THE SUPERSMALL SOFT PROCESSOR

James Robinson, Sam Vafaee, Jonathan Scobbie, Michael Ritche and Jonathan Rose

The Edward S. Rogers Sr.
Department of Electrical and Computer Engineering,
University of Toronto, Toronto, Ontario, Canada
email: jayar@eecg.utoronto.ca

ABSTRACT

Soft processors have become an increasingly common component of systems that use Field-Programmable Gate Arrays (FPGAs), and are used to implement a wide variety of control and data processing functionality. Often, some additional functionality needs to be added to a system when there is very little space left on the physical device. This functionality may not be performance critical, and so could be implemented on a slow soft processor. For this reason it may be useful to have a processor that is as small as possible yet similar to other commonly-used processors. This paper describes the design, implementation and release of a 32-bit soft processor based on the MIPS-I instruction set and optimized for minimal use of FPGA resources. The 'supersmall' soft processor is as much as 2.2 times smaller than Altera's Nios II/e (the smallest of their 3 processors) yet only a factor of 10 times slower.

1. INTRODUCTION

Soft processors — processors implemented in programmable logic, have become an increasingly common component of FPGA-based designs [1]. Although slower than "hard" processors — processors implemented directly in transistors, soft processors have found wide application where the fastest speeds are not required, because they are easily integrated along with other logic in an FPGA. For example, in [2] a soft processor is used to implement some subsystems of the robot's control system, while time-critical components are performed in hardware. Other examples of this kind of use include [3] and [4].

The speed requirements of software applications running on a soft processor will vary dramatically; we believe that there will be some applications that have very low speed requirements, perhaps far less than the capabilities of any currently available soft processors. In this case, it would be beneficial to have a processor that is as small as possible, allowing the designer to perhaps use a smaller (and cheaper) FPGA, or to squeeze that 'last little bit' of functionality onto a system with just a few soft logic resources left. The *supersmall* soft processor is an attempt to explore the degree to which both of these design constraints (usability and low FPGA resource consumption) can be met. It is a 32-bit processor that implements a large subset of the MIPS-I instruction set architecture (ISA) [5]. At the same time, it has been designed to consume the fewest possible FPGA resources. Ideally, the supersmall processor is intended to be a drop-in replacement, supplanting another soft processor that is too large to fit in the remaining space of an FPGA. We note that a plausible alternative to this approach might be to run the desired code on a faster, (and likely bigger) processor that is already in place on the FPGA. This may be appropriate, but won't work if there isn't such a processor, or if the computational capacity of that processor is already used, or if the complexity of managing the resulting multi-threading is undesirable.

2. RELATED WORK

Prior research on soft processors [6] [7] has focused on exploring their architecture and improving the performance and/or the area-delay product. The present work focuses exclusively on area, to the exclusion of performance. Some previous efforts — such as Altera's Nios II processor [8], and the Eric5 super-small soft-core CPU [9], have also addressed area optimization. The work in [8] for instance, focused on the the design of area-efficient multiplexers in FPGAs, and the ensuing software and hardware workarounds. The work in [9] and [10] created a custom ISA designed for efficient implementation on an FPGA; this necessitated the creation of an entire custom software support toolchain, which prevents the re-use of a large software infrastructure. As well, almost all soft processors intended for area efficiency, including not only [9] but also Xilinx's Picoblaze [11] and Lattice's Mico8 [12] feature a word length of less than 32 bits, which limits the memory that can be addressed without the use of memory segmentation. Our supersmall processor employs full 32-bit memory access and data registers.

3. TARGET FPGA ARCHITECTURE, SOFTWARE TOOLS AND SETTINGS

The supersmall soft processor is implemented on an Altera Stratix III FPGA [13]. A key feature of this FPGA is its fundamental logic unit, consisting of Adaptive Logic Modules (ALMs), which are themselves composed of two Adaptive Look-up Tables (ALUTs) and two registers. These ALMs can implement certain commonly used logic structures, such as shift registers and counters with particularly good efficiency, something we make use of in this work. On the Stratix III, ten ALMs are coalesced into groups called Logic Array Blocks, or LABs.

Note that this processor could been implemented on other Altera FPGAs; the one vendor-specific part of the Verilog code that we use is the instantiation of memories inside the processor. Although some optimizations have been made that take advantage of the Stratix III's ALM, the core methodology for resource minimization is fairly device-agnostic, and the Stratix III results should be fairly representative for FPGAs in general.

4. INSTRUCTION SET ARCHITECTURE

The MIPS-I instruction set architecture (ISA) was originally introduced by Hennessy et al. in [14] and was designed both for performance and simplicity of hardware implementation. The MIPS ISA is one of the broadly used embedded processor instruction sets. One of the original features of the MIPS-I ISA was a feature known as "branch delay slots," whose purpose was to give the pipeline useful work to do while a branch is resolved. This is a performance-oriented architectural feature, but because our goal is to make a very small processor, not a very fast one, it simply gets in the way. However, we chose not to remove the branch delay slot as this would make the object code of our processor completely incompatible with that of MIPS-I.

Branch delay slots are supported on the supersmall through the use of a second instruction register, a design that costs no performance and uses minimal logic, but does consume an additional 32 registers. In this regard, designs such as [9], which created their own custom ISA, are ultimately more capable of minimizing their resource footprint.

The following instructions from the MIPS-I instruction set are also not supported: multiplication and division (mult, div), floating point operations (add.s, mul.s) and unaligned loads and stores (lwl, swr). For complete details on the instruction set coverage, see: `http://www.eecg.utoronto.ca/~jayar/software/SuperSmallProcessor/isa.html`. Note also that there were some MIPS-I instructions that were covered by patents, but those patents have now expired.

5. PROCESSOR DESIGN

The goal of the supersmall processor design is to be as small as possible in the Altera Stratix III FPGA, without regard to performance. Figure 1 gives an overview of the architecture of the processor. In the following sections, we describe each of the major elements of the processor (the arithmetic logic unit (ALU), the multiplexers, and the exceptions support) and how we strove to minimize its area.

5.1. Arithmetic Logic Unit

The key area saving achieved in the supersmall processor is to implement the 32-bit arithmetic and logic operations one bit at a time, in a serial ALU.

The ALU is fed by two 32-bit shift registers, *A* and *B*. In each clock cycle, a single bit result is computed, which is then fed back into either *A* or the NPC register, as appropriate. This serialization gives rise to the key performance penalty (but also area win) of the supersmall processor, as it will be roughly 32 times slower than a full 32-bit ALU.

5.2. Multiplexers

According to Metzgen in [8], "The key to optimizing designs for an FPGA is to optimize the multiplexers," since the implementation of multiplexers in programmable logic is particularly inefficient.

One method that we employ to decrease multiplexer resource usage is similar to that used to implement the ALU: by serializing the datapath. When a 32-bit bus is multiplexed, whatever logic is needed to implement the multiplexer must be multiplied by 32 to implement it for the entire bus. By serializing a bus, a 32-bit multiplexor becomes a 1-bit multiplexor, with tremendous area saving.

This concept was applied when designing the supersmall's memory subsystem. The MIPS instruction set has separate instructions to access a word, half-word, or byte from the 32-bit memory bus. Our initial implementation simply multiplexed the 32-bit bus for each instruction. Instead, a far more area-efficient method is to use a shift register to align the data as needed; this again trades away performance for area, reducing the overall area by roughly 65 ALUTs.

5.3. Exceptions

We envision that the supersmall processor will be used in control operations, which will have need for interrupts and other types of exceptions. For this reason we have included an option for the processor to handle exceptions, which, if not needed, can be eliminated for more area savings.

The MIPS-I specification declares that exceptions support is implementation-dependent; we chose to create an

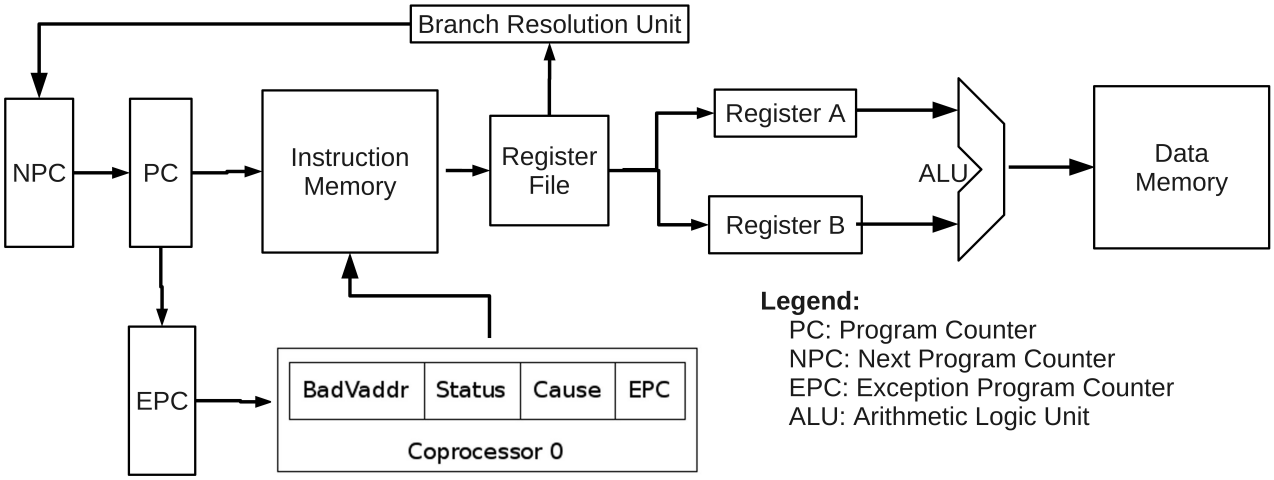

Fig. 1. Overall Architecture of Supersmall Processor

interface similar to that found in the earliest MIPS processors [5]: this consists of a set of registers contained in the system coprocessor, or "coprocessor 0", which are updated whenever an exception occurs and are accessed in the same way as data memory. When an exception occurs, execution jumps to a hard-wired location in memory, from which software handles the exception. The co-processor exception registers are contain information about the exception, including the address where the exception occurred and the type of exception. Interrupts can be enabled or disabled by writing to these registers.

The coprocessor registers were also implemented as shift registers, and so their values are also accessed serially. These registers, however, are relatively expensive: the overall cost of full exception support is 94 ALUTs and 167 flip-flops. While this would small in most processors, it is significant given our area minimization goal. As a result we have included an option in the the supersmall processor Verilog code to reduce or remove the exception support, as summarized in Table 1.

Support Level	# ALUTs	# Flip-Flops	# ALMs	# LABs
Basic	76	136	73	9
Arith Ovflow	4	0	10	1
Coproc Unuse	0	0	0	0
Res Instr	7	0	13	0
Addr Err	10	32	22	2

Table 1. Area Cost of exceptions support

5.4. Integration with Altera System Construction Environment

The supersmall soft processor has been designed, so far, for the Altera FPGA environment. Altera provides a tool, called 'System on a Programmable Chip' (SOPC) Builder, which allows easy connection of different components to its soft processors, their peripheral devices, as well as connections among multiple processors.

SOPC builder uses the Avalon Bus standard [15] to connect the various components. To enable integration as an SOPC Builder component, we added an optional Avalon Bus interface to the supersmall processor's external memory bus. If this option is enabled in the processor's Verilog code, this interface is exposed for external connection; the alternative (if the option isn't used) declarations of on-chip internal data and instruction memory are disabled. The Avalon Bus port costs an additional 32 flip-flops, since it assumes a registered instruction bus, something that the Avalon Bus does not provide.

6. MEASUREMENT RESULTS

In this section we give measurements of the size and performance of various versions of the supersmall processor, and compare it to the Altera Nios II 32-bit processor. The Nios II comes in three separate versions which provide a tradeoff between area and speed. The Nios II/e has been specifically optimized for efficient area usage [16].

Processor	ALUTs	Flip-flops	ALMs	LABs	LAB count relative to w/o exceptions	LAB count relative to w/exceptions
Supersmall w/o Exceptions	138	200	152	16	1.0	0.6
Supersmall w/Exceptions	236	368	242	27	1.7	1.0
Nios II/e	491	302	337	35	2.2	1.3
Nios II/s	787	592	589	68	4.2	2.5
Nios II/f	1085	984	897	103	6.4	3.8

Table 2. Measured Area Comparison between Supersmall and Nios II

6.1. Measurement Methodology

All measurements of processor area - the number of ALUTs, ALMs, and LABs - were performed using Altera's Quartus II version 9.0 software. To do this, we created a simple SOPC builder design, consisting of a 128K bytes of read-only instruction memory and a 64K bytes read-write data memory connected to the processor's instruction and data busses, respectively, along with a JTAG UART connected to the data bus. Since the Nios and the supersmall's external interfaces both consist only of an instruction and a data bus, this presented no difficulty. SOPC builder could then be used to generate a new setup for each processor, on which measurements could be made.

The processor component of the SOPC generated system was placed in a Logic Lock region [17], which both constrained the full synthesis, packing, and placement algorithms to attempt to fit it into the smallest region possible, while also allowing measurement of all logic parameters of that section in isolation. All measurements given are for that specific logic-locked region, with no constraints placed on its size or location.

All compile options for Quartus were chosen to minimize the area occupied by the processor. The complete list of settings employed can be downloaded (in the file supersmall.qsf) from the website given at the end of this paper; several that were found to be of significant importance are described here. Timing-driven synthesis was disabled, (because we wish to optimize for area). This turned out, surprisingly, to be particularly important for the many shift registers we employed in the design - the timing-driven synthesis of parallel load shift registers used much more soft logic than the non-timing driven optimization.

As well, the Quartus setting "minimize area with chains" was set while packing registers, which ensures that the packing of flip-flops into ALUTs is as aggressive as possible. Both of these settings significantly reduced both ALM and LAB counts.

6.2. Area Measurements

Table 2 compares the area of the supersmall processor, with and without exception handling, to the area of the three versions of the Nios II processor from Altera.

In configuring the Nios II processor, to ensure a fair comparison, no memory management unit, debug console, or additional exceptions were enabled. Hardware multiply support was enabled where relevant, since this would likely be enabled on a Stratix III.

Table 2 gives the count of ALUTs, flip-flops, ALMs and LABS required for each processor. Recall that ALUTs and flip-flops are packed into ALMs, and ALMs are packed into LABS, and so the clearest view of area comes from the LAB counts. As can be seen, the supersmall processor without exceptions is 2.2 times smaller than the Nios II/e in terms of LAB count. The supersmall processor with exceptions is 1.3 times smaller than the Nios II/e.

Note that the supersmall with exceptions has a significantly greater advantage in ALUT count over the Nios II/e, but it actually uses more flip-flops than the Nios II/e; in a flip-flop-short environment, this may make it less desirable. This is a consequence of the supersmall's attempt at compatibility with the exception-handling interface of early MIPS processors, rather than devising its own FPGA-optimized scheme, such as that featured in the Nios II. A more carefully optimized implementation of the exceptions should reduce the size of the supersmall with exceptions.

6.3. Performance Measurements

In this section we describe the measurement of the performance of the supersmall processor against the Nios II processors. We benchmark them using the SPREE benchmark suite [7], which is a freely available set of 20 different embedded processor applications. These benchmarks were chosen because of their variety, public availability, and their previous usage in soft processor performance comparisons. No benchmark involved the use of exceptions, however, so this does not provide any indication of the relative performance of the processors' implementation of exceptions.

Each benchmark was compiled with a modified version of gcc (available at the download site given at the end of this paper), with the compile options *-O3 -mips1 -mgp32 -msoft-float -mlong32*. To obtain valid measurements, each of the benchmarks was executed on a software simulation of

	Execution Cycle Counts			
Benchmark	Supersmall	Nios II/e	Nios II/s	Nios II/f
bubble_sort	171620	13854	4316	3993
crc	871282	183360	31773	25889
des	94668	13492	3617	3533
fft	714642	113146	3650	3007
fir	90776	8864	1310	1752
quant	418589	127358	5944	5167
iquant	575782	119088	3286	3667
turbo	13752420	2233341	516404	424913
vlc	1364847	193697	44391	29911
bitcnts	1878424	214020	55995	41702
CRC32	22209701	503390	125886	142940
qsort	3720720	384057	132893	103568
sha	2690600	252155	58245	41677
stringsch	5402994	580213	134602	149204
FFT	51881436	6084367	883319	633635
dijkstra	19657914	1766350	427187	370040
patricia	6962527	619453	261025	193196
gol	12106839	2173464	353082	251718
dct	71264602	6948710	903412	646663
dhry	3636853	460323	139207	107328
Geo Mean	2606121	309051	53290	45439
v II/e	8.4	1.0	0.17	0.15
v II/f	57	6.8	1.2	1.0

Table 3. Benchmark Cycle Counts on each Processor

the respective processors, using Mentor Graphics' Modelsim simulator. These simulations gave a precise cycle count for each program on each processor, as given in Table 3.

The processor was then run through exactly the same CAD flow as that used for the size comparisons above, and the maximum frequency determined by Quartus' TimeQuest Timing Analyzer using the slow 1100mV 85°C timing model. To achieve the highest frequency from the placement and routing, the clock constraint was set to 1 GHz. All version of the supersmall processor were found to have the same critical path, and therefore, the amount of exceptions support included is not specified. The mean wall clock execution time could then be determined by the quotient of the geometric mean cycle count and the max frequency, as provided in Table 4.

Table 4 shows that the Nios II/e is about 10 times faster than the supersmall, and that the Nios II/f is about 60 times faster. As expected, there is a considerable speed penalty to be paid, particularly for the serialization of the processor. However, it is interesting to note that the speed penalty is not the 32 times cycle count penalty involved with ALU instructions, as other instructions do not require work on all 32 bits of a word. Also, as we discussed in the introduction, the processor's application is for non-speed critical workloads.

7. RELEASE OF CODE

The complete Verilog source code for the supersmall processor is available (under a liberal 2-clause BSD license) from the following location: `http://www.eecg.utoronto.ca/~jayar/software/SuperSmallProcessor/`.

That website contains documentation, as well as a slightly modified GNU compiler toolchain targeted towards the processor.

8. CONCLUSIONS AND FUTURE WORK

In this work we have designed, built, measured and released the smallest possible soft processor implementing a subset of the MIPS-I instruction set on an Altera Stratix III FPGA. This processor is meant to be used in the circumstance when area is the critical resource and performance much less of a constraint. As such, we have sacrificed significant (and disproportionate!) performance to achieve a minimum area. The smallest version of the processor is 2.2 times smaller than the Nios II/e commercial soft processor from Altera, and operates a factor of 10 slower in wall-clock performance.

In the future, we hope to explore the use of unused memory blocks for aiding the area reduction, as well as further

Processor	Mean Cycle Count	Fmax (MHz)	Wall Clock Execution Time (us)	Wall Clock Relative to Nios II/e	Wall Clock Relative to Nios II/f
Supersmall	2606121	221.2	11783	9.8	59
Nios II/e	309051	257.5	1200	1.0	6.1
Nios II/s	53289	201.0	265.1	0.22	1.3
Nios II/f	45438	229.1	198.3	0.17	1.0

Table 4. Performance Comparison of Supersmall and Nios II Processors

optimizations of the present core. In addition, we intend to make the device easily portable to other FPGA vendors' devices.

9. REFERENCES

[1] T. Allen, "Private communication, altera corporation," Altera Corporation, 2009.

[2] J. González-Gómez, E. Aguayo, and E. Boemo, "Locomotion of a Modular Worm-like Robot using a FPGA-based embedded MicroBlaze Soft-processor," in *Proceedings of International Conference on Climbing and Walking Robots (CLAWAR 2004)*. Springer, 2004.

[3] J. Wang, Y. Chen, J. Xie, B. Chen, and H. Lin, "System structure for FPGA-based SOPC design using hard tasks," in *2008 6th IEEE International Conference on Industrial Informatics (INDIN)*. The Institution of Engineering and Technology, 2008.

[4] Y. Wu, X. Duan, Z. Lian, T. Shen, S. Zhao, and L. Yan, "The design of storage system for digital airborne camera based on SOPC," in *Proceedings - 1st International Congress on Image and Signal Processing, (CISP) 2008*. Inst. of Elec. and Elec. Eng. Computer Society, 2008.

[5] D. Sweetman, *See MIPS Run, Second Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[6] R. Lysecky and F. Vahid, "A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning," in *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*. IEEE Computer Society Washington, DC, USA, 2005, pp. 18–23.

[7] P. Yiannacouras, J. Rose, and J. Steffan, "The microarchitecture of FPGA-based soft processors," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. ACM New York, NY, USA, 2005, pp. 202–212.

[8] P. Metzgen and D. Nancekievill, "Multiplexer restructuring for fpga implementation cost reduction," in *Design Automation Conference, 2005. Proceedings. 42nd*, June 2005, pp. 421–426.

[9] E. Electronics, "Eric5: 9bit-soft-core-cpu for fpgas," http://www.entner-electronics.com/tl/index.php/eric5.html.

[10] Z. Consulting, "Zpu," http://opensource.zylin.com/zpu.htm.

[11] X. Inc., "Picoblaze processor," http://www.xilinx.com/picoblaze.

[12] L. Semiconductor, "Latticemico8," http://www.latticesemi.com/products/intellectualproperty/referencedesigns/8bitmicrocontrollermico8.cfm.

[13] *Stratix III Device Handbook*, 1st ed., Altera Corporation, July 2009.

[14] J. Hennessy, N. Jouppi, F. Baskett, and J. Gill, "MIPS: a VLSI processor architecture," in *Proc. CMU Conference on VLSI Systems and Computations*, 1981, pp. 337–346.

[15] Altera, "Avalon bus specification," http://www.altera.com/literature/manual/mnl_avalon_spec.pdf.

[16] ——, "Nios ii/e core: Economy," http://www.altera.com/products/ip/processors/nios2/cores/economy/ni2-economy-core.html.

[17] ——, "Using the logiclock methodology in the quartus ii design software," http://www.altera.ru/Disks/Altera%20Documentation%20Library/literature/an/an161.pdf.