A Proposal of Application Specific Approach with RISC-V Processor on FPGA

Tetsuo Miyauchi

School of Information Science
Japan Advanced Institute of Science and Technology
Nomi, Ishikawa 923-1292 Japan
t-miyauc@jaist.ac.jp

Kiyofumi Tanaka

School of Advanced Science and Technology
Japan Advanced Instituted of Science and Technology
Nomi, Ishikawa 923-1292 Japan
kiyofumi@jaist.ac.jp

Abstract— Currently, the number of IoT(Internet of Things) devices is increasing. In IoT devices, small footprint is desirable. RISC-V is an open processor architecture, which is becoming popular for IoT devices. We implemented RISC-V soft processor core, of which instruction set is RV32IM (base implementation and multiple/division in 32 bit registers), on an FPGA with 5-stage pipeline. In this paper, we propose a method for reducing hardware resources by adapting the processor core to an application program. We show our approach can reduce necessary FPGA resources to 14.8% (Rijndael) and 14.4% (Matrix) of the full processor core implementation.

I. Introduction

Currently, the number of IoT(Internet of Things) devices is increasing, so that studies for domain specific architecture are widely undertaken. In IoT devices, small footprint is desirable due to cost reduction and constraint of the available resources.

We have been studying a framework of development environment for application-specific systems. In the literature [9], we introduced our framework for generating application-specific FPGA-based SoC. For a part of this framework, we describe how we implement the processor core and propose a method for minimizing hardware resources to fit the target application. We show the evaluation results comparing full and application specific implementation in the view of resource usage.

To implement the processor core on an FPGA, we adopt RISC-V architecture for the instruction set architecture (ISA). Recently, RISC-V has been gaining popularity for IoT devices since the architecture is open and suitable for IoT devices [6].

The structure of this paper is as follows. In Section II, we show the related researches about application specific systems as well as RISC-V architecture. Section III explains our implementation and features of the processor. In Section IV, how the processor resources are adapted to an application is described. In Section V, we illustrate the evaluation results. Finally, Section VI concludes the paper.

II. Related Work

MicroBlaze [13] of Xilinx and NiosII [14] of Intel are commonly used soft-processors. While we can use these soft-processors on an FPGA, we cannot build an application specific processor since the soft-processors are not open architecture.

ASIP (Application-domain Specific Instruction-set Processor) in the literature [1] is an environment to create an application specific processor. With their approach, a special developing environment for the processor is necessary. ASIP technologies are surveyed in [2].

RISC-V is a modern processor architecture [12]. Instruction set architecture of RISC-V is license-free. The popularity of RISC-V is increasing for IoT devices. There are several processor designs in both of industrial and academic fields.

RISC-V instruction set consists of the base integer instruction set and extensions. ISA RV32I stands for the 32-bit base integer instruction set, and the suffixes M, A, F, and D are the meaning of supporting multiply/divide, atomic instructions, floating point (single precision), and floating point (double precision), respectively. The suffix G supports all of I, M, A, F, and D.

Rocket Chip in the literature [3] is one of the well know RISC-V implementation. With using Rocket Chip generator, from high-performance designs to small embedded processors can be generated. For the small embedded processors, a generated processor is RV32IM ISA and uses 3-stage pipeline. In our approach, while we also developed a processor core with RV32IM ISA, our implementation is the 5-stage pipeline, which has higher throughput than the 3-stage pipeline.

In the literature [5], out-of-order RISC-V core, BOOM, is presented. It has RV64G RISC-V ISA, which is 64-bit processor with double precision floating point instructions. However, 32-bit processor is enough for our target system, so we adopt RV32IM RISC-V ISA.

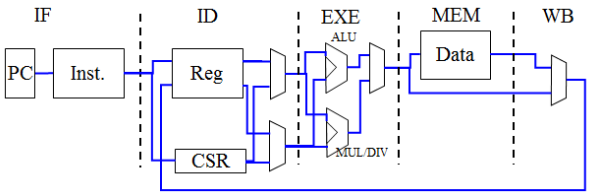

Fig. 1. Processor Block

The literature [4] describes a soft-processor framework for high performance CPU. They present a RISC-V 32-bit processor architecture with RV32IMA ISA. They implement TLBs and caches for OS(Linux) support in shared memory systems whereas our target application is a relatively small system with a real-time operating system without virtual memory.

In IoT systems, since a processor core with small footprint plays a significant role, we have been studying a framework for developing an application specific system. The literature [9] illustrates our whole framework. In this paper, we explain our proposal to create an application specific processor core with using RISC-V architecture.

III. Implementation

We have implemented a processor on an FPGA based on 32-bit RISC-V architecture. The instruction set is composed of a base integer instruction set, RV32I, and the instruction set extension of integer multiplication and division, so RV32IM is our supporting instruction set. We show the processor block diagram roughly in Figure 1, in which only primary data paths are shown for simplification.

In the RISC-V specification [7], how the hardware structure should be implemented is not specified, so we decided to design the processor as follows.

- *Memory Map for small footprint*: Memory map of this processor is illustrated in Figure 2. Program area is located from 0x0 to 0x7fff in 32K bytes area, which is enough for our target applications. The area from 0x8000 to 0xbfff is for read only memory, in which constant data are stored. RAM area for variable data is from 0xc000 to 0xcfff in 4K bytes area. While we decided the memory map as above for this implementation, we can change the memory map flexibly according to the application as long as FPGA resources are allowed.

- *5-stage pipeline*: We implemented this processor core with 5-stage pipeline, which consists of IF (Instruction Fetch), ID (Instruction Decode), EXE (Execution), MEM (Memory access), and WB (Write Back to a register).

- *Branch decision*: We placed the module to decide whether a branch is taken or not in ID stage. Since RISC-V architecture does not have a delay slot, an instruction is flushed in a pipeline register when the branch is taken.

- *CSR registers*: In the RISC-V specification, 4096 Control and Status registers (CSR) are indicated by bits 31-20 in csr instructions [8]. While these registers are defined in the specification, we implemented only actually used eleven registers.

- *Forwarding unit*: When an instruction uses the result of the former instruction, the data have to be forwarded from the corresponding stage of the pipeline. The example is as follows.

```
add x3, x2, x1
add x4, x3, x5
```

  In this case, the value of x3 register has to be forwarded from the MEM stage in order that the value is used in EXE stage for the second `add` instruction since the value to be written in x3 exists in MEM stage.

- *Hazard detection unit*: When an instruction uses the results of the former instruction and the forwarding unit cannot supply the results immediately, the pipeline detection unit works and stall the pipeline. While there are several cases to stall the pipeline, we show one example as follows. In this case, the value of x3 is used in ID stage to decide whether the branch operation should be performed, but the value of x3 has to be one which is being calculated in the next stage, EXE stage, so the pipeline has to be stalled to wait for the x3 register value to be generated.

```
add x3, x2, x1
beq x3, x4, label
```

IV. Building Application Specific Processor

In this section, we explain how the application specific processor is built. While there are 40 instructions in RV32I Base Instruction Set, 6 instructions in CSR Standard Extension, and 8 instructions in RV32M Standard Extension, all of these instructions are not used in an application program.

If the system can be implemented without using CSR instructions, we can delete the CSR related registers, wires, and multiplexers. Similarly, multiplication calculation unit can be removed if an application program does not use `mul, mulh, mulhsu` or `mulhu` instructions, and division calculation unit can be removed if an application

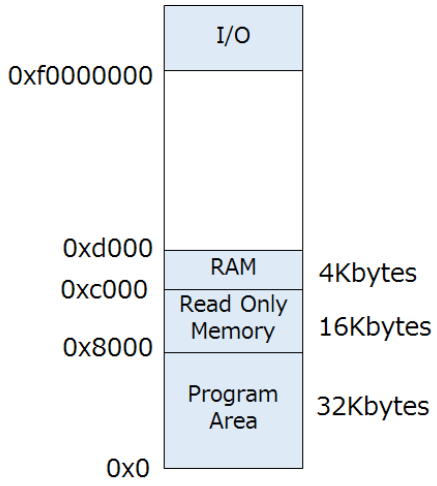

Fig. 2. Memory map

TABLE I
ALU OPERATION

bit		
30	14:12	operation
0	000	addition
1	000	subtraction
0	000	shift left
0	010	set less than (signed)
0	011	set less than (unsigned)
0	100	exclusive or
0	101	shift right logical
1	101	shift right arithmetic
0	111	and

program does not use `div, divu, rem` or `remu` instructions. For ALU (Arithmetic Logic Unit), an operation of ALU is decided by the field of bits 30,14-12 in R-type and I-type instructions as Table I. We can omit operations which are not used in an application program.

Table II shows all of RISC-V RV32IM instructions and ○ idicates the instructions which application programs of Rijndael and Matrix actually use. Rijndael is a commonly used block cipher algorithm known as Advanced Encryption Standard (AES) [11], and Matrix is a program for multiplication of 10 x 10 matrices.

In general, there are several types of memory accesses, read/write and byte/half word/word, in an application program. In 32-bit RISC-V (RV32) load instructions, there are two types of extension operations, signed extension and zero extension, and three types of data sizes, byte, half-word and word(32bits). To manipulate these types, a processor has a data selection function as depicted in Figure 3. If an extension type or a type of data size is not used in an application program, we can delete the corresponding hardware resources.

TABLE II
INSTRUCTIONS

Inst.	Rijndael	Matrix	Inst.	Rijndael	Matrix
lui	○	○	add	○	○
auipc		○	sub	○	
jal	○	○	sll		
jalr	○	○	slt		
beq	○	○	sltu	○	
bne	○		xor	○	
blt	○		srl		
bge	○	○	sra		
bltu			or	○	
bgeu			and	○	
lb		○	fence		
lh			ecall		
lw	○	○	ebreak		
lbu	○		csrrw		
lhu			csrrs		
sb	○	○	csrrc		
sh			csrrwi		
sw	○	○	csrrsi		
addi	○	○	csrrci		
slti			mul		○
sltiu			mulh		
xori			mulhsu		
ori			mulhu		
andi	○		div		
slli	○	○	divu		
srli	○		rem		
srai			remu		

After the resources are selected as above, in the case of Rijndael, the hatched modules in Figure 4 can be removed and only the dotted modules are actually embodied.

V. EVALUATION

We have implemented the proposed application specific processor in an FPGA device, Xilinx Artix-7(XC7A35T) [10]. The processor core runs at 50MHz. Rijndael and Matrix application programs are used for evaluation.

As Table II shows, the program code of Rijndael uses 21 instructions out of 54 instructions in RV32IM. On the other hand, the program code of Matrix uses 14 instructions. As described in the previous section, we meticulously remove unused modules, wires, multiplexers, and functions of arithmetic logic unit.

Table III shows FPGA resources in post implementation. The column of "Full" indicates the resource usage of the implementation for full functions of the processor without adaptation for an application program. The col-

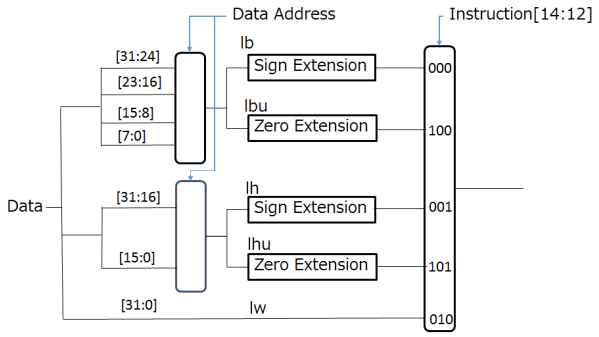

Fig. 3. Data Selection

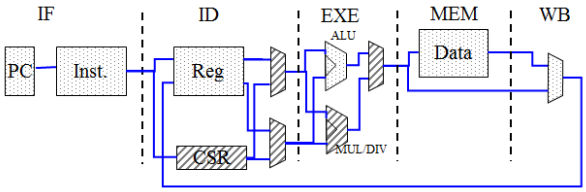

Fig. 4. Processor Block (Rijndael application specific)

umn of "Rijndael" is the resource usage of the implementation which is specific to the application program for Rijndael algorithm. Encrypted data are stored in the Read Only Memory and we confirm the data are properly decrypted. The column of "Matrix" is the resource usage of the implementation which is specific for Matrix application. Elements of two matrices are stored in the Read Only Memory in advance and we confirm the multiplied data are equal to the expected values. In this table, we can see that LUT is reduced from 13185 to 1962 (14.8%) of "Full" in Rijndael, and from 13185 to 1895 (14.4%) of "Full" in Matrix, respectively. This is the effect of removed unused modules such as CSR, division, wires, multiplexers, and functions of arithmetic logic unit.

TABLE III
FPGA Resources

Resources	Full	Rijndael	Matrix	Available
LUT	13185	1962	1895	20800
FF	1935	1479	1482	41600
BRAM	13.5	13.5	13.5	50
DSP	12	0	8	90

VI. Conclusions

This paper presented our method for developing an application specific processor with small footprint for IoT devices. To implement this processor, we developed a 32-bit processor core with RISC-V architecture in 5-stage pipeline. We show FPGA resources can be reduced to 14.8% (Rijndael) and 14.4% (Matrix) in our approach.

In future work, we are developing an environment to generate an application specific processor automatically.

Acknowledgements

This work is supported partly by JSPS KAKENHI Grant Number JP 19K11873 for developing adaptation technology, and partly by SHIBUYA Science Culture and Sports Foundation for designing a RISC-V processor.

References

[1] M. Imai, Y. Takeuchi, K. Sakanushi, N. Ishiura, Advantage and Possibility of Application-domain Specific Instruction-set Processor (ASIP), *IPSJ Transactions on System LSI Design Methodology Vol.3*, pages 161–178, 2010.

[2] M.J. Jain, M. Balakrishnan, A. Kumar, ASIP Design Methodologies: Survey and Issues, *Proceedings of 14th International Conference on VLSI Design*, pages 76–81, 2001.

[3] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin *et al*, "The Rocket Chip Generator", Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2016-17, April 2016

[4] E. Matthews, L. Shannon, "TAIGA: A new RISC-V soft-processor framework enabling high performance CPU architectural features", 27th International Conference on Field Programmable Logic and Applications (FPL) ,2017

[5] C. Celio, Pi-Feng Chiu, B. Nikolic, D. A. Patterson, K. Asanović, "BOOM v2:an open-source out-of-order RISC-V core, Computer Architecture Research with RISC-V, 2017

[6] D. A. Patterson, J. L. Hennessy, "Computer Organization and Design, RISC-V Edition", Morgan Kaufmann Publishers

[7] A. Waterman, K. Asanović, "The RISC-V Instruction Set Manual Volume I: Unprivileged ISA", Document Version 20190608-Base-Ratified

[8] A. Waterman, K Asanović, "The RISC-V Instruction Set Manual Volume II: Privileged Architecture", Document Version 20190608-Priv-MSU-Ratified

[9] T. Miyauchi, K. Tanaka, "A Framework for Automatic Generation of Application-Specific FPGA-based SoC", SASIMI 2016

[10] "Artix-7" [Online] Available `https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html`

[11] "Rijndael" [Online] Available `https://embeddedsw.net/Cipher_Reference_Home.html#AES`

[12] "RISC-V" [Online] Available `https://riscv.org/`

[13] "MicroBlaze" [Online] Available `http://www.xilinx.com/products/design-tools/microblaze.html`

[14] "NIOSII" [Online] Available `https://www.intel.com/content/www/us/en/products/programmable/processor/nios-ii.html`