KIPS Trans. Comp. and Comm. Sys.
Vol.11, No.1 pp.1~8
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2022.11.1.1

Design and Evaluation of 32-Bit RISC-V Processor Using FPGA

Sungyeong Jang† · Sangwoo Park†† · Guyun Kwon††† · Taeweon Suh††††

ABSTRACT

RISC-V is an open-source instruction set architecture which has a simple base structure and can be extensible depending on the purpose. In this paper, we designed a small and low-power 32-bit RISC-V processor to establish the base for research on RISC-V embedded systems. We designed a 2-stage pipelined processor which supports RISC-V base integer instruction set except for FENCE and EBREAK instructions. The processor also supports privileged ISA for trap handling. It used 1895 LUTs and 1195 flip-flops, and consumed 0.001W on Xilinx Zynq-7000 FPGA when synthesized using Vivado Design Suite. GPIO, UART, and timer peripherals are additionally used to compose the system. We verified the operation of the processor on FPGA with FreeRTOS at 16MHz. We used Dhrystone and Coremark benchmarks to measure the performance of the processor. This study aims to provide a low-power, high-efficiency microprocessor for future extension.

Keywords : RISC-V, FPGA, Processor

FPGA를 이용한 32-Bit RISC-V 프로세서 설계 및 평가

장 선 경† · 박 상 우†† · 권 구 윤††† · 서 태 원††††

요 약

RISC-V는 오픈 소스 명령어 집합 구조로, 간단한 기본 구조를 가지며 목적에 따라 명령어 집합을 유연하게 확장할 수 있다. 본 논문에서는 소형, 저전력 32-bit RISC-V 프로세서를 설계하여 RISC-V 임베디드 시스템 연구를 위한 기반을 마련하고자 하였다. 설계한 프로세서는 2단계 파이프라인으로 구성하였고, RISC-V ISA 중 FENCE, EBREAK 명령어를 제외한 32-bit 정수형 ISA 및 인터럽트 처리를 위한 특권 ISA를 지원한다. Vivado Design Suite를 이용하여 합성한 결과 Xilinx Zynq-7000 FPGA에서 1895개의 LUT 및 1195개의 플립플롭을 사용하였고, 0.001W의 전력을 소모하였다. 이를 GPIO, UART, 타이머와 함께 시스템을 구성하여 합성하였고, FPGA 상에서 FreeRTOS를 포팅하여 16MHz에서의 동작을 검증하였다. Dhrystone, Coremark 벤치마크를 통해 성능을 측정하여 목적에 따라 확장 가능한 저전력 고효율 프로세서임을 보였다.

키워드 : RISC-V, FPGA, Processor

1. 서 론

RISC-V는 2010년 버클리 대학에서 개발되기 시작한 RISC (Reduced Instruction Set Computer) 계열 명령어 집합 구조 (ISA, Instruction Set Architecture) 이다[1,2]. 2015년 비영리 법인 RISC-V Foundation이 설립되어 현재 구글, NVIDIA, 퀄컴 등을 포함한 2천여 개의 단체들이 연구를 지원하고 있다. RISC-V는 기존의 Arm, x86 등 상용 ISA와 달리 무료로 사용 가능한 오픈 소스(open source) ISA를 지향하며[3], 아키텍처 연구 및 교육 목적뿐만 아니라 유연한 명령어 집합 확장(extension)을 통해 클라우드, AI, 사물인터넷 등 광범위한 산업 분야에도 적용할 수 있다[4,5]. 또한, 이미 고정된 명령어 집합 외에도 사용자 임의의 명령어 집합을 추가할 수 있어 특정 목적에 특화된 프로세서를 설계할 수 있다[6,7]. 국내에서는 대부분의 RISC-V 관련 연구가 시뮬레이션 또는 이미 출시된 RISC-V SoC에 기반하고 있으며[8,9], 프로세서 설계 자체에 대한 주목도는 낮은 편이다. 본 논문에서는 Verilog HDL을 이용하여 RISC-V ISA를 기반으로 FPGA (Field Programmable Gate Array)에서 동작하는 in-house 프로세서를 설계 및 평가하였다. 제안하는 프로세서는 RISC-V 정수형 ISA 및 인터럽트 처리를 위한 특권(privileged) ISA

※ 본 연구는 삼성전자의 지원(과제번호IO210204-08384-02)을 받아 수행된 결과임.
※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2019R1A2C1088390).
† 준 회 원 : 고려대학교 컴퓨터학과 석사과정
†† 준 회 원 : 고려대학교 컴퓨터학과 박사과정
††† 비 회 원 : 고려대학교 반도체시스템공학과 석사과정
†††† 종신회원 : 고려대학교 컴퓨터학과 교수
Manuscript Received : September 28, 2021
First Revision : November 12, 2021
Accepted : November 22, 2021
* Corresponding Author : Taeweon Suh(suhtw@korea.ac.kr)

를 실행할 수 있다. 내부 구조가 간단한 2단계 파이프라인으로 구성하여 소형화하였고, 대기 상태에서의 전력 소모를 줄일 수 있는 클럭 게이팅(clock gating) 기법을 적용하였다. 오픈 소스 PULPissimo SoC[10]를 래퍼런스로 하여 시스템을 구성하였고, FreeRTOS[11] 및 Dhrystone[12], Coremark[13] 벤치마크를 이용하여 FPGA에서 설계한 프로세서의 동작을 검증하고 성능을 측정하였다. 본 논문은 이를 통해 소형, 저전력 RISC-V 프로세서를 확보하여 향후 AI 가속기(accelerator), 저전력 사물인터넷 시스템 등의 연구를 위한 기반을 제공하는데 목적이 있다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 소개하고, 3장에서 RISC-V 특권 및 비특권 ISA에 대하여 설명한다. 4장에서 설계한 프로세서의 구조 및 특징을 설명한다. 5장에서 주변장치를 포함한 시스템 구성을 설명하고, 실험 환경 및 FPGA를 이용한 실험 결과에 대해 논의한다. 6장에서 본 논문의 결론을 맺는다.

2. 관련 연구

RISC-V ISA를 기반으로 다양한 프로세서들이 연구 및 개발되었다. 취리히 연방 공과대학교는 볼로냐 대학과 함께 2013년부터 PULP(Parallel Ultra Low Power) 프로젝트를 진행하여 RISC-V 기반 오픈 플랫폼을 구축하고 있다. PULP 확장 명령어를 설계하여 디지털 신호 처리 성능을 높였으며, 이를 지원하는 프로세서를 발표하고 있다. Riscy[6]는 4단계 파이프라인으로 구성된 32-bit 프로세서로, RV32IMCF ISA를 지원한다. Zero-riscy[14]는 초저전력 시스템을 위해 설계된 32-bit 프로세서로, 2단계 파이프라인으로 구성되며 RV32IMCE ISA를 지원한다. 이를 이용하여 싱글 코어 PULPissimo SoC 및 멀티 코어 Mr.Wolf SoC[15] 등을 배포하였다. 또한, 6단계 파이프라인 구조의 64-bit 프로세서인 Ariane[16]은 운영체제 지원을 위한 특권 계층을 구현하였다.

버클리 대학은 Scala 언어에 기반한 하드웨어 설계 언어인 Chisel을 사용하여 RTL(Register Transfer Level)을 생성하는 Rocket Chip Generator 플랫폼을 구축하였다[17]. 각종 라이브러리를 연결하여 합성 가능한 RTL 및 시뮬레이션 플랫폼 등을 생성할 수 있다. Rocket core는 5단계 파이프라인으로 구성된 순차 실행(in-order) 프로세서로, RV32G 및 RV64G ISA를 구현하였다. 또한, 가상 메모리, 캐시, 분기 예측 등을 지원하는 MMU(Memory Management Unit)를 포함한다. Boom[18]은 Rocket chip을 기반으로 하는 비순차 실행(out-of-order) 슈퍼스칼라(superscalar) 프로세서로, 6단계 파이프라인으로 구성되며 RV64G ISA를 구현하였다.

PicoRV32[19]는 소형화에 초점을 맞춘 싱글 사이클(single cycle) 프로세서로, RV32IMC ISA를 지원한다. 클럭 주파수를 높이고자 명령어 1개를 처리하기 위해 최소 3 클럭 사이클(clock cycle)에서 최대 15 클럭 사이클을 소모한다. 하드웨어를 간소화하기 위해 인터럽트 모듈을 옵션으로 선택하도록 설계하였다. 자체적인 인터럽트 처리 명령어를 사용하였고, 이를 위해 RISC-V ISA에서 정의하는 32개의 일반 목적 레지스터 외에 4개의 레지스터를 추가하였다.

3. RISC-V ISA

3.1 Unprivileged ISA

RISC-V ISA는 기본이 되는 정수형(Integer) 명령어 집합과 추가적인 확장 명령어 집합들로 정의된다. 정수형 명령어 집합에는 RV32I, RV64I, RV128I가 있으며, 각각 32-bit, 64-bit, 128-bit 명령어를 제공한다. 여기에 소형 마이크로컨트롤러를 지원하기 위한 RV32E(Embedded)를 포함하여 현재 4개의 기본 명령어 집합(base ISA)이 있다. 목적에 따라 확장 가능한 명령어 집합에는 정수형 곱셈/나눗셈을 위한 M(Multiplication and Division), 단정도 부동소수점 연산을 위한 F(Floating-point), 코드의 크기를 줄일 수 있는 C(Compressed) 등이 있다. 본 논문에서는 임베디드 장치에 널리 사용되는 RV32I 명령어 집합을 실행 가능한 프로세서를 설계하였다.

3.2 Privileged ISA

RISC-V는 특권 명령어 집합 및 CSR(Control and Status Register)을 별도의 문서에 기술한다. 현재 U(User), S (Supervisor), M(Machine) 등 3개의 권한 수준이 정의되어 있다. M 모드가 가장 높은 권한 수준이며 하드웨어 플랫폼에서 필수적으로 지원해야 한다. RISC-V는 실행 중인 명령어와 연관된 비정상적인 상황을 예외(exception)라 하고, 외부의 비동기적인 이벤트를 인터럽트(interrupt)라 한다. 이 둘을 트랩(trap)으로 통칭하며, 트랩 발생 시 트랩 핸들러(trap handler)로 분기한다. 특권 명령어는 시스템 콜(system call), 인터럽트 처리 후 복귀 등의 기능을 제공한다. CSR은 프로세서의 상태를 관리하는 레지스터로, 프로세서 정보를 나타내거나 트랩을 처리하는 등 일반 목적 레지스터와 달리 정해진 역할이 있다. 본 논문에서 구현한 프로세서는 M 모드에서의 트랩 처리를 위한 특권 명령어 집합 및 CSR을 구현하였다.

4. 32-bit RISC-V 프로세서 설계 및 구조

본 논문에서 제안하는 32-bit 프로세서는 RV32I 기본 명령어 40개 중 데이터처리 명령어, 분기 명령어, 메모리 접근 명령어 등 38개를 실행할 수 있으며, CSR에 접근하여 원자적(atomic) 읽기 및 쓰기 작업을 하는 CSR 명령어를 지원한다. 설계한 프로세서는 싱글 코어 환경에서 명령어를 순차 실행하기 때문에 메모리 접근 순서를 정렬하는 FENCE 명령어를 NOP(No OPeration) 명령어로 대체하였다. 또한, 소형화를 위해 디버깅 모듈을 구현하지 않고 프린트를 통한 디버깅 환경을 지원하여 EBREAK 명령어를 ECALL 명령어로 대체하였다.

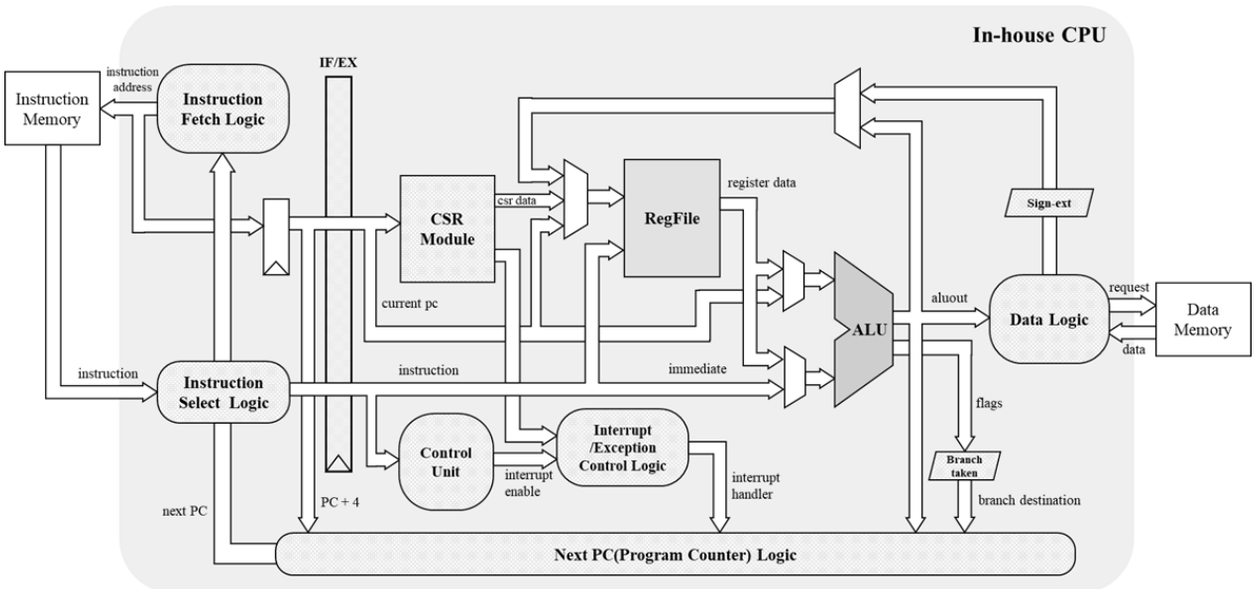


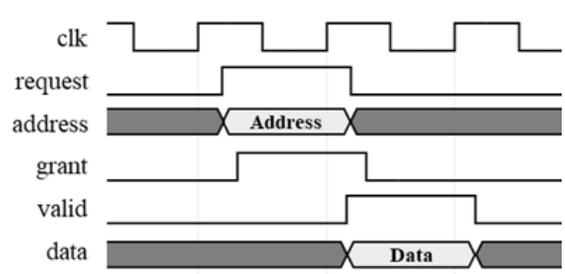
Fig. 1. Processor Schematic Diagram

설계한 프로세서는 명령어 인출(Fetch)과 명령어 실행(Execute)의 2단계 파이프라인으로 구성하였으며, Fig. 1에 간략화한 구조를 나타내었다. 레지스터 파일은 x0부터 x31까지 32개의 32-bit 일반 목적 레지스터를 플립플롭으로 구성하였다. 0으로 값이 고정된 x0를 제외한 나머지는 레지스터 쓰기 제어 신호를 이용하여 클럭의 상승 에지(rising edge)에서 명령어의 실행 결과를 저장한다. 명령어 메모리와 데이터 메모리를 각 32KB로 분리하여 같은 클럭에 동시에 접근할 수 있도록 하였다.

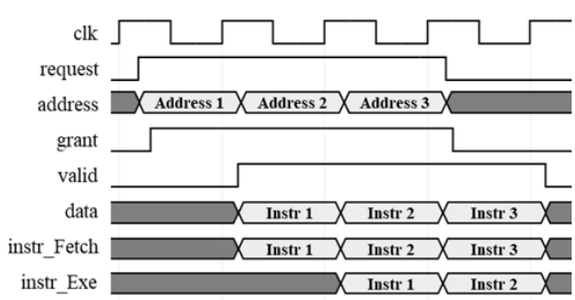
4.1 명령어 인출 단계

명령어 인출을 위해 프로세서의 상태를 제어하는 FSM (Finite State Machine)을 설계하였다. 프로세서의 현재 상태에 따라 인출할 명령어의 주소를 결정하고, 명령어 메모리에 해당 주소에 대한 요청(request)을 보낸다. 메모리는 요청을 받으면 Fig. 2(a)와 같이 해당 클럭에 승인(grant) 신호를 보내고, 다음 클럭에 유효(valid) 신호 전달 및 데이터 적지(load) 또는 저장(store) 작업을 수행하므로 메모리 접근 명령에는 2 클럭 사이클이 소모된다. 이를 1 클럭 사이클로 줄이기 위해 1개의 명령어를 미리 요청하는 로직을 설계하였다. 명령어 인출 로직에서 명령어를 요청하면 다음 클럭에 명령어를 받을 수 있다. 따라서 해당 명령어의 PC(Program Counter)를 플립플롭을 통해 1 클럭 지연시킨다. 이를 통해 Fig. 2(b)와 같이 명령어 주소와 명령어가 파이프라인에 채워지는 타이밍을 맞춰주며, 결과적으로 1 클럭 사이클에 명령어 1개를 가져오는 것과 같이 동작한다.

명령어 인출 단계에서는 제어 신호(control signal)에 따라 명령어를 선택하여 명령어 실행 단계로 넘겨준다. 일반적으로 명령어 메모리로부터 인출한 명령어를 전달한다. 분기



(a) Memory Transaction Timing Diagram



(b) Adjusted Memory Transaction Timing Diagram

Fig. 2. Memory Transaction Timing Diagram

시에는 인출했던 명령어 대신 NOP 명령어를 선택하여 인출 단계 파이프라인의 명령어를 flush 한다. 데이터 적재 및 WFI(Wait For Interrupt) 명령어 등을 실행하면 프로세서는 대기 상태(stalled)가 되어 PC의 값이 증가하지 않으며, 인출 단계의 명령어는 다음 클럭 사이클에 덮어 씌워진다. 따라서 프로세서를 재시작할 때 2 클럭 사이클을 소모하여 인출 단계의 PC에 대한 명령어를 메모리에 다시 요청해야 한다. 이러한 오버헤드는 데이터 적재가 빈번한 프로그램 실행 시 성능 저하의 원인이 되므로 플립플롭을 이용하여 Fig. 3과

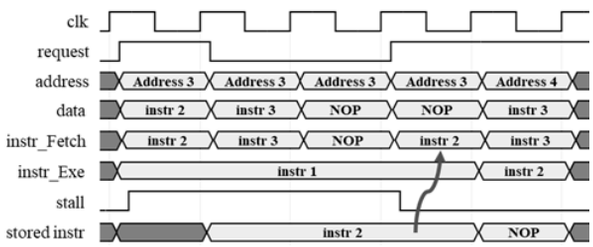


Fig. 3. Instruction Fetch after Stall

같이 프로세서가 대기 상태가 될 때의 명령어를 저장하였다. 2단계 파이프라인 구조에 적합한 크기인 32-bit 버퍼에 명령어 1개를 저장함으로써 프로세서를 재시작할 때 저장했던 명령어를 명령어 인출 단계로 전달하여 대기 상태에서 재시작할 때의 오버헤드를 없앴다.

4.2 명령어 실행 단계

명령어 실행 단계에서는 명령어를 디코딩(decoding)하여 실행한 후, 명령어에 따라 레지스터 파일이나 메모리에 접근 및 쓰기 작업을 수행한다.

제어 유닛(Control Unit)은 3종류의 디코더로 구분하였다. 메인 디코더는 명령어의 opcode(operation code)와 funct3 (function 3) 필드를 디코딩하여 메모리 접근, 레지스터 쓰기, 분기 등의 제어 신호를 생성한다. ALU(Arithmetic Logic Unit) 디코더는 덧셈, 뺄셈 등의 연산을 위한 ALU 제어 신호를 생성한다. 특권 명령어 및 CSR 명령어에 대한 디코더를 메인 디코더와 분리하여 비특권 명령어 제어 신호와 구분하였다.

ALU는 제어 신호에 따라 산술논리 연산 및 가산기를 이용한 플래그(flag) 연산을 수행한다. 가산기의 지연 시간(delay) 및 차지하는 면적을 알아보기 위해 Ripple Carry Adder (RCA), Ripple-block Carry-Lookahead Adder (RCLA), Kogge Stone Adder(KS)를 비교하였다. Xilinx Zynq-7000에서 합성한 결과 RCA의 지연 시간은 19.24ns이며 40개의 LUT(Look-Up Table)을 사용하였다. 8-bit 블록으로 구성된 RCLA의 지연 시간은 11.2ns이며 92개의 LUT을 사용하였다. Prefix adder의 일종인 KS는 200개의 LUT을 사용하였지만 8.06ns의 가장 짧은 지연 시간을 보였다. 2단계 파이프라인의 경우 명령어 실행 단계에서 최장 경로(critical path)가 길어지는 단점이 있어 이를 보완하기 위해 KS를 사용하였다.

메모리 접근 명령어의 경우 데이터 메모리 접근을 관리하는 FSM을 설계하였다. 데이터 저장 명령어 실행 시 승인 신호를 받으면 요청이 처리되었다고 보았다. 데이터 적재 명령어 실행 시 승인 신호를 받아도 다음 클럭에 유효한 데이터를 받기 때문에 stall 제어 신호를 생성하여 프로세서가 1 클럭 사이클만큼 대기하도록 하였다.

4.3 인터럽트 및 예외 처리 로직

설계한 프로세서에서 지원하는 M 모드 특권 명령어 및 CSR은 Table 1과 같다.

Table 1. Implemented RISC-V Privileged ISA

Type	Mnemonic	Description
Privileged instructions	MRET	Return from trap
	WFI	Wait for interrupt
Control and Status Registers	MISA	Machine ISA information
	MSTATUS	Machine status
	MTVEC	Machine trap-vector base address
	MEPC	Machine exception PC
	MCAUSE	Machine cause

부팅 시 MSTATUS 레지스터의 mie(interrupt enable) 비트를 0으로 초기화하여 인터럽트를 받지 않는 상태로 설정한다. 이후 사용자 프로그램에서 CSR 명령어를 통해 mie 비트를 1로 설정하여 인터럽트를 받도록 할 수 있다. 또한, 프로그램을 통해 MTVEC 레지스터에 트랩 벡터 테이블의 시작 주소를 저장한다. 이때, MTVEC 레지스터의 mode 비트의 값을 1로 고정하였다. 따라서 인터럽트가 발생하면 PC에 해당 인터럽트 ID를 4배 한 값을 더한 주소로 분기하는 반면, 예외는 발생 시 트랩 벡터 테이블의 시작 주소로 분기한다.

인터럽트 발생 시 MSTATUS 레지스터의 mie 비트가 1이면 명령어 인출 단계의 명령어를 NOP으로 변경하여 flush 한다. 명령어 실행 단계의 명령어에 대해서는 제어 신호를 모두 0으로 변경하여 실행되지 않은 것과 같은 효과를 낸다. CSR 모듈에서 MSTATUS 레지스터의 mie 비트를 mpie (previous interrupt enable) 비트에 저장한 후 mie 비트를 0으로 설정하여 새로운 인터럽트를 받지 않도록 한다. 이후 정책에 따라 프로그램에서 mie 비트를 1로 설정하여 중첩된 인터럽트를 받도록 할 수 있다. 인터럽트의 ID를 MCAUSE 레지스터에 저장하며, 이를 이용하여 분기할 벡터 테이블의 주소를 계산한다. 실행 중이던 명령어의 주소는 MEPC 레지스터에 저장했다가 인터럽트 처리 후 해당 명령어로 복귀 및 재실행하도록 한다. 인터럽트 처리 루틴은 MRET(return) 명령어 실행을 통해 이루어지며, MSTATUS 레지스터의 mpie 비트를 mie 비트에 복원하여 다시 인터럽트를 받을 수 있도록 한다.

예외는 인터럽트와 달리 동기적으로 발생하며, 설계한 프로세서는 ECALL 명령어에 의한 예외 및 illegal instruction, misaligned instruction address에 의한 예외를 처리할 수 있다. 제어 유닛에서 명령어 집합에 존재하지 않는 명령어를 탐지하면 illegal instruction 예외를 발생시키며, 분기 시 목적 주소가 아닌 분기 명령어의 주소를 저장한다.

4.4 클럭 게이팅

설계한 프로세서는 WFI 명령어 실행 시 클럭 게이팅 기법을 적용하여 대기 상태에 들어감으로써 전력 소모를 줄이고자 하였다. 제어 모듈에서 WFI 제어 신호를 생성하여 프로세서 FSM에 전달하면 프로세서는 sleep 상태가 되어 명령어 요청 없이 인터럽트를 기다린다. 이때, 프로세서에 공급되는

클럭을 끊어줌으로써 내부의 모든 플립플롭의 상태 변이를 막는다. 인터럽트가 발생하면 클럭을 다시 공급하도록 제어 신호를 생성하고, mie 비트가 1이면 인터럽트 처리 루틴을 수행한다. mie 비트가 0이면 프로세서는 WFI 다음 명령어부터 실행을 재개하며, 이후의 동작은 시스템의 정책에 따라 프로그램에서 결정한다. 이를 통해 대기 상태에서 사용자 입력 등의 이벤트를 장시간 기다려야 하는 저전력 기기 등에서 전력 소모를 줄일 수 있다.

5. 프로세서 동작 검증 및 성능 측정

5.1 실험 환경 및 시스템 구성

본 논문에서는 프로세서 검증을 위해 먼저 Siemens Modelsim을 이용하여 시뮬레이션하고, Vivado Design Suite를 이용하여 합성(synthesis) 및 place & route를 진행하였다. Xilinx Zynq-7000 FPGA를 사용하였고, 16MHz에서 동작 검증 및 성능 측정을 수행하였다.

설계한 프로세서에 PULPissimo SoC[10]를 래퍼런스로 하여 메모리, UART(Universal Asynchronous Receiver/Transmitter), GPIO(General Purpose I/O) 및 타이머(timer) 주변장치를 포함하는 시스템을 구성하였다. Fig. 4는 시스템을 Zynq-7000에서 place & route를 진행한 결과이다. 노란색은 구현한 프로세서, 빨간색은 메모리, 초록색은 주변장치를 나타낸다. 사용자 프로그램 실행을 위해 프로그램을 컴파일한 바이너리 값으로 메모리를 초기화하여 standalone 방식으로 동작하도록 하였다. UART를 통해 컴퓨터에서 minicom 등 시리얼 통신이 가능한 터미널 프로그램을 이용하여 구현한 프로세서와 통신하며, 이를 이용하여 프린트 방식으로 디버깅할 수 있다. GPIO는 FPGA의 버튼, 스위치, LED 등의 장치를 제어하며, 본 논문에서는 프로세서 검증을 위한 프로그램에서 이용하였다. 타이머는 클럭을 카운트하여 시간을 측정할 수 있으며, 설정에 따라 인터럽트를 발생시킬 수 있다. 본 논문에서는 성능 측정 및 작업 스케줄링(task scheduling)을 위해 사용하였다.

Fig. 4. Place&route Result of In-house Processor

Table 2. Hardware and Power Consumption Comparison

프로세서	LUT(%)	Flip-Flop(%)	Power(W)
Riscy	21.3%	3.22%	0.01W
Zero-riscy	6.33%	1.3%	0.004W
In-house	**3.56%**	**1.12%**	**0.001W**

Table 2에 프로세서의 하드웨어 사용률 및 소모 전력을 나타내었으며, 비교를 위해 래퍼런스로 사용한 PULPissimo SoC의 Riscy 및 Zero-riscy 프로세서의 하드웨어 사용률도 함께 나타내었다.

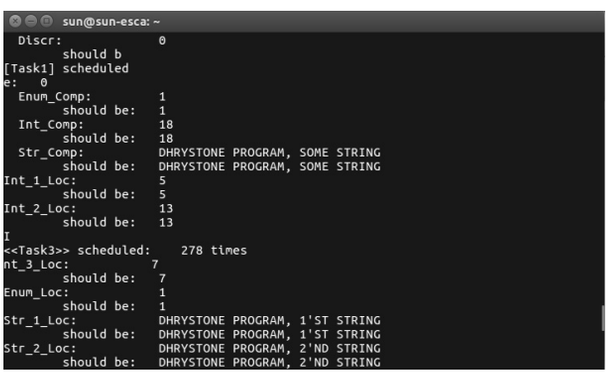
Riscy는 4단계 파이프라인 구조로 이루어진 프로세서로, 2단계 파이프라인 구조인 in-house 프로세서보다 critical path가 짧다. 따라서 클럭 주파수를 높일 수 있지만, hazard 발생에 따른 제어 로직의 증가로 인한 하드웨어 사용률 및 전력 소모량이 증가하게 된다. 또한, 제안하는 프로세서와 같은 2단계 파이프라인 구조인 Zero-riscy 프로세서를 비교했을 때, 명령어 인출 로직이 정수형 명령어 실행에 더 최적화되어 간결하며, 플립플롭 사용량 대부분을 레지스터 파일이 차지하므로 설계한 프로세서에서 하드웨어 사용률 및 소모 전력량이 비교적 낮음을 확인할 수 있다. 배터리로 동작하는 모니터링 시스템[20, 21] 및 항상 켜져 있는(always-on) 관리 시스템[22] 등 전력 및 시스템 크기가 제한적인 환경에서는 자원 사용률이 낮은 in-house 프로세서가 비용 측면에서 유리하다.

5.2 FreeRTOS 포팅을 통한 동작 검증

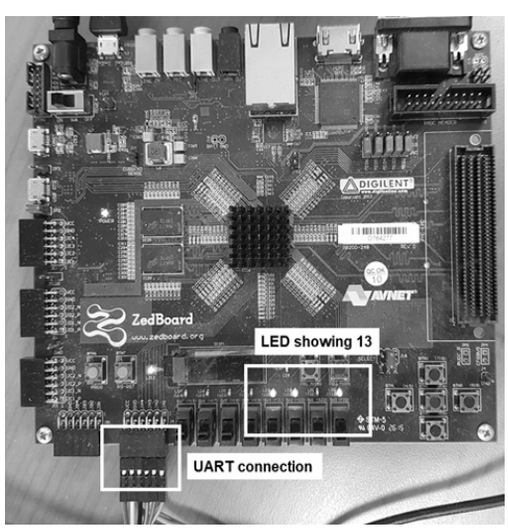
본 논문에서는 RTOS(Real-Time Operating System)의 일종인 FreeRTOS를 포팅(porting)하여 설계한 프로세서의 동작을 검증하였다. FreeRTOS는 무료로 배포되는 RTOS 커널(kernel)로, 바이너리 크기가 작고 구조가 간단하여 많은 마이크로컨트롤러에 포팅되었다. 본 논문에서는 검증을 위해 FreeRTOS v8.2.2를 이용하였다. 16MHz CPU 클럭과 32KHz 타이머 클럭을 사용하였고, 우선순위 없이 10ms 주기로 타이머 인터럽트를 통해 작업 스케줄링을 수행하도록 설정하였다. 수행한 작업은 약 1ms의 지연 후에 0부터 15까지 카운트하여 그 결과를 이진법으로 LED에 표시하는 작업 1, Dhrystone 벤치마크를 실행하는 작업 2, 약 1ms의 지연 후에 스케줄 된 횟수를 출력하는 작업 3 등 3개로 구성하였다. 힙(heap)의 크기는 5000 bytes이며, 작업 1은 480 bytes, 작업 2는 2800 bytes, 작업 3은 480 bytes의 스택을 할당하였다. Modelsim을 이용하여 시뮬레이션한 결과는 Fig. 5와 같으며, 타이머 인터럽트(0x0a) 발생 시 FSM 및 제어 신호에 따라 인터럽트 처리 루틴의 시작 주소(0x1C008028)로 분기하는 것을 확인할 수 있다.

이를 FPGA에 포팅하여 프로세서 및 주변장치의 동작을 확인하였다. Fig. 6(a)와 같이 작업 3개가 번갈아 가며 스케줄링된 결과를 UART를 통해 확인할 수 있으며, Fig. 6(b)에서 작업 1의 실행 결과가 LED에 표시되는 것을 볼 수 있다.

Fig. 5. Modelsim Simulation Screen. It Shows the Flow of PC and Instructions of Interrupt Handling



(a) Task Scheduling Result on Terminal



(b) LED Displaying 13(2'b1101)

Fig. 6. FreeRTOS Porting Result on Zynq-7000 FPGA

5.3 벤치마크 실행을 통한 성능 측정

설계한 프로세서의 성능을 측정하기 위해 Dhrystone 및 Coremark 벤치마크를 실행하였다. GCC 7.1.1 버전을 이용하여 컴파일하였으며, 반복 횟수는 각각 5만 회, 1천 회이다.

본 논문에서는 3단계 파이프라인으로 구성된 소형, 저전력 프로세서인 Arm Cortex-M0의 DMIPS(Dhrystone Millions of Instructions Per Second)와 간접적으로 비교하고자 Dhrystone 벤치마크를 실행하였다. 또한, Dhrystone 벤치마크가 컴파일러에 의한 최적화 등으로 인해 코어의 특성을 정확히 반영한다고 보기 어려우므로 Coremark 벤치마크 또한 실행하여 성능을 비교하였다. Coremark는 EEMBC에서 Dhrystone을 대체하기 위해 개발한 벤치마크로, 리스트 처리 및 행렬 연산 등 산술논리 연산에 대한 성능 측정을 포함한다. 또한, 라이브러리를 호출하는 대신 벤치마크 프로그램에 포함하여 성능 측정의 정확도를 높였다고 할 수 있다. 앞서 언급한 타이머를 이용하여 시간을 측정하였고, 비교를 위해 PULPissimo SoC의 Riscy 및 Zero-riscy 프로세서에도 동일한 옵션으로 컴파일한 벤치마크를 실행하였다. 아래 Table 3은 Arm Cortex-M0 프로세서를 포함한 Dhrystone 및 Coremark 벤치마크 결과를 나타낸다.

Table 3. Benchmark Result

Processor	DMIPS/MHz	Coremark/MHz
In-house	**1.23**	**1.06**
Riscy	1.33	0.93
Zero-riscy	1.05	0.93
Cortex-M0	0.87	2.33

In-house, Riscy, Zero-riscy 프로세서의 경우 "-O2" 최적화 레벨로 컴파일한 바이너리를 실행하여 성능을 측정하였고, Cortex-M0 프로세서의 경우 Arm에서 제공하는 문서를 참고하였다[23]. 성능 측정 결과 제안하는 프로세서가 Zero-riscy 프로세서 대비 좋은 성능을 보였다. Dhrystone 벤치마크에서 Riscy 프로세서의 성능이 더 높은 이유는 4단계 파이프라인 구조를 통해 1 클럭 사이클에 메모리에 접근할 수 있기 때문일 것으로 판단된다.

Coremark 벤치마크의 컴파일 옵션에 따른 성능 변화를 알아보기 위해 최적화를 하지 않은 코드와 비교하여 실행 시간을 측정하였다. 컴파일러의 대표적인 최적화 옵션인 최적화 레벨은 "-On"으로 나타내며, n의 값에 따라 다른 최적화 옵션들을 적용한다[24]. "-O0" 옵션은 최적화 없는 실행 파일을 생성하며, "-O1" 옵션은 컴파일 시간을 고려하여 코드 크기 및 실행 시간을 최적화한다. 해당 최적화 옵션에 포함되지 않는 "-funroll-all-loops" 옵션은 가능한 경우 반복문을 풀어 분기를 줄이는 대신 코드 크기가 커질 수 있다. Fig. 7은 실험 결과를 나타낸 그래프로, 모든 최적화 옵션에 대해

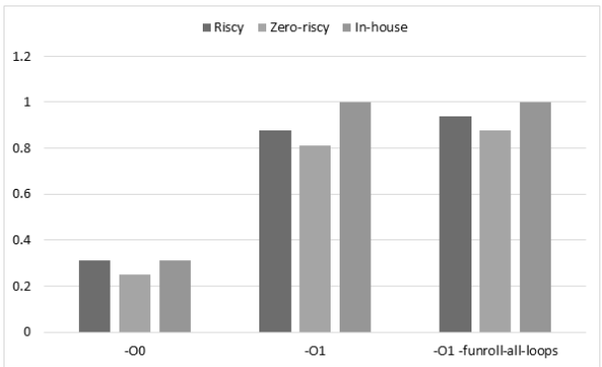


Fig. 7. Coremark Result with Different Optimization Flags

제안하는 프로세서의 벤치마크 결과가 가장 높았다. 최적화를 적용하면 연산량 및 메모리 접근이 최적화되어 벤치마크 성능이 개선되었다. "-funroll-all-loops" 옵션을 적용한 경우 파이프라인 flush로 인한 프로세서 대기를 줄일 수 있어 세 프로세서에서 모두 성능이 향상되었다.

세 프로세서 모두 일반적으로 1 클럭 사이클 당 1개의 명령어를 실행한다. 하지만 Riscy 프로세서의 경우 분기 시 2개의 명령어를 flush 해야 하고, Zero-riscy 프로세서의 경우 저장 명령어를 2 클럭 사이클에 수행한다. 따라서, 실험 결과 분기 및 저장 명령어를 각각 1 클럭 사이클에 수행 가능한 in-house 프로세서가 같은 2단계 파이프라인 구조의 Zero-riscy 프로세서보다 우수한 벤치마크 성능을 보였다. Riscy 프로세서에 대해서는 벤치마크에 따라 비교 결과가 달라진다. 하지만 사용하는 하드웨어 및 전력 대비 성능 면에서 제안하는 프로세서가 소형, 저전력 임베디드 시스템에 더 적합하다고 할 수 있다.

6. 결 론

본 논문은 RISC-V 정수형 명령어 집합 및 특권 명령어 집합을 실행 가능한 프로세서를 설계 및 평가하였다. 2단계 파이프라인으로 구성하였으며, 정수형 명령어 40개 중 38개의 명령어를 실행할 수 있다. 특권 명령어 집합 및 CSR을 구현하여 인터럽트를 처리할 수 있고, 클럭 게이팅을 통해 대기 상태에서의 전력 소모를 줄였다. 구현한 프로세서를 UART, GPIO, 타이머 등 주변장치와 함께 합성하여 차지하는 면적 및 소모 전력을 측정하였고, FPGA에서 FreeRTOS 포팅을 통해 동작을 검증하였다. 또한, 벤치마크 실행을 통해 성능을 검증하였고, Riscy 및 Zero-riscy 프로세서와 비교하여 제안하는 프로세서가 하드웨어 사용률 대비 성능이 뛰어남을 보였다. 이를 통해 스마트팜, 수질 관리 시스템 등 자원이 한정적인 분야에 적합한 저전력 고효율 RISC-V 프로세서를 확보하였다. 추후 파이프라인 단계 확장 및 명령어 집합 추가 등의 연구를 진행하여 고성능 프로세서로 확장하고, 이를 ASIC 칩으로 구현하는 방향을 모색할 예정이다.

References

[1] A. Waterman, K. Asanovi, and RISC-V Foundation, "The RISC-V instruction set manual, Volume I: User-Level ISA, document version 20191213", 2019.

[2] A. Waterman, K. Asanovi, and RISC-V Foundation, "The RISC-V instruction set manual, Volume II: Privileged architecture, document version 20190608-Priv-MSU-Ratified", 2019.

[3] Asanović, Krste, and David A. Patterson, "Instruction sets should be free: The case for risc-v," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146*, 2014.

[4] C. Chen, et al., "Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension: Industrial product," *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp.52-64, 2020.

[5] Western Digital, "RISC-V SweRV EH2 Programmer's Reference Manual Revision 1.4" [Internet], https://github.com/chipsalliance/Cores-SweRV-EH2, 2021.

[6] M. Gautschi, et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.25. No.10, pp.2700-2713, 2017.

[7] A. De, A. Basu, S. Ghosh, and T. Jaeger, "FIXER: Flow integrity extensions for embedded RISC-V," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE*, 2019.

[8] J. J. Lee, J. U. Park, M. J. Kim, and H. W. Kim, "Efficient ARIA cryptographic extension to a RISC-V processor," *Journal of The Korea Institute of Information Security & Cryptology*, Vol.31, No.3, pp.309-322, 2021.

[9] Y. K. Kwak, Y. B. Kim, and S. C. Seo, "Benchmarking Korean Block Ciphers on 32-Bit RISC-V Processor," *Journal of The Korea Institute of Information Security & Cryptology*, Vol.31, No.3, pp.331-340, 2021.

[10] PULPissimo Platform [Internet], https://github.com/pulp-platform/pulpissimo

[11] FreeRTOS™ [Internet], https://www.freertos.org

[12] R. P. Weicker, "Dhrystone: A synthetic systems programming benchmark," *Communications of the ACM*, Vol.27, No.10, pp.1013-1030, 1984.

[13] Coremark® [Internet], https://www.eembc.org/coremark

[14] P. Davide Schiavone et al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp.1-8, 2017.

[15] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.Wolf: An energy-precision scalable parallel ultra low power SoC for IoT edge processing," *IEEE Journal of Solid-State Circuits*, Vol.54, No.7, pp.1970-1981, 2019.
[16] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.27, No.11, pp.2629-2640, 2019.
[17] K. Asanovic, et al., "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.
[18] Celio, Christopher, David Patterson, and Krste Asanovic, "The Berkeley Out-of-Order Machine (BOOM) Design Specification," *University of California, Berkeley*, 2016.
[19] PicoRV32 [Internet], https://github.com/cliffordwolf/picorv32
[20] J. Huan, H. Li, F. Wu, and W. Cao, "Design of water quality monitoring system for aquaculture ponds based on NB-IoT," *Aquacultural Engineering*, Vol.90. No.102088, 2020.
[21] R. Senthilkumar, P. Venkatakrishnan, and N. Balaji, "Intelligent based novel embedded system based IoT enabled air pollution monitoring system," *Microprocessors and Microsystems*, Vol.77. No.103172, 2020.
[22] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem, "A survey on the role of IoT in agriculture for the implementation of smart farming," *IEEE Access*, Vol.7. pp.156237-156271, 2019.
[23] Arm Limited "Arm Cortex-M Processor Comparison Table" 2020 [Internet], https://developer.arm.com/documentation/102787/0100/.
[24] GCC, the GNU Compiler Collection [Internet], https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

장 선 경
https://orcid.org/0000-0002-7084-5573
e-mail : itcos294@korea.ac.kr
2020년 고려대학교 컴퓨터학과(학사)
2020년 ~ 현 재 고려대학교 컴퓨터학과 석사과정
관심분야 : Microprocessor Architecture, Hardware Security

박 상 우
https://orcid.org/0000-0002-5831-2176
e-mail : psw0113@korea.ac.kr
2017년 숭실대학교 컴퓨터학과(학사)
2019년 고려대학교 컴퓨터학과(석사)
2019년 ~ 현 재 고려대학교 컴퓨터학과 박사과정
관심분야 : AI Accelerator, Deep Learning, Computer Architecture 등

권 구 윤
https://orcid.org/0000-0001-6585-6748
e-mail : psw0113@korea.ac.kr
2021년 고려대학교 수학과(학사)
2021년 고려대학교 컴퓨터학과(학사)
2021년 ~ 현 재 고려대학교 반도체시스템공학과 석사과정
관심분야 : AI accelerator, DMA 등

서 태 원
https://orcid.org/0000-0002-6377-5482
e-mail : suhtw@korea.ac.kr
1993년 고려대학교 전기공학과(공학사)
1995년 서울대학교 전자공학과(공학석사)
2006년 Georgia Institute of Technology Computer Engineering(공학박사)
1995년 ~ 1998년 LG종합기술원 주임연구원
1998년 ~ 2001년 하이닉스반도체 선임연구원
2004년 ~ 2006년 Intel Corp. Research Intern
2007년 ~ 2008년 Intel Corp. Systems Engineer
2008년 ~ 현 재 고려대학교 컴퓨터학과 교수
관심분야 : 컴퓨터구조, 하드웨어 보안, AI accelerator, 블록체인 등