Comparing FPGA vs. Custom CMOS and the impact on processor microarchitecture

Conference Paper · January 2011

DOI: 10.1145/1950413.1950419 · Source: DBLP

3 authors:

Henry Wong
University of Toronto
11 PUBLICATIONS **1,803** CITATIONS
SEE PROFILE

Vaughn Betz
University of Toronto
189 PUBLICATIONS **5,572** CITATIONS
SEE PROFILE

Julie L Rose
University of Toledo
163 PUBLICATIONS **10,868** CITATIONS
SEE PROFILE

Comparing FPGA vs. Custom CMOS and the Impact on Processor Microarchitecture

Henry Wong
Department of Electrical and Computer Engineering
University of Toronto
henry@eecg.utoronto.ca

Vaughn Betz
Altera Corp.
vbetz@altera.com

Jonathan Rose
Department of Electrical and Computer Engineering
University of Toronto
jayar@eecg.utoronto.ca

ABSTRACT

As soft processors are increasingly used in diverse applications, there is a need to evolve their microarchitectures in a way that suits the FPGA implementation substrate. This paper compares the delay and area of a comprehensive set of processor building block circuits when implemented on custom CMOS and FPGA substrates. We then use the results of these comparisons to infer how the microarchitecture of soft processors on FPGAs should be different from hard processors on custom CMOS.

We find that the ratios of the area required by an FPGA to that of custom CMOS for different building blocks varies significantly more than the speed ratios. As area is often a key design constraint in FPGA circuits, area ratios have the most impact on microarchitecture choices. Complete processor cores have area ratios of 17-27× and delay ratios of 18-26×. Building blocks that have dedicated hardware support on FPGAs such as SRAMs, adders, and multipliers are particularly area-efficient (2-7× area ratio), while multiplexers and CAMs are particularly area-inefficient (> 100× area ratio), leading to cheaper ALUs, larger caches of low associativity, and more expensive bypass networks than on similar hard processors. We also find that a low delay ratio for pipeline latches (12-19×) suggests soft processors should have pipeline depths 20% greater than hard processors of similar complexity.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles

General Terms

Design, Measurement

Keywords

FPGA, CMOS, Area, Delay, Soft Processor

1. INTRODUCTION

Custom CMOS silicon logic processes, standard cell ASICs, and FPGAs are commonly-used substrates for implementing digital circuits, with circuits implemented on each one having different characteristics such as delay, area, and power. In this paper, we compare custom CMOS and FPGA substrates for implementing processors and explore the microarchitecture trade-off space of soft processors in light of the differences.

FPGA soft processors have mostly employed single-issue in-order microarchitectures due to the limited area budget available on FPGAs [1–3]. We believe that future soft processor microarchitecture design decisions can benefit from a quantitative understanding of the differences between custom CMOS and FPGA substrates. Another reason to revisit the soft processor microarchitecture trade-off space is that the increased capacity of modern FPGAs can accommodate much larger soft processors if it results in a performance benefit.

Previous studies have measured the relative delay and area of FPGA, standard cell, and custom CMOS substrates as an average 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 circuit designs that can be implemented on FPGAs, we need to compare the relative performance of various types of "building block" circuits in order to have enough detail to guide microarchitecture design decisions.

This paper makes two contributions:

- We compare delay and area between FPGA and custom CMOS implementations of a set of building block circuits. While prior work gave results for complete systems, we show specific strengths and weaknesses of the FPGA substrate for the different building blocks.
- Based on the delay and area ratios of building block circuits on FPGAs vs. custom CMOS, we discuss how processor microarchitecture design trade-offs change on an FPGA substrate and the suitability of existing processor microarchitecture techniques when used on FPGAs.

We begin with background in Section 2 and methodology in Section 3. We then present our comparison results and their impact on microarchitecture in Sections 4 and 5, and conclude in Section 6.

2. BACKGROUND

2.1 Technology Impact on Microarchitecture

Studies on how process technology trends impact microarchitecture are essential for designing effective microarchitectures that take advantage of ever-changing manufacturing

FPGA'11, February 27–March 1, 2011, Monterey, California, USA.

processes. 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].

FPGA-implemented circuits face a very different set of constraints. Power consumption is not currently the dominant design constraint due to lower clock speeds, while area is often the primary constraint due to high area overheads of FPGA circuits. Characteristics such as inefficient multiplexers and the need to map RAM structures into FPGA hard SRAM blocks are known and generally adjusted for by modifying circuit-level, but not microarchitecture-level, design [8–11].

2.2 Measurement of FPGAs

Kuon and Rose have measured the area, delay, and power overheads of FPGAs compared to a standard cell ASIC flow on 90 nm processes [4] using a benchmark set of complete circuits to measure the overall impact of using FPGAs compared to 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. Minimum cycle time of the FPGA circuits were not significantly affected by hard blocks, ranging from 3.0 to 3.5× greater than ASIC. Chinnery and Keutzer [5] made similar comparisons between standard cell and custom CMOS and reported a delay ratio of 3 to 8×. Combined, these reports suggest that the delay of circuits implemented on an FPGA would be 9 to 28× greater than on custom CMOS. However, data for full circuits are insufficiently detailed to guide microarchitecture-level decisions.

3. 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 ratios* 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 has more overhead. 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 FPGA.

As we do not have the expertise to implement highly-optimized custom CMOS circuits, our building block circuit comparisons use data from custom CMOS implementations found in the literature. The set of building block circuits is generic enough so that there are sufficient published delay and area data. We focus mainly on custom designs built in 65 nm processes, because it is the most recent process where design examples are readily available in the literature, and compare them to the Altera Stratix III 65 nm FPGA. In most cases, we implemented the equivalent FPGA circuits for comparison. 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% FO1 delay improvement between 90 nm and 65 nm, and 23% FO2 delay improvement between 65 nm and 45 nm [12, 13]. The area and delay scaling factors used are summarized in Table 1.

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

Table 1: Normalization Factors Between Processes

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

Table 2: Estimated FPGA Resource Area Usage

Delay is measured as the longest register to register path or input to output path in a circuit. In papers that describe CMOS circuits embedded in a larger unit (e.g., a shifter inside an 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 circuits that do not include a register (e.g., wire delay).

To measure FPGA resource usage, we implement circuits on the Stratix III FPGA and use the "logic utilization" metric as reported by Quartus II to account for lookup tables (LUT) and registers, and count partially-used memory and multiplier blocks as entirely used since it is unlikely another part of the design can use them. Table 2 shows the areas of the Stratix III FPGA resources. The FPGA tile areas include routing area so we do not track routing separately. The core of the the largest Stratix III (EP3LS340) FPGA contains 13 500 LABs of 10 ALMs each, 1 040 M9K memories, 48 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 using the fastest speed grade of the largest Stratix III. Circuits containing MLABs used Stratix IV due to hardware[1] issues. Registers delimit the inputs and outputs of our circuits under test. We set timing constraints to maximize clock speed, reflecting the use of these circuits as part of a larger circuit in the FPGA core.

3.1 Additional Limitations

The trade-off space for a given circuit structure on custom CMOS is huge and our comparison circuit examples may not all have the same optimization targets. Delay, area, power, and design effort can all be traded-off, resulting in vastly different circuit performance. We assume that designs published in the literature are optimized primarily for delay with reasonable values for the other metrics and we implement our FPGA equivalents with the same approach.

Another source of imprecision results from the differences in performance between chip manufacturers. For example,

[1] A Stratix III erratum halves the MLAB capacity to 320 bits and is fixed in Stratix IV. See Stratix III Device Family Errata Sheet

transistor drive strength impacts logic gate delay and is voltage- and process-dependent. At 100 nA/µm leakage current, Intel's 65 nm process [14] achieves 1.46 and 0.88 mA/µm at 1.2 V for NMOS and PMOS, respectively, and 1.1 and 0.66 mA/µm at 1.0 V, while TSMC's 65 nm process [15] achieves 1.01 and 0.48 mA/µm at 1.0 V. The choice of supply voltage results from both the power-performance tradeoff and the transistor reliability the manufacturer can achieve.

Finally, we note that high-performance microprocessor designs tend to have generous power budgets that FPGAs do not enjoy, and this has an impact that is embedded in the area and speed comparisons.

4. FPGA VS. CUSTOM CMOS

4.1 Complete Processor Cores

We begin by comparing complete processor cores implemented on an FPGA vs. custom CMOS to provide context for the building block measurements. Table 3 shows a comparison of the area and delay of four commercial processors that have both custom CMOS and FPGA implementations and includes in-order, multithreaded, and complex 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. Overall, custom processors have delay ratios of 18-26× and area ratios of 17-27×, with no obvious trends with processor complexity.

The OpenSPARC T1 and T2 cores are derived from the Sun UltraSPARC T1 and T2, respectively [21]. Both cores are in-order multithreaded, and use the 64-bit SPARC V9 instruction set. The OpenSPARC T2 has the encryption unit from the UltraSPARC T2 removed, although we believe the change is small enough to not significantly skew our measurements.

The Intel Atom is a reduced-voltage dual-issue in-order 64-bit x86 processor with two-way multithreading. The FPGA synthesis by Wang et al. includes only the processor core without L2 cache, and occupies 85% of the largest 65 nm Virtex-5 FPGA (XC5VLX330) [10]. Our FPGA area metric assumes the core area of the largest Virtex 5 is the same as the Stratix III, 412 mm^2.

The Intel Nehalem is a high-performance multiple-issue out-of-order 64-bit x86 processor with two-way multithreading. The Nehalem's core area was estimated from a die photo. The FPGA synthesis by Schelle et al. does not include the per-core L2 cache, and occupies roughly 300% of the largest Virtex-5 FPGA [19]. Because the FPGA synthesis is split over multiple chips and runs at 520 kHz, it is not useful for estimating delay ratio.

4.2 SRAM Blocks

SRAM blocks are commonly used in processors for building caches and register files. SRAM performance can be characterized by latency and throughput. Custom SRAM designs can trade latency and throughput by pipelining, while FPGA designs are typically limited to the types of hard SRAM blocks that exist on the FPGA. SRAMs on the Stratix III FPGA can be implemented using two sizes of hard block SRAM, LUTRAM, or in registers and LUTs. Their throughput and density are compared in Table 4 to five high-performance custom SRAMs in 65 nm processes.

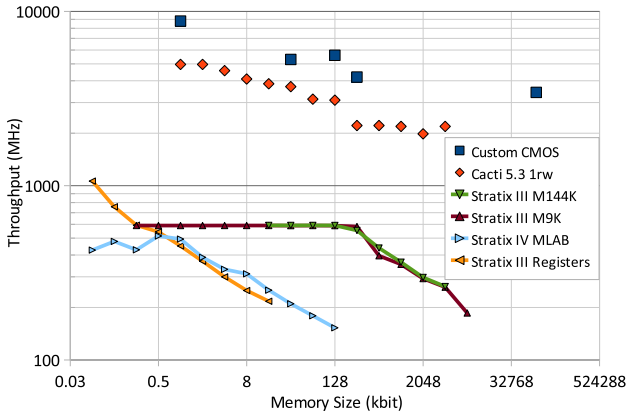

Figure 1: SRAM Throughput

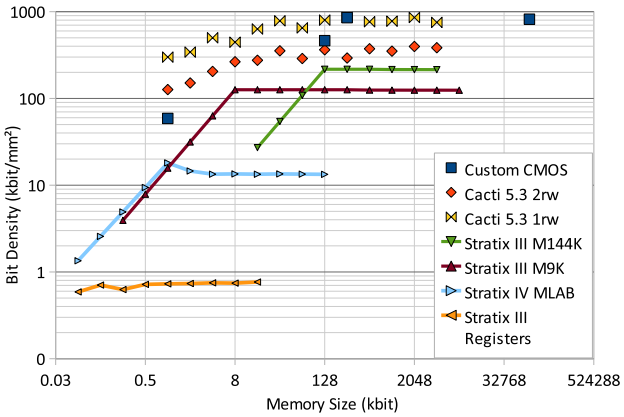

Figure 2: SRAM Density

The density and throughput of FPGA and custom SRAMs is plotted against memory size in Figures 1 and 2. The plots include data from CACTI 5.3, a CMOS memory performance and area model [27]. 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 lower because FPGA SRAM blocks have little programmability. The FPGA data above use 32-bit wide data ports that slightly underutilize the native 36-bit ports. The raw density of an FPGA SRAM block is listed in Table 4.

Below 9 kbit, the bit density of FPGA RAMs falls off nearly linearly with reducing RAM size because M9Ks are underutilized, and the MLAB density is low. For larger arrays with good utilization, FPGA SRAM arrays have a density ratio of only 2-5× vs. single read-write port (1rw) 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 CACTI's 2rw model for comparison. For arrays of similar size, the bit density of CACTI's 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 FPGA and custom 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 by using faster and larger storage cells. The 65 nm Pentium 4 register file trades 5× bit density for 9 GHz single-cycle performance

Processor	Custom CMOS f_{max} (MHz)	Custom CMOS Area (mm^2)	FPGA f_{max} (MHz)	FPGA Area (mm^2)	Logic Utilization (ALUT)	Ratios f_{max}	Ratios Area
OpenSPARC T1 (90 nm) [16]	1800	6.0	79	100	86 597	23	17
OpenSPARC T2 (65 nm) [17]	1600	11.7	88	294	250 235	18	25
Atom (45 nm) [10, 18]	>1300	12.8	50	350	85%	26	27
Nehalem (45 nm) [19, 20]	3000	51	-	1240	300%	-	24
Geometric Mean						22	23

Table 3: Complete Processor Cores. Area and delay normalized to 65 nm.

Design	Ports	Size (kbit)	f_{max} (MHz)	Area (mm^2)	Bit Density (kbit/mm^2)	Ratios f_{max}	Ratios Density
IBM 6T 65 nm [22]	2r or 1w	128	5600 1.2 V	0.276	464	9.5	2.1
Intel 6T 65 nm [23]	1rw	256	4200 1.2 V	0.3	853	7.6	3.9
Intel 6T 65 nm [14]	1rw	70 Mb	3430 1.2 V	110 [24]	820	–	–
IBM 8T 65 nm SOI [25]	1r1w	32	5300 1.2 V	–	–	9.0	–
Intel 65 nm Regfile [26]	1r1w	1	8800 1.2 V	0.017	59	18	3.3
Registers	1r1w	-	-	-	0.77		
MLAB	1r1w	640 b	∼600	0.033	23		
M9K	1r1w	9	590	0.064	142		
M144K	1r1w	144	590	0.59	244		

Table 4: Custom and FPGA SRAM Blocks

Ports	CACTI 5.3 f_{max} (MHz)	CACTI 5.3 Density ($\frac{kbit}{mm^2}$)	FPGA f_{max} (MHz)	FPGA Density ($\frac{kbit}{mm^2}$)	Ratios f_{max}	Ratios Density
2r1w	4430	109	501	15.7	9	7
4r2w	4200	35	320	0.25	13	143
6r3w	3800	17	286	0.20	13	89
8r4w	3970	9.8	269	0.15	15	65
10r5w	3740	6.3	266	0.10	14	61
12r6w	3520	4.5	249	0.090	14	51
14r7w	3330	3.4	237	0.080	14	43
16r8w	3160	2.7	224	0.068	14	39
Live Value Table (LVT)						
4r2w			420	1.50	10	23
8r4w			345	0.41	12	24

Table 5: Multiported 2 kbit SRAM. LVT data from LaForest et al. [28].

[26]. FPGA RAMs lack this flexibility, and the delay ratio is even greater (18×) for this specific use.

4.3 Multiported SRAM Blocks

FPGA hard SRAM blocks can typically implement up to two read-write ports (2rw). Increasing the number of read ports can be achieved reasonably efficiently by replicating the memory blocks but increasing the number of write ports cannot. A multi-write port RAM can be implemented using registers for storage, but it 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 indicate for each word which of the replicated memories holds the most recent copy [28]. We present data for multiported RAMs implemented using registers, LVT-based multiported memories [28], and CACTI 5.3 models of custom CMOS multiported RAMs. We focus on a 64×32-bit (2 kbit) memory array with twice as many read ports as write ports ($2N$ read, N write) because it is similar to register files in processors. Table 5 shows the throughput and density comparisons.

The custom vs. FPGA bit density ratio is 7× for 2r1w, and increases to 23× and 143× for 4r2w LVT- and register-based memories, respectively. The delay ratio is 9× for 2r1w, and increases to 10× and 13× for 4r2w LVT- and register-based memories, respectively, a smaller impact than the area ratio increase.

4.4 Content-Addressable Memories

A Content-Addressable Memory (CAM) is a logic circuit that allows associative searches of its stored contents. Custom CMOS CAMs are implemented as dense arrays using 9T to 11T cells compared to 6T in SRAM, and are about 2-3× less dense than SRAMs. Ternary CAMs use two storage cells per "bit" to store three states (0, 1, and don't-care), a capability often used for longest prefix searches in network packet routers. In processors, CAMs are used in tag arrays for high-associativity caches and TLBs. CAM-like structures are also used in out-of-order instruction schedulers. CAMs in processors require both frequent read and write capability, but not large capacities. Pagiamtzis and Sheikholeslami give a good overview of the CAM design space [29].

There are several methods of implementing CAM functionality on FPGAs [30]. CAMs implemented in soft logic use registers for storage and LUTs to read, write, and search the stored bits. Another proposal, which we call 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 [31]. The soft logic CAM is the only design that provides one-cycle writes, while the BRAM-CAM offers improved bit density with two-cycle writes. We do not consider CAM implementations with even longer write times.

Table 6 shows a variety of custom CMOS and FPGA CAM designs. Figures 3 and 4 plot these and also 8-bit

	Size (bits)	Search Time (ns)	Bit Density ($\frac{kbit}{mm^2}$)	Ratios Delay	Ratios Density
Ternary CAMs					
IBM 64×72 [32]	4 608	0.6	-	5.0	-
IBM 64×240 [32]	15 360	2.2	167	1.7	205
Binary CAMs					
POWER6 8×60 [33]	480	<0.2	-	14	-
Godson-3 64×64 [34]	4 096	0.55	76	5	99
Intel 64×128 [35]	8 192	0.25	167	14	209
FPGA Binary CAMs					
Soft logic 64×72	4 608	3.0	0.82		
Soft logic 64×240	15 360	3.8	0.81		
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		

Table 6: CAM Designs

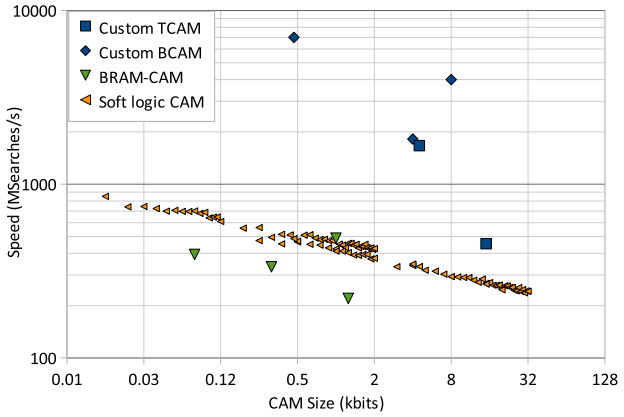

Figure 3: CAM Search Speed

and 128-bit soft logic CAMs of varying depth. Because of high power consumption, CAMs often have three competing design goals: delay, density, and energy per bit per search. CAMs can achieve delay comparable to SRAMs but at a high cost in power. Both the POWER6 TLB and Intel's 64×128 BCAM achieve at least 4 GHz, but the latter uses 13 fJ/bit/search while IBM's 450 MHz 64×240 TCAM uses 1 fJ/bit/search. While Intel's BCAM and the Godson-3 TLB are both 64-entry full-custom designs, Intel achieves twice the bit density and half the cycle time at equal energy per bit per search, highlighting the impact of good circuit design, layout, and process technology.

As shown in Table 6, soft logic CAMs have poor 100-210× bit density ratios vs. custom CAMs. Despite the low density, the delay ratio is only 14×. BRAM-CAMs built from M9Ks can offer 2.4× better density than soft logic CAMs but halved write bandwidth. Despite the higher port-width/depth aspect ratio, MLAB BRAM-CAMs have bit density worse than M9K-CAMs because of control logic overhead. The halved write bandwidth of BRAM-CAMs make 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 4), suggesting that the area inefficiency comes from using registers for storage, not the associative search itself.

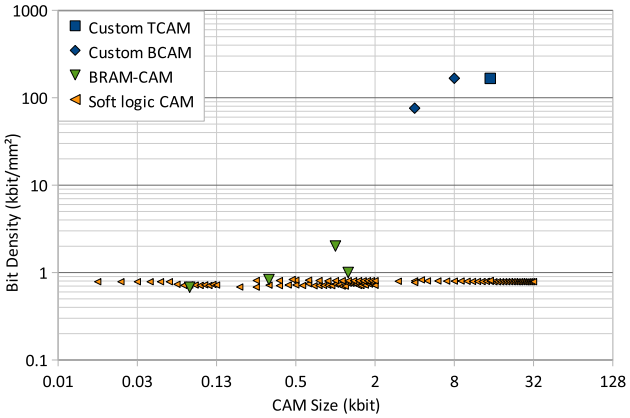

Figure 4: CAM Bit Density

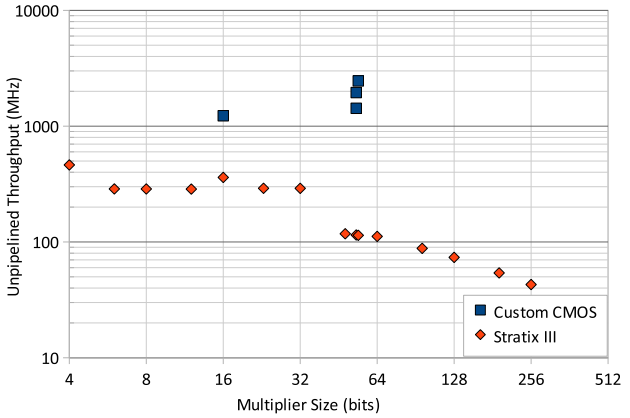

Figure 5: Multiplier Latency

4.5 Multipliers

Multiplication is an operation performed frequently in signal processing applications, but not used as often in processors. Multiplier blocks can also be used to inefficiently implement shifters and multiplexers [39].

Figure 5 shows the latency of multiplier circuits on custom CMOS and on FPGA using hard DSP blocks. Latency is 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 7 shows details of the design examples. The two IBM multipliers have latency ratios comparable to full processor cores. Intel's 16-bit multiplier design has much lower latency ratios as it appears to target low power instead of delay. Because custom multiplier de-

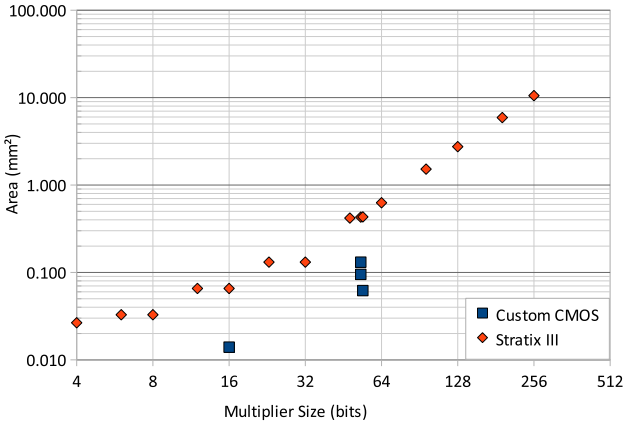

Figure 6: Multiplier Area

Design	Size	Stages	Latency (ns)	Area (mm^2)	Ratios Latency	Ratios Area
Intel 90 nm 1.3 V [36]	16×16	1	0.81	0.014	3.4	4.7
IBM 90 nm SOI 1.4 V [37]	54×54	4	0.41	0.062	22	7.0
IBM 90 nm SOI 1.3 V [38]	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		

Table 7: Multiplier area and delay, normalized to 65 nm process. Unpipelined latency is pipelined cycle time×stages.

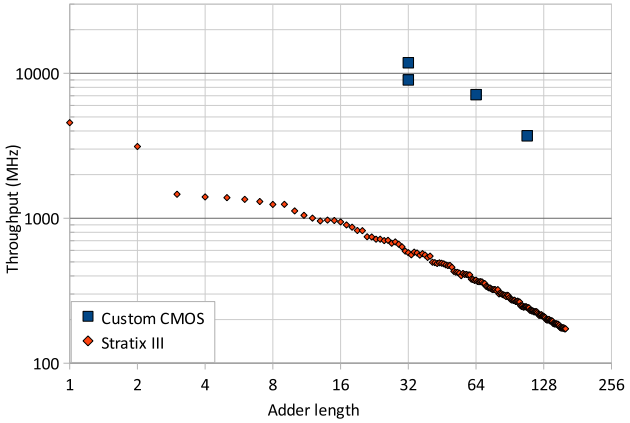

Figure 7: Adder Delay

signs are often more deeply pipelined (3 and 4 stages) than the hard multipliers on FPGAs (2 stages), throughput ratios are higher than latency ratios.

The area of the custom CMOS and FPGA multipliers are plotted in Figure 6. The area ratios of 4.5-7.0× are much lower than for full processor cores.

4.6 Adders

Custom CMOS adder circuit designs can span the area-delay trade-off space from slow ripple-carry adders to fast parallel 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 parallel adders can be implemented with soft logic and routing, the lack of dedicated circuitry means parallel adders are bigger and slower than the ripple-carry adder with hard carry chains [44].

Figure 7 plots a comparison of adder delay, with details in Table 8. The Pentium 4 delay is conservative as the delay is for the full integer ALU. FPGA adders achieve delay ratios of 15-20× and a low area ratio of around 4.5-7×.

Design	Size (bit)	f_{max} (MHz)	Area (mm^2)	Delay Ratio	Area Ratio
Agah et al. [40]	32	12 000 1.3 V	-	20	-
Kao et al. 90 nm [41]	64	7 100 1.3 V	0.016	19	4.5
Pentium 4 [42]	32	9 000 1.3 V	-	16	-
IBM [43]	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 8: Adders. Area and delay normalized to 65 nm process.

Mux Inputs	FPGA Area (mm^2)	FPGA Delay (ps)	Custom 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

Table 9: Analytical model of passive tree multiplexers [45] normalized to 65 nm process.

4.7 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 an RC analytical model and the delays of the Pentium 4 shifter unit and the Stratix III ALM. Our area ratio estimate comes from an indirect measurement using the ALM.

Table 9 shows a delay comparison between FPGA and an analytical model of switch-based tree multiplexers [45]. This passive 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 passive designs.

The 65 nm Pentium 4 integer shifter datapath [42] is dominated by small multiplexers (sizes 3, 4, and 8), but not in isolation. We implemented the datapath (Figure 8) excluding control logic on the Stratix III, with results shown in Table 10. 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. Another delay ratio estimate can be made by examining the Stratix III Adaptive Logic Module itself, as its delay consists mainly of multiplexers. Configuration RAM hold static values and do not affect delay. We implemented a circuit equivalent to an ALM as described in the Stratix III Handbook [46], 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. Each ALM input pin is modeled as a 21-to-1 multiplexer, as 21 to 30 are reasonable sizes according to Lewis et al. [47].

We examined one long path and one short path out of many possible timing paths through the ALM. The long and short timing paths begin after the input multiplexers for pins datab0 and dataf0, respectively, both terminating at the LUT register. We placed these two paths into separate clock domains to ensure they were independently optimized. Table 10 shows delay ratios of 7.1× and 11.7× for the long and short paths, respectively. These delay ratios are lower compared to previous examples due to the lower power and area budgets preventing custom FPGAs from being as aggressively delay-optimized as custom processors, and to

Circuit	FPGA Delay (ps)	Custom 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

Table 10: Delay of Multiplexer-Dominated Circuits

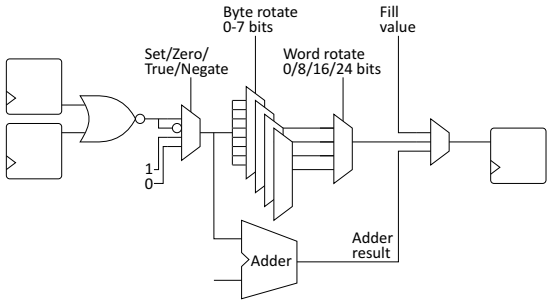

Figure 8: 65 nm Pentium 4 Integer ALU Shifter [42]

extra circuit complexity not shown in the Stratix III Handbook.

We estimate the multiplexer area ratio indirectly by again implementing the equivalent circuit of an ALM on a Stratix III. We find that our equivalent ALM consumes 104 ALUTs, or roughly 52 ALMs. An area ratio of 52× is the ratio of the area of the FPGA equivalent circuit (52 ALMs) compared to the silicon area of the custom circuit (1 ALM equivalent). The real area ratio is substantially greater, as we implemented only the ALM's 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, for an area ratio estimate of 104×. We expect this to be even higher once configuration RAM and other ALM complexities are accounted for.

Groups of multiplexers 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 is justified, especially from an area perspective.

4.8 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[2], through a pipeline stage's combinational logic, to the next set of pipeline latches. The delay of a pipeline latch (setup and clock-to-output times) impacts the speed of a circuit and the clock speed improvement when increasing pipeline depth.

The "effective" cost of inserting an extra pipeline register into LUT-based 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. Table 11 shows estimates of the delay cost of a custom CMOS pipeline stage. The 180 nm Pentium 4 design assumed 90 ps of pipeline latch delays including clock skew [48], which we

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

Design	Register Delay (ps)	Delay Ratio
Sprangle et al. [48]	35 (90 ps in 180 nm)	12
Hartstein et al. [49]	32 (2.5 FO4)	14
Hrishikesh et al. [50]	23 (1.8 FO4)	19
Geometric Mean	29.5	15
Stratix III	436	-

Table 11: Pipeline Latch Delay

scaled according to the FO1 ring oscillator delays for Intel's processes (11 ps at 180 nm to 4.25 ps at 65 nm) [14]. Hartstein et al. and Hrishikesh et al. present estimates expressed in FO4 delays, which were scaled to an estimated FO4 delay of 12.8 ps for Intel's 65 nm process.

The delay ratio for a pipeline latch is 10-15×. 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.

4.9 Point-to-Point Routing

Interconnect delay comprises a significant portion of the total delay in both FPGAs and modern CMOS processes. Figure 9 plots the point-to-point wire delays for three classes of interconnect (local, intermediate, and global), and FPGA routing delay both in absolute distance and normalized for decreased FPGA area efficiency.

We model each buffered segment as an RC delay as illustrated in Figure 10 and Equation 1, where h is the buffer transistor size and R_{tr} is the linearized buffer resistance. Buffered wires are modeled as multiple identical segments with the number of segments chosen to minimize delay, while minimum-delay buffer transistor sizing can be found by differentiating Equation 1 with respect to h. Equation 1 is modified from Bakoglu and Meindl to include transistor gate (C_g) and junction (C_d) lumped capacitances and to model 50%-to-50% delay instead of 0-to-90% [51].

$$T = R_{tr}C_d + (\frac{1}{h}R_{tr} + \frac{1}{2}R_{int})C_{int} + (\frac{1}{h}R_{tr} + R_{int})hC_g \quad (1)$$

We use interconnect and transistor parameters from the International Technology Roadmap for Semiconductors (ITRS) 2007 report [52]. We model the minimum pitch for each class of wiring (local, intermediate, global). Buffers are assumed to be inverters with 2:1 PMOS to NMOS transistor size ratio, and junction capacitance is approximated as half of gate capacitance. Gate and junction capacitances are modeled as lump capacitances and wires are modeled as distributed RC. Interconnect delays using ITRS 2007 data may be pessimistic, as Intel's 65 nm process reports lower delays using larger pitch and wire thicknesses [14].

The point-to-point delay on FPGA is measured using the delay between two manually-placed registers, with the delay of the register itself subtracted out, and includes the overhead of routing programmability. Assuming LABs have an aspect ratio (the vertical/horizontal ratio of delay for each LAB) of 1.6 gives a good delay vs. Manhattan distance fit. The physical distance is calculated from LAB coordinates using the aspect ratio and area of a LAB (Table 2).

In Figure 9, the delay of the CMOS load and driving buffers (one FO1) dominates the delay for CMOS short wire lengths under 20 µm. FPGA short local wires (100 µm) have a delay ratio around 9×. Long global wire delay is quite close

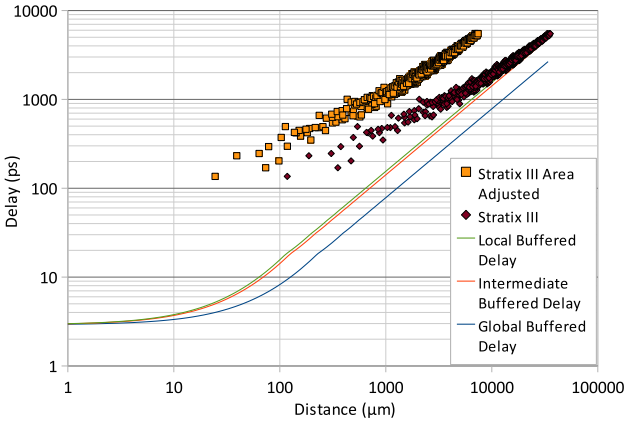

Figure 9: Point-to-Point Routing Delay

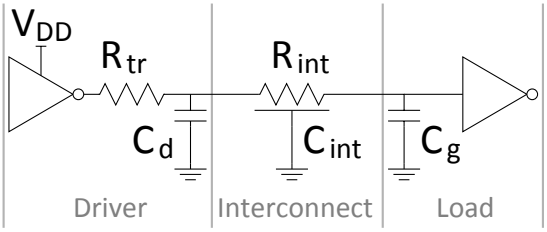

Figure 10: RC Model of Wire Delay

(2×) to CMOS for the same length of wire.

Routing delays are more meaningful when "distance" is normalized to the amount of "logic" that can be reached. To approximate this metric, we adjust the FPGA routing distance by the square-root of the FPGA's overall area overhead vs. custom CMOS ($\sqrt{23\times} = 4.8\times$). Short local FPGA wires (100 µm) have a logic density-normalized delay ratio 20×, while long global wires (7 500 µm) have a delay ratio of only 9×. The short wire delay ratio is comparable to the overall delay overhead for full processors, but the long wire delay ratio is half that, suggesting that FPGAs are less affected by long wire delays than custom CMOS.

4.10 Off-Chip Memory

Table 12 gives a brief overview of off-chip 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 the sum of the memory controller latency and closed-page DRAM access time. 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 in-core circuits like complete processors.

4.11 Summary of Building Block Circuits

A summary of our estimates for the FPGA vs. custom CMOS delay and area ratios can be found in Table 13. The range of delay ratios (8-75×) is smaller than the range of area ratios (2-210×). Multiplexers have the highest delay ratios and hard blocks only have a small impact on delay ratios. Hard blocks are particularly area-efficient (SRAM, adders, multipliers), while multiplexers and CAMs are particularly area-inefficient.

In previous work, Kuon and Rose reported an average of 3.0-3.5× delay ratio and 18-35× area ratio for FPGA vs. standard cell ASIC for a set of complete circuits [4]. 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, likely due to custom processors being optimized more for delay at the expense of area compared to typical standard cell circuits.

	Custom CMOS	FPGA [53]	Ratio
DDR2 Frequency (MHz)	533	400	1.3
DDR3 Frequency (MHz)	800	533	1.5
Read Latency (ns)	55-65	100-133	2

Table 12: Off-Chip DRAM Latency and Throughput. Latency assumes closed-page random accesses.

Design	Delay Ratio	Area Ratio
Processor Cores	18 - 26	17 - 27
SRAM 1rw	7 - 10	2 - 5
SRAM 4r2w LUTs / LVT	13 / 10	143 / 23
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 - 2	-

Table 13: Delay and Area Ratio Summary

5. IMPACT ON MICROARCHITECTURE

The area and delay differences between circuit types and substrates measured in Section 4 affects the microarchitectural design of circuits targeting custom CMOS vs. targeting FPGAs. As FPGA designs often have logic utilization (area) as a primary design constraint, we expect that area ratios will have more impact on microarchitecture than delay ratios.

5.1 Pipeline Depth

Pipeline depth is one of the fundamental choices in the design of a processor microarchitecture. Increasing pipeline depth results in higher clock speeds, but with diminishing returns due to pipeline latch delays. The analysis by Hartstein et al. shows that the optimal processor pipeline depth for performance is proportional to $\sqrt{\frac{t_p}{t_o}}$, where t_p is the total logic delay of the processor pipeline, and t_o is the delay overhead of a pipeline latch [49].

Section 4.8 showed that the delay ratio of registers (i.e., t_o FPGA vs. t_o custom CMOS, ∼15×) on FPGAs is lower than that of a complete processor (approximately t_p FPGA vs. t_p custom CMOS, ∼22×), 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 to an equivalent microarchitecture implemented in custom CMOS. Today's soft processors prefer short pipelines [54] because soft processors are simple and have low t_p, and not due to a property of the FPGA substrate.

5.2 Partitioning of Structures

The portion of a chip that can be reached in a single clock cycle is decreasing with each newer process, while transistor

switching speeds continue to improve. This leads to microarchitectures that partition large structures into smaller ones and partition the design into clusters or multiple cores to avoid global communication [7].

In Section 4.9, we observed that after adjustment for the reduced logic density of FPGAs, long wires have a delay ratio roughly half that of a 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.

5.3 ALUs and Bypassing

Multiplexers consume much more area on FPGAs than custom CMOS (Section 4.7), making bypass networks that shuffle operands between functional units more expensive on FPGAs. The functional units themselves are often composed of adders and multipliers and have a lower 4.5-7× 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 [55, 56]. Other proposals cluster instructions together to reduce the communication of operand values to instructions outside the group [57, 58]. These techniques may benefit soft processors even more than hard processors.

5.4 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× bit density ratio). In addition, custom CMOS caches built from tag CAM and data RAM blocks can have the CAM's decoded match lines directly drive the RAM's 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 circuits in Section 4.4. In comparison, custom CMOS CAMs have minimal delay and 2-3× area overhead compared to RAM allowing for high-associativity caches to have an amortized area overhead of around 10%, with minimal change in delay compared to set-associative caches [59].

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

5.5 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 bandwidth to DRAM main memory. The lower clock speeds of FPGA circuits further reduce off-chip bandwidth demand. The latency and bandwidth of off-chip memory is only slightly worse on FPGAs than on custom CMOS processors.

Low off-chip memory system demands suggest that more resources should be dedicated to improving the performance of the processor core than improving memory bandwidth or tolerating latency. Techniques used to improve the memory system in hard processors include DRAM access scheduling, non-blocking caches, prefetching, memory dependence speculation, and out of order memory accesses.

5.6 Out-of-Order Microarchitecture

Section 4.1 suggests that processor complexity does not have a strong correlation with area ratio, so out-of-order microarchitectures seem to be a reasonable method for improving soft processor performance. 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 [60, 61]. The main variations between the microarchitecture styles concern the organization of the reorder buffer, register renaming logic, register file, and instruction scheduler and whether each component uses RAM- or CAM-based implementations.

FPGA RAMs have particularly low area cost (Section 4.2), but CAMs are area expensive (Section 4.4). These characteristics favour microarchitecture styles that minimize the use of CAM-like structures. Reorder buffers, register renaming logic, and register files are commonly implemented without CAMs. There are CAM-free instruction scheduler techniques that are not widely implemented [6, 62], but may become more favourable in soft processors.

If a traditional CAM-based scheduler is used in a soft processor, its capacity would tend to be smaller than on hard processors due to area, but the delay ratio of CAMs (15×) is not particularly poor. Reducing scheduler area can be done by reducing the number of entries or the amount of storage required per entry. Schedulers can be data-capturing where operand values are captured and stored in the scheduler, or non-data-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. Non-data-capturing schedulers reduce the amount of data that must be stored in each entry of a scheduler.

On FPGAs, block RAMs come in a limited selection of sizes, with the smallest commonly being 4.5 kbit to 18 kbit. Reorder buffers and register files are usually even smaller but are limited by port width or port count so processors on FPGAs can have larger capacity ROBs, register files, and other port-limited RAM structures at 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 [63].

6. CONCLUSIONS

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 find FPGA implementations of processor cores have 18-26× greater delay and 17-27× greater area usage than the same processors implemented using custom CMOS. We find that the FPGA vs. custom delay ratios of most processor building block circuits fall within the delay ratio range for complete processor cores, but that area ratios vary more. Building block circuits such as adders and SRAMs that have dedicated hardware support on FPGAs are particularly

area-efficient, while multiplexers and CAMs are particularly area-inefficient. The measurements' precision is limited by the availability of custom CMOS design examples in the literature. The measurements can also change as both CMOS technology and FPGA architectures evolve and may lead to different design choices.

We have discussed how our measurements impact microarchitecture design choices: Differences in the FPGA substrate encourage soft processors to have larger, low-associativity caches, deeper pipelines, and fewer bypass networks than similar hard processors. Also, out-of-order execution is a valid design option for soft processors, although scheduling windows should be kept small.

7. REFERENCES

[1] Altera. Nios II processor. http://www.altera.com/products/ip/processors/nios2/.

[2] Xilinx. MicroBlaze soft processor. http://www.xilinx.com/tools/microblaze.htm.

[3] ARM. Cortex-M1 processor. http://www.arm.com/products/processors/cortex-m/cortex-m1.php.

[4] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 26(2), Feb. 2007.

[5] D. Chinnery and K. Keutzer. *Closing the Gap Between ASIC & Custom, Tools and Techniques for High-Performance ASIC Design*. Kluwer Academic Publishers, 2002.

[6] S. Palacharla et al. Complexity-Effective Superscalar Processors. *SIGARCH Comp. Arch. News*, 25(2), 1997.

[7] V. Agarwal et al. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. *SIGARCH Comp. Arch. News*, 28(2), 2000.

[8] P. Metzgen and D. Nancekievill. Multiplexer Restructuring for FPGA Implementation Cost Reduction. In *Proc. DAC*, 2005.

[9] P. Metzgen. A High Performance 32-bit ALU for Programmable Logic. In *Proc. FPGA*, 2004.

[10] P. H. Wang et al. Intel Atom Processor Core Made FPGA-Synthesizable. In *Proc. FPGA*, 2009.

[11] S.-L. Lu et al. An FPGA-based Pentium in a Complete Desktop System. In *Proc. FPGA*, 2007.

[12] S. Tyagi et al. An Advanced Low Power, High Performance, Strained Channel 65nm Technology. In *Proc. IEDM*, 2005.

[13] K. Mistry et al. A 45nm logic technology with high-k+metal gate transistors, strained silicon, 9 cu interconnect layers, 193nm dry patterning, and 100% pb-free packaging. In *Proc. IEDM*, 2007.

[14] P. Bai et al. A 65nm Logic Technology Featuring 35nm Gate Lengths, Enhanced Channel Strain, 8 Cu Interconnect Layers, Low-k ILD and 0.57 μm^2 SRAM Cell. In *Proc. IEDM*, 2004.

[15] S.K.H. Fung et al. 65nm CMOS High Speed, General Purpose and Low Power Transistor Technology for High Volume Foundry Application. In *Proc. VLSI*, 2004.

[16] A. S. Leon et al. A Power-Efficient High-Throughput 32-Thread SPARC Processor. *IEEE Journal of Solid-State Circuits*, 42(1), 2007.

[17] U.G. Nawathe et al. Implementation of an 8-Core, 64-Thread, Power-Efficient SPARC Server on a Chip. *IEEE Journal of Solid-State Circuits*, 43(1), 2008.

[18] 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 Journal of Solid-State Circuits*, 44(1), 2009.

[19] G. Schelle et al. Intel Nehalem Processor Core Made FPGA Synthesizable. In *Proc. FPGA*, 2010.

[20] R. Kumar and G. Hinton. A Family of 45nm IA Processors. In *Proc. ISSCC*, 2009.

[21] Sun Microsystems. OpenSPARC. http://www.opensparc.net/.

[22] J. Davis et al. A 5.6GHz 64kB Dual-Read Data Cache for the POWER6 Processor. In *Proc. ISSCC*, 2006.

[23] M. Khellah et al. A 4.2GHz 0.3mm^2 256kb Dual-V_{cc} SRAM Building Block in 65nm CMOS. In *Proc. ISSCC*, 2006.

[24] P. Bai. Foils from "A 65nm Logic Technology Featuring 35nm Gate Lengths, Enhanced Channel Strain, 8 Cu Interconnect Layers, Low-k ILD and 0.57 0.57 μm^2 SRAM Cell". IEDM, 2004.

[25] L. Chang et al. A 5.3GHz 8T-SRAM with Operation Down to 0.41V in 65nm CMOS. In *Proc. VLSI*, 2007.

[26] S. Hsu et al. An 8.8GHz 198mW 16x64b 1R/1W Variation-Tolerant Register File in 65nm CMOS. In *Proc. ISSCC*, 2006.

[27] S. Thoziyoor et al. CACTI 5.1. Technical report, HP Laboratories, Palo Alto, 2008.

[28] C. E. LaForest and J. G. Steffan. Efficient Multi-Ported Memories for FPGAs. In *Proc. FPGA*, 2010.

[29] K. Pagiamtzis and A. Sheikholeslami. Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey. *IEEE Journal of Solid-State Circuits*, 2006.

[30] K. McLaughlin et al. Exploring CAM Design For Network Processing Using FPGA Technology. In *Proc. AICT-ICIW*, 2006.

[31] J.-L. Brelet and L. Gopalakrishnan. Using Virtex-II Block RAM for High Performance Read/Write CAMs. http://www.xilinx.com/support/documentation/application_notes/xapp260.pdf, 2002.

[32] I. Arsovski and R. Wistort. Self-Referenced Sense Amplifier for Across-Chip-Variation Immune Sensing in High-Performance Content-Addressable Memories. In *Proc. CICC*, 2006.

[33] D. W. Plass and Y. H. Chan. IBM POWER6 SRAM Arrays. *IBM Journal of Research and Development*, 51(6), 2007.

[34] W. Hu et al. Godson-3: A Scalable Multicore RISC Processor with x86 Emulation. *IEEE Micro*, 29(2), 2009.

[35] A. Agarwal et al. A Dual-Supply 4GHz 13fJ/bit/search 64×128b CAM in 65nm CMOS. In *Proc. ESSCIRC 32*, 2006.

[36] S.K. Hsu et al. A 110 GOPS/W 16-bit Multiplier and Reconfigurable PLA Loop in 90-nm CMOS. *IEEE Journal of Solid-State Circuits*, 41(1), 2006.

[37] W. Belluomini et al. An 8GHz Floating-Point Multiply. In *Proc. ISSCC*, 2005.

[38] J.B. Kuang et al. The Design and Implementation of Double-Precision Multiplier in a First-Generation CELL Processor. In *Proc. ICIDT*, 2005.

[39] P. Jamieson and J. Rose. Mapping Multiplexers onto Hard Multipliers in FPGAs. In *IEEE-NEWCAS*, 2005.

[40] A. Agah et al. Tertiary-Tree 12-GHz 32-bit Adder in 65nm Technology. In *Proc. ISCAS*, 2007.

[41] S. Kao et al. A 240ps 64b Carry-Lookahead Adder in 90nm CMOS. In *Proc. ISSCC*, 2006.

[42] S. B. Wijeratne et al. A 9-GHz 65-nm Intel Pentium 4 Processor Integer Execution Unit. *IEEE Journal of Solid-State Circuits*, 42(1), 2007.

[43] X. Y. Zhang et al. A 270ps 20mW 108-bit End-around Carry Adder for Multiply-Add Fused Floating Point Unit. *Signal Processing Systems*, 58(2), 2010.

[44] K. Vitoroulis and A.J. Al-Khalili. Performance of Parallel Prefix Adders Implemented with FPGA Technology. In *Proc. NEWCAS Workshop*, 2007.

[45] M. Alioto and G. Palumbo. Interconnect-Aware Design of Fast Large Fan-In CMOS Multiplexers. *IEEE Trans. Circuits and Systems II*, 2007.

[46] Altera. *Stratix III Device Handbook Volume 1*. 2009.

[47] D. Lewis et al. The Stratix II Logic and Routing Architecture. In *Proc. FPGA*, 2005.

[48] E. Sprangle and D. Carmean. Increasing Processor Performance by Implementing Deeper Pipelines. *SIGARCH Comp. Arch. News*, 30(2), 2002.

[49] A. Hartstein and T. R. Puzak. The Optimum Pipeline Depth for a Microprocessor. *SIGARCH Comp. Arch. News*, 30(2), 2002.

[50] M. S. Hrishikesh et al. The Optimal Logic Depth per Pipeline Stage is 6 to 8 FO4 Inverter Delays. In *Proc. ISCA 29*, 2002.

[51] H.B. Bakoglu and J.D. Meindl. Optimal Interconnection Circuits for VLSI. *IEEE Trans. Electron Devices*, 32(5), 1985.

[52] International Technology Roadmap for Semiconductors. http://www.itrs.net/Links/2007ITRS/Home2007.htm, 2007.

[53] Altera. *External Memory Interface Handbook, Volume 3*. 2010.

[54] Peter Yiannacouras et al. The Microarchitecture of FPGA-based Soft Processors. In *Proc. CASES*, 2005.

[55] N. Malik et al. Interlock Collapsing ALU for Increased Instruction-Level Parallelism. *SIGMICRO Newsl.*, 23(1-2), 1992.

[56] J. Phillips and S. Vassiliadis. High-Performance 3-1 Interlock Collapsing ALU's. *IEEE Trans. Computers*, 43(3), 1994.

[57] P. G. Sassone and D. S. Wills. Dynamic Strands: Collapsing Speculative Dependence Chains for Reducing Pipeline Communication. In *Proc. MICRO 37*, 2004.

[58] A. W. Bracy. *Mini-Graph Processing*. PhD thesis, University of Pennsylvania, 2008.

[59] M. Zhang and K. Asanovic. Highly-Associative Caches for Low-Power Processors. In *Kool Chips Workshop, Micro-33*, 2000.

[60] J.E. Smith and A.R. Pleszkun. Implementing Precise Interrupts in Pipelined Processors. *IEEE Trans. Computers*, 37(5), 1988.

[61] G.S. Sohi. Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers. *IEEE Trans. Computers*, 39:349–359, 1990.

[62] F. J. Mesa-Martínez et al. SEED: Scalable, Efficient Enforcement of Dependences. In *Proc. PACT*, 2006.

[63] M. P. et al. A Decoupled KILO-Instruction Processor. *Proc. HPCA*, 2006.