인터페이스 프로토콜 기술언어에 근거한 트랜잭션 모니터 모듈의 생성

윤창열[O] 장경선
충남대학교 컴퓨터공학과
{yun[O], ksjhang}@ce.cnu.ac.kr

The Generation of Transaction Monitor Modules based on a Interface Protocol Description

Chang-Ryul Yun[0] Kyung-Sun Jhang
Dept. of Computer Engineering, ChungNam National Univ.

요 약

SoC 설계의 검증 비용이 전체 설계 비용의 70%를 차지한다. 이런 검증을 위한 노력과 시간을 줄이기 위해서는 SoC 설계 검증 수준을 시그널 수준 또는 사이클 수준에서 트랜잭션 수준으로 높여야 할 필요성이 있으며, 또한 그렇게 하는 것이 바람직하다. 이 논문은 IP의 인터페이스를 통해 수행되는 트랜잭션을 사이클 수준에서 기술하는 트랜잭션 중심의 인터페이스 기술 언어로부터 트랜잭션 모니터 모듈을 생성하는 방법을 제시한다. 트랜잭션 모니터모듈은 시뮬레이션을 통한 검증에 파형을 통한 결과의 관찰대신, 트랜잭션 단위의 결과의 관찰을 가능하게 함으로써 검증시간을 줄이는데 공헌할 것으로 기대된다.

1. 서 론[1)]

SoC 설계의 검증 비용이 전체 설계 비용의 많은 부분을 차지한다. 이런 검증을 위한 노력과 시간을 줄이기 위해서는 SoC 설계 검증 수준을 시그널 수준 또는 사이클 수준에서 트랜잭션 수준으로 높여야 할 필요성이 있다. IP의 인터페이스를 통해 수행되는 트랜잭션을 사이클 수준에서 기술하는 트랜잭션 중심의 인터페이스 기술 언어를 사용한다. 이 언어는 타이밍 다이아그램을 대신 할 수 있을 뿐만 아니라 모니터 모듈 생성기, 테스트벤치 생성기 또는 IP 래퍼 합성기의 입력으로도 사용될 수 있다. 이런 인터페이스 기술로부터 트랜잭션 모니터 모듈을 자동 생성하여 올바르게 동작함을 검증하였다.

SoC 설계에서 설계 검증이 임계경로에 있음을 알 수 있다 [1]. 검증시간을 줄이는 방법은 잘 정의된 인터페이스를 통해 설계와 검증 작업을 동시에 수행하는 것이다. 또 다른 방법은 검증수준을 하위수준의 0과 1을 다루는 수준에서 더 높은 수준 - 트랜잭션 또는 버스 사이클 수준으로 추상화하는 방법이다 [1]. 사이클 수준 또는 하위수준으로 트랜잭션 자체를 기술하는 것과, 트랜잭션 수준으로 테스트 벤치를 개발하는 것을 분리함으로써, 많은 새롭고 복잡한 테스트벤치를 빠르게 개발 할 수 있게 될 것이다. 상세한 프로토콜에 관해서는 트랜잭션 기술 [2] 에 포함되어 있다. 이런 방식을 통해 테스트 벤치 코드의 재사용성을 높일 수 있다.

IP 또는 코어 기반 설계 [5] 에서, 보통 IP는 읽기, 쓰기 등과 같은 고정 인터페이스 프로토콜을 갖기 때문에, IP의 인터페이스 프로토콜은 타이밍 다이아그램과 같은 방식으로 기술될 수 있다. 그러나 타이밍 다이어그램 형태로 인터페이스를 기술하는 것은 설계자에게는 유용하지만 컴퓨터로 처리하기에는 어렵다. 타이밍 다이어그램을 대신하면서, 모니터모듈 생성기 등의 입력으로도 사용될 수 있는 형태, 즉 컴퓨터로 처리할 수 있는 형태로 인터페이스를 기술하는 것이 필요하다.

본 논문에서는 사이클 수준(cycle-accurate level)에서 IP의 포트들을 통해 수행되는 트랜잭션들을 기술하는 인터페이스 기술 언어(IPL)를 사용하였다. PIG[6]의 기술 방법을 확장하여 다중 트랜잭션, 내부 레지스터, 고정 또는 가변 버스트 전송의 기술, 조합회로 출력포트(combinational output port)등을 허용한다.

IPL로부터 VHDL로 기술된 트랜잭션 모니터 모듈을 생성하는 방법을 기술한다. 해당 신호들에 연결된 모니터 모듈은, 연결된 신호를 통해 수행되는 트랜잭션을 모니터하고, 그 결과를 파일에 기록하고, 오류가 발생하면 그 정보를 기록하게 된다. 이런 모니터 모듈을 이용하여, 인터페이스 프로토콜의 오류 위치를 알아내는 시간을 줄이고, 시뮬레이션 결과를 트랜잭션 수준에서 알 수 있으므로, 시뮬레이션 결과(타이밍 다이아그램 등)를 분석하는 시간을 줄일 수 있다.

2. 인터페이스 프로토콜 기술 언어(IPL)

논리 합성 툴들이 일반화되고, 많은 코어가 RTL수준으로 기술되고 있기 때문에, IPL은 RTL 수준의 인터페이스 프로토콜 언어로 기술하였다. IPL은 테스트 벤치 생성기, 트랜잭션 모니터 생성기, IP 래퍼 합성기 등과 같은 인터페이스 관련 설계자동화 툴의 입력으로 사용될 수 있도록 구성되었다.

자동으로 테스트 벤치를 생성하기 위해서 IPL은 트랜잭션 수준에서 C/C++ 등 프로그래밍 언어와의 인터페이스가 필요하다 [3][4]. 그러기 위해서는 데이터 유형의 정의가 가능해야 한다. IP는 여러 개의 트랜잭션이 있고, 각 트랜잭션은 트랜잭션 수행시 전달되는 인수를 갖는다. 많은 IP나 온칩 버스 프로

1) "이 연구는 BK21충남대학교 정보통신인력양성사업단의 지원을 받았음."

토콜에서 종종 사용되는 버스트 전송을 기술이 가능하도록 설계되었다. IPL에서는 3상태(tri-state), 양방향, dont care조건, 그리고 포트 값의 유효시간 등 포트와 관련된 특징을 기술 할 수 있다. IPL에서 설계자는 리셋(reset)과 클럭에 대해 매개변수(parameter)를 이용하여 표현한다. IPL은 단일 클럭 도메인(single clock domain)으로 가정한다. 다음 그림 1은 IPL로 SDRAM 제어기가 프로세서와 연결되는 쪽의 인터페이스를 기술한 것이다. 먼저 데이터의 형이 정의되고, 트랜잭션의 전달 인수도 정의된다. IP의 포트에 대한 기술과 텀의 기술이 오고, 텀들의 조합으로 트랜잭션을 기술한다. 마지막에 IP에 제공하는 트랜잭션을 기술한다.

```
type basic bit[31:0];  /* 자료형 정의 */
type Bsize bit[2:0];  type twobit bit[1:0];
type SADD bit[19:0];
type mode {    /* 트랜잭션의 전달인수 정의 */
   Bsize N; /*Burst length*/ Bsize CAS; /* latency */
}
protocol SDRAMctr {
  master bit data_addr_n;
  master bit we_rn;    /* IP의 포트 */
  master bit rst;
  maslave basic AD;
  parameter clock clkp double; /* 클럭 정보 기술 */
  parameter reset rst asynchronous positive; /* 리셋 신호 */
  register Bsize n(7), cas(3);/* 내부 레지스터 정의와 초기화*/
  type block {
          SADD Addr;      /* Address */
          basic Data[0:n];   /* DATA */
  }                          /* 텀 정의 */
  term int_0() { 0, 1, 0, ["--10----------"] > }
  term int_1(Bsize a, twobit b) {1, 1, 0,
       ["--00----------"&a&"-"&b&"-"] >  }
  term Write_0(SADD a) { 0, 1, 0, ["--00------"&a&"--"]>  }
  term Write_1(basic b){ 1, 1, 0, b>}
  term Read_0(SADD c){ 0, 0, 0, ["--00------"&c&"--"]>  }
  term Read_1(){ 1, 0, 0, -}
  term Read_2(basic d){ 1, 0, 0, d<}
  term ready() { 1, 0, 0, -}
       /* 트랜잭션 정의 */
  transaction initiate(mode mdset)
    : n <= mdset.N, cas <= mdset.CAS; /* 전달인수 */
  { ready()*, int_0(), int_1(mdset.n, mdset.cas) }
  transaction Read(block Imp) /*  */
  { ready()*, Read_0(Imp.Addr),Read_1()^(9+cas),
          Read_2(Imp.Data[0])...Read_2(Imp.Data[n-1]) }
  transaction Write(block Imp)
  { ready()*, Write_0(Imp.Addr),
         Write_1(Imp.Data[0])...Write_1(Imp.Data[n]) }
    /* IP시 제공하는 트랜잭션 */
  SDRAMctr = initiate | Read | Write; /*IP 제공 트랜잭션*/
}
```

그림1 IPL로 기술한 SDRAM 제어기의 인터페이스 프로토콜

조합회로의 출력을 갖는 포트(combinational output ports)의 동작을 표현하기 위해 텀 선언부에 마스터 포트와 슬래이브 포트간의 부분순서관계(partial order relationship)을 기술하도록 추가하였다. 예를 들어 다음 기술은 슬래이브 포트 done이 조합회로의 출력을 갖는 포트이고, 마스터 포트 start의 값이 0이 될 때, done 신호는 1이 됨을 나타낸다. 마스터 포트 start는 순차적(sequential) 또는 레지스터드(registered) 출력이고, 마스터는 해당 클럭 사이클의 시작에서부터 값을 구동해야 한다.

```
master bit start;
slave bit done combinational;
term a() { 1, 0 } { start -> done }
term b() { 0, 1 } { start -> done }
```

3. 트랜잭션 모니터모듈의 생성

트랜잭션 모니터 모듈은 두개의 연결된 IP 사이의 신호들을 통해 실행되는 트랜잭션을 기록하는 역할을 한다. 모니터 모듈은 오류가 발생한다면 오류를 기록한다. 모니터 모듈은 인터페이스 프로토콜 오류의 위치를 찾는 시간을 줄일 수 있고, 트랜잭션 수준에서 시뮬레이션 결과를 보여줌으로써, 타이밍 다이아그램의 파형을 보는 것과 같은 시뮬레이션 결과에서 트랜잭션을 찾는 시간을 줄여준다. 각 트랜잭션에 대한 기록은 트랜잭션의 시작과 종료시간, 실행된 트랜잭션의 이름, 전달된 데이터 또는 주소, 오류와 그 원인 등이 파일에 저장된다.

추상적 FSM의 생성은 derivative construction [14, 15] 알고리즘을 이용했다. IPL은 마스터쪽의 기술도 아니고 슬래이브쪽의 기술도 아니다. IPL은 마스터를 위한 FSM, 슬래이브를 위한 FSM, 모니터 모듈을 위한 FSM으로 변환 될 수 있기 때문에, 생성된 FSM은 추상적(abstract)으로 불린다. 생성흐름의 다음 단계는 추가된 반복 연산자 ^n(n회 반복), #n(n회 이하반복가능)를 처리하는 후처리 단계이다. 마지막으로 IP들간의 특정 신호들에 연결되어서, 연결된 신호들을 통해 수행되는 트랜잭션을 모니터 할 있는 모듈을 만드는 코드생성 단계이다.

3.1 Abstract FSM 생성

인터페이스의 각 트랜잭션은 내부적으로 정규식 형태의 트리와 추가된 데이터구조를 통해 표현된다. 다른 트랜잭션에 속한 정규식의 트리를 이용하여 하나의 추상적인 FSM을 생성한다. 이 생성된 FSM의 상태는 다음에 오는 텀의 집합이 되고, 전이의 조건은 텀이 된다. FSM을 생성하는 알고리즘은 다음과 같이 요약된다.

```
1. 처음 오는 텀으로 초기상태 S0을 구성한다.
2. S0의 각 텀 t에 대해서
3. A. 텀 t 다음에 오는 텀의 집합 S1 생성
   B. 만약 집합 S1이 존재하지 않는다면, 상태 S1을 만들고, S0로 부터 S1으로 전이 t를 만든다.
   C. 만약 S1이 존재한다면, S0로 부터 S1으로 전이 t를 만든다.
더 이상 새로운 상태가 만들어지지 않을 때까지, 단계 2를 반복하여 새로운 상태를 추가한다.
```

3.2 후처리

추상적 FSM 생성과정에서 직접 반복연산자를 다루는 것이 쉽지 않다. 따라서 추상적 FSM 생성단계에서 반복연산자 #는 *(0회 이상 반복)연산자와 동일하게 취급하고, ^와 ·연산자는 +(1회 이상 반복)연산자와 동일하게 취급하였다. 그리고 후처리 단계를 통해 각 연산자와 관련 있는 전이에 특별한 표시를 한

다. 예로, k, a, b가 텀인 정규식 k, a^3, b의 정규식은 k의 인식한 후에 세번의 a텀이 오고, b텀이 옴을 의미한다. 그래서 k가 인식되었을 때 카운터의 초기화가 필요하다. 그리고 매번 a가 인식되었을 때, 카운터를 증가시켜야 하고, b가 인식되고 카운터가 3과 같을 때, 반복 동작 a^3가 올바르게 마치는 것이다. 이 예에서 알 수 있듯이, entry, increment, exit의 세 타입의 표시가 필요하다. 따라서 텀 k는 entry 전이가 되고, a는 increment 전이, b는 exit 전이가 된다. 각 전이의 조건은 다음과 같다.

entry 전이 : $S \rightarrow^{k} S_t(\hat{}) \cup \{k\}$, where $k \in S_p(\hat{})$

increment 전이 : $S_1 \rightarrow^{k} S_2$, where $k \in S_t(\hat{})$

exit 전이 : $S_1 \rightarrow^{k} S_2$, where $k \in S_n(\hat{})$

*$S_p(\hat{})$: 노드 ^의 바로 이전에 올 수 있는 노드의 집합

*$S_t(\hat{})$: 노드 ^의 하위 트리에서 맨 처음 올 수 있는 노드의 집합

*$S_n(\hat{})$: 노드 ^의 바로 다음에 올 수 있는 노드의 집합

3.3 VHDL 코드생성

모니터 모듈은 하나의 프로세스 문으로 구성된다. IP의 모든 신호가 모니터 모듈의 입력이 되고, 모니터 모듈은 입력받는 신호의 값을 파악하여 내부 FSM의 상태를 결정하고, FSM의 현재 상태와 입력되는 값에 의해 전이를 하면서 트랜잭션을 모니터링 하게 된다. 이때 전달되는 데이터, 시간, 트랜잭션 이름, 트랜잭션의 동작, 오류의 기록과 원인에 대한 정보가 파일로 저장된다. 내부 FSM은 IPL에 기술된 텀과, 텀의 조합으로 기술된 트랜잭션을 근거로 만들어진다.

4. 실험 결과

SDRAM 컨트롤러 인터페이스가 그림 2과 같은 환경에서 시험되었다. 모티터링 결과의 일부분이 그림 3에 나타나 있다. write 트랜잭션이 150사이클에서 시작하고 158사이클에서 종료됨을 알 수 있다. 그림 4는 150 사이클(1200ns)에서 시작되고 158(1264ns)에서 끝나는 Write 트랜잭션의 타이밍 다이아그램을 보인다. 생성된 로그 파일과 시뮬레이션 결과와 같음을 알 수 있다. 트랜잭션 수준에서 IP동작을 분석함으로써 IP의 검증 비용을 줄일 수 있다.

5. 요약과 향후 과제

본 논문은 트랜잭션 중심으로 인터페이스를 기술한 언어를 사용하여 트랜잭션 모니터 모듈의 생성에 방법에 대해 기술했다. 트랜잭션 모니터링의 예로 SDRAM 컨트롤러, PCI Target, DES 코어에서, 생성된 모니터 모듈이 연관된 인터페이스와 연결되어서 올바르게 동작함을 살펴보았다. 트랜잭션 모니터 모듈은 시뮬레이션을 통한 시스템 검증시에 트랜잭션 단위로 시뮬레이션 결과를 관찰할 수 있게 하여, 검증시간의 단축에 기여할 것으로 기대된다.

사용된 언어는 마스터로 동작하는 테스트 벤치 모듈의 일부인 트랜잭션을 구동하는 모듈을 생성하는데 사용될 수 있다. 이 언어는 IP의 인터페이스를 기술한 언어이기 때문에, IP 래퍼 합성의 입력으로도 사용될 수 있다. 앞으로 이런 자동 생성툴의 개발을 계속할 계획이다.

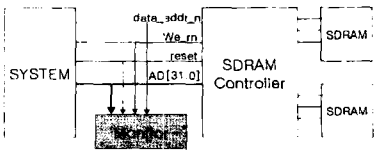

그림 2. SDRAM 제어기의 모니터모듈 테스트 환경

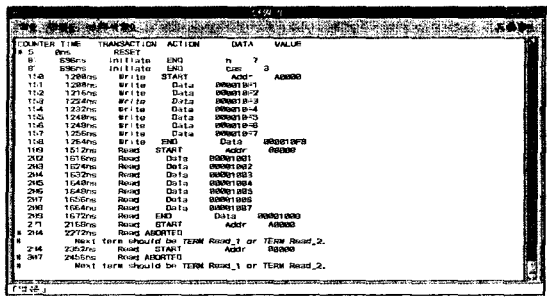

그림 3. SDRAM 제어기의 모니터링 결과

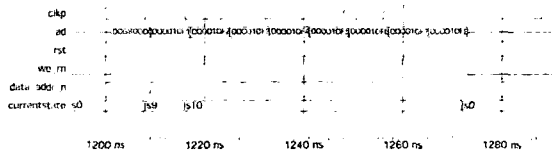

그림 4. SDRAM 제어기의 타이밍 다이아그램

참고문헌

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

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

[3] A. Gerstlauer, S. Zhao, D. D. Gajski, VHDL+/SpecC Comparisons A Case Study, Technical Report ICS-98-23, May 19, 1998, Dept. of Information and Computer Science, Univ. of California, Irvine.

[4] Funtional Specification for System C 2.0, version 2.0-M, Jan. 2001, available on the web site www.systemc.org .

[5] J. Smith and G. De Micheli, "Automated composition of hardware components," Proc. Of DAC 98, 1998.

[6] R. Passerone, J. A. Rowson, A. Sangiovanni-Vincentelli, "Automatic synthesis of interfaces between incompatible protocols," in Proc. of DAC '98, 1998 pp.8-13.

[7] J. A. Brzozowski, "Derivatives of regular expressions," Journal of the Association for Computing Machinery, vol. 11, pp. 481-494, Oct. 1964.