# 디바이스 드라이버 개발 도구 동향

A Trend of Device Driver Development Tool

임베디드 S/W 기술 동향 및 연구 개발 현황


| | |
|---|---|
| 임채덕 (C.D. Lim) | S/W개발도구연구팀 팀장 |
| 김태호 (T.H. Kim) | S/W개발도구연구팀 선임연구원 |
| 김정시 (J.S. Kim) | S/W개발도구연구팀 선임연구원 |
| 마유승 (Y.S. Ma) | S/W개발도구연구팀 선임연구원 |
| 권우일 (W.I. Kwon) | S/W개발도구연구팀 연구원 |
| 최용훈 (Y.H. Choi) | S/W개발도구연구팀 연구원 |


목 차

Ⅰ. 서론
Ⅱ. 요소 기술
Ⅲ. 제품 동향
Ⅳ. QuickDriver 소개
Ⅴ. 관련 도구 간의 비교
Ⅵ. 결론

임베디드 소프트웨어(embedded software)에서 디바이스 드라이버(device driver)는 하드웨어와 운영체제 및 응용 프로그램 사이의 연결고리 역할을 하는 핵심 구성 요소로서, 응용 프로그램이 하드웨어에서 제공하는 기능을 사용할 수 있도록 제어 및 상호 동작을 위한 일관된 인터페이스를 제공하는 소프트웨어이다. 이러한 디바이스 드라이버는 하드웨어와 소프트웨어의 양쪽 측면에 모두 관련이 있어서 개발이 어렵기 때문에 개발을 지원하는 도구가 필요하다. 본 원고에서는 디바이스 드라이버 개발 도구가 갖추어야 할 기능을 크게 소스 코드 자동생성 기능, 테스팅 기능, 정형 검증 기능, 통합 개발 환경 및 개발 편의 유틸리티 기능으로 나누어서 각각의 기술에 대해서 살펴보고, 현재 상용 제품들을 살펴보았다. 그리고, ETRI의 본 연구팀에서 개발한 디바이스 드라이버 통합 개발 도구인 “QuickDriver”를 기술하고, 이 도구와 상용 제품과의 비교를 수행하였다.

# Ⅰ. 서론

임베디드 소프트웨어(embedded software)에서 디바이스 드라이버(device driver)는 하드웨어와 운영체제 및 응용 프로그램 사이의 연결고리 역할을 하는 핵심 구성요소로서, 응용 프로그램이 하드웨어에서 제공하는 기능을 사용할 수 있도록 제어 및 상호 동작을 위한 일관된 인터페이스를 제공하는 소프트웨어이다. 최근 들어 유비쿼터스 컴퓨팅 사회를 구현하기 위해 임베디드 시스템으로 대표되는 많은 디지털 기기들 간의 컨버전스 혹은 기존의 아날로그 기기와의 컨버전스가 빈번해지면서, 임베디드 소프트웨어의 효율적인 개발 방법, 특히 하드웨어와 맞닿아 있는 저수준(low-level)의 소프트웨어인 디바이스 드라이버를 개발하는 체계적이고 효율적인 방법에 대한 중요성이 증대되고 있다.

이와 같이 디바이스 드라이버의 개발은 기존의 일반적인 소프트웨어 개발과는 다르게 하드웨어와 소프트웨어의 양쪽 측면에 관련이 있기 때문에, 다른 과정을 거치며, 결과적으로 개발 도구가 필요로 하는 기능에도 차이가 있다.

하지만, 현재 상용 개발 도구의 기능이 충분하지 못하기 때문에 디바이스 드라이버의 개발을 충분하게 지원하지 못하고 있다.

본 원고는 디바이스 드라이버 개발에 필요로 하는 요소 기술에는 어떤 것이 있는지 구분하여 살펴보았다. 각각의 요소 기술은 소스 코드 자동생성 기능, 테스팅 기능, 정형 검증 기능, 유틸리티 및 통합 개발 환경 기능으로 나누었다. 그리고, 현재 상용 제품을 살펴보고, 또한 ETRI의 본 연구팀에서 개발한 디바이스 드라이버 통합 개발 도구인 "Quick-Driver"를 기술하였다. 마지막으로 이 도구와 상용 제품과의 비교를 수행하고, 결론을 맺었다.

# Ⅱ. 요소 기술

## 1. 소스 코드 자동생성 기능

Linux의 운영체제나 사용자 프로그램 개발은 많은 경우 공개 소스 코드 기반으로 이루어지고 있고, 개발자는 유사한 기능의 프로그램 소스 코드로부터 개발을 시작하는 경향이 있다. 이와 같이 기존의 소스 코드를 수정하고 이를 기반으로 개발을 수행하는 개발 방식은 해당 분야 기술의 발전과 개발 기간 단축 등의 긍정적인 영향도 가져올 수 있으나, 때로는 검증되지 않은 소스 코드의 반복적 유포와 재사용으로 인해, 오작동과 성능 저하 등의 심각한 부작용을 가져오기도 한다.

디바이스 드라이버는 큰 의미로 운영체제의 일부에 속하기 때문에 소프트웨어의 성능과 신뢰성이 전체 시스템의 품질에 매우 중요하기 때문에, 앞서 언급한 것과 같은 개발 방식에 변화가 필요하다. 디바이스 드라이버의 소스 코드 자동생성 기능은 소프트웨어 개발을 백지 상태부터 시작하기를 꺼려하는 Linux 소프트웨어 개발자의 성향에도 만족시킬 수 있을 것으로 판단된다.

소스 코드 자동생성은 다음과 같은 효과가 있다.

- 소프트웨어 개발 시간, 노력, 비용을 감소시킨다.
- 기능이나 성능에서 검증된 소스 코드를 생성해 줌으로써 소프트웨어의 신뢰성을 향상시킨다.
- 기술 지식이 풍부하지 않은 개발자에게 쉬운 개발 시작점을 제공하여 개발의 진입 장벽을 낮추는 효과가 있다.

코드 자동생성 기술은 생성 과정의 복잡도에 따라 여러 가지 형태를 가질 수 있다. 주어진 입력 데이터에 따라 바로 실행이 가능한 형태의 바이너리 출력물이 생성되는 가장 간단한 형태부터, 입력에 대한 중간 생성물이 생성되고 중간 생성물의 변형을 다시 입력으로 하여, 최종적으로 실행이 가능한 바이너리 형태로 생성하는 다단계의 형태가 될 수도 있다.

최신의 소프트웨어 통합개발도구에서 주로 제공되는 코드 자동생성 기술은 새로운 프로젝트를 생성하거나, 새로운 프로그램 모듈 또는, 객체지향 프로그래밍의 경우는 클래스를 추가하고자 할 때, 이름이나 속성 등을 프로젝트 마법사 형식의 사용자 인터페이스를 통해 입력 받아서, 그들의 템플릿 소스

코드를 생성하는 것이다. 이러한 소스 코드 생성기는 제작하려는 소프트웨어의 복잡한 핵심 동작 논리 부분이 아니지만 공통적으로 삽입되어야 하는 소스 코드 부분을 생성하여, 소프트웨어 개발자의 시간과 수고를 덜어준다.

Linux의 디바이스 드라이버 소스 코드는 일부 C++ 언어를 사용하여 작성되는 경우도 있으나, 대부분 저수준의 프로그래밍 언어인 C를 사용하여 작성된다. C언어로 작성된 디바이스 드라이버는 패키지, 클래스, 객체 등과 같은 개념을 나타내는 명시적인 표현이 없어서, 소스 코드의 모듈화 정도가 낮은 편이기 때문에, 객체지향 프로그래밍을 위한 자동생성기법은 적용이 어렵다. 또한, 디바이스 드라이버 소스 코드는 소프트웨어 자체에 대한 정보뿐 아니라, 운영체제와 하드웨어에 관한 많은 정보를 포함하고 있기 때문에, 기존의 소프트웨어 통합개발도구에서의 소스 코드 자동생성 기법과는 다른 접근이 필요하게 된다.

디바이스 드라이버 소스 코드 자동생성의 입력에는 다음과 같은 정보들이 포함되어야 한다.

- 운영체제에 관한 정보: 디바이스 드라이버는 그 자체로 운영체제의 일부분이기 때문에, 하나의 시스템으로 유기적으로 동작하기 위해, 필요한 운영체제 정보가 소스 코드 자동생성의 입력으로 필요하다.
- 하드웨어 정보: 제품/제조업체 인식번호, 메모리 구조, 통신 프로토콜 등 하드웨어 명세서(데이터 시트)의 다양한 정보가 소스 코드 생성에 영향을 미친다.
- 디바이스 드라이버 소프트웨어 자체의 구조 및 기능: 디바이스의 타입에 따라 다양한 소프트웨어 구조가 가능하고, 디바이스의 기능에 따른 처리 논리에 대한 정보가 입력되어야 한다.

위와 같은 입력정보가 반영되어 효율적으로 디바이스 드라이버 소스 코드가 자동생성 되기 위해서는 입력정보를 잘 표현할 수 있는 명세 정의 기술, 정의된 명세를 사용자가 쉽게 편집할 수 있도록 하는 명세 편집기 기술, 작성된 명세를 바로 컴파일하여 실행이 가능한 소스 코드로 번역해주는 명세 번역기 기술이 필요하다.

## 2. 테스팅 기능

임베디드 시스템에서 소프트웨어가 전체 시스템의 품질에 가장 큰 영향을 미치기 때문에, 소프트웨어의 품질에 대한 요구가 높아가고 있다. 소프트웨어 품질을 원하는 수준으로 유지하려면 소프트웨어 제작 과정에서 결함이 소프트웨어에 유입되지 않도록 노력하는 것이 중요하고, 이런 노력 끝에 완성된 소프트웨어를 수행시켜 보고, 결함이 있다면 제거해야 한다. 소프트웨어에 유입된 결함을 찾아내는 데 가장 효율적인 방법 중의 하나는 소프트웨어 테스팅이다. 또한, 테스팅은 구현된 소프트웨어를 제어 가능하고 실제 운용 환경에 가까운 상황 하에서 실행시켜 정상적 작동 여부를 확인해 볼 수 있는 장점을 가지고 있다.

디바이스 드라이버는 그 구현에 복잡하고 어려워, 개발자가 쉽게 오류를 범하기 쉬운 소프트웨어로, 운영체제 시스템 오류의 상당 부분을 차지한다. Microsoft사에서 발표한 보고서에 따르면 Window XP에서 발생하는 오류 중 상당수가 디바이스 드라이버의 오류로 인해 발생했다고 한다. 따라서, 이 오류를 찾아내기 위해서는 테스팅이 중요하다.

소프트웨어 테스팅은 구현된 소프트웨어를 입력 데이터, 즉 테스트 데이터를 가지고 실행시킨 다음에 출력 결과가 예상했던 결과와 일치하는가를 살펴보는 기법이다. 만일 예상했던 출력이 얻어지면 이 테스트 데이터에 대하여 소프트웨어가 올바르게 동작한다는 결론을 얻을 수 있으나, 일치하지 않을 경우는 오류가 있음을 의미하므로 이 때에는 그 원인을 찾아내 코드를 수정하여 오류를 제거해야 한다.

소프트웨어 테스팅은 테스트 데이터를 생성하는 방법에 따라 크게 블랙박스 테스팅과 화이트박스 테스팅으로 나뉘며, 테스트 단계에 따라 단위 테스팅, 통합 테스팅, 시스템 테스팅 등으로 나뉘는데, 이러

한 전통적인 테스트 방식은 디바이스 드라이버 소프트웨어에도 적용될 수 있다. 하지만, 디바이스 드라이버(특히 임베디드 시스템의 디바이스 드라이버)가 테스트 대상인 경우에는 아래와 같은 추가적인 고려사항이 필요하다.

첫째, 임베디드 시스템의 경우 타깃 시스템 자원의 한계 때문에 테스트 프로세스의 전 과정을 타깃 머신에서 수행할 수 없다. 따라서 테스트 데이터의 수행을 제외한 나머지 작업(예를 들어, 테스트 데이터의 생성 및 테스트 결과 분석 등)들은 호스트 머신에서 수행되는 것이 바람직하다.

둘째, 디바이스 드라이버는 일반 소프트웨어와 달리, 커널 모드에서 동작한다. 따라서, 테스트 수행 중 발견된 오류가 시스템 전체에 문제를 야기시킬 수 있으므로(예를 들어, 커널 패닉 현상) 테스팅 시 이에 대한 적절한 처리가 필요하다.

셋째, 디바이스 드라이버는 정해진 운영체제 하에서 정해진 기능을 수행하는 소프트웨어이다. 따라서 동일한 종류의 디바이스 드라이버의 경우 공통된 기능이 존재하므로, 테스트 케이스를 공유할 수 있다.

## 3. 정형 검증 도구

테스팅은 버그를 찾아내는 데 효율적인 방법이지만, 버그가 없음을 보이는 데는 효율적이지 못하다. 따라서, 버그가 없음을 보이는 수학적인 방법인 정형 검증(formal verification)이 대두되었다. 그 중에 대표적인 방법이 모델 검사 기법(model checking)이다. 모델 검사 기법은 검증 과정이 자동화 될 수 있기 때문에 산업계에서 받아들이기 쉽지만, 복잡성이 높아지면 상태 폭발 문제로 검증을 수행하기 어려운 한계가 있기 때문에 상대적으로 단순한 하드웨어 논리를 검증하는 데 많이 사용되어 왔다. 모델 검사 기법에서는 일반적으로 $10^{20}$개의 상태를 처리할 수 있는 한계로 인식하고 있다. 따라서, 소프트웨어의 경우에는 정수형 변수($2^{32}$개의 상태 = $4\times10^{9}$개의 상태)가 3개만 존재해도 $10^{20}$개의 상태를 넘어가기 때문에 소프트웨어의 검증은 어려운 것으로 판단되고 있었다. 하지만, 최근에 프로그램 정적 분석 기법과 프로그램 요약화(abstraction)의 연구 결과로 소프트웨어에도 정형 검증 기법이 적용되기 시작하고 있다. 소프트웨어 정형 검증으로 검증의 대상이 되는 것 중 대표적인 것이 API 사용 규칙 검증이다.

API 사용 규칙이란 특정한 API들을 사용하는 데 있어서 선후 관계 등의 제약 사항을 의미한다. 이러한 API 사용 규칙을 검증 조건으로 기술하고, 그 API를 호출하는 프로그램이 API 규칙을 어기지 않고 프로그래밍 되었는지 검증함으로써 단순해 보이지만 발견하기 어려운 종류의 오류가 없음을 검증할 수 있다.

대표적인 API 규칙으로 커널 락 규칙이 있다. 이것은 lock_kernel()과 unlock_kernel()은 서로 번갈아 가면서 호출되어야 하지, lock_kernel()이 호출된 이후에 lock_kernel()이 호출되거나 unlock_kernel()이 호출된 이후에 unlock_kernel()이 호출된다면 문제가 있다는 것이다. 또한, 프로그램에서 lock_kernel()이 unlock_kernel() 보다 먼저 호출되어야 하며, unlock_kernel()이 호출된 다음에 프로그램이 종료되어야 한다는 것이다.

정형 검증은 오류가 없음을 보이기 위해서 기본적으로 프로그램 소스 코드 내의 모든 경로를 조사하기 때문에, 오류가 존재하는 경우에 오류에 도달하는 경로를 출력할 수도 있다. 이러한 오류에 도달하는 경로는 디버깅 정보로 사용하여 프로그램의 오류를 수정하는 데 도움을 줄 수 있다. 이것은 정형 검증의 부수적인 효과로 생각할 수 있다.

기존에도 API 규칙을 검사하는 도구로 소스 코드 스캐닝(scanning) 도구나 소스 코드 인스펙트(inspect) 도구가 있었으나 정형 검증 도구와의 차이점은 기존 도구들은 분기문과 순환문 등에 나타나는 조건 표현을 고려하지 않기 때문에 오류가 존재하는 경로가 있다고 출력하더라도 실제로는 그 경로가 불가능한 경로인지 아닌지를 분석할 수 없기 때문에 결과적으로 잘못된 오류 정보를 출력하는 경우가 많다.

〈표 1〉 lock_kernel/unlock_kernel 예제

```
1: void kernel_lock_example() {
2:    int x, y, z, new, old;
3:    do {
4:        lock_kernel();
5:        old = new;
6:        if (x == y + z) {
7:          unlock_kernel();
8:          new = new + 1;
9:        }
10:    } while (new != old);
11:    unlock_kernel();
12: }
```

예를 들어 <표 1>과 같은 소스 코드는 기존의 소스 코드 스캐닝 도구로는 분석이 불가능하다. 즉, 기존의 도구는 4: lock_kernel(), 7: unlock_kernel(), 11: unlock_kernel()을 호출하는 경로를 출력함으로써 unlock_kernel()이 연속으로 두 번 호출되는 오류가 존재한다는 출력이 나온다. 하지만 실제로 소스 코드를 보면 7: unlock_kernel()이 호출되는 경우에는 5: old=new가 수행된 상태에서 8: new=new+1이 수행되어, 결과적으로 old!= new이기 때문에 10: while 순환문을 빠져나올 수 없다. 즉, 4, 7, 11의 경로는 불가능한 경로이다. 이와 같이 프로그램 내의 조건문의 표현을 고려하여야만 정확한 검증 결과를 얻을 수 있다.

## 4. 통합 개발 환경 및 개발 편의 유틸리티

통합 개발 환경은 디바이스 드라이버 작성 및 빌딩, 소스 코드 자동생성, 테스팅 및 검증에 이르는 개발 전 과정을 개발자들이 일관성 있는 그래픽 사용자 인터페이스로 접근할 수 있도록 하는 개발 도구 환경 프레임워크이다. 소프트웨어 개발 도구의 사용자 편의성은 기능성만큼이나 개발자들의 도구 활용 여부를 결정하는 중요한 요소로 간주되고 있으므로 통합 개발 환경의 중요성이 매우 중요해지고 있다.

임베디드 시스템을 위한 디바이스 드라이버 개발 지원 통합 개발 환경에서 제공되어야 하는 기본적인 기능은 다음과 같다.

- 코드 편집기 및 컴파일 툴 체인: 디바이스 드라이버 소스 코드를 편집 및 관리할 수 있는 소스 코드 편집기와 타깃 시스템에 알맞은 드라이버로 빌드할 수 있도록 하는 컴파일러를 포함하는 각종 컴파일 유틸리티 제공
- 프로젝트 관리기: 드라이버를 구성하는 소스 파일들의 관리와 빌드 명령 처리를 개발자의 추가적인 작업 없이 자동으로 수행하는 기능으로 기존에 존재하는 소스 파일들의 재활용성을 높이기 위한 소스 파일 임포트 기능 등도 필수적
- 원격 개발 지원: 임베디드 시스템을 위한 소프트웨어 개발환경 지원을 위해 개발된 디바이스 드라이버의 원격 설치 및 실행 등의 기능을 말함

그리고 디바이스 드라이버 개발 환경을 위한 개발 편의 유틸리티는 하드웨어나 커널 관련 전문 지식을 요구하거나 단순 반복적인 처리가 필요한 디바이스 드라이버의 개발 과정에서 활용도 높은 처리 절차를 효과적으로 제공하기 위한 것으로 대표적인 유틸리티 기능은 다음과 같다.

- 디바이스 테스트: 하드웨어가 할당하여 사용하는 메모리, 인터럽트 등의 하드웨어 또는 운영체제 자원을 테스트하는 기능
- 커널 디버거: 디바이스 드라이버와 같이 커널 모드에서 동작하는 시스템 소프트웨어를 디버깅하기 위한 기술로서 소스 레벨 커널 디버거나 커널 시스템 모니터링 기능 등이 있음
- 드라이버 프로그래밍 지원 기능: 디바이스 드라이버 개발 커널 API 활용 문서나 개발하고자 하는 드라이버와 유사한 기능을 가진 샘플 소스 코드의 온라인/오프라인 제공 등과 같은 프로그래밍 속도를 향상시킬 수 있는 다양한 문서 라이브러리 제공 기능

# Ⅲ. 제품 동향

## 1. DriverStudio(Compuware)

Compuware사의 DriverStudio[1]는 Windows용 디바이스 드라이버의 개발, 디버깅, 테스팅, 배포

를 통합적으로 제공하는 디바이스 드라이버 개발 도구이다. DriverStudio에 포함되어 있는 주요 제품 기능들의 특징을 나열하면 다음과 같다.

- DriverWorks: 마법사를 통해 드라이버 개발자가 필요로 하는 하드웨어 정보를 제공하면서 Microsoft사의 DDK로 빌드 가능한 WDM 계열의 Windows용 드라이버 골격 코드를 자동생성
- VtoolsD: 마법사 기반의 VxD 계열의 Windows용 드라이버 골격 코드 생성 및 개발을 지원하는 도구
- DriverNetworks: Windows를 위한 네트워크 드라이버 제작 및 커스트마이징을 지원하는 도구로 통합된 마법사를 통해 제공되는 네트워크 드라이버 표준 코드를 이용하면 신속한 네트워크 드라이버 개발이 가능
- SoftICE: 드라이버를 위한 디버깅 도구로써 소스 레벨 디버거 기반으로 각종 하드웨어 정보의 확인과 수정이 가능한 커널 디버깅 환경을 제공
- TrueTime & TrueCoverage: 드라이버 성능을 저해하는 병목지점을 분석하고 드라이버 테스팅을 통한 드라이버 코드 커버리지 정보를 수집하여 드라이버의 신뢰성을 검증할 수 있는 도구

DriverStudio를 통해 생성된 드라이버 골격소스들은 Windows 기반 개발자에게 친숙한 visual studio에 연동되어 동작하므로 도구 활용을 위한 편의성이 높고 디버깅과 테스팅 및 성능 분석 도구들을 제공함으로써 디바이스 드라이버의 신뢰성 향상에 도움이 되는 도구이다. 특히 SoftICE는 블루스크린을 발생시키는 커널 오류에 대한 디버깅 기능으로 많은 Windows 개발자들에게 활용되고 있다.

하지만 자동생성된 골격 코드들의 주요 내용들이 DriverStudio 고유의 라이브러리에 정의된 API들로 구성되므로 드라이버 완성에 있어 드라이버 개발자들로 하여금 추가적인 개발을 위해서는 추가적인 노력을 요구하게 된다. 또한 데스크톱 시스템을 위한 Windows용 디바이스 드라이버 개발만을 지원하므로 도구가 활용될 수 있는 시스템 플랫폼이 제한적이다.

## 2. WinDriver/KernelDriver(Jungo)

Jungo사의 WinDriver[2]와 KernelDriver[3]는 Windows 및 Linux용 디바이스 드라이버의 신속한 개발을 위해 드라이버 골격 코드의 자동생성과 드라이버 작성에 필요한 하드웨어 및 커널 정보 추출 기능 등을 제공하여 드라이버 작성 단계의 노력을 줄여 줄 수 있는 디바이스 드라이버 개발 도구이다. WinDriver/KernelDriver의 주요 기능은 다음과 같다.

- 하드웨어 테스트: 마법사를 통해 드라이버 개발자가 필요로 하는 하드웨어 정보를 추출하여 제공하며 디바이스 버스 타입에 따라 해당 하드웨어 자원들의 상태 및 데이터 테스팅 기능을 제공
- 드라이버 골격코드 자동생성: 하드웨어 특성을 반영한 하드웨어 접근 코드가 포함된 드라이버 골격코드를 자동생성
- 커널 모니터링: 드라이버 실행에 따른 커널 내부의 동작에서 발생하는 디버그 메시지들을 로깅하여 제공

특히 WinDriver는 USB, PCI, CardBus, Compact PCI, ISA, PMC, PCMCIA 등의 다양한 디바이스 타입에 대해 사용자 모드에서 실행될 수 있는 드라이버 골격코드를 자동으로 생성한다. 또한 자동생성된 골격 코드는 Windows와 Linux 운영체제 환경 모두에 호환성을 갖는 것이 특징이다.

이러한 WinDriver로부터 자동생성된 드라이버 골격코드의 유연성은 골격코드 주요 구성이 이 제품 고유의 라이브러리에 기반한 API들을 활용함으로써 얻어진 것이다. 이에 따라 기존의 드라이버 개발자들은 추가적인 프로그래밍 학습 노력과 이 도구 없이 개발된 드라이버 소스들과의 연동이 어렵다.

## 3. Windows Driver Development Kit(Microsoft)

Microsoft사의 Driver Development Kit(DDK)[4]은 Windows용 디바이스 드라이버 개발을 위한

컴파일 툴 체인 및 라이브러리/유틸리티와 디바이스의 종류별 샘플 드라이버 코드 그리고 관련 문서들의 패키지들로 구성된 통합 툴킷이다. 별도의 GUI 기반 통합 개발 환경을 포함하지 않지만 visual studio와 연동이 가능하며 디바이스 드라이버 개발에 필요한 API가 잘 정의되어 있으며, 특히 많은 사용자층을 보유하고 있는 방대한 개발자용 온라인 라이브러리인 MSDN을 통해 샘플 소스 코드와 기술 지원 및 개발자간 정보 공유를 지원하므로 Windows용 디바이스 드라이버 개발자들에서 필수적인 툴킷으로 알려져 있다.

임베디드 시스템을 위한 디바이스 드라이버 개발의 경우 Windows CE용 DDK를 기반으로 eMbedded Visual C++과 연동하여 기본 디바이스 드라이버 개발 환경을 구축할 수 있다. 그리고 Windows CE test kit을 이용하여 드라이버의 성능이나 기능을 테스팅 할 수 있으며, 최근에는 비록 시험판이지만 정형 검증 기능이 추가되었다. Platform builder를 활용하여 타깃 시스템으로의 배포 환경을 제공한다.

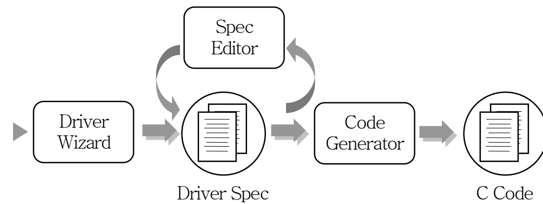
그러므로 DDK 기반 디바이스 드라이버 개발 환경은 단일 통합개발환경의 부재로 개발자들은 다양한 도구들을 숙지하여 필요한 개발 단계에 적절히 활용해야 하는 부담이 단점으로 지적되고 있다.

## Ⅳ. QuickDriver 소개

### 1. 소스 코드 자동생성 도구

QuickDriver의 소스 코드 자동생성 대상은 x86과 XScale 마이크로 아키텍처를 사용하는 임베디드 시스템에서 작동하는 Linux 운영체제의 디바이스 드라이버이다. 버스 타입별로는, USB, PCI, PCMCIA를 지원하고, 문자, 블록, 네트워크를 포함하는 다양한 타입의 디바이스 드라이버 소스 코드 자동생성을 지원한다.

소스 코드 자동생성은 (그림 1)에서와 같이 디바이스 드라이버 프로젝트 마법사를 통한 디바이스 기본정보 입력과 디바이스 드라이버 명세 생성으로 시작하여, 명세 편집기를 통한 추가 명세 편집을 거치고, 명세 번역기를 통해 최종 소스 코드를 얻는 과정을 거치게 된다.

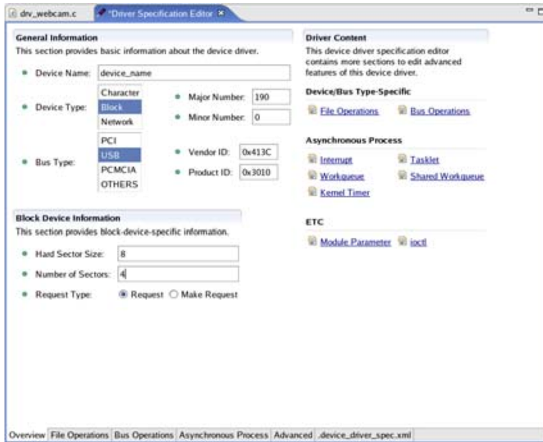



(그림 1) 소스 코드 자동생성 과정

디바이스 드라이버 프로젝트 마법사에서는 리눅스 디바이스 드라이버 분류의 두 가지 주요 기준인 디바이스 타입과 버스 타입을 결정하면, 그에 종속적인 여러 가지 정보를 입력하게 되고, 제한된 수의 다이얼로그 입력을 거친 후, 디바이스 드라이버 명세와 이로부터 생성되는 소스 코드를 얻게 된다. 프로젝트 마법사의 진행과정에서 디바이스 드라이버의 대상이 되는 하드웨어가 개발 타깃 시스템에 연결되어 있을 때는, 마법사가 자동으로 하드웨어의 기본 정보를 읽어와서, 사용자가 하드웨어 명세서(데이터 시트)를 보고 하드웨어의 정보를 입력하는 수고를 덜어준다.

프로젝트 마법사 방식으로 얻은 소스 코드는 디바이스 드라이버의 동작에 대한 부분을 제외한 골격 소스 코드 형태로, 추가적인 프로그래밍이 필요하다. 이 단계에서 개발자는 더 이상 소스 코드 자동생성기의 도움을 받지 않고, 수작업으로 개발을 진행하여 최종 결과물을 얻을 수도 있고, (그림 2)와 같은 디바이스 드라이버 명세 편집기를 사용하여, 조금 더 많고 복잡한 소스 코드 부분을 추가로 자동생성 할 수 있다. 명세 편집기에서 수정 또는 추가되어 자동생성 될 수 있는 요소는 다음과 같다.

- 커널 모듈 관련 정보: QuickDriver를 통해 개발되는 디바이스 드라이버는 Linux의 커널 모듈 형태로 제작이 되는데, 이 모듈의 생성 및 소멸에 관계된 정보와 모듈의 실행에 쓰이는 파라미터 처리 정보가 편집 및 생성될 수 있다.




(그림 2) 디바이스 드라이버 명세 편집기

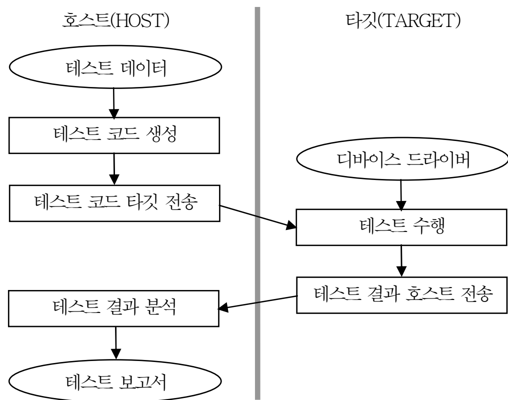
- 파일 조작 함수: Linux는 모든 운영 체제의 자원을 파일로 추상화하여 관리하는데, 디바이스 드라이버에서도 마찬가지로, 해당 디바이스를 파일로 인식시키고, 디바이스에 대한 접근을 파일 조작함수로 처리하게 되는데, 개발자는 명세 편집기를 통해 사용할 파일 조작함수를 선택하고, 각 조작함수별 부가 정보를 편집할 수 있다.
- 버스 관련 구조체 및 함수: 파일 조작함수와 마찬가지로, 디바이스가 USB, PCMCIA, PCI 등 특정 타입의 버스를 사용하게 되면, 이에 따른 데이터 구조와 처리 함수가 생성되게 된다. 또한, 전력 관리나 버스 프로토콜에 관한 추가 정보를 편집하여 소스 코드에 반영시킬 수도 있다.
- 인터럽트 및 지연 가능한 함수 처리: 하드웨어 디바이스의 작동은 많은 부분이 비동기 처리로 이루어져 있고, 인터럽트 핸들링은 가장 많이 사용되는 비동기 처리이므로, 인터럽트 번호 및 타입, 핸들링 이름 등의 간단한 정보의 입력을 통해, 인터럽트 핸들러의 등록, 해체, 처리에 관한 소스 코드를 손쉽게 생성하여, 개발자를 인터럽트 핸들러 내부의 논리에만 집중할 수 있도록 한다. 또한, 태스클릿이나 워크큐, 공유 워크큐 등의 지연 가능한 함수 처리도 명세할 수 있도록 하여, 좀 더 복잡한 형태의 비동기 처리 자동생성을 가능하게 한다.
- 커널 타이머: 또 다른 종류의 비동기 처리인 커널 타이머도 인터럽트와 비슷한 방식의 명세 편집을 통해, 손쉽게 등록/해체/처리 루틴 부분의 소스 코드를 생성할 수 있다.

현재 개발된 QuickDriver의 자동생성기능을 사용하더라도, 핵심 논리 부분의 소스 코드는 많은 부분 여전히 개발자가 직접 프로그램 해야 한다. 더욱 활용도가 높은 개발 도구가 되기 위해서는 좀 더 세밀히 분류된 디바이스 클래스에 초점을 맞추어, 소스 코드의 물리적/논리적 생성률을 증가시켜야 한다. 또한, 생성된 소스 코드를 개발자가 수정한 경우나 이미 다른 개발자나 개발도구를 통해 개발된 디바이스 드라이버 소스 코드를 가져와서 작업할 경우, 이들 소스 코드로부터 디바이스 드라이버 명세 언어로 표현이 가능한 부분을 자동으로 역공학(reverse engineering)하여, 소스 코드 수준의 편집이 아닌, 명세언어 수준의 편집을 가능하게 한다면, 좀 더 활용도 높은 자동생성 도구가 될 수 있을 것이다.

## 2. 테스트 도구

QuickDriver의 테스트 도구는 구현이 완료된 디바이스 드라이버를 타깃 머신에서 수행시켜 보면서, 주어진 테스트 데이터에 대해서 디바이스 드라이버가 올바르게 동작을 하는지 살펴보는 기능 테스트를 호스트 머신에서 수행할 수 있도록 해주는 도구이다. 앞 절에서 언급한 바와 같이 타깃 머신은 자원이 한정되어 있으므로, 반드시 타깃 머신에서 수행해야 하는 작업을 제외하고 대다수의 테스트 관련 작업들은 호스트 머신에서 수행이 된다. 예를 들어, 테스트 데이터를 생성하거나 테스트 결과를 분석하는 작업은 반드시 타깃 머신에서 수행해야 할 필요가 없는 작업이므로, 이 작업들은 호스트 머신에서 수행된다. 호스트와 타깃 간의 관계가 고려된 QuickDriver 테스트 도구의 전체적인 수행 흐름은 (그림 3)과 같다.

타깃 머신에서 실질적인 테스트 수행이 끝난 뒤, 테스트 결과는 내부 통신 모듈에 의해 정해진 형태




(그림 3) QuickDriver 테스트 도구의 수행 흐름도

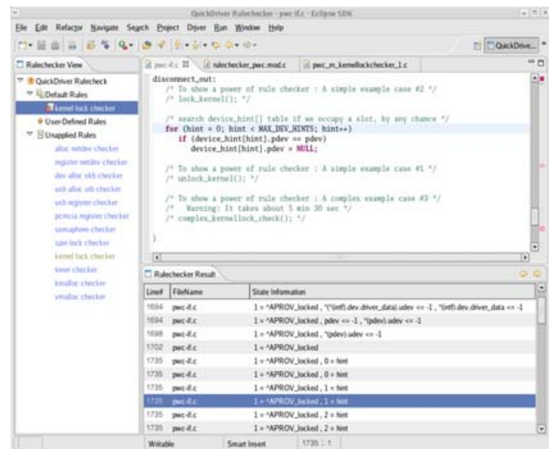
로 호스트 머신으로 전달된다. 호스트 머신에서는 전달받은 테스트 결과를 분석하여 테스트 보고서를 생성해 내는데, 다음의 두 가지 내용이 테스트 보고서에는 포함된다. 첫째, 각각의 테스트 케이스에 대한 테스트 성공 여부가 true/false로 표현된다. 둘째, 전체 테스트 케이스에 의해 수행이 된 디바이스 드라이버 소스 코드 중 어느 정도 수행되었는지를 커버리지 정보를 통해 보여준다. 테스터는 수행 커버리지 정보를 이용하여 테스트가 불충분한 코드부분에 대하여 추가적인 테스트를 수행할 수 있다.

QuickDriver 테스트 도구는 크게 디폴트 테스트 데이터와 사용자 정의 테스트 데이터의 두 가지 종류의 테스트 데이터가 지원된다. 디폴트 테스트 데이터는 디바이스 종류별(저장장치, 네트워크 장치 등)로 공통된 기능에 대해 테스트하기 위해 도구 내부적으로 미리 만들어진 테스트 데이터이다. 따라서 사용자는 이와 같이 디바이스 드라이버에서 반드시 필요한 기능에 대한 테스트 데이터를 새로 만들 필요 없이 도구에서 제공되는 디폴트 테스트 데이터를 이용하여 쉽게 테스트를 수행할 수 있다.

사용자 정의 테스트 데이터는 디폴트 테스트 데이터를 보완하기 위한 테스트 데이터이다. 디폴트 테스트 데이터의 경우, 디바이스 종류별로 반드시 구현이 되어야 하는 기능만을 테스트하므로, 특수 목적의 디바이스 드라이버나 공통 기능 외에 추가적 기능에 대해서는 테스트 할 수 없다. 따라서, 이러한 부분에 대하여 사용자가 추가적으로 테스트 데이터를 작성하여야 하는데, 디바이스 드라이버의 테스트 데이터를 작성하는 것은 디바이스 드라이버를 개발하는 것만큼이나 복잡한 작업이다. 이를 위해 본 도구는 구현이 복잡한 테스트 코드를 사용자가 쉽게 작성할 수 있도록 테스트 코드 템플릿을 제공한다. 뿐만 아니라, 테스트 데이터 작성에 유용한 라이브러리를 제공하여, 사용자가 쉽게 테스트 데이터를 작성할 수 있도록 도와준다.

## 3. 정형 검증 도구

QuickDriver의 정형 검증 도구, 구체적으로는 API 규칙 검증 도구는 다음과 같은 구조를 가지고 있다. 사용자 인터페이스는 Eclipse(QuickDriver) 플러그인으로 개발되었고, 인터페이스는 규칙을 선택하고 검증을 수행할 소스 코드를 선택하고, 결과를 출력한다. 검증의 결과로 API 규칙을 모두 따랐을 경우에는 pass로 출력이 되고, 만약에 API 규칙을 따르지 않았을 경우에는 fail로 출력된다. 그리고, fail이 되는 과정은 상태의 변화 순차와 각 상태에서의 변수 값이 출력되는데, 이 때 각 상태를 선택하면 해당되는 소스 코드가 역상으로 출력되어 디버깅에 도움이 되며, (그림 4)는 커널 락을 검증했을 때 fail이 된 경우로서, 오른쪽 아래에 fail이 되는 과정이



(그림 4) 규칙 검증기 결과 화면

출력되어 있다.

실제 검증을 수행하는 과정은 2단계로 이루어져 있는데, 실제 소스 코드에 검증 조건을 삽입하여 소스 코드를 변환하는 단계와 변경된 소스 코드에 대해서 실제로 검증을 수행하는 단계로 구성되어 있다. API 규칙은 APROV-SL(a Specification Language for Another PROgram Verifier)라는 언어를 정의하였다. APROV-SL은 precondition/postcondition 표현을 이용하여 기술되며, <표 2>는 APROV-SL로 기술된 커널 락 규칙이다. APROV-SL에서 TYPE은 타입 선언, GLOBAL은 변수 선언에 해당된다. 또한 INITIAL은 변수의 초기값이고, 각각의 EVENT는 특정 패턴일 때 어떤 행동을 하는지를 기술한다. 예를 들어, 첫번째 EVENT의 경우에 PATTERN에 나타나 있는 lock_kernel()이 호출되면 그때 PRECHECK의 내용인 APROV_locked == init||APROV_locked == unlocked를 검사한다. 만약에 이 조건을 만족시키지 않으면, 오류가 있다는 의미이다. 이후에 PRECHECK의 내용을 만족시키면 ACTION 문의 APROV_locked = locked의 내용을 수행한다. 이와 같은 방법으로 특정 API에 대한 행동을 기술할 수 있다. 마지막으로 FINAL에는 프로그램이 종료되었을 때 만족해야 하는 조건을 기술한다.

〈표 2〉 APROV-SL로 기술된 커널 락 규칙

```
kernellockchecker {
  TYPE {enum m_locked {init, locked, unlocked};}
  GLOBAL {enum m_locked APROV_locked;}

  INITIAL {APROV_locked = init;}
  EVENT {
    PATTERN {lock_kernel();}
    PRECHECK {APROV_locked == init||APROV_locked
  == unlocked}
    ACTION {APROV_locked = locked;}
  }
  EVENT {
     PATTERN {unlock_kernel();}
     PRECHECK {APROV_locked == locked}
      ACTION {APROV_locked = unlocked;}
  }
  FINAL {APROV_locked == unlocked}
```

## 4. 통합 개발 환경 및 개발 편의 유틸리티

QuickDriver의 통합 개발 환경은 이클립스 개발 환경 플랫폼을 기반으로 디바이스 드라이버 개발을 위한 프로젝트 관리기, 원격 개발 지원 등과 디바이스 드라이버 개발 단계별 도구들이 통합되어 있는 원격 디바이스 드라이버 통합 개발 환경이다. QuickDriver의 개발 편의 유틸리티는 디바이스 드라이버 개발 과정 중에 개발자의 디바이스 드라이버 개발 편의를 도모하기 위한 것으로 디바이스 리소스 진단도구, 커널 셸, 커널 메시지 로깅 도구, 커널 API 도움말 등으로 구성된다.

### 가. 통합 개발 환경

통합 개발 환경은 디바이스 드라이버를 개발하는데 편리한 환경을 지원하는 것을 주 목적으로 한다. 디바이스 드라이버 개발의 모든 단계가 본 통합개발 환경에서 진행되며, 디바이스 드라이버 개발 단계에 필요한 여러 가지 개별 도구들을 단일화된 사용자 인터페이스 시작점상에서 실행할 수 있다. 일반적인 소프트웨어 개발 도구 IDE에서 지원하는 기본 기능을 대부분 포함하고 있으면서 디바이스 드라이버 개발에 필요한 기능을 이클립스 플러그 인으로 개발하였다.

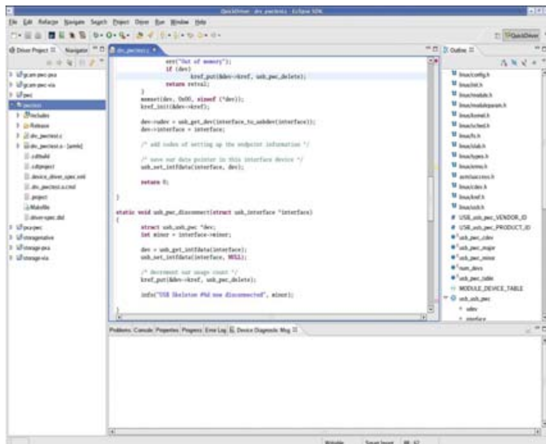
QuickDriver IDE는 크게 이클립스 기반 IDE, 디바이스 드라이버 프로젝트 관리기 및 원격 드라이버 개발 지원 통신 모듈 등으로 구성되어 있다.

• 이클립스 기반 IDE

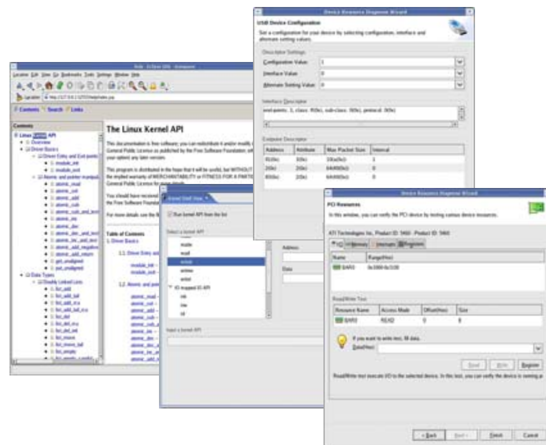
전술한 바와 같이 QuickDriver IDE는 이클립스 CDT를 기반으로 한다. 따라서 (그림 5)와 같이 이클립스 IDE가 제공하는 편의성 및 CDT에서 제공하는 C/C++ 개발 환경의 장점을 그대로 유지하면서 디바이스 드라이버 개발을 위한 커널 빌드 환경을 추가로 지원한다.

• 디바이스 드라이버 프로젝트 관리기

커널 모듈 타입의 디바이스 드라이버 생성을 위한 프로젝트 형식의 관리 기능을 제공하며, 프로젝



(그림 5) QuickDriver IDE



(그림 6) QuickDriver 개발 편의 유틸리티

트 마법사를 이용한 드라이버의 생성이 가능하다. 프로젝트 마법사는 개발자가 특정 드라이버를 개발하기 위해 필요한 정보를 단계적으로 입력할 수 있는 인터페이스를 제공하며, 최종적으로 드라이버 스펙을 생성할 수 있는 자료구조를 만들어낸다. 생성된 드라이버 프로젝트는 KBuild 시스템을 지원하는 Makefile 형식의 빌드가 가능하며, IDE에서 이를 관리한다.

• 원격 드라이버 개발 지원 통신 모듈

드라이버를 개발하는 과정에서 하드웨어 디바이스 리소스 진단 도구 등을 이용하기 위해 원격 타깃 시스템과 통신할 수 있는 프로토콜을 정의하며, 타깃 시스템 통신 관련 설정을 편집할 수 있는 GUI를 제공한다.

통신 프로토콜은 원격으로 타깃 시스템상의 드라이버를 제어하는 데 필요한 기능을 정의하였으며, 각 유틸리티 도구 기능 구현을 위한 PCI, USB, PCMCIA 등 물리적 특성이 다른 디바이스를 제어할 수 있는 프로토콜을 포함하고 있다.

### 나. 개발 편의 유틸리티

개발 편의 유틸리티에는 리소스 진단 도구, 커널 셸, 커널 메시지 로깅 도구, 커널 API 도움말 등이 있으며 (그림 6)은 개발 편의 유틸리티 화면이다.

• 디바이스 리소스 진단 도구

디바이스 리소스 진단도구는 다양한 하드웨어 디바이스 타입별 기본 시스템 리소스에 대한 진단을 통해 드라이버 실행 시 접근할 수 있는 유효한 리소스 범위와 상태를 해당 디바이스 드라이버 없이 검사할 수 있는 기능을 지원한다. 현재는 디바이스 검색/검출기, PCI, USB, PCMCIA, I2C 디바이스를 지원하며, 각 모듈의 사용자 인터페이스는 이클립스 플러그 인으로 구현되어 있다.

리소스 진단도구의 주요 기능을 살펴보면, 먼저 디바이스 검색/검출기는 타깃 시스템에 장착된 디바이스의 목록과 각 I/O 포트, 인터럽트 등 디바이스의 하드웨어 정보를 추출한다.

PCI 디바이스 리소스 진단도구는 타깃 시스템에서 해당 버스를 사용하는 디바이스에 대한 메모리 또는 I/O 포트를 직접 읽고 쓸 수 있다. 또 인터럽트 리소스에 대한 진단 기능을 포함하며, 사용자 레지스터를 정의한 후 정의된 레지스터에 대한 읽기/쓰기 기능을 제공하며, 해당 디바이스가 사용하는 메모리 영역을 덤프하여 직접 편집할 수 있다.

USB 디바이스 리소스 진단도구는 USB 디바이스를 위한 CONFIGURATION 정보 추출과 인터페이스 변경 기능을 갖고 있으며, CONTROL 파이프에 대한 CONTROL 메시지 읽기/쓰기, BULK, INTERRUPT, ISO 파이프를 통한 데이터 읽기/쓰기

기능을 제공한다.

PCMCIA 디바이스 리소스 진단도구는 PCMCIA 디바이스 내의 CIS 정보를 추출하는 기능과 드라이버 바인딩을 통한 PCMCIA 메모리 영역에 대한 읽기/쓰기가 가능하다.

I2C 디바이스 리소스 진단도구는 타깃 시스템상의 I2C 버스에 연결된 디바이스의 메모리 영역에 대한 읽기/쓰기 기능을 제공한다.

• 커널 셸

디바이스 드라이버는 커널의 일부로, 드라이버 작성 시 다양한 커널 API의 활용이 필수적이다. 그러나 각각의 커널 API 실행결과를 확인하기 위해서는 복잡한 매개변수 값을 지정하고, 매번 빌드 및 커널에 적재하여 실행하는 과정을 반복해야 한다. 커널 셸은 이러한 커널 API들을 개발하고자 하는 드라이버 코드에 포함시키기 전에 미리 실행결과를 확인할 수 있는 대화식 커널 API 실행환경이다.

• 커널 메시지 로깅 도구

디바이스의 설치/해제나 디바이스 드라이버의 적재/실행/제거 과정에서 발생하는 커널 메시지를 확인하는 것은 디바이스 드라이버 개발자들이 드라이버 개발 과정에서 빈번하게 행하는 작업으로 원격 타깃 시스템에서 출력되는 커널 메시지를 통합개발 환경에서 바로 확인할 수 있는 기능을 제공한다.

• 커널 API 도움말

디바이스 드라이버 코드 작성에서 다양하게 활용되는 커널 API들의 사용법을 오프라인의 텍스트 없이도 통합 개발 환경에서 색인을 이용하나 검색 기능을 통해 편리하게 참고할 수 있도록 커널 API 도움말 뷰어를 제공한다.

## V. 관련 도구 간의 비교

<표 3>은 QuickDriver와 기타 도구와의 비교이다. QuickDriver는 (Embedded) Linux용 디바이스 드라이버 개발 도구이고, 다른 도구는 Windows용 또는 Linux용 디바이스 드라이버의 개발 도구이다. 소스 코드를 생성함에 있어서는 QuickDriver와 WinDriver/KernelDriver만이 하드웨어 정보를 반영한 소스 코드를 생성한다. 또한 생성된 소스 코드가 컴파일되고 디바이스 드라이버 프로그램(모듈)로 만들어질 때, QuckDriver를 제외한 다른 도구들은 도구에서 정의되어 있는 라이브러리가 생성되는 디바이스 드라이버 프로그램에 필요하다. 하지만, 이와 같이 도구에서 만들어진 라이브러리에 의존적인 프로그램은 일반적으로 효율이 떨어지는 것으로 알려져 있다. 하지만, QuickDriver에서 생성되는 소스 코드는 사전에 정의된 라이브러리에 의존성이 없다.

〈표 3〉 QuickDriver와 다른 도구와의 비교

| | DriverStudio | WinDriver/KernelDriver | DDK | QuickDriver |
|---|---|---|---|---|
| 지원 운영 체제 | Windows | Windows/Linux | Windows | Linux |
| 소스 코드 생성 범위 | 골격 코드 | 골격 코드+ 하드웨어 정보 코드 | 없음 | 골격 코드+ 하드웨어 정보 코드 |
| 자체 라이브러리 의존성 | 있음 | 있음 | - | 없음 |
| 테스팅 기능 | 커버리지 출력 | 없음 | 기능 테스팅 | 기능 테스팅, 커버리지 출력 |
| 정형 검증 기능 | 없음 | 없음 | API 규칙 검증(시험판) | API 규칙 검증 |
| 자체 통합 개발 환경 | 없음 | 없음 | 없음 | 있음 |
| 유틸리티 도구 기능 | 적음 | 적음 | 적음 | 많음 |
| 소스수준 디바이스 드라이버 디버거 | 있음 | 없음 | 없음 | 있음 |

QuickDriver는 기능 테스트의 수행 기능과 함께, 소스 코드의 어느 부분이 수행되었는지를 나타내는 지표인 커버리지를 출력하는 유일한 도구이다. 정형 검증 도구는 최근에 연구가 진행되는 부분이기 때문에 DDK에서도 현재 시험판만이 공개된 상태이고, 역시 QuickDriver에서 지원하고 있다.

통합 개발 환경은 다른 도구에서는 부분 개발 환경 도구와 통합한 경우는 있지만, QuickDriver와 같이 유틸리티 기능을 포함한 디바이스 드라이버 개발을 위한 전체적인 과정이 통합된 환경을 제공하는 도구는 없다.

개발 유틸리티 또한 다른 도구에 비해서 제공되는 기능이 많으며, 소스 수준의 디바이스 드라이버 디버거가 제공된다.

이상의 비교로 QuickDriver는 다른 도구와 비교했을 때 우수한 디바이스 드라이버 개발 도구임을 판단할 수 있다.

## Ⅵ. 결론

임베디드 소프트웨어의 개발에서 어려움이었던 디바이스 드라이버 개발 도구가 갖추어야 할 기능을 소스 코드 자동생성 기능, 테스팅 기능, 정형 검증 기능, 통합 개발 환경 및 개발 편의 유틸리티 기능의 측면에서 살펴보았다. 그리고, 현재 상용 디바이스 드라이버 도구인 DriverStudio, WinDriver/KernelDriver, DDK와 ETRI의 본 연구팀에서 개발한 QuickDriver에 대해서 살펴보았으며, 이를 통해 도구들 간의 비교를 수행하였다. 이 비교를 통하여 QuickDriver가 기능이 풍부한 디바이스 드라이버 개발도구임을 판단할 수 있었다. 향후 QuickDriver가 계속적으로 발전되어 임베디드 소프트웨어 개발의 가장 큰 장벽이었던 디바이스 드라이버의 개발이 더욱 쉬워질 것으로 기대한다.

## 약어 정리

| | |
|---|---|
| API | Application Programming Interface |
| CDT | C/C++ Development Toolkit |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| ISA | Industry Standard Architecture |
| MSDN | Microsoft Developer Network |
| PCI | Peripheral Component Interface |
| PCMCIA | Personal Computer Memory Card International Association |
| USB | Universal Serial Bus |
| WDM | Windows Driver Model |
| CIS | Card Information Structure |

## 참 고 문 헌

[1] Compuware, DriverStudio, http://www.compuware.com/products/driverstudio/default.htm

[2] Jungo, WinDriver, http://www.jungo.com/windriver.html

[3] Jungo, KernelDriver, http://www.jungo.com/kernel-driver.html

[4] Microsoft, Windows Driver Development Kit, http://www.microsoft.com/whdc/devtools/ddk/default.mspx