Exploration and Customization of FPGA-Based Soft Processors

Peter Yiannacouras, *Member, IEEE* J. Gregory Steffan, *Member, IEEE* and Jonathan Rose, *Member, IEEE*.

***Abstract*— As embedded systems designers increasingly use FPGAs while pursuing single-chip designs, they are motivated to have their designs also include *soft processors*, processors built using FPGA programmable logic. In this work we provide (i) an exploration of the microarchitectural trade-offs for soft processors, and (ii) a set of customization techniques that capitalize on these trade-offs to improve the efficiency of soft processors for specific applications. Using our infrastructure for automatically generating soft processor implementations (which span a large area/speed design space while remaining competitive with Altera's Nios II variations), we quantify tradeoffs within soft processor microarchitecture and explore the impact of tuning the microarchitecture to the application. In addition, we apply a technique of subsetting the instruction set to use only the portion utilized by the application. Through these two techniques we can improve the performance-per-area of a soft processor for a specific application by an average of 25%.**

***Index Terms*— FPGA-based soft-core processors, processor generator, design space exploration, customization.**

I. Introduction

FPGA vendors now support processors on their FPGA devices to allow complete systems to be implemented on a single programmable chip. Although some vendors have incorporated fixed *hard* processors on their FPGA die, there has been significant adoption of *soft* processors [1], [2] which are constructed using the FPGA's programmable logic itself. When a soft processor can meet the constraints of a portion of a design, the designer has the advantage of describing that portion of the application using a high-level programming language such as C/C++. More than 16% of all FPGA designs [3] contain soft processors, even though a soft processor cannot match the performance, area, and power of a hard processor [4]. FPGA platforms differ vastly from transistor-level platforms—hence previous research in microprocessor architecture is not necessarily applicable to soft processors implemented on FPGA fabrics, and we are therefore motivated to revisit processor architecture in an FPGA context.

Soft processors are compelling because of the flexibility of the underlying reconfigurable hardware in which they are implemented. This flexibility leads to two important areas of investigation for soft processor architecture that we address in this article. First, we want to understand the architectural trade-offs that exist in FPGA-based processors, by exploring a broad range of high-level microarchitectural features such as pipeline depth and functional unit implementation. Our long-term goal is to move towards a CAD system which can decide the best soft processor architecture for given area, power, or speed constraints. Second, we capitalize on the flexibility of the underlying FPGA by customizing the soft processor architecture to match the specific needs of a given application: (i) we improve the efficiency of a soft processor by eliminating support for instructions that are unused by an application; (ii) we also demonstrate how microarchitectural trade-offs can vary across applications, and how these also can be exploited to improve efficiency with application-specific soft processor designs. Note that these optimizations are orthogonal to implementing custom instructions that target custom functional units/coprocessors, which is beyond the scope of this work.

To facilitate the exploration and customization of soft processor architecture, we have developed the Soft Processor Rapid Exploration Environment (SPREE) to serve as the core of our software infrastructure. SPREE's ability to generate synthesizable Register-Transfer Level (RTL) implementations from higher-level architecture descriptions allows us to rapidly explore the interactions between architecture and both hardware platform and application, as presented in two previous publications [5], [6]. This article unifies our previous results and includes the exploration of a more broad architectural space.

A. Related Work

Commercial customizable processors are available from Tensilica [7] for ASICs, Stretch [8] as an off-the-shelf part, and others which allow designers to tune the processor with additional hardware instructions to better match their application requirements. Altera Nios [1] and Xilinx Microblaze [2] are processors meant for FPGA designs which also allow customized instructions or hardware, and are typically available in only a few microarchitectural variants.

Research in adding custom hardware to accelerate a processor has shown large potential. The GARP project [9] can provide 2-24x speedup for some microkernels using a custom coprocessor. More recent work [10] in generating custom functional units while considering communication latencies between the processor and custom hardware can achieve 41% speedup and similar energy savings. Dynamic custom hardware generation [11] delivers 5.8x speedups and 57% less energy. However these approaches which transform critical code segments into custom hardware can also benefit from the customizations in our own work: (i) additional real estate

Funding for this work was provided by NSERC and Altera Corp.

P. Yiannacouras, J. G. Steffan, and J. Rose are with the Edward S. Sr Rogers Department of Electrical and Computer Engineering—University of Toronto, Toronto, Canada (email: {yiannac,steffan,jayar}@eecg.utoronto.ca).

can be afforded for custom hardware by careful architecting of the processor and appropriate subsetting of the required instruction-set using SPREE; (ii) once the custom hardware is implemented, the criticality is more evenly distributed over the application complicating the identification of worthwhile custom hardware blocks. As a result, re-architecting is required for the new instruction stream if more efficiency is sought, a task SPREE is designed to facilitate.

The CUSTARD [12] customizable threaded soft processor is an FPGA implementation of a parameterizable core supporting the following options: different number of hardware threads and types, custom instructions, branch delay slot, load delay slot, forwarding, and register file size. While the available architectural axes seem interesting the results show large overheads in the processor design: clock speed varied only between 20 and 30 MHz on the 0.15 micron XC2V2000, and the single-threaded base processor consumed 1800 slices while the commercial Microblaze typically consumes less than 1000 slices on the same device.

PEAS-III [13], EXPRESSION [14], LISA [15], MADL [16], RDLC [17] and many others use Architecture Description Languages (ADLs) to allow designers to build a specific processor or explore a design space. However we notice three main drawbacks in these approaches for our purposes: (i) the verbose processor descriptions severely limit the speed of design space exploration; (ii) the generated RTL, if any, is often of poor quality and does not employ necessary special constructs which encourage efficient FPGA synthesis; and (iii) most of these systems are not readily available or have had only subcomponents released. To the best of our knowledge, no real exploration was done beyond the component level [13], [14] and certainly not done in a soft processor context.

Jan Gray has studied the optimization of CPU cores for FPGAs [18]. In those processors synthesis and technology mapping tricks are applied to all aspects of the design of a processor from the instruction set to the architecture. While that work documents constructs used for efficient synthesis, our work is somewhat orthogonal: we are not focussed on customizing the processor to its hardware platform and discovering "free" FPGA optimizations, but rather we assume (and have created) a variety of "good" FPGA-optimized hardware configurations and explore their different benefits to different applications. Once the best configuration is known, one can apply synthesis tricks at the component level ontop of our application-level customizations to further improve the efficiency of SPREE generated processors.

II. Generating Soft Processors with SPREE

The evaluations presented in this article use the *Soft Processor Rapid Exploration Environment* (SPREE) [5], a system we developed to allow the fast and easy generation of a large number of soft processor designs. In particular, SPREE takes as input a high-level, text-based description of the target ISA and datapath, and generates an RTL description of a working soft processor.

Figure 1 shows an overview of the SPREE system. Taking a high-level description of an architecture as input, SPREE

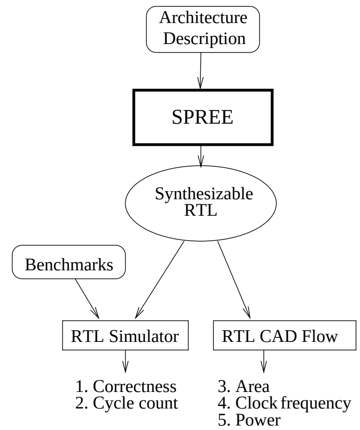


Fig. 1. Overview of the SPREE system.

automatically generates synthesizable RTL (in Verilog). We then simulate the RTL description on benchmark applications to both ensure correctness of the processor, and to measure the total number of cycles required to execute each application. The RTL is also processed by CAD tools which accurately measure the area, clock frequency, and power of the generated soft processor. The following discussion describes how SPREE generates a soft processor in more detail—complete descriptions of SPREE are available in previous publications [5], [19], [20].

A. Input: The Architecture Description

The input to SPREE is a description of the desired processor, composed of textual descriptions of both the target ISA and the processor datapath. Each instruction in the ISA is described as a directed graph of generic operations (GENOPs), such as `ADD`, `XOR`, `PCWRITE`, `LOADBYTE`, and `REGREAD`. The graph indicates the flow of data from one GENOP to another required by that instruction. SPREE provides a library of basic components (e.g., a register file, adder, sign-extender, instruction fetch unit, forwarding line, and more). A processor datapath is described by the user as an interconnection of these basic components. As we describe below, SPREE ensures that the described datapath is capable of implementing the target ISA.

The decision to use a structural architectural description in SPREE reflects our goal of efficient implementation. Structural descriptions provide users with the ability to better manage the placement of all components including registers, and multiplexers in the datapath. This management is crucial for balancing the logic delay between registers to achieve fast clock speeds. By analyzing the critical path reported by the CAD tool, users can identify the components which limit the clock frequency and take one of three actions: (i) reducing the internal logic delay of a component, for example, making a unit complete in two cycles instead of one; (ii) moving some of the logic (such as multiplexers and sign-extenders) from the high delay path into neighboring pipeline stages to reduce the amount of logic in the high delay path; (iii) adding non-pipelined registers in the high delay path causing a pipeline stall. The latter two of these actions depend critically on this ability to manually arrange the pipeline stages, referred to as *retiming*, which is difficult for modern synthesis tools because of the complexity in the logic for controlling the pipeline registers. Without a good ability to optimize delay we risk making incorrect conclusions based on poor implementations. For example, one might conclude that the addition of a component does not impact clock frequency because the impact is hidden by the overhead in a poorly designed pipeline. For this reason, architectural exploration in academia has traditionally neglected clock frequency considerations.

B. Generating a Soft Processor

From the above inputs, SPREE generates a complete Verilog RTL model of the desired processor in three phases: (i) datapath verification, (ii) datapath instantiation, and (iii) control generation. In the datapath verification phase, SPREE compares the submitted ISA description and datapath description, ensuring that the datapath is functionally capable of executing all of the instructions in the ISA description. The datapath instantiation phase automatically generates multiplexers for sinks with multiple sources and eliminates any components that are not required by the ISA. Finally, the control generation phase implements the control logic necessary to correctly operate the datapath, and emits the Verilog descriptions of the complete processor design. Control generation is where the SPREE environment adds the most value since it automatically handles multi-cycle and variable cycle functional units, the select signals for multiplexers, the operation codes for each functional unit, the interlocking between pipeline stages, and the complexities in branching including the proper handling of branch delay slots.

C. Limitations

There are several limitations to the scope of soft processor microarchitectures that we study in this article. For now we consider simple, in-order issue processors that use only on-chip memory and hence have no cache—since the relative speeds of memory and logic on a typical FPGA are much closer than for a hard processor chip, we are less motivated to explore an on-chip memory hierarchy for soft processors. The largest FPGA devices have more than one megabyte of on chip memory which is adequate for the applications that we study in this article—however, in the future we do plan to broaden our application base to those requiring off-chip RAM, which would motivate caches. We do not yet include support for dynamic branch prediction, exceptions, or operating systems. Finally, in this article we do not add new instructions to the ISA, we restrict ourselves to a subset of MIPS-I and have not modified the compiler, with the exception of evaluating software versus hardware support for multiplication due to the large impact of this aspect on cycle time and area.

The MIPS instruction-set was chosen tentatively due to the abundance of available software tools (compilers, instruction simulators, etc) available for it and because of its striking similarities with the Nios II ISA. The appropriateness of a C sequential programming model and sequential Harvard architecture model for soft processors is not evaluated in this work. In addition, we have not evaluated other ISAs, but we expect similar RISC architectures to follow the same trends.

III. Experimental Framework

Having described the SPREE system, we now describe our framework for measuring and comparing the soft processors that it produces. We present methods for verifying the processors, employing FPGA CAD tools, measuring and comparing soft processors, and we discuss the benchmark applications that we use to do so.

A. Processor Verification

SPREE verifies that the datapath is capable of executing the target ISA—however, we must also verify that the generated control logic and the complete system function correctly. We implement trace-based verification by using a cycle-accurate industrial RTL simulator (Modelsim) that generates a trace of all writes to the register file and memory as it executes an application. We compare this trace to one generated by MINT [21] (a MIPS instruction set simulator) and ensure that the traces match. SPREE automatically generates test benches for creating traces and also creates debug signals to ease the debugging of pipelined processors.

B. FPGAs, CAD, and Soft Processors

While SPREE itself emits Verilog which is synthesizable to any target FPGA architecture, we have selected Altera's Stratix [22] device for performing our FPGA-based exploration. The library of processor components thus targets Stratix I FPGAs. We use Quartus II v4.2 CAD software for synthesis, technology mapping, placement and routing. We synthesize all designs to a Stratix EP1S40F780C5 device (a middle-sized device in the family, with the fastest speed grade) and extract and compare area, clock frequency, and power measurements as reported by Quartus.

We have taken the following measures to counteract variation caused by the non-determinism of CAD tool output: (i) we have coded our designs structurally to avoid the creation of inefficient logic from behavioral synthesis; (ii) we have experimented with optimization settings and ensured that our conclusions do not depend on them, and (iii) for the area and clock frequency of each soft processor design we determine the arithmetic mean across 10 seeds (different initial placements before placement and routing) so that we are 95% confident that our final reported value is within 2% of the true mean.

The difference between ASIC and FPGA platforms is large enough that we are motivated to revisit the microarchitectural design space in an FPGA context. However, FPGA devices differ among themselves: across device families and vendors the resources and routing architecture on each FPGA vary greatly. We have focused on a single FPGA device, the Altera Stratix, to enable efficient synthesis through device-specific optimizations. Our hypothesis, is that in spite of differences in FPGA architecture, the conclusions drawn about soft processor architecture will be transferable between many FPGA families.

In the future, we plan to investigate this across a range of different FPGA families. For now, we have migrated from Stratix I to Stratix II and observed that there is some noise in the results, but most of the conclusions still hold.

C. Metrics for Measuring Soft Processors

To measure area, performance, and efficiency, we must decide on an appropriate set of specific metrics. For an FPGA, one typically measures area by counting the number of resources used. In Stratix, the main resource is the *Logic Element* (LE), where each LE is composed of a 4-input *lookup table* (LUT) and a flip flop. Other resources, such as the hardware multiplier block, and memory blocks can be converted into an equivalent number of LEs based on the relative areas of each in silicon.[1] Hence we report actual silicon area of the design including routing in terms of *equivalent LEs*.

To measure performance, we account for both the clock frequency and instructions-per-cycle (IPC) behavior of the architecture by measuring either wall-clock-time or instruction throughput per second in MIPS (millions of instructions per second). Reporting either clock frequency or IPC alone can be misleading and in this work we have the unique ability to capture both accurately. To be precise, we multiply the clock period (determined by the Quartus timing analyzer after routing) with the number of cycles spent executing the benchmark to attain the wall-clock-time execution for each benchmark. Dividing the total number of instructions by the wall-clock-time gives instruction throughput.

We measure the efficiency by computing the performance gained in instruction throughput per unit area in LEs, thus it is measured in units of MIPS/LE. This metric is analogous to the inverse of area-delay product, an often used but debatable metric for simultaneously capturing area and speed. The best processor for a specific application depends on that application's weighting of area and speed (as an extreme, some application's might care only about one and not the other), this metric assumes a balanced emphasis on both area and speed.

D. Benchmark Applications

We measure the performance of our soft processors using 20 embedded benchmark applications from four sources (as summarized in Table I). Some applications operate solely on integers, and others on floating point values (although for now we use only software floating point emulation); some are compute intensive, while others are control intensive. Table I also indicates any changes we have made to the application to support measurement, including reducing the size of the input data set to fit in on-chip memory (`d`), and decreasing the number of iterations executed in the main loop to reduce simulation times (`i`). Additionally, all file and other I/O were removed since we do not yet support an operating system.

[1]The relative area of these blocks was provided by Altera [23] and are proprietary.

TABLE I
BENCHMARK APPLICATIONS EVALUATED.

Source	Benchmark	Modified	Dyn. Instr. Counts
MiBench [24]	BITCNTS	di	26,175
	CRC32	d	109,414
	QSORT*	d	42,754
	SHA	d	34,394
	STRINGSEARCH	d	88,937
	FFT*	di	242,339
	DIJKSTRA*	d	214,408
	PATRICIA	di	84,028
XiRisc [25]	BUBBLE_SORT		1,824
	CRC		14,353
	DES		1,516
	FFT*		1,901
	FIR*		822
	QUANT*		2,342
	IQUANT*		1,896
	TURBO		195,914
	VLC		17,860
Freescale [26]	DHRY*	i	47,564
RATES [27]	GOL	di	129,750
	DCT*	di	269,953

* Contains multiply
d Reduced data input set
i Reduced number of iterations

IV. COMPARISON WITH NIOS II VARIATIONS

To ensure that our generated designs are indeed interesting and do not suffer from prohibitive overheads, we have selected Altera's Nios II version 1.0 family of processors for comparison. Nios II has three mostly-unparameterized variations: `Nios II/e`, a small unpipelined 6-CPI processor with serial shifter and software multiplication; `Nios II/s`, a 5-stage pipeline with multiplier-based barrel shifter, hardware multiplication, and instruction cache; and `Nios II/f`, a large 6-stage pipeline with dynamic branch prediction, and instruction and data caches.

We have taken several measures to ensure that comparison against the Nios II variations is as fair as possible. We have generated each of the Nios processors with memory systems identical to those of our designs: two 64KB blocks of RAM are used for separate instruction and data memories. We do not include cache area in our measurements, though some logic required to support the caches will inevitably count towards the Nios II areas. The Nios II instruction set is very similar to the MIPS-I ISA with some minor modifications in favor of Nios (for example, the Nios ISA has no tricky branch delay slots)—hence Nios II and our generated processors are very similar in terms of ISA. Nios II supports exceptions and OS instructions, which are so far ignored by SPREE meaning SPREE processors save on the hardware costs in implementing these. Finally, like Nios II, we also use `gcc` as our compiler, though we did not modify any machine specific parameters nor alter the instruction scheduling. Despite these differences, we believe that comparisons between Nios II and our generated processors are relatively fair, and that we can be confident that our architectural conclusions are sound.

For this experiment we generated all three Nios II variations

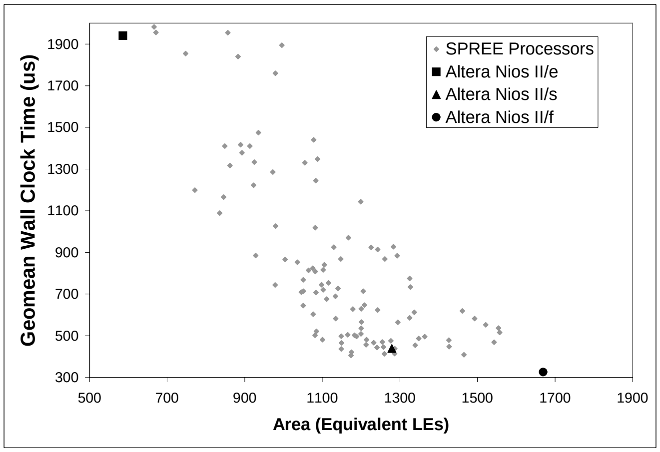


Fig. 2. Comparison of our generated designs vs the three Altera Nios II variations.

in the manner outlined above and we also generated several different SPREE processors that varied in their pipelines and functional units. All of these processors are benchmarked using the same applications from our benchmark set and synthesized to the same device. Their area and performance is measured using the measurement methodology outlined in the previous section.

Figure 2 illustrates our comparison of SPREE generated processors to the commercial Altera Nios II variations in the performance-area space. SPREE's generated processors span the design space between Nios II variations, while allowing more fine-grained microarchitectural customization. The figure also shows that SPREE processors remain competitive with the commercial Nios II. In fact, one of our generated processors is both smaller and faster than the `Nios II/s`—hence we examine that processor in greater detail.

The processor of interest is an 80MHz 3-stage pipelined processor, which is 9% smaller and 11% faster in wall-clock-time than the `Nios II/s`, suggesting that the extra area used to deepen `Nios II/s`'s pipeline succeeded in increasing the frequency, but increased overall wall-clock-time. The generated processor has full inter-stage forwarding support and hence no data hazards, and suffers no branching penalty because of the branch delay slot instruction in MIPS. The CPI of this processor is 1.36 whereas the CPIs of `Nios II/s` and `Nios II/f` are 2.36 and 1.97 respectively. However, this large gap in CPI is countered by a large gap in clock frequency: `Nios II/s` and `Nios II/f` achieve clock speeds of 120 MHz and 135 MHz respectively, while the generated processor has a clock of only 80MHz. These results demonstrate the importance of evaluating wall-clock-time over clock frequency or CPI alone, and that faster frequency is not always better.

V. Exploring Soft Processor Architecture

In this section, we employ the SPREE soft processor generation system to explore the architectural terrain of soft processors when implemented and executed on FPGA hardware. The goal here is to seek and understand tradeoffs that may be employed to tune a processor to its application. We vary a number of core architectural parameters and measure their effects on the processor. Additionally we attempt to attribute non-intuitive exploratory results to fundamental differences of an FPGA versus an ASIC: (i) Multiplexing is costly—their high number of inputs and low computational density means they generally map poorly to LUTs; (ii) Multiplication is efficient—FPGA vendors now include dedicated multiplier circuitry meaning performing multiplication can be done comparatively more efficient than in an ASIC (relative to other logic on the same fabric); (iii) Storage is cheap—with every lookup table containing a flip-flop and dedicated memory blocks scattered throughout the device, storage space is abundant in modern FPGAs; (iv) Memories are fast—the dedicated memories on the device can be clocked as fast as a simple binary counter [28] (v) More coarse-grained progression of logic levels—in an FPGA a lookup table is considered a single level of logic but in fact can encompass several levels of logic worth of ASIC gates, however a steep inter-cluster routing penalty is paid for connecting multiple LUTs.

A. Functional Units

The largest integer functional units in a soft processor are the shifter, the multiplier, and the divider. The divider is excluded from any study as it is too large (measured up to 1500 LEs compared to 1000 LEs for the rest of the processor), and it seldomly appears in the instruction streams of our benchmarks (only four benchmarks contain divides but in each case they make up less then half a percent of the instruction stream). Thus we eliminate the divider unit and support division using a software subroutine. We hence focus on only the shifter and the multiplier.

1) Shifter Implementation: The shifter unit can be implemented in one of four ways: in a shift register which requires one clock cycle for every bit shifted, in LUTs as a tree of multiplexers, in the dedicated multipliers as a separate functional unit, or in the dedicated multipliers as a shared multiplier/shifter unit as used by Metzgen [29]. We implement each of these in four different pipelines and contrast the different processors with respect to their area, performance, and energy on our set of benchmarks.

With respect to area, the processors with shared multiplier-based shifter are 186 equivalent LEs smaller than the LUT-based and 147 equivalent LEs smaller than the unshared multiplier based shifter. The performance of the three were very similar (save for minor variations in clock frequency). This leads us to conclude that because of the large area savings and matched performance, this implementation is generally favorable over both the LUT-based shifter and the unshared multiplier-based shifter.

Figure 3 shows the performance of all benchmarks on a 3-stage pipeline with either the serial shifter or multiplier-based shifter. The processor with serial shifting is smaller by 64 LEs, but it also pays a heavy performance penalty when the shifter is used frequently. For example, the CRC, TURBO, and VLC benchmarks are slowed by 3-4x. Hence this application-specific trade-off is worthy of exploring as a potential customization.

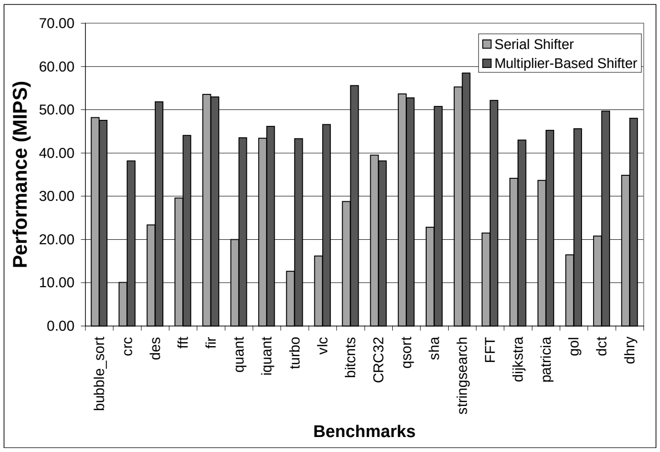


Fig. 3. Performance of a 3-stage pipelined processor with two different shifter implementations.

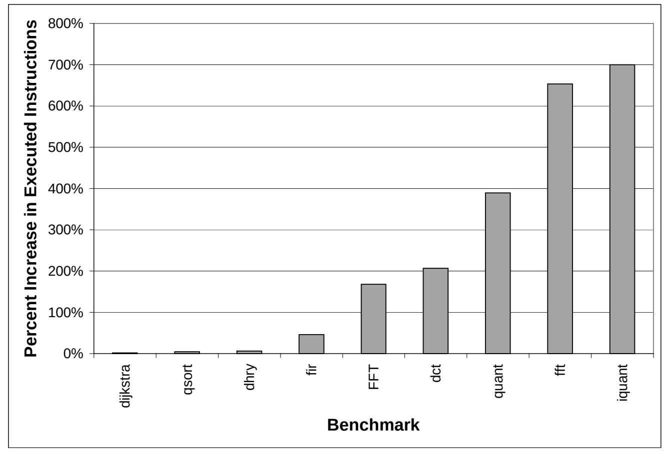


Fig. 4. The increase in total executed instructions when using a software multiplication subroutine instead of a single-instruction multipliy in hardware.

2) Multiplication Support: Whether multiplication is supported in hardware or software can greatly affect the area, performance, and power of a soft processor. There may be many variations of multiplication support which trade area for cycle time; we consider only full multiplication support using the dedicated multipliers in the FPGA. We make this simplification because we found that the hard multipliers on FPGAs are so area efficient that alternative implementations made of LUTs and flip-flops (whether Booth, array, etc.) are consistently less efficient.

The area of the hardware multiplier is generally 230 equivalent LEs; however only 160 of those are attributed to the actual multiplier. The remaining 70 LEs compose the glue logic required to hook the functional unit into the datapath including the MIPS `HI/LO` registers—the creators of the MIPS ISA separated the multiplier from the normal datapath by having it write its result to dedicated registers (`HI` and `LO`) instead of to the register file. This decision was later criticized [30], and we also find it to be a problem for FPGA designs: since multiplication can be performed quickly in FPGAs (only one or sometimes two cycles longer than an adder), it doesn't require special hardware to help overcome its cycle latency. Rather, the special registers and multiplexing prove to be wasted hardware, especially in the case of a shared multiplier/shifter since the shift result must be written to the register file anyway (hence the path from the multiply unit to the register file exists in spite of the attempts by the ISA to prevent it). We therefore agree with the approach taken in the Nios II ISA where separate multiply instructions compute the upper and lower words of the product.

Figure 4 indicates that the performance of a processor that supports multiplication in hardware can vastly exceed one with only software multiplication support. Half of our twenty benchmarks do not contain multiplication, but for the other half the results vary from suffering 8x more instructions executed as for IQUANT to an insignificant 1.8% increase for DIJKSTRA. Depending on the frequency of multiply instructions in the application, if any exist at all, a designer may look more favorably on the reduced area of a software implementation. We therefore deem multiplication support to

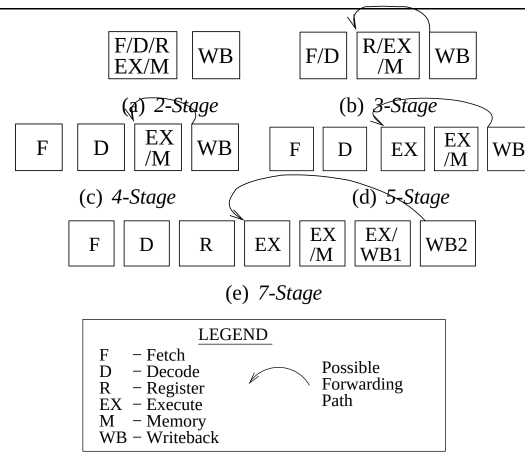


Fig. 5. Processor pipeline organizations studied. Arrows indicate possible forwarding lines.

be an important potential customization axis.

Our study of functional units has identified and quantified the hardware trade-offs in implementing different shifter and multiplication support, which will both later be used to tune a processor to its application. In addition, we have pointed out the inappropriateness of MIPS to force separation of multiplies from the normal datapath through the use of the `HI/LO` registers: for FPGA-based designs where the multiplier is not dramatically slower than any other functional unit this "special case" handling of multiplication is unnecessary.

B. Pipelining

We now use SPREE to study the impact of pipelining in soft processor architectures by generating processors with pipeline depths between two and seven stages, the organizations of which are shown in Figure 5. A 1-stage pipeline (or purely unpipelined processor) is not considered since it provides no benefit over the 2-stage pipeline: the writeback stage can be pipelined with the rest of the execution of that instruction for free, increasing the throughput of the system and increasing the size of the control logic by an insignificant amount. This

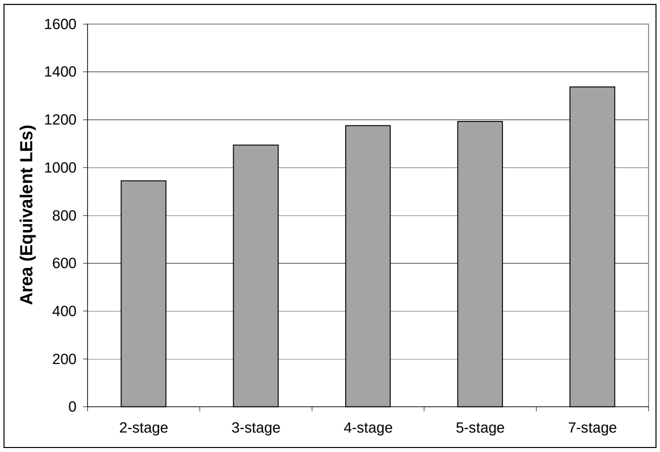


Fig. 6. Area across different pipeline depths.

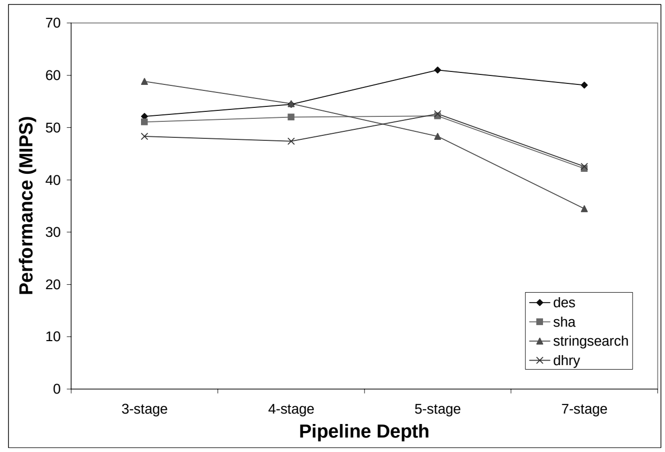


Fig. 7. Performance impact of varying pipeline depth for select benchmarks.

free pipelining arises from the fact that both the instruction memory and register file are implemented in synchronous RAMs which require registered inputs. Note that we similarly do not consider a 6-stage pipeline since the 5-stage pipeline has competing critical paths in the writeback stage and decode stage which require both stages to be split to achieve a significant clock frequency gain. For every pipeline, data hazards are prevented through interlocking, branches are statically predicted to be not-taken, and mis-speculated instructions are squashed.

1) Pipeline Depth: Figure 6 shows (as expected) that area increases with the number of pipeline stages due to the addition of pipeline registers and data hazard detection logic. However, we notice that the increase in area is mostly in combinational logic and not registers: even the 7-stage pipeline has only a dozen LEs occupied with only a register, while 601 LEs are occupied without a register. Register-packing algorithms can typically combine these, but likely did not for performance reasons. As such, there is plenty space for the design to absorb flip flops invisibly, since we expect register packing to place these in the 601 LEs occupied without a register; but inserting these registers into the design breaks up logic into smaller pieces which are less likely to be optimized into LUTs. This causes combinational logic to be mapped into more LUTs which increases area, along with the necessary data hazard detection and stalling/squashing logic which also contribute to the increased area.

Figure 7 shows the performance impact of varying pipeline depth for four applications which are representative of several trends that we observed. The performance is measured in instruction throughput which accounts for both the frequency of the processor and its cycles-per-instruction behavior. The figure does not show the 2-stage pipeline as it performs poorly compared to the rest: the synchronous RAMs in Stratix must be read from in a single stage of the pipeline for this design, hence it suffers a stall cycle to accommodate the registered inputs of the RAM. The 7-stage pipeline also has a disadvantage: branch delay slot instructions are much more difficult to support in such a deep pipeline, increasing the complexity of the control logic for this design. In contrast, the trends for the 3, 4, and 5-stage pipelines vary widely by application. DES experiences up to 17% improved performance as the pipeline depth increases from 3 to 5 stages, while for STRINGSEARCH performance degrades by 18%. SHA maintains consistent performance across the pipelines, which is a typical trend for many applications. For DHRY, performance decreases by only 2% and then increases by 11%. Pipeline depth is therefore another application-specific trade-off, due to the fact that some applications suffer more than others from branch penalties, and data hazards of varying distances.

For most individual benchmarks, and when considering the average across all benchmarks, the pipelines perform the same for the three, four, and five stage pipelines, while the 7-stage pipeline performs slightly worse for the reasons mentioned above. As such we are inclined to conclude that the 3-stage pipeline is most efficient since it performs equally well while using less area. We suspect that this is caused partly by the coarse-grained positioning of flip flops and the large logic capacity of a LUT which is under-utilized when there is little logic between registers. However there is another factor to consider: there are many architectural features which SPREE does not currently support that could be added to favor the deeper pipelines, for example better branch prediction and more aggressive forwarding. Nonetheless, this sentiment that shorter pipelines are better is echoed by Xilinx's Microblaze [2] which also has only 3-stages, and also Tensilica's Diamond 570T [7] processor which has only 5-stages (but is designed for an ASIC process).

2) Pipeline Organization: Trade-offs exist not only in the number of pipeline stages, but also in the placement of these stages. While deciding the stage boundaries for our 3-stage pipeline was obvious and intuitive, deciding how to add a fourth pipeline stage was not. One can add a decode stage as shown in Figure 5(c), or further divide the execution stage. We implemented both pipelines for all three shifters and observed that although the pipeline in Figure 5(c) is larger by 5%, its performance is 16% better. Hence there is an area-performance trade-off, proving that such trade-offs exist not only in pipeline depth, but also in pipeline organization.

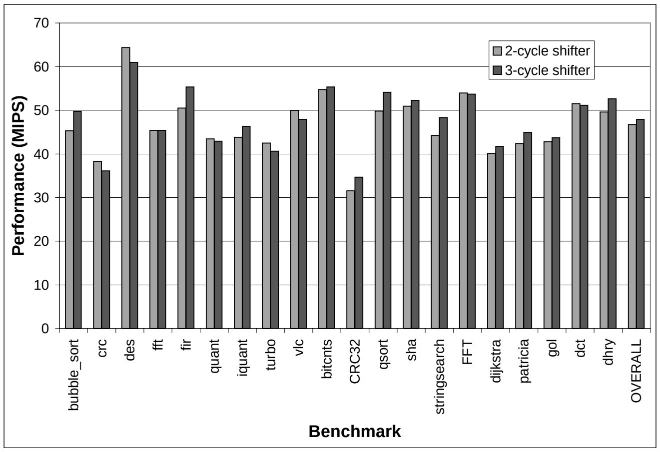


Fig. 8. The performance trade-off in implementing unpipelined multi-cycle paths on a processor across the benchmark set.

3) Forwarding: We also examined the effect of implementing the forwarding paths shown in figure 5 either for both MIPS source operands, one of them, or none at all. Although not shown, we found that the variance in trends between applications for different forwarding paths is insignificant. We found that forwarding can provide area/performance trade-offs in general, but none that differ significantly on a per-application basis. Typically forwarding is a "win" for all applications sometimes providing 20% faster processors at the expense of approximately 100 LEs. While this result matches what is expected in ASIC-implemented processors, we project that for deeper pipelines that more aggressive forwarding would result in more costly multiplexing, leading to unique FPGA-specific trade-offs that differ from those of ASIC processors.

C. Unpipelined Multi-Cycle Paths

Adding pipeline registers increases frequency but can also increase total CPI, as data hazards and branch penalties result in additional pipeline stalls. Alternatively, registers can be used in a more direct way for trading clock frequency and CPI: registers can be inserted *within* a bottleneck pipeline stage that occasionally prevents that stage from completing in a single cycle, but that also allows the stage (and hence the entire pipeline) to run at a higher clock frequency. Moreover, with flip flops readily available in every LE and embedded in the block RAMs and multipliers, this register insertion can come with only small area increases.

As a concrete example, we consider the 5-stage pipeline with 2-cycle multiplier-based barrel shifter. This processor has a critical path through the shifter which limits the clock speed to 82.0 MHz while achieving 1.80 average CPI across the benchmark set. We can create another unpipelined multi-cycle path by making the multiplier-based shifter a 3-cycle unpipelined execution unit which results in a clock frequency of 90.2 MHz and 1.92 average CPI. The 10% clock frequency improvement is countered by an average CPI increase of 6.7%. Figure 8 shows the instruction throughput in MIPS of both processors for each benchmark and indicates that benchmarks can favor either implementation. For example, BUBBLE_SORT achieves 10% increased performance when using the 3-cycle multiplier-based shifter while CRC achieves 6% increased performance with the 2-cycle implementation. With respect to area, the two processors differ in area by only a single LE. Hence we can use unpipelined multi-cycle paths to make application-specific trade-offs between clock frequency and CPI. Note that this technique is not limited to the execution stage, and can be applied anywhere in the processor pipeline. In the set of explored processors this technique was explored in large execution units (either the shifter or multiplier) whenever these units lay in the critical path.

VI. THE IMPACT OF CUSTOMIZING SOFT PROCESSORS

In this section we use the SPREE system to measure the impact of customizing soft processors to meet the needs of individual applications. We demonstrate the impact of three techniques: (i) tuning the microarchitecture for a given application by selecting architectural features which favor that application but do no alter the ISA; (ii) subsetting the ISA to eliminate hardware not used by the application (for example if there is no multiplication we can eliminate the multiplier functional unit); and (iii) the combination of these two techniques.

A. Application-Tuned vs General Purpose

We have demonstrated that many microarchitectural axes provide application-specific trade-offs that can be tuned in soft processors to better meet application requirements. In this section we use SPREE to implement all combinations of these architectural axes—3 shifter implementations, 5 pipeline depths, HW/SW multiplying, 4 forwarding configurations, 2-3 separately adjusted functional unit latencies, as well as miscellaneous pipeline organizations. We exhaustively search for the best processor for each application in our benchmark set. Specifically, we described each processor, generated it using SPREE, synthesized and placed-and-routed it using our CAD flow, and finally computed and compared the performance-per-area for each benchmark and processor pair. Performance-per-area is used as our metric since many of our architectural axes trade area and performance (for example, the benefit of using a serial shifter or software multiply is in reducing area at the expense of performance). We call the best processor the *application-tuned* processor, which ideally is the processor a designer (or intelligent software) would choose given the application and this set of processors. We also determine the processor that performed best on average over the complete benchmark set—this we refer to as the *general-purpose* processor. We then analyze the difference in efficiency between the general purpose processor and the application-tuned processors and hence evaluate the potential for making application-specific trade-offs in soft processor microarchitecture.

Figure 9 shows the measured efficiency in MIPS/LE in four bars: (i) the best-on-average (general-purpose) processor of those we generated using SPREE—this processor (the 3-stage pipeline with multiplier-based shifter) was found to

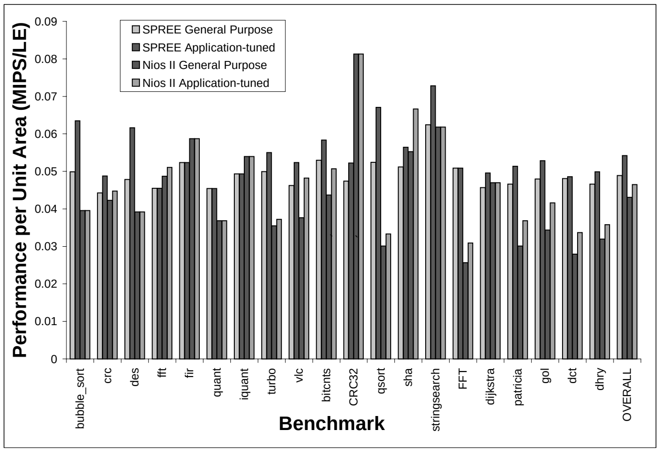


Fig. 9. Performance-per-area for each benchmark for SPREE (i) best-on-average (*general-purpose*) processor and (ii) best per benchmark (*application-tuned*) processor as well as for Nios II (iii) general purpose and (iv) application tuned.

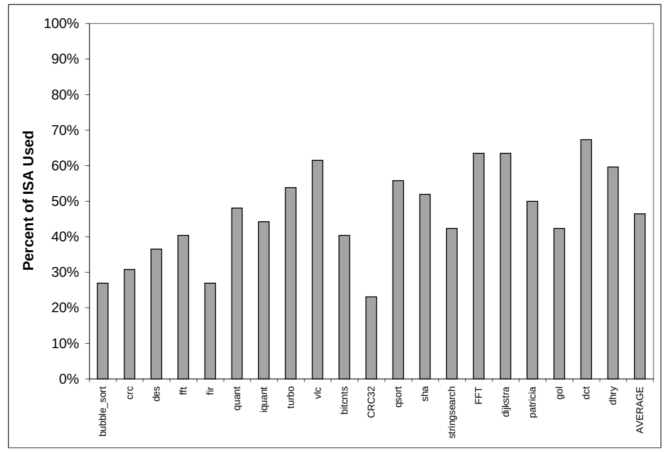


Fig. 10. Instruction set utilization across benchmark set.

provide the best geometric mean performance-per-area across the entire benchmark set; (ii) the best per benchmark SPREE processor; (iii) the best-on-average of the Nios II variations—experiment showed it was the `Nios II/s`; and (iv) the best per benchmark of the Nios II variations from either `Nios II/s`, `Nios II/e`, or `Nios II/f`.

Focussing only on the SPREE processors in the first two bars, we noticed only 6 of the 20 benchmarks achieve their highest performance-per-area using the best overall processor; instead, the best processor for each benchmark varies and offers significantly better efficiency. By choosing an application-tuned processor, average performance-per-area is improved by 14.1% over the best overall processor across the entire benchmark set; furthermore, STRINGSEARCH, QSORT, CRC32, and BUBBLE_SORT improve performance-per-area by approximately 30%. The results indicate that these ISA-independent modifications made in the processor core can be substantial and are certainly worth pursuing in a system-on-programmable-chip platform where general purpose efficiency is not a key design consideration and the reconfiguration of the processor is free. In future work, we expect this benefit to increase significantly when supporting more advanced architectural axes such as datapath widths, branch predictors, aggressive forwarding, caches, and VLIWs.

The best per-benchmark processors had many predictable trends. Benchmarks without significant shifting benefited from a smaller serial shifter, benchmarks with little or no multiplying utilized software multiplication instead of a multiplier functional unit. Full forwarding on both operands was always present except for three benchmarks which chose no forwarding. In terms of pipeline stages, all benchmarks used 3-stage pipelines except for three (a different triplet than those with no forwarding) which used 5-stage pipelines likely due to reduced stress on the pipeline allowing those benchmarks to enjoy the higher frequencies. In the future we hope to select the appropriate architecture from analysis of the application.

We now compare our results with the `Nios II` processor variations shown in the latter two bars. For most of the benchmarks, the application-tuned SPREE processors yields significantly better performance-per-area than any of either the general-purpose SPREE processor or the commercial Nios II variants. On average across all benchmarks, the application-tuned processors generated by SPREE are the most efficient, yielding 16% better results than Nios II because of the more fine-grained customization space afforded by SPREE—the Nios II actually spans a larger design space than SPREE with the Nios II/e being smaller and the Nios II/f being faster than any current SPREE processor. This leads us to suggest that many fine grain micro-architectural customizations can yield better efficiencies than a few separate hand-optimized cores targeting specific design space points.

B. ISA Subsetting

So far we have investigated microarchitectural customizations that favor an individual application but still fully support the original ISA. In this section, we propose to capitalize on situations where: (i) only one application will run on the soft processor; (ii) there exists a reconfigurable environment allowing the hardware to be rapidly reconfigured to support different applications or different phases of an application. We customize the soft processor by having it support only the fraction of the ISA which is actually used by the application. SPREE performs this *ISA subsetting* by parsing the application binary to decide the subsetted ISA, removing unused connections and components from the input datapath, and then generating simpler control. Figure 10 shows the fraction of the 50 MIPS-I instructions supported by SPREE that are used by each benchmark, which is rarely more than 50%. BUBBLE_SORT, FIR, and CRC32 use only about one quarter of the ISA. With such sparse use of the ISA, we are motivated to investigate the effect of eliminating the architectural support for unused instructions.

To evaluate the impact of ISA subsetting, for each of the 20 benchmarks we subsetted three processor architectures: (i) A 2-stage pipeline with LUT-based barrel shifting; (ii) The 3-stage pipeline with multiplier-based barrel shifting; (iii) a 5-stage pipeline with LUT-based barrel shifting. Note that the processors with LUT-based shifting are somewhat contrived since having the separate large LUT-shifter will emphasize

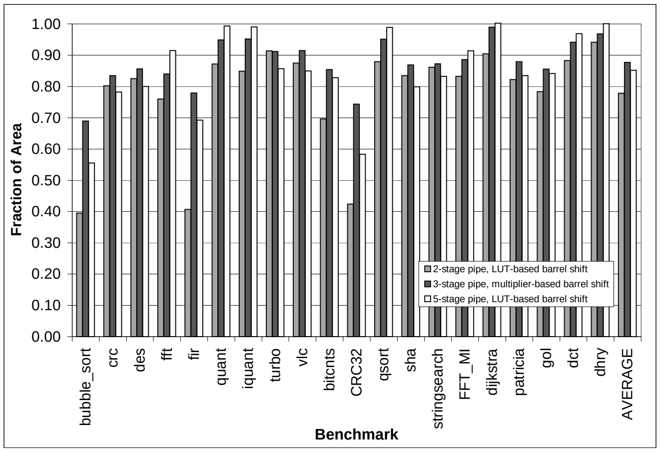


Fig. 11. The impact on area of ISA subsetting on three architectures.

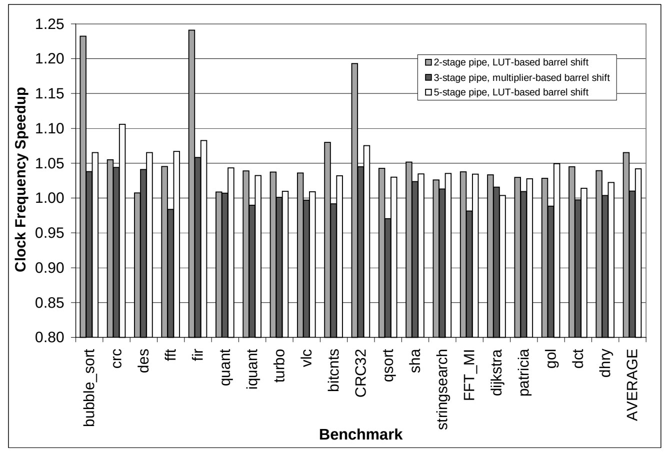


Fig. 12. The impact on clock speed of ISA subsetting on three architectures.

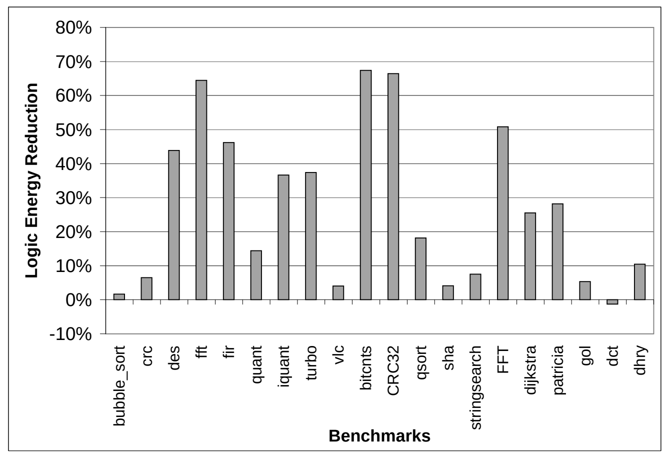


Fig. 13. Energy reduction of subsetting on 3-stage pipeline.

the effects of subsetting; however such a situation may arise in low-end devices which do not contain dedicated multipliers. All three processors utilize hardware multiplication support. Since the cycle-by-cycle execution of each benchmark is unaffected by this experiment, we use clock frequency to measure performance gain.

The relative area of each subsetted processor with respect to its non-subsetted version is shown in Figure 11. In general on our best processor, the 3-stage with multiplier-based shifting, we can expect 12.5% area reductions. The three benchmarks which use only 25% of the ISA (BUBBLE_SORT, FIR, and CRC32) obtain the most significant area savings. For the contrived 2-stage architecture, these 3 benchmarks obtain a 60% area savings, while most other benchmarks save 10-25% area. Closer inspection of these three benchmarks reveal that they are the only benchmarks which do not contain shift operations—shifters are large functional units in FPGAs, and their removal leads to a large area savings. However the MIPS ISA obstructs such a removal because there is no explicit `nop` instruction, instead `nop`s are encoded as a shift-left-by-zero. Therefore to remove the shifter, one must include special hardware to handle these `nop` instructions, or else re-encode the `nop`. In this work `nop`s are re-encoded as `add` zero (similar to Nios II) to allow for complete removal of the shifter, since all benchmarks use `add`s. The savings are more pronounced in the 2 and 5-stage pipeline where the shifter is LUT-based and hence larger.

Figure 12 shows the clock frequency improvement for the subsetted architectures. In general we see modest speedups of 7% and 4% on average for the 2 and 5-stage pipelines, respectively. The 3-stage pipeline is not improved at all, as its critical path is in the data hazard detection logic and hence cannot be removed. More positively, the modest clock frequency speedups indicate that our pipelines have well-balanced logic delays: when logic is removed from a given path there is often another path to maintain the previous critical path length, hence the odds of a given subsetting reducing all paths is relatively small. However, there is notable performance improvement in the 2-stage pipeline for the three benchmarks without shifts, BUBBLE_SORT, FIR, and CRC32. This is because the LUT-based shifter lay in the critical path of the pipeline and caused poor balancing of logic delay. Removing the shifter allows for a roughly 20% improvement in clock frequency for these benchmarks.

Figure 13 shows the reduction in energy resulting from ISA subsetting. Note that energy dissipated by the memory is ignored since it accounts for 80-90% of the energy, and it can never be removed by subsetting since all applications fetch and write to memory. The figure demonstrates that the removal of high-toggle rate components and simplified control result in significant energy savings in the processor pipeline. The subsetted processors of some benchmarks such as FFT, BITCNTS, and CRC32 provide greater than 65% energy savings. On average across all the subsetted processors, approximately 27% of the non-memory energy can be saved. A miniscule increase in energy was seen for the DCT benchmark which we attribute to noise in the CAD system, total system energy decreased very slightly but the fraction attributed to logic increased unexpectedly.

C. Combining Customization Techniques

We have presented two methods for creating application-specific soft processors: (i) architectural tuning, which alters soft processor architecture to favor a specific benchmark;

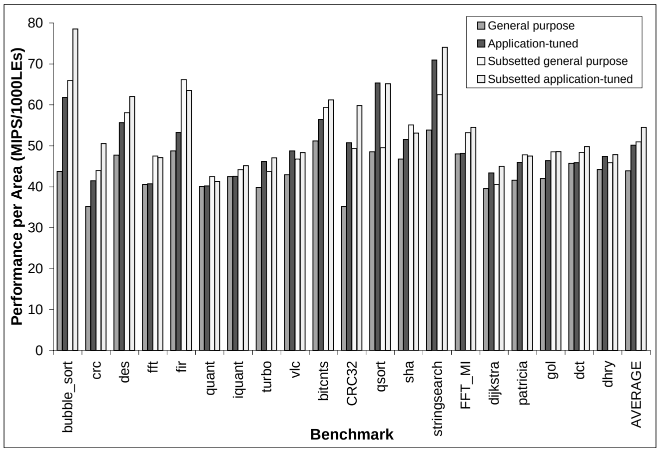


Fig. 14. Performance-per-area of tuning, subsetting, and their combination.

and (ii) ISA subsetting, which removes architectural support that is not utilized by the benchmark. In this section we compare the effectiveness of the two techniques in terms of performance-per-area both individually and combined. We define the best general-purpose processor as the single processor which achieves the greatest performance-per-area on average across all benchmarks, and the best application-tuned processors as the set of processors which achieve the best performance-per-area for each benchmark. For each processor and benchmark we then perform ISA subsetting, and measure the performance-per-area of the four combinations: general-purpose, application-tuned, subsetted general-purpose, and subsetted application-tuned.

Figure 14 shows the performance-per-area for all four combinations. As shown previously, the application-tuned processor is consistently better than the general-purpose processor. ISA subsetting is more effective on the general-purpose processor than on the application-tuned processors: the performance-per-area is improved by 16.2% on average for the general-purpose processor while by only 8.6% for the application-tuned processor. This is intuitive since the hardware which was eliminated during subsetting was likely reduced in size during the tuning of the application-tuned processor. For example, FIR uses no shifting, therefore a small serial shifter is chosen during tuning and later removed in subsetting, resulting in a less dramatic area reduction. There is a large variation when deciding between these two methods: some benchmarks such as FIR achieve up to 25% increased performance-per-area by using the application-tuned processor over the subsetted general-purpose processor, while others such as QSORT achieve a 25% increase by using the subsetted general-purpose processor over the application-tuned processor (i.e., they are opposite). These two methods are very competitive as summarized in Figure 14 by the AVERAGE bars, which show the subsetted general-purpose processor with slightly higher performance-per-area than the application-tuned (by only 2.2%).

The subsetted application-tuned processor combines all customizations (both the microarchitectural tuning and the ISA subsetting) and therefore often achieves the highest performance-per-area. In some cases a single technique actually does better but closer inspection reveals that these are typically a result of the inherent noise in CAD algorithms. For example, for FIR subsetting the general purpose processor achieves a frequency improvement of 5.5% while subsetting the tuned architecture actually resulted in a frequency decrease of 3.2% which when combined created a less efficient processor in spite of any area savings. The combination of the two techniques is mostly complementary: on average, subsetted application-tuned processors achieve more than 10% better performance-per-area across the benchmark set than either microarchitectural tuning or ISA subsetting alone. However, for each benchmark, either technique on its own can come to within 4% of the combined approach. Overall, the combined approach can improve performance-per-area by 24.5% on average across all benchmarks.

VII. Conclusions

The reconfigurability of soft processors can be exploited to meet design constraints by making application-specific trade-offs in their microarchitecture, a method that requires a complete understanding of soft processor microarchitectures and how they perform/map on/to FPGA devices. In this article we use the soft processor RTL implementations generated by the SPREE infrastructure to study soft processors on both of these fronts: (i) we explore the influence of this new FPGA hardware platform on the trade-offs within soft processor microarchitecture; and (ii) we explore the impact of customizing soft processors to their applications including through the use of our ISA subsetting technique which removes all hardware not required by the application.

Using our SPREE soft processor generator, we explored tuning the microarchitecture by selecting the best processor from a large number of processor variants for a specific application. We determined that the best of these application-specific processors offers considerable advantages over the best-on-average general-purpose processor: an improvement in performance-per-area of 14.1% on average across all benchmarks. This significant efficiency improvement will likely increase as we consider additional architectural axes in future work. Also, we saw that the wide range of processors provided by SPREE were more efficiently mapped to applications than the three Nios II variants which span a larger design space.

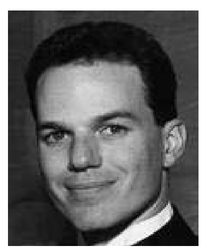
Finally we used the SPREE infrastructure to perform ISA subsetting, where for each application the hardware support for unused features of the ISA are removed. With this technique we obtain large reductions in the area and power of the resulting processors—reductions of approximately 25% for each metric on average and up to 60% for some applications. Combining our techniques for microarchitectural tuning with ISA subsetting results in an even more dramatic benefit where performance-per-area is improved by 24.5% on average across all applications.

In the future we will explore a more broad set of customizations including branch prediction, caches, datapath width, VLIW datapath parallelism, and other more advanced architectural features. We also plan to investigate more aggressive customization of these processors, including changing the ISA to encourage customization. Finally, we are interested in exploring the benefits of tuning the compiler based on exact knowledge of the target architecture.

REFERENCES

[1] "Nios II," http://www.altera.com/products/ip/processors/nios2, Altera.
[2] "MicroBlaze," http://www.xilinx.com/microblaze, Xilinx.
[3] J. Turley, "Survey: Who uses custom chips," *Embedded Systems Programming*, August 2005.
[4] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," in *FPGA '06: Proceedings of the 2006 international symposium on Field-programmable gate arrays*. ACM Press, 2006.
[5] P. Yiannacouras, J. Rose, and J. G. Steffan, "The Microarchitecture of FPGA Based Soft Processors," in *CASES'05: International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM Press, 2005, pp. 202–212.
[6] P. Yiannacouras, J. G. Steffan, and J. Rose, "Application-specific customization of soft processor microarchitecture," in *FPGA'06: Proceedings of the International Symposium on Field Programmable Gate Arrays*. New York, NY, USA: ACM Press, 2006, pp. 201–210.
[7] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," in *Proceedings of the international conference on Compilers, architectures and synthesis for embedded systems*. ACM Press, 2003, pp. 137–147.
[8] J. M. Arnold, "S5: the architecture and development flow of a software configurable processor," in *FPL'05: Proceedings of the International Conference on Field Programmable Technology*. IEEE, December 2005, pp. 121–128.
[9] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable coprocessor," in *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1997, pp. 12–21. [Online]. Available: citeseer.ist.psu.edu/hauser97garp.html
[10] P. Biswas, S. Banerjee, N. Dutt, P. Ienne, and L. Pozzi, "Performance and energy benefits of instruction set extensions in an fpga soft core," in *VLSID '06: Proceedings of the 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 651–656.
[11] R. Lysecky and F. Vahid, "A study of the speedups and competitiveness of fpga soft processor cores using dynamic hardware/software partitioning," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 18–23.
[12] R. Dimond, O. Mencer, and W. Luk, " CUSTARD - A Customisable Threaded FPGA Soft Processor and Tools ," in *International Conference on Field Programmable Logic (FPL)*, August 2005.
[13] M. Itoh, S. Higaki, J. Sato, A. Shiomi, Y. Takeuchi, A. Kitajima, and M. Imai, "PEAS-III: An ASIP Design Environment," Austin, TX, September 2000.
[14] P. Mishra, A. Kejariwal, and N. Dutt, "Synthesis-driven Exploration of Pipelined Embedded Processors." in *VLSI Design*, 2004, pp. 921–926.
[15] G. Braun, A. Nohl, W. Sheng, J. Ceng, M. Hohenauer, H. Scharwachter, R. Leupers, and H. Meyr, "A novel approach for flexible and consistent ADL-driven ASIP design," in *DAC '04: Proceedings of the 41st annual conference on Design automation*. New York, NY, USA: ACM Press, 2004, pp. 717–722.
[16] K. K. e. Matthias Gries, *Building ASIPs: The Mescal Methodology*. Springer, 2005. [Online]. Available: http://www.gigascale.org/pubs/603.html
[17] G. Gibeling, A. Schultz, and K. Asanovic, "RAMP Architecture Description Language," in *Workshop on Architecture Research using FPGA Platforms in the 12th International Symposium on High-Performance Computer Architecture*, 2006.
[18] J. Gray, "Designing a simple fpga-optimized risc cpu and system-on-a-chip," 2000. [Online]. Available: citeseer.ist.psu.edu/article/gray00designing.html
[19] P. Yiannacouras, "The Microarchitecture of FPGA-Based Soft Processors," Master's thesis, University of Toronto, 2005, http://www.eecg.toronto.edu/∼jayar/pubs/theses/Yiannacouras/PeterYiannacouras.pdf.
[20] ——, "SPREE," http://www.eecg.utoronto.ca/∼yiannac/SPREE/.
[21] "MINT simulation software," http://www.cs.rochester.edu/u/veenstra/, University of Rochester.
[22] D. M. Lewis and et al., "The Stratix^{TM} routing and logic architecture." in *International symposium on Field-programmable gate arrays*, 2003, pp. 12–20.
[23] R. Cliff, "Altera Corporation," Private Comm, 2005.
[24] M. Guthaus and et al., " MiBench: A free, comercially representative embedded benchmark suite," in *In Proc. IEEE 4th Annual Workshop on Workload Characterisation*, December 2001.
[25] A. Lodi, M. Toma, and F. Campi, "A pipelined configurable gate array for embedded processors," in *FPGA '03: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*. New York, NY, USA: ACM Press, 2003, pp. 21–30.
[26] "Dhrystone 2.1," http://www.freescale.com, Freescale.
[27] "RATES - A Reconfigurable Architecture TEsting Suite," http://www.eecg.utoronto.ca/∼lesley/benchmarks/rates/, University of Toronto.
[28] "Stratix Device Handbook," http://www.altera.com/literature/lit-stx.jsp, Altera.
[29] P. Metzgen, "Optimizing a High-Performance 32-bit Processor for Programmable Logic," in *International Symposium on System-on-Chip*, 2004.
[30] J. Mashey, "Internet Nuggets," *Computer Architecture News*, vol. 32, no. 4, September 2004.

Peter Yiannacouras Peter Yiannacouras is a PhD candidate at the University of Toronto ECE department. He completed his MASc from the same department and received his BASc from the Engineering Science program at U of T. He has also worked at Intel Microarchitecture Research Labs. His research interests include processor architecture, embedded processing, FPGA logic architecture, and automatic customization.



J. Gregory Steffan J. Gregory Steffan is currently an Assistant Professor of Computer Engineering at the University of Toronto. He received his B.A.Sc. and M.A.Sc. in Computer Engineering from the University of Toronto in 1995 and 1997 respectively, and his Ph.D. in Computer Science from Carnegie Mellon University in 2003. He has also worked in the architecture groups of MIPS and Compaq. His research interests include computer architecture and compilers, distributed and parallel systems, and reconfigurable computing.

Jonathan Rose Jonathan Rose is a Professor and Chair of the Edward S. Rogers Sr. Department of Electrical and Computer Engineering at the University of Toronto. He received the Ph.D. degree in Electrical Engineering in 1986 from the University of Toronto. From 1986 to 1989, he was a Post-Doctoral Scholar and then Research Associate in the Computer Systems Laboratory at Stanford University. In 1989, he joined the faculty of the University of Toronto. He spent the 1995-1996 year as a Senior Research Scientist at Xilinx. In October 1998, he co-founded Right Track CAD Corporation, which delivered architecture for FPGAs and Packing, Placement and Routing software for FPGAs to FPGA device vendors. He was President and CEO of Right Track until May 1, 2000. Right Track was purchased by Altera, and became part of the Altera Toronto Technology Centre, where Rose was Senior Director until April 30, 2003. His research covers all aspects of FPGAs including their architecture, Computer-Aided Design (CAD), Field-Programmable Systems, Soft Processors, and graphics, vision and bio-informatic applications of programmable hardware.