CAMs and multiplexers mean that even particularly high-area ratios for CAMs and multiplexers cause only a small impact to the area of the whole processor core.

Fig. 10 shows the high-level organization of a typical out-of-order processor. Fetch, decode, register rename, and instruction commit are done in program order. The ROB tracks instructions as they progress through the out-of-order section of the processor. Out-of-order execution usually includes a CAM-based instruction scheduler, a register file, some execution units (ALUs), and bypass networks. The memory load/store units and the memory hierarchy are not shown in this diagram.

There are several styles of microarchitectures commonly used to implement precise interrupt support in pipelined or out-of-order processors and many variations are used in modern processors [58], [59]. The main variations between the microarchitecture styles concern the organization of the ROB, register renaming logic, register file, and instruction scheduler and whether each component uses a RAM- or CAM-based implementation. Some common organizations used in recent out-of-order processors are shown in Fig. 11. These organizations have important implications on the RAM and CAM size and port counts used by a processor.

The Intel P6-derived microarchitectures (from Pentium Pro to Nehalem) use reservation stations and a separate committed register file (Fig. 11(a)) [60]. Operand values are stored in one of three places: retired register file, ROB, or reservation stations. The retired register file stores register values that are already committed. The ROB stores register values that are produced by completed, but not committed, instructions. When an instruction is dispatched, it reads any operands that are available from the retired register file (already committed) or ROB (not committed), stores the values in the reservation station entry, and waits until the remaining operand values become available on the bypass networks. When an instruction commits, its result value is copied from the ROB into the retired register file. This organization requires several multiported RAM structures (ROB and retired register file) and a scheduler CAM that stores operand values (any number of waiting instructions may capture a previous instruction's result).

The organization used in the AMD K7 and derivatives (K7–K10) unifies the speculative (future file) and retired register files into a single multiported RAM structure (labeled RegFile RAM in Fig. 11(b) [61]). Like the P6, register values are stored in three places: ROB, register file, and reservation stations. Unlike the P6, dispatching instructions only need to read the future file RAM but not from the ROB. However, result values are still written into the ROB, and, like the P6, are copied into the register file when an instruction commits. Using a combined future file and register file reduces the number of read ports required for the ROB (the ROB is read only for committing results), but increases the number of read ports for the register file. Like the P6, the K7 uses a reservation station scheduler that stores operand values in a CAM. For FPGA implementations, the K7 organization seems to be a slight improvement over the P6 because only the register file is highly multiported (the ROB only needs multiple write ports and sequential read for commit), and the total number of RAM ports is reduced slightly.

The physical register file (PRF) organization (Fig. 11(c)) has been used in many hard processor designs, such as the MIPS R10K, IBM Power4, Power5, and Power7, Intel Pentium 4 and Sandy Bridge, DEC 21264, and AMD Bobcat and

Bulldozer [62]–[67]. In a PRF organization, operand values are stored in one central register file. Both speculative and committed register values are stored in the same structure. The register renamer explicitly renames architectural register numbers into indices into the PRF, and must be able to track which physical registers are in use and roll back register mappings during a pipeline flush. After an instruction is dispatched into the scheduler, it waits until all of its operands are available. Once the instruction is chosen to be issued, it reads all of its operands from the PRF RAM or bypass networks, normally taking one extra cycle compared with the P6 and K7. The instruction's result is written back into the register file RAM and bypass networks. When an instruction commits, only the state of the register renamer needs to be updated, and there is no copying of register values as in the previous two organizations.

The PRF organization has several advantages that are particularly significant for FPGA implementations. Register values are only stored in one structure (the PRF), reducing the number of multiported structures required. In addition, the scheduler's CAM does not store operand values, allowing the area-inefficient CAM to be smaller, with operand values stored in a more area-efficient register file RAM. This organization adds some complexity to track free physical registers and an extra pipeline stage to access the PRF. FPGA RAMs have particularly low area cost (Section IV-B), but CAMs are area expensive (Section IV-D). The benefits of reducing CAM size and multiported RAMs suggest that the PRF organization would be particularly preferred for FPGA implementations.

The delay ratio of CAMs (15×) is not particularly poor, so CAM-based schedulers are reasonable on FPGA soft processors. However, the high area cost of FPGA CAMs means scheduler capacity should be kept small. In addition to reducing the number of scheduler entries, reducing scheduler area can be done by reducing the number of entries or the amount of storage required per entry. One method is to choose an organization that does not store operand values in the CAM, like the PRF organization (Fig. 11(c)). Schedulers can be data capturing where operand values are captured and stored in the scheduler, or nondata capturing where the scheduler tracks only the availability of operands, with values fetched from the register file or bypass networks when an instruction is finally issued. Nondata capturing schedulers reduce the amount of data that must be stored in each entry of a scheduler.

The processor organizations described above all use a CAM for instruction scheduling. It may be possible to further reduce the area cost by removing the CAM. There are CAM-free instruction scheduler techniques that are not widely implemented [6], [68], but may become more favorable in soft processors. ROBs, register renaming logic, and register files have occasionally been built using CAMs in earlier processors, but are commonly implemented without CAMs.

On FPGAs, block RAMs come in a limited selection of sizes, with the smallest block RAMs commonly being 4.5–20 kb. ROBs and register files are usually even smaller in capacity but are limited by port width or count so processors on FPGAs can have larger capacity ROBs, register files, and other port-limited RAM structures at a little extra cost. In contrast, expensive CAMs limit soft processors to small scheduling windows (instruction scheduler size). Microarchitectures that address this particular problem of large instruction windows with small scheduling windows may be useful in soft processors [69].

VI. Conclusion

We have presented area and delay comparisons of processors and their building block circuits implemented on custom CMOS and FPGA substrates. In 65-nm processes, we found FPGA implementations of processor cores have 18–26× greater delay and 17–27× greater area usage than the same processors in custom CMOS. The FPGA versus custom CMOS delay ratios for most of the processor building block circuits fall within the relatively narrow delay ratio range for complete processor cores, but area ratios have much wider variation. Building blocks such as adders and SRAMs that have dedicated hardware support on FPGAs are particularly area efficient, while multiplexers and CAMs are particularly area inefficient.

In the second part of this paper, we discussed the impact of these measurements on microarchitecture design choices. The FPGA substrate encourages soft processors to have larger, low-associativity caches, deeper pipelines, and fewer bypass networks than similar hard processors. In addition, while current soft processors tend to be in-order, out-of-order execution is a valid design option for soft processors, although scheduling windows should be kept small and a PRF organization should be used to reduce the area impact of using a CAM-based instruction scheduler.

References

[1] H. Wong, *et al.*, "Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture," in *Proc. FPGA*, 2011, pp. 5–14.
[2] *Nios II Processor*, Altera, San Jose, CA, USA, May 2011.
[3] *MicroBlaze Soft Processor*, Xilinx, San Jose, CA, USA, Apr. 2004.
[4] I. Kuon, *et al.*, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.
[5] D. Chinnery, *et al.*, *Closing the Gap Between ASIC & Custom, Tools and Techniques for High-Performance ASIC Design*. Norwell, MA, USA: Kluwer, 2002.
[6] S. Palacharla, *et al.*, "Complexity-effective superscalar processors," *SIGARCH Comput. Archit. News*, vol. 25, no. 2, pp. 206–218, May 1997.
[7] V. Agarwal, *et al.*, "Clock rate versus IPC: The end of the road for conventional microarchitectures," *SIGARCH Comp. Archit. News*, vol. 28, no. 2, pp. 248–259, May 2000.
[8] P. Metzgen, *et al.*, "Multiplexer restructuring for FPGA implementation cost reduction," in *Proc. 42nd DAC*, Jun. 2005, pp. 421–426.
[9] P. Metzgen, "A high performance 32-bit ALU for programmable logic," in *Proc. 12th Int. Symp. FPGA*, 2004, pp. 61–70.
[10] P. H. Wang, *et al.*, "Intel Atom processor core made FPGA-synthesizable," in *Proc. FPGA*, 2009, pp. 209–218.
[11] S.-L. L. Lu, *et al.*, "An FPGA-based Pentium in a complete desktop system," in *Proc. FPGA*, 2007, pp. 53–59.
[12] S. Tyagi, *et al.*, "An advanced low power, high performance, strained channel 65 nm technology," in *Proc. IEEE IEDM*, Dec. 2005, pp. 245–247.
[13] K. Mistry, *et al.*, "A 45-nm logic technology with high-k+metal gate transistors, strained silicon, 9 Cu interconnect layers, 193 nm dry patterning, and 100% Pb-free packaging," in *Proc. IEEE IEDM*, Dec. 2007, pp. 247–250.
[14] A. S. Leon, *et al.*, "A power-efficient high-throughput 32-thread SPARC processor," *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 295–304, Jan. 2007.

[15] U. Nawathe, *et al.*, "Implementation of an 8-core, 64-thread, power-efficient SPARC server on a chip," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 6–20, Jan. 2008.

[16] G. Gerosa, *et al.*, "A sub-2 W low power IA processor for mobile internet devices in 45 nm high-k metal gate CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 73–82, Jan. 2009.

[17] G. Schelle, *et al.*, "Intel Nehalem processor core made FPGA synthesizable," in *Proc. FPGA*, 2010, pp. 3–12.

[18] R. Kumar, *et al.*, "A family of 45 nm IA processors," in *Proc. IEEE ISSCC*, Feb. 2009, pp. 58–59.

[19] Sun Microsystems. (2010). *OpenSPARC*, Santa Clara, CA, USA [Online]. Available: http://www.opensparc.net/

[20] J. Davis, *et al.*, "A 5.6 GHz 64 kB dual-read data cache for the POWER6 processor," in *Proc. IEEE ISSCC*, Feb. 2006, pp. 2564–2571.

[21] M. Khellah, *et al.*, "A 4.2 GHz 0.3 mm^2 256 kB dual-V_{CC} SRAM building block in 65-nm CMOS," in *Proc. IEEE ISSCC*, Feb. 2006, pp. 2572–2581.

[22] P. Bai, *et al.*, "A 65-nm logic technology featuring 35 nm gate lengths, enhanced channel strain, 8 Cu interconnect layers, low-k ILD and 0.57 μm^2 SRAM cell," in *Proc. IEEE IEDM*, Dec. 2004, pp. 657–660.

[23] P. Bai, *et al.*, "Foils from 'a 65-nm logic technology featuring 35 nm gate lengths, enhanced channel strain, 8 Cu interconnect layers, low-k ILD and 0.57 μm^2 SRAM cell'," in *Proc. IEEE IEDM*, 2004.

[24] L. Chang, *et al.*, "A 5.3 GHz 8T-SRAM with operation down to 0.41 V in 65-nm CMOS," in *Proc. VLSI Symp.*, Jun. 2007, pp. 252–253.

[25] S. Hsu, *et al.*, "An 8.8 GHz 198mW 16×64b 1R/1W variation-tolerant register file in 65-nm CMOS," in *Proc. IEEE ISSCC*, Feb. 2006, pp. 1785–1797.

[26] S. Thoziyoor, *et al.*, "CACTI 5.1," HP Lab., Palo Alto, CA, USA, Tech. Rep. HPL-2008-20, Apr. 2008.

[27] C. E. LaForest, *et al.*, "Efficient multi-ported memories for FPGAs," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. FPGA*, 2010, pp. 41–50.

[28] K. Pagiamtzis, *et al.*, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[29] K. McLaughlin, *et al.*, "Exploring CAM design for network processing using FPGA technology," in *Proc. AICT-ICIW*, Feb. 2006, paper 84.

[30] J.-L. Brelet, *et al.*, "Using Virtex-II block RAM for high performance read/write CAMs," Xilinx, San Jose, CA, USA, Tech. Rep. XAPP260, May 2002.

[31] I. Arsovski, *et al.*, "Self-referenced sense amplifier for across-chip-variation immune sensing in high-performance content-addressable memories," in *Proc. IEEE CICC*, Sep. 2006, pp. 453–456.

[32] D. W. Plass, *et al.*, "IBM POWER6 SRAM arrays," *IBM J. Res. Develop.*, vol. 51, no. 6, pp. 747–756, 2007.

[33] W. Hu, *et al.*, "Godson-3: A scalable multicore RISC processor with x86 emulation," *IEEE Micro*, vol. 29, no. 2, pp. 17–29, Mar./Apr. 2009.

[34] A. Agarwal, *et al.*, "A dual-supply 4 GHz 13fJ/bit/search 64×128b CAM in 65 nm CMOS," in *Proc. 32nd ESSCIRC*, Sep. 2006, pp. 303–306.

[35] S. Hsu, *et al.*, "A 110 GOPS/W 16-bit multiplier and reconfigurable PLA loop in 90-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 256–264, Jan. 2006.

[36] W. Belluomini, *et al.*, "An 8 GHz floating-point multiply," in *Proc. IEEE ISSCC*, vol. 1. Feb. 2005, pp. 374–604.

[37] J. Kuang, *et al.*, "The design and implementation of double-precision multiplier in a first-generation CELL processor," in *Proc. ICIDT*, May 2005, pp. 11–14.

[38] P. Jamieson, *et al.*, "Mapping multiplexers onto hard multipliers in FPGAs," in *Proc. 3rd Int. IEEE-NEWCAS*, Jun. 2005, pp. 323–326.

[39] A. Agah, *et al.*, "Tertiary-tree 12-GHz 32-bit adder in 65 nm technology," in *Proc. IEEE ISCAS*, May 2007, pp. 3006–3009.

[40] S. Kao, *et al.*, "A 240 ps 64b carry-lookahead adder in 90 nm CMOS," in *Proc. IEEE ISSCC*, Feb. 2006, pp. 1735–1744.

[41] S. B. Wijeratne, *et al.*, "A 9-GHz 65-nm Intel Pentium 4 processor integer execution unit," *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 26–37, Jan. 2007.

[42] X. Y. Zhang, *et al.*, "A 270 ps 20 mW 108-bit end-around carry adder for multiply-add fused floating point unit," *Signal Process. Syst.*, vol. 58, no. 2, pp. 139–144, Feb. 2010.

[43] K. Vitoroulis, *et al.*, "Performance of parallel prefix adders implemented with FPGA technology," in *Proc. IEEE NEWCAS Workshop*, Aug. 2007, pp. 498–501.

[44] M. Alioto, *et al.*, "Interconnect-aware design of fast large fan-in CMOS multiplexers," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 6, pp. 484–488, Jun. 2007.

[45] *Stratix III Device Handbook Volume 1*, Altera, San Jose, CA, USA, 2009.

[46] D. Lewis, *et al.*, "The Stratix II logic and routing architecture," in *Proc. 13th Int. Symp. FPGA*, 2005, pp. 14–20.

[47] E. Sprangle, *et al.*, "Increasing processor performance by implementing deeper pipelines," *SIGARCH Comput. Archit. News*, vol. 30, no. 2, pp. 25–34, 2002.

[48] A. Hartstein, *et al.*, "The optimum pipeline depth for a microprocessor," *SIGARCH Comput. Archit. News*, vol. 30, no. 2, pp. 7–13, May 2002.

[49] M. S. Hrishikesh, *et al.*, "The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays," in *Proc. ISCA*, 2002, pp. 14–24.

[50] (2007). *International Technology Roadmap for Semiconductors* [Online]. Available: http://www.itrs.net/Links/2007ITRS/Home2007.htm

[51] *External Memory Interface Handbook, Volume 3*, Altera, San Jose, CA, USA, Nov. 2011.

[52] P. Yiannacouras, *et al.*, "The microarchitecture of FPGA-based soft processors," in *Proc. CASES*, 2005, pp. 202–212.

[53] N. Malik, *et al.*, "Interlock collapsing ALU for increased instruction-level parallelism," *SIGMICRO Newslett.*, vol. 23, nos. 1–2, pp. 149–157, Dec. 1992.

[54] J. Phillips, *et al.*, "High-performance 3-1 interlock collapsing ALU's," *IEEE Trans. Comput.*, vol. 43, no. 3, pp. 257–268, Mar. 1994.

[55] P. G. Sassone, *et al.*, "Dynamic strands: Collapsing speculative dependence chains for reducing pipeline communication," in *Proc. 37th MICRO*, Dec. 2004, pp. 7–17.

[56] A. W. Bracy, "Mini-graph processing," Ph.D. dissertation, Dept. Comput. Inf. Sci., Univ. Pennsylvania, Philadelphia, PA, USA, 2008.

[57] M. Zhang, *et al.*, "Highly-associative caches for low-power processors," in *Proc. 33rd Int. Symp. Microarchit. Appears Kool Chips Workshop*, Dec. 2000, pp. 1–6.

[58] J. Smith, *et al.*, "Implementing precise interrupts in pipelined processors," *IEEE Trans. Comput.*, vol. 37, no. 5, pp. 562–573, May 1988.

[59] G. Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers," *IEEE Trans. Comput.*, vol. 39, no. 3, pp. 349–359, Mar. 1990.

[60] L. Gwennap, "Intel's P6 uses decoupled superscalar design," *Microprocessor Rep.*, vol. 9, no. 2, pp. 9–15, Feb. 1995.

[61] M. Golden, *et al.*, "A seventh-generation x86 microprocessor," *IEEE J. Solid-State Circuits*, vol. 34, no. 11, pp. 1466–1477, Nov. 1999.

[62] K. Yeager, "The MIPS R10000 superscalar microprocessor," *IEEE Micro*, vol. 16, no. 2, pp. 28–41, Apr. 1996.

[63] G. Hinton, *et al.*, "A 0.18-μm CMOS IA-32 processor with a 4-GHz integer execution unit," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1617–1627, Nov. 2001.

[64] T. N. Buti, *et al.*, "Organization and implementation of the register-renaming mapper for out-of-order IBM POWER4 processors," *IBM J. Res. Develop.*, vol. 49, no. 1, pp. 167–188, Jan. 2005.

[65] R. Kalla, *et al.*, "IBM Power5 chip: A dual-core multithreaded processor," *IEEE Micro*, vol. 24, no. 2, pp. 40–47, Mar./Apr. 2004.

[66] B. Burgess, *et al.*, "Bobcat: AMD's low-power x86 processor," *IEEE Micro*, vol. 31, no. 2, pp. 16–25, Mar./Apr. 2011.

[67] M. Golden, *et al.*, "40-entry unified out-of-order scheduler and integer execution unit for the AMD Bulldozer x86–64 core," in *Proc. IEEE ISSCC*, Feb. 2011, pp. 80–82.

[68] F. J. Mesa-Martínez, *et al.*, "SEED: Scalable, efficient enforcement of dependences," in *Proc. PACT*, Sep. 2006, pp. 254–264.

[69] M. Pericas, *et al.*, "A decoupled KILO-instruction processor," in *Proc. 12th Int. Symp. HPCA*, Feb. 2006, pp. 53–64.

Henry Wong is pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada.

Vaughn Betz is an Associate Professor of the Electrical and Computer Engineering Department, University of Toronto, Toronto, ON, Canada. He was previously a Senior Director of Software Engineering at Altera, where he was one of the architects of the Quartus II CAD system and the Stratix and Cyclone FPGA families.

Jonathan Rose (F'09) is a Professor with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. He has worked in the area of FPGA CAD and architecture for over 20 years, including stints at the two major vendors, Xilinx and Altera, as well as a startup.

Prof. Rose is a Fellow of the ACM and a Foreign Associate of the American National Academy of Engineering.