NVRAM 기반 간헐적 컴퓨팅 구현을 위한 SW 추상화 계층 설계

이성빈*, 조정훈*
*경북대학교 전자전기공학부
dltjs8649@gmail.com, jcho@knu.ac.kr

Implementation of SW abstraction layer for Intermittent computing development based NVRAM

Sung-Bin Lee*, Jeonghun Cho*
*School of Electronic and Electrical Engineering, Kyungpook National University

요 약

NVRAM(Non-volatile RAM)이란 전원을 공급하지 않아도 데이터를 유지할 수 있는 RAM 이다. 비휘발성 메모리이기 때문에 Flash 와 동일한 기능을 제공할 수 있다. 또한 Flash 에 비해 저전력으로 동작하고, 읽고 쓰는 동작도 더 빠르며 내구성까지 뛰어나다. 즉, NVRAM 은 리소스가 제한적인 사물인터넷(IoT) 장치에서 Flash 를 대신하여 전력소모 및 지연시간 측면에서 효과적으로 사용될 수 있는 메모리이다. IoT 장치는 일반적으로 배터리와 같은 독립전원 장치로 작동하거나, 최근에는 에너지 하베스터를 활용한 간헐적 컴퓨팅 방식도 활용되고 있다. 간헐적 컴퓨팅 방식에서는 전원이 꺼졌을 때도 프로그램의 상태를 유지하기 위해 비휘발성 메모리에 백업동작이 필수적이다. 그러므로 백업을 위한 메모리를 Flash 가 아닌 NVRAM 으로 대체하게 되면 효율적이고, 상대적으로 백업 및 복구에 의한 비휘발성 메모리에 접근이 많은 간헐적 컴퓨팅에서는 더 큰 효율을 볼 수 있다. 하지만 현재 NVRAM 이 내장된 개발보드가 제한적이고, NVRAM 을 외부 모듈로서 사용하기 위해 SPI 또는 I2C 통신을 사용해야 한다. 그 외에도 동시에 공유 메모리에 접근하는 등의 문제를 막아야 한다. 이러한 문제를 막고, NVRAM 을 편리하게 사용할 수 있도록 추상화 계층을 만들어 NVRAM 테스팅 환경을 제공하여 해당 분야의 연구개발을 가속화할 수 있을 것으로 기대된다. 본 논문에서는 NVRAM 의 한 종류인 FRAM 을 사용하여 추상화 계층을 구현하였다.

1. 서론

IoT 기술의 발전에 따라 우리는 모든 것이 연결된 환경에서 살고 있고, 네트워크로 연결된 기기들은 계속해서 증가하고 있다. 일반적인 IoT 장치들은 배터리를 사용하여 동작하고, 이는 배터리 수명이 다할 경우 IoT 장치가 동작하지 못함을 의미한다. 이러한 경우 배터리를 교체해주어야 하지만 교체하기가 까다롭고, 환경에 따라 직접 IoT 장치에 접근하기 힘든 경우가 있다. 이에 따라 최근 에너지 하베스팅 기술이 발전하며 자체적으로 에너지를 생산하여 동작하는 간헐적 컴퓨팅 기술이 개발되고 있다.

간헐적 컴퓨팅 기술은 배터리를 사용하지 않기 때문에 별도의 유지보수를 필요로 하거나 전원 공급을 위해 특정 공간에 위치해야 하는 제약 없이 주변으로부터 수확된 에너지를 사용하여 동작하는 것을 가능하게 한다. 하지만 수급 되는 에너지양은 주변환경에 따라 달라지기 때문에 상황에 따라 전원이 꺼질 수 있다 [1]. 이러한 상황에서 간헐적 컴퓨팅은 전원이 꺼졌다가 켜지는 경우에도 프로그램이 정상적으로 수행되는 것을 보장해야 한다.

기본적인 동작 방식은 전원이 꺼지기 전에 레지스터, 스택, 전역변수 등 프로그램의 현재 상태를 비휘발성 메모리에 저장하고, 켜졌을 때 복구하는 것이다. 즉, 일반적인 컴퓨팅 방식보다 비휘발성 메모리에 접근하는 횟수가 더 많아지고, 이는 리소스의 제약을 크게 받는 IoT 장치에서 오버헤드로 작용한다. 또한, Flash 에 쓰기 동작을 수행하기 위해서는 고전압을 필요로 하기 때문에 전력소모가 중요한 IoT 장치에 적

합하지 않다. 그렇기 때문에 Flash 에 비해 높은 내구성, 빠른 접근속도, 낮은 전력소모를 가지는 NVRAM 이 점차 사용될 것이다. 하지만 현재 NVRAM 의 경우 내장된 개발보드가 제한적이다. 그렇기 때문에 NVRAM 을 활용하여 프로그램을 개발하기 위해서는 외장 NVRAM 모듈을 사용해야 한다. 외장 모듈은 I2C 또는 SPI 통신을 사용해야 하고, 이는 개발자가 프로그램의 동작 측면의 주된 개발이 아닌 별도의 시간을 할애하여 해당 드라이버를 제작하게 한다. 또한 구현된 프로그램에 따라 인터럽트를 사용하는 경우 인터럽트 핸들러에서 NVRAM 에 접근하거나, RTOS 에서 여러 태스크들이 공유 메모리인 NVRAM 에 접근하는 동작을 관리해주어야 한다.

본 논문에서는 IoT 장치에서 최근 수행되는 컴퓨팅 환경인 간헐적 컴퓨팅 기술과 시너지를 낼 수 있는 NVRAM 을 활용한 개발에 편의를 제공할 수 있는 NVRAM 추상화 계층의 플랫폼화를 제안한다.

2. 관련 연구: 간헐적 컴퓨팅

2.1 소개

에너지 하베스팅 기술의 성장에 따라 주변 환경에서 에너지를 수확하여 배터리 없이 동작하는 시스템인 간헐적 컴퓨팅이 연구되고 있다. 하지만 에너지 하베스팅은 공간적, 시간적 제약에 따라 수확되는 에너지가 충분하지 않을 수 있고, 이는 전원이 불안정하여 장치가 꺼질 수 있음을 의미한다. 전원이 꺼졌다 켜지며 간헐적으로 동작하면서도 프로그램이 정상적으로 수행되게 하는 것이 간헐적 컴퓨팅 기술이다.

2.2 메모리 구조

일반적인 컴퓨팅 시스템은 전원이 꺼졌다가 켜지는 경우 프로그램이 새롭게 다시 시작한다. 그 이유는 디바이스 내 RAM 과 같은 휘발성 메모리가 초기화 되었기 때문이다. IoT 장치에서 프로그램의 진행상태 등의 정보는 RAM 에 저장될 수 있고, 전원이 꺼졌다 켜지면 그러한 정보들은 모두 사라진다. 그럼에도 RAM 을 사용하는 이유는 RAM 이 비휘발성 메모리에 비해 접근 속도 및 전력 소모 측면에서 효율적이기 때문이다. 리소스가 제한적인 IoT 장치 특성상 모든 연산들이 비휘발성 메모리에서 수행되면 에너지 소모량이 증가하여 디바이스의 수명이 줄어들기 때문이다. 하지만 프로그램의 정상적인 수행에 중점을 두고 메모리 구조를 비휘발성 메모리로만 구성하는 경우도 존재한다. 하지만 해당 논문에서는 RAM 이 포함된 메모리 구조를 다룰 것이다.

결국, 간헐적 컴퓨팅이 수행되기 위해서는 전원이 꺼지기 전에 RAM 에 저장된 데이터를 비휘발성 메모리에 백업하고, 켜졌을 때 다시 비휘발성 메모리로부터 데이터를 복구하는 과정이 필요하다. 이러한 측면에서 간헐적 컴퓨팅에서는 비휘발성 메모리에 접근이 많아지고, 이는 전력 소모를 증가시켜 IoT 장치에서 오버헤드로 작용한다. 그래서 최근에는 FRAM 과 같은 NVRAM 이 Flash 를 대체하여 사용되는 추세이다.

2.3 실행 모델

대표적인 실행 모델로는 체크포인트 기반 모델이 있다. 체크포인트 기반 모델은 컴파일 타임에 체크포인트를 삽입하는 정적 모델과 실시간으로 저장된 에너지를 측정하여 에너지 상태에 따라 체크포인트 동작이 수행되는 동적 모델이 있다 [2].

컴파일 타임에 체크포인트가 삽입되는 모델은 컴파일러가 어플리케이션을 분석하여 WAR 의존성이 있는 곳과 베이직 블록 사이 등 필요한 곳에 체크포인트가 삽입된다. 이러한 모델은 안전한 동작을 보장하기 위해 체크포인트가 많이 삽입되고, 이로 인해 코드 사이즈와 전력 소모가 증가하게 되어 오버헤드로 작용하는 문제가 있다.

런타임에 체크포인트 동작이 수행되는 모델은 커패시터의 에너지 레벨에서 낮은 임계값, 높은 임계값을 결정하고, 커패시터에 저장된 에너지가 낮은 임계값 이하로 떨어질 경우 인터럽트를 발생시켜 백업 동작을 수행하고, 에너지가 임계값 이상으로 높아지면 인터럽트를 발생시켜 복구 동작을 수행한다. 하지만 런타임 모델의 경우 에너지를 체크하기 위한 하드웨어의 지원을 필요로 한다 [3].

이외에도 정적, 동적 모델의 단점을 보완하기 위한 연구와 정적, 동적 모델의 하이브리드 모델도 연구되고 있다

2.4 문제점

간헐적 컴퓨팅 환경에서 해결해야 할 문제가 있다. 그 중 대표적인 문제가 실행흐름 제어와 신뢰성 있는 데이터라고 할 수 있다. [4].

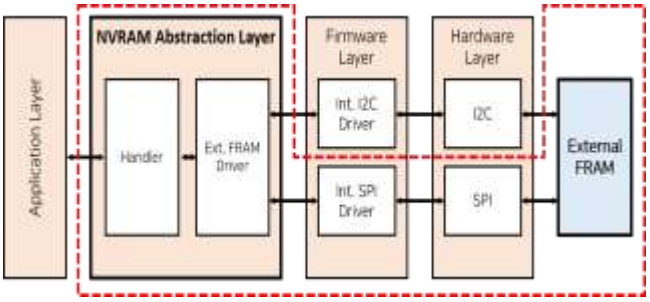
실행흐름 제어 측면에서 발생하는 문제는 프로그램이 더 이상 앞으로 진행되지 못하는 경우가 존재할 수 있다는 것이다. 간헐적 컴퓨팅에서는 전원이 꺼졌다가 켜지면 가장 최근의 체크포인트에서 다시 시작하게 되는데, 디바이스의 동작이 다음 체크포인트로 넘어가지 못하고 전원이 꺼지게 되어 두 체크포인트 사이에서 반복적으로 수행되고 프로그램이 진행되지 못할 수 있다.

데이터의 신뢰성이 깨지는 경우는 다양한 상황에서 발생할 수 있고, 대표적으로 WAR(Write After Read)과 RIO(Repeated Input Operation) 문제가 있다 [4]. 이러한

문제는 코드 간의 종속성과 비휘발성 메모리 접근에 의해 발생하게 된다. WAR 문제의 예로 (1)x:=y (2)y:=5 라는 코드가 연속으로 수행된다고 가정했을 때, 정상적인 경우 (1)에서 x 에 기존의 y 값이 들어가고, (2)에서 y 에 5 라는 값이 들어가게 된다. 하지만 (2)가 끝나고 전원이 꺼져서 (1)앞에 위치한 체크포인트부터 코드가 다시 시작하게 되면 x 값이 기존의 y 값이 아닌 5 가 된다. 즉, x 가 잘못된 값을 가질 수 있는 것이다.

3. 제안 설계

3.1 NVRAM 추상화 계층 설계



(그림 1) NVRAM 추상화 계층

어플리케이션 계층에서는 NVRAM 추상화 계층에서 제공되는 API 를 사용하여 외부에 있는 FRAM 에 접근할 수 있다. 해당 논문에서는 SPI 통신에 대한 디바이스 드라이버만을 구현했지만, 추후에 I2 에 대한 내용이 포함될 수 있다. 이를 통해 어플리케이션 계층에서는 FRAM 이 어떤 통신으로 사용되는 지를 고려하지 않아도 FRAM 을 사용할 수 있다. 또한, 핸들러에서는 메모리에 동시에 접근하는 경우를 막고, 순차적으로 메모리 접근에 대한 동작을 수행할 수 있도록 설계하였다.

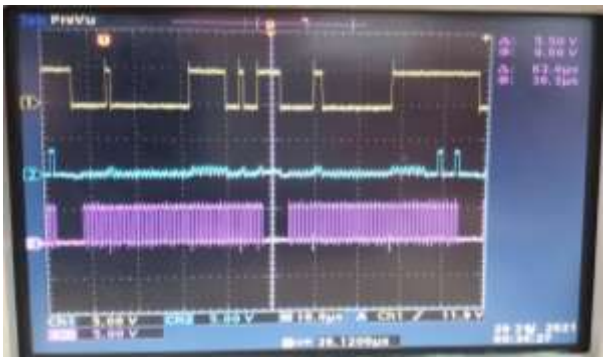
3.2 FRAM 드라이버 구현

외부 FRAM 모듈은 adafruit 사의 MB85RS4MT 를 사용하였고, SPI 통신의 마스터가 되는 개발보드는 S32K144 보드를 사용하였다.

API	Description
void WREN_NVRAM(void)	Set Write Enable Latch
void WRDI_NVRAM(void)	Reset Write Enable Latch
uint8_t RDSR_NVRAM(void)	Read Status Register
void WRSR_NVRAM(void)	Write Status Register
uint8_t READ_NVRAM(uint32_t address)	Read Memory Code
void WRITE_NVRAM(uint32_t address, uint8_t value)	Write Memory Code
uint32_t RDID_NVRAM(void)	Read Device ID

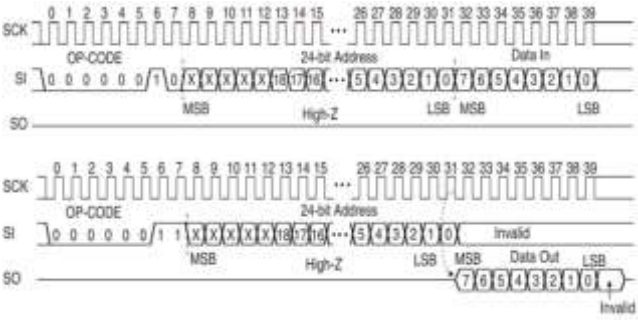
(표 1) FRAM 사용 API

구현한 API 는 표 1 과 같다. WRITE_NVRAM 을 사용하여 값을 쓰고, READ_NVRAM 을 사용하여 값을 읽어 올 수 있다.



(그림 2) 오실로스코프 측정

그림 2 는 WRITE_NVRAM 과 READ_NVRAM API 를 사용한 동작의 파형을 차례대로 SI, SO, SCK 순으로 나타낸 것이다.



(그림 3) FRAM write/read 명령어

그림 3 은 write 와 read 명령의 SPI 통신을 나타낸 것이다. write 와 read 명령은 40 clock 으로 수행되어 진다. 가장 상위 8 비트는 OP-CODE 이고, 24-8 비트는 주소, 하위 8 비트의 경우 write 는 메모리에 저장할 데이터 값, read 는 더미 값이 보내진다.

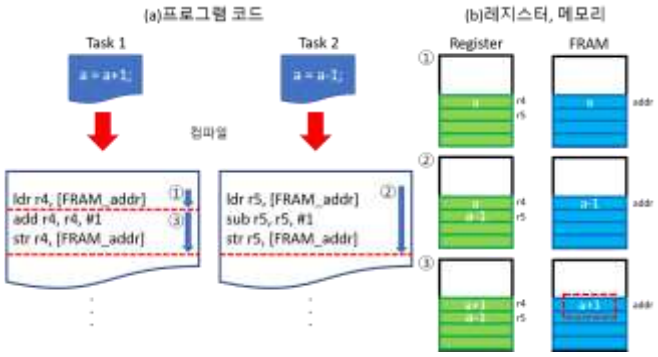
그림 2 의 1 번 채널이 SI 로 write 명령과 read 명령을 나타낸 것이다. write 는 0xFF 주소에 0x11 값을 쓴 것이고, read 는 0xFF 주소에서 값을 읽어온다. 2 번 채널의 SO 에서 read 동작의 마지막 8 클럭에 해당 주소에 쓰여진 값을 출력한 것 볼 수 있다.

오실로스코프로 측정한 파형을 통해 read 에서 읽어온 데이터 값을 정상적으로 받아온 것을 확인했다. 또한, 콘솔창을 통해서도 읽어온 값을 출력하여 정상적으로 write 및 read 동작이 정상적으로 수행된 것을 확인할 수 있었다.

3.3 FRAM 드라이버 핸들러 구현

구현된 FRAM 드라이버를 S32K144 SDK 에서 제공하는 FreeRTOS 세마포어를 사용하여 핸들러를 구현하

였다. 세마포어는 멀티프로그래밍 환경에서 공유자원에 대한 접근을 제한하는 방법으로 사용된다. 공유자원에 대한 접근을 관리해야 하는 이유는 데이터의 신뢰성을 보장하기 위해서이다.

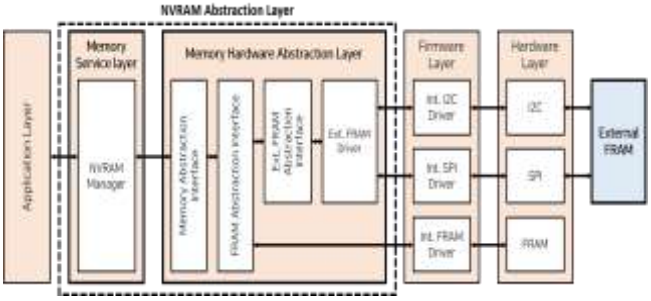


(그림 4) 공유 메모리 접근 문제 예시

그림 4 는 동시에 메모리에 접근한 경우 발생할 수 있는 문제의 예시이다. 그림 4 (b)에서 FRAM 의 addr 에 초기에 저장된 값을 변수 a 라고 가정한다. (a)프로그램 코드의 Task1 과 Task2 는 각각 a 라는 변수에 각각 1 을 더하고 빼는 동작을 한다. C 언어로 표현했을 때는 단순히 1 줄로 이루어져 있지만, 실제 임베디드 장치에서 동작할 때는 컴파일을 거쳐 머신 코드로 바뀐다. 그림 4 (a) 아래의 코드가 머신 코드와 일대일 대응되는 어셈블리어이다. 어셈블리어로 보면 taks1 과 task2 의 동작은 FRAM 에 접근해서 addr 로부터 데이터를 레지스터로 읽어오고, 각각 1 을 더하고, 빼서 이 값을 FRAM 의 addr 에 저장하는 동작을 수행한다. Task1 과 Task2 가 한 번씩 수행되면 정상적인 경우 addr 에 저장된 데이터가 기존의 값인 a 가 되는 것이 정상적이다. 하지만 그림 4 와 같이 동작의 순서가 Task1 이 FRAM 에서 a 값을 레지스터 r4 에 load 해오고 나서, 동작하는 태스크가 Task2 로 바뀌어 Task2 는 FRAM 에 addr 에 저장된 a 값을 레지스터 r5 로 load 해오고, r5 에 저장된 a 에서 1 을 빼서 a-1 을 r5 에 다시 저장한 뒤, r5 에 저장된 a-1 을 FRAM addr 에 store 한다. 이 후, Task1 이 다시 수행된다면, 이미 r4 에 load 해온 값이 α 이기 때문에 1 을 더한 α+1 을 최종적으로 addr 에 store 하게 된다. 즉, a 가 되어야 하는 값이 a+1 이라는 잘못된 값이 addr 에 쓰여진 것이다. 이러한 일을 방지하기 위해 공유 메모리에 동시에 접근하는 것을 막는 것이 일반적이고, 해당 연구에서는 세마포어를 통해 구현한 핸들러가 이를 관리한다.

4. 결론

본 논문에서는 NVRAM 추상화 계층을 설계하고, 외부 FRAM 모듈을 사용하는 단편적인 추상화 계층을 구현하였다. NVRAM 추상화 계층은 간헐적 컴퓨팅 기술의 발전에 따라 NVRAM 의 활용을 촉진하여 IoT 프로그램 개발을 가속화할 것이다.



(그림 5) 최종 NVRAM 추상화 계층

그림 5 는 최종 NVRAM 추상화 계층이다. NVRAM 이 내장된 개발보드에서도 메모리가 부족한 경우 NVRAM 모듈을 추가적으로 사용하는 경우를 고려하여 내부에 존재하는 NVRAM 과 외부에 존재하는 NVRAM 을 사용자가 고려할 필요없이 사용할 수 있도록 메모리 추상화 인터페이스 계층이 추가될 수 있다. 또한 메모리 서비스 계층에서는 핸들러를 포함한 별도의 서비스를 제공할 수 있다. 최종 NVRAM 추상화 계층을 통해 어플리케이션 계층에서 내부/외부 NVRAM 이 하나의 논리적인 메모리로 추상화된 통합 NVRAM 에 대한 접근을 제공할 수 있을 것이다.

감사의 글

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2020R1A2C1013836).

참고문헌

[1] N.A. Bhatti, M.H. Alizai, A.A. Syed, and L. Mottola. Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences. ACM Transactions on Sensor Networks, vol. 12, no. 3, 2016, p.p. 24:1-24:40

[2] B.Lucia, V.Balaji, A.Colin, K.Maeng, E.Ruppel “Intermittent Computing: Challenges and Opportunities,” 2nd Summit on Advances in Programming Languages, Dagstuhl Germany, 2017, pp. 8:1-8:14.

[3] D.Balsamo, A.S.Weddell, G.V.Merrett, B.M.Al-Hashimi, D.Brunelli and L.Benini, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," in IEEE Embedded Systems Letters, vol. 7, no. 1, 2015, pp. 15-18,

[4] M.Surbatovich, B.Lucia and L.Jia “Towards a formal foundation of intermittent computing”, Proceedings of the ACM on Programming Languages, vol. 4, no.163, 2020, pp. 163:1-163:31-