논문 2012-49CI-1-4

임베디드 시스템에서 실행 가능 압축 기법을 이용한 프로그램 초기 실행 속도 향상

(Program Execution Speed Improvement using Executable Compression Method on Embedded Systems)

전 창 규*, 류 경 식**, 김 용 득**

(Chang Kyu Jeon, Kyeung Seek Lew, and Yong Deak Kim)

요 약

주 기억 장치인 메모리의 전송 속도와 프로세서의 처리 성능 향상에 비해 보조 기억 장치의 속도 향상은 매우 느리다. 응용 프로그램의 실행을 위해서는 보조 저장 장치에서 메모리로의 적재 과정을 거쳐야 하며 이 구간에서 병목현상이 발생한다. 본 논문은 응용 프로그램의 초기 적재 시간의 감소를 위하여 실행 가능한 압축 기법을 구현하고 이의 성능 향상 정도를 실험하였다. 이를 위해서 퍼스널 컴퓨터 상에서 실행하는 실행 바이너리 파일 압축기와 임베디드 환경에서 실행되는 압축 해제기를 각각 구현하였다. 파일의 크기가 다른 6개의 테스트 바이너리 파일을 이용하여 실험한 결과 파일의 크기가 작아 성능이 감소한 경우를 제외하고 평균 약 29%의 프로그램 초기 실행 시간이 감소되었다. 각 파일의 특성에 따라 압축률이 다르고 성능 향상 정도가 다르기 때문에 해당 파일의 특성에 최적화된 압축 알고리즘의 적용이 필요할 것으로 보인다.

Abstract

The performance improvement of the secondary storage is very slow compared to the main memory and processor. The data is loaded from secondary storage to memory for the execution of an application. At this time, there is a bottleneck. In this paper, we propose an Executable Compression Method to speed up the initial loading time of application. and we examined the performance. So we implemented the two applications. The one is a compressor for Execution Binary File. and The other is a decoder of Executable Compressed application file on the Embedded System. Using the test binary files, we performed the speed test in the six files. At the result, one result showed that the performance was decreased. but others had a increased performance. the average increasing rate was almost 29% at the initial loading time. The level of compression had different characteristics of the file. And the performance level was dependent on the file compressed size and uncompress time. so the optimized compression algorithm will be needed to apply the execution binary file.

Keywords : Embedded Systems, Compression, Packing, Loading, Executable File

Ⅰ. 서 론

최근 반도체 제조 기술의 향상으로 프로세서의 동작 속도와 주 메모리의 전송 속도가 빠르게 향상되고 있다. 반면에 보조 저장 장치로 사용되는 하드디스크나 SD카드, USB와 같은 기기들은 전송 속도가 상대적으로 느리게 향상되고 있다. 따라서 프로세서가 연산을 처리하기 위해 데이터를 보조 저장 장치에서 주 메모리로 가져올 때 시간 지연을 발생시킨다. 이는 전체적인 시스템 동작의 성능 저하를 가져오는 문제점을 안고 있다[1]. 이러한 문제점을 해결하기 위해서는 필요한 데이터를 주 메모리에 상주시켜 다시 연산을 수행할 때는 데이터를 주 메모리에서 가져오는 캐싱 방식을 사용할 수 있다. 하지만 이 방식의 경우 주 메모리의 크기를 키

* 학생회원, ** 정회원, 아주대학교 전자공학과
(Department of Electronic Engineering, Ajou University)
접수일자: 2011년12월1일, 수정완료일: 2012년1월2일

워야 하기 때문에 비용적인 부담이 발생하게 된다. 또 다른 해결책은 보조 저장 장치를 병렬로 연결하여 전송 속도의 대역폭을 확장하는 방법이 있다. 이 방법의 경우 저속의 저장 장치를 병렬 확장하기 때문에 전송 속도 면에서 성능 향상을 기대할 수 있지만, 전체적인 시스템 크기의 증가와 비용 증가의 문제점을 갖는다.

임베디드 환경에서 내부에 내장된 특정 프로그램만을 구동시키는 것이 아니라 SD에 저장된 응용 프로그램들 중 선택에 의해 구동된다고 할 때, 외부의 개입으로 인한 응용 프로그램의 불법 복제 및 코드 변조가 발생 할 수 있다[2]. 안드로이드나 아이폰의 응용 프로그램들을 보면 프로그램 실행파일을 역분석 작업을 통해 수정을 가하는 일이 발생하고 있다. 이는 임베디드 시스템뿐만이 아니라 기존 컴퓨터 시스템에서도 발생하던 문제점이다. 컴퓨터 환경에서는 불법 복제를 막기 위한 하나의 방법으로 실행 가능한 압축 기법을 사용하기도 한다.

본 논문에서는 컴퓨터 시스템에서 사용하는 실행 가능한 압축 기법을 임베디드 시스템 환경에 적용 가능하도록 구현하고, 이를 통해 응용 프로그램의 초기 적재 속도 향상 및 무단 복제로부터의 보안을 강화하고자 한다. Ⅱ장에서는 컴퓨터 환경에서 사용하고 있는 실행 가능 압축 기법에 대해서 설명하고, Ⅲ장에서는 임베디드 시스템에 실행 압축 기법의 적용, Ⅳ장은 해당 기법을 적용하였을 때의 성능평가 및 분석, 그리고 Ⅴ장에서 설계한 기법에 대한 결론 순으로 서술한다.

Ⅱ. 실행 가능 압축 기법

1. 실행 가능 압축

실행 가능한 파일 압축은 과거부터 존재한 방법으로 주로 소프트웨어에 의해 사용되는 기억 장치의 용량을 더 효율적으로 활용하기 위해 사용하였다. 디스켓이나 시디와 같이 제한된 저장 매체에 파일을 저장할 때나 인터넷에서의 파일 다운로드 시간을 줄이기 위해 사용할 수 있다. 본 논문에서 제안하는 실행 가능한 압축 파일은 프로그램의 초기 적재 시간을 줄이고, 코드 분석을 막는 기능에 초점을 두고 있다.

일반 압축 파일과 실행 가능 압축 파일을 비교하면 아래 표 1과 같다. 실행 가능 압축을 하게 되면 파일을 실행할 때 마다 압축 해제 루틴이 실행되어야 하기 때문에 시간 손실이 발생한다. 하지만 저장 매체로부터의 전송 속도보다 프로세서의 처리 속도가 매우 빠르기 때문에 실제 프로그램 실행을 위한 처리 시간은 감소하는 결과를 가져온다.

표 1. 일반 압축과 실행 가능 압축 파일 비교
Table 1. Comparison of General Compression and Executable Compression File.

분류	일반 압축	실행 가능 압축
압축 대상	모든 종류 파일	실행 파일
압축 결과물	압축 파일 (zip, rar,...)	실행 파일
압축 해제	전용 압축 해제 프로그램 사용	파일 내부에 해제 루틴 포함
파일 실행	자체 실행 불가	자체 실행 가능
장점	모든 파일에 대해 압축을 할 수 있음	별도 해제 프로그램 없이 실행 가능
단점	전용 압축 해제 프로그램이 반드시 필요함	실행할 때 마다 압축 해제 루틴 필요

실행 가능 압축 파일은 압축 알고리즘이나 보안 방식에 따라 다양한 압축 프로그램이 존재하며, 각 방식에 따라 압축률 및 해제 루틴 처리 시간에 차이가 있다.

2. 데이터 압축 알고리즘

2.1 Lempel-Ziv 부호화

사전식 압축 방법 중 하나로 한번 사용된 문자열을 만나면 기존 문자열로부터의 거리와 일치하는 문자열 길이의 정보를 저장하는 방식이다. 따라서 일치하는 문자열이 많을수록 문자열의 재사용을 줄여 압축률이 높아진다. 인코딩 과정에서는 이미 처리가 끝난 문자열들과 지금의 문자열을 대조하여야 하기 때문에 처리 시간이 긴 단점을 갖는다. 반면에 디코딩 과정은 이미 알고 있는 문자열을 대입하기만 하면 되기 때문에 매우 빠르다. 이 알고리즘을 기반으로 하여 변형된 알고리즘에는 LZ78, LZSS, LZW 등이 있다[3].

2.2 DEFLATE

입력하는 데이터를 여러 개의 블록으로 나누어 사전식 압축 알고리즘인 LZ77을 적용한다. 그 후 중복되는 내용에 대한 포인터는 허프만 코딩을 이용해서 데이터의 크기를 줄인다. 따라서 LZ77만을 사용하였을 때보다 압축률 면에서는 좋으나 인코딩과 디코딩에 더 많은 시간을 필요로 한다. DEFLATE 압축 알고리즘의 경우 PNG 포맷의 이미지 파일에서도 사용하고 있으며, ZIP

파일의 경우도 이 방식을 사용한다[4].

2.3 LZMA

사전식 압축 알고리즘인 LZ77의 변형 압축 알고리즘으로 LZ77와 Markov Chain, Range Encoding 알고리즘의 결합된 형태가 LZMA 방식이다.[3] 이 방식은 LZ77에 의해 압축된 데이터를 Range Encoder를 이용하여 인코딩하게 된다. 이 때 다음 문자열로 올 가능성이 높은 문자열을 예측하여 모델을 적용하는 것이 LZMA 방식이다. 이 방식은 현재 7-Zip 압축 프로그램에서 사용하고 있으며, 압축률 면에서 가장 뛰어난 성능을 갖고 있다[5].

3. 실행 압축 프로그램

컴퓨터 시스템에서 사용하는 실행 압축 프로그램은 원본 실행 파일을 압축을 하고, 압축 해제 루틴을 포함하여 새로운 실행 파일을 만들게 된다. 이때 PE(Portable Executable)의 헤더는 압축 방식에 따라 다르게 구성된다.

3.1 UPX

윈도우나 리눅스에서 사용하는 실행 파일을 LZMA 압축 알고리즘 기반으로 실행 파일을 재구성하는 프로그램이다. 압축 과정에서의 별도 암호 설정 기능은 없으며, 압축된 실행 파일은 누구나 압축 해제를 할 수 있다. 소스 코드 또한 공개되어 있기 때문에 UPX의 변형 실행 압축 프로그램이 만들어지기도 한다[6].

3.2 ASPack

윈도우 환경에서만 동작하는 실행 파일 압축기로 상용 판매되고 있는 프로그램이다. 이 프로그램은 UPX 다음으로 가장 많이 사용되는 압축 프로그램이다. 상용이기 때문에 압축을 해제하려면 별도의 프로그램이 있어야만 하며, UPX에 비해 프로그램의 코드 보안이 강하다는 장점을 갖고 있다. 또 다른 특징은 코드, 데이터, 리소스의 암호화와 압축을 제공한다는 점이고, 다른 압축 도구들에 비해 빠른 복호화 루틴을 갖고 있어서 압축 해제에 의한 시간 소모가 적다.

3.3 변종 압축 프로그램

공개된 실행 압축 소스코드를 응용하여 새로운 실행 압축 프로그램을 만드는 경우도 있고, 또는 여러 개의 압축 프로그램을 혼합하여 이중 또는 삼중으로 압축하는 경우도 있다. 예를 들면 'Yoda's crypt-or/modified+ASProtect -> Alexey Solodovnikov'가 있다. 이 방식은 파일을 이중 압축하고 자신의 이름을 끝에 붙인 경우이다. 이러한 내용은 해당 실행 파일의 PE 정보를 보면 알 수 있다. 이와 같은 파일은 자신 프로그램의 코드 보호를 위한 목적이 강하다고 할 수 있다.

Ⅲ. 동작 구현

1. 실행 파일 압축기

컴퓨터에서 사용되고 있는 실행 가능 압축파일 프로그램은 압축된 실행 파일과 디코딩 루틴이 하나의 파일로 통합되어 있다. 마찬가지로 임베디드 환경에서 동작하는 압축된 실행 파일을 만들기 위해서는 디코딩 루틴 바이너리 파일과 응용 프로그램 파일이 필요하다. 두 개로 나누어진 파일을 하나의 파일로 통합하면 실행 압축 파일이 만들어진다.

본 논문에서는 압축 해제 속도에서 유리한 렘펠-지브 압축 알고리즘을 이용하여 실행 압축 통합 파일 생성기를 설계하였다. 이를 위해서 Lempel-Ziv-Oberumer 압축 라이브러리를 사용하였다. PC 환경에서 임베디드 시스템을 위해 ADS(ARM Developer Suite) 컴파일러로 생성한 바이너리 파일을 불러와서 LZ압축을 수행하고 압축 해제 루틴을 포함하여 새로운 바이너리 파일을 생성한다. 두 파일을 합칠 때 고려해야 할 부분은 디코딩 루틴 파일에서 압축된 실행파일을 불러오는 것 이다. 이는 디코딩 루틴 파일의 크기를 오프셋으

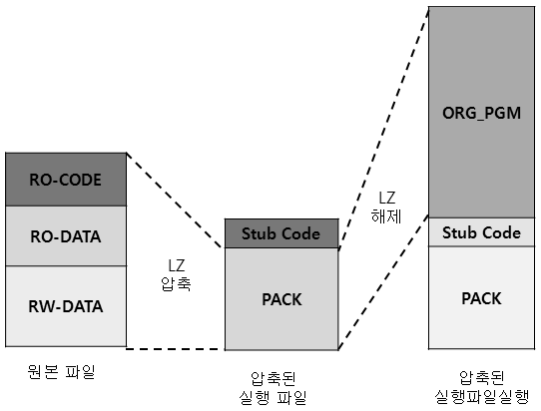

그림 1. 임베디드 시스템에서의 실행 압축 파일 동작
Fig. 1. Operation of Executable Compression File with Embedded System.

로 하여 계산할 수 있다. 이를 위해서는 압축 해제 루틴에 자신의 코드 크기 값을 넣어서 해당 위치를 알 수 있도록 하였다.

PC 환경에서 압축된 실행 파일은 위의 그림 1의 가운데와 같은 구조로 생성된다. 파일의 처음 진입부에 압축 해제 루틴이 포함되며, 그 뒤로 압축된 실행 바이너리 파일이 연속해서 있다. 이 파일을 보조 저장 장치에 복사한 후 임베디드 시스템의 부팅 후 응용 프로그램을 실행하면 압축된 실행 파일의 압축해제 루틴의 처음 부분으로 진입한다.

2. 실행 압축 해제기

실행 압축 해제기는 임베디드 시스템 환경에서 수행되어야 하는 프로그램이기 때문에 ADS를 이용하여 제작하여야 한다. 원본 라이브러리에서 압축 해제 루틴만을 이용하여 프로그램을 구현하였다.

응용 프로그램이 적재되어 수행할 공간의 기본 메모리 주소를 0x3010000번지로 하고, HEAP 영역은 0x30600000번지부터 사용한다고 메모리 맵을 설정하였다. 이와 같이 설정하는 이유는 초기 압축된 실행 파일을 메모리에 적재 후 압축해제가 끝나면 해당 부분은 더 이상 필요 없기 때문이다. 따라서 압축된 실행 파일은 실제 응용프로그램 입장에서는 HEAP 영역에 적재하여 압축 해제 후에는 메모리 낭비를 줄이기 위함이다. 실제 동작 과정을 그림으로 표현한 것은 아래와 같다.

HEAP 영역에 적재가 완료되면 압축 해제를 위한 Stub 코드를 실행한다. 이 프로그램은 압축된 실행 파일을 0x30100000번지에 해제하고 해당 작업이 종료가 되면 응용 프로그램의 초기 진입부로 이동하여 실행시킨다.

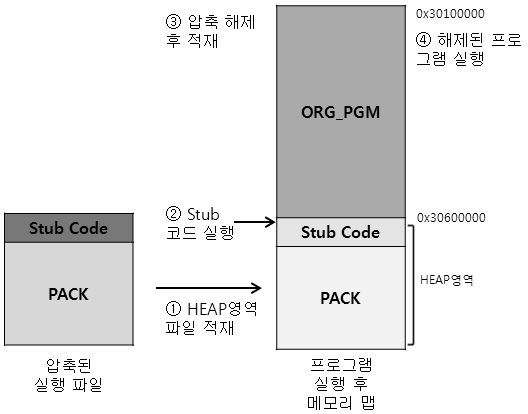

그림 2 실행 압축 해제 프로그램의 동작과 메모리 맵
Fig. 2. Operation of Executable Uncompress Program and Memory Map.

Ⅳ. 성능 평가

1. 실험 환경

ARM v4T 아키텍처 기반의 ARM920T 코어를 사용하는 S3C2440 프로세서를 사용하였다. 이 프로세서를 사용한 것은 네비게이션이나 PMP와 같이 SD카드로부터 데이터를 접근하는 제품에 널리 사용되었기 때문이다. 참고로 보드에 장착된 주 기억 메모리는 32MB이며, 128MB NAND 플래시 메모리를 사용하고 있다.

본 논문에서 실험을 위해 사용한 실행 파일 압축기는 PC 환경에서 비쥬얼 스튜디오를 통해 작업하였고, 임베디드 시스템을 위한 바이너리 파일은 ADS v1.2를 이용하여 컴파일하였으며, 응용 프로그램의 경우 용량이 크고, 많은 수의 프로그램을 사용한다는 전제하에 SD카드에 기록한다. SD카드의 파일시스템은 FAT32를 사용한다.

2. 실험 결과 및 분석

실험을 위해서 프로그램들을 컴파일 후 파일의 크기가 다른 6개의 파일을 이용하였다. 각 파일에 대한 크기와 압축률에 대한 정보는 다음의 표 2와 같다.

실험에 사용한 바이너리 압축 파일은 평균 47.7%의 압축률을 보이며, 마이크로 SD카드에서 메모리로의 각 파일을 전송하는 데 걸리는 시간은 프로세서 내부의 타이머를 이용하여 측정하였다. 하지만 압축 해제 코드의 크기가 약 35KB로 실험 파일 ①의 경우 오히려 성능이 저하되는 문제점이 발생하였다. 이는 압축 해제 코드의 최적화 문제로 앞으로 개선이 필요한 부분이다.

표 2. 실험을 위해 사용한 파일 용량 정보
Table 2. File Size Information for Experiment.

번호	원본 용량(kB)	압축 용량(kB)	압축률(%)
①	101,228	70,962	29.9
②	445,836	260,523	41.6
③	572,072	257,166	55.1
④	958,516	363,734	62.1
⑤	1,513,720	829,855	40.4
⑥	2,158,384	841,460	61.0

표 3. 파일별 전송 및 압축 해제 시간
Tabl 3. Each File Transfer and Decompression Time.

번호	원본 전송 시간(us)	압축 전송 시간(us)	압축 해제 시간(us)	증감 시간 (us)	증감률 (%)
①	21,285	22,280	3,745	+4,740	22.27
②	91,555	60,685	17,990	-12,880	-14.07
③	116,900	59,700	21,205	-35,995	-30.79
④	195,390	81,280	32,730	-81,380	-41.62
⑤	307,420	175,640	65,485	-66,295	-21.56
⑥	435,750	176,945	80,225	-178,580	-40.98

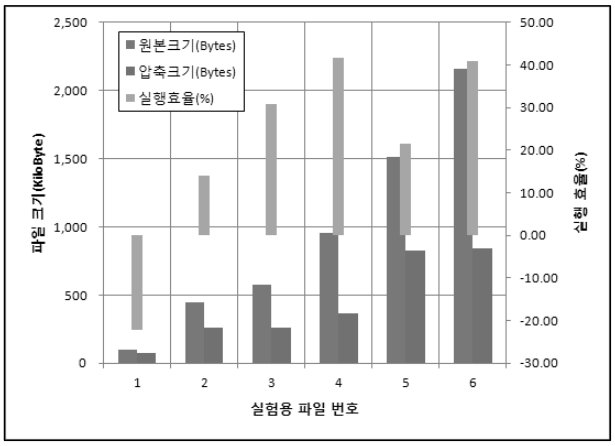

그림 3. LZO 기반 실행 압축 파일 성능 그래프
Fig. 3. Executable Compression File Performance Graph with LZO Algorithm.

각 파일별 원본 파일의 전송 시간 및 압축 해제 코드를 포함한 실행 압축 파일의 전송 시간, 압축 해제 시간, 실제 실행 시간 증감률을 표 3을 통해 정리하였다.

파일 압축률이 좋을수록 전송할 파일의 크기가 많이 줄어들기 때문에 파일 ②의 경우 시간이 14% 감소하는 것을 볼 수 있다. 하지만 파일 ①의 경우 실행 파일의 크기가 작은 편이며, 압축률 또한 높지 않고, 압축 해제 루틴의 크기 문제로 효율이 나빠지는 문제점을 보였다. 이 문제는 실행 압축 파일을 만드는 과정에서 실행 파일을 불러오는 과정에서 실행 성능의 향상을 기대할 수 없다고 판단될 경우 사전에 알릴 필요가 있어 보인다. 이 파일을 제외하고 다른 5가지 파일에 대해서는 평균 약 29%의 성능 향상을 보이고 있다.

위의 그림 3은 파일의 압축률과 실제 성능 향상을 한눈에 볼 수 있도록 그래프화 한 것이다. 파일의 크기가 100KB에 가까운 파일 ①을 제외하고는 모두 실행 성능이 향상된 것을 볼 수 있다. 그리고 상대적으로 압축 효율이 낮은 파일 ⑤의 경우 성능 향상 정도는 약 21%이다.

V. 결 론

임베디드 시스템에 사용되는 프로세서의 성능향상과 더불어 사용되는 응용 프로그램의 용량이 증가하고 있다. 또한 이를 저장하기 위한 보조 기억 매체는 프로세서의 속도 증가에 비해 전송 속도가 느리다. 따라서 본 논문에서는 보조 기억 장치로부터 불러오는 데이터를 압축하여 프로그램 실행을 위한 준비 시간을 줄이고자 하였다. 이를 위해서 구현한 프로그램을 기반으로 실험을 수행한 결과 총 6개의 실험 파일들 중 5개의 실험군은 평균 약 29%의 시간이 절감되는 것을 볼 수 있었다. 특히 컴파일한 응용 프로그램의 압축률이 60%로 가장 높게 나온 실험에서는 프로그램 실행 시간 절감이 약 40%까지 되었다. 실험을 위해 사용한 파일들의 평균 압축률이 53%인 것을 감안하면 압축률이 약 50% 이상일 때 본 논문에서 제시한 방안의 효율이 더 높다고 말할 수 있다.

반면에 초기 프로그램 적재 시간이 증가한 파일 ①의 경우는 압축한 파일의 패턴 특성에 의해 디코딩 복잡도가 증가한 것으로 보인다. 본 실험에서 사용한 압축 기법은 사전식 압축 알고리즘을 사용하고 있고, 압축 하는 파일의 특성에 따라 압축률이 같더라도 사전 인덱스 구성의 차이로 인한 압축 해제 시간이 달라 질 수 있다.

본 논문에서 구현한 방법을 이용하게 될 경우 모든 응용 프로그램은 원본 바이너리 파일이 아닌 압축된 형태의 바이너리 파일을 사용한다. 파일을 압축한 뒤에 파일 형식을 숨길 경우 압축 해제가 어려우며 역공학에 의한 코드 분석이 불가능하다. 따라서 압축된 실행 파일을 사용하게 될 경우 파일 압축에 의해 프로그램 실행 준비 시간을 줄일 수 있는 효과와 더불어 프로그램의 변조를 막을 수 있는 장점을 갖는다. 또한 프로세서 동작 속도를 향상 시킬 경우 압축 해제 시간을 줄일 수 있기 때문에 동작 효율을 더 높일 수 있다.

추후에는 현재 사용한 압축 알고리즘 방식이 아닌 개선된 압축 알고리즘을 이용하여 압축의 효율은 증가시키면서 압축 해제 시간을 단축시키는 방안을 적용할 것이다. 또한 실행 압축 파일의 적용을 하였을 때 성능 향상을 기대할 수 없는 경우 대처 방안을 제시하거나 다

른 압축 알고리즘을 적용하는 형식으로 개선 연구를 진행할 필요가 있다.

참 고 문 헌

[1] 우장복, 최병창, 서효중, “임베디드 시스템을 위한 효율적인 메모리 압축 기법”, *한국정보과학회 제32회 추계학술발표회* 논문집, Vol.32, No.2(Ⅰ), 871-873쪽, 2005.
[2] 장영준 외, “임베디드 운영체제 보안 기술 동향”, *전자통신동향분석*, 제 23권 제1호, 1-11쪽, 2008. 2.
[3] E.-h.Yang and J. C. Kieffer, “On the redundancy of the fixed database Lempel-Ziv algorithm for mixing sources.”, *IEEE Trans. Inform. Theory*, vol.43, pp. 1101-1111, July 1997.
[4] P. Deutsch, “Deflate Compressed Data Format Specification version”, *Internet RFC 1951*, Networking Working Group, May 1996.
[5] 김인홍, 강진구, 이인환, “임베디드 시스템에서의 메모리 압축 스왑 기법”, *한국정보과학회 2010 한국컴퓨터 학술발표논문집*, 제 37권 제 2호(B), pp. 304-308, 2010. 11.
[6] “Ultimate Packer for eXecutables Overview”. http://upx.sourceforge.net/

저 자 소 개

전 창 규(학생회원)
2010년 아주대학교 전자공학과 졸업 (공학사)
2011년 아주대학교 대학원 전자공학과 (석사과정)
<주관심분야 : 임베디드시스템, 네트워크>

류 경 식(정회원)
1991년 아주대학교 전자공학과 졸업 (공학사)
1993년 아주대학교 대학원 전자공학과 졸업 (공학석사)
2011년 아주대학교 대학원 전자공학과 졸업 (공학박사)
2000년～현재 (주)윌텍 대표이사
1993년～1997년 (주)인켈 기술연구소
1997년～1998년 (사)고등기술연구원 정보통신연구실
1998년～2000년 유레카시스템 대표
1998년～2000년 용인송담대학 겸임교수
<주관심분야 : 임베디드시스템, 통신>

김 용 득(정회원)
1971년 연세대학교 전자공학전공 (학사)
1973년 연세대학교 전자공학전공 (석사)
1978년 연세대학교 전자공학전공 (박사)
1979년～현재 아주대학교 전자공학부 정교수
1973년～1974년 불란서 E. S. E 전자공학 연구실
1973년～1974년 미국 STANFORD 대학교 연구교수
1981년～1982년 한국전자통신연구소 위촉연구위원
1994년～1998년 ITS 연구기획단연구위원, 전자부문총괄
<주관심분야 : 통신, 컴퓨터, ITS>