Experiences with Soft-Core Processor Design

Franjo Plavec Blair Fort Zvonko G. Vranesic
Stephen D. Brown
University of Toronto
Department of Electrical and Computer Engineering
Toronto,ON, Canada
{plavec, fort, zvonko, brown}@eecg.toronto.edu

Abstract

Soft-core processors exploit the flexibility of Field Programmable Gate Arrays (FPGAs) to allow a system designer to customize the processor to the needs of a target application. This paper describes the UT Nios implementation of Altera's Nios architecture. A benchmark set appropriate for soft-core processors is defined. Using the benchmark set, the performance of UT Nios is explored and compared with the commercial implementation.

1. Introduction

Field Programmable Gate Arrays (FPGAs) are becoming increasingly popular for implementation of logic circuits. Inclusion of abundant memory resources in FPGAs has made them suitable for implementation of embedded SOPC systems, where a complete system fits on a single programmable chip. A processor unit in such a system is usually a soft-core processor.

A soft-core processor is a microprocessor fully described in software, usually in an HDL, which can be synthesized in programmable hardware, such as FPGAs. Soft-core processors implemented in FPGAs can be easily customized to the needs of a specific target application. The two major FPGA manufacturers provide commercial soft-core processors. Xilinx offers its MicroBlaze processor [1], while Altera has Nios and Nios II processors [2]. In this paper we focus on the Nios architecture.

Nios is a RISC soft-core processor optimized for implementation in Altera FPGAs. Many architectural parameters of Nios can be customized at design time, including the datapath width, register file size, cache size, custom instructions and others. In this work, we present a new implementation of the Nios architecture, called UT Nios [3].

In the paper we refer to the original Nios implementation as Altera Nios, and use the term Nios for the Nios architecture, independent of the implementation. To assess the performance of the UT Nios and compare it with Altera Nios, a UT Nios benchmark set was defined.

The paper is organized as follows. In section 2, the key features of the Nios architecture are described, and the UT Nios implementation is presented. Section 3 defines the UT Nios benchmark set, and gives the performance comparison of UT and Altera Nios is given. Section 4 concludes the paper.

2. UT Nios

2.1. Nios architecture

Nios is a highly customizable RISC architecture. One of the architectural parameters that is selected at design time is the width of the datapath, which can be 16 or 32 bits [4]. The instruction word is 16 bits wide, regardless of the datapath width, to reduce the code size compared to the 32 bits wide instruction word. We will focus on the 32-bit datapath, which is more likely to be used in high performance applications.

The Nios instruction set supports only the basic arithmetic and logic instructions, with optional support for integer multiplication. The instruction set can be augmented with up to five custom instructions. A custom instruction performs a user-defined operation in hardware, thus speeding-up the critical parts of an application [5]. Nios has a load-store architecture, with most instructions using register file operands. Although the size of the register file is configurable, only 32 registers are visible to programs at any given time. The register file is divided into register windows. The sliding windows overlap, facilitating fast context switching on entrance and exit from procedures.

This work was supported in part by NSERC, CITR and University of Toronto Master's Fellowship

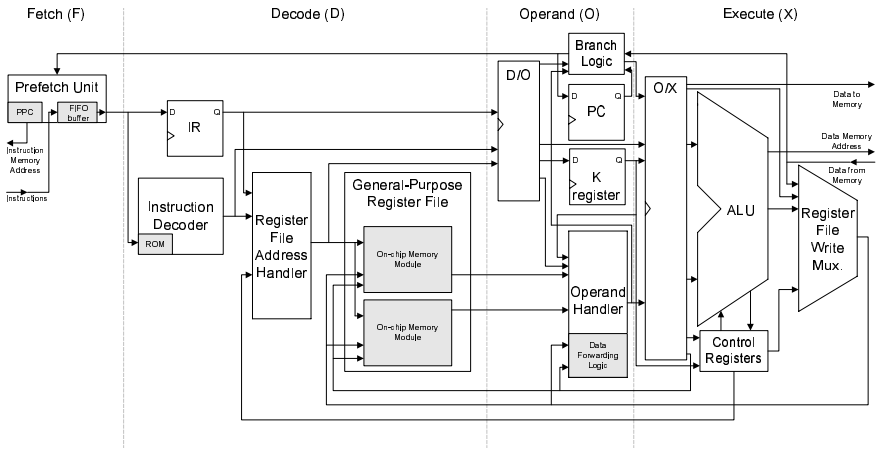

Figure 1. UT Nios datapath

Processor, memory and other peripherals are interconnected using the automatically generated Avalon bus [6]. This is a synchronous bus defining the interface and protocol for connecting master and slave components in the system. Memory modules and peripherals in the system can reside on or off-chip, but the bus logic is always placed on-chip.

2.2. UT Nios datapath

UT Nios datapath, shown in Figure 1, is organized in 4 pipeline stages: fetch (F), decode (D), operand (O), and execute (X). Prefetch unit fetches instructions from the instruction memory. The address of the next instruction to be fetched is stored in the prefetch program counter (PPC). If the pipeline stalls, the fetched instruction is temporarily stored in a FIFO buffer. Otherwise, the instruction is forwarded to the decode stage, where it is decoded and addresses of its register operands are calculated.

In the operand stage, immediate operands are formed. Since the instruction word is only 16 bits wide, it is not possible to specify a 16-bit immediate constant in a single instruction word. 16-bit constants are formed by preloading an 11-bit prefix into the K register using the PFX instruction. An instruction following PFX specifies the remaining 5 bits to form a 16-bit immediate operand. Since the Nios architecture supports several different addressing modes, depending on the addressing mode used the operands may be coming from several different sources. Therefore, proper operands have to be selected from the possible sources. Operand handling also includes the data forwarding logic. Branch instructions commit in the operand stage, because all of the necessary information is available (through the data forwarding logic); this reduces the branch penalty.

Most instructions commit in the execute stage. The required calculations are performed by the arithmetic and logic unit (ALU). The results are stored in the register file and control registers. Since the result of the instruction execution may be coming from different sources, such as memory for load instructions or ALU for arithmetic and logic instructions, the result is selected by a register file write multiplexer.

All UT Nios components were described in Verilog HDL. The current version of UT Nios does not support some optional components, such as instruction and data caches, multiplication support in hardware, and advanced debug support [3].

3. Performance evaluation

3.1. UT Nios benchmark set

A set of benchmark programs was defined to assess the performance of UT Nios and compare it with the performance of Altera Nios. Although several benchmark sets for embedded systems already exist, we found most of them to be inappropriate for benchmarking soft-core processors. Since many of these benchmarks contain computationally intensive programs, which are much more efficiently implemented in custom hardware, it is not likely that a soft-core processor would be used for such an application. A new set of benchmark programs was defined to measure the performance of soft-core processors.

The UT Nios benchmark set is based on MiBench [7], a freely available benchmark suite. A subset of the MiBench benchmarks was selected and augmented with other programs that better characterize the performance of UT Nios. Since Nios-based systems generally do not include a file-

system, all benchmarks were adapted to use the input data from memory.

The UT Nios benchmark set is divided into three categories: *test benchmarks*, which test the performance of specific architectural features of the processor, *toy benchmarks*, which are small programs performing simple calculations, and *application benchmarks* that represent the real applications expected to run on the processor.

Test benchmarks include *Loops*, that tests the performance of branch instructions, *Memory*, which performs operations on an array residing in memory, *Pipeline*, which tests the data forwarding capabilities of the pipeline implementation, and *PipelineM*, which tests the data forwarding when instruction operands are in the data memory.

Toy benchmarks include *Fibo*, which calculates the 9^{th} Fibonacci number using a recursive procedure, *Multiply*, which uses a triply nested loop to multiply two integer matrices, and *QsortInt*, a program that sorts an integer array using the qsort algorithm.

Application benchmarks include *Bitcount*, which counts the number of bits equal to 1 in an integer array using five different bit counting algorithms, *CRC32*, a 32-bit Cyclic Redundancy Check (CRC) algorithm, *Dijkstra*, a shortest-path algorithm, John Conway's *Game of Life (GOL)*, *Patricia*, an algorithm that searches the *Patricia trie* data structure, *Qsort* algorithm to sort a large data set, *SHA*, which produces a message digest using the secure hash algorithm, and *SSearch*, which performs a search for a given string in an input text. Some of the application benchmarks have two datasets. Details on the benchmarks and dataset sizes can be found in [3].

3.2. Experimental environment

Performance comparison between the UT and Altera Nios was performed using a Nios-based system consisting of the following components:

- 32-bit Altera Nios 3.0 or UT Nios 1.01 without cache and multiplication support, with 512 registers in the register file, interrupt support, writable WVALID register and RLC/RRC instruction support. Altera Nios's pipeline optimization was selected for "Fewer Stalls / More LEs" and its instruction decoder was implemented in logic. UT Nios had the decoder implemented in memory, and two registers in a FIFO buffer. These parameters were selected to maximize the performance of both implementations, while keeping them comparable (e.g. enabling cache support for Altera Nios would increase its performance, but since UT Nios does not support cache, the comparison would not be fair).
- 16 Kbytes of 32-bit on-chip memory with a monitor program
- 1 MB of 32-bit off-chip SRAM that contains the program, the data, and the interrupt vector table.
- Avalon tri-state bridge with default settings
- UART peripheral with default settings used to download the programs to the development board and communicate with the program.
- Timer peripheral with default settings used to measure the program execution time.

The processor's instruction and data masters were both connected to the on-chip memory and the tri-state bridge, while the data master was also connected to the UART and timer peripherals. The tri-state bridge was connected to the off-chip SRAM. All arbitration priorities were set to 1. Both systems were compiled using Quartus II 4.1 SP2 targeting the Stratix *EP1S10F780C6ES* FPGA chip, optimizing the design for speed, and using the standard fit (highest effort) fitter option. The benchmarks were compiled using the gcc compiler for Nios, with optimization level 3. Benchmark programs were run on the Stratix development board at 50 MHz. Since both designs ran at the same operating frequency, comparison of the resulting execution times directly shows the cycle count ratio of the two processors. To obtain the wall clock execution times, the times measured were prorated to account for the difference in the F_{max}. The F_{max} was obtained by synthesizing each design using ten different seeds (used by the placement algorithm in the CAD design flow), and choosing the best result obtained. The F_{max} for the Altera Nios is 112 MHz, while the UT Nios achieves the F_{max} of 77 MHz.

3.3. Performance comparison

The results of the performance comparison of Altera and UT Nios are presented in Table 1. All values show the cycle count improvement and wall clock speedup of UT Nios over Altera Nios. Considering the cycle count ratio helps to understand the performance difference of the two processor implementations. For benchmarks that have two dataset sizes, the average of the two is given. Similarly, for the Bitcount benchmark, the average of all five algorithms is given.

The UT Nios has lower cycle count for Loops and Memory benchmarks, because it implements branches and memory operations more efficiently. Branches are committed early in the pipeline (in the operand stage), while memory operations do not have a dedicated pipeline stage. Since the Altera Nios has a deeper pipeline, it is expected that branch penalties will be greater. Results for the Memory benchmark, together with a significantly higher operating frequency of Altera Nios (compared to UT Nios) suggest that the original implementation has a memory pipeline stage. The difference in the F_{max} results in the UT Nios being

Table 1: Performance comparison of UT and Altera Nios

Benchmark	Loops	Memory	Pipeline	PipelineM	Fibo	Multiply	QsortInt	Bitcount	CRC32	Dijkstra	GOL	Patricia	Qsort	SHA	SSearch	Average
Cycle Count Advantage	1.56	1.43	1.05	1.18	1.26	2.26	1.68	1.45	1.32	1.47	1.40	1.68	1.51	1.43	1.35	1.46
Wall Clock Time Advantage	1.08	0.98	0.72	0.82	0.87	1.56	1.16	1.00	0.91	1.01	0.97	1.16	1.04	0.99	0.93	1.01

only 8% faster than the Altera Nios for the Loops benchmark, and 2% slower for the Memory benchmark.

The Pipeline benchmark performance in terms of cycle count reveals that both the UT and Altera Nios implement full data forwarding when operands are in the register file. The UT Nios performs better in terms of the cycle count for PipelineM benchmark because it performs memory operations better, as argued above. The difference in F_{max} causes poor overall performance of the UT Nios for these two benchmarks.

The above results for the test benchmarks help us understand the performance of other benchmarks. For instance, UT Nios exhibits the biggest advantage over Altera Nios for the Multiply benchmark. The reason is that the Multiply benchmark is both memory- and control-flow-intensive, which are the two characteristics for which the UT Nios performs better. Similarly, Qsort, and Patricia are also memory intensive programs, and they experience better performance on UT Nios. On average, over all benchmarks, UT Nios has 46% advantage in terms of cycle count, but is only 1% faster than Altera Nios in terms of the wall clock execution time.

Another important factor for processor implementation is the amount of logic (in LEs) and memory resources (in bits) used on the chip. The UT Nios system uses 3,000 LEs (2,498 LEs for the processor) and 166,304 memory bits (35,232 bits for the processor). The Altera Nios system uses 2,293 LEs (1,888 LEs for the processor), and 163,840 memory bits (32,768 for the processor). Hence, the UT Nios based system uses 31% more LEs and 2% more memory bits than the Altera Nios based system. We believe that this is due to the device specific optimizations done for the Altera Nios. Examples of such optimizations applied to the Altera Nios ALU targeting the APEX device family can be found in [8]. The UT Nios does not include such optimizations, because our aim was to produce a generic implementation that can be implemented in most modern FPGAs.

4. Conclusions

Soft-core processors play a key role in building systems on a programmable chip. In this paper the UT Nios implementation of the Nios architecture is presented. The UT Nios benchmark set, which can be used to evaluate the performance of soft-core processor architectures is defined. It was used to evaluate the performance of UT Nios.

The UT Nios exhibits performance comparable to that of Altera Nios. On average the UT Nios was found to be 1% faster than the Altera Nios. It is up to 56% faster for some programs, due to different pipeline organization. The UT Nios based system requires 31% more LEs and 2% more memory bits than an equivalent Altera Nios based system, mainly due to device specific optimizations employed in the Altera Nios design.

The UT Nios benchmark set is available at [9].

References

[1] Xilinx, Inc. MicroBlaze Soft Processor, August 2004. htpp://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=microblaze.

[2] Altera Corporation. Embedded Processor Solutions Overview, January 2005. http://www.altera.com/technology/embedded/embedded/emb-embedded_processor_solutions.html.

[3] F. Plavec. Soft-Core Processor Design. Master's thesis, University of Toronto, 2004.

[4] Altera Corporation. Nios Features: CPU Architecture, August 2004. http://www.altera.com/products/ip/processors/nios/features/nio-cpu_architecture.html.

[5] Altera Corporation. AN188: Custom Instructions for the Nios Embedded Processor, September 2002. http://www.altera.com/literature/an/an188.pdf.

[6] Altera Corporation. Avalon Bus Specification Reference Manual, July 2003. http://www.altera.com/literature/manual/mnl_avalon_bus.pdf.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R.B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proc. of the* 4^{th} *IEEE Workshop on Workload Characterization*, pages 3–14, 2001. Austin, TX.

[8] P. Metzgen. A High Performance 32-bit ALU for Programmable Logic. In *Proc. of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 61–70. ACM Press, 2004.

[9] UT Nios Homepage, August 2004. http://www.eecg.toronto.edu/~plavec/utnios.html.