# 내장형 시스템의 통합 설계를 위한 검증 및 구현

안영정[0], 김진현, 최진영
고려대학교 컴퓨터학과
{yjahn, jhkim, choi}@formal.korea.ac.kr

# Verification and Implementation for Co-Design of Embedded System

Young Jung Ahn[0], Jin Hyun Kim, Jin Young Choi
Dept. of Computer Science and Engineering, KOREA University

**요 약**

내장형 시스템은 산업전반에 다양한 방법으로 응용되고 있다. 하지만 항공 분야나 원자력 분야의 내장형 시스템은 안정성과 신뢰성이 절대적으로 보장되어야 하는 시스템으로 그 설계부터 구현에 이르기까지 다양한 방법으로 검사되고 검증되어야 한다. 본 논문에서는 내장형 시스템의 통합설계를 위한 기반으로 하드웨어 설계 언어인 Verilog를 입력언어로 받아들여 이를 정형검증 도구인 VIS를 통하여 검증한 다음 이를 바로 구현하는 방법론 및 예를 보이고자 한다. 이러한 방법을 통해 내장형 시스템의 안정성과 신뢰성의 수준을 향상시키고자 한다.

## 1. 서론

내장형 시스템은 시스템의 개발 및 유지 보수의 효용성 및 그리고 확장에 대한 융통성 때문에 그 가치가 날로 증가하고 있다. 하지만 내장형 시스템이 규모, 기능면에서 점점 대형화되고 복잡해지고, 특히 원자력 분야나 의료 분야, 항공 관제의 내장형 시스템과 같이 safety-critical 시스템은 그 안정성과 신뢰성을 보장하기 위해 설계 초기 단계부터 검사 및 검증이 이루어져야 한다. 하지만 기존의 설계에 대한 검사나 구현은 Test-case에 의존하기 때문에 그 안정성이나 신뢰성을 보장할 수 없다. 근래에는 이러한 검사 방법과는 달리 시스템이 가질 수 있는 모든 상태를 분석하는 정형 기법[1]을 통한 내장형 시스템의 통합설계가 다양하게 연구되고 있다.

본 논문에서는 안정성과 신뢰성을 보장하는 내장형 소프트웨어를 만들기 위해 하드웨어를 소프트웨어로 구현할 수 있는 기반 기술을 제안하고자 한다. 하드웨어 설계를 소프트웨어로 구현하기 위한 기술로서 본 논문에서는 하드웨어 설계 언어인 Verilog[2]를 검증하는 기법과 그 검증된 코드를 내장형 소프트웨어로 만들기 위한 중간 코드 및 그 코드의 구현을 제안하고자 한다.

2장에서는 내장형 시스템의 통합 설계를 위한 HDL 언어의 내장형 소프트웨어의 설계 및 구현 과정을 설명하고 3장에서는 설계의 검증을 위한 모델 체킹과 그 도구에 대해 논하고자 한다. 4장에서는 그렇게 검증된 시스템의 구현에 대해 논하고 5장에서는 결론 및 향후 연구 방향에 대해 논한다.

## 2. HDL 언어의 내장형 소프트웨어의 설계 및 구현 과정

내장형 시스템을 설계하기 위해서 정형 명세에 기반한 하드웨어-소프트웨어 통합설계, CoWare N2C를 이용한 통합설계, SoC를 위한 하드웨어 기반의 통합 검증, 그리고 SystemC를 통한 소프트웨어 기반의 통합 설계등 많은 연구가 활발히 이루어져 가고 있다. 본 논문은 하드웨어-소프트웨어 통합설계를 할 때 Verilog를 통하여 모델링하고 VIS를 통해 검증한다. 이 검증된 모델을 하드웨어 부분과 소프트웨어 부분으로 나누어서 하드웨어 부분은 그대로 합성 tool을 이용하여 내장형 하드웨어 시스템으로, 소프트웨어 부분은 C-code로 변환 함으로써 내장형 소프트웨어 시스템으로 전환한다. 다시 말해서 최대한의 안정성과 신뢰성, 구현 및 유지보수의 편리함을 도모하기 위해 Verilog를 통한 하드웨어 기반의 하드웨어-소프트웨어 통합 설계에 정형 기법을 적용하여 시행하고자 한다.

## 3. 내장형 시스템의 설계에 대한 검증

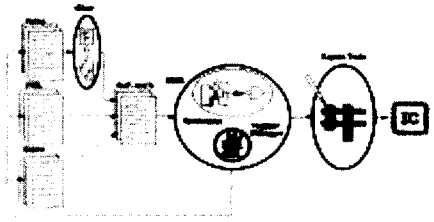
### 3.1 모델체킹

모델체킹[3]은 상태 탐색을 기반으로 하는 정형 기법으로써 상태 전이 시스템과 특성이 주어지면, 모델체킹 알고리즘은 주어진 시스템이 검증하고자 하는 특성을 만족하는지를 알아보기 위해여 전체 상태 공간을 검사한다. 특히, 심볼릭 모델 체킹은 OBDDs를 이용함으로써, 엄청난 수($10^{120}$이상)의 상태를 가지는 시스템도 증명할수 있게 되었다.

### 3.2 모델체커 - VIS

VIS (Verification Interacting with Synthesis)[4]는 검증, Simulation, 유한 상태 하드웨어 시스템의 Synthesis를 종합한 도구이다. 이 도구는 입력 언어로 Verilog를 사용하고 fair CTL 모델체킹, Language Emptiness 검사, 조합 순차 동등성 검사, cycle-based simulation과 hierarchical synthesis를 지원한다. VIS는 BLIF-MV[5]라는 중간 형태에서 작동하며 BLIF-MV 파일은 VL2MV라 불리는 컴파일러에 의해 생성된다. BLIF-MV 묘사가 VIS로 읽혀질 때 계층적 트리 형태로 저장된다.[6] VIS는 [그림 1]과 같이 Verilog, VHDL, ESTEREL을 받아 들여 중간 코드인 BLIF-MV로 바꾸어 HSIS로 synthesis 및 debug를 한다. 물론 Verilog는 VL2MV라는 compiler를 이용하여 BLIF-MV로 바꾼다. HSIS를 이용한 결과를 layout tool을 이용하여 집

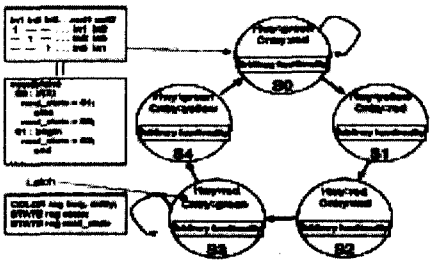
적회로를 만든다. [그림 1]에서는 검증을 통한 직접회로 설계에 대한 과정을 보여주고 있다.



[그림 1] 검증을 통한 직접회로 합성과정

### 3.3 BLIF-MV

Verilog로 설계된 시스템이 검증을 위해 거치게 될 중간 포맷은 BLIF-MV(Berkeley Logic Interchange Foremat - Multi-Valued Variable)이다. BLIF_MV는 비결정적으로 계층적 순차 시스템을 묘사하기위해 디자인된 언어이다. BLIF-MV 파일은 VL2MV라 불리는 컴파일러에 의해 생성된다. 아래 [그림 2] 같이 BLIF-MV은 오토마타로 묘사된다.
이렇게 표현된 모델은 VIS로 입력되어 모델체킹을 통해 시스템의 다양한 특성을 검증하게 된다.



[그림 2] 신호등 시스템의 BLIF-MV로의 변환

## 4. 검증된 코드의 구현

### 4.1 BLIF-MV의 C코드의 변환

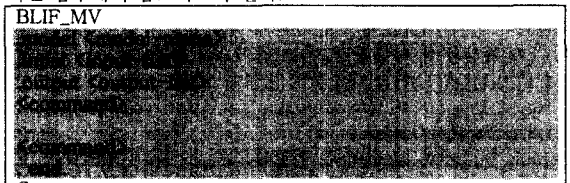
본 논문에서 제시하고 있는 구현 방법은 앞에서 언급한 것처럼 오토마타로 표현된 시스템을 C로 구현함으로 행위적인 측면에서의 안정성을 도모하고자 한다. 즉, 시스템이 오토마타로 표현되어 있기 때문에 다양한 행위를 보다 평이하고 단순하게 묘사할 수 있고 이를 바탕으로 내장형 시스템에 내장되는 소프트웨어의 안정성을 꾀하고자 하는 것이다. 따라서 본 논문에서는 Verilog로 명세된 시스템을 BLIF-MV 형태의 오토마타로 바꾸고 이를 C 코드로 만들어 시스템의 행위를 묘사하고자 하는 것이다. BLIF_MV는 다음과 같은 형태로 범용 언어인 C로 변환한다. 우선 C-code 변환에 앞서서 생각해 볼 문제로 Verilog에서는 존재하는데 C에 존재하지 않는 concurrency, time, non-determinism, 그리고 boolean 형의 자료형에 관한 문제이다. Concurrency에 관한 문제는 Verilog에서 BLIF_MV로 변환할 때 오토마타로 표현되면서 자동적으로 해결되고 time은 오토마타의 arbitrary functionality를 나타내는 Catesian product로 표기한 테이블내에 표현되어 변환되므로써 해결된다. non-determinism은 검증하기 위한 testbench(stimulus module)에서만 사용하므로 생각하지 않기로 한다. 마지막 고려사항으로 boolean형의 자료형이다. 하드웨어의 입력, 출력은 ' 0' 과 ' 1' 의 값만을 가지는 boolean형식으로 되어있는데, ANSI C에서 boolean이라는 자료형에 관해서는 선언이 되어 있지 않으므로 아래와 같이 비트필드를 사용하여 새로운 자료형을 만들어 사용한다. 본 논문에서는 BLIF_MV가 오토마타를 표현하는데 있어 필요한 구문을 5가지로 나누고 5가지 구문별로 변환 문법은 다음과 같다.

```
struct BOOLEAN {
  unsigned bit : 1;
};
typedef BOOLEAN boolean;
```

#### 4.1.1 models

model은 계층적 시스템을 정의할 때 사용되는 시스템이다. model의 키워드는 " .model" 과 " .end" 를 가지고 선언하고 model명은 아래와 같이 선언된다. C로의 표현은 함수로 표현되며 <model-name>이 어떤 종류의 함수인가를 컴파일러에게 알려주는 프로토타입이 된다. BLIF_MV가 하나 이상의 model 정의로 구성되어 있다면 한 model은 " .model" 키워드 밑에 .root 키워드로 루트 모델로 명시되는데 Verilog에서 main으로 선언된 module이 루트 model로 명시되고 main module이 없다면 첫 번째 model이 루트 model이 된다. 이 루트 모델은 설계 블록이 완성되었을 때 테스트나 검증을 하기 위해서 설계 블록의 입력 인수에 대해 값을 넣고 출력 인수를 모니터링하여 결과를 보기위한 testbench나 stimulus module이다. 본 논문에서는 실행 코드로 변환하는 것이 목적이므로 main model의 변환은 " .model main" 문을 만나면 다음 " .end" 문까지 스킵하는 방식으로 배제하기로 한다. 아래의 <input-list>와 <output-list>은 정형 입력단의 문자열들이다. 공백으로 구분된다. " .input" 과 " .output" 이란 키워드로 선언되며 아무것도 명시가 안될수도 있다. C로의 표현은 함수 외부에서 extern으로 선언하여 함수의 다른 함수에서 참조하도록 한다.

BLIF_MV



C

#### 4.1.2 Multi-valued Variables

Multi-valued Variable은 하나 이상의 한정된 수의 값들을 가진 변수이다. Multi-valued Variable의 키워드는 " .mv " 를 가지고 선언하고 구성은 다음과 같다. variable-name-list는 선언된 값들의 이름으로 구성되고 콤마로 구분된다. number-of-values는 값들의 수를 명시한 자연수이다. value-list는 기호적 변수로 선언된 변수에만 적용되는데 값들의 리스트들로 구성된다. 공백으로 구분한다. C로의 표현은 enum 키워드를 사용하여 열거형(enumerated type)으로 나타낸다. 열거형 자료 선언은 " enum 열거형명 { 열거상수 리스트 } 변수 리스트" 로 선언한다. value-list는 열거 상수 리스트로 열거형명은 value-list들간에 ' _' 를 붙여 선언한다. variable-name-list는 변수 리스트로 변환한다.

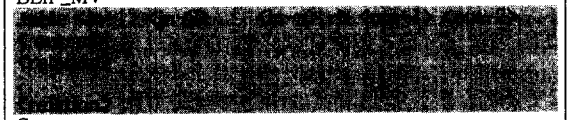
BLIF_MV

C

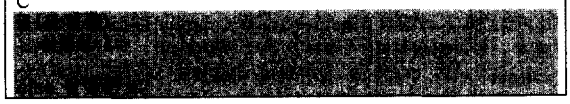
#### 4.1.3 Tables

table은 물리적 게이트의 추상적 표현이다. table은 입력에 의해 유도되고 물리적 게이트의 기능으로 출력이 생성된다. 비록 물리적 게이트는 어떠한 입력이 들어오던간에 결정적으로 출력이 생성된다. Table은 키워드 " .table" 로 선언하고 다음과 같다. in-1, . . .,in-n는 입력들의 이름으로 주어지는 문자열들이다. out-1, . . .,out-n는 출력들의 이름으로 주어지는 문자열들이다. table은 적어도 하나의 출력을 가져야한다. relation은 입력과 출력의 관계를 나타내고 변수들이 가지는 값들은 Catesian product로 표기된다. 예를 들어 x가 1, 2, 3, 4의 값을 가지는 변수라면 ' - ' 는 모든 값, {1-3}은 (1, 2, 3), !{2-3}은 (1, 4)을 나타낸다. C로의 표현은 if else 문으로 대체한다. in-1, . . .,in-n과 relation은 조건문의 조건식으로 변환한다. 이때 생성되는 조건들은 키워드 " &&" 로 연결한다. 그리고 out-1, . . .,out-n과 relation은 위의 조건이 성립되면 실행하는 실행문으로 변환한다. Relation이 여러 줄일때는 키워드 " if else" 로 선언하여 변환한다.

BLIF_MV



C



##### 4.1.3.1 Construct

키워드 ' =' 은 table에서 선언되고 출력변수의 값이 입력변수의 값과 똑같을 때 사용한다. 이는 위의 조건문을 생성할 때 실행문으로 변환 된다. 이는 대입문의 형식이 된다.

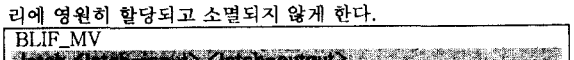
##### 4.1.3.2 Default Output

Default Output은 .default란 선언문으로 명시되고 출력값의 일반적인 패턴을 나타낼 때 명시한다. 이는 위의 조건문에서의 키워드 " else" 로 변환한다.

#### 4.1.4 Latchs and Reset Tables

latch는 매 클럭마다 update되는 값이 저장되는 저장 요소이다. 다음 클럭이 발생하기전까지 현재 값을 계속 유지한다. 다음과 같이 선언된다. reset은 latch의 초기화값을 명시한다. 이는 오토마타의 state의 상태를 나타내는 변수므로 항상 존재하여 하므로 C로 변환할 때 키워드 " extern" 을 이용해서 모든 함수에서 참조가 가능한 광역 변수로 선언하여 프로그램의 종료시까지 메모리에 영원히 할당되고 소멸되지 않게 한다.

BLIF_MV



C

#### 4.1.5 Subcircuits

model이 다른 모델을 참조할 때 아래와 같이 키워드 " .subckt" 로 선언된다. 현재 model에서 instance-name 대신해 model-name로서 참조한다. formal-actual-list은 formal변수와 actual변수로 구성되고 formal변수의 값을 actual변수로 참조한다. C로의 변환은 model-name을 사용하여 procedure

BLIF_MV

C

call을 한다. formal-list로 parameta passing한다.

아래의 C-code는 위의 문법에 의하여 BLIF_MV가 변환된 신호등 제어기이다.

```
#include <stdio.h>
#include <stdlib.h>
struct BOOLEAN {
        unsigned bit : 1;
};
typedef BOOLEAN, boolean;
enum RED_YELLOW_GREEN {RED, YELLOW, GREEN};
extern boolean X;
extern boolean clear;
. . .
void sig_control(void)
{
enum S0_S1_S2_S3_S4 {S0, S1, S2, S3, S4};
. . .
cntry_raw_n9 = RED;
. . .
if (_nd == state) _nc = 1;
else _nc = 0;
. . .
if (_nb == 1) _n91 = 1;
else if ((_nb == 0)&&(_n10 == 1)) _n91 = 1;
. . .
else _n91 = 1;
. . .
if (_n9a  == 1) _n9b = state_raw_na;
else _n9b = S0;
}
```

### 5. 결론 및 향후 연구 과제

본 논문은 안정성이 검증된 설계를 내장형 소프트웨어로 제작하는데 그 목적을 두고 있다. 이 과정에서 사람의 손이 개입하지 않고 직접 검증된 설계를 직접 구현함으로 구현 과정에서 생길 수 있는 에러를 최소화하고 또한 구현된 소프트웨어 역시 행위적인 측면에서 보다 나은 안정을 꾀하고자 한다.

내장형 시스템을 만드는데 있어서 하드웨어 언어인 Verilog를 이를 정형 검증한 후, 이를 하드웨어 부분과 소프트웨어 부분으로 나누어 구현할 때, 소프트웨어 부분은 지금까지 사람이 직접 짜야 했지만 본 논문에서는 검증된 코드를 직접 C로 구현할 수 있는 방법을 제시함으로 보다 안정된 실행 코드를 얻고자 한다.

향후 연구 과제로는 이렇게 만들어진 내장형 소프트웨어 코드의 적용과 이 코드의 최적화가 필요할 것이다. 즉 오토마타를 구현한 것이므로 그 코드의 양이 매우 크다. 따라서 이에 대한 최적화 작업이 필요할 것으로 여겨진다.

### 참고 문헌


[1] E. M. Clarke and J. M. Wing, " Formal Methods: State of Art and Future Directions" , ACM Computing Surveys, 28(4), pp.626-643, December 1996

[2] Bob ZeidMan, *Verilog Designer' s Library*, prentice-Hall,Inc.,Upper Saddle River, New jersey, 1999

[3] Kenneth L. McMillan, Symbolic Model Cheking, Kluwer Acadecmic Publisher, 1993

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

[5] Yuji Kukimoto, " BLIF-MV" ,The VIS Group,University of Califonia, Berkely, May 31, 1996

[6] Szu-Tsung Cheng,Robert K. Brayton, " Compiling Verilog into Automata" , University of California, Berkeley, CA 94720, May 18, 1994