

논문 2016-11-12

핫스팟 접근영역 인식에 기반한 바이너리 코드 역전 기법을 사용한 저전력 IoT MCU 코드 메모리 인터페이스 구조 연구

(Low-Power IoT Microcontroller Code Memory Interface using Binary Code Inversion Technique Based on Hot-Spot Access Region Detection)

박 대 진*
(Daejin Park)

Abstract : Microcontrollers (MCUs) for endpoint smart sensor devices of internet-of-thing (IoT) are being implemented as system-on-chip (SoC) with on-chip instruction flash memory, in which user firmware is embedded. MCUs directly fetch binary code-based instructions through bit-line sense amplifier (S/A) integrated with on-chip flash memory. The S/A compares bit cell current with reference current to identify which data are programmed. The S/A in reading '0' (erased) cell data consumes a large sink current, which is greater than off-current for '1' (programmed) cell data. The main motivation of our approach is to reduce the number of accesses of erased cells by binary code level transformation. This paper proposes a built-in write/read path architecture using binary code inversion method based on hot-spot region detection of instruction code access to reduce sensing current in S/A. From the profiling result of instruction access patterns, hot-spot region of an original compiled binary code is conditionally inverted with the proposed bit-inversion techniques. The de-inversion hardware only consumes small logic current instead of analog sink current in S/A and it is integrated with the conventional S/A to restore original binary instructions. The proposed techniques are applied to the fully-custom designed MCU with ARM Cortex-M0™ using 0.18um Magnachip Flash-embedded CMOS process and the benefits in terms of power consumption reduction are evaluated for Dhrystone™ benchmark. The profiling environment of instruction code executions is implemented by extending commercial ARM KEIL™ MDK (MCU Development Kit) with our custom-designed access analyzer.

Keywords : Bit-line sense amplifier, Instruction access pattern, Low-power instruction access, Flash read-path architecture

1. 서론

사물인터넷(Internet of Thing, 이하 IoT) 개념의 실제 구현은 통신 기능을 가진 초소형 임베디드 시스템을 설계하고 이를 상호 연결하는 것으로부터 출발한다. 설계된 임베디드 시스템은 IoT의 종단에 해당하는 사물에 장착 되거나 원격에 설치되어 대상 사물을 관찰 하며 목적에 맞는 신호를 감지, 수집하며 원하는 제어를 수행한다 [1].

IoT 디바이스의 동작은 독립형 임베디드 시스템

*Corresponding Author (boltanut@knu.ac.kr)

Received: 27 Jan. 2016, Revised: 9 Mar. 2016, Accepted: 18 Mar. 2016.

D. Park: Kyungpook National University

※ 본 논문은 교육부의 재원으로 한국연구재단의 기초과학연구 프로그램의 지원을 받아 수행된 연구결과임 (No. 2014R1A6A3A04059410).

98 핫스팟 접근영역 인식에 기반한 바이너리 코드 역전 기법을 사용한 저전력 IoT MCU 코드 메모리 인터페이스 구조 연구

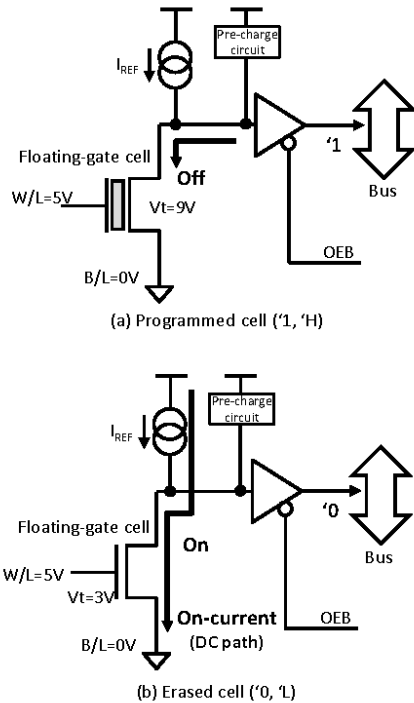


그림 1. 플래쉬 메모리 데이터에 따른 메모리 Bus에서 S/A 전류차이

Fig. 1 Difference in read current according to the programmed data

이 수행하는 단일 기능을 확장하여 네트워크를 기반으로 상호 연결된 거대한(large-scaled) 단말 집합 간에 데이터 공유를 통해 협력적인(cooperative) 작업을 수행한다 [2].

따라서 단일 시스템의 관점이 아니라 사물들의 집합으로 구성된 거시적인 관점에서의 협력적 시스템을 고려하여 IoT용 임베디드 시스템을 설계해야 한다. 이것은 IoT 개별 장치의 기능에 대한 요구사항을 정적으로 결정하여 순수한 회로적인 하드웨어로만으로 시스템을 구현하기 어렵게 만든다. IoT 단말 디바이스의 기능을 동적으로 변경하기 용이하도록 시스템 내장 기능을 활용하도록 사용자 코드를 적절히 프로그래밍하고 이로부터 생성된 기계어 명령어들을 온칩에 내장하는 구조가 적절하다 [3].

따라서 명령어 메모리를 온칩에 내장한 저가격, 저전력 마이크로컨트롤러 (MCU)를 메인제어기로 사용하여 IoT용 임베디드 시스템을 구현하는 것이 일반적이다 [4]. ARM사에서는 고성능 모바일 CPU

인 Application Processor (AP)급 코어를 경량화한 IoT 응용에 최적화된 MCU 전용 Cortex-M계열의 프로세서를 출시하였다.

이러한 MCU는 EEPROM 타입의 플래쉬 메모리를 온칩으로 탑재하여 사용자 소프트웨어로부터 컴파일/어셈블링된 비트들의 집합을 내부의 Floating Gate Cell에 미세하게 프로그래밍할 수 있으며 그림 1에서와 같이 센스 앰프 (Sense Amplifier, 이하 S/A)를 통해 '1'과 '0'으로 구분되어 프로그램된 데이터에 대한 전류 변화를 감지하여 데이터를 읽을 수 있다 [5-6].

전통적인 임베디드 시스템에 적용된 고성능 CPU는 외부 메모리 (External Memory)로부터 바이너리 코드를 간접적으로 온칩 레지스터 버퍼에 먼저 로딩한 뒤 로직 셀에 접근하여 명령어를 패치하기 때문에 작은 로직 전류만 소모한다.

반면 대부분의 저가격 MCU는 게이트수를 많이 요구하는 플립플롭(F/F) 기반 캐시 버퍼를 포함하기 어려우며, 캐시를 사용할 경우 처리과정에서 발생하는 CPU 동작 지연은 리얼타임 인터럽트 서비스 처리를 수행해야 하는 MCU의 기본 특성과 대치된다 [7]. 이로 인해 MCU는 컴파일된 사용자 바이너리 코드를 담고 있는 플래쉬 메모리에 직접 접근하는 방식을 취하며 내장 S/A를 통해 저장된 0또는 1 데이터를 읽는다.

S/A는 미세한 전압차이를 보이도록 프로그래밍된 Floating Gate의 데이터를 구분하기 위해 풀업 (pullup) 전류를 먼저 인가한 후에 워드라인 (W/L)과 비트라인 (B/L)의 바이어스 조건을 인가했을 때 흐르는 sink 전류를 증폭하여 데이터를 읽는다. '0' 데이터를 읽을 때 short current가 발생하며 이는 메모리 접근 전류의 주된 요인으로 작용한다. 전력 소모를 고려하여 절충하여 설계된 pull-up 소스의 크기 및 sink 경로에 설정되는 저항의 크기에 따라 S/A의 접근 속도의 제약이 발생된다 [8].

그러나 IoT 장치가 제공해야할 기능이 더욱 복잡해짐에 MCU의 동작속도가 빨라져야하므로 코드 패치를 위한 코드 플래쉬 메모리로의 빠른 접근이 더욱 요구된다. 이에 상용 MCU 제품들은 전통적인 캐시를 사용하지 않고도 명령어 접근/처리 속도를 향상시키기 위해 메모리 비트라인/워드라인 구조를 변형하여 입출력 인터페이스를 4-way등으로 확장하여 한꺼번에 병렬로 명령어 코드를 메모리로부터 읽는 방법을 통해 명령어 패치 및 실행 가속을 수행한다 [9].

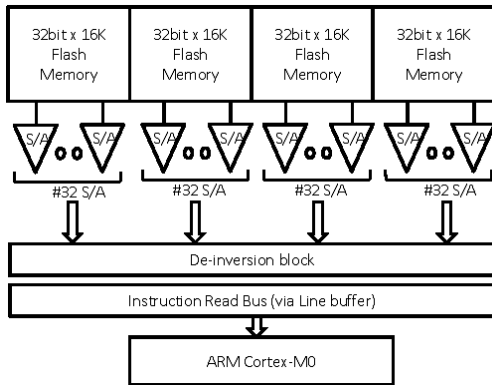


그림 2. 병렬적인 명령어 패치를 위한 멀티웨이 확장 메모리 버스

Fig. 2 Multi-way extended memory bus architecture to fetch instructions in parallel

II. 연구 동기

그림 2는 본 연구를 위해 개발 중인 MCU내부의 명령어 코드 메모리에 접근하기 위해 메모리 버스 구조로 보여준다. 기존에 설계된 플래시 메모리는 매그나칩 0.18um EEPROM CMOS Process에서 구현되었으며 S/A는 비트라인당 20MHz급으로 설계되었다. ARM Cortex-M계열은 공정 Design Kit마다 다르지만 보통 80MHz이상으로 동작하도록 아키텍처가 설계되어 있으나 코드 플래시 메모리의 속도 제약으로 인해 전체 시스템 동작 속도에도 제약이 발생한다.

상용 제품과 마찬가지로 본 연구에서도 캐쉬 사

용 없이 80MHz급으로 ARM Core를 동작시키도록 성능향상을 하기 위해 그림 2와 같이 멀티웨이로 내부 코드 메모리에 접근하도록 메모리 인터페이스를 확장하는 방식을 취하였다. 이러한 방식은 더 많은 S/A의 내장을 요구하며 이는 전체 S/A에서의 동작 전류의 증가로 이어진다.

연구에서 사용한 S/A는 한 개의 bit-line마다 10uA이상의 순간전류가 소모되며 1word (32 S/A) x 4-way = 128 S/A로 구성된다. 플래쉬 데이터를 읽는데 2클럭이 필요하도록 설계하여 짧은 순간만 read 하더라도 많은 수의 S/A에서의 전류의 합으로 인해 큰 전력 소모로 이어질 수 있음을 예상할 수 있다. IoT 시스템이 대부분 온 보드 배터리로 동작하므로 저전력 동작을 위해 S/A에서 발생하는 코드 메모리 접근 전류의 절감이 본 연구의 목적이다.

디지털/아날로그 회로의 단일 하드웨어로 시스템을 구현할 경우 설계된 회로 구조에서부터 이미 동작 전류가 결정되는 반면 소프트웨어 내장형 MCU, 특히 코드 메모리 접근 시 S/A에서의 동작 전류는 내장된 소프트웨어의 패치 과정의 명령어의 비트 패턴에 따라 전류소모가 동적으로 바뀐다 [10].

본 논문에서는 사용자 코드에 따라 동적으로 바뀌는 S/A의 전력소모 감소를 목표로 한다. 이를 위해 내장 코드 메모리에 적재된 바이너리 코드의 변환기법에서 출발하여 [11] 코드 메모리로부터 명령어를 패치하는 과정에서 발생하는 S/A에서의 전류소모를 감소시키기 위한 방법을 제안한다. 본 기법에 관한 초기 연구에서는 컴파일 타임에 코드 분석을 통한 핫스팟 주소 영역을 정적으로 찾고 코드 다운로드 시 결정된 역전기법을 적용하였다 [12].

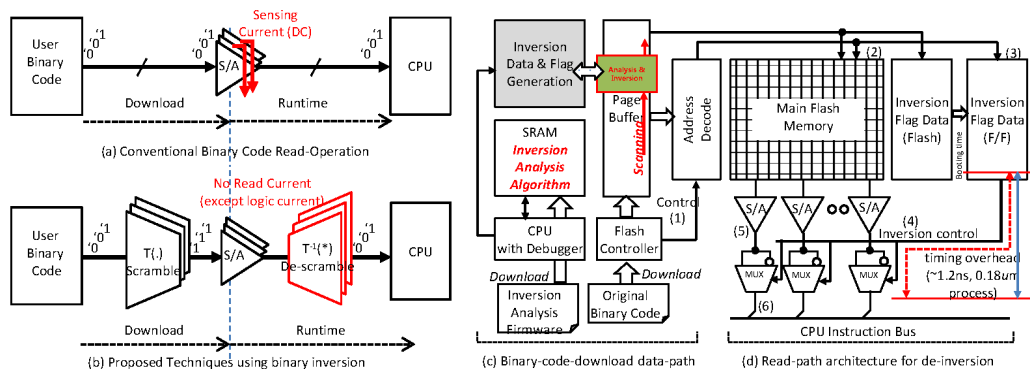


그림 3. 바이너리 비트 역전에 의한 S/A의 전류를 로직 전류로 변환하는 기법

Fig. 3 Binary code inversion/de-inversion to replace S/A access current with logic current

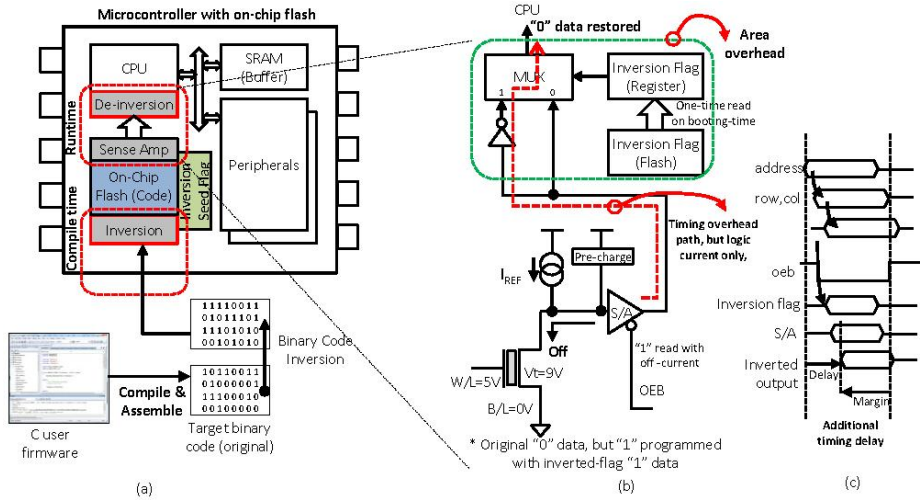


그림 4. 바이너리 비트 역전 (inversion) 기반 MCU 구조와 S/A에 적용된 read 버스 아키텍처
Fig. 4 Proposed microcontroller architecture using bit inversion methods based on the built-in inversion flow and de-inversion logic

본 연구에서는 사용자 코드로부터 컴파일된 바이너리 코드의 동적인 실행을 통해 코드 흐름 패턴을 프로파일링(profiling)하여 핫스팟(hot-spot) 영역을 좀 더 정확하게 파악하고 이에 대한 코드 역전(inversion) 기법을 집중적으로 적용하고 이를 기반으로 동작하는 S/A 구조를 포함한 코드 플래쉬 메모리 인터페이스에 대해 소개한다.

본 논문은 다음과 같이 구성된다. 2장에서 제안하는 기법의 핵심 개념을 소개하고 고안된 S/A의 하드웨어 구조와 바이너리 코드 역전 기법을 상세하게 다룬다. 3장에서 구현결과와 성능평가를 보여주고 4장에서 결론을 맺는다.

III. 본 론

1. Sense Amplifier (S/A) Read Operation

그림 1은 온칩 플래쉬 메모리에 프로그램된 데이터의 종류에 따른 S/A에서의 전류 소모 경로의 차이를 보여준다. 전류 차이를 통해 데이터 읽기 과정에서 데이터 값을 0과 1로 구분하여 메모리 버스에 적재할 수 있게 된다.

기준 전류 원을 통해 read line을 풀업 (pull-up charge) 시킨 상태에서 읽고자 하는 주소에 해당하는 플로팅 게이트의 워드라인 (W/L)과 비트라인 (B/L)을 절히 선택하여 읽기 위한 바이어스 조건을

인가한다.

그림 1(a)와 같이 '1'로 프로그래밍 되었다면 플로팅 게이트가 off되어 전류 흐름이 감지되지 않으며, 그림 1(b)와 같이 '0'으로 프로그램 되었을 경우 게이트가 on되어 풀업 전류 소스로부터 CPU read 경로를 따라 흐르는 전류가 전압으로 변환되어 최종 로직 0/1로 데이터를 구분할 수 있게 된다.

S/A는 데이터를 구분하여 읽기 위해 의도적으로 전류 차이를 발생시키는 구조를 이용하므로 반드시 데이터 값 중에 하나는 반드시 전류 흐름이 발생하게 되며 적재된 바이너리 코드의 비트 데이터 패턴에 따라 메모리 접근 전류가 동적으로 달라질 수 있음을 알 수 있다.

2. Binary Code Inversion Technique

그림 3(a),(b)는 본 연구의 핵심 연구 동기가 되는 개념을 설명하는 것으로, 전류 소모가 더 큰 '0' 데이터를 '1'로 역전(inversion)시켜 프로그래밍하고 이에 대한 정보를 별도로 기록한 뒤 CPU가 실제로 동작할 때는 '1' 데이터를 읽게 되고 이를 다시 로직 inverter로 재역전(de-inversion)을 시키는 기법을 보여준다. 이는 '0' 데이터 접근 시 발생하는 S/A sink 전류를 로직 inverter의 전류로 변환하는 것이다. 하지만 이를 위해서 바이너리 코드를 분석하여 어떤 데이터를 역전시킬지 분석하는 기법이

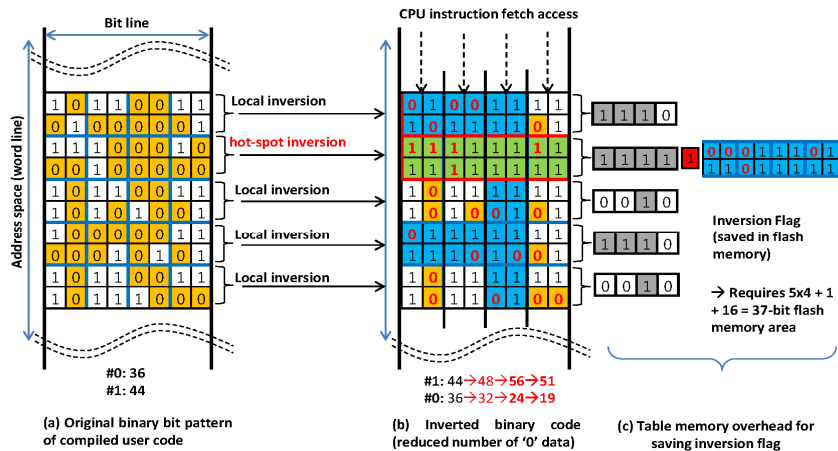


그림 5. 바이너리 비트 역전 (inversion) 기반 MCU 구조와 S/A에 적용된 read 버스 아키텍처
Fig. 5 Proposed micro-controller architecture using bit inversion methods based on the built-in inversion flow and de-inversion logic

필요하며 역전 정보를 담은 별도의 플래그 정보와 영역정보공간 및 재역전을 위한 inverter와 mux등의 부가적인 하드웨어가 read 경로에 추가되어야 한다.

제안하는 방법은 두 단계의 독립적인 절차를 통해 수행되며 그 과정을 그림 3(c)(d)에 도시하였다. 먼저 사용자 소프트웨어로부터 컴파일된 바이너리 코드에 대해 역전 기법을 적용하여 가능하면 '0'보다는 '1' 데이터 많이지도록 비트를 개별적으로 변환하고 이를 MCU 내장 플래시 메모리에 다운로드한 뒤, 추가적으로 역전 (inversion) 위치를 추가로 저장해두어야 한다.

MCU가 부팅되면 온칩 플래시 메모리에 적재된 코드를 개별적으로 접근 및 코드 패치를 통해 내장 명령어를 수행하게 되다. 이때 '1' 데이터로 역전되어 프로그램된 데이터에 대해서는 별도로 저장된 역전 위치 정보를 바탕으로 로직 inverter를 통해 재역전 시키고 최종적으로 mux를 거쳐 CPU read 버스로 로딩된다. 가능하면 '1'로 역전되어 저장함으로써 가능하면 S/A에서는 '0'을 읽을 빈도가 줄어들며 short current 전류의 흐름이 억제된다. 추가된 inverter와 mux에서 아주 미세한 로직 transition 전류만 소모되며 추가적인 gate 전달 지연이 발생하나 그 크기는 $0.18\mu\text{m}$ 공정에서 수 ns 수준으로 무시할 정도로 작다.

3. Proposed MCU Architecture

그림 4는 제안한 데이터 역전 기법을 활용한 MCU구조와 그 안에 포함된 임베디드 코드의 적재 경로 및 온칩 read 버스 아키텍처를 보여준다. 그림 4(a)은 수정된 MCU의 명령어 코드 버스 구조를 표현한다. 기존의 온칩 플래시 메모리와 S/A에 추가적으로 역전 인코딩 하드웨어와 재역전을 위한 디코더 그리고 역전위치를 담고 있는 별도의 변환 테이블을 위한 공간이 필요하다.

그림 4(b)는 바이너리 역전과정과 그 위치 저장 경로, 그리고 S/A에서 데이터 읽는 과정의 순서 흐름을 보여주며 이때 발생하는 제어 신호 및 버스 데이터의 타이밍 다이어그램은 그림 4(c)와 같으며 본 기법에 추가된 로직 inverter와 mux의 게이트 delay에 의한 약간의 time overhead만 존재한다.

4. Hot-Spot Detection and Region Inversion

제안하는 역전 기법은 역전된 비트의 위치를 저장한 별도의 테이블 공간이 필요하게 된다. 따라서 모든 워드라인마다 그리고 모든 비트라인마다 개별적으로 역전 여부를 결정하게 되면 정밀하게 0과 1이 구분되기 때문에 전류 소모를 많이 줄일 수 있으나 그로 인한 큰 테이블 공간이 필요하게 되며, 오히려 이 테이블에 접근하는 전류도 오버헤드로 작용할 수 도 있다. 역전 플래그 테이블의 적절한 크기와 전력 절감의 성능 간에 상호 절충이 필요하다.

102 핫스팟 접근영역 인식에 기반한 바이너리 코드 역전 기법을 사용한 저전력 IoT MCU 코드 메모리 인터페이스 구조 연구

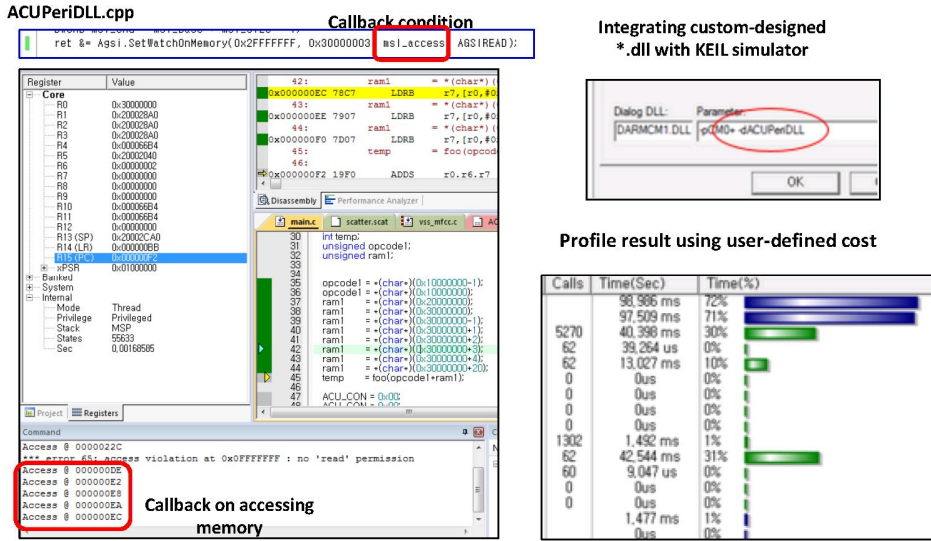


그림 6. 사용자 정의 call-back 함수를 활용한 MDK-ARM MCU Development Kit 확장 및 코드 실행 흐름 프로파일링을 통한 핫스팟 인식

Fig. 6 Profiling Code execution flow and detecting hot-spot region by extending KEIL MDK-ARM MCU development kit with user-defined dll call-back interface

본 연구에서는 워드라인과 비트라인의 두 방향으로 명령어 접근 패턴을 분석하고 일정이상 빈번하게 접근하는 주소 영역에 대해 선별적으로 바이너리 코드 역전 기법을 적용하여 역전 플래그 테이블의 크기를 효율적으로 유지하도록 한다.

그림5는 빈번하게 접근하는 영역인 핫스팟을 찾는 과정을 보여주며, ARM사에서 제공하는 상용 컴파일러/시뮬레이터/디버거인 MDK-ARM툴을 확장하여 코드가 수행될 때마다 이벤트를 발생시키고 이때 call-back 함수를 별도로 구현하여 코드 수행 흐름의 프로파일링을 통해 핫스팟을 분석하였고 이에 대해 3장 실험에서 다시 설명한다.

획득된 핫스팟영역에 대해서만 역전 플래그를 할당하여 라인마다 역전 플래그를 저장하지 않고도 작은 양의 플래그 테이블을 유지하여 접근 전류 감소 효과를 보도록 고안하였다. 그림 5(a)와 같이 특정 메모리 세그먼트에 대해 전체적인 0의 개수는 작지만 세분화해서 0의 위치를 분석하면 노란색으로 표시한 것과 같다. 이를 분석하여 비트 2x2마다 플래그를 남기도록 설계한다면, 그림 5(b)와 같이 2바이트마다 역전 플래그 데이터를 저장해야 한다.

2x2의 해상도로 정밀하게 비트 inversion관리를 수행하지 않고 $N \times M$ 으로 크게 범위를 넓히게 되면 역전 플래그 정보 공간을 크게 줄일 수 있게 된

다. 반면 대부분의 명령어 액세스가 이루어지는 핫스팟에 대해서는 정밀하게 플래그를 남기면 연두색과 같이 바이너리 비트 변환을 할 수 있게 되고 이를 통해 0의 개수는 기존 36개에서 19로 대폭감소하게 된다. 이때 필요한 역전 정보를 담은 플래그는 그림 5(c)와 같은 구조로 테이블에 저장된다.

그림 5는 특정 사용자 바이너리 코드에 대한 하나의 사례를 보여주는 것으로 사용자 바이너리 코드마다 프로파일링을 수행하여 비트 역전 윈도우($N \times M$)크기를 적절하게 조절하고 그 코드 특징을 분석하여 핫스팟 영역을 검출한 뒤 효율적인 바이너리 역전(inversion)을 수행하도록 하였다.

IV. 실험

제안한 기법은 ARM Cortex-M0+ CPU 코어와 온칩 64KB EEPROM기반 플래쉬 메모리를 탑재한 상용 MCU 제품에 적용하여 하드웨어/소프트웨어 모두를 수정하였다.

하드웨어적으로는 내부 MCU read 버스를 변경하도록 Verilog RTL기반 로직 설계를 수행하였고 매그나칩 반도체 0.18um Embedded Flash CMOS Process에서 구현된 S/A회로와 통합하였다. 32비

트로 구성된 명령어 read 버스를 4-way로 결합하여 총 128 비트 라인 S/A를 MCU 내부에 포함시켰다.

소프트웨어로는 최종 바이너리 코드를 플래시 메모리에 다운로드 하기 전에 binary transformation을 수행하는 변환기를 구현하였다. 사용자 C언어 코드로부터 컴파일된 최종 바이너리 코드의 실행 흐름을 분석하기 위해 ARM사의 MDK-ARM에 디버거 환경 [13]을 확장하여 코드 접근영역에 대한 카운터 기능을 구현하였다.

매 코드 라인마다 call-back을 호출하도록 시뮬레이터 인터페이스를 확장하고 [14] 이때 역전 위치를 결정할 데이터 분석을 수행하는 코드를 구현하여 이를 그림 6와 같이 dll파일을 생성하여 MDK-ARM에 동적 연동하였다. 이를 통해 사용자 코드의 핫스팟 지점을 결정하고 이 정보를 바탕으로 비트 역전 (binary bit inversion)을 통한 바이너리 변환 (binary transformation)을 한 뒤 최종적으로 내장 플래시 메모리로 다운로드하도록 변경하였다.

MCU가 부팅하여 명령어를 패치하여 이때 S/A로부터 읽힌 데이터와 역전된 위치 정보를 바탕으로 재역전을 위한 inverter와 mux회로가 128개가 필요하나 이는 S/A크기와 전체 MCU에 들어간 게이트 수 (수만 게이트) 비해 무시할 정도로 작다.

역전 정보를 기록하는 플래그 테이블의 경우 라인마다 구분하여 처리할 경우 전체 명령어 메모리 크기비트 만큼 필요하게 된다. 본 구현에서는 64K bit 만큼 필요하며 이는 무시하기 힘든 메모리 크기 증가를 필요로 한다.

따라서 본 연구에서는 라인마다 역전을 수행하지 않고 핫스팟 지점을 분석하여 4개의 구분된 영역으로 할당하도록 하였으며 결정된 역전을 수행할 메모리 공간의 시작주소와 끝 주소와 이에 대한 역전정보 플래그를 4개의 영역에 대해 별도로 저장/유지하도록 하였다. 이를 위해 $(32\text{bit} \times 2 + 1) \times 4 = 260\text{비트}$ 영역이 플래시 메모리에 별도로 필요하다. 이는 전체 64KB바이트에 비하여 추가로 요구되는 메모리 공간이 크지 않음을 알 수 있다.

핫스팟 영역을 분석하여 결정된 시작과 끝 주소 및 역전 여부 정보를 사용자 바이너리 코드와 임베딩하여 플래시 메모리 다운로드한다. 이후 최초 MCU가 파워가 공급되어 부팅될 때 플래그 정보가 내부 레지스터 구현된 테이블로 로딩 된다. 여기에 필요한 내부 레지스터 크기도 전체 로직 크기에 비하면 무시할 만한 크기이다.

본 기법을 적용하여 회로레벨 전력 절감효과를

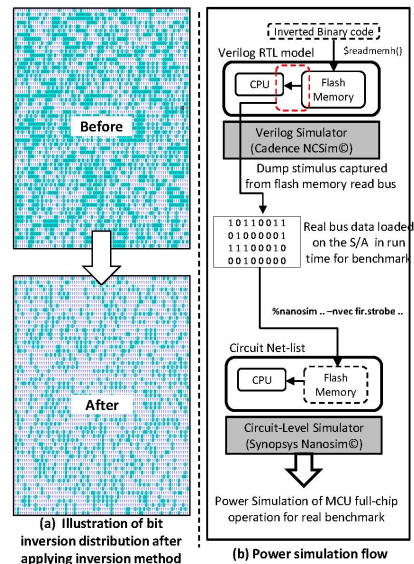


그림 7. 전력 시뮬레이션 환경

Fig. 7 Power simulation flow

분석하기 위해서는 다운로드된 메모리 패턴을 담은 메모리 회로, 이로부터 읽어 들이는 S/A의 아날로그 회로, 그리고 버스 아키텍처를 포함한 MCU 로직에 대한 전체 칩 (full-chip) 시뮬레이션이 복합적으로 이루어져야 한다.

미세 전류 흐름을 측정하기 위해 SPICE 시뮬레이션이 필요하나 S/A를 포함한 수만 게이트로 구성된 전체 MCU에 대해 적용하기에 속도 문제로 인해 불가능한 접근이라 판단하였다. 이에 본 연구에서는 일차적으로 플래시 메모리에 바이너리 역전기법을 적용하여 프로그래밍 하는 과정을 Verilog logic으로 설계하고 이를 시뮬레이션을 통해 명령어 버스에 로딩되는 메모리 버스 트랜잭션 패턴을 파일로 덤프하였다. 이렇게 캡처된 strobe 패턴 파일을 사용하여 회로 레벨 전력 시뮬레이터의 입력 테스트 패턴으로 적용하여 풀칩 (full-chip) 시뮬레이션을 수행하였으며 그 과정을 그림 7에 도시하였다.

Dhrystone [15] 벤치마크를 일부 변형한 사용자 코드에 대한 명령어 접근 패턴을 분석한 결과는 그림 8의 (a)(b)와 같다. 분석된 정보를 바탕으로 역전 처리를 통해 다운로드된 바이너리 코드를 MCU가 적재하고 이를 패치하며 실행할 때의 전체 칩에 대한 전력소모 변화는 그림 8(c)와 같다. 전력소모를 관찰하기 위한 파워 시뮬레이터로 Synopsys사의 nanosim™을 사용하였으며, 변형된 바이너리 코드 패턴이 적용 되었을 때의 메모리 트

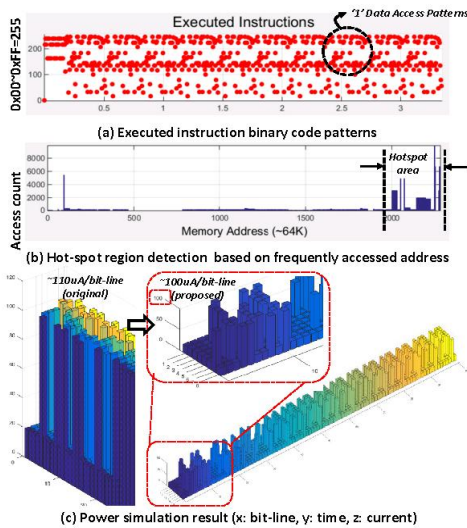


그림 8. DhrystoneTM benchmark에 대한 명령어 접근 패턴 분석 및 핫스팟 영역의 바이너리 비트 역전에 의한 메모리 접근 전류 측정결과
Fig. 8 Flash memory access current and code access pattern analysis based on hot-spot detection for Dhrystone benchmark

랜 액세스를 반영하여 동적인 전력소모량을 측정하였다.

그림 8(c)의 왼쪽 그림은 제안한 역전 기법을 적용하기 전에 100uA의 기본 동작 전류에 추가적으로 '0' 데이터에 접근할 때 10uA의 접근 전류가 필요함을 보여준다. 오른쪽 그림은 역전 기법을 적용하여 사용자 코드 접근에 따른 전력 변화가 대부분 100uA peak 전류 이하로 떨어졌음을 알 수 있다. 제안하는 기법은 적재될 사용자 바이너리 코드의 내용에 민감하나 대체적으로 핫스팟 지점을 4개로 관리했을 때 10%이상의 전력 소모 효과를 볼 수 있었다. '0' 데이터가 작아지는 방향으로 역전 (inversion)을 수행하기 때문에 전력이 오히려 증가하는 경우는 발생하지 않는다.

V. 결 론

본 논문에서는 온칩 코드 메모리에 직접 접근하여 사용자 코드를 실행하는 MCU 구조에서 명령어 접근 및 패치 과정에서 S/A에서 발생하는 전력소모를 줄이기 위한 바이너리 역전 기법을 제안한다. MCU 내부의 온칩 임베디드 플래쉬 메모리에 적재

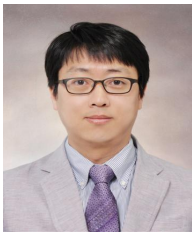
된 명령어 (instruction)로의 접근 패턴에 기반을 둔 핫스팟 (hot-spot) 지점에 집중적이고 효율적인 역전 변환을 수행하여 재역전 (de-inversion)을 위한 디코딩 과정에서 필요한 변환 테이블의 크기를 줄이면서 명령어 접근 전류를 줄이는 방법을 제안하였다. 제안하는 기법은 온칩 S/A 하드웨어의 변형된 구조와 바이너리 코드 역전 소프트웨어의 상호 협력을 통해 약간의 하드웨어 증가비용만으로 효과적인 MCU의 전력 소모 절감이 가능함을 보여주었다.

References

- [1] IEEE Internet Initiative, "Towards a Definition of the Internet of Things (IoT)," Available: <http://iot.ieee.org/definition.html>, May 2015.
- [2] L. Atzori, A. Iera, G. Morabito, "From "smart objects" to "social objects": The next evolutionary step of the internet of things," IEEE Communications Magazine, Vol. 52, No. 1, pp. 97-105, 2014.
- [3] S. Distefano, G. Merlino, A. Puliafito, "Application deployment for IoT: An infrastructure approach," Proceedings of IEEE Global Communications Conference, pp. 2798-2803, 2013.
- [4] K. Itoh, H. Kurata, K. Osada, T. Sekiguchi, "Memory at VLSI circuits symposium," IEEE Journal of Solid-State Circuits, Vol. 43, No. 4, pp. 762-768, 2008.
- [5] P. Cappelletti, "Flash memories," Kluwer Academic Publishers, 1999.
- [6] T. Tanzawa, Y. Takano, T. Taura, S. Atsumi, "Design of a sense circuit for low-voltage flash memories," IEEE Journal of Solid-State Circuits, Vol. 35, No. 10, pp. 1415-1421, 2000.
- [7] R. Micheloni, L. Crippa, M. Sangalli, G. Campardo, "The flash memory read path: building blocks and critical aspects," Proceedings of the IEEE, Vol. 91, No. 4, pp. 537-553, 2003.
- [8] A. Conte, G. Lo Giudice, G. Palumbo, A. Signorello, "A 1.35v sense amplifier for non volatile memories based on current mode

- approach," Proceeding of IEEE the 30th European Solid-State Circuits Conference, pp. 471 - 474, 2004.
- [9] STMicroelectronics, "STM32 ARM Cortex M3 Microcontroller with ART AcceleratorTM," Available: <http://www.st.com/>
- [10] H. Joe, J. Park, C. Lim, D. Woo, H. Kim, "Instruction-Level Power Estimator for Sensor Networks," ETRI Journal, Vol. 30, No. 1, pp. 47-58, 2008,
- [11] D. Park, T.G. Kim, "A sense amplifier using binary code inversion encoder-decoder for on-chip flash read current reduction," Proceedings of International Conference on Electronics, Information and Communication, pp. 235-236, 2011.
- [12] D. Park, T.G. Kim, "Built-In Binary Code Inversion Technique for On-Chip Flash Memory Sense Amplifier With Reduced Read Currenta Consumption," IEEE Transactions on Very Large Scale Integration Systems, Vol. 22, No. 5, pp. 1187-1191, 2014.
- [13] ARM Corp. "MDK-ARM Development Kit," Available: <http://www2.keil.com/mdk5/>
- [14] ARM Corp. "Simulation DLLs for μ Vision using Advanced Generic Simulator Interface (AGSI)," Application Note 196.
- [15] ARM Corp. "Dhrystone Benchmark for ARM Cortex Processors," Application Node 273.

Daejin Park (박 대 진)



He received the B.S. degree in electronics engineering from Kyungpook National University, Daegu, Korea in 2001, the M.S. degree and Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2003, and 2014, respectively. He was a Research Engineer at Major Semiconductor Companies such as SK Hynix Semiconductor, Samsung Electronics, and ABOV Semiconductor over 12 years from 2003 to 2014, respectively and have worked on processor architecture design and lowpower ASIC implementation with custom designed software algorithm optimization. Dr. Park is now with School of Electronics Engineering in Kyungpook National University, Daegu, Korea and a visiting research professor as presidential post-doctoral fellow.

Email: boltanut@knu.ac.kr