ANTLR 을 이용한 임베디드 시스템 테스트 스크립트 언어 구현 방안

신현규, 이재승, 최종욱, 천이진
한국항공우주연구원
e-mail : hkshin@kari.re.kr

A Study on Implementation of Test Script Language for Embedded System using ANTLR

Hyun-Kyu Shin, Jae-Seung Lee, Jong-Wook Choi, Yee-Jin Cheon
KARI

요 약

위성 전체 시스템의 동작과 임무 수행을 책임지고 있는 위성 탑재 소프트웨어의 개발 과정에서 위성 탑재 컴퓨터를 이해하고 소프트웨어가 동작하기 위한 환경을 구성하는 작업은 필수적인 과정이다. 위성 탑재 소프트웨어 개발의 초기 과정은 하드웨어와 매우 밀접하게 관련되어 있으며, 이러한 하드웨어의 동작을 보다 쉽게 테스트하기 위한 환경이 필요하게 된다. 최근 위성 탑재 컴퓨터로 널리 쓰이고 있는 LEON 2/3 플랫폼은 AHB-UART 를 이용하여 Memory 에 대한 직접적인 R/W Operation 을 지원하고 있는데, 본 논문에서는 이 기능을 이용하여 위성 탑재 컴퓨터를 보다 쉽게 테스트할 수 있는 테스트 스크립트 언어의 구현 방안에 대하여 기술하며, 더불어 이러한 테스트 언어의 구현에 있어 ANTLR 을 이용하는 방안도 함께 소개한다.

1. 서론

위성 탑재 소프트웨어는 위성 전체 시스템의 동작을 제어하며, 지상으로부터의 명령 수신 및 위성 데이터의 전송, 위성 임무 수행을 책임지게 된다. 이러한 위성 탑재 소프트웨어의 개발 과정에는 위성 탑재 컴퓨터에 대한 하드웨어적인 이해 및 탑재 소프트웨어의 초기 구동을 위한 BSP 의 작성, Real-time OS 구성 등의 과정이 필수적이다. 이러한 과정에서는 하드웨어와 매우 밀접한 작업을 수행하게 되는데, 일반적으로 하드웨어를 테스트하기 위하여 크로스 컴파일러를 이용하여 테스트 프로그램을 작성하고, 이를 해당 하드웨어에 로드하고 그 결과를 분석하는 방법으로 초기 개발을 진행하게 된다. 최근 위성 탑재 컴퓨터로 널리 쓰이고 있는 LEON 2/3 플랫폼의 경우, AHB-UART 와 APB/AHB Bridge 를 이용하여 Memory 에 대한 직접적인 R/W Operation 을 지원하고 있다. 본 논문에서는 이를 이용하여 Leon 2/3 플랫폼 기반의 임베디드 시스템에서 하드웨어적인 테스트를 수행할 수 있는 테스트 스크립트 언어의 구현 방안에 대하여 기술한다.

2. ANTLR

ANTLR(Another Tool for Language Recognition)은 번역기(Translator), 컴파일러, 구문 인식기, 정적/동적 프로그램 분석기 등의 소프트웨어 툴을 만드는데 사용될 수 있는 도구이다. ANTLR 을 이용하면 이러한 프로그램을 만들고 유지하는 데 드는 노력을 상당 부분 줄일 수 있다. 일반적인 관점에서 보면, ANTLR 은 Lex/Flex 와 Yacc/Bison 으로 대변되는 컴파일러 생성기나 컴파일러를 위한 컴파일러(Compiler Compiler)라고 할 수 있다. ANTLR 의 입력은 문법(Grammar)이 되며, 이를 통해 소스 코드와 다른 보조적인 데이터를 만들어 내게 된다. 생성된 소스 코드의 대상 언어(Target Language)는 문법 안에 기술되게 된다. ANTLR 을 이용하면 Java 뿐 아니라, C/C++, C#, Python, Ruby 등의 환경에서 작업을 진행 할 수 있다. 또한 DSL(Domain Specific Language)을 구현하기 위해 ANTLR 을 사용할 수 도 있다. ANTLR 은 Java 환경에서 쉽게 사용할 수 있으며, Eclipse 를 위한 Plug-in 및 그림과 같이 ANTLRworks 라는 자체 IDE 를 지원하고 있다.

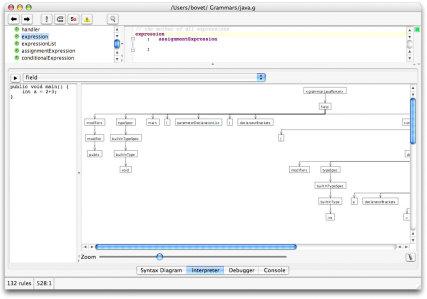

그림 1 ANTLRworks

본 연구에서는 임베디드 시스템을 하드웨어적으로 테스트하기 위한 테스트 스크립트 언어를 구현하기 위하여 ANTLR 을 이용하였다.

3. 실행 환경

LEON 2/3 플랫폼을 이용하여 위성 소프트웨어를 개발하는 초기 과정에서의 테스트 및 디버깅은 아래와 같은 형태로 이루어지게 된다.

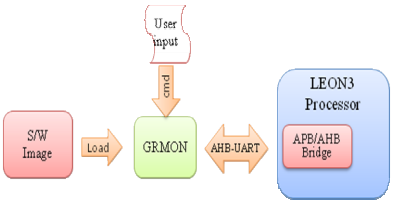

그림 2 GRMON 을 이용한 Testing/Debugging

그림에서와 같이 사용자가 작성한 테스트 프로그램을 크로스 컴파일러를 통해 타겟에서 실행 가능한 형태로 빌드한 후, GRMON 을 통하여 로드하여 이를 테스트하게 된다.

본 연구에서 논의하고 있는 테스트 환경은 아래와 같이 테스트 언어를 이용하여 별도의 컴파일 및 타겟으로의 로드 없이 자체 인터프리터(그림에서는 LEON3-RT-MON)를 이용하여 테스트를 수행하게 된다.

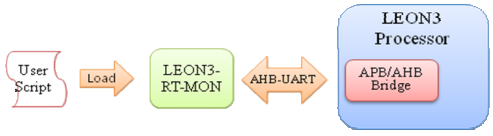

그림 3 테스트 스크립트 언어를 이용한 Testing/Debugging

이러한 테스트 스크립트 언어를 이용하는 경우, 보다 손쉽게 하드웨어를 직접적으로 테스트 할 수 있다.

4. 요구 사항

위성 탑재 컴퓨터를 테스트하기 위한 테스트 스크립트 언어 (이하 TSLL: Test Script Language for Leon) 구현을 위한 주요 요구 사항은 아래와 같다.

- 개발자에게 친숙한 C 언어를 기반으로 할 것 – 변수 선언, 대입문, 제어구문 등
- AHB-UART 를 통해 전달되는 Memory Operation 을 위한 별도의 Built-in 함수를 지원할 것
- TSLL 을 이용하여 구현되는 테스트 환경(예: LEON3-RT-MON)에서의 GUI 및 Target 에 연결된 UART 를 통한 출력을 지원할 것
- 테스트의 용이성을 위해 2 진수, 8 진수, 16 진수의 표기 및 관련 연산 함수를 지원할 것

5. 개발 및 테스트 환경

본 연구에서는 Windows XP 환경에서 Java SE 1.6, Eclipse 3.6.1, ANTLR 3.3 을 이용하여 개발하였으며, Target 과의 연결을 위한 호스트 역시 Windows XP 환경에서 테스트를 수행하였다.

6. ANTLR 을 이용한 Grammar 작성 및 Tree 구현

Lex/Yacc 등을 이용하여 스크립트에 대한 인터프리터를 작성하는 경우, Yacc 의 parser grammar 내에 처리 루틴을 삽입하는 것이 일반적이다. 또한, 스크립트의 내용이 반복적으로 수행되거나, 성능 요구 사항을 만족하기 위하여 Intermediate Form (일반적으로 Tree 형태) 을 만들고 이를 다시 해석하는 형태로 구현되기도 한다.

TSLL 의 경우, 후자의 방안을 선택하여, TSLL 의 처리 과정에서 일차적으로 Parsing Tree 를 만들게 되고 이를 다시 해석, 관련 Action 을 수행하도록 하였다.

MemScript.g

```
block
   : lc='{' variable* stat* '}'
     -> ^(SLIST[$lc,"SLIST"] variable* stat*)
   ;

forStat
   : 'for' '(' first=assignStat ';' expr ';' inc=assignStat ')' block
     -> ^(FOR $first expr $inc block)
   ;

ifStat
   : 'if' '(' expr ')' tBlock=block  ('else' fBlock=block)?
       -> ^(IF expr $tBlock $fBlock?)
   ;

whileStat
   : 'while' '(' expr ')' block -> ^(WHILE expr block)
   ;

assignStat
   : ID '=' expr -> ^(ASSIGN ID expr)
   ;
```

그림 4 Grammar for TTSL

그림 4 에서와 같이 Eclipse 환경에서 TTSL 을 위한 Grammar 를 작성하였고, TTSL 에서 요구되는 기본 요구 사항인 C 스타일의 문법을 적용하였다. 각 grammar rule 은 실제 처리 과정에서 쓰일 Tree 를 만들기 위하여 Rewrite 방법을 사용하였다.

```
1  UINT32 ptr;
2  UINT32 value;
3  UINT32 i;
4
5  ptr = 0x4000000;
6
7  for(i=0;i<0x100;i=i+1)
8  {
9     ptr = ptr + i * 4;
10    value = mem(ptr);
11    print("ADDRESS: " , hex(ptr), " --> ", hex(value) );
12
13 }
14
```

그림 5 Test Script #01

그림에 표시한 Test Script #01 은 세 개의 변수를 이용

하여 0x40000000 부터 0x400003FC 까지의 메모리 값을 읽는 테스트 스크립트이다. 이에 대한 파싱 트리는 다음과 같다.

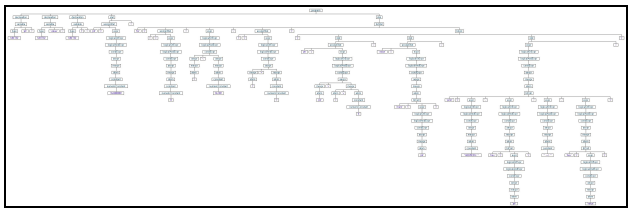

그림 6 Parsing Tree for Test Script #01

Test Script #01 에 대한 Tree 표현식은 아래와 같다.

```
(PROGRAM
  (VAR UINT32 ptr)
  (VAR UINT32 value)
  (VAR UINT32 i)
  (ASSIGN ptr 0x4000000)
  (FOR
    (ASSIGN i 0)
    (< i 0x100)
    (ASSIGN i (+ i 1))
    (SLIST
          (ASSIGN ptr (+ ptr (* i 4)))
          (ASSIGN value (MEM ptr))
          (PRINT "ADDRESS: " (HEX ptr) " --> " (HEX value)))))
```

그림 7 Tree Representation for Test Script #01

7. 테스트

KARI-MON (앞서 기술한 LEON3-RT-MON 의 Prototype, TTSL 을 스크립트 해석에 사용) 이라는 임베디드 시스템 테스트 툴을 이용하여 실제 하드웨어에 대한 테스트를 수행하였다.

```
UINT32 a;
UINT32 b;
UINT32 i;
UINT32 j;

a= 0x40000000;
b= 0x12345678;

msg("Fill memory as 0x",hex(b));

for(j=0;j<0x10;j=j+1)
{
   wmem( a + (j*4), b);
}

msg("Reading Memory:");

for(j=0;j<0x10;j=j+1)
{
 b = a + (j*4);
 msg( "0x",hex(b),":0x", hex(mem( a + (j*4))));
 print("SCU-DM alive!");
 wait(500);
}
```

그림 8 Test Script #02

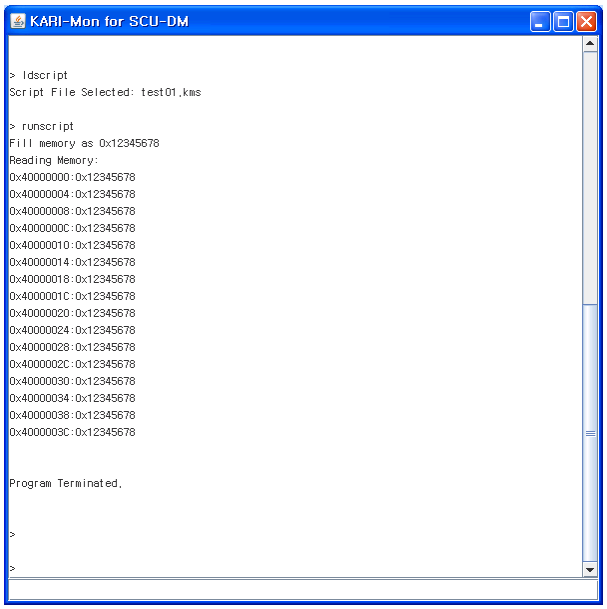

그림 9 Test on KARI-MON

그림 9에서 보는 바와 같이, 메모리에 대한 R/W 가 정상적으로 수행되며, 제어 구문이 올바르게 동작함을 확인할 수 있었다.

8. 결론

본 논문에서는 LEON 2/3 플랫폼에서 별도의 테스트 프로그램을 작성하고 컴파일할 필요없이 테스트 스크립트만을 작성하여 실행함으로써 테스트를 수행할 수 있는 환경을 구성하기 위한 TTSL 의 구현 방안에 대하여 알아보고, KARI-MON 을 이용하여 실제 하드웨어에 대한 테스트를 수행해보았다.

DSL 을 구현하는 측면에서, ANTLR 은 임베디드 시스템 테스트를 위한 테스트 스크립트 언어 구현에 필요한 기능 요구 사항을 만족하였고, Eclipse Plug-in 을 비롯하여, ANTLRworks 등은 개발 과정에서 상당한 편의성을 제공해주었다.

현재 개발된 TTSL 은 위성 탑재 컴퓨터를 테스트하기에는 부족한 점들이 많이 있으나, 그 가능성에 대한 검증의 측면에서 상당한 진전을 이루었다고 판단되며, TTSL 의 상세 명세 및 구현, 위성 소프트웨어의 초기 개발 과정 및 테스트 과정에 유용하게 사용될 수 있는 추가 기능 구현 등에 대한 추가 연구가 가능할 것으로 기대된다.

참고문헌

[1] Terence Parr, “The Definitive ANTLR Reference”
[2] Terence Parr, “Language Implementation Patterns”