논리 회로 시뮬레이션을 위한 효율적인 C 코드 발생

한기영[0], 표창우
홍익대학교 컴퓨터공학과
kyhan@cs.hongik.ac.kr, pyo@wow.hongik.ac.kr

Efficient C Code Generation for Logic Circuit Simulation

Ki-young Han, Chang-woo Pyo
Dept. of Computer Engineering, Hongik University

요 약

논리 회로는 하드웨어로 직접 구현할 수도 있으나, 설계된 디지털 논리를 프로그램으로 구현하고 이를 내장형 프로세서가 실행하게 하는 방식으로 구현할 수 있다. 이러한 구현은 내장형 시스템에 적합하도록 코드 크기와 실행 속도가 효율적이어야 한다. 본 논문에서는 복사 전파, 데드코드 삭제, 위상 정렬 등의 컴파일러 최적화 기술과 함수 단위 모듈 개념을 도입하여 신호 흐름 분석 기법의 C 코드 자동 발생기를 구현하였다. 회로 시뮬레이션 구현 기법 중 유한 상태 천이 모델 기법으로 코드를 발생하는 Esterel 시스템과 성능 비교 실험을 한 결과, 코드 크기는 평균 84.17%로 감소되었으며 실행 속도는 평균 4.68배로 향상되었다.

1. 서 론

전력 시스템의 보호를 위한 과전류 차단 제어 회로는 논리 회로 그 자체를 구현할 수도 있으나, 설계된 논리 회로가 수행하는 디지털 논리를 프로그램으로 구현하고 이를 내장형 프로세서가 실행하게 하는 방식으로 구현할 수도 있다. 논리 회로의 소프트웨어적 구현은 회로의 제어 논리를 바꾸고자 할 때에 프로그램만 수정하면 되는 점, 회로의 하드웨어 설계·구현·검증 과정에 필요한 많은 비용과 시간을 절약할 수 있는 점 등의 장점이 있다. 내장형 프로세서 기반의 시스템이 안정화되고 값이 낮아질수록 이러한 개발 방법은 더욱 확산될 것이다.

본 논문은 BLD 문법 코드로부터 원시 C 프로그램을 생성하는 코드 발생기를 신호 흐름 분석 기법으로 구현하고, 상태 천이 모델 기법으로 코드를 발생하는 시뮬레이션 도구 중에서 Esterel 시스템과의 성능을 비교 분석한다.

생성 코드의 목적 기계가 내장형 시스템이므로, 코드 크기와 실행 속도의 효율성이 구현의 핵심이 되어야 한다. 이러한 측면에서 코드 발생기는 최적화 기능을 포함하고 있어야 하며, 하드웨어 모듈을 적절히 추상화하는 함수 단위 코드 발생이 가능한 것이어야 한다. 이와 같은 목표가 달성되면 코드 재사용에 의해 코드 크기가 줄어들고 가독성이 높아진다. 코드 크기는 시스템의 메모리 요구량과 직결되어 있어 전체 시스템 비용과 연관되어 있으며, 코드의 높은 가독성은 프로그램 디버깅 작업에 긍정적 영향을 줄 것이다. 함수 단위의 코드 발생은 서로 다른 응용 사이의 재사용을 높이는 효과도 갖고 있다. 또한 새로운 회로를 설계하고 이에 대응하는 프로그램을 발생하고자 할 때, 이미 제작해 놓은 함수를 라이브러리로 구축하여 사용하면 설계시간을 단축하고 과정 자체를 단순화하는 효과를 거둘 수 있다. 결과적으로 경제적인 제어 회로 설계가 가능하게 되며, 발생된 코드는 시스템 정확성 검증을 위한 시뮬레이션에 직간접적으로 사용될 수 있다.

2장에서는 논리 회로를 시뮬레이션하는 관련 연구들을 소개하고, 3장에서 본 논문에서 구현된 C 코드 자동 발생기를 설명하며, 4장에서 C 코드 자동 발생기와 Esterel의 성능 비교 실험 결과를 분석한 다음, 마지막으로 5장에서 결론과 향후 연구방향에 대해 언급하고자 한다.

2. 관련 연구

논리 회로를 시뮬레이션하는 연구는 회로를 하드웨어로 제작하기 전에 설계를 검증하려는 목적에서 이루어져 왔으며, 대부분의 연구는 회로 기술을 위한 언어와 이를 시뮬레이션하는 시스템을 제공하였다.

회로의 설계를 위한 언어로는 VHDL과 Verilog가 대표적이며 이 언어로 설계한 회로를 검증하려면 특정 시뮬레이터를 통해야만 한다[1]. 설계와 시뮬레이션, 분석을 모두 할 수 있는 도구로는 Simulink가 있다[2]. 이 도구는 GUI를 제공하여 시각적인 설계가 가능하며 시뮬레이션과 분석 도구를 각각 제공하지만, 설계를 검증하는 과정까지만 제공하고 설계에 대응하는 응용 소프트웨어를 제공하지는 않는다. 응용 소프트웨어를 생성하는 시스템인 Terse는 Terse라는 언어를 제공하여 회로를 설계하고 이를 컴파일하여 기계어 코드를 생성한다[3]. 이는 최적화 단계를 포함하기는 하나 목적코드가 기계어 코드이므로 컴포넌트별 재사용은 불가능한 단점이 있다. 회로 설계를 C 언어로 하는 시스템으로는 ALHARD가 있다[4]. 이 시스템은 C 언어를 확장하여 만든 언어인 ALHARD로 회로 를 구성한 후 이를 C 언어로 바꾼다. 그러나 이를 시뮬레이션 하기 위해서는 ALHARD에서 제공하는 시뮬레이터가 필요하다.

Esterel은 본 논문의 C 코드 자동 발생기와 유사한 기능을 가진 시스템이다[5]. Esterel 언어로 회로를 구성하여 이를 Ansi-C 코드로 생성하여 응용 소프트웨어로 사용할 수 있으며, Esterel에서 제공하는 시뮬레이터인 xes를 이용하여 시뮬레이션이 가능하다.

3. C 코드 자동 발생기

C 코드 자동 발생기는 논리 회로의 컴포넌트 단위로 코드 생성 과정을 진행하며, 각 컴포넌트의 동작에 대응하는 코드를 함수로 생성한다. 컴포넌트 단위 코드 발생은 회로 전체를 하나의 함수로 발생하는 경우에 비해 코드 재사용이 증가하여 발생되는 코드 크기가 적어지며 발생 코드의 가독성이 높아져 디버깅과 유지보

수 과정에 도움이 된다. 생성 코드에서 컴포넌트는 비트-벡터 또는 문자열 단위의 변수로 나타나게 된다. 비트-벡터의 경우, 컴포넌트를 비트 단위의 변수로 처리하므로 사용되는 메모리를 최소화할 수 있지만 변수 접근을 위해 많은 기계어 명령을 필요로 하게 되므로 결과적으로 생성된 프로그램의 실행 속도가 느려진다. 문자형 단위로 컴포넌트를 나타내게 되면 변수 접근이 빨라지므로 프로그램의 실행 속도는 향상되지만 변수의 단위가 바이트이므로 약간의 메모리 낭비가 생긴다. C 코드 자동 발생기에서는 생성된 코드의 컴파일 옵션을 통해 두 방법 중 한 가지를 선택할 수 있도록 하였다. 변수의 스코프 규칙으로 지역 변수를 사용하되 버퍼 노드, 타이머 노드의 경우에만 전역 변수를 사용하였다.

논리 회로로부터 C 프로그램을 발생하려면, 논리 회로가 나타내는 컴포넌트 사이의 연결 관계와 신호 흐름 전후 관계가 파악되어야 한다. 하드웨어 상의 연결 관계 파악은 방향성 비순환 그래프(directed acyclic graph : DAG)에 대응하며 위상정렬(topological sort) 기법을 사용하여 신호 흐름을 파악할 수 있다. 회로의 전후 관계를 반영하는 제어 흐름에서 발생하는 복사 전파, 데드코드 삭제를 통하여 프로그램 최적화가 가능하다.

3.1 BLD 문법

논리 회로가 포함하는 개체 정보와 개체들의 연결 관계, 함수로 구현되는 컴포넌트 정보를 나타내기 위하여 BLD 문법을 정의하였다. BLD 문법은 회로에서 사용된 개체와 이 개체들이 갖는 특성에 대한 정보, 그리고 개체들간의 연결관계를 컴포넌트 단위로 나타내게 된다. 이 문법은 미국의 Berkeley 대학에서 만든 문법인 BLIF 문법에서 몇 가지 요소를 빼고 약간 변형시킨 문법으로써, 정규 표현으로 나타내면 <그림 1>과 같다.

3.2 코드 발생 단계

컴파일러는 원시 코드 형태의 소스 코드를 입력으로, 목적 기계에 종속적인 기계어 코드를 생성해내는 프로그램이다. C 코드 자동 발생기는 일반적인 컴파일러 구성 단계를 기반으로 제작되었다. C 코드 자동 발생기의 시스템 구성은 <그림 2>와 같다.

```
architecture   → compDecList
compDecList    → compDec compDecList | ε
compDec        → COMPONENT compName
                 input
                 output
                 nodeDec
                 edgeDec
                 ENDCOMPONENT
compName       → NAME
input          → IN ':' idList
output         → OUT ':' idList
nodeDec        → NODE nodeList
nodeList       → node nodeList | ε
node           → PORT nodeId nodeName
                 | BUFFER nodeId nodeName initVal
                 | CONNECTOR nodeId nodeName
                 | COMP nodeId nodeName compName
                 | TIMER nodeId nodeName length timerKind reset
nodeId         → id
nodeName       → NAME
tag            → TAG | ε
initVal        → INT
length         → INT
timerKind      → ON | OFF
reset          → RESET | ε
edgeDec        → EDGE edgeList
edgeList       → edge edgeList | ε
edge           → '(' source target ')'
source         → INT
target         → INT
idList         → id idList | ε
id             → INT

INT : 정수
NAME : 알파벳(대소문자)으로 시작하는 알파벳(대소문자), 정수, '_'의 조합
```

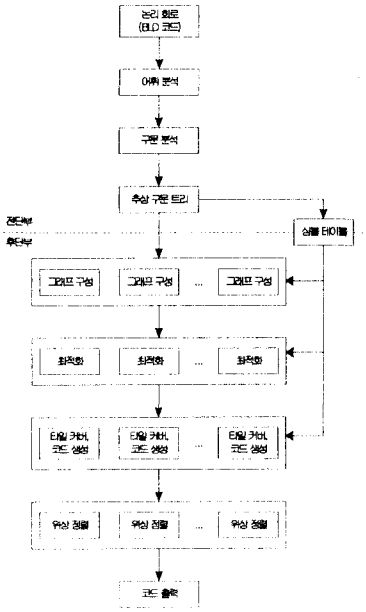
<그림 1> BLD 문법

코드 발생 단계는 크게 전단부(front-end)와 후단부(back-end)로 나뉘어진다. 코드 발생기는 우선 BLD 코드를 입력으로 어휘 분석, 구문 분석 과정을 통해 어휘와 구문 형태가 문법 구조에 적법한지 검사한 다음 추상 구문 트리 (Abstract Syntax Tree : AST)를 구성하여 회로의 정보를 데이터로 저장한다. 추상 구문 트리는 컴포넌트 리스트로 구성되고 컴포넌트는 노드 리스트와 에지 리스트로 구성된다. 심볼 테이블은 각 노드의 정보를 쉽고 빠르게 접근하게 한다. 심볼 테이블의 구조는 노드 번호를 심볼 이름으로 하고 노드 이름 등 각 노드의 정보를 내용으로 구성하며, C 코드로의 번역에 필요한 단계를 모두 수행한 후에는 삭제된다. 추상 구문 트리로부터 컴포넌트 단위로 회로를 나타내는 그래프를 작성하는데, 그래프는 노드에 에지 정보를 첨가하여 완성한다. 노드들의 연결 관계를 그래프로 구성하여 데이터의 흐름을 알 수 있다. 최적화 과정은 복사 전파, 데드 코드 제거의 과정을 통해 포트와 포트를 단순 연결하는 불필요한 값 복사를 제거한다. 타일은 그래프에서 일련의 C 코드로 단순 번역될 수 있는 서브그래프를 말한다. 타일로 빈틈없이 그래프를 덮었을 때 그래프의 커버가 완성되었다고 한다. 각 타일에 대해서는 대응되는 단순한 C 코드를 발생하여 연결시켜 놓는다. 커버가 완성된 그래프의 각 타일에는 C 코드 조각이 붙어 있는데, 이 그래프를 타일 단위로 위상 정렬한 뒤 파일로 출력하면 회로의 데이터 흐름과 일치하는 C 프로그램을 생성하게 된다.

C 코드 자동 발생기와 같이 사용되는 GUI 환경과 생성 코드 검증 도구에 대해서는 이미 연구된 바가 있다.[6].

4. Esterel을 사용한 검증 및 성능 실험

C 코드 자동 발생기의 검증 및 성능 실험의 비교 대상으로 Esterel (Ver. 5.92) 시스템을 선택하였다. Esterel은 동기적인 컨트롤에 의해 제어되는 소프트웨어나 하드웨어의 동시 시스템 시뮬레이션이 가능하며, 오토마타 기반의 유한 상태 천이 모델 기법을 사용한다. Esterel 언어로 기술된 논리 회로는 Ansi-C 코드 발생을 통해 응용 소프트웨어로 변환될 수 있다.



<그림 2> 코드 발생 단계

4.1 bld2strl

Esterel 프로그램으로 C 코드 자동 발생기에 입력되는 논리 회로와 동일한 의미의 회로를 구성하였다면, 변환된 코드는 C 코드 자동 발생기가 생성한 코드의 실행 결과와 일치하는 실행 결과를 나타내야 한다. 동일한 의미의 회로 입력을 보장하기 위해 BLD 파일을 Esterel 파일로 변환해 주는 프로그램 bld2strl을 개발하였다.

bld2strl 프로그램은 BLD 형태를 입력으로 받아 Esterel 문법 형태의 목적 프로그램을 생성해 주는 코드 발생기이다. 입력 형태가 C 코드 자동 발생기와 동일하므로, 코드 발생 단계의 전단부는 C 코드 자동 발생기의 전단부와 동일하다. 후단부에서는 코드 생성이 이루어진다. C 코드 자동 발생기에서 컴포넌트에 대응되는 함수가 생성되는 것처럼, bld2strl은 컴포넌트에 대응되는 모듈을 생성한다. 생성되는 모듈로 컴포넌트 간의 내포 관계를 나타내게 된다. 컴포넌트의 입출력 포트는 각각 모듈의 입출력 시그널로 선언되며, 내부 포트는 지역 시그널로 선언된다.

4.2 검증 및 성능 실험 결과

C 코드 자동 발생기가 발생하는 코드와 Esterel 프로그램이 발생하는 코드에 대해 Sun Ultra-1 (SunOS 5.5.1)과 GNU gcc (Ver. 2.7.2.2.f.2) 컴파일러 환경에서 성능 비교를 수행하였다. <표 1>은 테스트 회로의 세부 사항을 나타낸다.

회로마다 임의로 발생시킨 입력을 이용하여 500,000번을 반복 실행한 출력을 비교하여 본 결과, C 코드 자동 발생기의 발생 코드와 Esterel의 발생 코드가 동일한 외부 출력을 나타냄을 알 수 있었다. 이는 두 프로그램이 논리적으로 동일한 의미를 지닌다는 것을 뜻한다.

두 프로그램의 코드 크기 비교를 위해 실행 파일을 Dump한 결과, C 코드 자동 발생기가 발생하는 코드는 Esterel이 발생하는 코드에 비해, 실행 명령어 크기에서 66.68% ~ 94.36%, 평균 86.66%, 메모리 할당 크기에서 44.10% ~ 93.93%, 평균 77.49%로 감소되었다. 전체 코드 크기는 59.79% ~ 93.99%, 평균 84.17%로 감소된 것이며, <그림 3>은 코드 크기 비교 결과이다. 실행 시간을 비교하여 보면, C 코드 자동 발생기가 발생하는 코드는 Esterel의 발생 코드에 비해 3.25배 ~ 5.33배, 평균 4.68배로 실행 속도가 향상되었다. <그림 4>는 실행 시간 비교 결과이다.

C 코드 자동 발생기가 발생하는 코드의 성능이 우월한 이유는, 내장형 시스템에서의 실행을 목적으로 프로그램의 코드 크기와 메모리 요구량을 최소화하면서 필요한 정보만을 포함하기 때문이다. 또한 문맥 전환을 최소화하기 때문에 실행 시간이 순수한 신호 전달 과정에 쓰이게 된다. C 코드 자동 발생기는 신호 흐름을 분석하는 과정에 복사 전파, 데드코드 삭제의 컴파일러 최적화 기법이 포함되어 있다.

반면, Esterel 의 유한 상태 천이 모델 기법은 신호 흐름 분석 기법에 비해 최적화가 어렵고, 컴포넌트들의 상태 확인을 위한 부분 때문에 처리하는 코드 양이 많아서 실행 시간 또한 길어진다. Esterel은 동시 시스템의 시뮬레이션을 지원하므로, 발생 코드에서 신호의 존재 여부를 확인하는 코드가 추가적으로 더 발생된다.

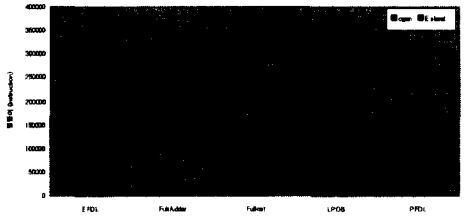
5. 결론 및 향후 연구 방향

C 코드 자동 발생기는 내장형 시스템에서 논리 회로를 시뮬레이션 하는 응용 프로그램을 자동으로 생성함으로써, 회로를 설계, 구현하는 과정에서 발생하는 비효율성을 제거하였다. 사용자는 프로그램에 대한 전문적인 지식 없이 논리 회로를 설계하는 것만으로 응용 프로그램을 생성할 수 있다. C 코드 자동 발생기가 생성한 응용 프로그램은 복잡한 하드웨어를 시뮬레이션을 통해 테스트하는 용도, 또는 내장형 시스템에서 직접 실행하여 하드웨어 논리 회로를 대치하는 용도로 사용될 수 있다.

논리 회로의 포트는 char 또는 bit-vector를 사용하여 나타냄으로써 메모리 사용량과 실행 시간을 최소화 하였으며, 복사 전파와 데드코드 삭제 등의 최적화 기법을 통하여 프로그램의 성능을 최적화하였다.

	EFDL	Full Adder	Fullost	LPOB	PFDL
외부입력	42	3	8	13	43
외부출력	24	2	1	6	18
내부노드	514	25	88	160	310
컴포넌트	109	5	21	39	63
에지정보	588	27	96	177	331

<표 1> 테스트 회로 세부사항



<그림 3> 코드 크기 비교

<그림 4> 실행 시간 비교

회로 시뮬레이션의 범용 도구인 Esterel을 이용하여 코드 발생기가 만들어내는 프로그램이 입력 회로의 의미를 그대로 보전함이 입증되었고, 코드 발생기가 생성한 코드가 범용 도구가 생성한 코드보다 공간 효율성, 시간 효율성 면에서 내장형 시스템에 적합함이 증명되었다.

생성된 함수 단위, 혹은 입력 회로 단위로 독자적인 실행과 재사용이 가능하도록 라이브러리로 구축하여 관리하는 부분은 향후 지속적으로 연구되어야 할 과제이다.

6. 참고문헌

[1] R. Lipsett, E. Marshner, " VHDL-The Language" , IEEE Design and Test, 28~41, 1986

[2] " www.mathworks.com" , The MathWorks, Inc.

[3] Craig Hansen, " Hardware Logic simulation by Compilation" , In Proceedings of the Design Automation, 712~715, 1988

[4] Taizo Kojima, Kaoru Tsuru, " ALHARD : An Application-Oriented Language for Hardware Simulation" , ACM, 223~229, 1990

[5] G. Berry, " Esterel on hardware" , Philosophical Transactions Royal Society of London A, 217~248, 1992

[6] 김진현, 김지영, 이상한, 표창우, 김지인, 최진영, 송호엽, 손혁수, " 시각적 컴포넌트 기반의 Embedded Application Software 개발 도구" , 한국정보처리학회, 961~964, 2000