2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

Test System용 가상기계 설계

고훈준, 안용균, 조선문, 유원희
인하대학교 전자계산공학과
g2001418@inhavision.inha.ac.kr

Design of a Virtual Machine for the Test System

Hoon-Joon Kouh, Yong-Koun Ahn, Sun-Moon Jo, Weon-Hee Yoo
Dept. Of Computer Science Engineering, Inha University

요약

테스트 시스템(Test System)은 반도체 제품을 웨이퍼(Wafer) 또는 완성된 제품 상태 하에서 전기적 특성과 성능을 검사하고 그 결과를 산출해내는 검사장치이다. 테스트 시스템은 크게 하드웨어와 소프트웨어로 이루어져 있으며 시스템을 제어하고 사용자 인터페이스 및 각종 자료를 처리하는 소프트웨어는 그 중요성이 한층 더 부각되고 있다. 그러나 국내 고성능의 테스트 시스템을 개발하는 기업들의 하드웨어 개발은 잘 이루어지고 있으나 소프트웨어의 개발은 어려운 실정이다.

본 논문에서는 테스트 시스템에서 사용하고 있는 테스트 프로그램의 문제점을 지적하고, 문제점을 해결할 수 있는 가상기계를 설계한다. 그리고 가상기계를 테스트 관리 프로그램 내에 내장하여 테스트 관리 시스템의 소프트웨어를 향상시키고자 한다.

1. 서 론

최근 반도체 산업의 발전으로 반도체 종류의 다변화와 생산량이 크게 증가하고 있으며 반도체가 고집적, 고성능화 됨에 따라 세계 반도체업계의 테스트 기술에 대한 관심과 기술은 점차 발전하고 있다.

테스트 시스템(Test System)은 반도체 제품을 웨이퍼(Wafer) 또는 완성된 제품 상태 하에서 전기적 특성과 성능을 검사하고 그 결과를 산출해내는 검사장치이다. 최근까지도 테스트 시스템은 그 고유의 기술적 장벽으로 몇몇 유명한 외국 시스템업체가 세계시장 및 국내시장을 독점하고 있으며 최근 국내에서도 독자적인 기술력으로 테스트 시스템을 개발하여 국산화를 이루는 업체가 자생하고 있다[5,6].

테스트 시스템은 크게 하드웨어와 소프트웨어로 이루어져 있으며 시스템을 제어하고 사용자 인터페이스 및 각종 자료를 처리하는 소프트웨어는 그 중요성이 한층 더 부각되고 있다. 테스트 시스템은 다양한 종류와 기능의 반도체 제품을 검사한다. 따라서 여러 가지 특수기능의 하드웨어 모듈(Module)과 각 제품의 특성에 맞게 프로그램이 된 테스트 프로그램이 필요하다. 또한 그 시스템에 알맞은 프로그래밍 언어와 테스트 프로그램을 실행하기 위한 컴파일러가 필요하다[5].

* 본 논문의 연구는 2000년도 정보통신부 대학기초 연구지원 사업에 의해 수행되었음.
* 본 논문의 연구는 2000년도 STATEC Inc.의 산학 위탁연구 지원 사업에 의해 수행되었음

전 세계적으로 가장 큰 시장점유율을 갖고있는 TERADYNE INC.는 테스트 프로그램 언어로 파스칼 언어를 기반으로 한 PASCAL/STEPS 와 C++ 언어를 기반으로 개발한 언어를 사용하고 있다. 이 두 언어는 전 세계적으로 TERADYNE 테스트 시스템을 사용하고 있는 수많은 테스트 엔지니어들에 의해 사용되고 있으며 그 편리성과 안전성을 인정받고 있다. 현재 많은 반도체 제품을 생산하는 기업국내 기업 중에서도 TERADYNE의 언어 형식을 표준으로 컴파일러를 개발하여 사용하고 있는 기업이 있다[4,5].

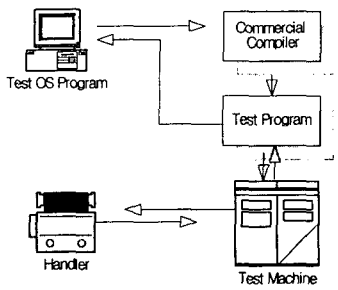
국내 벤처기업인 STATEC INC.는 지금까지 고성능의 테스트 시스템을 개발하여 왔으나 테스트 프로그램은 상용 C 컴파일러를 이용하여 테스트 시스템에 적용하여 왔다[6].
그러나 이는 다음과 같은 몇 가지 문제점을 안고 있다.

첫째, 테스트 실행 시 매번 프로세스를 생성한다.
둘째, 시스템 관리 프로그램(OS)과 프로세스간의 데이터 처리가 자유롭지 못하다.
셋째, 테스트 엔지니어에게 불편한 시스템 제어명령을 제공한다.

본 논문에서는 이와 같은 문제를 해결하기 위해 새로운 가상기계(virtual machine)를 설계한다.

2. 기존 테스트 시스템의 구조

기존 테스트 시스템의 구조는 [그림 1]과 같다.



[그림 1] Test 시스템의 구성도

시스템 관리 프로그램(System Management Program)은 테스트 기계(Test Machine)를 제어하고 프로그램을 컴파일하여 실행하는 OS이다. 테스트 프로그램(Test Program)은 반도체 제품의 종류마다 사용되는 여러 가지 프로그램이다. 핸들러(Handler)는 반도체 제품과 연결되어 제품을 측정하고 구별하는 장치이다[6].

테스트 시스템의 동작 구조는 다음과 같다.

첫째, 테스트할 반도체 제품의 특성에 맞는 테스트 프로그램을 작성하고 컴파일한다.
둘째, 반도체 제품의 준비가 이루어지면 테스트 프로그램을 실행하여 그 특성을 검사한다.
셋째, 측정된 제품의 특성에 따라 핸들러가 제품을 구별한다.

이때, C 언어를 이용하여 프로그래밍을 하고 기존의 상용 C 컴파일러를 이용하여 실행 파일을 만들어 사용하면 다음과 같은 몇 가지 문제점이 있다.

첫째, 테스트 실행 시 매번 프로세스를 생성한다. 이는 전체적인 테스트 시간을 증가시키는데 영향을 준다.
둘째, 시스템 관리 프로그램(OS)과 프로세스간의 데이터 처리가 자유롭지 못하고 콘솔(console)로 실행이 된다. 이는 시스템의 유연성을 저하시킨다.
셋째, 테스트 엔지니어에게 불편한 시스템 제어명령을 제공한다. 표준 C 언어의 구문 형태로 시스템 제어명령을 제공하게 되므로 기존 시스템에 비하여 불편하다. 다음은 제어 명령에 대한 예이다.

SET S1 V= 10V I=10MA; → set_source(1, 10*V, 10*MA);

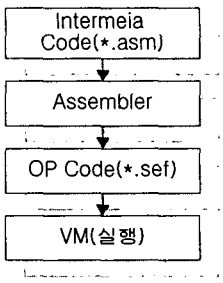
위의 예와 같이 기존 외국 시스템은 앞의 문장을 사용하고 국내 개발 시스템은 뒤의 문장을 사용하고 있다. 따라서 앞의 문장에 익숙한 엔지니어에게는 두 번째 문장이 불편하다. 이를 해결하기 위해 T 언어를 설계하였다[7].

본 논문에서는 첫째와 둘째의 문제점을 해결하기 위해 새로운 가상기계를 설계하고 테스트 관리 프로그램에 가상기계를 내장시킨다.

3. 실행 구조

본 장에서는 테스트 시스템에 새롭게 사용하는 프로그래밍 언어 T를 실행 할 수 있는 가상기계를 설계하였다. [그림 2]는 본 논문에서 설계한 구조이다. [그림 2]의 중간코드는 스택 가상기계에서 실행 가능한 명령어들로 구성된다[7].

중간코드는 어셈블러에 의해 숫자 형식으로 구성된 op code로 변환이 되고 가상기계에서 op code를 입력으로 실행이 된다. 따라서 한번만 op code를 생성하면 이 코드를 이용해서 여러 번 실행할 수 있기 때문에 매번 프로세스를 생성하던 기존 구조에 비해 매우 효과적이다.



[그림2] 제안한 실행 구조

4. 중간코드

중간코드는 스택 가상기계에서 수행하기 위한 명령어이다. 이 명령어는 기본 명령어와 하드웨어 제어구문을 위한 명령어로 구분한다.

[표 1]은 본 논문에서 사용한 중간언어의 기본 명령어이다.

[표 1] 기본 명령어

№	명령어	№	명령어	№	명령어	№	명령어
1	IADD	16	ILSSEQL	31	IPUSH variable	46	SETSP length
2	ISUB	17	INOT	32	IPOPC variable	47	SETP
3	IMUL	18	DEQUAL	33	IPOP	48	RDCHAR
4	IDIV	19	DNOTEQL	34	DPUSHC double value	49	RDINT
5	INEG	20	DGREATER	35	DPUSH variable	50	RDDOUBLE
6	DADD	21	DLESS	36	DPOPC variable	51	WRCHAR
7	DSUB	22	DGTREQL	37	DPOP	52	WRINT
8	DMUL	23	DLSSEQL	38	AREF size	53	WRDOUBLE
9	DDIV	24	DNOT	39	BRANCH address	54	CONTENTS
10	DNEG	25	PUSHC char value	40	JUMP	55	ICONTENTS
11	IEQUAL	26	PUSH variable	41	BREQL address	56	DCONTENTS
12	INOTEQL	27	PUSHL offset	42	BRLSS address	57	HALT
13	IGREATER	28	POPC variable	43	BRGTR address	58	I2D
14	ILESS	29	CPOP	44	CALL address	59	D2I
15	IGTREQL	30	IPUSHC integer value	45	RETURN		

IADD, DADD에서 I는 정수형을 뜻하고, D는 실수형을 뜻한다. PUSHL은 현재의 FramePtr와 offset을 더해서 그 값을 스택에 PUSH하는 명령어이다. SETSP는 스택에 [length]만큼 공간을 확보하고 스택 포인터를 바꾸고, SETP는 현재의 스택 포인터를 지역변수의 시작주소로 하는 명령어이다. CONTENTS는 스택에서 POP한 값을 주소로 하여 그 주소의 값을 스택에 저장하는 명령어이다. 또한 I2D는 스택에 있는 값을 POP해서 그 값을 실수 값으로 타입 변환 후 스택에 저장하는 명령어이고 D2I는 반대로 스택에 있는 값을 POP해서 그 값을 정수 값으로 타입 변환 후 스택에 저장하는 명령어이다. 번호는 op code 번호를 나타낸다.

[표 2]는 시스템 하드웨어의 제어 명령어이다. 명령어 형식은 '명령어 [파라미터 개수] [타입, …]'이다. 만약 하드웨어 제어 명령어들을 직접 프로그램 소스에 넣어 컴파일하고 실행할 경우 프로그램 크기가 매우 커져서 속도에 영향을 미치게 된다. 따라서, 하드웨어 제어 명령어는 C언어 형식의 함수로 만들어서 라이브러리로 만들고 [표 2]의 명령어들로 라이브러리의 함수를 호출하도록 하였다.

[표 2] 제어 명령어

No	명령어	No	명령어	No	명령어
59	RESET_CBIT	87	SET_IAS_START	115	SET_SFT_ACTIVE
60	SET_CBIT	88	SET_IAS_VGON	116	RESET_TIMER
61	READ_CBIT	89	READ_IAS_DATA	117	READ_TIMER
62	SET_CBIT_MASK	90	GET_IAS_IH	118	SET_TIMER_START
63	READ	91	GET_IAS_IL	119	SET_TIMER_STOP
64	RESET	92	SET_IAS_CLR	120	SET_TIMER_TRIGGER
65	RESETALL	93	SET_IAS_PULSE	121	SET_TIMER_FILTER
66	WAITTIME	94	RESET_MUX	122	SET_TIMER_MODE
67	SETPOWER	95	PROGRAM	123	SET_TIMER_TIMEOUT
68	DELAY	96	LIMIT	124	SET_TIMER_PRESCALER
69	INITSYSTEM	97	SORT_BIN	125	SET_TIMER_BCHANNEL
70	AVGREADMETER	98	CALC	126	SET_TIMER_ECHANNEL
71	LIBERROR	99	PRINT	127	SET_TIMER_BSLOPE
72	RESET_HSOURCE	100	HW_STATUS	128	SET_TIMER_ESLOPE
73	RESET_HVS	101	TRAP	129	SET_TIMER_PSLOPE
74	SET_HSOURCE_V	102	RESET_SOURCE	130	SET_TIMER_V
75	SET_HSOURCE_I	103	SET_SOURCE_VR	131	SET_TIMER_CHANNEL
76	SET_HSOURCE_VR	104	SET_SOURCE_IR	132	SET_TIMER_SLOPE
77	SET_HSOURCE_IR	105	SET_SOURCE_FORCE	133	SET_TIMER_PERIOD
78	SET_HSOURCE_FORCE	106	SET_SOURCE_LINE	134	RESET_METER
79	SET_HSOURCE_LINE	107	SET_SOURCE_V	135	READ_METER
80	SET_HSOURCE_VI	108	SET_SOURCE_I	136	SET_TIMER_INPUT
81	SET_HSOURCE_VVI	109	SET_SOURCE_VI	137	SET_TIMER_VRANGE
82	SET_HSOURCE_IIR	110	SET_SOURCE_VVR	138	SET_TIMER_FILTER
83	SET_HSOURCE	111	SET_SOURCE_IIR	139	SET_TIMER_INVR
84	RESET_IAS	112	SET_SOURCE	140	SET_TIMER
85	SET_IAS_IH	113	SET_FTIME	141	SET_TIMER_SADI
86	SET_IAS_IL	114	SET_POWER_MODE	142	SET_TIMER_SADV

[표 1]과 [표 2]의 명령어들은 스택머신에서 수행 가능하도록 정의된 명령어들이다. 따라서 이와 같은 명령어 코드로 생성만 하면 하드웨어에 상관없이 본 논문에서 설계한 스택머신이 있는 컴퓨터 어디에서나 실행이 가능하다. 그러나 중간코드 생성을 위해서는 소스 프로그램 언어에 대한 중간코드의 구성이 필요하다. 다음 [표 3]은 산술문, 조건문, 루프문의 소스 프로그램 언어에 대한 중간코드의 일반적인 표현들이다.

[표 3] 기본 구문에 대한 중간코드

a = a + 1;	if(a > 2) stmt else stmt	while(a < 100) stmt
PUSHL -4 ICONTENTS IPUSHC 1 IADD PUSHL -4 IPOP	IPUSHC 2 IPUSHC a ICONTENTS IGREATER BREQL L1 (code for stmt) BRANCH L2 L1: (code for stmt1) L2:	L1: IPUSHC 100 IPUSHC a ICONTENTS ILESS BREQL L2 (code for stmt) BRANCH L1 L2:

for(i=0;i<100;i=i+1) stmt

IPUSHC 0	(code for stmt)
IPUSHC i	IPUSHC i
IPOP	ICONTENTS
L1:	IPUSHC 1
IPUSHC 100	IADD
IPUSHC i	IPUSHC i
ICONTENTS	IPOP
ILESS	BRANCH L1
BREQL L2	L2:

[그림 3]은 정수 5와 6을 덧셈하는 프로그램에 대한 중간코드이다.

```
CALL    _main       IPOP
HALT                PUSHL  -4
_main:              ICONTENTS
SETSP   12          PUSHL  -8
IPUSHC  5           ICONTENTS
PUSHL   -4          IADD
IPOP                PUSHL  -12
IPUSHC  6           IPOP
PUSHL   -8          RETURN 0
```

[그림 3] 중간코드(sample.asm)

[그림 3]의 프로그램은 CALL 명령어로 시작하여 _main을 호출하고, SETSP에서 스택에 공간을 확보한다. IPUSHC 명령어로 5와 6를 스택에 넣고 IADD 명령어로 두 값을 더한다. 그리고 RETURN을 만나면 다시 돌아와 다음 줄의 HALT 명령을 실행함으로써 종료된다.

4. 어셈블러와 가상기계

4.1 어셈블러

어셈블러는 [그림 3]과 같은 중간 코드를 가상머신에서 실행하기 위해 숫자형식의 op code로 변환을 하는 프로그램이다. 이는 two pass로 이루어져 있다. one pass에서는 중간 코드(*.asm)로부터 먼저 변수와 레이블의 이름과 주소 값을 레이블 테이블에 저장하고, two pass에서는 중간 코드와 레이블 테이블을 사용하여 가상기계에서 실행할 수 있는 op code를 생성한다.

[그림 4]는 [그림 3]에 대한 op code 이다.

2	57	44	6	0	0	0	57	46	12
0	0	0	30	5	0	0	0	27	252
255	255	255	33	30	6	0	0	0	27
248	255	255	255	33	27	252	255	255	255
55	27	248	255	255	255	55	1	27	244
255	255	255	33	45	0	0	0	0	

[그림 4] [그림 3]의 op code

op code의 처음 숫자 2는 시작주소를 나타내고 두 번째 숫자 57은 머신코드의 개수를 나타낸다. 나머지 숫자는 머신 코드이며 명령어와 인자로 구성되어 있다. 예를 들면, 세 번째 숫자 44는 CALL 명령어이므로 다음의 6은 다음 명령어의 주소를 지시한다. op code에서 스택에 저장되는 정수(옵셋, 주소)는 4 바이트로 저장된다. 예를 들어 정수 2이면 '2 0 0 0'으로 저장이 된다.

4.2 가상기계

가상기계는 스택머신으로 설계하였으며 다음 여섯 가지 기본적인 전제조건을 가지고 설계하였다.

첫째, 메모리가 unsigned char 일차원 배열로 선언하였다.

둘째, 변수 타입은 char, int, double만 허용된다.

셋째, 다중 배열은 안 되고 일차원 배열은 가능하다.

넷째, 실행코드는 메모리 처음부터 저장을 하고, 스택을 사용할 때는 메모리의 끝에서부터 사용한다.

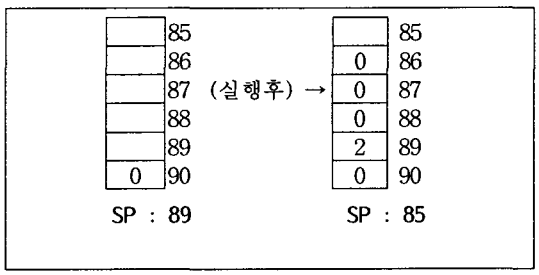
다섯째, 레지스터는 프로그램 카운터, 스택포인터, 프레임, 포인터, 지역변수 시작 포인터가 있고 함수를 복귀할 때 값을 저장할 버퍼가 있다.
여섯째, 명령어는 1 바이트, 주소는 4 바이트, 정수는 4 바이트, 실수는 8 바이트이다.

여섯 가지 전제 조건을 가지고 가상 기계가 실행되는 과정은 [알고리즘 1]과 같다.

[알고리즘 1] 가상기계 수행 알고리즘

입력 : op code
1. op code에서 실행코드를 메모리에 저장한다.
2. 메모리로부터 코드 한 개 씩 읽는다.
 2.1 코드가 명령어이면 해당하는 명령어 함수를 호출한다.
 2.2 코드가 명령어 다음에 인자이면 명령어 함수에 인자를 넘긴다.
 2.3 명령어가 57이면 HALT이므로 프로그램을 종료한다.
 2.4 명령어를 실행하면 프로그램 카운터를 1 증가시키고, 오퍼랜드가 정수이면 4를 실수이면 8을 증가시킨다.
 2.5 스택을 사용할 경우에는 스택 포인터를 감소시킨다.
3. 스택과 메모리로부터 모든 정보를 삭제한다.

[알고리즘 1]의 2.4는 전제조건 여섯 번째와 같이 프로그램 카운터를 해당하는 크기만큼 증가시키는 것이다. 2.5의 경우는 스택의 주소가 스택의 증가 방향과 반대이기 때문에 스택 포인터가 감소하는 것이다. 예를 들어 명령어 'ipushc 2' 인 경우는 op code가 '30 2 0 0 0'이다. 이 경우 스택 포인터(SP)의 변화는 [그림 5]와 같다.



[그림 5] 스택 포인터의 이동

현재 스택 포인터가 89 라고 가정할 때 'ipushc 2'를 실행하면 [그림 5]와 같이 정수 2가 4 바이트로 저장이 되기 때문에 스택 포인터가 85로 변한다.

5. 결론

기존 테스트 시스템 장비는 테스트 관리 프로그램과 컴파일러가 분리되어 있어 프로그램을 컴파일하면 실행파일이 생성되어 시스템과의 유연성이 떨어졌다. 그래서 본 논문에서는 기존 테스트 관리 프로그램 내에 내장하여 사용할 수 있는 가상기계를 설계하여 기존 테스트 시스템 장비의 문제점을 해결하였다. 에디터와 컴파일러를 별도로 사용하지 않고 시스템 관리 프로그램 내에서 에디터를 이용하여 프로그래밍을 하고, 곧 바로 컴파일을 할 수 있게 되었다. 또한 한번만 op code를 생성하면 이 코드를 이용해서 여러 번 실행할 수 있기 때문에 매번 프로세스를 생성하던 기존 구조에 비해 매우 효과적이다.

참고문헌

[1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, " Compilers principles, Techniques, and Tools ," Addison-Wesley, Bell Telephone lab. 1986
[2] Allen I. Holub, "Compiler Design in C ,"Prentice-Hall International, Inc. 1990
[3]Jonathan Amsterdam, "A SIMPL Compiler", BYTE, Vol.110,Byte, No8,10,11, 1985
[4] Teradyne Inc. "PASCAL/STEPS Reference Manual ," Boston, 1985
[5] Teradyne Inc. "PASCAL/STEPS & Catalyst Digital ," http://www.teradyne.com
[6] STATEC Inc. "AZ400 Manual ," Korea, 2000
[7] 고훈준, 류진수, 김기태, 유원희, "Test System 용 번역기 설계," '2001춘계 학술발표논문집, 한국정보과학회, 계제 예정, 2001.