Leros: A Tiny Microcontroller for FPGAs

Martin Schoeberl
Department of Informatics and Mathematical Modeling
Technical University of Denmark
masca@imm.dtu.dk

***Abstract*—Leros is a tiny microcontroller that is optimized for current low-cost FPGAs. Leros is designed with a balanced logic to on-chip memory relation. The design goal is a microcontroller that can be clocked in about half of the speed a pipelined on-chip memory and consuming less than 300 logic cells.**

The architecture, which follows from the design goals, is a pipelined 16-bit accumulator processor. An implementation of Leros needs at least one on-chip memory block and a few hundred logic cells.

The application areas of Leros are twofold: First, it can be used as an intelligent peripheral device for auxiliary functions in an FPGA based system-on-chip design. Second, the very small size of Leros makes it an attractive softcore for many-core research with low-cost FPGAs.

I. Introduction

In this paper we present Leros, a microcontroller that is optimized for low-cost FPGAs. Leros is a minimalistic 16-bit processor intended for utility functions in an FPGA based System-on-Chip (SoC) design. The design goals of Leros are a good balance between the number of logic cells and on-chip memories, reasonable performance, and a high maximum clock frequency. The last point is important for usage as utility processor in an SoC to not be the frequency bottleneck of the whole design.

These design goals are achievable by a pipelined accumulator architecture with additional directly addressable *registers* in an on-chip memory for local variables. The on-chip data memory is shared for those registers and general data. With an additional on-chip memory for the instructions only two memory blocks are needed and the pipeline can execute one instruction per clock cycle. For short programs the instruction memory can even be built out of logic cells (LC).

Leros is named after the greek island Leros,[1] where it has been designed during an enjoyable vacation. Leros is available under open-source and if you use Leros in your application or research, consider a visit to the nice island Leros and sending a postcard from there to the author.

The paper is organized as follows. Section II presents related work. Background on the design decisions for Leros is presented in Section III. The concrete implementation is described in Section IV, followed by an evaluation and comparison with other FPGA microcontrollers in Section V. The paper is concluded with Section VI.

[1] http://www.leros.gr/

II. Related Work

Many small processor cores for FPGAs are available and University courses on computer architecture often contain assignments to implement a simple pipeline in an FPGA. In the following section we restrict the description to a few successful cores and point out the differences of our Leros design.

PicoBlaze is an 8-bit microcontroller for Xilinx FPGAs [12]. The processor is highly optimized for low resource usage. This optimization results in restrictions such as maximal program size of 1024 instructions and 64 byte data memory. The benefit of this puristic design is a processor that can be implemented with one on-chip memory and 96 logic slices in a Spartan-3 FPGA. PicoBlaze provides 16 8-bit registers and executes one instruction in two clock cycles. The interface to I/O devices is minimalistic in the positive sense: it is simple and very efficient to connect simple I/O devices to the processor.

The Leros approach is similar to the concept of PicoBlaze to provide a small processor for utility functions. Compared to PicoBlaze, Leros has fewer restrictions on program and data size as it is a full blown 16-bit processor. Leros is optimized to balance the resource usage between on-chip memory and logic cells. Therefore, the LC count of Leros is slightly higher than the one of PicoBlaze. PicoBlaze is coded at a very low-level abstraction by using Xilinx primitive components such as LUT4 or MUXCY. Therefore, the design is optimized for Xilinx FPGAs and practically not portable. Leros is written in vendor agnostic VHDL and compiles unmodified for Altera and Xilinx devices.

The SpartanMC is a small microcontroller optimized for FPGA technology [4]. One interesting feature is that the instruction width *and* the data width is 18 bits. The argument is that current FPGAs contain on-chip memory blocks that are 18-bit wide (originally intended to contain parity protection). The processor is a 16 register RISC architecture with two operand instructions and is implemented in a three-stage pipeline. To avoid data forwarding within the register file, the instruction fetch and the write-back stage are split into two phases, similar to the original MIPS pipeline [5]. This decision slightly complicates the design as two phase-shifted clocks are needed. We assume that this phase splitting also limits the maximum clock frequency. As on-chip memories for register files are large, this resource is utilized by a sliding register window to speedup function calls. SpartanMC performs comparable to the 32-bit RISC processors LEON-II [2] and MicroBlaze [13] on the Dhrystone benchmark.

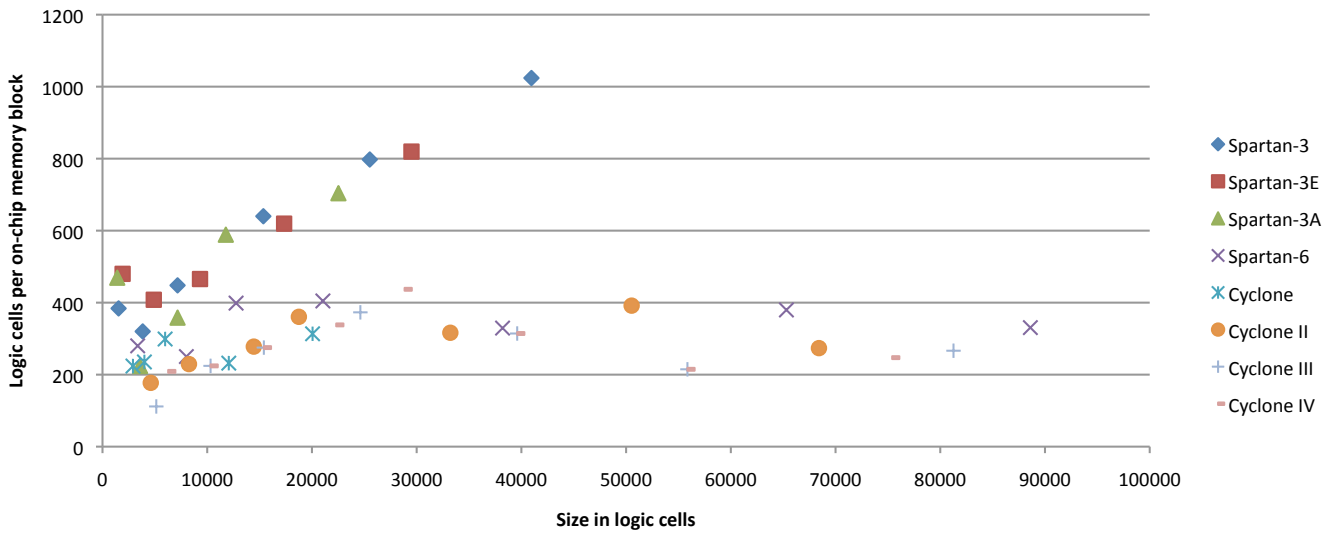

Fig. 1. Number of logic cells per on-chip memory block of low-cost FPGAs from Xilinx and Altera.

Compared to the SpartanMC, Leros is further optimized for FPGAs using fewer resources and avoiding unusual clocking of pipeline stages. Leros simplifies the access to registers in on-chip memory by implementing an accumulator architecture instead of a register architecture. Although an accumulator architecture is theoretical less efficient, the resulting maximum achievable clock frequency offsets the higher instruction count.

Altera provides a softcore, the Nios II [1], for Altera FPGAs. The Nios RISC architecture implements a 32-bit instruction set similar to the MIPS instruction set architecture. Although Nios II represents a different design point than Leros, it is interesting to note that Nios II can be customized to meet the application requirements. Three different models are available [1]: the *Fast* core is optimized for high performance; the *Standard* core is intended to balance performance and size; and the *Economy* core is optimized for smallest size. The smallest core is comparable to Leros and can be implemented in less than 700 LCs. It is a sequential implementation and each instruction takes at least 6 clock cycles. Leros is a smaller (16-bit), accumulator-based architecture and full pipelining allows to execute each instruction in a single clock cycle.

The *super small* processor [9] is optimized for low resource consumption (half of the NIOS economy version). Resources are reduced by serializing ALU operations to single bit operations. The LC consumption is comparable with Leros, but the on-chip memory consumption is not reported.

III. Leros Design Decisions

To optimize a tiny processor core we evaluate the relation of on-chip memories to logic resources on current low-cost FPGAs. Furthermore, the maximum clock frequency of Leros shall be high enough to not constraint the overall maximum clock frequency of an SoC, where Leros is used as an intelligent peripheral device. We aim for a clock frequency half of the frequency of on-chip memories with input and output registers.

A. Optimal Logic Size

The basic building blocks in current FPGAs are logic cells, on-chip memories, and DSP blocks. For a utility processor we are interested in the optimal relation between logic cell and on-chip memory consumption. We find the main difference between Xilinx and Altera FPGAs in the granularity of the on-chip memory blocks. With the Spartan-3 series Xilinx has switch from 4 Kbit memories to 16 Kbit memories. Altera uses in the low-cost devices Cyclone and Cyclone II 4 Kbit memories and switched to 8 Kbit memories with the Cyclone III device.[2]

Figure 1 shows the number of logic cells per on-chip memory block for the low-cost devices from Xilinx and Altera. We have omitted the largest device from the Spartan-6, Cyclone III, and Cyclone IV devices to keep the numbers for older devices readable. The Spartan-6 device contains 6-bit LUTs and the numbers are scaled to 4-bit LUTs. We used the Leros design to derive the scaling factor of one 6-bit LUT being worth 1.4 4-bit LUT.

The trend is that smaller FPGAs contain more memory blocks in relation to the logic, about 200 LCs per memory block. The larger devices of the Spartan-3 series increase the logic resources considerable more then the memory resources. It is interesting to note that for Altera Cyclone chips and newer Spartan chips the relation between LCs and memory blocks stays in the range of 200 to 400 LCs per memory block independent of the device size. Therefore we conclude that the sweet spot for a microcontroller in current FPGAs is around 300 LCs per on-chip memory block.

[2]Current devices contain additional parity bits that can also be used as data bits.

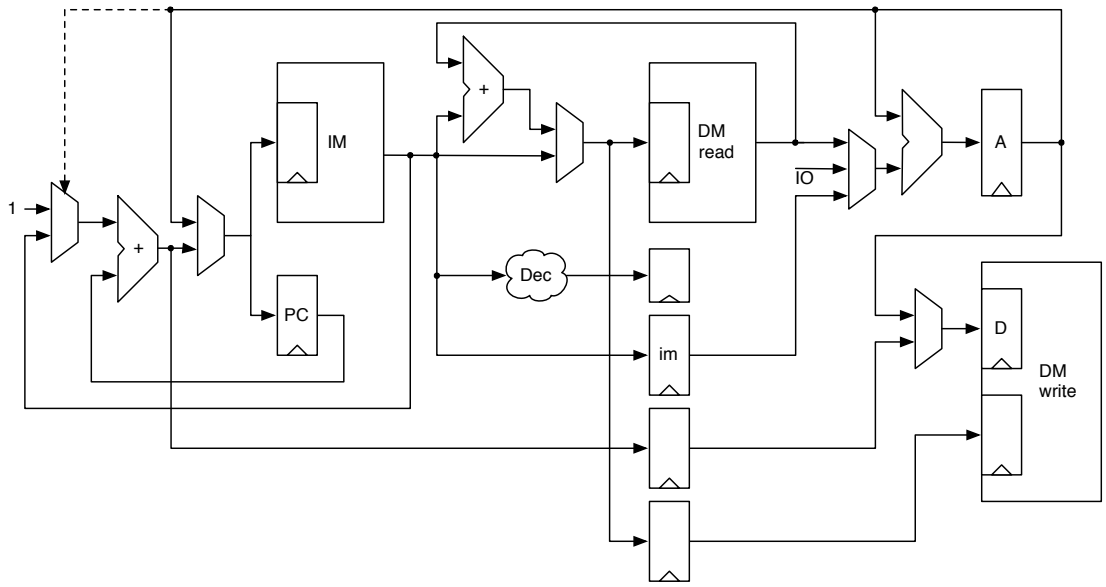

Fig. 2. Pipeline of Leros with the fetch/decode and the read/execute/memory stages.

B. Architectural Decisions

To restrict the consumption of on-chip memories they are used only for the instruction memory[3] and the data memory. Therefore, we avoid using additional on-chip memories for a register file. A standard register file would need two read and one write ports and therefore would consume another two on-chip memory blocks (to implement the two read ports).

Minimal resource consumption can be achieved by an accumulator design. Only a single dedicated register (the accumulator) is connected to the ALU output and provides one input to the ALU. To provide fast data locations, similar to a register file, the first 256 words in the on-chip data memory can be directly addressed for an ALU operation. Therefore, an ALU operation on two local variables takes two cycles to execute and another cycle if the result needs to written back to a local variable. This might sound expensive, but compared to the PicoBlaze and Nios II it is not so expensive. PicoBlaze needs two clock cycles for each instruction and the economy version of Nios II 6 clock cycles.

C. Instruction Set

Leros is a 16-bit architecture with 16-bit data and instructions. In an accumulator design the addresses of one source operand and the destination are implicit. Therefore, only one operand (address) needs to be encoded in the instruction. Furthermore, this relaxed instruction encoding allows for 8-bit immediate values in the instruction. The common ALU operations are supported with one operand from the register area (256 words) in the data memory or with an 8-bit immediate value. The data memory can be accessed via indirect loads and stores. I/O ports are accessed with dedicated instructions and an 8-bit address.

Conditional and unconditional branches use 8 bit relative offsets. For longer jump destinations, function calls, return from a function call, and computed jump destinations an indirect jump via the address in the accumulator is supported (jump-and-link). The jump-and-link instruction saves the program counter into a register.

[3]Very small programs can even be implemented using LCs for the instruction memory. Quartus automatically instantiates either an on-chip memory or LC based instruction memory from the same VHDL source.

IV. Leros Implementation

Based on the former described design decisions, Leros is implemented in a two-stage pipeline with following visible architectural state: the program counter (PC), the accumulator register (A), the instruction memory (IM), and the data memory (DM).

A. The Pipeline

Figure 2 shows the pipeline of Leros. It shows the main data path, the pipeline registers, and the on-chip memories. The DM is shown twice as it is read in one pipeline stage and written in a different one. Register A is the accumulator and PC the program counter.

Leros is implemented with two pipeline stages. The first stage fetches an instruction from the IM and decodes the instruction. The second stage reads an operand from the DM and executes. In both stages the combinational logic is fed by the asynchronous output of the memory. According to our design decisions, the pipeline stage is balanced when the combinational logic (plus the routing) delay is as long as the access time of the on-chip memory.

As the input registers of on-chip memories cannot be read, the PC is duplicated. The address of the IM and the PC is either an increment of the PC by 1, an addition with a constant from the instruction (relative branch), or the content of A (indirect jump).

The read and write address of the DM is either a constant from the instruction (for the on-chip registers) or an indirection via the DM plus an offset (for loads and stores). The write data for the DM is either A for store instructions or the PC for a jump-and-link instruction to save the PC in a register.

A memory load and store uses a based register and an offset for the address. As the data memory is shared for registers and general data, load and stores are implemented by two instructions. With the first instruction the address for the register, which is used as base address for the load/store, is sent to the DM. The following instruction uses the value of the DM (the register content) and adds an offset, which is part of that instruction, to form the effective address. The data word to be written is provided by A; the result of a load is stored into A. This split of the indirect memory access into two instructions costs an additional cycle, but allows efficient reuse of the DM for registers and general data.

For on-chip memories with independent read and write port the question arises what happens on a concurrent write to and read from the same address in the same cycle. There are three options: (1) read the newly written value, (2) read the old value, or (3) undefined. The actual behavior depends on the FPGA family and is sometimes configurable. A safe way would be to forward the written result to the read output, which is in the actual version of Quartus automatically inferred (depending on the VHDL style used to describe memory). However, for an accumulator machine there is no benefit in forwarding this memory access. The last value written to DM is still in register A and can just be reused when needed by the next instruction.

B. I/O Interface

I/O read data is fed into the pipeline via the ALU. The I/O write data is the content of A. The address for the I/O instruction is part of the instruction.

The interface to I/O devices is similar to the PicoBlaze design and consists of an 8-bit I/O address, a read and write signal, and 16-bit input and output data signals. For the write instructions the address, the data, and the write signal are valid for a single cycle. For the read instruction the I/O device needs to deliver the result in the same cycle as the address and the read signal are valid. No busy or wait signals are supported. For devices with a longer access time software based polling needs to be implemented. The argument for this I/O interface is simplicity and the fact that most I/O devices already support single cycle access.

C. Extensions

One application area of Leros is a processor to implement *intelligent* I/O devices. Such devices often need to generate exactly timed outputs. To simplify this timing generation a so-called *deadline* instruction can be used [6], [11]. A deadline instruction is a programmable timer with combined with the facility in the pipeline to wait cycle accurate (or in the range of a few cycles) until the timeout. With such an instruction a PWM based DA converter or a serial interface (UART) can be implemented completely in software.

For experiments with a Leros based many-core system it shall be possible to dynamically change the content of the instruction memory and the data memory. To access the shared off-chip memory a memory controller and an arbiter needs to be attached to the I/O interface. We will reuse the available components from a Java chip-multiprocessor [8].

To transfer instructions from the shared main memory into the instruction memory of Leros, the on-chip instruction memory needs an additional write port, which is already available (instruction fetch uses only a read port). Furthermore an instruction needs to be defined to write the content of A into the instruction memory.

We plan to use Leros for experiments with a ring-based network-on-chip [3]. With the small size of a processor node we can perform real experiments in an FPGA with a huge number of processing nodes (e.g., 100+ in the FPGA of the Altera DE2-70 board).

D. Software Tools

We started with a simple assembler written in Java with some reuse of code from the JOP project [10]. Later we adapted the lexer and grammar tool ANTLR [7] for the assembler. Using the parser generator ANTLR might sound like an overkill for the implementation of an assembler. However, the simplicity of defining a grammar enables the implementation of a higher level assembler, which can simplify assembler programming. Furthermore, we started to retarget the muvium Java compiler[4] for Leros. Muvium is optimized to compile simplified Java for resource-constrained microcontrollers, such as Microchip processors. Therefore, this compiler is an ideal tool for Leros.

V. Evaluation

We have implemented Leros on several different FPGA boards with Altera and Xilinx FPGAs. The VHDL code of Leros is highly portable. The only changes needed for a port are the pin definitions for the board and a device specific PLL component. For the evaluation we used the free versions of Altera Quartus 10.1 and Xilinx ISE 12.4.

Table I shows synthesis results of Leros for different FPGA families. The last column shows the maximum clock frequency of an on-chip memory of that FPGA family. We have selected the fastest speed grade for all devices for this table. To derive the maximum clock frequencies for the Leros design and the on-chip memory, the designs use a PLL and have been over-constraint with a high clock frequency. From the table we see that we have achieved our goals to build a small processor with a maximum clock frequency in the range of half the on-chip memory clock frequency. In the benchmarked configuration Leros consumes only a single on-chip memory. This can be explained by the small test program (a LED blinking at 0.5 Hz, implemented by 36 instructions) where the ROM is implemented in logic cells instead of an on-chip memory.

[4] http://www.muvium.com/

TABLE I
SYNTHESIS RESULTS OF LEROS FOR DIFFERENT FPGAS

FPGA	Logic (LC)	Memory (blocks)	Fmax (MHz)	BRAM Fmax (MHz)
Cyclon	195	1	132	256
Cyclon II	190	1	146	235
Cyclon III	188	1	150	315
Cyclon IV	189	1	160	315
Spartan-3E	188	1	129	297
Spartan-6	112	1	182	320

The Spartan-6 device contains 6-bit LUTs and therefore the LC count is lower. Furthermore, the available 16 Kbit BRAM is logically split into two 8 Kbit memories for the instruction and data memory.

We compare Leros with PicoBlaze on the Nexys2 FPGA board, which contains a Spartan XC3S500E-4. Both processors contain an interface to the LEDs and buttons and use a PLL (DCM) to clock the processor at 100 MHz. To report the maximum clock frequency we over-constrained the design by generating a 150 MHz clock with the PLL. For the comparison of Leros with PicoBlaze we wrote a minimalistic embedded application: a light control. The light control uses a LED on the FPGA board as output and can be switched with one button between three states (off, dimmed, on). Dimming of the LED is generated by pulse width modulation (PWM) in software.

Table II shows the resource utilization and maximum clock frequency of the design for a Spartan-3E device.[5] Leros is comparable in resource consumption and maximum clock frequency with PicoBlaze. However, Leros implements a 16-bit data path and can execute an instruction in a single cycle, whereas PicoBlaze is a 8-bit processor and needs 2 clock cycles for each instruction. In contrast to Leros, SpartanMC implements a register machine and needs therefore more resources. Compared to PicoBlaze, SpartanMC executes basic operations in a single cycle. The lower clock frequency of SpartanMC is due to the usage of two phase-shifted clocks for the sub-dividing of two pipeline stages into two phases.

The implementation of Leros on several FPGA boards and the comparison with PicoBlaze shows that we have achieved our design goal of a small, fully pipelined processor, which can be clocked at a reasonable frequency. Except for the PLL component, all VHDL sources are vendor agnostic. This experiments also shows that usage of vendor specific components, as done in the PicoBlaze design, have minimal, or perhaps no, impact on the achievable size and performance. Current synthesize tools are mature enough to infer a good hardware implementation from carefully written VHDL code.

VI. CONCLUSION

This paper presents the design and implementation of a tiny microcontoller for FPGAs. Leros is a pipelined processor, optimized to balance the resource consumption between logic cells and on-chip memory. The 16-bit processor Leros consumes less than 200 logic cells and 1–2 on-chip memories. To achieve optimal timing, the combinational logic and routing resource delays are designed to be equal to the access time of on-chip memories. Therefore, the maximum clock frequency of Leros is in the range of half of the maximum clock frequency of on-chip memories. This tiny processor is an ideal choice for utility functions, such as implementing an intelligent peripheral component for an SoC. Due to the small size it can also be used for research on many-core architectures in medium sized FPGAs.

[5]The maximum clock frequency is slightly lower as in Table I, as the FPGA on the Nexys2 board is a slower speed grade.

TABLE II
COMPARISON OF LEROS WITH PICOBLAZE AND SPARTANMC ON A SPARTAN XC3S500E-4

Processor	Logic (LC)	Memory (blocks)	Fmax (MHz)
Leros	188	1	115
PicoBlaze	177	1	117
SpartanMC	1271	3	50

SOURCE ACCESS

The Leros design is open source and available from https://github.com/schoeberl/leros. The distribution currently supports following FPGA boards directly: Altera DE2-70, Nexys2, and Cycore. As the interface to the off-chip world is minimal (clock, buttons, and LEDs), a port to a different FPGA board is trivial.

REFERENCES

[1] Altera Corporation. Nios II processor reference handbook, 2010. Version 10.1.

[2] J. Gaisler. A portable and fault-tolerant microprocessor based on the SPARC v8 architecture. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, page 409, Washington, DC, USA, 2002. IEEE Computer Society.

[3] F. Gruian and M. Schoeberl. NoC-based CSP support for a Java chip multiprocessor. In *Proceedings of the 28th Norchip Conference*, Tampere, Finland, November 2010. IEEE Computer Society.

[4] G. Hempel and C. Hochberger. A resource optimized processor core for FPGA based SoCs. In H. Kubatova, editor, *Proceedings of the 10th Euromicro Conference on Digital System Design (DSD 2007)*, pages 51–58. IEEE, 2007.

[5] J. L. Hennessy. VLSI processor architecture. *Computers, IEEE Transactions on*, C-33(12):1221–1246, Dec. 1984.

[6] N. J. H. Ip and S. A. Edwards. A processor extension for cycle-accurate real-time software. In *IFIP International Conference on Embedded and Ubiquitous Computing (EUC)*, volume LNCS 4096, pages 449–458, Seoul, Korea, 2006. Springer.

[7] T. J. Parr and R. W. Quong. Antlr: A predicated-ll(k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.

[8] C. Pitter and M. Schoeberl. A real-time Java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst.*, 10(1):9:1–34, 2010.

[9] J. Robinson, S. Vafaee, J. Scobbie, M. Ritche, and J. Rose. The supersmall soft processor. In *Programmable Logic Conference (SPL), 2010 VI Southern*, pages 3 –8, march 2010.

[10] M. Schoeberl. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, 54/1–2:265–286, 2008.

[11] M. Schoeberl, H. D. Patel, and E. A. Lee. Fun with a deadline instruction. Technical Report UCB/EECS-2009-149, EECS Department, University of California, Berkeley, October 2009.

[12] Xilinx. PicoBlaze 8-bit embedded microcontroller user guide, 2010.

[13] Xilinx Inc. MicroBlaze processor reference guide, 2008. Version 9.0.