블랙박스 하드웨어 프로토타이핑을 위한 명령어 추출 기법

이상민*, 정대진*, 최재윤**1), 심재균*, 안정호*
*서울대학교 융합과학부
**삼성전자
e-mail:{leesmin6, haidj, ddozzip, acvbf7sim7, gajh}@snu.ac.kr

An Instruction Extraction Method For Dashboard Camera Prototyping

Sangmin Lee*, Daejin Jung*, Jaeyoon Choi**, Jaekyun Shim*, Jung Ho Ahn*
*Department of Transdisciplinary Studies, Seoul National University
**Samsung Electronics

요 약

현대인들의 생활수준 향상과 기술의 발전에 따라 보안, 안전 등 미처 고려되지 않던 분야들이 다양한 영역에서 부각되면서 CCTV, IP카메라, 차량용 블랙박스와 같은 영상기반 시스템에 대한 시장수요가 증가하고 있다. 이에 맞춰 다양한 영상기반 시스템들이 개발되고, 개발 단계에서 낭비되는 시간을 줄이기 위해 프로토타이핑의 필요성이 대두되고 있지만 기존의 프로토타이핑을 위한 도구는 비용이나 속도측면에서 제한적이다. 본 논문에서는 영상기반 시스템 중 블랙박스의 하드웨어를 풀 시스템 에뮬레이터를 이용하여 모델링하고 수행되는 명령어 추출을 통해 시스템의 특성을 예측할 수 있는 하드웨어 프로토타이핑 도구를 제안한다. 또한 ARM 시스템용으로 컴파일 된 프로그램의 실행 여부를 확인하고, 프로그램을 구성하는 명령어와 추출도구를 통해 추출된 명령어를 비교하여 동작을 확인한다.

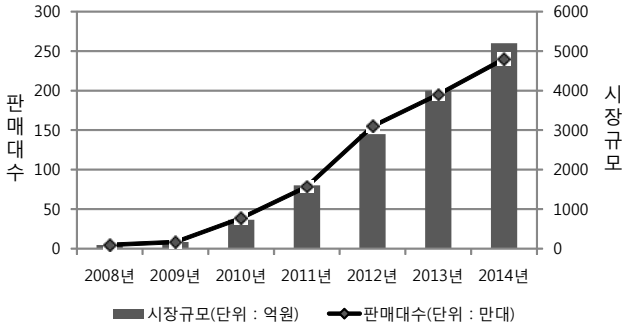
1. 서론

최근 현대인들의 생활수준 향상과 기술의 발전에 따라 보안, 안전 등 미처 고려되지 않던 분야들이 다양한 영역에서 부각되면서 CCTV, IP카메라, 차량용 블랙박스와 같은 영상기반 시스템에 대한 시장수요가 증가하고 있다. 그 중 차량용 블랙박스의 국내 시장규모는 2008년 90억 원에서 2014년 5200억 원으로 50배 가량 증가하였고, 같은 기간 판매대수도 대폭 증가하였다[1]. 또한 해외에서도 차량용 블랙박스 장착 의무화 법안이 통과되는 등 향후 시장규모가 꾸준히 증가할 것으로 전망된다[2].

시장규모가 증가하면서 영상관련 장치를 개발, 제작하던 다양한 기업들의 기존 하드웨어 기술에 기반한 참여 증가로 시장경쟁은 더욱 치열해질 것으로 예상된다. 따라서 차별화된 신제품의 빠른 개발을 위해 펌웨어 및 응용 프로그램 개발자가 SoC와 하드웨어가 개발되기 전에 예상되는 문제점을 파악하고, 실제 환경과 유사한 가상 플랫폼 환경에서 소프트웨어를 미리 검증하여 개발기간을 줄일 수 있는 프로토타이핑 도구가 필요하지만 비용 부담과 실행속도 측면에서의 제약으로 도입이 쉽지 않았다.

본 논문은 ARM ISA(Instruction Set Architecture)를 지원하는 풀 시스템 에뮬레이터인 QEMU를 사용하여 가상 프로토타이핑 방식의 고속 저비용 블랙박스 하드웨어 프로토타이핑 도구를 제안하고, 실행되는 명령어 추출을 통해 시스템의 동작을 분석하고 최적화에 활용한다.

1) 저자가 서울대학교에 재학 중에 수행한 연구임



(그림 1) 국내 차량용 블랙박스 시장규모[1]

2. 관련 연구

2.1 하드웨어 프로토타이핑

하드웨어를 프로토타이핑 하는 방법은 ASIC 풀 칩 시뮬레이션과 FPGA 에뮬레이션 그리고 가상 프로토타이핑으로 나눌 수 있다.

ASIC 풀 칩 시뮬레이션은 전체 하드웨어를 소프트웨어와 함께 클락 단위로 시뮬레이션 하는 방식으로 가장 고전적인 방식이면서 신뢰도가 매우 높다. 하지만 느린 실행속도로 인하여 상대적으로 검증할 수 있는 케이스가 적고, 펌웨어 수준의 소프트웨어만 검증할 수 있다는 단점이 있다. FPGA 에뮬레이션은 매우 빠른 속도를 내지만 FPGA칩의 가격이 비싸고, 보드 및 회로 이외에도 각종 환경을 준비에 많은 노력이 요구되며 특정 IP는 검증이 불가능하다.

가상 프로토타이핑은 HDL수준의 코드를 직접 에뮬레이션 하는 수준으로 구동되며, 소프트웨어 구동 프로토타이핑과 하이브리드 프로토타이핑으로 나뉜다. 소프트웨어 구동 프로토타이핑은 하드웨어를 단순화하여 변환하고 목표 소프트웨어를 CPU모델에서 실행한다. 풀 칩 시뮬레이션보다 실행속도가 빠르고 사이클 단위 정확도와 구동 속도 모드 변환을 제공한다. 대표적인 제품으로는 Carbon Design Systems의 SoC Designer Plus가 있다[3]. 하이브리드 프로토타이핑은 소프트웨어 기반의 프로토타이핑 플랫폼에 하드웨어 가속기인 FPGA를 추가한 것으로 소프트웨어 구동 프로토타이핑보다 더욱 빠른 실행속도를 가지며, Synopsys의 Virtualizer가 대표적이다[4].

2.2 풀 시스템 에뮬레이터를 이용한 성능 분석

풀 시스템 에뮬레이터인 QEMU를 이용하여 응용프로그램이나 시스템의 성능 및 특성을 분석하는 연구들이 여러 차례 진행되었다. QEMU를 안드로이드 플랫폼에 맞춰 수정된 안드로이드 에뮬레이터를 개조하여 안드로이드 시스템 상에서 스마트폰 응용프로그램들의 메인 메모리 접근 특성을 분석하는 도구가 제안되었다[5]. 해당 분석 도구에서는 안드로이드 에뮬레이터에 캐시 시뮬레이터를 결합 후, 명령어를 읽어오는 경우와 Load/Store에 의해 데이터 메모리에 접근하는 경우에 대해 메모리 접근 주소를 추출하는 방식으로 메모리 접근 특성을 분석하였다.

3. 블랙박스 하드웨어 프로토타이핑 도구

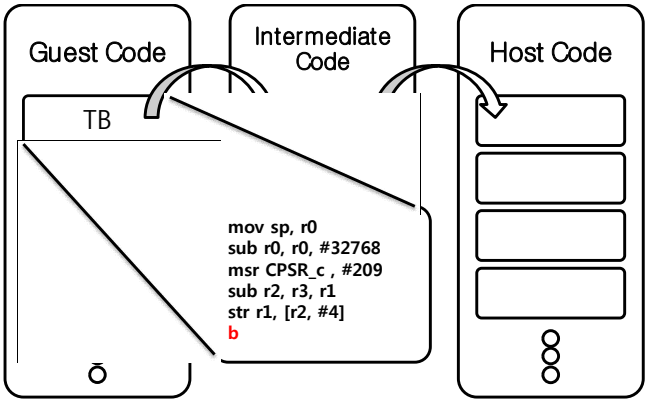
본 장에서는 블랙박스 하드웨어 프로토타이핑 도구의 구성요소 중 하나인 명령어 추출 도구를 추가한 하드웨어 에뮬레이터와 블랙박스의 기본 동작을 수행할 수 있는 블랙박스 응용프로그램을 설명한다.

3.1 하드웨어 에뮬레이션

블랙박스 하드웨어 프로토타이핑 도구는 풀 시스템 에뮬레이터인 QEMU를 이용하여 하드웨어를 에뮬레이션 한다[6]. QEMU는 오픈소스 머신 에뮬레이터로 2005년 USENIX Annual Technical Conference에서 Fabrice Bellard에 의해 처음 소개되었고, 특정한 머신에서 사용하기 위해 작성된 프로그램을 다른 머신에서 실행이 가능하게 해준다. 이는 데스크톱의 플랫폼에서 지원하는 않는 소프트웨어를 테스트하거나 실행시킬 때 유용하다. 또한 QEMU는 x86, ARM, MIPS, SPARC, PowerPC 등 다양한 ISA를 지원하며, 코드변환 방식은 코드변환 필요시에 변환을 수행하는 Dynamic Translation 방식을 사용하며, 변환단위는 Branch 명령어를 기준으로 프로그램을 나눈 TB (Translation Block)를 사용한다[7].

3.2 블랙박스 응용프로그램

블랙박스 응용프로그램은 블랙박스가 갖춰야 할 가장 기본적인 기능인 영상 녹화와 영상 재생이 가능하며 쉽게



(그림 2) QEMU 명령어 변환

조작할 수 있도록 GUI기반 응용프로그램으로 개발되었다.

블랙박스 응용프로그램은 영상을 녹화하고 재생하기 위해 해당 기능들을 지원하는 오픈소스 라이브러리인 FFmpeg과 MPlayer을 이용하였다[8][9]. FFmpeg은 멀티미디어 프레임워크로 Fabrice Bellard에 의해 처음 시작되어 현재는 Michael Niedermayer에 의해 지속되고 있다. 대부분의 멀티미디어 데이터에 대한 Decode/Encode, Mux/Demux, Filter가 가능하며 비교적 오래된 파일 형식까지도 지원한다. MPlayer는 Linux, Windows, MacOS 등 다양한 시스템에서 사용 가능한 동영상 플레이어로 MPEG, AVI, Ogg, ASF, MP4 등 수많은 영상 포맷을 지원하며, X11, OpenGL, libcaca, DirectFB 등 다양한 출력 드라이버를 지원한다.

응용프로그램을 GUI (Graphic User Interface)기반으로 개발하기 위하여 Qt 프레임워크를 이용하였다. Qt는 데스크톱, 임베디드 시스템용 GUI 응용프로그램 개발에 널리 사용되는 크로스플랫폼 프레임워크로 Linux, Windows, Android, iOS 등의 플랫폼을 지원한다. C++를 주로 사용하고 Python, Ruby, Perl, C 등도 연동이 가능하다[10].

3.3 명령어 추출 도구

3.3.1 에뮬레이터 개조

QEMU의 코드변환 과정은 두 단계로 진행된다. 앞 단계에서는 목표 ISA마다 가지고 있는 변환기를 이용하여 게스트코드를 중간코드(Intermediate code)로 변환하고, 뒷 단계에서는 TCG(Tiny Code Generator)를 이용하여 중간코드를 호스트코드로 변환한다. 이 중 앞 단계에서 변환을 위해 입력된 게스트코드를 획득하고 저장하는 방식으로 명령어를 추출하도록 에뮬레이터를 개조하였다.

또한 ARM은 ARM, Thumb, Thumb-2, Coprocessor, NEON 등 총 5가지 명령어군으로 구성되어 있으므로, 이에 따라 명령어 추출 도구는 명령어를 명령어군에 따라 구분하여 호출 횟수를 저장하고, 호출 빈도순으로 상위 10개의 명령어를 결과로 출력한다.[11]

3.3.2 Linux 커널 개조

특정기간이나 특정프로세스에 대한 명령어를 추출하기

위해서 Linux 커널의 수정이 필요하다. 명령어를 추출할 기간의 시작과 끝을 지정하고, 명령어를 추출할 프로세스를 생성하기 위해 3개의 시스템 호출을 추가하였다.

Linux 커널의 do_execve함수를 수정하여 명령어를 추출하고자 하는 프로세스에게는 생성 시 추가적인 정보를 부여하여 일반 프로세스와 구분하였다. 또한 명령어를 추출할 프로세스가 CPU를 선점하였을 경우에만 명령어를 획득하기 위해, 문맥교환 시 명령어를 추출하고자 하는 프로세스가 CPU를 할당받으면 시뮬레이션을 시작하고 CPU를 반납하면 시뮬레이션을 중지하여 특정 프로세스에 대해서 명령어를 추출하도록 변경하였다.

3.3.3 에뮬레이터 커널 간 통신

ARM프로세서는 코어 외에도 메모리 제어기, UART, MMC, USB 등 다양한 주변 장치들로 구성되어 있다. 각 주변 장치들은 특정한 메모리 주소를 할당 받고, Linux 커널은 해당 주변 장치를 제어하기 위해서 그 주변 장치에게 할당된 메모리 영역을 접근한다. 동일한 방식으로 QEMU에 구현된 ARM PrimeCell UART PL011의 Reserved영역에 추출도구 활성화 비트를 할당하고, Linux 커널도 시스템호출을 통하여 디바이스 드라이버가 미리 약속된 추출도구 활성화 비트에 접근하도록 하여 에뮬레이터와 Linux 커널 간 통신을 구현하였다[12].

4. 실험

이 장에서는 블랙박스 하드웨어 프로토타이핑 도구의 동작을 확인하기 위해 ARM 시스템 에뮬레이션과 명령어 추출 가능여부를 확인하고, 영상 녹화, 재생 시 명령어 추출을 진행하였다.

4.1 실험 환경 구성

실험은 ARM 시스템 에뮬레이션과 명령어 추출 가능 여부를 확인하고, 영상 녹화와 재생 시 수행되는 명령어를 추출한다. 실험 결과는 명령어군 별 실행횟수와 실행 빈도순으로 상위 10개의 명령어와 횟수, 그리고 상위 10개의 명령어가 전체 명령어에서 차지하는 비율을 출력한다.

<표 1> 실험 환경

항목		환경
CPU	호스트	Intel i5-3470
	게스트	ARM Cortex-A8
Linux 커널		2.6.35.8
QEMU 버전		1.6.2
입력 장치 (버스)		화상카메라 (USB)
컴파일러 버전		gcc 4.7.3

표 1은 블랙박스 하드웨어 프로토타이핑 도구의 실험 환경을 나타낸다. QEMU는 안정화되고 비교적 최신 버전인 1.6.2를 사용하고, Linux 커널은 임베디드 시스템에서 널리 사용되는 2.6버전 중 사용할 디바이스 드라이버들이 내장되어 있는 2.6.35.8을 이용하였다. 실제 보드와의 호환성을 높이기 위해 보드에서 사용된 CPU인 ARM Cortex-A8을 목표로 선정하였으며, 영상 입력 장치로는 화상카메라를 USB버스로 연결하였다.

4.2 프로토타이핑 도구의 동작 확인

프로토타이핑 도구의 ARM 시스템 에뮬레이션 여부를 확인하기 위해 ARM 시스템용으로 컴파일 된 간단한 문구를 출력하는 프로그램을 실행하였다. 프로그램은 정상 실행되었고 결과는 표 2의 문구출력과 같다.

또한 명령어 추출 도구의 동작을 확인하기 위해 프로그램을 일부 수정하였다. 이전 실험의 결과에서 볼 수 있듯이 간단히 문구를 출력하는 프로그램도 많은 종류의 명령어로 수행되므로 기존 프로그램에 특정 명령어를 삽입하여 기존 결과와 비교해보는 방식으로 실험하였다. 피연산자와 조건실행이 없고 특별한 동작 없이 사이클만 소비하는 nop명령어를 삽입하였다. 표2의 nop 추가(ARM)은 nop을 삽입한 프로그램의 실행결과로 이전 실험 결과에 추가적으로 삽입한 nop명령어가 추출되었다.

또한 Thumb명령어 확인을 위하여 프로그램을 Thumb 모드로 다시 컴파일 하여 동일한 환경에서 실험하였다. 표 2의 nop 추가(Thumb)은 Thumb 모드로 컴파일 한 프로그램의 실행 결과이다. 기존에 ARM명령어로 실행되었던

<표 2> 명령어 추출 확인 결과

항목	문구출력		nop 추가(ARM)		nop 추가(Thumb)	
총 명령어 수	20,665		30,652		30,672	
ARM 명령어	20,665		30,652		20,658	
Thumb 명령어	0		0		10,014	
빈도순 상위 10개 명령어	ldr	6,577	nop	10,000	nop	10,000
	b	3,424	ldr	6,577	ldr	6,576
	cmp	3,017	b	3,431	b	3,422
	str	2,368	cmp	3,020	cmp	3,016
	mov	1,472	str	2,362	str	2,362
	add	1,449	mov	1,469	mov	1,473
	push	627	add	1,443	add	1,448
	adn	535	push	627	push	626
	sub	424	and	534	and	536
	lsr	280	sub	424	sub	423
상위 10개 명령어 수(비율)	20,173(97%)		29,887(97%)		29,882(97%)	

<표 3> 녹화/재생 시 명령어 추출

항목	녹화		재생	
총 명령어 수	2,591,571,013		1,254,671,358	
Coprocessor	156		1	
ARM	2,591,570,857		1,254,671,357	
빈도순 상위 10개 명령어	ldr	780,515,545	ldr	243,865,905
	add	525,629,771	add	210,034,296
	str	279,359,990	str	166,230,461
	rsb	213,515,093	b	148,188,826
	cmp	183,168,799	cmp	126,506,759
	b	128,830,520	mov	99,674,744
	mov	94,318,404	sub	43,416,028
	sub	89,341,832	and	30,861,677
	asr	35,995,556	lsl	20,984,782
	and	34,532,855	orr	20,900,365
상위 명령어 수(비율)	2,365,208,365 (91%)		1,110,663,843 (88%)	

nop명령어가 Thumb명령어로 컴파일 되었고, nop명령어 외에도 14개의 명령어가 Thumb명령어로 컴파일 되었다.

4.3 영상 녹화, 재생 시 명령어 추출 결과

본 절에서는 블랙박스 하드웨어 프로토타이핑 도구의 블랙박스 응용프로그램에서 영상 녹화와 재생 시의 수행되는 명령어를 측정하였다. 표 3은 영상 녹화와 재생 시 수행되는 명령어를 나타내며, 녹화와 재생 시 수행되는 명령어가 유사함을 확인하였다. 블랙박스 응용프로그램은 GUI프로그램이 녹화와 재생을 위해 FFmpeg과 MPlayer을 자식프로세스로 생성하는 구조를 가지므로 GUI프로그램이 수행한 명령어에 영향을 준다. GUI프로그램의 영향을 배제하기 위하여 생성되는 자식프로세스들에 대해서만 명령어를 추출하도록 수정하였고 결과는 표 4와 같다. GUI프로그램의 영향을 배제하여도 ldr, add, str과 같은 기본적인 명령어들이 많이 출력되는 것을 확인할 수 있다.

5. 결론

본 논문에서는 풀 시스템 에뮬레이터를 기반으로 하고 시스템의 명령어 추출이 가능하도록 개조하여 블랙박스와 같은 영상기반 시스템의 개발에 적용할 수 있는 프로토타이핑 도구를 제안하고 실행결과를 정리하였다. 프로토타이핑 도구의 명령어 추출과 ARM 시스템 에뮬레이션을 확인하기 위해 특정 명령어를 삽입한 ARM 시스템용으로 컴파일 된 프로그램을 실행하였고 삽입한 특정 명령어가 제대로 추출되는 것과 ARM 시스템용 프로그램이 실행되는 것을 확인하였다. 블랙박스 응용프로그램의 녹화와 재생 시의 명령어를 추출하여 시스템에서 주로 호출되는 명령어를 파악해보았다. 모든 상황에서 ldr, add, str, mov와 같이 가장 기본적인 명령어들이 가장 많이 실행된다.

본 프로토타이핑 도구는 외부환경을 고려하지 않지만, 일반적으로 시스템들은 외부로부터 적지 않은 영향을 받는다. 이런 외부 영향을 변수로 입력받아 프로토타이핑에 적용하는 추가적인 연구가 필요할 것으로 보인다.

<표 4> 자식프로세스의 녹화/재생 시 명령어 추출

항목	녹화		재생	
총 명령어 수	2,125,446,679		464,953,101	
Coprocessor	147		0	
ARM	2,125,446,532		464,953,101	
빈도순 상위 10개 명령어	ldr	680,519,852	ldr	117,995,988
	add	424,744,428	add	100,317,478
	rsb	237,038,690	str	36,894,473
	str	222,696,262	mov	28,971,753
	cmp	160,459,637	sub	21,941,859
	b	80,470,166	cmp	21,462,307
	sub	73,018,026	b	20,576,090
	mov	38,212,535	lsl	19,710,165
	asr	31,718,237	orr	12,765,749
	mla	30,013,854	and	11,076,596
상위 명령어 수(비율)	1,978,891,687 (93%)		391,712,458 (84%)	

6. 사사

본 연구는 삼성전자의 지원을 받아 수행하였으며, 또한 미래창조과학부/경기도의 재원으로 'SW융합클러스터 사업'의 지원에 의해 수행된 연구결과임 (RA1501).

참고문헌

[1] IRS글로벌 "2014년 자동차용 블랙박스 시장전망과 사업전략 모색을 위한 종합 분석"
[2] 49 CFR Part 571 Federal Motor Vehicle Safety Standards and Regulations
[3] www.carbondesignsystems.com/soc-designer-plus
[4] Synopsys "Virtualizer datasheet"
[5] 이동열 외, "Full-system Emulator를 활용한 모바일 향 시스템의 메모리 접근 분석도구 및 응용", 정보과학회논문지 : 시스템 및 이론 제 40권 제 2호, Apr. 2013.
[6] F.Bellard, "QEMU, a Fast and Portable Dynamic Translator", USENIX Annual Technical Conference, Apr. 2005.
[7] Chad D. Kersey, "QEMU internal", The Linux Users Group at Georgia Tech Meeting, Jan. 2009
[8] http://ffmpeg.org/about.html
[9] https://mplayerhq.hu/design7/info.html
[10] http://wiki.qt.io/About_Qt
[11] Andrew N. Sloss et al., "ARM System Developer's Guide Designing and Optimizing System Software", Morgan Kaufmann, PP.47-100, Mar. 2004
[12] ARM "PrimeCell UART (PL011) Technical Reference Manual", r1p5, Dec. 2007