C-to-SystemC 합성기의 설계 및 구현
Design and Implementation of a C-to-SystemC Synthesizer

유명근, 송기용
Myoung-Keun You, Gi-Yong Song

요 약

본 논문에서는 입력 동작에 대하여 상위수준 합성을 수행한 후, 합성결과를 SystemC 코드로 전환하는 C-to-SystemC 합성기를 설계 및 구현하였다. 구현된 합성기의 처리과정은 C 소스코드로 기술된 입력 동작을 list 스케줄링 알고리즘을 이용하여 스케줄링한 후, 스케줄링 결과에 left-edge 알고리즘을 이용하여 레지스터 할당을 수행한다. 레지스터 할당 정보를 이용하여 합성기는 채널 및 포트와 같은 SystemC 특성들로 표현된 SystemC 모듈의 코드를 최종적으로 생성한다. C-to-SystemC 합성기의 동작은 EWF(elliptic wave filter)의 합성결과인 SystemC 모듈의 코드를 시뮬레이션하여 검증한다. C-to-SystemC 합성기는 SystemC 설계방법론의 모델링단계부터 합성단계에 이르는 툴 체인의 한 부분으로 사용될 수 있으며, 생성된 SystemC 모듈은 C 모듈에 비해 재사용이 용이하고 다른 SystemC 모듈과 SystemC 채널을 통하여 별도의 추가처리 없이 통신할 수 있다.

Abstract

A C-to-SystemC synthesizer which processes the input behavior according to high-level synthesis, and then transforms the synthesis result into SystemC module code is implemented in this paper. In the synthesis process, the input behavioral description in C source code is scheduled using list scheduling algorithm and register allocation is performed using left-edge algorithm on the result of scheduling. In the SystemC process, the output from high-level synthesis process is transformed into SystemC module code by combining it with SystemC features such as channels and ports. The operation of the implemented C-to-SystemC synthesizer is validated through simulating the synthesis of elliptic wave filter in SystemC code. C-to-SystemC synthesizer can be used as a part of tool-chain which helps to implement SystemC design methodology covering from modeling to synthesis.

Keywords : C-to-SystemC synthesizer, high-level synthesis, list scheduling algorithm, left-edge algorithm

Ⅰ. 서 론

최근 시스템의 설계가 대형화되고 복잡해지면서 상위수준 추상화에 기반한 설계범위 탐색이 보다 중요시되고 있다. 이에 따라 상위수준 추상화를 이용하여 시스템을 설계하는 SystemC [1-3]와 같은 시스템수준 설계언어가 주목을 받고 있다. SystemC는 하드웨어 모델링용 라이브러리를 포함시킨 C++ 클래스 라이브러리와 시뮬레이션 커널로 구성된 시스템수준 설계언어로, 하드웨어 유닛과 소프트웨어 모듈을 동시설계 할 수 있는 환경을 제공한다.

본 논문에서 구현된 C-to-SystemC 합성기는 먼저 C 소스코드로 기술된 동작에 대해 스케줄링단계를 수행한 후, 스케줄링된 결과를 이용하여 레지스터를 할당한다. 그리고 레지스터 할당의 결과를 이용하여 SystemC 코드로 전환하여 출력파일을 생성한다. 스케줄링과 레지스터 할당은 상위수준 합성의 방법에 따라 행위적 기술(behavioral description) 입력에 대하여 수행하며, 상위수준 합성은 행위적 모델을 저장장치와 기능유닛의 연결로 변환한다.

일반적으로 행위적 모델은 데이터 흐름 또는 순차 그래프로 표현되며 상위수준 합성의 처리과정은 분할, 스케줄링, 할당이라는 3단계로 구성된다. 분할은 행위적 모델을 하위 동작으로 구분하는 것을, 스케줄링은 변수 및 연산에 시간 간격을 부여하는 것을, 그리고 할당은 변수 및 연산에 저장장치 및 기능유닛을 배정하는 것을 의미한다 [4-5]. 본 논문에서는 ASAP(as-soon-as-possible), ALAP(as-late-as-possible), list 스케줄링 알고리즘과 레지스터 할당을 위하여 LEA(left-edge algorithm)을 이용하였다.

C 소스코드로 기술된 동작은 데이터 의존성과 연산들의 병렬성을 고려하여 파싱테이블을 이용한 중간 표현형으로 변환된다. C-to-SystemC 합성기의 입력은 C 소스코드이며 출력은 SystemC 모듈의 코드이다. C-to-SystemC 합성기의 동작은 EWF의 합성결과인 SystemC 모듈의 코드를 시뮬레이션하여 검증한다. C-to-SystemC 합성기는 SystemC 설계방법론의 모델링단계부터 합성단계에 이르는 툴 체인의 한 부분으로 사용될 수 있다.

*충북대학교
접수 일자 : 2009. 2. 6 수정 완료 : 2009. 4. 24
게재확정일자 : 2009. 4. 29
※이 논문은 2007학년도 충북대학교 학술연구지원사업의 연구비지원에 의하여 연구되었음.

본 논문의 구성은 다음과 같다. 2장은 본 논문에서 이용한 상위수준 합성 알고리즘과 SystemC에 대하여 간단히 설명하고, 3장은 C-to-SystemC 합성기의 처리과정을 기술한다. 4장은 EWF를 이용한 합성 및 시뮬레이션을 통하여 C-to-SystemC 합성기의 동작검증을 기술하며, 마지막으로 5장에서 결론을 맺는다.

II. 기본 이론

2.1 ASAP・ALAP 스케줄링

ASAP는 제약사항 없이 최소 지연을 갖는 스케줄링 알고리즘으로 각 연산들의 스케줄단계는 데이터 의존성을 만족하는 최소의 단계가 된다. ALAP는 ASAP에 상반되는 알고리즘으로 각 연산들의 스케줄단계는 스케줄 가능한 범위 가운데 가장 늦은 단계가 된다.

다수의 스케줄링 알고리즘은 스케줄링하고자 하는 연산들의 가장 빠른 스케줄단계와 가장 늦은 스케줄단계로 결정되는 이동성(mobility)을 이용한다. 즉, 연산자의 이동성은 ASAP 및 ALAP 스케줄링 알고리즘을 이용하여 얻을 수 있다 [4].

2.2 List 스케줄링

List 스케줄링 알고리즘은 주어진 자원 제약사항을 만족시키는 ASAP 스케줄링 알고리즘으로, 스케줄링된 선행연산자를 가지는 연산자들에 대하여 우선순위 리스트를 이용하여 스케줄링한다. 우선순위 리스트는 스케줄링될 연산자들의 자원경쟁을 해결하는 우선권 부여기능(priority function)을 기준으로 정렬된다.

List 스케줄링은 ready 연산리스트의 처음에 위치한 연산들을 모든 자원이 할당되어 더 이상 할당할 자원이 없을 때까지 스케줄링하는 동작을 반복한다. List 스케줄링에서 연산자의 이동성과 같은 우선권 부여기능은 중요한 역할을 수행한다. 그림 1은 [4]에서 발췌한 그림으로 자원 제약사항이 곱셈기 2개, 덧셈기 1개, 뺄셈기 1개, 비교기 1개인 list 스케줄링의 예이다. 그림에서 각 연산의 이동성을 우선권 부여기능으로 사용하였으며 < > 안에 표기하였다 [4].

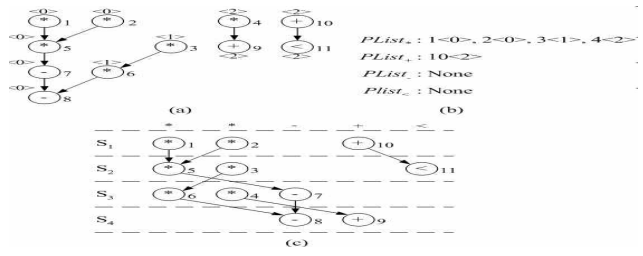

그림 1. (a) 이동성을 라벨로 표시한 DFG
(b) 제어 상태 S1의 ready 연산자들의 리스트
(c) 스케줄링된 DFG
Fig. 1. (a) DFG with mobility labeling
(b) list of ready operations for control state S1
(c) scheduled DFG

2.3 Left-Edge 알고리즘

LEA [6]는 물리적 설계 자동화의 채널 라우팅을 위해 제안된 알고리즘으로, 채널 라우팅의 주된 목적은 채널경계에 위치한 핀들을 연결하는데 사용되는 트랙의 수를 최소화하는 것이다. LEA는 [7]에서 레지스터 할당의 문제를 해결하는데 적용되었으며, LEA는 변수들의 리스트와 각 변수의 존속 기간(lifetime interval)을 입력받아 레지스터를 할당한다.

LEA는 입력받은 모든 변수들이 레지스터에 할당될 때까지 여러 번의 반복동작을 수행하는데, 각 반복동작에서는 변수들의 존속기간이 최대한 겹치지 않는 범위에서 많은 변수를 각 레지스터에 할당하게 된다 [4]. 그림 2는 [5]에서 발췌한 그림으로 LEA의 예이다.

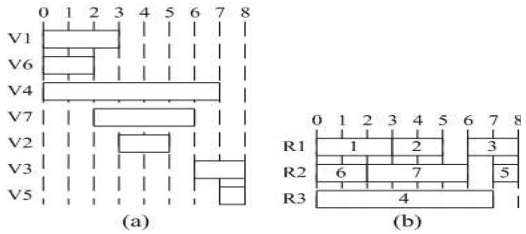

그림 2. (a) 정렬된 변수의 존속기간
(b) 3개의 레지스터 할당
Fig. 2. (a) lifetime interval of variables
(b) register allocation

2.4 SystemC

SystemC [1-3]는 소프트웨어 모듈, 하드웨어 구성요소, 인터페이스를 사이클 정확도 모델로 설계할 수 있도록 제공되는 C++ 클래스 라이브러리인 동시에 설계방법론이다. SystemC는 하드웨어 구성요소들의 병렬성을 프로세스로, 시스템의 구조적인 구성을 모듈, 포트, 채널로 처리하며, 또한 클럭의 시간 개념을 지원한다.

SystemC는 상위수준 추상화를 이용한 설계부터 점진적 구체화를 통한 합성 가능한 RT수준의 디자인에 이르기까지 설계측면에서 다양한 수준의 추상화를 제공한다. SystemC의 특징으로는 모듈, 프로세스, 포트, 시그널, 클럭 등이 있다. 그림 3은 SystemC의 설계방법론을 나타낸다.

그림 3. SystemC 설계 방법론
Fig. 3. SystemC design methodology

SystemC로 설계된 디자인은 점진적인 구체화를 통하여 시스템구성에 필요한 하드웨어 구성요소를 추가할 수 있으며, 또한 비교적 쉽게 설계오류들을 찾아 디버깅할 수 있다. 즉, SystemC는 단일 언어를 이용하여 시스템수준에서 RT수준에 이르기까지 설계가 가능하다는 특징을 갖는다 [2-3].

III. C-to-SystemC 합성기

C-to-SystemC 합성기의 처리과정은 크게 다음과 같이 4단계로 구성된다.

1. 입력받은 C 소스코드를 이항 연산자를 기준으로 파싱하는 단계
2. List 스케줄링 알고리즘을 이용한 스케줄링 단계
3. LEA를 이용한 레지스터 할당단계
4. 상위수준 합성처리 결과를 SystemC 모듈 코드로 생성하는 단계

그림 4는 C-to-SystemC 합성기의 처리과정이며, 각 단계의 세부내용을 아래에 기술한다.

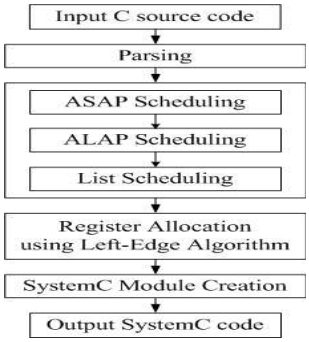

그림 4. C-to-SystemC 합성기의 처리과정

Fig. 4. Process of C-to-SystemC synthesizer

3.1 파싱

파싱단계에서는 연산자들을 스케줄링하기 위하여 입력된 C 소스코드를 이항 연산자를 기준으로 파싱한다. 이는 입력된 C 소스코드가 각 구문에 두개 이상의 연산자를 가질 수 있기 때문이다. 본 논문에서는 모든 변수는 두 번 이상 할당되지 않는다고 가정을 하였다.

하나의 구문에 두개 이상의 연산자가 포함된 경우, 파싱단계는 연산자 우선순위를 고려한 임시변수들을 생성하여 이용한다. 파싱단계의 결과로 생성되는 파싱테이블은 그림 5와 같은 *Table_entity* 구조체 타입의 배열이다. *Table_entity* 구조체 타입의 멤버변수 중 *char ** 타입으로 선언된 *output*, *left_input*, *right_input*은 문자열로 표시되는 변수들을 가리키며, *char* 타입의 *op*는 연산자를 저장한다. 멤버변수 중 *control_step*과 *scheduled_ flag*는 스케줄링 단계에서 사용된다.

```
struct Table_entity
{   char * output;
    char * left_input;
    char * right_input;
    char op;
    int control_step;
    bool scheduled_flag; };
```

그림 5. *Table_entity* 구조 타입

Fig. 5. *Table_entity* structure type

03.2 스케줄링

스케줄링단계는 ASAP 스케줄링, ALAP 스케줄링, list 스케줄링단계로 다시 세분화되어 수행된다. 각 연산의 이동성은 ASAP 스케줄링과 ALAP 스케줄링을 차례로 수행하여 얻은 *Table_entity* 구조체 타입의 배열로 구성된 스케줄링테이블을 이용하여 계산된다. List 스케줄링은 연산자 이동성에 따라 ready 연산자의 우선순위를 결정하고, 우선순위로 정렬된 ready 연산자리스트의 처음에 위치한 연산자들을 자원에 할당한다.

List 스케줄링 알고리즘에 주어지는 자원 제약사항들은 프로그램을 실행시키는 명령어의 파라미터(command line parameter)로 전달되며, 스케줄링 수행결과로 얻어지는 list 스케줄링테이블은 *Table_entity* 구조체 타입의 배열로 구성된다.

3.3 할당

할당단계는 본래(primary) 입출력과 파싱단계에서 생성한 임시변수들을 LEA을 이용하여 레지스터에 바인딩하는 동작을 수행한다. 그림 6과 같은 *Variable_Info* 구조체 타입의 배열로 구성되는 할당테이블은 모든 입출력 및 변수들의 이름과 존속기간을 포함한다. 할당테이블은 list 스케줄링테이블 정보를 이용하여 얻을 수 있으며, 할당테이블의 배열 구성요소는 *Variable_Info* 구조체 타입의 멤버변수인 *start*를 기준으로 정렬된다.

```
struct Variable_Info
{   char * name;
    int start;
    int stop; };
```

그림 6. *Variable_Info* 구조 타입

Fig. 6. *Variable_Info* structure type

각 변수의 존속기간은 *Variable_Info* 구조체 타입의 멤버변수인 *start*와 *stop* 값을 이용하여 계산되며, 본래 입력의 *start* 값은 시작 제어단계로, 출력의 *stop* 값은 마지막 제어단계로 각각 간주한다. 본래 입력은 임시변수들을 레지스터에 할당하기 전에 미리 저장되며, 출력은 레지스터 할당이 모두 끝난 후 레지스터로부터 읽혀진다.

Integer 타입의 2차원 배열 *reg[ ][ ]*는 LEA을 이용하여 레지스터를 할당할 때 생성된다. 2차원 배열의 행 인덱스는 할당된 레지스터를, 열 인덱스는 각 제어단계에 스케줄링된 변수의 정보를 각각 의미한다. List 스케줄링테이블에서 본래 입출력과 임시변수들을 할당된 레지스터로 교체하면 상위수준 합성 처리결과의 테이블을 얻을 수 있다.

3.4 SystemC 모듈 생성

이번 단계에서는 레지스터 할당된 테이블정보를 이용하여 SystemC 모듈을 생성하며, SystemC 모듈을 생성하는

데 필요한 모듈이름과 포트 폭은 명령어 파라미터로 전달된다. 레지스터 할당된 테이블정보를 이용하여 본래 입출력은 *sc_in<sc_int< > >* 타입과 *sc_out<sc_int< > >* 타입의 포트로 각각 매핑되고, 임시변수들은 *sc_signal<sc_int< > >* 타입의 채널로 매핑된다.

상위수준 합성결과는 *SC_THREAD* 매크로를 통하여 SystemC 모듈의 프로세스로 구현되며, 시뮬레이션의 elaboration 단계에서 입력포트를 읽어 할당된 레지스터에 입력값을 저장한다. 상위수준 합성결과의 제어단계는 SystemC의 *wait( )* 구문으로 나타나며, *sc_time_stamp( )* 구문을 추가하여 각 연산들의 실행시간 정보를 출력한다. C-to-SystemC 합성기의 최종 출력인 SystemC 모듈코드에 대하여 SystemC 커널을 이용하여 시뮬레이션 함으로써 C-to-SystemC 합성기의 동작을 검증한다.

IV. SystemC 합성 및 시뮬레이션

본 논문에서는 C-to-SystemC 합성기의 타겟모듈로 5차 EWF [5]을 사용하여 list 스케줄링한 후, 생성된 CDFG(control and data flow graph)를 그림 7에 보인다. 명령어 파라미터를 이용하여 자원 제약사항으로 덧셈기 2개와 곱셈기 1개로 설정하였으며, 본래 출력은 내부노드가 아닌 단말노드에서 생성되도록 하였다.

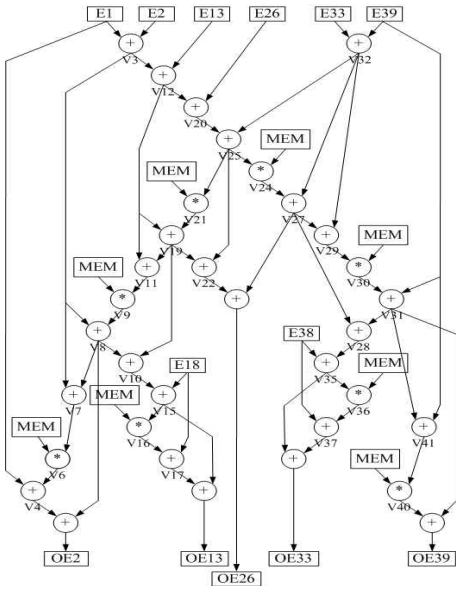

그림 7. EWF를 list 스케줄링 후 생성된 CDFG
Fig. 7. CDFG of list-scheduled EWF

List 스케줄링테이블을 이용하여 레지스터 할당한 결과를 그림 8에 보이며, C-to-SystemC 합성기의 최종 출력인 SystemC 모듈코드의 일부를 그림 9에 보인다. 그림 9는 SystemC 모듈코드 생성시 명령어 파라미터로 SystemC 모듈의 이름과 입출력 포트의 폭을 각각 output_module과 16으로 전달한 경우이다.

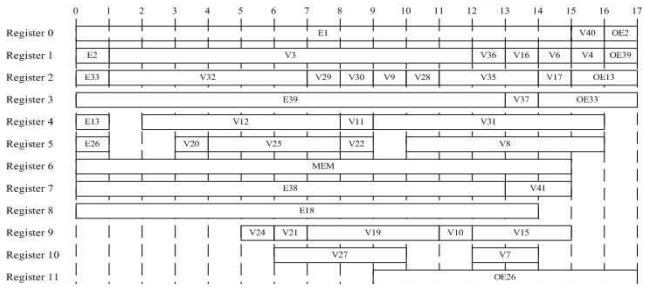

그림 8. 레지스터 할당
Fig. 8. Register allocation

```
#include "systemc.h"
sc_time CLK_Cycle(10, SC_NS);
SC_MODULE(output_module)
{
 sc_in<sc_int<16> > E1;      sc_in<sc_int<16> > E2;
 sc_in<sc_int<16> > E33;     sc_in<sc_int<16> > E39;
 sc_in<sc_int<16> > E13;     sc_in<sc_int<16> > E26;
 sc_in<sc_int<16> > MEM;     sc_in<sc_int<16> > E38;
 sc_in<sc_int<16> > E18;     sc_out<sc_int<16> > OE26;
 sc_out<sc_int<16> > OE33;   sc_out<sc_int<16> > OE13;
 sc_out<sc_int<16> > OE2;    sc_out<sc_int<16> > OE39;

 sc_signal<sc_int<16> > r0;  sc_signal<sc_int<16> r1;
 sc_signal<sc_int<16> > r2;  sc_signal<sc_int<16> r3;
 sc_signal<sc_int<16> > r4;  sc_signal<sc_int<16> r5;
 sc_signal<sc_int<16> > r6;  sc_signal<sc_int<16> r7;
 sc_signal<sc_int<16> > r8;  sc_signal<sc_int<16> r9;
 sc_signal<sc_int<16> > r10; sc_signal<sc_int<16> r11;
 void proc_output_module( )
 { r0 = E1.read();   r1 = E2.read();   r2 = E33.read();
   r3 = E39.read();  r4 = E13.read();  r5 = E26.read();
   r6 = MEM.read();  r7 = E38.read();  r8 = E18.read();
   wait(CLK_Cycle);
   r1 = r0.read() + r1.read();
   cout << "\n[ " << sc_time_stamp() << " ]"
     << "r1=r0+r1" <<  '\t' << "      " << " = "  << r0
     << " + " << r1;
   wait(0, SC_NS);
   cout << "\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b " << r1;
   r2 = r2.read() + r3.read();
   cout << "\n[ " << sc_time_stamp() << " ] "
    << "r2=r2+r3" <<  '\t' << "      " << " = "  << r2
    << " + " << r3;
   wait(0, SC_NS);
   cout << "\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b " << r2;
   wait(CLK_Cycle);
   ......
   wait(CLK_Cycle);
   OE26.write(r11); OE33.write(r3);  OE13.write(r2);
   OE2.write(r0);   OE39.write(r1); }
 SC_CTOR(output_module)
 { SC_THREAD(proc_output_module); }
};
```

그림 9. SystemC 출력 파일
Fig. 9. SystemC output file

C-to-SystemC 합성기의 EWF에 대한 최종 출력인 SystemC 모듈의 코드를 *sc_main( )* 함수와 *sc_trace( )* 함수를 이용하여 시뮬레이션한 후, 올바른 동작여부 판별을 통하여 C-to-SystemC 합성기의 동작검증을 수행한다. 그림 10은 할당된 레지스터를 *sc_trace( )* 함수를 이용하여 VCD(value change dump) 파일의 파형에 등록한 후, 시뮬레이션 결과로 생성된 각 레지스터의 값의 변화를 보인 파형이다.

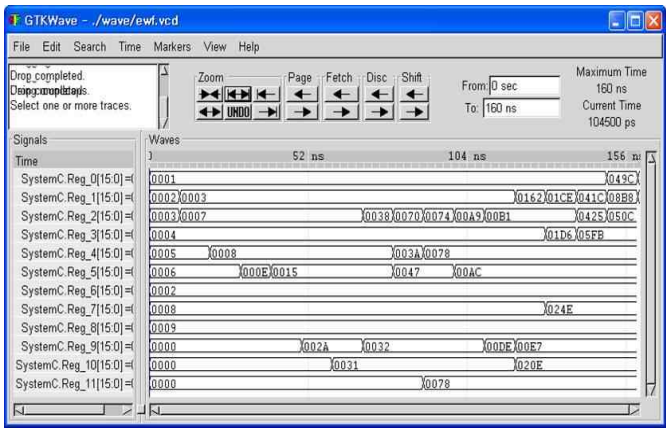

그림 10. VCD 파일의 레지스터 파형
Fig. 10. Register waveform of VCD file

위의 그림에서 *SystemC.Reg_1[15:0]*의 값의 변화는 그림 8에서 *Register 1*에 제어단계별로 할당된 입출력과 변수들의 값의 변화를 의미한다.

Ⅴ. 결 론

본 논문에서는 C 소스코드를 입력받아 상위수준 합성의 기법들을 이용하여 합성한 결과를 SystemC 모듈코드로 생성하는 C-to-SystemC 합성기를 설계 및 구현하였다. 구현된 C-to-SystemC 합성기에서 list 스케줄링 알고리즘과 레지스터 할당을 위한 LEA 알고리즘을 사용하였으며, 합성기의 최종 합성결과인 SystemC 모듈코드 생성을 위하여 상위수준 합성의 결과를 SystemC의 채널, 포트와 같은 구성요소로 변환하였다.

C-to-SystemC 합성기의 동작검증은 5차 EWF의 합성결과인 SystemC 모듈코드를 시뮬레이션한 후, 올바른 동작여부를 판별하였다. C-to-SystemC 합성기는 SystemC 설계방법론의 모델링단계부터 합성단계에 이르는 툴 체인의 한 부분으로 사용될 수 있을 것으로 판단된다.

참고 문헌

[1] J.Bhasker, *A SystemC Primer*, Star Galaxy Publishing, pp. 3-6, 2002.

[2] David C. Black, Jack Donovan, *SystemC : From the Ground Up*, Eklectic Ally, Inc., pp. 33-37, 2004.

[3] OSCI, *SystemC version 2.0 User Guide*, Synopsys Inc., 2001.

[4] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, Steve Y-L Lin, *High-Level Synthesis Introduction to Chip and System Design*, Kluwer Academic Publishers, pp. 215-217,233-236,283-285, 1991.

[5] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuits*, McGray-Hill, pp. 63-65,1994.

[6] A. Hashimoto and J. Stevens, "Wire Routing by Optimizing Channel Assignment within Large Apertures," *The 8th Design Automation Conference Workshop*, pp. 155-169, 1971.

[7] F.J. Kurdahi and A.C. Parker, "REAL:A Program for Register Allocation," *Proceedings of the 24th Design Automation Conference*, pp. 210-215, 1987.

[8] P.G. Paulin, *"High-Level Synthesis of Digital Circuits using Global Scheduling and Binding Algorithms,"* Ph.D. dissertation, Carlton Univ., Jan. 1988.

유 명 근(Myoung-Keun You)

2003년 충북대 컴퓨터공학과 학사
2006년 충북대 대학원 컴퓨터공학과 석사
2006년～현재 충북대 대학원 컴퓨터공학과 박사과정
※주관심분야 : 임베디드시스템 설계・검증, 상위수준 설계・검증

송 기 용(Gi-Yong Song)

1978년 서울대 공교과 학사
1980년 서울대 대학원 전자과 석사
1995년 미 루이지애나대 박사
1983년～현재 충북대 전자정보대학 교수
※주관심분야 : SoC 설계・검증, 상위수준설계, C++ 기반 하드웨어 검증