IEICE TRANS. INF. & SYST., VOL.E98–D, NO.12 DECEMBER 2015

PAPER Special Section on Parallel and Distributed Computing and Networking

Ultrasmall: A Tiny Soft Processor Architecture with Multi-Bit Serial Datapaths for FPGAs

Shinya TAKAMAEDA-YAMAZAKI[†a)], ***Member***, **Hiroshi NAKATSUKA**[††b)], **Yuichiro TANAKA**[††c)], ***Nonmembers***, ***and*** **Kenji KISE**[††d)], ***Member***

SUMMARY Soft processors are widely used in FPGA-based embedded computing systems. For such purposes, efficiency in resource utilization is as important as high performance. This paper proposes **Ultrasmall**, a new soft processor architecture for FPGAs. Ultrasmall supports a subset of the MIPS-I instruction set architecture and employs an area efficient microarchitecture to reduce the use of FPGA resources. While supporting the original 32-bit ISA, Ultrasmall uses a 2-bit serial ALU for all of its operations. This approach significantly reduces the resource utilization instead of increasing the performance overheads. In addition to these device-independent optimizations, we applied several device-dependent optimizations for Xilinx Spartan-3E FPGAs using 4-input lookup tables (LUTs). Optimizations using specific primitives aggressively reduce the number of occupied slices. Our evaluation result shows that Ultrasmall occupies only 84% of the previous small soft processor. In addition to the utilized resource reduction, Ultrasmall achieves 2.9 times higher performance than the previous approach.

key words: *soft processor, processor architecture, FPGA*

1. Introduction

FPGA is an emerging hardware to accelerate large-scale computing in various applications, such as high frequency trading, web search, and database. In both such modern and large-scale FPGA systems, and embedded systems, soft processors have become a common and important component. In contrast to general hard macro processors, soft processors are realized by utilizing on-chip fabrics on FPGAs, such as lookup tables (LUTs), registers, and memory blocks. To achieve adequate efficiency in performance and energy consumption, FPGA-based computing systems employ optimized dataflow pipelines using inherent FPGA resources.

A soft processor is often used to handle the accessorial and non-performance-critical parts of the target application. Since it is easier to design and implement an application as software than to design a dedicated hardware, employing a soft processor improves the development efficiency of FPGA-based systems.

FPGA vendors provide such general soft processors as Xilinx MicroBlaze [1] and Altera Nios [2] on their EDA tools. These soft processors have a large capability to execute various kinds of applications. Unfortunately, these soft processors consume a large amount of FPGA resources and energy. Such inefficiency in resource utilization and energy is critical to embedded systems that use small low-end FPGAs. To overcome this inefficiency, various small soft processors have been proposed by FPGA vendors, academia, and open source communities. To reduce the amount of occupied FPGA resources, previous researches on soft processor have focused on two approaches.

The first approach is to employ a narrow instruction set architecture (ISA), such as an 8- or 16-bit ISA. Since the general soft processors provided by the vendors usually use a 32-bit ISA, they require wide datapaths and control units. In contrast, datapaths and control units for narrow ISAs can be implemented by utilizing many fewer resources. However, they cannot directly execute the original 32-bit instructions.

The other approach employs bit serial structures instead of the original bit parallel structures. Since a bit serial unit can generate a partial result of an instruction, a unique bit serial unit is used multiple times for generating a complete result. For example, to execute a 32-bit instruction on a 1-bit serial unit, the unit is repeatedly used 32 times. Therefore, the execution stage takes 32 or more clock cycles. By employing a narrow datapath, the amount of occupied FPGA resources is efficiently reduced, while the original wide ISA is supported. However, the maximum performance is obviously reduced by the bit serial structure.

In this paper, we propose **Ultrasmall**, which is a tiny soft processor architecture that supports a subset of the MIPS-I instruction set architecture. For higher resource efficiency, Ultrasmall uses a 2-bit serial ALU for all operations instead of the original 32-bit ALU. By employing a 32-bit MIPS ISA, many existing development environments are available, such as compilers, debuggers, and manuals. Therefore, designers can develop applications using Ultrasmall with less effort.

Ultrasmall is based on Supersmall [3], a previous tiny soft processor of the MIPS ISA with a 1-bit serial ALU. Supersmall has a multi-cycle structure, not a pipelined structure, for low resource utilization. Therefore, most instructions require 32 or more clock cycles to complete execution. In this work, we found that employing a 2-bit serial datapath improves the efficiency in both the performance

Manuscript received January 9, 2015.
Manuscript revised May 18, 2015.
Manuscript publicized September 15, 2015.

[†]The author is with Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.
[††]The authors are with Tokyo Institute of Technology, Tokyo, 152–8552 Japan.
a) E-mail: shinya@is.naist.jp
b) E-mail: nakatsuka@arch.cs.titech.ac.jp
c) E-mail: tanaka@arch.cs.titech.ac.jp
d) E-mail: kise@cs.titech.ac.jp

DOI: 10.1587/transinf.2015PAP0022

and resource utilization more than the original 1-bit serial datapath. Additionally, Ultrasmall employs several device-specific optimizations using primitives for Spartan-3E FPGAs with 4-input LUTs to further reduce the required hardware resources.

This paper is based on our previous works [4], [5], where we proposed the baseline architecture and showed preliminary evaluation results. The following are this paper's main supplementary contributions: the first contribution is to present a more detailed architecture and implementation of Ultrasmall. As a whole, we have described overall sections in a careful manner. Especially, we introduce the finite state machine (FSM) structure and FSM sequence of each instruction. Then, we present how to implement a byte- and half-word load/store instructions on Xilinx Spartan-3E FPGAs without byte-enable access capability. The second contribution is a detailed discussion why Ultrasmall achieves the better resource utilization and why its maximum clock frequency is lower than the others.

The rest of this paper is presented as follows. In Sect. 2, we discuss previous related works. In Sect. 3, we describe the motivation of our research and Ultrasmall's architecture. Section 4 compares Ultrasmall's evaluation results in terms of performance and hardware resource utilization to previous work. Finally, a summary is provided in Sect. 5.

2. Related Work

2.1 Previous Soft Processors

FPGA vendors provide some general soft processors. MicroBlaze [1] is one of the most famous 32-bit soft processors provided by Xilinx, which has various configuration options in pipeline depth, floating point unit, cache memory, memory management, and bus interface. By using XPS [6] or Vivado [7], designers can make a customized soft processor with adequate performance and resource utilization for a specific application. Nios [2], which is a widely used 32-bit soft processor in Altera FPGAs, also has various architectural options. Nios is also customizable on the EDA tools provided by the vendor. These soft processors have modern on-chip interconnection interfaces, such as AMBA AXI4 and Altera Avalon. Designers can easily integrate various application IP-cores with soft processors, which can access and control the IP-core through the interconnection.

Unfortunately, these advanced features require a certain amount of overhead in hardware resource utilization. The increase of required hardware resources is especially critical for small, low-end FPGAs. Xilinx also provides MicroBlaze MCS (Micro Controller System) [8], a simple 32-bit soft processor without cache memory and other advanced features that are possessed by the original MicroBlaze. MicroBlaze MCS has no on-chip interconnection interfaces that connect with other IP-cores using common bus interfaces, such as AMBA AXI4. These features reduce the required amount of hardware resources, compared to the original MicroBlaze.

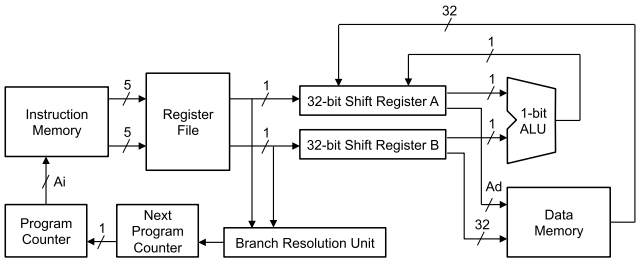


Fig. 1 Supersmall architecture

Using another approach, Xilinx provides PicoBlaze [9]. In contrast to standard 32-bit soft processors, PicoBlaze employs 8-bit architecture for its small footprint. For further footprint reduction, it has device-specific optimization that directly uses such primitive components as LUT4 and MUXF5. As a result, PicoBlaze consumes just 96 slices on Spartan-3 FPGAs. However, the size of the instruction memory is limited by the architecture to 1024 instructions. The size of the data memory is also limited to 64 bytes. Since 8-bit architecture is used, computations of multi-byte values take long latencies.

Not only by FPGA vendors, some soft processor architectures have been developed by the research and open-source community. Leros [10] is a small soft processor that is designed as an accumulator machine for reducing resource usage. It is implemented using 188 slices on a Spartan-3E FPGA, which supports an original 16-bit ISA and has a similar design concept as PicoBlaze. Therefore, when processing multi-byte data, Leros requires longer operation cycles than the other 32-bit soft processors. ZPU [11] is a small 32-bit soft processor that reduces FPGA resource usage with a stack machine architecture. While ZPU uses its original 32-bit ISA, a customized GCC toolchain is available.

In some projects, new ISAs are proposed with RTL implementations. OpenRISC [12] is an open-sourced ISA whose Verilog HDL implementation as a soft processor is released on OpenCores [13], which has a memory management capability, an operating system support, and a wishbone interconnection interface [14]. RISV-V [15] is an open-sourced ISA for more efficiency in performance, energy efficiency, and code size. The RTL implementation of RISC-V is written in Chisel [16], a domain specific language in Scala. The implementation also has operating system support. Since these soft processor implementations require additional hardware resources for memory management and interconnection supports, they are not suitable for low-end small FPGAs.

2.2 Supersmall Soft Processor

The purpose of Supersmall [3], a small soft processor that supports a subset of the MIPS-I ISA, is to implement a 32-bit microprocessor within a limited and small hardware amount of FPGAs. Figure 1 shows Supersmall's architecture. It employs non-pipelined, multi-cycle architecture

for reducing the required hardware resources and has some standard elements, an instruction memory, a register file, and a data memory. These memory components are implemented using inherent on-chip memory blocks that standard FPGAs have, such as block RAMs. Supersmall's key feature is employing 1-bit serial ALU architecture. It has two 32-bit shift registers, A and B, to pass data from the register file to the ALU.

For every clock cycle, the shift registers 1-bit output from 32-bit operand data. The 1-bit serial ALU receives the serial data from the shift registers and calculates the partial result of an instruction for every clock cycle. Therefore, it takes 32 clock cycles to perform a 32-bit operation; serialization obviously degrades the performance. However, since the amount of wires is dramatically reduced using serial lines between the register file and the ALU, Supersmall requires fewer hardware resources than conventional 32-bit soft processors. To the best of our knowledge, Supersmall is the smallest 32-bit soft processor implementation for FPGAs, except for our proposal.

In order to access to the data memory, an address value and data value are passed from the shift register A and B, respectively. The path between the shift register B and the data memory is used for data, and its width is 32-bit. The path between the shift register A and the data memory is used for address, and the its width depends on the capacity of the data memory. In the figure, the width is represented as *Ad*. As well as the address path for the data memory, the path between the program counter and the instruction memory depends on the capacity of the instruction memory. In the figure, the width is represented as *Ai*.

3. Ultrasmall

We propose Ultrasmall, a new tiny soft processor architecture with a 2-bit datapath and an ALU for FPGAs. Ultrasmall is based on Supersmall, the tiny 32-bit soft processor with a 1-bit serial ALU, as described above. Ultrasmall has two goals: (1) to reduce the occupied hardware resources on an FPGA and (2) to increase performance just by architectural improvement. In this section, we describe why we chose a 2-bit width for the datapath and ALU and show its overall architecture and implementation.

3.1 Motivation: Are 1-bit Datapath and ALU Optimal?

The original Supersmall soft processor employs a 1-bit serial datapath and ALU architecture to reduce hardware usage. Even though this approach does reduce the occupied resources on FPGAs, it also degrades the computing performance by increasing the cycle per instruction (CPI) ratio. Since the increase of the CPI also increases the complexity of the finite state machine on the processor, it requires additional hardware resources. To achieve higher performance, the CPI rate should be as small as possible.

Most FPGAs have island-style architecture with many logic and switch blocks. Each logic block has LUTs to realize combinational circuits and flip flops (FFs) to realize sequential circuits. For instance, a logic element (called *a slice*) on an Xilinx Spartan-3E FPGA has 2 LUTs [17]. The logic synthesizer of EDA tools determines which LUT is consumed based on the RTL design. Not all of the LUTs on each logic block are usually utilized due to the patterns of the target logic. Finally, the amount of occupied resources is increased if the target logic is not suitable for logic blocks on FPGAs. To avoid reducing the number of occupied LUTs but to reduce the number of occupied logic blocks for the final resource reduction, constructing logic-block-friendly logics is important; the LUTs in each logic slice must be efficiently utilized.

Table 1 Implementation result of serial ALU on a Spartan-3E FPGA when changing ALU bit width

ALU bit width	1-bit	2-bit	4-bit	8-bit	16-bit
Reg	2	2	2	2	2
LUT	8	9	12	20	36
Slice	4	5	7	12	20

Table 2 Implementation result of Supersmall with 1- and 2-bit serial ALU architectures

Datapath width	1-bit	2-bit
Reg	164	141
LUT	253	244
Slice	164	163

To achieve higher efficiency of resource utilization for soft processors with serial datapaths, we estimate the impact of datapath width on soft processors and evaluate the resource usage of a serial ALU that has a different datapath width. The experimental setup is the same as the one described in Sect. 4. Table 1 shows the resource usage of each ALU setup. The result shows that the 2-bit serial ALU is very efficient because it requires only one additional slice compared to the baseline 1-bit ALU; the 2-bit serial ALU can compute a result at twice the speed of the 1-bit serial ALU. In contrast, the other ALUs, 4-bit, 8-bit, and 16-bit, require more slices. Additionally, using the 2-bit ALU simplifies the finite state machine because the elapsed clock cycles are reduced for each instruction. Using the 2-bit serial ALU improves performance and maintains small hardware occupation.

According to the above estimation, employing the 2-bit serial ALU improves the efficiency of the performance and resource utilization. Next we implemented and evaluated two setups of serial soft processors. The first is a soft processor with the 1-bit serial datapath and ALU. The original Supersmall is designed for Altera's FPGAs. We developed its transplant implementation for Xilinx's FPGAs. The other has a 2-bit serial datapath and ALU. Table 2 shows the resource usage of each processor setup. The amount of required hardware resources for the entire processor is reduced by employing the 2-bit datapath and ALU, compared to the original 1-bit serial datapath and ALU. We chose the 2-bit datapath and ALU architecture for the baseline of our proposal.

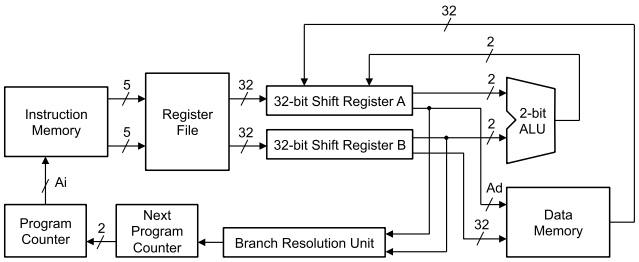


Fig. 2 Ultrasmall architecture

3.2 Architecture

Figure 2 shows the architecture of Ultrasmall, which employs a non-pipelined, multi-cycle architecture with the 2-bit serial datapath and ALU. Ultrasmall has instruction memory, a register file, and data memory, like Supersmall. These memory components are implemented using on-chip memory blocks, such as block RAM. Two shift registers also pass operand data from the register file to the ALU. As well as Supersmall, the bit width of the path between the shift register A and the data memory depends on the capacity of the data memory, and it is represented as *Ad*. The bit width of the path between the program counter and the instruction memory also depends on the capacity of the instruction memory, and it is represented as *Ai*. In contrast to the original Supersmall, Ultrasmall has advanced datapaths for optimizing both the performance and the resource occupation. There are two key enhancements. The first is employing the **2-bit serial datapath and ALU** for the execution unit, while the original Supersmall has a 1-bit serial datapath to reduce the amount of occupied resources. The other is employing a 32-bit parallel register file that supplies operand data to the shift registers in parallel. The output port of the register file in Supersmall is 1-bit. In contrast, Ultrasmall has a standard register file structure as well as the other soft processors.

Since Ultrasmall is built on MIPS-I ISA, the operand IDs are directly specified as a bit vector in an instruction. At first, two 32-bit operands are fetched from the register file using the bit vector in an instruction. These operand data are written to the shift registers, A and B, within a 1 clock cycle. These shift registers have 2-bit wide output ports. After that, the 2-bit data from the LSB side in the shift registers are sequentially supplied to the 2-bit serial ALU, which builds a complete execution result of an instruction by consuming the given 2-bit partial data. The ALU calculates a temporal consequence of the instruction with the given data. The temporal result into the MSB side of shift register A is done sequentially. Since the ALU takes 1 clock cycle per partial operation with two inputs of the 2-bit partial operand data, ALU operation with all 32-bit operands requires 16 clock cycles for execution.

Since Ultrasmall is based on MIPS-I ISA, which is a major and simple 32-bit ISA, many existing environments are available as is. Programmers can use existing eco-systems, such as the compiler, for implementing software for Ultrasmall. Note that Ultrasmall supports almost all of the elementary instructions defined in the MIPS-I ISA. However, some instructions, such as privileged instructions, multiply and divide instructions, and unaligned load and store instructions are not supported in the current implementation.

3.3 State Machine

As described above, Ultrasmall has a multi-cycle, non-pipelined architecture. Every instruction (except shift instructions) requires a fixed clock cycle for its execution. The number of elapsed cycles to execute a shift instruction depends on the amount of shifts. Figure 3 illustrates a finite state machine of Ultrasmall. Each square in the figure represents a state. The first line in a state represents the state's name. The second line in a state represents the number of elapsed clock cycles. Figure 4 shows the list of state machine sequences with the number of elapsed clock cycles for each instruction. For example, the ADDU (Add Unsigned) instruction takes 20 cycles to complete, including instruction fetch, execution, and writeback steps.

Every instruction takes two common states: START_1 and START_2. At these states, instruction fetch, register fetch, and instruction decode are processed in 3 clock cycles. Then the branch instructions take two states: BRANCH_1 and BRANCH_2. At BRANCH_1, the condition of the conditional branch is determined. At state BRANCH_2, the value of the next program counter is written in the program counter (PC) register. Since each of these two states takes 16 clock cycles, they require 35 (= 2 + 1 + 16 + 16) clock cycles to execute a branch instruction. In contrast, since the jump instructions (*j* and *jr*) only require the state to writeback the next PC at the JUMP_1 state, they require 19 (= 2 + 1 + 16) clock cycles. The jump-and-link, arithmetic, and logical instructions require 16 clock cycles for the third state and 1 clock cycle for the writeback step from the shift register to the register file. Therefore they require a total of 20 (= 2 + 1 + 16 + 1) clock cycles.

The store-word instruction requires 16 clock cycles for calculating its destination address and 1 cycle for writing back its stored data in the data memory. Therefore, it takes a total of 20 (= 2 + 1 + 16 + 1) clock cycles. The load-word instruction also requires 16 clock cycles for calculating its destination address and 2 clock cycles for getting the data from the data memory and writing them in the register file. Therefore, it takes 22 (= 2 + 1+ 16 + 2 + 1) clock cycles.

3.4 Shift Operation

Both Supersmall and Ultrasmall handle shift operations by using their shift registers. The number of elapsed clock cycles depends on the shift amount. In Supersmall, the shift operations can be naturally processed by using the bit-serial shift registers. The target data saved in the shift register are rotated and the unnecessary bits are discarded bit by bit. In

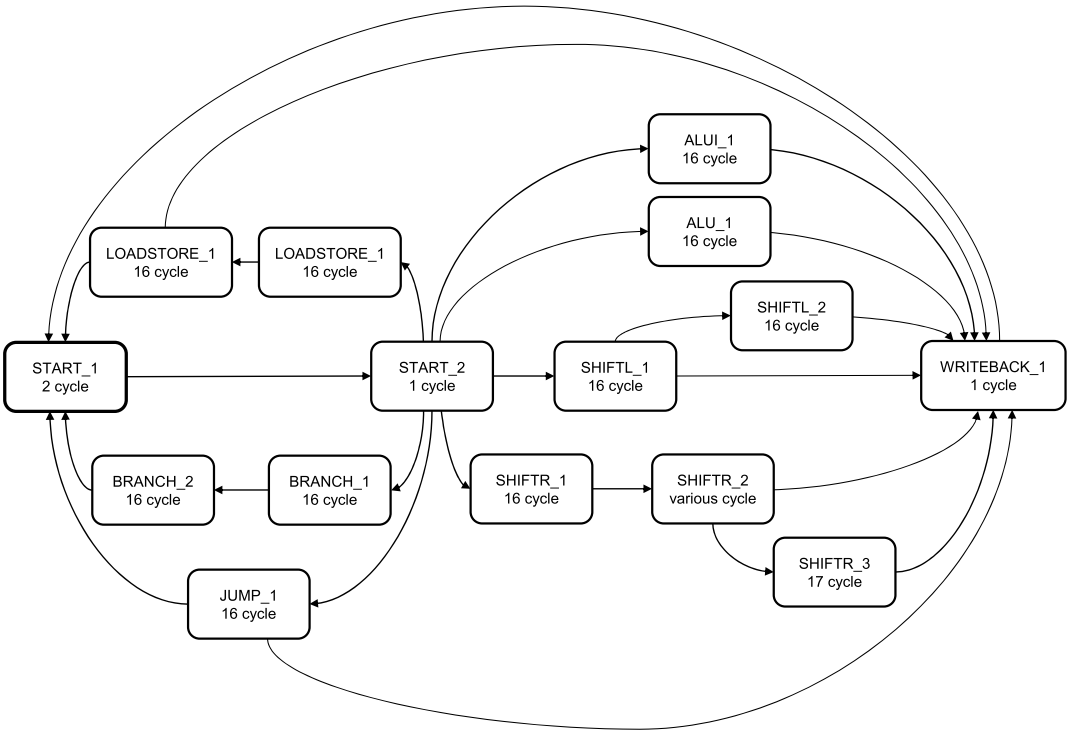


Fig. 3 Finite state machine of Ultrasmall

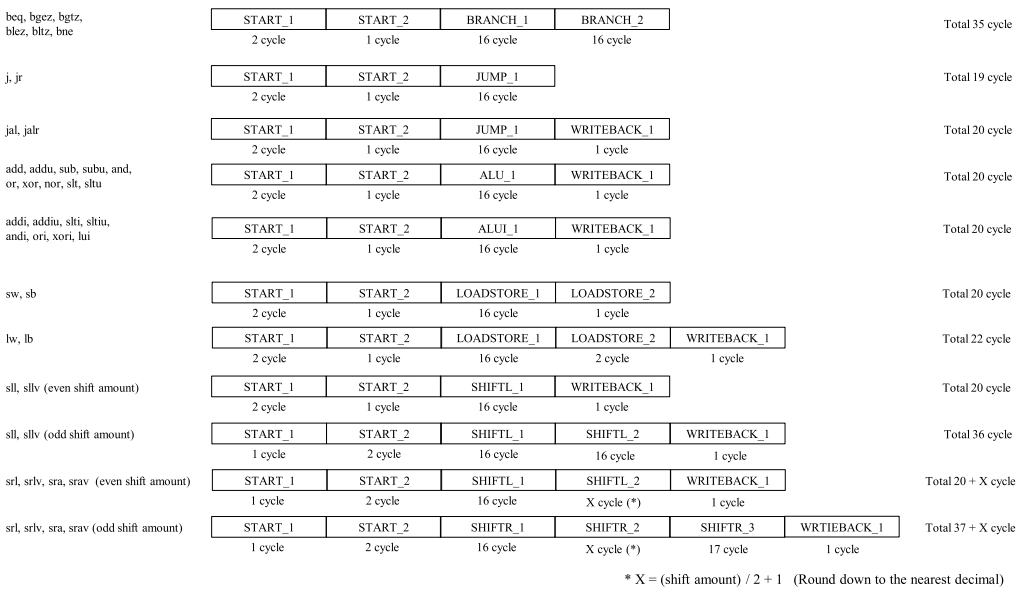
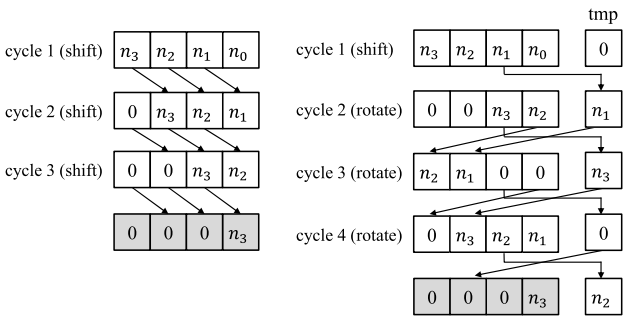


Fig. 4 State machine sequences

contrast, since Ultrasmall has 2-bit shift registers, an odd-numbered shift amount presents some challenges to deal with the shift operation. For even-numbered shift amounts, the unnecessary bits are removed 2-bit by 2-bit from the shift register. In contrast, since odd-numbered shift amounts finally require a 1-bit shift, the shift register should originally support both the 2- and 1-bit shifts, which increases the hardware complexity.

To overcome this problem, we designed an area efficient shift register structure for the 2-bit serial datapath. We assume a shift register with 4-bit entries to simplify the explanation while the actual shift registers in Ultrasmall have 32-bit entries.

Figure 5 (a) shows the behavior of the 3-bit shift-right operations on the 1-bit datapath of Supersmall. The 3-bit shift-right operation is performed in 3 clock cycles. In general, an n-bit shift operation takes n clock cycles to complete on the 1-bit datapath of Supersmall. For the 2-bit datapath on Ultrasmall, 2-bit shift operation is completed in a 1 clock cycle. Therefore, an n-bit shift operation can be done in $n/2$



(a) 3-bit right shift on the 1-bit wide datapath

(b) 3-bit right shift on the 2-bit wide datapath

Fig. 5 3-bit right shift on: (a) 1-bit wide datapath and (b) 2-bit wide datapath

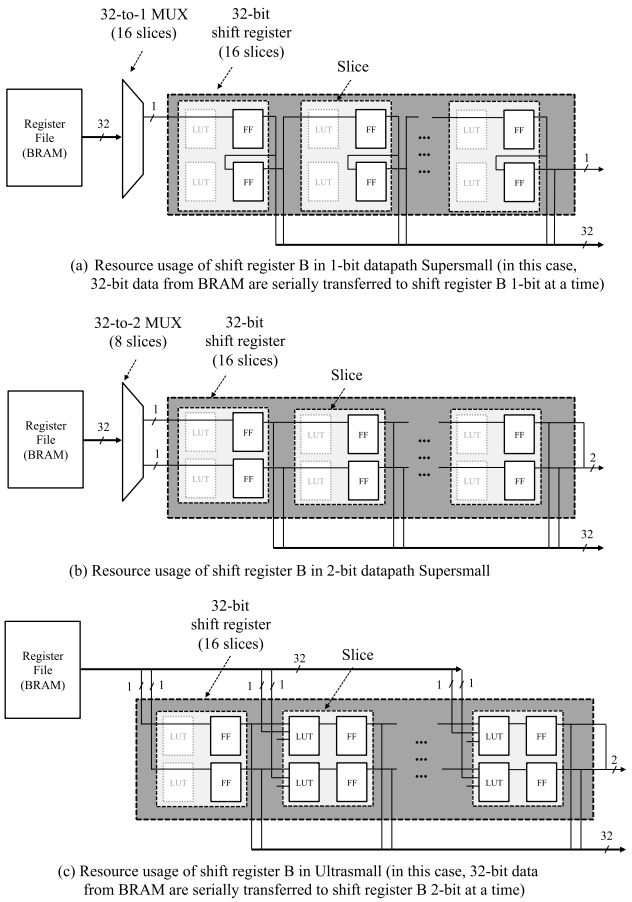
clock cycles where n is an even number. However, it is difficult to handle an n-bit shift operation where n is an odd number.

Figure 5 (b) illustrates our implementation for efficiently supporting odd-numbered shift amounts when the shift amount is 3 again. A small but crucial modification is using the tmp register. The second bit from the LSB is stored on the tmp register at the next clock cycle. In the example, n_1 is stored on the tmp register at the first clock cycle. For $(2n + 1)$-bit shift operation, several clock cycles are required to do several 2-bit shift operations. There is a residual bit, because the shift amount of the operation is odd-numbered. Therefore, it takes extra clock cycles for 2-bit *rotations* using the tmp register. In the example, *cycles 2, 3, and 4* belong to the rotation processes. On the actual Ultrasmall, since the shift registers have 32-bit entries, 17 extra clock cycles are required to complete the 2-bit rotations using the tmp register after n clock cycles for the 2-bit shift operations.

3.5 Optimization for Spartan-3E FPGAs

For further hardware resource reduction of Ultrasmall, we applied device-dependent but aggressive optimizations employing the characteristics of inherent FPGA resources. We chose Xilinx Spartan-3E FPGAs as the target device. To show the optimization details, we first describe the architecture of the slice and BRAM on Spartan-3E FPGAs. A slice is a logic element with LUTs, FFs, carry-chain logics, and multiplexers. LUTs are used to implement combinational circuits. A slice on Xilinx Spartan-3E FPGA has 2 4-input LUTs and 2 FFs [17]. If the LUTs in each slice are efficiently utilized, the number of occupied slices will be reduced. Therefore, designing slice friendly logic is important.

In addition, an FPGA has an on-chip SRAM block called a block RAM (BRAM), which is usually used for memory systems, such as an instruction memory and a register file. A block RAM entry on Spartan-3E FPGAs has 18K-bit capacity and 2 synchronous read/write ports (dual-ports). A block RAM has two output registers with a clock



(a) Resource usage of shift register B in 1-bit datapath Supersmall (in this case, 32-bit data from BRAM are serially transferred to shift register B 1-bit at a time)

(b) Resource usage of shift register B in 2-bit datapath Supersmall

(c) Resource usage of shift register B in Ultrasmall (in this case, 32-bit data from BRAM are serially transferred to shift register B 2-bit at a time)

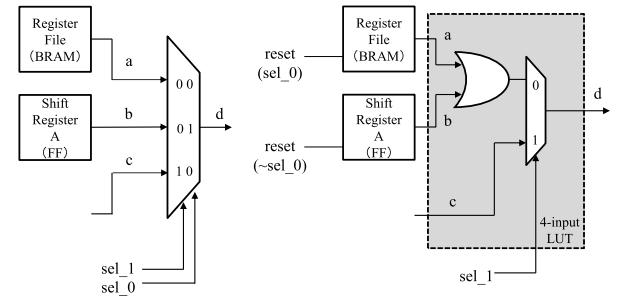
Fig. 6 Resource mappings of datapath from register file to shift register

enable port and a reset port. In Ultrasmall, the reset signals are effectively used to provide a specific value as an output.

3.5.1 Datapath from Register File to Shift Register

The memory components in Ultrasmall, such as instruction memory, data memory, and register file, use block RAMs. While the datapath width of Ultrasmall is 2-bit, these memory components use 32-bit port instead of 2-bit, and each bit of these ports is directly connected to each bit entry of the shift registers. Since the shift register uses 32 FFs, it can be composed on 16 slices. So there are 32 available LUTs on the slices.

Figure 6 shows the mapping of LUTs and flip flops for the datapath between the register file and shift register B on each architecture. In the 1-bit datapath on Supersmall in Fig. 6 (a), each bit is selected from the 32-bit output of the register file, and the bit is serially connected to shift register B. To this end, an additional multiplexer is used that selects 1 bit from 32 bits and consumes 16 slices. Figure 6 (b) shows the mapping in the case of a naive 2-bit datapath with a multiplexer that select 2 bits from 32 bits. This multiplexer consumes eight slices, because each slice has 2 LUTs that are naturally utilized. However, in these two cases, the LUTs in the slices, which are used for the shift register, are



(a) 3-to-1 multiplexer containing more than 2 inputs from BRAM or FF.

(b) 3-to-1 multiplexer with uncommon usage of reset signals (only 1 LUT is required).

Fig. 7 3-to-1 Multiplexer

not utilized.

Figure 6 (c) shows the mapping of LUTs and flip flops in Ultrasmall. Each bit of the 32-bit output is connected to each corresponding LUT in the slices in parallel. The LUTs are synthesized as a 2-to-1 multiplexer that selects a bit from a bit of the register file and a bit of the previous stage of the shift register. Finally, the LUTs and the FFs in the slices are effectively utilized, making the additional multiplexer no longer necessary. It saves eight slices compared to the naive 2-bit datapath implementation in Fig. 6 (b).

3.5.2 3-to-1 Multiplexer on Shift Register

As shown in Fig. 2, there are three inputs into shift register A: from the previous stage of the shift register or the ALU, from the register file, and the data memory. Each bit entry of the shift register uses the 3-to-1 multiplexer that has 5 bits as input, which consists of 3 bits for data (*a*, *b*, and *c*), and 2 bits for selection, (*sel_0* and *sel_1*). It usually requires two 4-input LUTs to emulate such a 5-bit function, as shown in Fig. 7 (a). In Ultrasmall, the 3-to-1 multiplexer for the shift register is realized by just utilizing a single LUT with the reset functionality of the output register on the block RAMs, and the flip flops, as shown in Fig. 7 (b). In this situation, Fig. 7 (b) is equivalent to Fig. 7 (a).

We configured the value of an output register with a specific value when the reset port is asserted. By using the synchronous reset on an output register of a block RAM, an output register can have an initial and reset value. Both input port A and input port B of 3-to-1 multiplexer use zero as the initial and reset value, but they use the country reset condition each other. *sel_0* is connected to the reset port of the output registers on the register file, but the logic is reversed. When the reset port is asserted, the value of the output register is 0. By combining this characteristic and a logical OR, a temporal 2-to-1 multiplexer is realized. By selecting one from the temporal result and another input, a 3-to-1 multiplexer is implemented using a single LUT.

3.5.3 Load/Store Instructions with Byte and Half Word Size

Supersmall uses an FPGA by Altera that has TriMatrix memory [18] to implement the data memory. TriMatrix memory supports the byte enable function for accessing the memory in the byte unit. Supersmall utilizes the byte enable function to implement the byte/half word load instruction (*lb*, *lhw*) and the store byte/half word instruction (*sb*, *shw*). Unfortunately, the block RAMs on Xilinx Spartan-3E FPGAs do not support the byte enable function. In Ultrasmall, we address this issue by supporting byte access but not half word access. Block RAM on Spartan-3E FPGAs has two independent output ports. One is used for byte access and the other is used for the word (4 bytes) access; Ultrasmall supports both the byte/word load and byte/word store instructions.

3.6 Implementation Issue

In this implementation using Xilinx Spartan-3E FPGAs, the number of BRAMs is restricted that Ultrasmall can use to implement the data memory. There are only four choices: 1 BRAM (the width of two ports is configured as 8- and 32-bits), 2 BRAMs (the width of two ports is configured as 4- and 16-bits), 4 BRAMs (the width of two ports is configured as 2- and 8-bits), and 8 BRAMs (the width of two ports is configured as 1- and 4-bits). Therefore, the size of the data memory is restricted to 2KB, 4KB, 8KB, and 16KB.

In addition to the optimizations provided by the logic synthesis tools, we manually use FPGA primitive-optimizations to further reduce the required hardware resources. Generally, in a development targeting FPGA, the logic synthesis tool optimizes the HDL code and translates the described circuits into FPGA primitives. After that, an FPGA bitstream file is created by the place and route tools. However, there are some insufficient optimization parts in the current logic synthesis tools. We found that Xilinx ISE inadequately handles multiplexer and carry-chain primitives. By the manually instantiation of these FPGA primitives as well as in PicoBlaze, it is possible to optimize the resource utilization further. In Ultrasmall, we manually instantiate the FPGA primitives to optimize the multi-stage multiplexers and 2-bit adders.

4. Evaluation

In this section, we evaluate Ultrasmall in terms of hardware utilization and performance. We implemented it based on our proposed architecture and optimization techniques in Verilog HDL. Ultrasmall's source codes are very compact: only about 2,000 lines of code.

We used Stanford integer benchmarks [19] for the CPI evaluation and the Xilinx ISE WebPack 14.7 as the synthesis tool. The optimization goal of the synthesis is the area-first mode, and the optimization effort level is high. The target FPGA is Spartan-3E XC3S500E with a speed grade of −5, which is a common low-end and small FPGA. The benchmark programs were compiled with GCC 4.3.6 with an O2 optimization level.

To compare Ultrasmall with the original Supersmall,

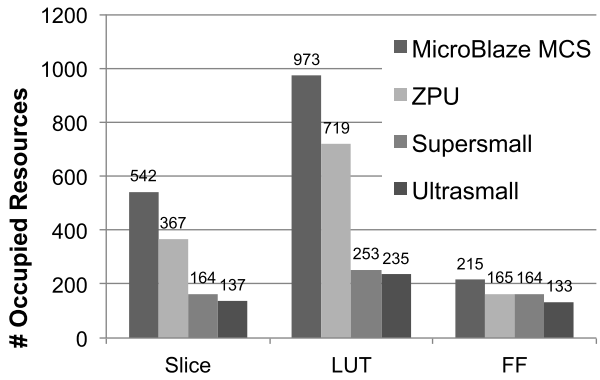


Fig. 8 Amount of occupied hardware resources of soft processors supporting 32-bit ISA

we modified the Verilog HDL source code of Supersmall for Xilinx FPGAs. Supersmall uses TriMatrix memory as the instruction and data memory, which is an on-chip memory element on Altera's FPGAs. It supports byte-enable access to change a portion of the data on the destination location by byte. Supersmall supports byte and half-word load/store instructions naturally by using the byte-enable access function. Supersmall uses a direct definition of memory primitive that creates an instance of TriMatrix memory.

However, Xilinx FPGAs have just a block RAM as an on-chip memory element, instead of TriMatrix memory. Therefore, the part of the instruction memory and data memory should be modified in order to use block RAM on Xilinx Spartan-3E FPGAs. We have replaced the source code of the instances of TriMatrix memory with the device independent descriptions that the synthesis compiler can infer an on-chip memory element from the source code. Other than TriMatrix memory, there are no device-dependent descriptions in Supersmall, so that we have easily developed the implementation of Supersmall on a Xilinx Spartan-3E FPGA.

We compared the hardware resource utilization of Ultrasmall with previous soft processors: MicroBlaze MCS, ZPU, and Supersmall. Figure 8 shows the amount of occupied hardware resources of each soft processor. All of these soft processors have 32-bit ISA. Ultrasmall's result is lower than the other processors. It consumes 137 slices, which correspond to 84% of the number of occupied slices of Supersmall. Since the slices around the datapaths are efficiently utilized by employing the 2-bit datapath and ALU, the amount of occupied slices is reduced. Additionally, since employing the 2-bit datapath and ALU reduces the rate of cycles per instruction, the circuit is also reduced for a finite state machine.

We then measured the performance of Ultrasmall in terms of the rate of cycles per instruction (CPI) and the maximum clock frequency. Cycles per instruction represent the average value of the clock cycles elapsed for each instruction; a lower CPI is better. Table 3 shows the CPI rates of Ultrasmall and Supersmall. In this experiment, we selected the following five programs from the Stanford integer benchmarks: bubble sort (Bubble), permutations (Perm), eight-queens (Queens), quick sort (Quick), and the towers of Hanoi (Towers).

Table 3 Cycles per instructions (CPI) of Ultrasmall and Supersmall

	Bubble	Perm	Queens	Quick	Towers	**Average**
Supersmall	71.4	72.0	71.6	71.3	71.7	**71.2**
Ultrasmall	23.7	21.4	22.5	23.2	22.1	**22.3**

Table 4 Max frequency of soft processors that supports 32-bit ISA

	MCS	ZPU	Supersmall	Ultrasmall
Max Freq [MHz]	112.2	113.7	70.3	65.0

On average, the CPI rate of Ultrasmall was reduced by 68% from that of Supersmall. The largest part of the performance improvement was achieved by employing the 2-bit datapath and ALU, which obviously reduces the number of clock cycles for a computation on ALU. The other reason is the optimization of the finite state machine. In the original Supersmall, since the next value of the program counter (PC) is originally calculated on the ALU for computation, it increases the performance overheads to update the PC. In Ultrasmall, the branch resolution unit has an additional arithmetic unit to calculate the next PC. Actually, it requires additional hardware resources and also reduces the complexity of the original ALU, because it does not calculate the PC value any longer.

Table 4 shows the maximum clock frequencies of Ultrasmall, Supersmall, ZPU, and MicroBlaze MCS. The maximum clock frequencies of Ultrasmall and Supersmall are 65.0 MHz and 70.3 MHz. The maximum clock frequency of Ultrasmall is less than that of ZPU and MicroBlaze MCS. The lower frequency is due to the uncommon use of reset signals for constructing the 3-to-1 multiplexers. On the other hand, compared to MicroBlaze MCS and ZPU, the required hardware resources of Ultrasmall are significantly reduced. In particular, the amount of occupied slices of Ultrasmall is about two to four times smaller than the amount of occupied slices of MicroBlaze MCS and ZPU.

Based on the evaluation results of the CPI rates, we estimated the actual performance in million instructions per second (MIPS), which was estimated using the maximum clock frequency and the average CPI. The MIPS rates of Ultrasmall and Supersmall are 2.90 and 0.98. Ultrasmall achieves 2.9 times better performance in spite of smaller hardware resource utilization.

According to [1], the original Microblaze archives approximately 1 DMIPS per MHz (1757 MIPS), so that the CPI rate of Microblaze is about 0.57, which is less than 1.0. Note that the performance of such standard soft processors is very high compared to the serial processors.

5. Conclusion

In this paper, we presented Ultrasmall, a tiny soft processor architecture that supports a subset of the MIPS-I instruction set architecture. Ultrasmall uses the 2-bit serial ALU for all operations, instead of the original 32-bit ALU,

for high resource efficiency. It achieves better performance than the previous tiny soft processor with a 1-bit datapath and a serial ALU. In addition to these device-independent optimizations, we applied several device-dependent optimizations for Xilinx Spartan-3E FPGAs using 4-input LUTs. Primitive-based optimizations aggressively reduce the amount of occupied slices. Our evaluation result shows that Ultrasmall occupies only 84% of the previous small soft processor. In addition to the utilized resource reduction, Ultrasmall achieves 2.9 times higher performance than the previous one.

While we developed and evaluated 2-bit serial processor in this work, there is an opportunity to improve the resource efficiency by employing 4-bit or more data width, as a future work. However, it seems that the increase of the data width increases the complexity of shift operations. To improve resource efficiency in spite of wider ALU, a sophisticated shift register is mandatory.

References

[1] "MicroBlaze Soft Processor Core." http://www.xilinx.com/tools/microblaze.htm.
[2] "Nios II Processor: The World's Most Versatile Embedded Processor." http://www.altera.com/devices/processor/nios2/ni2-index.html.
[3] J. Robinson, S. Vafaee, J. Scobbie, M. Ritche, and J. Rose, "The supersmall soft processor," Programmable Logic Conference (SPL), 2010 VI Southern, pp.3–8, March 2010.
[4] Y. Tanaka, S. Sato, and K. Kise, "The Ultrasmall Soft Processor," SIGARCH Comput. Archit. News, vol.41, no.5, pp.95–100, Dec. 2013.
[5] H. Nakatsuka, Y. Tanaka, T.V. Chu, S. Takamaeda-Yamazaki, and K. Kise, "Ultrasmall: The smallest MIPS soft processor," Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, pp.1–4, Sept. 2014.
[6] "Xilinx Platform Studio." http://www.xilinx.com/tools/xps.htm.
[7] "Vivado design suite." http://www.xilinx.com/products/design-tools/vivado/index.htm.
[8] "MicroBlaze Micro Controller System (MCS)." http://www.xilinx.com/tools/mb_mcs.htm.
[9] "PicoBlaze 8-bit Microcontroller." http://xilinx.com/products/intellectual-property/picoblaze.htm.
[10] M. Schoeberl, "Leros: A tiny microcontroller for fpgas," Field Programmable Logic and Applications (FPL), 2011 International Conference on, pp.10–14, Sept. 2011.
[11] M. Zandrahimi, H.R. Zarandi, and A. Rohani, "An analysis of fault effects and propagations in zpu: The world's smallest 32 bit cpu," Quality Electronic Design (ASQED), 2010 2nd Asia Symposium on, pp.308–313, Aug. 2010.
[12] D. Lampret, C.M. Chen, M. Mlinar, J. Rydberg, M. Ziv-Av, C. Ziomkowski, G. McGary, B. Gardner, R. Mathur, and M. Bolado, "Openrisc 1000 architecture manual," Description of assembler mnemonics and other for OR1200, 2003.
[13] "Opencores." http://opencores.org/.
[14] R. Herveille et al., "Wishbone system-on-chip (soc) interconnection architecture for portable ip cores," OpenCores Organization, 2002.
[15] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanovic, and K. Asanovic, "A 45nm 1.3ghz 16.7 double-precision gflops/w risc-v processor with vector accelerators," European Solid State Circuits Conference (ESSCIRC), ESSCIRC 2014 - 40th, pp.199–202, Sept. 2014.
[16] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," Proceedings of the 49th Annual Design Automation Conference, DAC '12, New York, NY, USA, pp.1216–1225, ACM, 2012.
[17] "Xilinx DS312 Spartan-3E FPGA Family Data Sheet." http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf.
[18] "Stratix Device Handbook." http://www.altera.com/literature/hb/stx/stratix_handbook.pdf.
[19] J. Hennessy and P. Nye, "Stanford integer benchmarks," Personal communication, 1988.

Shinya Takamaeda-Yamazaki received the B.E., M.E., and D.E. degrees from Tokyo Institute of Technology, Japan in 2009, 2011, and 2014 respectively. From 2011 to 2014, he was a JSPS research fellow (DC1). Since 2014, he has been an assistant professor of the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His research interests include memory system, FPGA computing, high level synthesis and processor architecture. He is a member of IEEE and IPSJ.

Hiroshi Nakatsuka received the B.E. degree from Tokyo Institute of Technology, Japan in 2014. He is currently a master course student of the Graduate School of Science and Engineering, Tokyo Institute of Technology, Japan. His research interests include electrical design automation and computer architecture.

Yuichiro Tanaka received the B.E. and M.E. degrees from Tokyo Institute of Technology, Japan in 2013 and 2015 respectively. His research interests include electrical design automation and computer architecture.

Kenji Kise received the B.E. degree from Nagoya University in 1995, the M.E. degree and the Ph.D. degree from the University of Tokyo in 1997 and 2000 respectively. He is currently an associate professor of the Graduate School of Information Science and Engineering, Tokyo Institute of Technology. His research interests include computer architecture and parallel processing. He is a member of ACM, IEEE, and IPSJ.