SystemC로 구현된 하드웨어 명세의 정형 검증*

김민숙[0], 안영정, 방기석, 최진영

고려대학교 컴퓨터학과

{mskim[0], yjahn, kbang, choi} @formal.korea.ac.kr

Formal Verification of Hardware Implemented in SystemC

Min-Suk Kim[0], Young-Jung Ahn , Ki-Seok Bang, Jin-Young Choi

Dept. of Computer Science and Engineering, Korea University

요 약

내장형 시스템의 개발에 있어서 자원의 효율적인 활용과 정확한 설계를 위해 SystemC를 이용한 통합설계 방식이 많이 사용되고 있다. 하지만 시스템이 점점 복잡해 지면서 단순한 언어차원에서의 개발 뿐 아니라 개발 이전에 시스템의 정확성을 검증해야 할 필요성이 대두되었다. 이를 위해 정형기법 및 테스팅과 같은 방법을 사용하게 되었다. 본 논문에서는 SystemC로부터 정형기법 도구인 VIS의 입력 언어인 BLIF-MV로 자동 변환하는 알고리즘을 제시하고, SystemC 코드의 자동 검증 방법을 제시하고 실제 시스템에 적용해 보았다.

1. 서 론

기존의 내장형 시스템[1]의 설계 과정을 살펴보면 모델링 단계 후 하드웨어와 소프트웨어 분할을 거쳐 각각 개별적인 설계를 했다. 소프트웨어 부분은 모델링 단계의 언어를 그대로 사용하였고 하드웨어 부분은 하드웨어 명세 언어를 사용하여 재 모델링 단계를 거친 후 합성단계를 통해 하드웨어 칩을 생산하여 내장형 시스템을 구축하였다. 하지만 설계 과정에서 설계자간의 아무런 의사소통이 없이 설계되는 방법은 각각의 설계를 마쳤을 때 서로 맞지 않아 재설계를 해야 하는 등의 시간과 비용의 낭비를 가져오고 시스템의 안전성과 신뢰성을 극도로 떨어뜨렸다. 이러한 단점을 보완하기 위해 모델링 단계시 명세 언어인 C++에 하드웨어 특성을 추가하여 하드웨어 또한 명세할 수 있는 내장형 시스템 통합 설계 언어인 SystemC[2]가 탄생하게 되었다. SystemC는 상위단계에서 기술된 고성능의 복잡한 시스템을 하나의 언어로 하드웨어와 소프트웨어 부분을 모두 명세하는 내장형 시스템 설계 언어이다. SystemC를 이용하면 고성능의 복잡한 시스템을 하드웨어와 소프트웨어 부분을 적절히 혼합하여 최적의 성능을 가지도록 상위 단계에서 설계할 수 있어 시스템의 성능 향상 및 설계 비용의 최소화 면에서 많은 기여를 하고 있다. SystemC와 같은 내장형 시스템 통합설계 언어의 탄생으로 전체 시스템을 하나의 칩 안에 구축하는 SoC (System On a Chip)에 활발한 연구 동기를 부여하는 계기가 되었다. 그리고 하드웨어 설계 단계에서 기존의 하드웨어 명세 언어로 설계된 부분이 SystemC으로 대체 되고 있다. 하지만 내장형 시스템이 복잡해지면서 무작위 시뮬레이션 기법이나 수작업을 통하여 테스트 케이스를 선정하고 테스팅[2] 하는 작업은 점점 더 어려워지고 있다. 이러한 이유로 복잡한 시스템을 검증할 수 있는 보다 효율적이며 신뢰할 만한 방법론의 필요성이 제기되었다. 모델 체킹은 비록 적용분야는 유한 시스템으로 제한되지만, 도구를 이용하여 손쉽고 빠르게 검증할 수 있으며 완전히 자동화 될 수 있다는 장점이 있다. 모델 체킹은 상태 전이 시스템과 특성이 주어지면, 모델 체킹 알고리즘을 통해 주어진 시스템이 검증하고자 하는 특성을 만족하는지를 알아보기 위해서 전체 상태 공간을 검사하고 그 결과를 알려준다. 본 논문에서는 SystemC의 하드웨어 명세 부분을 모델 체킹하여 정형검증 하는 방법에 대해 논한다. 좀 더 자세히 말하자면, SystemC로 하드웨어를 명세한 부분을 VIS[3]라는 모델 체킹 도구의 입력 언어인 BLIF-MV[4]로 변환하여 검증을 수행하는 방법이다. 이 방법의 이점은 SystemC 을 기반으로 명세된 모델에 대해 검증하는 기법을 제공하여 요구한 시스템을 설계할 수 있도록 도와 줄 수 있으며, C 프로그래밍 언어로 구현된 프로그램에 대한 정형검증 기법으로 확장을 기대할 수 있다.

2장에서는 하드웨어, 소프트웨어, 시스템 정형검증과 연관된 관련 연구를, 3장에서는 통합 설계를 위한 방법론인 SystemC에 대해 논하고, 4장에서는 본 논문에서 제안하고자 하는 SystemC 프로그램의 정형검증 방법론에 대해 논하고, 5장에서는 SC2MV컴파일러를 통해 SystemC를 입력으로 받아들여 모델 체커의 입력 언어인 BLIF-MV 생성시키는 방법에 대해 설명한다. 6장에서는 적용 사례를 들고 마지막으로 결론 및 향후 연구 방향에 대해 논한 후 본 논문을 마치겠다.

2. 관련 연구

컴퓨터의 활용분야가 많아지면서 하드웨어, 소프트웨어, 내장형 시스템과 같은 시스템의 안정성이 매우 중요해 지고 있다. 소프트웨어가 요구사항을 만족하는지 관해서 많은 연구가 활발히 전개 되고 있다. 하드웨어 분야는 다양한 기법들이 개발되어 상업용으로 적용이 되고 있다. 하드웨어 정형검증의 대표적인 도구는 VIS 와 SMV[5]가 있다. VIS를 이용해서 Cache Coherence Protocol, PCI local BUS, MPEG System Decoder등을 검증하였고, SMV를 이용해서 IEEE Futurebus+ Standard프로토콜을 검증함으로써, 프로토콜 설계 시 발견하지 못했던 에러들을 발견하였다. 한편, 내장형 시스템의 통합 설계 과정에도 정형검증 기법을 도입하고자 하는 시도가 이루어지고 있다. heterogeneous system인 내장형 시스템을 통합적으로 모델링하고 디자인하고 시뮬레이션 하기 위한 PtolemyII[6][7]의 개발이 이루어지고 있으며 프랑스의 INRIA에서는 ESTEREL toolset[8]을 개발하고 있다.

3. SystemC

SystemC는 C++ 라이브러리를 기반으로 소프트웨어 알고리즘, 하드웨어 아키텍쳐, 시스템 레벨 디자인, SoC를 효과적으로 생성하는데 사용할 수 있는 언어이다. C, C++은 효과적인 시스템 기술, 제어와 컴팩트한 시스템을 개발하는데 필요한 데이터 추상을 제공해주기 때문에 소프트웨어 알고리즘을 위한 선택이고 인터페이스 사양이다. 대부분의 설계자들은 이러한 언어들에 익숙해져 있고 많은 수의 개발 장비들이 이와 연관되어 있다. SystemC 클래스 라이브러리는 표준 C++에서는 찾아볼 수 없는 하드웨어 타이밍,

* 본 연구는 한국과학재단 목적기초연구 (R01-2000-00287) 지원으로 수행되었음

병렬 동작, 반응 동작(reaction behavior) 등을 포함하는 시스템 아키텍처를 모델링하기 위한 필요한 구조를 제공한다. 이러한 구조를 C에 포함시키는 것은 언어에 대한 독점적인 확장을 요구한다. SystemC는 C++에 필요한 클래스들을 제공하고 설계자들에게 친숙한 C++언어와 개발 장비를 계속 사용할 수 있도록 해 만약 독자들이 C++프로그래밍 언어에 친숙해 있다면 추가적인 문법을 배우지 않고도 SystemC 클래스에서 소개되는 추가적인 의미를 이해함으로써 SystemC로 프로그램하는 방법을 배울 수 있다.

4. SystemC 프로그램의 정형 검증방법론

현재까지 SystemC로 명세된 시스템의 오류를 찾아 내기 위해서 주로 사용되는 방식은 테스팅이었다. 그 동안 많이 사용되어져 왔던 시뮬레이션과 테스팅은 개념적으로 간단하다는 장점은 있지만, 검사한 입력 값에 대해서만 시스템의 완전성을 보장하며, 대형 시스템의 경우 검사해야 할 입력 값이 테스팅이 불가능할 정도로 많다는 단점 때문에 불완전성 문제를 내포하고 있다. 이와는 달리 정형검증은 쉽게 찾아내기 힘든 불일치성, 애매모호함, 불완전성을 검증함으로써 보다 완벽한 시스템을 구축할 수 있게 한다. 특히, 자동적으로 검증할 수 있는 검증 도구의 개발로 인해 학계에서 뿐만 아니라 산업계에서도 최근 들어 상당한 관심을 받아 오고 있다. 이에 본 논문에서는 C/C++을 이용해 Functional Specification을 작성하고 이를 SystemC의 모듈 구조를 사용하여 소프트웨어와 하드웨어 부분으로 나누어 그 중 하드웨어 부분을 표현한 SystemC 프로그램을 BLIF-MV 코드로 변환하여 VIS를 이용하여 모델 체킹하는 방법을 제안하고자 한다.

VIS는 모델 체킹 도구로써, 검증하고자 하는 시스템을 BLIF-MV형식의 유한 상태 기계로 기술하고 유한 상태 기계에 의해 도달 가능한 상태들이 검증하고자 하는 속성을 만족하는지를 알고리즘을 이용해 밝혀낼 수 있다. 모델 체킹의 가장 큰 특징이자 장점은 테스팅과는 다르게 자동화할 수 있다는 것이다. 모델이 특성을 만족하는지에 대한 검증은 컴퓨터가 해 준다. 만약 모델이 특성을 만족하지 않는 에러 상태가 발생하면, 어떠한 상황에서 에러가 일어나는지를 반례로 보여준다. SystemC 프로그램으로부터 VIS의 입력 언어인 BLIF-MV를 생성시켜, 검증하고자 하는 하드웨어 시스템을 입력하고 검증하고자 하는 속성을 CTL(Computation Tree Logic)을 이용하여 입력하여, 검증 결과를 알아낼 수 있다. 본 논문에서는 SystemC 프로그램에서 BLIF-MV를 생성시키는 변환기를 sc2mv라 이름 지었다. sc2mv는 기존의 하드웨어 명세 언어인 Verilog[9]에서 BLIF-MV를 생성시키는 vl2mv를 참고로 하여 구현한다.

5. sc2mv의 구현

SystemC 프로그램에서 BLIF-MV를 생성해내는 변환기를 sc2mv라 이름 붙였다. sc2mv의 변환 규칙은 기존의 Verilog에서 BLIF-MV를 생성해내는 vl2mv와 유사하다.

5.1 Net variables & Register variables

변수의 선언 및 사용 시에는 [그림 1]과 같이 Verilog에서 사용하는 포트 연결 규칙을 따른다. 넷은 하드웨어 요소 사이에 연결을 나타낸다. SystemC에서 넷은 "sc_in<type> 이름" 또는 "sc_inout<type> 이름"을 통해 정의된다.

레지스터는 데이터를 저장할 수 있다. 레지스터는 다른 논리값이 들어오기 전까지는 그 값을 유지할 수 있다. SystemC에서 레지스터는 "sc_out<type> 이름" 또는 일반 데이터 선언을 통해 정의 된다. SystemC(우측)에 따른 BLIF-MV(좌측)의 변환 규칙을 나타낸다. 변수가 latch로 선언되었을 때 변수이름_ps는 현재 상태를 변수이름_ns는 다음 상태를 나타낸다.

1) net variable

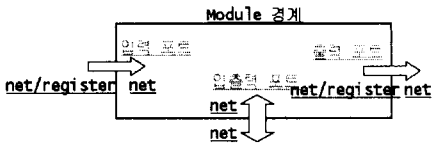
sc_in<bool> x,y; sc_out a; a = x; a = y;	.names x y a - - = x - - = y

2) register variable

sc_out<sc_bit> a;
if (c1) a = e1
if (c2) a = e2
	.names c1 e1 c2 e2 a_ps a_ns
	1 - - - -= e1
	- 1 - - -= e2
	0 0 - - - =a_ps

5.2 Statements

Concatenation 은 결합 연산자로 여러 개의 피연산자들을 한데 묶는 메커니즘을 제공한다. 피연산자들은 크기가 항상 정해져 있어야 하며, (,)을 통해 표현된다. 특정 조건에 따라서 문장을 수행할지



[그림 1] 포트 연결 규칙

말지를 결정하는 조건문은 C-언어에서와 같이 if-else 문을 사용하여 표현된다. 분기 조건이 3 개 이상인 경우 if-else-if 문을 다루기가 불편하게 된다. 이 경우 case 문을 사용해서 간단하게 같은 결과를 얻을 수 있다. Statements 의 SystemC 에 따른 BLIF-MV 의 변환 규칙을 나타낸다.

1) concatenation

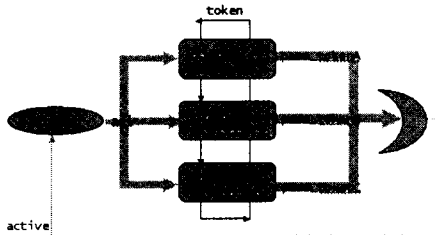
sc_in<bool> b; sc_in<bool> c, d; sc_in<sc_int<4>> a; a = ('2', b, c&d);
.names t<0> 0 .names t<1> 0 .names b t<2> - =b .names c<0> d<0> t<3> .def 0 1 1 1 .names t<0> a<0> - =t<0> .names t<1> a<1> - =t<1> .names t<2> a<2> - =t<2> .names t<3> a<3> - =t<3> .names t<4> a<4> - =t<4>

2) if statement

If (c1) a= e1 Else a = e2;
.names c1 e1 e2 a_ps a_ns 1 - - - =e1 0 - - - =e2

6. 적용 사례

본 논문에서는 SystemC 프로그램의 정형검증하는 방법을 위해 sc2mv를 구현하였다. 이를 실제 예제인 arbiter에 적용시킨 예를 보이겠다. [그림 2]은 A, B, C 3개의 클라이언트를 가지고 있는 arbiter를 표현한다. 요청이 들어온 클라이언트는 arbiter에 의해 토큰이 주어진 경우에만 ack를 발생할 수 있도록 arbiter에서 중재를 한다. arbiter는 A, B, C 클라이언트 중에서 req를 처리하여 pass_token이 주어진 경우에, A, B, C순서로 token을 부여 한다.



[그림 2] Arbiter 소개

[표 1]은 Arbiter 예제의 SystemC 프로그램의 일부이다. 1개의 arbiter와 3개의 client, controller를 호출하는 최상위 main모듈이다.

```
SC_MODULE (main){
SC_CTOR(main){
        arbiter *arbiter = new arbiter;
```

```
        arbiter(clk, active, sel);
        controller *controllerA = new controller;
        client *clientA = new client;
        clientC(clk, ackC, reqC);
    }
}
```

[표 1] Arbiter의 SystemC프로그램의 일부

위와 같은 프로그램을 sc2mv를 통해 변환 과정을 거치면 [표 2]와 같은 BLIF-MV코드가 생성된다.

```
.model main
.outputs ackA
.outputs ackB
.outputs ackC
.mv sel 4 A B C X
.subckt arbiter arbiter active=active  sel=sel
.subckt client clientC ack=ackC  req=reqC
...
.model client
...
.end
```

[표 2] Arbiter의 변환된 blif-mv코드

위의 Arbiter가 반드시 만족해야 할 특성으로 다음과 같이 크게 2가지로 나눠 CTL문법으로 나타낼 수 있다.

1) Safety 조건

Safety조건이란 어떠한 이벤트가 절대 발생하지 않기를 바란다는 특성으로 Arbiter예제에서는 동시에 두 개의 client가 요청을 처리했음을 알리는 ack신호가 발생하지 않기를 바라는 것으로, 즉, a와 b가 동시에, 또는 b와 c, c와 a가 동시에 ack를 발생하는 일이 절대 없기를 바라는 safety 조건을 다음과 같이 줄 수 있다.

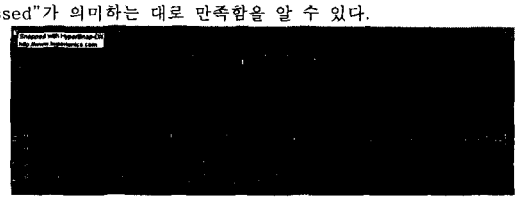
AG (!(ackA = 1*ackB = 1+ ackB = 1*ackC = 1+ ackC = 1*ackA=1))

2) Liveness 조건

Liveness조건이란 임의의 상황에서 어떤 조건이 언젠가는 반드시 일어나길 바란다는 특성이다. Arbiter예제에서는 client가 req를 요청했다면, 앞으로 언젠가는 반드시 ack가 발생하길 바란다는 liveness조건을 다음과 같이 A, B, C 세 클라이언트에 줄 수 있다.

AG((reqA = 1) -> AF(ackA = 1))
AG((reqB = 1) -> AF(ackB = 1))
AG((reqC = 1) -> AF(ackC = 1))

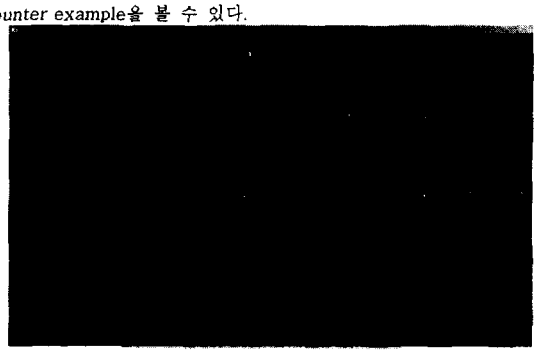
위의 Arbiter가 만족하길 바라는 특성을 arbiter.ctl 파일에 작성한다. 첫번째로 VIS를 구동 시켜 sc2mv를 통해 변환된 arbiter.mv 파일을 읽게 후 arbiter.ctl에 표현된 특성을 VIS를 통해 모델 체킹을 한 결과는 [그림 3]와 같이 4가지 모든 특성은 "formula passed"가 의미하는 대로 만족함을 알 수 있다.



[그림 3] 특성을 만족한 결과 화면

본 논문에서는 SystemC프로그램에 조작을 가하면 어떠한 결과를 얻게 되는지 실험해 보았다. arbiter가 active신호의 유무에 상관없이 arbiter모듈이 활성화 될 때마다 A, B, C순서로 토큰을 넘겨주도록 원래 arbiter의 System프로그램에 수정을 가했다.

전과 동일하게 수정이 가해진 SystemC프로그램을 arbiter_bug라 이름을 붙이고 sc2mv를 통해 변환을 하여 arbiter_bug.mv파일을 얻는다. VIS를 구동 시켜 arbiter_bug.mv파일을 읽게 한 후 전과 동일한 특성을 표현하고 있는 arbiter.ctl을 모델 체킹하면 [그림 4]와 같은 결과를 얻게 된다. 4가지 특성 중에서 첫번째로 표현된 safety 특성은 "formula passed"라는 결과를 통해 만족함을 볼 수 있지만 3가지 liveness특성은 "formula failed"라는 결과를 통해 만족하지 못함을 알 수 있다. 그리고 어떠한 경우에 만족하지 못하는지 counter example을 볼 수 있다.



[그림 4] 특성을 만족하지 못한 결과 화면

7. 결론 및 향후 연구 방향

반도체 설계 및 생산 공정의 급속한 발달로 마이크로 프로세서를 전자기기 내에 탑재하는 것이 훨씬 쉬워지고, 비용이나 제품 출시 기간에 있어 경쟁력을 가지게 되면서 내장형 시스템을 얼마나 효과적으로 구축하느냐가 필요한 제품을 설계하는데 큰 비중을 차지하게 되었다. 이에 본 논문에서는 내장형 시스템을 설계하기 위해 C프로그래밍 언어의 변형으로 개발된 SystemC 프로그래밍 언어로 만들어진 프로그램을 정형검증하는 방법을 제시하고 있다. SystemC 프로그램은 C와 유사하기 때문에 사용자가 익히기 쉬운 장점을 갖고 있다. 시스템 기술 언어인 SystemC 프로그램을 sc2mv 변화기를 통해 VIS라는 모델 체킹 도구의 입력 언어인 BLIF-MV로 변환하여 검증을 수행하는 방법을 통해 다음과 같은 장점을 기대할 수 있다. 첫째, 시스템 수준 명세언어를 이용하여 시스템을 기술할 경우 복잡한 시스템의 세부 사항들을 나타내지 않고 구현과 독립적인 기술을 할 수 있어 최상위 수준에서 조기에 시스템의 결함을 찾아 낼 수 있다는 장점과 둘째, 모델 체킹과 같은 정형검증을 이용함으로써 명세에 대하여 요구 조건이 만족하는지의 결과를 알아 낼 수 있다. 또한 향후 C 프로그램 언어로 구현된 프로그램에 대한 정형검증 기법으로 확장이 가능하다.

본 연구를 통해 보인 SystemC 프로그램 언어의 정형기법 방법에서는 SystemC의 일부분에 대해서는 제안을 하고 있다. 향후 연구 과제로는 첫째, SystemC 프로그램의 표현력을 좀 더 넓힐 수 있는 방법을 생각해야 하며, 하드웨어 부분으로 제한하여 모델 체킹을 수행하는 방법을 SPIN이나 Verisoft와 같은 툴과 연동하여 소프트웨어 부분까지 확대하여 정형검증하는 방법을 접목시키고자 한다.

참고 문헌

[1] W.H.Wolf, *"Hardware-software co-design of embedded systems"*, Proc.IEEE, vol. 82., pp.967~989, July 1994

[2] SystemC.org. *SystemC Version2.0 User's Guide*, SystemC.org, 2002

[3] Tiziano Villa, Gitanjali Swamy, Thomas Shiple, *VIS User's Manual*, University of California, Berkeley, 1996

[4] Yuji Kukimoto, *BLIF-MV*, The VIS Group, University California, Berkely, May 1996

[5] Kenneth L.McMillan, *Symbolic Model Checking*, Kluwer Acadecmic Publisher, 1993

[6] http://www-cad.eecs.berkeley.edu/~polis./

[7] Edward A. Lee, *"Overview of the Ptolemy Project," Technical Memorandum UCB/ERL M01/11*, University of California, Berkeley, March 6, 2001

[8] .Berry, *The Esterel v5 Language Primer Version v5_91*, INRIA,June 5, 2000

[9] Palnitkar Samir, *Verilog HDL*, Prentice Hall, 1996