C 언어 중심의 테스트 시나리오 기술을 허용하는 테스트벤치 자동화 도구의 개발에 관한 연구

정 성 헌, 장 경 선, *조한진
충남대학교 컴퓨터공학과, *ETRI
전화 : 042-821-7720 / 핸드폰 : 016-427-2032

A study on the generation of test benches from a C-like test scenario description

Sung-heon Jung, Kyoung-son Jhang, Han-jin Joo
Dept. of Computer Engineering, ChungNam National University
E-mail : shjung@ce.cnu.ac.kr

Abstract

It is said that the verification effort occupies about 50-70 percent of the total effort of a System-On-A-Chip. This paper aims to develop a test bench automation tool based on the abstraction of the interface protocols. This tool will allow designers to describe their test benches in a high level language such as C rather than VHDL or Verilog. It helps designers to save their verification time and effort.

I. 서론

VLSI 공정 기술의 발달로 한 칩에 점점 더 많은 기능을 넣을 수 있게 됨에 따라, 그 이전에 PCB 보드상에 있던 많은 칩들은 한 칩에 집적화 시킬 수 있게 되었으며, 이것을 시스템 온 칩(System-On-A-Chip, SOC) 이라고 한다[1]. 이런 SOC 설계 및 구현 과정에서는 이에 대한 검증이 필요하고, 이것은 전체 SOC 구현 노력의 약 70% 정도를 차지한다고 한다[2]. 이러한 검증을 위한 도구 중 하나로서 테스트벤치 자동화 도구가 있다. 테스트벤치 자동화 도구는 테스트를 작성하는 과정을 자동화 해주는 도구라고 할 수 있다. 테스트벤치의 자동화에서는 VHDL 이나 Verilog 로 직접 작성하여 사용할 수 있으나, 이것은 시간과 노력이 많이 든다. 테스트벤치 자동화에서 중요한 것은 테스트벤치 작성을 RTL 수준에서 하는 것보다는 행위수준 또는 트랜잭션 수준이상에서 작성하도록 지원해주는 것이다[4]. 즉, C 나 C++ 와 같은 상위 수준 언어를 테스트 시나리오 기술에 이용함으로써 테스트벤치 작성 수준을 한 단계 올릴 수 있다[4]. 이를 위해서는 BFM (Bus Function Model) 에서 BPC (Bus Protocol Component) 와 같이 트랜잭션을 인터페이스 포트에 대한 사이클 수준 동작으로 변환해주는 IPC (Interface Protocol Component) 를 필요로 한다. 트랜잭션 수준에서 테스트 시나리오를 작성하도록 지원하는 프로그램 중 대표적인 것으로 오픈소스코드인 TestBuilder 가 있고, Cadence 의 TBV (Transaction-Based Verification) 의 개념은 다음 그림 1과 같이 설명될 수 있다. 즉 하드웨어 모듈과 직접 인터페이스 하면서 주어진 트랜잭션을 수행하는 모듈(IPC, Interface Protocol Component)의 작성과 그런 트랜잭션들을 이용하여 알고리즘 형태로 기술하는 테스트벤치 작성을 분리시키는 것이다. IPC 는 BFM 에서 BPC 와 같이 트랜잭션을 인터페이스 포트에 대한 사이클 수준 동작으로 변환해주는 BPC 와 같은 기능을 갖는 것이다. 즉 IPC 는 IP (Intellectual Property) 개발자가 작성하고, 그 IP를 사용하는 사람, 또는 IP 통합자(integrator)는 트랜잭션 수준에서 인터페이스 프로토콜과 같은 하드웨어적인 상세한 내용을 모르고서도 테스트벤치를 개발함으로써, 테스트벤치 개발의 생산성을 높이자는 것이다[4]. 하지만 이런 개념에 C++를 가지고 접근함으로서 테스트벤치 개발자는 방대한 메소드와 클래스 라이브러리에 대한 이해가 선행되어야만 한다.

본 연구는 IPC 에 근거하여 C 언어와 유사한 테스트 시나리오 기술언어 (Test Bench Modeling Language, TBML) 를 사용하여, 트랜잭션 수준에서 테스트 벤치를 자동으로 생성하는 시스템에 관한 것이다.

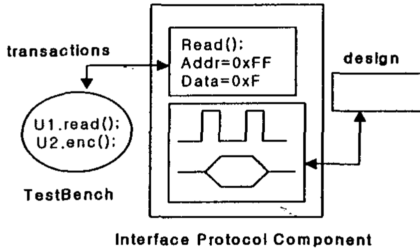

그림 1. 트랜잭션 기반 검증 방법[3]

II. TBML

TBML 은 행위 수준 또는 트랜잭션 수준 이상에서 테스트 벤치를 기술하기 위해 이 연구에서 제안된 언어이다. 대부분의 설계자들이 가장 쉽게 이해 할 수 있는 C 언어에 기반 하였지만, C 언어에서 사용하는 모든 자료형과 구문을 허용하지는 않으며, 하드웨어적인 특성을 나타내기 위해 비트벡트형과 FIFO 타입, 그리고, 특정 싸이클 동안 기다리는 역할을 하는 waitfor() 문을 추가하였다. TBML 은 크게 interface 선언과 behavior 기술, 두 개의 부분으로 구성된다. interface 선언 부에서는 테스트하려는 IP 대한 트랜잭션 정보들을 선언해 주는 부분이다. 테스트벤치 기술자는 실제 트랜잭션이 어떻게 동작하는지 알 필요 없이 이 인터페이스 정보만을 가지고서 시나리오를 기술하면 된다. behavior 기술 부분에서는 실제 트랜잭션들에 대한 순서 관계를 기술한다. 즉, 테스트벤치 기술자가 테스트 하고자 하는 내용을 알고리즘적 흐름으로 기술하면 된다.

그림 2는 DES 와 SRAM 에 대한 TBML 기술 예제이다. 인터페이스 DES 는 encryption 과 decryption 두 개의 트랜잭션을 가지고 그 전달 인자는 64비트의 키, 암호화 또는 복호화 하려는 64비트의 값, 그리고 암호화 또는 복호화 된 값을 저장 또는 비교하기 위한 64비트의 변수 또는 값이 필요하다. 인터페이스 SRAM 은 read 와 write 트랜잭션을 가지며, 전달인자는 16비트의 주소와 8비트의 데이터이다. 시나리오 기술 부분에서는 먼저 각 인터페이스에 대한 인스턴스 d1 과 s1을 선언하였고, 필요한 변수를 선언한 후, 그 변수에 값을 대입하고, DES 의 트랜잭션인 encryption 을 수행하였다. 그 결과 값은 tmp_ctext 에 저장된다. 만약 이 값이 기대한 값과 같으면 waitfor() 문을 수행하여 10 싸이클 동안 기다린 후 DES 의 decrytpion 을 수행하고, 그 결과를 세 번째 인자 값과 비교한다. 처음에encryption 된 결과는 write 트랜잭션을 이용하여 SRAM 에 기록한다.

```
interface DES {
      transaction encryption(in bit[63:0] key,
        in bit[63:0] ptext, out bit[63:0] ctext);
      transaction decryption(in bit[63:0] key,
        in bit[63:0] ctext, out bit[63:0] ptext);
  };
  interface SRAM {
    transaction read(in bit[15:0] addr, outbit[7:0] data);
    transaction write(in bit[15:0] addr, in bit[7:0] data);
  };
  testbench mytestbench(DES d1, SRAM s1)
  {
    bit[63:0] tmp_key;   bit[63:0] tmp_ptext;
    bit[63:0] tmp_ctext;  bit[7:0] tmp_data;
    tmp_key=0x133457799bbcdff1;
    tmp_ptext=0x0123456789abcdef;
    d1.encryption(tmp_key, tmp_ptext, tmp_ctext);

    if(tmp_ctext==0x85e813540f0ab405) {
      waitfor(10);
      d1.decryption(tmp_key, tmp_ctext,
                        0x0123456789abcdef);
    }
    tmp_data[7:0]=tmp_ctext[7:0];
    s1.write(0x0000, tmp_data);
    tmp_data[7:0]=tmp_ctext[15:8];
    s1.write(0x0001, tmp_data);
    ...
  }
```

그림 2 DES 와 SRAM 에 대한 TBML 기술 예제

그림 3은 AMBA 시스템 버스인 AHB 를 위한 TBML 기술이다. AHB 는 버스트 전송이 가능하므로 TBML 에서 FIFO 타입을 이용하여 먼저 값을 저장한 다음, 그것을 전달인자로 사용하여 버스트 트랜잭션인 bwrite() 와 bread() 를 수행한다.

III. TBML의 구현 구조

3.1 TBML의 기본 구조

테스트벤치 기술자에 의해서 기술된 테스트 시나리오는 그림 5에서처럼 ModelSim 의 *FLI* (Foreign Language Interface) 프로시저로 자동 변환된다. 이렇게 변환된 FLI 의 C 모듈은 다수의 IP 를 구동 할 수 있는 VHDL 모듈의 스케쥴러의 일부분이 되어 동작하게 된다. VHDL 모듈의 스케쥴러는 FLI Call 로

```
interface AHB {
  transaction read (in bit[31:0] ADDR,
                           out bit[31:0] Data);
  transaction bread (in FIFO bit[31:0] ADDR[8],
                 out FIFO bit[31:0] Data[8]);
  transaction write (in bit[31:0] ADDR[8],
                           in bit[31:0] Data);
  transaction bwrite (in FIFO bit[31:0] ADDR[8],
                 in FIFO bit[31:0] Data[8]);
};
testbench mytestbench(AHB p1)
{
  FIFO bit[31:0] burst_addr[8];
  FIFO bit[31:0] burst_data[8];
  burst_addr.insert(0x00000000);
  ...
  burst_data.insert(0x00000000);
  ...
  p1.bwrite(burst_addr, burst_data);
  burst_addr.insert(0x00000000);
  ...
  p1.bread(burst_addr, burst_data);
  return;
}
```

그림 3 AHB 에 대한 TBML 기술 예제

FLI 프로시저들을 호출하게 된다. nexttran() 은 IPC 스케쥴러 모듈에서 IP 가 실행되는 중 이 프로시저를 호출하면 다음에 실행 될 트랜잭션을 위한 전달인자를 준비하고, 실행될 트랜잭션 코드 값을 전달한다. 스케쥴러는 이 코드 값에 따라 해당 IPC 의 트랜잭션을 시작시킨다. 그리고 IPC 는 스케쥴러에게 트랜잭션 엔드 신호를 보내어 트랜잭션이 종료되었다는 것을 알려주어서 다음 트랜잭션을 시작할 수 있게 된다. 그림 4는 DES 와 SRAM 에 대한 TBML 에 대해 자동 생성되어진 nexttran() 코드의 일부분을 보여준다. nexttran() 은 프로그램 카운트 값을 유지하며, 호출이 있을 시에 현재 프로그램 카운터의 값에 대응되는 레이블로 분기하여 그 역할을 수행한다. retrieve() 는 nexttran() 에 의해 준비된 트랜잭션의 전달인자를 VHDL 모듈이 트랜잭션을 수행 중 필요한 인자에 대한 정보를 읽어 가려고 할 때 호출되는 프로시저이다. store() 는 VHDL 모듈에서 생성된 트랜잭션 인자 값을 C 모듈로 전달하기 위해 호출하는 프로시저이다. store() 의 역할은 단지 해당 변수에 값을 저장하는 것뿐만 아니라, 만약 저장하려고 하는 위치에 특정 값이 있다면 그 값과 비교한다. 이와 같이 스케쥴러는 실제 IPC 를 통해 TBML 에서 호출하는 트랜잭션들을 수행한다.

3.2 FIFO 구조

AHB 의 TBML 기술 예에서 보듯이 FIFO 타입을 사용하는 경우에는 두 모듈간에 더 많은 인터페이스가 필요하다. remain() 은 FIFO 에 남아 있는 유효한 데이터 값을 검사하며, 비슷한 개념으로 count() 는 몇 개를 데이터를 사용할 수 있는 지를 알려주며, FIFO 의 현재 사용량을 알아보기 위한 empty(), 와 full() 이 있다. 그리고 peek() 는 현재 FIFO 의 가장 앞 부분에 있는 데이터의 내용을 보여준다. FIFO 에 값을 저장하는 것과 읽어 오는 것은 store() 와 retrieve() 가 수행한다.

3.3 로그파일

그림 5에서처럼 FLI Call 이 있을 때는 두 모듈사이에 전달되는 값들을 로그파일로 기록함으로서 IP 의 검증을 쉽게 할 수 있다. nexttran() 이 호출되었을 때에는 다음 실행될 트랜잭션이 무엇인지와, 그 트랜잭션이 어떤 전달인자를 가지고 실행하는지에 대한 정보를 기록해둘 필요가 있으며, retrieve() 호출 시에는 C 모듈에서 VHDL 모듈로 어떤 값이 전달되었는지에 대한 기록이 필요하며, store() 호출 시에는 VHDL 모듈에서 C 모듈로 어떤 값이 전달되었는지와, 만약 저장하려는 전달인자의 위치에 값이 있다면, 그 값과 비교한 결과를 로그파일에 기록한다. 그리고 각 프로시저의 호출 시에는 해당 시뮬레이션 시간이 기록된다.

```
switch(pc) {
 case 0 : goto label0
 case 2 : goto label1
...
}
...
label1:
 interface[0].tran[0].arg[0].type = REGISTER_PT;
 interface[0].tran[0].arg[0].value.preg = &tmp_key;
 ...
 interface[0].tran[0].arg[0].type = REGISTER_PT;
 interface[0].tran[0].arg[0].value.preg = &tmp_ptext;
 ...
 setTranCode(tran_code, 1); //DES 의 encryption
 pc=2;
 return;

label2:
 if(cmpRwithC(&tmp_ctext, "0x0123456789abcdef"))
 {
  wait_arg = 10;
  setTranCode(tran_code , 0); //wait transaction
  pc=3;
  return;
  ...
  ...
```

그림 4 자동 생성된 nexttran() 프로시저의 일부 코드

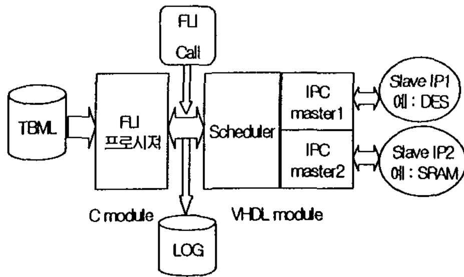

그림 5. 테스트벤치 모듈과 IP 들의 연결 구조

IV. 실험결과

그림 5와 같이 연결된 DES IP 코어와 SRAM IP 코어에 대한 TBML 을 그림 2와 같이 기술하여 테스트해본 결과가 그림 6과 같은 로그 파일이 생성되었다. 먼저 DES 의 encryption 트랜잭션을 120 ns 에서 실행하였고, 그 결과를 tmp_ctext 에 저장하였다. 그리고 10 싸이클 동안 wait 트랜잭션 수행 후, dectypiton 트랜잭션을 수행하고 decrpypion 의 세 번 째 전달인자로 사용된 값과 비교하여 그 결과를 기록하였다. 다음으로 SRAM 의 write 트랜잭션을 수행하여 데이터를 SRAM 에 저장하였다.

그림 7 은 그림 3에서 기술된 AHB 를 테스트하여 생성된 로그파일이다. 버스트 트랜잭션이 시작되었을 때 FIFO 타입을 사용한 전달인자에서 연속적으로 값을 읽거나 쓰는 것을 알 수 있다.

DES 와 SRAM, AHB 이외에도 PVCI, UTOPIA 에 대한 검증을 완료하였다. 표 1은 실험에 사용된 IPC 에 대한 특성을 나타낸다.

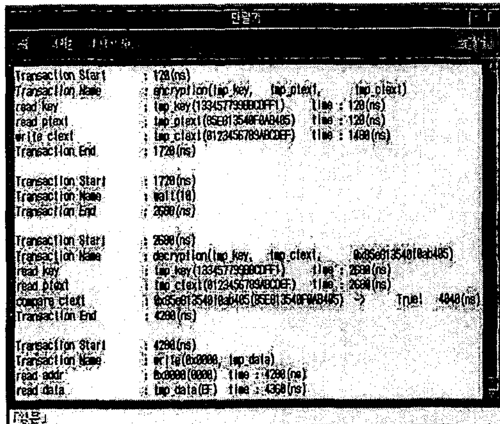

그림 6 DES 와 SRAM 대한 실행 로그파일

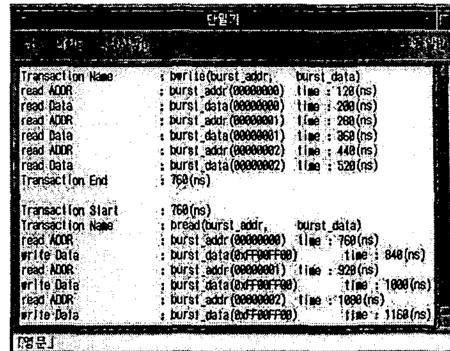

그림 7 AHB 에 대한 실행 로그 파일

IPC 이름	포트수	트랜잭션수
DES	11	2
SRAM	10	2
PVCI	12	4
AHB	14	4
UTOPIA Tx	9	1

표 1. 실험에 사용된 IPC 특성 테이블

V. 결론

이 논문은 SOC 설계 시 많은 시간과 노력을 필요로 하는 검증 시간을 단축시키는 데 많은 도움을 줄 것이라 사료된다. 또한, 본 연구 결과는 BFM 개발 또는 자동생성 등에도 응용 될 수 있으며, 특정 인터페이스 프로토콜 (예: VCI, AHB, APB등) 에 대한 컴플라이언스 체킹 (Compliance Checking) 도구 등에 대한 개발에도 응용 할 수 있다.

현재는 순차적 구조를 갖는 테스트 시나리오만을 처리하며, 병렬적 구조를 갖는 테스트 시나리오의 처리는 추후 구현 예정이다.

참고문헌(또는 Reference)

[1] 장경선외 2인, "온칩버스를 위한 IP 인터페이스 프로토콜 추출기 개발에 관한 연구", ETRI 2000

[2] J. Bergeron, Writing Testbenches, Kluwer Academic Publishers, 2000.

[3] D.S. Brahme, *et. al*, Transaction-Based Verification Methodology, Cadence Berkeley Labs, Technical Report #CDNL-TR-2000-0825, Aug. 2000.

[4] 장경선외 3인, "온칩 버스를 위한 IP 인터페이스 프로토콜 검증 도구의 개발에 관한 연구", ETRI, 2001