Journal of the Korea Academia-Industrial cooperation Society
Vol. 17, No. 3 pp. 699-707, 2016

http://dx.doi.org/10.5762/KAIS.2016.17.3.699
ISSN 1975-4701 / eISSN 2288-4688

낮은 복잡도의 Deeply Embedded 중앙처리장치 및 시스템온칩 구현

박성정[1], 박성경[2*]
[1]건국대학교 전자공학부, [2]부산대학교 전자공학과

Low-Complexity Deeply Embedded CPU and SoC Implementation

Chester Sungchung Park[1], Sungkyung Park[2*]
[1]**Department of Electronics Engineering, Konkuk University**
[2]**Department of Electronics Engineering, Pusan National University**

요 약 중앙처리장치를 중심으로 하는 각종 내장형 시스템은 현재 각종 산업에 매우 광범위하게 쓰이고 있다. 특히 사물인터넷 등의 deeply embedded (심층 내장형) 시스템은 저비용, 소면적, 저전력, 빠른 시장 출시, 높은 코드 밀도 등을 요구한다. 본 논문에서는 이러한 요구 조건을 만족시키는 중앙처리장치를 제안하고, 이를 중심으로 한 시스템온칩 플랫폼을 소개한다. 제안하는 중앙처리장치는 16 비트라는 짧은 명령어로만 이루어진 확장형 명령어 집합 구조를 갖고 있어 코드 밀도를 높일 수 있다. 그리고, 다중사이클 아키텍처, 카운터 기반 제어 장치, 가산기 공유 등을 통하여 로직 게이트가 차지하는 면적을 줄였다. 이 코어를 중심으로, 코프로세서, 명령어 캐시, 버스, 내부 메모리, 외장 메모리, 온칩디버거 및 주변 입출력 장치들로 이루어진 시스템온칩 플랫폼을 개발하였다. 개발된 시스템온칩 플랫폼은 변형된 하버드 구조를 갖고 있어, 메모리 접근 시 필요한 클락 사이클 수를 감소시킬 수 있었다. 코어를 포함한 시스템온칩 플랫폼은 상위 언어 수준과 어셈블리어 수준에서 모의실험 및 검증하였고, FPGA 프로토타이핑과 통합형 로직 분석 및 보드 수준 검증을 완료하였다. 0.18μm 디지털 CMOS 공정과 1.8V 공급 전압 하에서 ASIC 프론트-엔드 게이트 수준 로직 합성 결과, 50MHz 동작 주파수에서 중앙처리장치 코어의 논리 게이트 개수는 7700 수준이었다. 개발된 시스템온칩 플랫폼은 초소형 보드의 FPGA에 내장되어 사물인터넷 분야에 응용된다.

Abstract This paper proposes a low-complexity central processing unit (CPU) that is suitable for deeply embedded systems, including Internet of things (IoT) applications. The core features a 16-bit instruction set architecture (ISA) that leads to high code density, as well as a multicycle architecture with a counter-based control unit and adder sharing that lead to a small hardware area. A co-processor, instruction cache, AMBA bus, internal SRAM, external memory, on-chip debugger (OCD), and peripheral I/Os are placed around the core to make a system-on-a-chip (SoC) platform. This platform is based on a modified Harvard architecture to facilitate memory access by reducing the number of access clock cycles. The SoC platform and CPU were simulated and verified at the C and the assembly levels, and FPGA prototyping with integrated logic analysis was carried out. The CPU was synthesized at the ASIC front-end gate netlist level using a 0.18μm digital CMOS technology with 1.8V supply, resulting in a gate count of merely 7700 at a 50MHz clock speed. The SoC platform was embedded in an FPGA on a miniature board and applied to deeply embedded IoT applications.

Keywords : control unit, CPU, EISC, IoT, modified Harvard architecture, multicycle architecture, SoC platform

본 연구는 IDEC의 EDA Tool 또는 MPW 에서 지원하여 수행하였음.
이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2015R1D1A3A01018739).
*Corresponding Author: Sungkyung Park (Pusan National Univ.)
Tel: +82-51-510-2368 email: fspark@pusan.ac.kr
Received January 12, 2016 Revised (1st February 15, 2016, 2nd February 25, 2016, 3rd March 2, 2016)
Accepted March 3, 2016 Published March 31, 2016

1. 서론

차량, 선박, 모바일, 의료, 저장 장치, 보안, 영상, 산업 제어, 가정용 전자장치, 네트워크 등의 다양한 응용 분야의 수요가 나날이 증가하여, 내장형 프로세서나 마이크로컨트롤러는 오늘날 매우 널리 쓰이고 있다. 특히, deeply embedded (심층 내장형) 장치는 낮은 대역폭의 데이터 사용, 자주 반복되지 않는 데이터 획득 등에 쓰이는 장치나 기구를 의미하는데, 각종 계량기, 정보 디스플레이, 보안용 멀티미디어, 센서, 장난감 등 많은 응용 분야에 쓰이고 있다. 본 논문에서는 사물인터넷(IoT)용 [1] 정보 디스플레이 및 보안용 카메라, 의용생체공학 분야 (심박 모니터, 생체 측정용 스캐너 등), 네트워크 접속 스토리지, OSEK 기반 [2] 차량용 응용 분야 등의 deeply embedded 저가 시장을 고려한 프로세서 및 시스템온칩 개발을 다룬다. 이러한 deeply embedded 분야는 저비용, 낮은 복잡도, 작은 TTM (time to market), 소면적, 낮은 지연 (latency), 저전력의 중앙처리장치(CPU or central processing unit)를 필요로 한다.

본 논문에서는 파이프라이닝되지 않은 다중사이클 [3] CPU의 설계를 다룬다. 이러한 구조의 CPU는 저비용에서 (수백 MHz 이상의) 고성능을 보이기는 어렵지만, 파이프라이닝 기법에서 나타나는 각종 해저드 및 분기 처리 문제가 없고, 적은 게이트 개수와 낮은 복잡도, 그리고 저전력을 달성하기에 용이하다. 다중사이클 CPU는 파이프라인 CPU가 필요로 하는 해저드 검출 로직, 분기 예측 로직, 파이프라인 레지스터 등을 요구하지 않으므로, 구현 복잡도와 면적을 낮출 수 있다. 또한, 개발 기간 및 개발 비용도 단축할 수 있고, 100MHz 미만의 속도 성능을 요구하는 deeply embedded 시장에서는 고성능을 위한 파이프라인 구조의 CPU를 대체할 수 있다.

CPU는 데이터패스와 제어 장치로 구성되는데, 다중사이클 아키텍처의 CPU는 가산기 등의 데이터패스 로직을 서로 다른 동작 사이클마다 공유 및 다시 사용할 수 있다는 장점이 있다. 즉, 다중사이클 아키텍처에서는 동시 동작해야 하는 데이터패스 구성 블록의 개수가 파이프라인 아키텍처의 경우에 비해 적어진다는 장점이 있다. 반면에 다중사이클 구조의 단점은 제어 장치 (control unit) 구조가 복잡해지기 쉽다는 것이다. 그래서 본 논문에서는 일반적인 유한 상태 머신 (FSM) 제어 장치 대신 카운터 기반 제어 장치를 이용하여 제어 장치의 면적 및 복잡도를 줄였다.

본 논문에서는 deeply embedded 사물인터넷 시스템에서 요구하는 소면적, 저전력, 높은 코드 밀도, 저비용, 저복잡도, 짧은 개발 시간, 그리고 낮은 지연 시간을 갖는 CPU와 SoC 플랫폼을 개발하였다. 위의 여러 요구조건을 만족하기 위해 16 비트 길이의 짧은 명령어들만 사용하여 코드 밀도를 높였고, 다중사이클 구조, 가산기 공유, 그리고 카운터 기반 제어 장치 등을 이용하여 낮은 구현 복잡도를 성취하였다. 그리고 명령어 집합 구조 (ISA) 중 확장형 구조 (extendable instruction set architecture or EISC) [4] 채택으로 코드 밀도를 더욱 높였다. 임베디드 시스템에서 코드 밀도를 높이는 것은 메모리 용량을 확보하기 위해 중요하다.

설계된 프로세서는 CPU, simple MMU (memory management unit), 인터럽트 장치 및 코프로세서를 포함하고 있다. 프로세서를 감싸고 있는 사물인터넷용 시스템온칩 (SoC) 플랫폼도 개발하였는데, 이 플랫폼은 AMBA 버스 기반으로서, 변형된 하버드 아키텍처를 갖고 있다. 즉, 내부 SRAM과 명령어 캐시는 로컬 버스로 연결하고, 나머지 IP들은 AMBA 시스템 버스로 연결하여, 메모리 접근 시 소모되는 클락 사이클 개수를 줄였다. 설계한 명령어 캐시는 파이프라인 구조를 갖고 있고, 섹터 개념을 도입하여 하드웨어 구현 비용을 절감하였다.

개발된 플랫폼은 초소형 보드 상의 FPGA에 내장되어 다양한 deeply embedded 사물인터넷 분야에 응용된다. CPU 및 SoC는 FPGA에 내장되어 상세 검증 및 프로토타이핑을 거쳤다. ASIC을 위하여 CPU를 0.18μm 디지털 CMOS 공정과 1.8V 공급 전압 하에서 프론트-엔드 로직 게이트 수준으로 합성한 결과, 7700 수준의 낮은 게이트 개수를 가짐을 보였다.

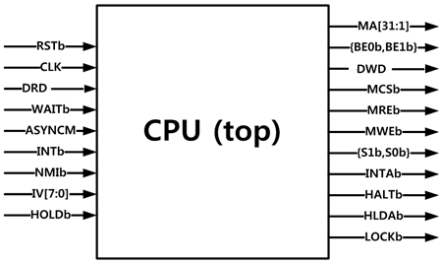
본 논문의 구성은 다음과 같다. 제 2장에서는 CPU 및 SoC 플랫폼의 설계와 ASIC 게이트 수준 합성에 대해 기술한다. 즉, 2.1절에서는 제안하는 CPU에 대해 설명하고, 2.2절에서는 제안하는 제어 장치에 대해 기술한다. CPU의 ASIC 게이트 수준 합성 및 성능 평가는 2.3절에 나타나 있다. 변형된 하버드 아키텍처 기반의 SoC 플랫폼에 대해서는 2.4절에서 설명하고, 설계한 명령어 캐시에 대해서는 2.5절에서 기술한다. 제 3장에서는 개발된 CPU 및 SoC 플랫폼의 검증 및 FPGA 프로토타이핑에 관해 기술하고, 제 4장에서 결론을 맺는다.

2. CPU 및 SoC 플랫폼의 설계와 ASIC 게이트 수준 합성

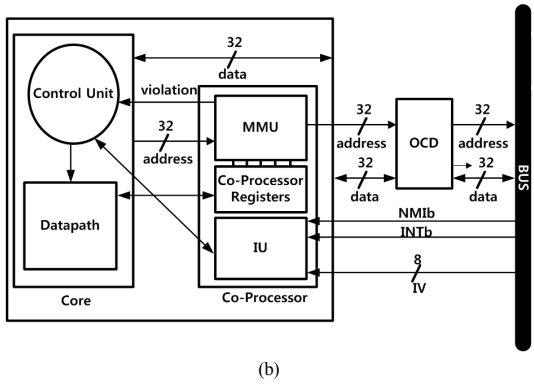
2.1 중앙처리장치 (CPU)

본 논문의 CPU는 32 비트 길이 명령어의 MIPS나 ARM과 달리, 확장형 명령어 집합 구조 (EISC) 채택으로 16 비트의 짧은 명령어들로 구성돼 있고, LERI 명령어, 32 비트 확장 레지스터 (ER or extension register) 및 1 비트 확장 플래그 (E-flag) [4] 채용으로 큰 즉치 값 (상수) 연산들을 효과적으로 처리할 수 있다. 즉, 큰 즉치 값이 요구되는 연산이나 분기 명령 직전에 한 개 이상의 LERI (load ER with immediate) 명령을 수행시켜, LERI 명령이 수행될 때마다 즉치 값을 ER에 순서대로 저장하고, 이를 알려주는 E flag를 발생시킨다. 이렇게 하여 ER에 저장된 큰 값의 누적 즉치 값을 관련 연산에서 이용할 수 있게 한다. 그 외의 ISA 구조는 기본적으로 RISC와 동일하다고 볼 수 있다. 짧은 16 비트 명령어들만 씀으로써 코드 밀도를 높일 수 있는데, 이는 임베디드 메모리에서 중요하다. 설계한 CPU의 명령어 집합 구조 (ISA) 속 명령어 타입 개수는 146개이고, 보통의 곱셈 연산 외에 암호화 연산 관련 Galois-field 곱셈 연산이 더 들어가 있는 게 특징이다.

그림 1(a)에는 CPU 최상단의 입출력 핀이 보이는데, 리셋, 클락, 데이터, 인터럽트, NMI (non-maskable interrupt), 메모리 주소 및 byte enable, chip select, read/write, 기타 부가적인 신호들이 표시되어 있다. 코어와 코프로세서(co-processor)로 구성된 CPU의 간략한 블록도는 그림 1(b)에 있는데, 주소와 데이터는 온칩디버거(OCD)와 버스를 거쳐 메모리와 주변 입출력 장치로 전달된다. 코어는 데이터패스와 제어 장치로 다시 나뉘고, 코프로세서는 간략한 메모리 관리 장치, 인터럽트 벡터 (IV) 처리용 인터럽트 장치 (interrupt unit 혹은 IU) 및 20여 개의 레지스터로 세분화할 수 있다.



(a)



(b)

Fig. 1. CPU interface and block diagram
(a) I/O pins of the CPU top
(b) Simplified block diagram of the CPU

코어 속 데이터패스는 12개의 먹스 (multiplexer 혹은 mux), 19개의 레지스터, ALU (arithmetic logic unit), 프로그램 카운터 (PC) 가산기, 그리고 곱셈 연산기 등으로 구성돼 있다. 하드웨어 면적을 줄이기 위해 레지스터 파일 (RF) 포트를 3개로 제한했고, MOVE 명령어를 ALU 경로로 흡수하였으며, PC와 무관한 주소 계산도 PC 가산기에서 처리하게 했고 (가산기 공유), 다중사이클 아키텍처를 사용했으며, 카운터 기반 제어 장치를 도입하였다.

2.2 카운터 기반 제어 장치

CPU를 크게 데이터패스와 제어 장치로 나눌 수 있는데, 이 중 제어 장치는 CPU 구조에 따라 조금 달라진다. 제어 장치 중 핵심적인 부분이 명령어 해독기인데, 파이프라인 CPU의 경우에는 각 명령어 당 제어 신호가 명령어 해독 단에서 한꺼번에 생성돼 다음 단으로 전달된다. 즉, 파이프라인 단 사이에 필요한 파이프 레지스터들만큼 하드웨어 면적이 더 커지기는 하지만, 명령어 해독기는 덜 복잡하다. 파이프라인이 없는 단일 사이클 CPU의 경우에도 명령어 당 상태가 하나이므로 파이프라인 CPU의 명령어 해독기와 유사해진다. 하지만 본 논문에서 낮은 복잡도를 위해 채택한 파이프라인 없는 다중사이클 CPU의 경우, 명령어 해독기에서 나오는 명령어 별 제어 신호들은 그 명령어의 클락 사이클마다 나오게 된다. 이를 위해 일반적으로는 FSM을 이용하여 명령어 당 각 사이클을 현재 상태와 다음 상태로 표현할 수 있다. 그 외 마이크로프로그래밍 방식[5]의 명령어 해독기도 있지만 이는 큰 면적을 차지하기 마련이므로 선택에서

제외한다.

본 논문에서는 카운터 기반 제어 장치를 설계하여 다중사이클 CPU의 제어 장치를 덜 복잡하게 만들고 면적을 줄였다. 그림 2(a)와 같은 일반적인 FSM 방식 제어 장치에서는 다음 상태가 현재 상태와 명령어에 의해 결정되고, 제어 신호들은 조합 논리에 인가되는 현재 상태에 의해 생성된다. 한편, 그림 2(b)와 같은 카운터 기반 제어 장치에서는 단순한 카운터와 먹스를 이용해 각 명령어의 사이클을 센다. 각 명령어의 마지막 사이클 직후 카운터는 리셋된다. 설계한 CPU의 경우에는 halt_b와 wait_b 신호가 모두 de-assert될 때 카운터가 정상적으로 카운팅된다. (halt_b 신호는 halt 명령어가 수행될 때 assert되고, wait_b 신호는 메모리로부터 외부 wait 신호를 받았을 때 assert된다.) 디코더 로직을 써서 상태들을 서로 구분할 수도 있지만, 그림 2(b) 구조처럼 명령어와 count 신호를 출력 로직 블록으로 바로 인가하게 되면, 디코더를 쓸 필요가 없어 면적을 더 줄일 수 있다. 그림 2(a) 구조는 Moore machine인데 비해, 그림 2(b) 구조는 Mealy machine이라 상태 개수 측면에서 그림 2(b)가 좀 더 낫다.

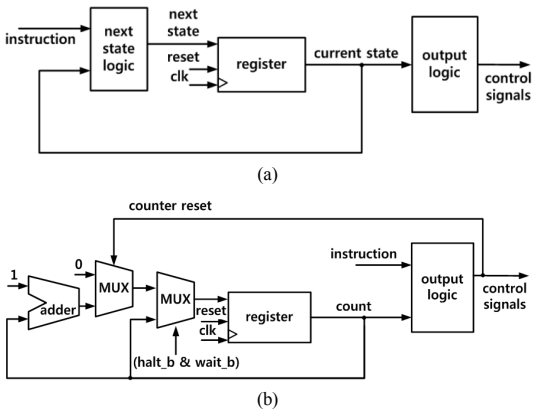


Fig. 2. Two types of control units for the CPU core
(a) General FSM-based control unit
(b) Proposed structure of the counter-based control unit

2.3 ASIC 게이트 수준 합성

그림 2의 두 가지 제어 장치의 구현 복잡도를 비교해 보기 위해 먼저 좀 더 간단한 다중사이클 CPU용 카운터 기반 제어 장치의 합성 ASIC 게이트 개수를 측정해 보았다. 즉, 26 종류의 제어 신호, 42 비트짜리 제어 장치 출력 신호, 77개 명령어 타입을 갖는 4 비트 다중사이클 CPU를 설계해 게이트 개수를 측정해 보았다. 총 165개의 상태가 존재하였고, 일반적인 FSM 방식 제어 장치의 경우 최소 16개의 플립-플롭(레지스터)이 요구되는 반면에, 카운터 기반 제어 장치의 경우 (명령어 당 사이클 개수가 최대 11이었으므로) 4개 플립-플롭만 있으면 된다. 0.18μm 디지털 CMOS 표준 라이브러리 셀과 1.8V 공급 전압 하에서 ASIC 게이트 수준 합성을 했을 때의 두 가지 제어 장치에 대한 성능을 표 1에 나타냈다. 게이트 개수는 2.3배 감소하고, 레이턴시도 감소함을 알 수 있다. 카운터 기반 제어 장치를 쓸 경우 일반적인 FSM 기반 제어 장치를 쓸 경우와 비교할 때, CPU 전체에서 제어 장치가 차지하는 면적 비중은 44%에서 25%로 줄어든다.

암호화용 Galois-field [6] 곱셈 연산까지 들어있는 본 논문의 16 비트 명령어 EISC CPU에 대해서도 두 제어 장치의 게이트 개수를 비교해 보았다. 설계된 코어는 146개의 명령어 종류를 갖고 있고, 총 상태 개수는 600개 남짓이었다. 일반적인 FSM 방식 제어 장치에서는 20개의 플립-플롭이 필요하지만, 카운터 기반 제어 장치에서는 (명령어 당 사이클 개수의 최대값이 17이었으므로) 5개의 플립-플롭만 있으면 된다. 제어 신호의 종류는 72가지였고, 사이클 당 출력 제어 신호의 폭은 총 97 비트였다. 표 2에서 볼 수 있듯이, 일반적인 FSM 기반 제어 장치를 쓸 경우에 비해, 카운터 기반 제어 장치를 사용하면 게이트 개수를 1800 정도 절약할 수 있었다. 면적 비중은 38%에서 27%로 줄일 수 있었다.

Table 1. Gate-level netlist synthesis results of the general FSM-based and the counter-based control units in application to the 4-bit deeply embedded CPU operating at 10MHz.

Control unit	Gate count	Power (μW)	Latency (ns)
General FSM-based	920	70	6.8
Proposed counter-based	390	60	3.5

Table 2. Gate counts of the general FSM-based and the counter-based control units in application to the 32-bit deeply embedded CPU operating at 50MHz.

Control unit	Gate count	Area percentage
General FSM-based	4800	38
Proposed counter-based	3000	27

설계된 CPU의 전력 소모는 50MHz에서 2.3mW, 10MHz에서 약 0.5mW였고, 게이트 개수는 10MHz-50MHz에서 약 14000이었다. 코프로세서 속 20여 개 레지스터와 코어 속 곱셈기를 제외하면 게이트 개수는 7700 정도밖에 되지 않는다. 반면에 ARM7과 이의 변종 [7] 구조는 게이트 개수가 각각 50600과 20200 이상이었다. 위에서 언급한 카운터 기반 제어 장치, 다중사이클 아키텍처, 가산기 공유 등을 통하여 CPU의 구현 복잡도와 면적을 줄일 수 있었다.

2.4 변형된 하버드 아키텍처 기반 SoC 플랫폼

Deeply embedded IoT용 CPU에서 요구되는 데이터 메모리는 비교적 작은 용량이므로, 메모리 접근 시간 3ns 내외인 동기형 (synchronous or sync) 내부 SRAM을 프로세서(CPU)에 로컬 버스로 직접 연결하였고, 프로세서의 출력은 내부 SRAM을 위한 영역과 이 영역 외의 영역으로 구분하였다. 내부 SRAM은 주로 연산 중 발생하는 데이터의 읽기/쓰기를 위해 사용되고, 필요한 경우 자주 사용되는 명령어를 저장해 둘 수 있으며, 가끔은 명령어를 인출하는 용도로 사용되기도 한다. 기존 하버드 구조의 프로세서 출력이 명령어 인출을 위한 I-영역과 데이터 읽기/쓰기를 위한 D-영역으로 구분되는 것과 달리, 본 논문의 변형된 하버드 구조에서는 프로세서 출력이 내부 SRAM을 위한 영역과 그 외의 영역으로 구분된다.

일반적으로 레지스터 개수가 16개인 프로세서에서는 메모리 읽기/쓰기 관련 명령어의 출현 빈도가 25% 내외여서, 메모리 접근에 소요되는 시간이 프로세서 성능에 큰 영향을 미친다. 기존 하버드 구조에서 데이터 캐시 또는 tightly coupled memory (TCM) 형태를 사용하지 않는 경우, 프로세서는 시스템 버스를 통해 데이터 메모리에 접근하게 되는데, 이 경우 프로세서가 시스템 버스에 접근한 후 메모리에서 값을 읽어오기까지 시간이 오래 걸린다. 하지만, 변형된 하버드 구조에서는 내부 SRAM을 통한 단일 사이클 메모리 접근이 가능하다. 그림 3은 변형된 하버드 구조의 간략화된 블록도이다. 프로세서의 출력은 SRAM을 위한 D-bus, SRAM 이외의 영역을 위한 I-bus 및 B-bus로 각각 연결되고, I-bus는 CPU를 명령어 인출용 I-cache 혹은 PROM으로 연결해 준다. B-bus는 시스템 버스로 연결되고, I/O등의 장치를 위해 사용된다. 변형된 하버드 구조도 일반적인 하버드 구조처럼 데이터 버스와 명령어 버스가 물리적으로 분리돼 있기는 하지만, 데이터 버스가 다시 시스템 버스와 (프로세서에 직접 연결된) 내부 데이터 메모리 버스로 구별된다.

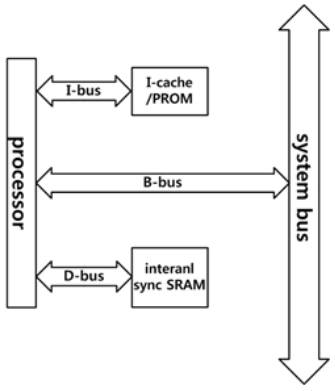


Fig. 3. Modified Harvard architecture

설계한 SoC 플랫폼의 상세 블록도는 그림 4에 도시돼 있다. 플랫폼은 정수 연산용 CPU 코어, 코프로세서(CP), 16MB 용량의 내부 sync SRAM, 1KB 용량의 명령어 캐시 (I-cache), BIU, AHB 버스, NOR 플래시 메모리 외 여러 IP들로 구성된다. AHB 버스를 통해 연결된 주변 입출력 장치로는 타이머, UART, 스위치 컨트롤러, LED 컨트롤러 등이 있다. 인터럽트 (예외상황) 컨트롤러는 (deeply embedded 시스템에서 중요한) 낮은 레이턴시 달성을 위해 코어에 직접 연결하였다.

그림 4의 코어에서 생성된 명령어 주소와 데이터 주소는 코프로세서 속 MMU로 들어간다. 코어는 주소 변환 시 필요한 정보를 미리 코프로세서 레지스터들(CRs)에 저장해 놓는다. CPU 주소는 MMU에 의해 외부 메모리 물리주소(physical address)로 변환(mapping)되고, priority logic에 의해 중재돼 SRAM 컨트롤러, I-cache 컨트롤러 혹은 BIU로 전달된다. I-cache 컨트롤러는 물리주소를 받고, 코어에서 다음 클락에 사용될 가상주소를 받아 캐시 히트/미스를 판별한다. 캐시 히트일 경우 I-cache 메모리에서 명령어를 읽어, 코프로세서를 거쳐 코어로 전송한다. 캐시 미스일 경우 미스가 발생한 주소를 PROM 컨트롤러를 거쳐 외부 NOR 플래시 메모리로 전달하고, 해당 명령어를 읽어 와 I-cache 컨트롤러를 경유하여 I-cache 메모리에 저장한다. 이러한 과정을 반복해 (해당 섹터의) 모든 캐시 엔트리가 교체되면 명령어가 integer core로 전송된다. PROM 컨트롤러에서는 캐

시 미스 시에 캐시 엔트리 교체 시간을 단축시키기 위한 버스트 모드를 지원한다. BIU는 물리주소를 전달 받고, 이를 AHB 버스 전송 타이밍에 맞춰 Haddr를 통해 버스로 전송한다. AHB는 특정 주변 I/O 장치에게 주소를 전달하고, 이에 대응하는 데이터를 받는다. 데이터는 Hdata를 통해 BIU로 가고, 다시 코프로세서를 경유해 코어로 전송된다.

이러한 흐름으로 프로그램을 수행하다가 예외상황이 발생하면, 코어는 INT 신호를 받게 된다. 코어는 예외 발생 주소 및 관련 데이터를 CRs에 저장하고 INTA신호를 인터럽트 컨트롤러에 전달한다. 인터럽트 컨트롤러는 8 비트 IV(인터럽트 벡터)를 코어로 전송하고, 코어는 ISR(interrupt service routine)을 수행한다.

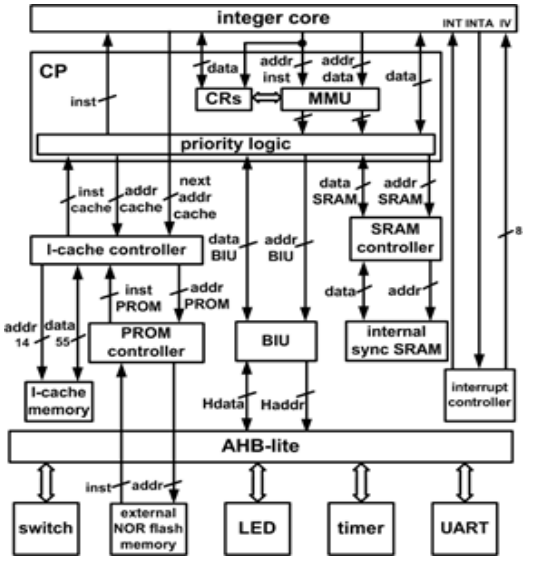


Fig. 4. Detailed block diagram of the SoC platform

2.5 명령어 캐시

캐시는 주 메모리와 코어 사이의 속도 차이를 보상[8]하여 전체 시스템의 성능을 높여 준다. 본 논문에서의 명령어 캐시 (I-cache) 구조는 직접 사상 (direct mapped) 구조이고, 1kB 용량의 sync SRAM 메모리를 캐시 메모리로 사용했다. 특정 명령어를 통해 캐시의 탈부착이 가능하게끔 설계했다. 본 논문의 명령어 캐시에는 섹터링 기법을 도입해 태그 메모리가 차지하는 면적 및 비용을 줄였다. 캐시 메모리를 16개의 섹터로 구분하였고, 섹터 당 4개 라인, 라인 당 4개 워드가 들어가게 했다. 하나의 섹터에는 태그 및 valid 비트가 하나씩 존재한다. 즉, 섹터링 기법을 써서 태그 메모리 크기를 1/4로 줄였다. 섹터링에 의해 캐시 엔트리 교체 시 2×4×4=32개의 명령어를 burst 모드로 캐시 메모리에 업데이트하게 된다.

명령어 캐시는 히트 발생 시 지연 시간 없이 정상적인 2단 파이프라인이 진행되도록 설계했다. (캐시 접근 시 항상 한 사이클의 지연 시간이 생기는데, 이를 해결하기 위해 캐시에 파이프라인 기법[9]을 적용했다.) 미스 발생 시에는 34 사이클 (= valid 비트 클리어용 1 사이클 + 캐시 엔트리 교체 지연 32 사이클 + 메모리 접근 1 사이클) 오버헤드가 발생한다.

코어에서는 다음 사이클에 인출할 명령어 주소가 담긴 (그림 4의) next addr cache를 생성하여 (MMU를 통과하지 않은) 가상주소 형태로 캐시 메모리에 전달한다. 이는 한 클락 사이클 이전에 캐시 태그 메모리 및 캐시 데이터 메모리에 접근하여, 다음 사이클에 지연 없이 명령어를 읽을 수 있도록 하기 위함이다. 동시에 캐시는 현재 사이클에서 MMU 및 priority logic을 통과한 물리주소(그림 4의 addr cache)를 받아, 태그 매칭을 통해 히트/미스 판별을 한다. 즉, 캐시 메모리 접근 속도 향상을 위해 virtually indexed physically tagged (VIPT) [10-11] 형태로 구현했다.

I-cache 컨트롤러는 그림 5와 같이 5가지 상태의 FSM으로 기술할 수 있다. NORMAL 상태는 히트 또는 캐시가 동작하지 않는 상태를 나타낸다. INVAL 상태는 코어에서 명령을 받아 초기화를 수행하기 위해 메모리에 접근하는 상태이다. CLEAR 상태는 해당 섹터의 valid 비트를 클리어하는 상태이다. FILL 상태는 캐시 미스 시 해당 섹터를 교체하는 상태이다. REMATCH 상태는 캐시 미스가 일어나 해당 섹터를 모두 교체한 뒤에 히트/미스 판별을 위한 태그 비교를 실시하는 상태다.

초기화된 캐시는 NORMAL 상태를 유지하고, 첫 번째 명령어가 인출될 때 캐시 미스가 일어난다. 이때 캐시 컨트롤러는 CLEAR 상태로 천이되어, 미스가 일어난 섹터의 valid 비트를 0으로 클리어한다. 이후 FILL 상태로 천이되어 외부 PROM에 명령어를 요청하게 된다. 해당 섹터에 모든 명령어를 채우고 나면 fill_end 신호가 1이 되고, 캐시 컨트롤러는 REMATCH 상태로 천이돼 히트/미스 판별을 수행한다. 이후 항상 NORMAL 상태로 천이되고, 히트일 경우에는 캐시 메모리에서 명령어를 인출하여 CPU로 전달하고, 미스일 경우에는 다시 CLEAR 상태로 천이된다.

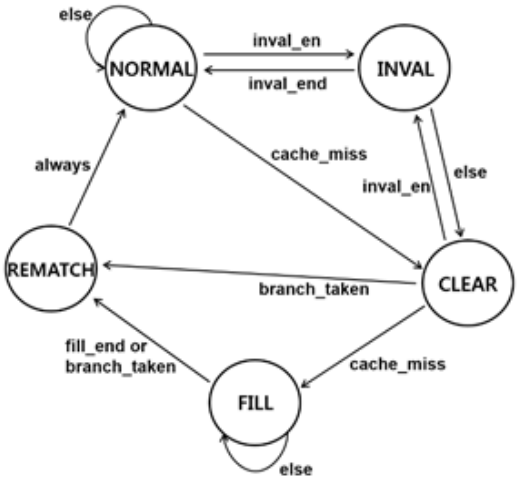


Fig. 5. State transition diagram of the finite state machine (FSM) for the cache controller

잘못된 데이터의 전달을 막기 위해 캐시를 초기화할 필요성이 있는데, 명령어 캐시는 캐시 엔트리 속 vaild 비트를 0으로 클리어시킴으로써 초기화를 수행한다. Dynamic binary translation (DBT) 지원을 위해 명령어 캐시에 self-snooping 기능[12]을 넣을 수 있는데, 이 기능을 추가하게 되면 회로의 복잡도가 증가하므로, 대신 invalidate 명령어를 두어서 DBT로 인해 생성된 기계어 코드를 수행하기 전에 캐시 전체를 invalidate시키는 것으로 해결한다.

그림 5에서 프로세서가 invalidate enable 신호(inval_en)를 캐시 컨트롤러에 전달하게 되면, 캐시 컨트롤러는 INVAL 상태로 천이되고, invalidate를 수행하려는 태그를 비교한다. 태그가 일치하면 CLEAR 상태로 전환하여 vaild 비트를 0으로 클리어한다. 본 논문의 캐시에서는 캐시 전체를 invalidate하기 때문에, 모든 섹터의 vaild 비트를 클리어할 때까지 INVAL 상태와 CLEAR 상태를 반복해 수행한다 (32 사이클 소모). 모든 섹터가 invalidate되면 inval_end 신호가 1이 되어 NORMAL 상태로 돌아가게 된다.

3. 검증 및 FPGA 프로토타이핑

설계 개발한 CPU 및 SoC 플랫폼의 각 IP를 모의실험, 검증하였다. 또한, 명령어의 타입 테스트, 벤치마킹 테스트, GDB 검증, 캐시 유무에 따른 컴파일러와 OCD (온칩디버거) 통합 보드 검증까지 실시하여 시제품화를 위한 준비를 마무리하였다. SoC 플랫폼은 우선 하드웨어 수준의 모의실험을 수행하였다. 어셈블리 수준에서 프로그램을 작성하여 특정 동작의 수행을 검증하는 것을 먼저 수행하였다. 작성된 검증 프로그램은 16진수 코드로 변환하여 SoC 플랫폼의 프로그램 메모리 모델에 초기화시켰다. ISS (명령어 집합 시뮬레이터) 개발 후 하드웨어 모의실험과의 병렬적 검증을 통해 검증의 효율성을 높였다.

그림 6은 SoC 플랫폼의 검증 시나리오의 한 예를 나타내고 있다. 프로세서 래퍼는 코어, 코프로세서, 내장 sync SRAM, 명령어 캐시 및 BIU를 포함한다. 절차는 다음과 같다. (1) 타이머로부터 발생한 인터럽트가 인터럽트 컨트롤러로 전달된다. (2) 인터럽트 컨트롤러에서 코어로 INT 신호와 IV[7:0] 신호가 전송된다. (3) 코어는 INTA (INT acknowledge) 신호를 전송한다. (4) UART를 이용해 문자열을 출력하는 ISR을 수행한다. 스위치 입력이 가해지지 않으면 (1)-(4)를 반복 수행한다. (5) 문자열 출력 ISR 수행 중 스위치가 입력되면 인터럽트가 발생돼 인터럽트 컨트롤러로 전달된다. (6) 코어로 INT와 IV[7:0]가 전달되지만, 코어는 문자열 출력 ISR을 수행하는 도중이므로 인터럽트를 받지 못한다. 문자열 출력 ISR이 종료됨과 동시에 코어는 스위치에 대한 INT 신호와 IV[7:0]을 받는다. (7) INTA가 인터럽트 컨트롤러로 전송된다. (8) 스위치 인터럽트에 대해 LED를 점멸하는 ISR이 수행된다.

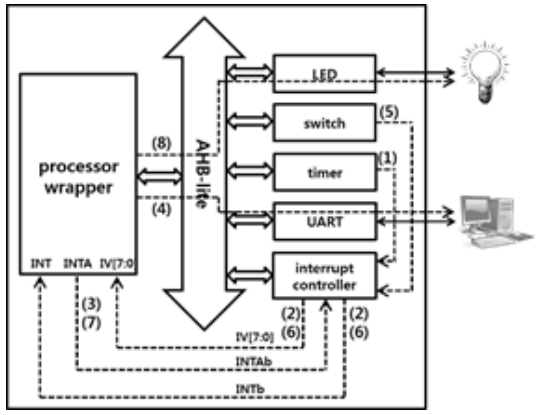


Fig. 6. SoC verification scenario example

FPGA 프로토타이핑은 오류 발생 시 즉시 수정이 가능하고, 개발기간이 짧으며, 초기 개발비용이 저렴하다.

본 논문에서 설계한 CPU 및 SoC 플랫폼은 FPGA 프로토타이핑[13-14]을 수행하였다. 통합 (내장) 로직 분석기를 통해 정상 동작하는 플랫폼의 파형을 관찰했고, UART를 통해 검증 결과를 확인하였다. 그림 7은 검증 시나리오가 정상적으로 수행된 후의 UART와 LED 출력이고, 그림 8은 검증 시나리오가 정상적으로 수행된 후의 내장형 로직 분석기 출력 파형으로서, 저장된 명령어와 현재 PC 값을 나타내고 있다.

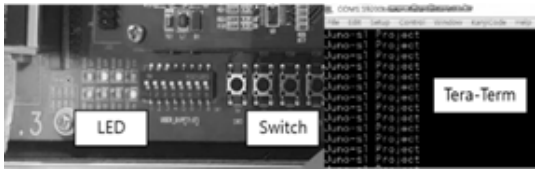


Fig. 7. UART and LED outputs for the FPGA prototype of the SoC platform

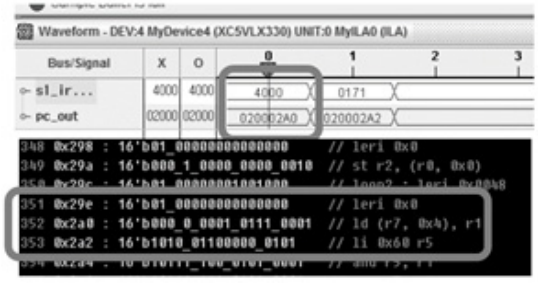


Fig. 8. Integrated logic analysis for functionality verification of the SoC platform

설계 및 개발된 CPU와 SoC 플랫폼에 대하여 온칩디버거(OCD)를 이용하여 보드 수준 검증 및 에뮬레이션을 완료하였고, GDB 소프트웨어 디버거, 컴파일러까지 포함한 통합 검증, Dhrystone, CoreMark, EEMBC의 각종 벤치마킹 테스트도 완료하였다.

4. 결론

본 논문에서는 EISC 기반 deeply embedded CPU를 제안하였고, CPU 및 제어 장치 설계와 SoC 플랫폼 개발에 대해 기술하였다. 사물인터넷용으로 설계된 CPU는 그 목적에 맞는 작은 면적, 낮은 복잡도, 저전력, 짧은 개발 시간, 낮은 지연 처리 시간, 높은 코드 밀도를 달성하기 위해 다중사이클 아키텍처, 카운터 기반 제어 장치, 가산기 공유 등을 특징으로 하고, 16 비트의 짧은 고정 길이 명령어 집합을 사용하였다. 개발된 SoC 플랫폼은 코프로세서, 명령어 캐시, 내부 SRAM, 버스 및 입출력 장치 등으로 구성돼 있고, 변형된 하버드 구조를 채택해 deeply embedded 분야에 적합하게 하였다. 설계된 CPU 및 SoC 플랫폼은 컴파일러/OCD 통합 검증을 거쳐 FPGA로 프로토타이핑하였고, CPU 성능 측정을 위해 ASIC 게이트 수준으로 합성하였다. 개발된 플랫폼은 아두이노 우노와 동일 크기의 초소형 보드 상의 FPGA에 내장하여 사물인터넷용으로 시제품화하는 단계에 있다.

References

[1] M. Wright, “Deeply Embedded Devices: The Internet of Things,” *Electronic Design*, http://electronicdesign.com/energy/deeply-embedded-devices-internet-things, Sep. 23, 2009.

[2] Y. Sun, W.-L. Huang, S.-M. Tang, X. Qiao, and F.-Y. Wang, “Design of an OSEK/VDX and OSGi-Based Embedded Software Platform for Vehicular Applications,” *IEEE International Conference on Vehicular Electronics and Safety*, pp. 1-6, Dec. 2007. DOI: http://dx.doi.org/10.1109/ICVES.2007.4456366

[3] D. A. Patterson and J. L. Hennessy, Computer Organization and Design, Elsevier, Morgan Kaufmann Publishers, 2010, 4th Ed.

[4] Extendable Instruction Set Computer, http://en.wikipedia.org/wiki/Extendable_instruction_set_computer, Wikipedia.

[5] B. Parhami, Computer Architecture, Oxford University Press, New York, NY, USA, 2005.

[6] H. Eberle, A. Wander, N. Gura, S. Chang-Shantz, and V. Gupta, “Architectural Extensions for Elliptic Curve Cryptography over GF(2^m) on 8-Bit Microprocessors,” *IEEE Proceedings of the 16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP’05)*, pp. 343-349, July, 2005. DOI: http://dx.doi.org/10.1109/ASAP.2005.15

[7] F.-C. Yang and I.-J. Huang, “An Embedded Low Power/Cost 16-Bit Data/Instruction Microprocessor Compatible with ARM7 Software Tools,” *Asia and South Pacific Design Automation Conference*, Yokohama, Japan, pp. 902-907, Jan. 2007. DOI: http://dx.doi.org/10.1109/aspdac.2007.358104

[8] A. Asaduzzaman, “An Efficient Memory Block Selection Strategy to Improve the Performance of Cache Memory Subsystem,” *14th International Conference on Computer and Information Technology*, pp. 22-24, Dec. 2011. DOI: http://dx.doi.org/10.1109/iccitechn.2011.6164798

[9] Chih-Wen Hsueh, Jen-Feng Chung, Lan-Da Van, and Chin-Teng Lin, “Anticipatory Access Pipeline Design for Phased Cache,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 18-21, May, 2008.

DOI: http://dx.doi.org/10.1109/ISCAS.2008.4541924

[10] Peter Petrov and Daniel Tracy, "Energy-Efficient Physically Tagged Caches for Embedded Processors with Virtual Memory," *Proceedings of 42nd Design Automation Conference*, pp. 17-22, 2005.
DOI: http://dx.doi.org/10.1109/dac.2005.193765

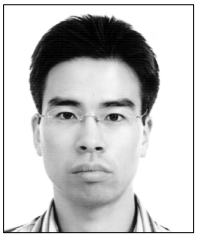
[11] Long Zheng, Mianxiong Dong, Song Guo, Minyi Guo, and Li Li, "I-Cache Tag Reduction for Low Power Chip Multiprocessor," *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 196-202, 2009.
DOI: http://dx.doi.org/10.1109/ispa.2009.85

[12] R. V. Batchu and D. A. Jumenez, "Exploiting Procedure Level Locality to Reduce Instruction Cache Misses," *Proceedings of the Eighth Workshop on Interaction between Compilers and Computer Architectures*, pp. 75-84, 2004.
DOI: http://dx.doi.org/10.1109/intera.2004.1299512

[13] Cillani Chayoor Abbas, Yian Zhu, Amjad Hafiz Muhammad, Ahmad Waqar, and Jianfeng An, "Backplane Bus Controller Implementation in FPGA for Hard Real Time Control Systems," *3rd International Conference on Communication Software and Networks*, pp. 451-456, May, 2011.
DOI: http://dx.doi.org/10.1109/iccsn.2011.6013870

[14] Chun-Ming Huang, Chien-Ming Wu, Chih- Chyau Yang, Wei-De Chien, Shih-Lun Chen, Chi-Shi Chen, Jiann-Jenn Wang, and Chin-Long Wey, "Implementation and Prototyping of a Complex Multi-Project System-on-a-Chip," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2321-2324, May, 2009.
DOI: http://dx.doi.org/10.1109/ISCAS.2009.5118264

박 성 정(Chester Sungchung Park) **[정회원]**



- 2000년 2월 : 한국과학기술원 (KAIST) 전자전기공학과 (전자전기공학학사)
- 2002년 2월 : 한국과학기술원 (KAIST) 전자전기공학과 (전자공학석사)
- 2006년 2월 : 한국과학기술원 (KAIST) 전자전기공학과 (전자공학박사)
- 2006년 3월 ~ 2007년 10월 : 삼성전자 책임연구원
- 2007년 11월 ~ 2013년 2월 : 미국 Ericsson Research Senior Staff Engineer
- 2013년 3월 ~ 현재 : 건국대학교 전자공학부 조교수

<관심분야>
SoC 설계, 무선 통신 알고리듬, SDR

박 성 경(Sungkyung Park) **[종신회원]**

- 1995년 2월 : 서울대학교 공과대학 자원공학과 (자원공학학사, 전자공학 부전공)
- 1997년 2월 : 서울대학교 공과대학 전자공학과 (전자공학석사)
- 2002년 8월 : 서울대학교 공과대학 전기.컴퓨터공학부 (전자공학박사)
- 2002년 8월 ~ 2004년 9월 : 삼성전자 책임연구원
- 2004년 10월 ~ 2006년 10월 : 한국전자통신연구원 (ETRI) 선임연구원
- 2006년 11월 ~ 2009년 7월 : 미국 Ericsson Inc. Senior Staff Hardware Designer
- 2009년 9월 ~ 2013년 8월 : 부산대학교 전자전기공학부 조교수
- 2013년 9월 ~ 현재 : 부산대학교 전자공학과 부교수

<관심분야>
SoC 설계, 무선통신용 알고리듬, 회로 및 시스템