재귀호출을 위한 합성 가능한 VHDL 코드 변환기 설계

홍승완, 안성용, 이정아
조선대학교 전자계산학과
swhong@rain.chosun.ac.kr dis@rain.chosun.ac.kr jeong@rain.chosun.ac.kr

Design of synthesizable VHDL transrator for recursive call

Seung--Wan Hong, Seong-Yong Ahn, Jeong-A Lee
Dept. Computer Science, Chosun University

요 약

시스템을 설계함에 있어 시스템의 성능과 비용 및 시간을 고려한 하드웨어 소프트웨어를 혼합한 통합설계(codesign) 환경이 많이 연구되고 있다. 통합 설계 과정을 자동화하기 위해서는 기술 언어를 툴에 맞게 자동적으로 바꾸어주는 기능이 필요하게 된다. C를 VHDL로 변환하는 방법에서 특히 동적 할당, 포인터, 재귀 호출에 대한 변환이 어렵다.

본 논문은 재귀 호출 부분을 제어부, 연산부, 입력부, 메모리로 나누어 각각을 component로 설계하게 만들었다. C언어로부터 합성 가능한 VHDL로의 변환 중 재귀 호출에 관한 연구를 수행함으로써 상위 수준에서의 시스템 설계를 할 수 있도록 도와주고, C로부터 VHDL로의 변환에 유연성을 부여하여, 설계를 자동화시키는데 기여할 수 있을 것이다.

1. 서 론

점점 더 복잡해지고 다양화 되어지는 시스템을 구성하는 각 요소(component)를 설계함에 있어 시스템의 성능과 비용 및 시간을 고려한 하드웨어와 소프트웨어를 혼합한 통합 설계(codesign) 환경에 대한 연구가 활발히 진행되고 있다.

최근에는 합성 가능한 상위 수준 합성 툴(high level synthesizer)에 관련된 기술의 발달은 개발자의 설계 추상화 수준을 높여 구조적 기술(structural description)뿐만 아니라, 함수적 기술(behavioral description)을 통한 하드웨어의 설계도 가능하게 하였다. 이러한 발달은 하드웨어로만 이루어진 시스템의 설계를 용이하게 하였을 뿐만 아니라 하드웨어로 만들 대상의 범위에 유연성을 요구하는 통합 설계 환경이 실제적인 설계환경으로서 설득력을 가질 수 있는 요인이 되었다. 보통 하드웨어 시스템 설계에서는 하드웨어 기술 언어(hardware description language, HDL)을 쓰고, 소프트웨어 시스템의 설계에서는 C와 같은 상위 수준 프로그래밍 언어(high level language, HLL)을 쓴다. 서로 다른 이질적인 부분들로 구성된 소프트웨어 부분과 하드웨어 부분이 혼합된 시스템(codesign) 환경에서는 어떤 기술 방식을 써야 할지가 분명하지 않다. 각 부품(component)를 어떤 기술 언어를 사용하는가에 따라 합성에 영향을 미치기 때문이다. 왜냐면 실제로 쓰이는 합성 툴들이 모든 기술 언어를 지원하지는 않기 때문이다. 이를테면 Synopsys Design Compiler의 경우는 VHDL로 된 입력은 받아들이지만 C로 된 입력은 받아들이지 않는다. 일반적으로 하드웨어로 합성하기 위한 알고리즘을 C로 기술하지는 않기 때문이다. 그러나 통합 설계환경의 분할 단계에서는 C로 기술되었던 부분을 다시 VHDL로, VHDL로 기술되었던 부분을 C 언어로 변환하는 경우가 발생한다. 따라서 통합 설계 과정을 자동화하기 위해서는 기술언어를 툴에 맞게 자동적으로 변환해 주는 기능이 필요하게 된다. VHDL을 C 언어로 변환은 compiled-code simulation 과 관련하여 많은 연구가 되어져 왔으나, 상대적으로 C를 VHDL로 변환하는 연구는 미미하다. 특히 변환 과정에서 동적 할당, 포인터 그리고 재귀 호출에 관한 문제를 해결해야 하는데, 포인터에 관한 연구는 많이 진행되고 있지만, 재귀 호출에 관한 연구는 전무한 상태라 할 수 있다.

본 논문은 이러한 기술 언어의 자동 변환의 한 예인 C로부터 VHDL로 전환하는 방법 중 재귀 호출 구문을 합성 가능한 VHDL 구문으로 변환하는 과정을 다룬다.

2. 국내외 연구 동향

최근 들어 통합설계를 연구하는 그룹들에 의하여 C로부터 VHDL로의 전환을 자동화한 시스템들이 등장하기 시작했다. 간단하게 살펴보면,

• C2VHDL : Queensland 대학에서 통합설계 시스템 구축의 일환으로 개발된 툴로써, 표준 C의 대부분에 구문을 전환 할 수 있으나, 포인터 구문은 전환하지 못하며, 계층적인 구조를 가지지 않는 단순한 형태의 입력만을 수행한다.[1]

• Cosyma : Braunschweig 대학에서 개발한 통합설계 시스템으로, 소프트웨어로부터 시간이 오래 걸리는 부분을 하드웨어로 분할해 나가는 방식이다. 하드웨어 기술 언어로 VHDL 대신 HardwareC를 사용하였으며, 시스템의 성능에 초점을 맞추고 개발되었다.[2]

• 서울대 연구 : C로부터 합성 가능한 VHDL로의 자동 변환을 수행하며, 포인터 구문에 중점을 두고 개발되었다. 동적할당 및 재귀호출, 이중 포인터에 대한 것은 수행하지 못한다. 본 논문은 이 연구의 계속으로 재귀호출에 관하여 중점적으로 연구한다.[4]

• SpC : Stanford 대학에서 개발한 툴로써, 그래프 이론을 바탕으로, 포인터에 중점을 두고 합성가능하고, 최적화된 VHDL 코드를 개발하는데 중점을 두었다. 그러나 이 툴도 동적 할당과 재귀 호출을 수행하지 못한다.[5]

3. VHDL에서의 재귀 호출의 문제점

일반적으로 재귀 호출을 사용하는 이유는 재귀적인 함수 정의가 인간의 자연적인 사고를 표현하기에 적합한 경우가 많기 때문이다.

C에서의 재귀 호출 구문은 다음과 같은 형태이다.

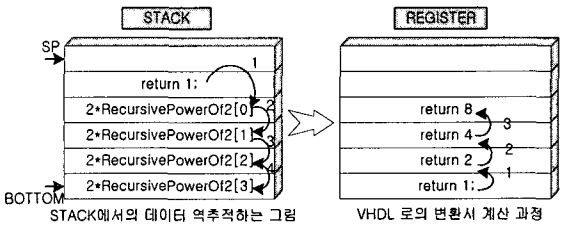
• 기본 조항 : 재귀조건을 종료하게 만드는 간단한 문장

• 재귀 조항 : 재귀 조건을 명시한다.

• 한계 조항 : 예외사항에 대한 조건을 제시하는 문장

C 언어에서는 재귀 호출을 수행은 STACK 이라는 구조를 이용하게 된다. STACK은 초기 조건에 도달할 때까지 push를 수행하고, 초기 조건에 도달하게 되면 주어진 초기 값을 이용하여 pop를 수행하여 나온 결과 값을 함수에 되돌려 주는 것이다. 하드웨어 기술 언어에서의 재귀 호출의 어려움은 재귀 호출이 몇 번 수행될 것인가를 알지 못한다는 것이다. 이러한 동적 객체는 하드웨어에서는 구현하기가 어렵다. 왜냐하면, 일반적인 하드웨어에서는 모든 객체가 hard-wired되어 있어야 하고, 이는 합성 초기 단계에서 정적으로 결정되기 때문에 이러한 동작이 불가능하게 되는 것이다. 즉, hardware에서는 함수의 호출이 그 함수에 해당하는 component의 instantiation 형태로 구현되어 지기 때문에 상위 수준 합성에서 할당(allocation)과 스케줄링(schedulling) & 바인딩(binding)을 위해서는 그 함수가 몇 번 불리 것인지를 정적으로 알아야 한다. VHDL에서도 C와 마찬가지로 포인터와 같은 일을 수행하는 access라는 구문이 있다. 이를 사용하여 변환을 하면 simulation은 할 수 있지만, 합성이 되지 않는다. 본 논문에서는 이를 위해서는 별도의 메모리 시스템과 이를 관리하기 위한 제어기를 두고 이곳에서 호출 횟수를 결정하여[3] 재귀 호출을 수행한다.

그림 1은 C에서 재귀 호출시 데이터 저장 상태와 VHDL로의 변환시 데이터의 흐름을 보여준다. C에서는 STACK에 push할 때 정해지지 않은 크기 및 값들이 역추적에 의하여 정해진 중간 결과 값들을 생성해 낸다. 본 논문의 변환기에서는 값이 정해지지 않은 중간 단계를 없애기 위해 제어기에서 몇 번 수행되는가를 결정하기 때문에 C에서의 역추적이 필요 없게 된다. VHDL로의 변환에서는 중간에 생성되는 값들은 필요가 없기 때문에 하나의 레지스터에서 계속 값을 갱신하게 된다. 갱신된 마지막 결과 값은 RAM에 저장된다.

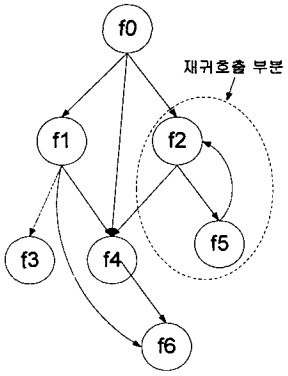


<그림 1> 2^3에 대한 스택 수행과정 및 변환시 수행 과정

재귀 호출시 쓰이게 되는 메모리 시스템(RAM)은 STACK의 역할을 할 수 있는 특별한 형태의 합성 가능한 RAM을 구현하여, 일반적인 소프트웨어 기술(software description language) 언어에서 수행되는 재귀 호출의 방식과 비슷하게 수행되도록 구현하였다.

4. 재귀 호출 구문의 VHDL 전환

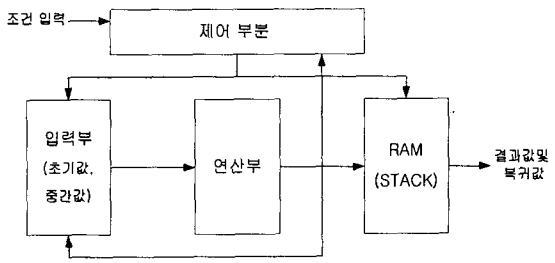
우선 C의 함수 구문 중에서 어느 부분이 재귀 호출이 이루어지고 있는가를 알아야 한다. 즉, 함수에서 caller가 callee를 호출하는 관계를 고려할 때, 만일 호출 관계에서 caller를 포함하는 cycle이 존재하면 이 함수(caller)는 재귀 호출의 성격을 갖는다고 판단할 수 있다. 변환기에서는 각 함수(caller)들을 조사하여 이 사항에 해당하는 caller를 찾아내어 VHDL 구문으로 변환하는 루틴으로 넘어가게 된다.



<그림 2> 함수의 호출 관계

C에서의 재귀 호출을 VHDL로 변환시 크게 4부분으로

나눌 수가 있다.



<그림 3> VHDL에서의 재귀호출 구성도

- 초기값, 중간 결과값 입력 부분 : 초기값과 중간 결과값을 넘겨 받아, MUX에 의해서 초기값이나 중간 결과값을 넘겨준다.
- 재귀 조항 부분 : 재귀 조항에 의해 연산이 이루어진다.
- 메모리 부분 : RAM을 이용하여 결과값(return value)과 메인 함수에서 재귀 호출 함수로 넘어올 때 저장하여야 할 변수들의 값(복귀값)들을 STACK과 같은 방법으로 저장한다.
- 제어 부분 : 초기값 및 종결조건, 중간 결과값 입력 부분과 메모리 부분을 제어한다. C 에서는 재귀 호출을 몇번 수행해야 되는지를 알려주는 값을 파라미터로 넘겨주게 된다. 이 파라미터를 이용하여 제어기에서 반복횟수를 계산하여 제어 신호를 발생하게 된다.

5. 변환 과정

VHDL 구문에서는 기본조항(초기값)과 수행 횟수를 entity에서 입력으로 받아들이고, 결과 값과 복귀 값은 출력으로 표시되어 지며, 재귀 조항과, 한계 조항은 architecture의 몸체에서 component화되어 제시되게 된다.

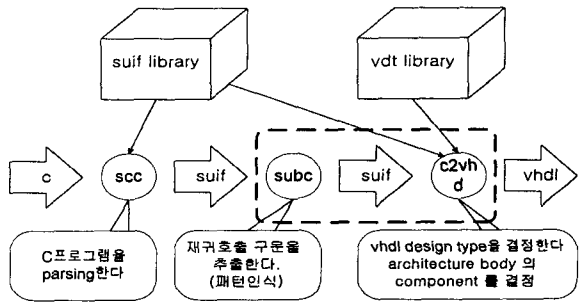
```
fibo(int n)
{
   if (n==0 || n==1)
   return 1;
   else
   return fibo(n-1) + fibo(n-2);
}

void main()
{
   int m,i=0;
   i = 6;
   m= fibo(i);
   cout << "fibo = " << m << "\n";
}
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity fibo is
   port( i : in std_logic_vector(7 downto 0);
         init 1: in std_logic_vector(7 downto 0);
         init2 : in std_logic_vector(7 downto 0);
         return : out std_logic_vector(7 downto0));
end fibo;

architecture behav of fibo is
         .....
         .....
component adder
   port (a : in std_logic_vector(7 downto 0);
         b : in std_logic_vector(7 downto 0);
         sum : out std_logic_vector( 7 downto 0));
end component;
         .....
         .....
```

<그림 4> C에서 VHDL로의 변환 예

결과 값과 복귀 값은 RAM에 저장되게 되며, push라는 신호를 받아들이게 되면 address를 일정하게 증가시키고, pop이라는 신호를 받으면 address를 일정하게 감소시킬 수 있게 설계를 했다.

재귀호출 구문을 추출하는데는 SUIF(Stanford University Intermediate Form) library를 사용하였다.[4] 구문분석이 끝나면, 패턴 인식에 의해 구문 중에서 재귀 호출 구문을 추출하고, 이 재귀 호출에 맞는 VHDL구문으로 변환을 수행한다. VHDL 코드를 생성하는 부분은 VDT(VHDL Developer's Toolkit)을 이용하였다. VHDL 구문의 합성 여부는 Synopsys Design Compiler를 통하여 합성 가능 여부를 검증하였다.



<그림 5> 변환 수행 과정

6. 결론

변환기를 이용하여 재귀 호출 구문을 변환하였을 때, 파라미터의 값이 정해진 재귀 호출(피보나치, 수열)은 원활히 수행되는 것을 볼 수 있으나, 파라미터의 값이 정해지지 않은 재귀호출(탐색 알고리즘, fan-out tree)은 수행횟수를 매뉴얼하게 적용해야 된다는 문제점이 있다.

본 논문의 변환 방법을 적용함으로써, 상위수준의 시스템 설계에 도움을 줄 수 있으며, C에서 VHDL로의 변환에 유연성을 부여할 수 있을것으로 기대된다.

7. 참고 문헌

[1]M.F.Parkinson, P.M.Taylor, and Sri Parameswaran, "C to VHDL Converter in a Codesign Environment", Proceedings of VHDL International User's Forum, 1994.

[2] R.Ernst, J.Henkel, and T.Benner, "Hardware-Software Cosynthesis for Microcontrollers", IEEE Design & Test of Computers,pp.64-75, December 1993.

[2] G.D.Jong, B.Lin, C. Verdonck, S.Wuytack, and F. Catthor, "Background Memory Management for Dynamic Data Structure Intensive Processing Systems", Proceedings of International Conference of Computer-Aided Design, pp. 515-520, 1995.

[4] 이화용, "C로부터 합성 가능한 VHDL로의 변환" 서울대학교 석사학위 논문, 1998

[5] Luc Semeria, Giovanni De Micheli "spC : Sybthesis of Pointers in C. Application of Pointer Analysis to the Behavioral Synthesis from C", proceeding of the 1998 ICCAD, pp.340-346, November 1998.