JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 17, No. 7 : 1653~1659, July. 2013

효율적인 HDL 디버깅을 위한 아키텍쳐 자동 생성 시스템

문대철[1] · Xie Cheng[1] · 박인학[2*]

Automatic Visual Architecture Generation System for Efficient HDL Debugging

Dai-tchul Moon[1] · Xie Cheng[1] · In-hag Park[2*]

[1] Department of Information & Communication Eng., Hoseo University, 165, Sechul-ri, Baebang-Myeon, Asan, Choongnam 336-795, Korea

[2] Department of Information Communication Engineering, Hoseo University, 165, Sechul-ri, Baebang-Myeon, Asan 336-795, Korea

요 약

본 논문은 Verilog HDL이나 VHDL로 설계된 디지털 회로의 구조를 효율적으로 분석하고 디버깅 할 수 있는 ECAD 소프트웨어를 제안한다. 이 소프트웨어는 HDL 코드를 파싱하여 내부 구조에 대한 정보를 추출한 후 여러 가지 종류의 그래픽 도우미 예를 들면, 배치배선 알고리즘을 적용하여 생성된 계층구조의 논리회로도, 각 모듈을 구성하는 요소들을 나타내는 객체 나무 그래픽, 인스턴스들의 계층구조를 나타내는 인스턴스 나무 그래픽, 내부 시그널 간의 관계를 나타내는 시그널 관계도(SPD, signal propagation diagram) 등으로 표현된다. 디버깅에 가장 중요한 기능은 여러 가지 다른 관점의 설계 정보(HDL 코드, 객체 나무, 인스턴스 나무, SPD, 파형 등)에서 임의의 객체로부터 출발하여 동일한 설계 정보를 찾아내는 기능이다. 이러한 기능들은 설계자가 수작업으로 HDL 코드를 분석하고 버그를 찾아내는 기능을 효율적으로 수행할 수 있도록 돕는다.

ABSTRACT

In this paper, we propose a new ECAD software for efficiently analyzing and debugging of digital architecture implemented in Verilog HDL or VHDL codes. This software firstly elaborates HDL codes so as to extract internal architecture structure, then generates several graphical aids such as hierarchical schematics by applying placement and routing algorithm, object tree to show configuration of each module, instance tree to show hierarchical structure of instances, and SPD (Signal Propagation Diagram) to show internal interconnections. It is more important function that same objects in different views(HDL codes, object tree, instance tree, SPD, waveform etc.) can be highlighted at the starting any object. These functions are sure to improve efficiency of manual job to fix bugs or to analyze HDL codes.

키워드 : Verilog/VHDL, SPD, Archian, 디버깅, 배치/배선, 메시지 패싱

Key word : Verilog/VHDL SPD, Archian, debugging, placement/routing, message passing

접수일자 : 2013. 02. 14 심사완료일자 : 2013. 04. 29 게재확정일자 : 2013. 05. 10

* **Corresponding Author** Xie Cheng(E-mail:nightkillerc@nate.com, Tel:+82-41-540-5378)
Department of Information Communication Engineering, Hoseo University, 165, Sechul-ri, Baebang-Myeon, Asan 336-795, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2013.17.7.1653 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

오늘날 시스템칩(SoC)은 점점 더 복잡해짐에 따라 시스템칩을 디버깅하는데 소요되는 시간과 비용이 증가하고 있다. 시스템칩을 디버깅 하는 방법으로는 시뮬레이터를 이용하는 검증과 FPGA 에뮬레이터를 이용하여 검증하는 두 가지 방법이 일반적으로 사용되고 있다. 전자는 시그널이나 모듈의 상세한 정보를 제공하는 점과 간결한 소프트웨어 환경이라는 장점을 갖는다. 반면에 시뮬레이션 시간이 많이 소요되는 점과 설계자가 테스트벤치를 수작업으로 생성하기 때문에 현실성이 떨어지는 단점이 있다. 시뮬레이션 검증 방법은 시스템칩 설계 초기에 알고리즘을 검증하거나 버그를 추출하는데 널리 사용되고 있다. 후자의 에뮬레이터 검증 방법은 검증 속도가 빠른 점과 시스템 보드 상에서 실제 부품을 이용한 다양한 테스트 벡터 생성을 통해 구현 및 검증이 가능하다는 장점을 제공한다. 단점이라면 Verilog 코드에서 FPGA 합성을 거쳐 에뮬레이션 검증까지 피드백 루프가 길다는 것과 추적할 수 있는 내부 시그널을 뽑아낼 I/O 핀 수가 제한된다는 단점을 가진다. 일반적으로 FPGA 에뮬레이션 검증은 시스템칩 설계의 마지막 단계에서 설계된 시스템칩이 시스템 보드에서 실제로 정확하게 동작하는지를 검증하는데 사용된다[1,2].

대용량의 복잡한 시스템 칩을 적절한 시간 안에 디버깅 하려면 시뮬레이션 벡터를 사용하여 Verilog 코드 상의 버그를 추적하는 툴이 효율적이어야만 한다. 검증 툴이 더욱 효율적일수록 설계자가 소모하는 디버깅 시간을 더욱 짧아지게 된다. EDA 툴의 검증 효율성을 높이기 위한 작업은 진화하는 설계 요구사항을 충족시키기 위해 지속적으로 발전하고 있다.

개발 초기에 설계자들이 버그나 오류를 찾아내는 작업은 매우 어렵고 피곤한 작업이다. HDL 코드의 간단한 문법 오류는 컴파일러에 의해 쉽게 검출된다. 그러나 HDL 소스 코드에 감추어져 있는 알고리즘 상의 버그는 수작업으로밖에 찾을 수 없다. 이 작업은 응용 회로에 따라 다르며 고도의 전문지식이 요구되기 때문에 자동화 될 수 없으며 많은 시간이 소요된다. 그러므로 설계자들이 쉽게 설계를 이해하고 버그를 추적할 수 있도록 돕는 스마트하고 지적인 새로운 디버깅 툴이 요구된다[3,5].

본 논문은 그래픽을 통한 아키텍쳐 디버깅을 지원하는 Archian이라는 이름의 새로운 툴을 소개한다. Archian은 HDL 코드를 분석하고 디버깅 할 수 있도록 개발되었다. Archian이 지원하는 효율적인 디버깅의 주요 기능은 다음과 같다. HDL 코드로부터 내부 구조의 논리회로도를 자동으로 생성하며, 다양한 종류의 설계 정보- 시뮬레이션 벡터, HDL 텍스트 코드, 인스턴스 나무, 객체 나무, 논리회로도 및 SPD -를 함께 고려하여 디버깅 하는 기능을 지원한다.

Ⅱ. Archian

Archian은 HDL 코드 디버깅을 위해 개발된 ECAD 툴이다. 주요 기능은 HDL 코드의 파싱, 아키텍쳐의 논리회로도 생성, 인스턴스들의 심볼 생성과 설계 정보의 하이라이팅(highlighting) 기능이다. HDL 파서는 IEEE 표준 문법에 따라 HDL 소스 코드에서 모듈 이름, 단자 정보, 시그널과 인스턴스 정보 등을 추출하여 모듈별로 파싱 파일에 저장한다. 파싱 파일은 설계 구조에 대한 모든 정보를 갖고 있기 때문에 Archian의 모든 기능의 베이스 캠프와 같은 역할을 한다. 모듈별로 내부의 구조를 구성하는 인스턴스의 심볼과 입출력 단자를 배치하고 이들을 연결함으로서 논리회로도를 생성한다. 구조에 대한 다른 표현을 나타내는 그래픽 도우미들도 파싱 파일로부터 생성된다. 설계의 관점은 다르지만 각각의 표현에는 동일한 설계 객체들이 존재한다. 다른 설계 표현에서 동일한 객체를 하이라이트 시키는 기능은 메시지 패싱 기술로 구현되었다[4,6].

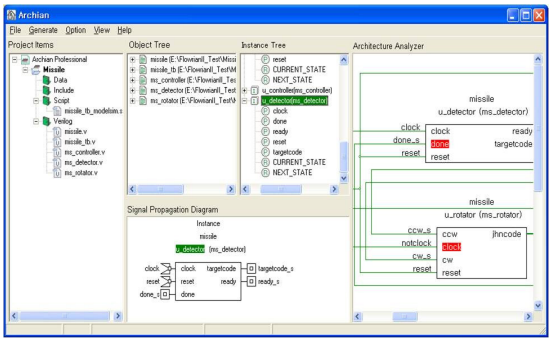


그림 1. 그래픽 창들
Fig. 1 Graphic windows

2.1. 메인 화면의 구성

그림 1에 나타난 Archian 화면은 프로젝트 창, 객체 나무 창, 인스턴스 나무 창, SPD 창과 아키텍처 창으로 구성된다.

2.2. 아키텍쳐 생성 흐름

그림 2는 아키텍쳐를 생성하는 흐름도를 나타낸다. HDL 파서는 HDL 프로젝트에 등록된 HDL 코드를 분석하기 위하여 제일 처음에 수행된다. 다음으로 파싱 정보를 이용하여 객체 나무와 인스턴스 나무가 생성된다. 배치 및 배선 알고리즘에서는 아키텍처의 논리회로도를 생성하기 위하여 구조를 이루는 요소들인 인스턴스 및 단자의 배치와 연결선의 배선이 이루어진다. 텍스트 편집기와 파형분석기는 사용자의 디버깅 요청이 발생할 때 나타난다.

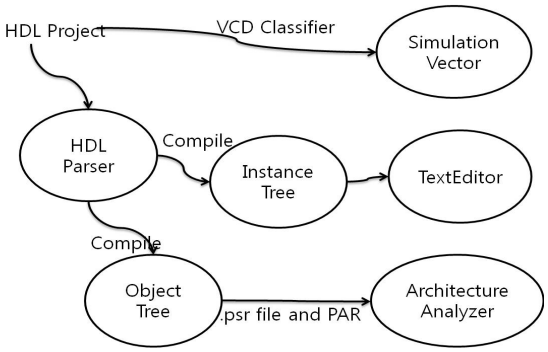


그림 2. 아키텍처 생성 프로세스 흐름도
Fig. 2 Architecture process work flowchart.

Ⅲ. 구현 알고리즘

3.1. 하이라이팅을 위한 메시지 패싱

본 절에서는 다양한 설계 표현과 이들 간의 메시지 패싱에 대해 설명한다. 설계 표현은 객체 나무, 인스턴스 나무, 아키텍처, HDL 코드, 시뮬레이션 파형 등이 다양하게 존재한다. 객체 나무는 각 모듈들을 구성하는 요소들을 의미한다. 인스턴스 나무는 인스턴스들 간의 계층 구조를 나타낸다. 아키텍처는 계층구조의 논리회로도로서 인스턴스와 시그널 간의 관계, 그리고 모듈간의 계층구조를 나타낸다. HDL 코드는 소스 코드의 텍스트이고, 시뮬레이션 파형은 시뮬레이션 결과인 시그널들의 파형 논리값의 집합이다[8,9,11].

메시지 패싱이란 이러한 다양한 설계 표현들 간에 실시간으로 정보를 주고받는 기술이다. 메시지 패싱 함수는 그림 3과 같이 일 대 다수의 방향으로 메시지를 전달한다. 메시지 패싱 시스템에서 인스턴스 나무 창이 메시지 전달의 센터와 아비터 역할을 하고, 아키텍처 분석 창과 텍스트 편집 창 그리고 SPD 분석 창은 메시지의 발송자 혹은 수신자의 역할을 한다. 하나의 설계 표현에서 메시지를 받으면 나머지 모든 설계 표현으로 메시지를 전달하게 된다. 설계자의 명령에 의해 아비터는 임의의 수신자를 제거할 수 있다. 메시지를 받으면 수신자는 곧바로 상응되는 설계 객체를 하이라이트 시킨다. 대응되는 설계 객체가 없는 경우 수신자는 메시지를 무시하고 발송자에게 오류 메시지로 반응하지는 않는다[7].

메시지 발송 시스템에 대한 자세한 동작은 다음과 같다. 맨 먼저 파싱 파일이 HDL 컴파일러에 의해 생성된다. 그림 3에 나타난 바와 같이 메시지는 모듈 이름, 포트 혹은 내부 시그널 이름으로 구성된다. 텍스트 편집기와 인스턴스 나무, 혹은 아키텍쳐 분석기와 인스턴스 나무 간의 메시지에는 HDL 코드의 라인 번호가 추가되어야 한다. 이 모든 정보는 파싱 파일로부터 얻는다.

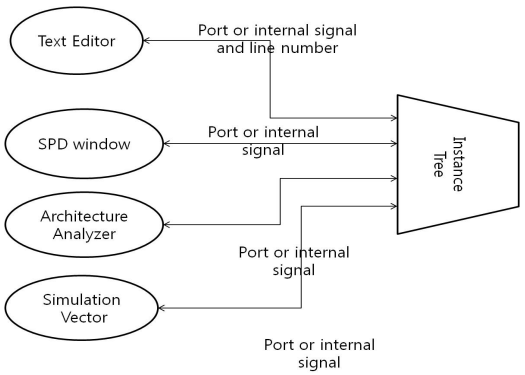


그림 3. 다양한 설계 표현간의 메시지 패싱
Fig. 3 Message passing among various design views

메시지 패싱은 발송자와 수신자의 종류에 따라 일 대 일 프로토콜로 동작한다. 설계 표현에 존재하는 모든 객체들은 시그널에 대한 모든 정보를 포함하기 때

문에 인코딩과 디코딩이 필요 없이 빠르게 메시지의 송수신이 이루어진다. 에러를 효율적으로 처리하기 위하여 수신자는 수신된 데이터가 자신의 로컬 데이터에 존재하는지를 검사하는 확인 과정을 거친다. 만약 데이터가 존재하면, 수신자는 대응되는 설계 표현을 하이라이트 한다. 그렇지 않으면 수신된 데이터는 무시된다(그림 4).

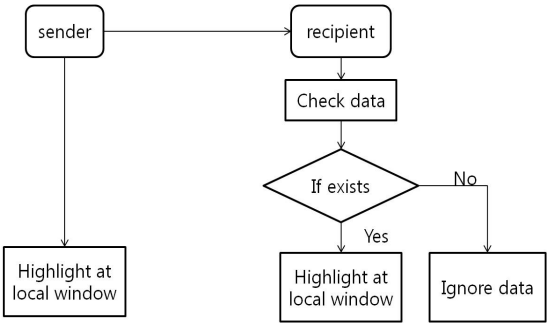


그림 4. 일 대 일 프로토콜 처리 흐름도
Fig. 4 End-to-end protocol flowchart

3.2. 배치배선 알고리즘

배치배선 알고리즘은 그림 5와 같이 다섯 단계로 진행된다. 개략적인 크기와 모양을 예측, 모듈 배치, 입출력 단자 배치, 시그널 배선 및 압축의 다섯 단계 이다.

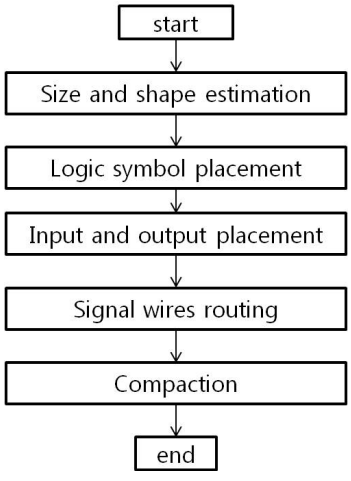


그림 5. 배치배선 알고리즘 흐름도
Fig. 5 Placement and routing algorithm flowchart

첫 번째 단계로 배치배선 알고리즘은 인스턴스 심볼의 크기와 모양을 결정한다. 일반적인 배치배선 알고리듬에서 심볼은 네모난 박스로 간주하고 입출력 단자의 최대 수로부터 심볼의 크기를 결정한다.

Archian은 심볼의 크기와 모양을 결정하는데 수동모드와 자동모드를 모두 지원한다. 자동모드에서는 일반 알고리즘과 동일하게 심볼의 크기와 모양을 결정한다. 그러나 수동모드에서는 설계자가 심볼의 모양과 크기, 그리고 단자들의 위치를 결정한다. 심볼 모양의 크기와 모양이 더욱 정확히 예측이 되어야 심볼의 배치 정보가 보다 구체적으로 결정된다. 설계자가 의도적으로 임의로 심볼의 크기와 단자 위치를 바꿈으로서 최적의 배치를 얻을 수 있다[4,10].

HDL 코드에 선언된 모든 인스턴스의 심볼의 크기와 모양이 결정되면 배치 알고리즘이 모든 심볼의 수평과 수직 레벨을 결정하고 배선 영역을 위해 인접한 심볼간의 최소 간격을 결정한다. 배치의 최종 결과는 레벨에 맞추어진 심볼의 (x, y) 좌표가 된다. 배치 과정은 다음과 같이 진행된다[12].

단계 1 : 모든 심볼의 수평과 수직 레벨은 1 로, 우상단 위치는 (0, 0)으로 초기화 한다.

단계 2 : $Symbol_{(n,1)}$의 우상단 좌표를 x_n,y_1 라고 가정한다. $Symbol_{(n,1)}$ 과 $Symbol_{(n+1,1)}$는 동일한 수평 레벨에 위치하므로 이들의 y 좌표는 동일하게 y_1 값을 갖는다. 그러므로 $Symbol_{(n+1,1)}$의 우상단 좌표는 아래 수식에 의해 계산된 (x_{n+1},y_1) 좌표 값을 갖는다.

$$x_{n+1} = x_n + x_{span} + x_{n-width} (x_1 = 0, n \geq 1) \qquad (1)$$

x_{span} 은 인접한 심볼 간의 최소 수평 간격을 의미한다. 그리고 $x_{n-width}$ 은 $Symbol_{(n,1)}$ 심볼의 폭을 의미한다.

단계 3 : $Symbol_{(n,n)}$ 심볼의 우상단 좌표를 x_n,y_n으로 가정한다. $Symbol_{(n,n)}$, $Symbol_{(n,n+1)}$ 은 동일한 수평 레벨을 가지므로 이들의 우상단 y 좌표값은 동일하게 x_1 값을 가진다.

$Symbol_{(n,n+1)}$ 심볼의 우상단 좌표는 아래 수식에 의해 (x_n,y_{n+1}) 좌표 값을 갖는다.

$$y_{n+1} = y_n + y_{span} + y_{n-height} (y_1 = 0, n \geq 1) \quad (2)$$

y_{span} 은 인접한 심볼 간의 최소 수직 간격이고, $y_{n-height}$ 는 $Symbol_{(n,n)}$ 심볼의 높이이다.

다음 단계로 입출력 단자의 위치를 결정하는데 단자가 연결된 심볼의 위치에 의해 좌표가 결정된다. 입출력 단자에 연결된 심볼 단자를 찾아낸 후에 입출력 단자의 y 값은 연결된 심볼 단자의 y 값과 동일하게 결정한다. 입출력 단자의 좌표가 결정되어 고정된다.

다음 단계는 내부 시그널의 배선이다. 배선 알고리즘은 채널 정의, 개략 배선, 교차점 배치, 그리고 상세 배선의 과정을 거친다. 채널 정의는 논리회로도의 구조를 사각형 영역으로 분할한다.

개략 배선은 각각의 시그널이 거치는 채널과 모듈의 주변 영역을 결정한다. 교차점 배치는 채널간의 교차점을 결정한다. 상세 배선은 각 채널 내에서의 배선을 먼저 수행한 후에 점진적으로 3개의 다른 채널 배선 알고리즘을 적용한다. 만약 배선에 실패하면 배선 알고리즘은 채널 영역을 확대한 후에 배선을 다시 시도하는데 배선이 성공할 때까지 반복하게 된다. 이때 파라미터에 의해 가변되는 버스의 버스 폭이 계산되어 심볼 주변에 배치된다.

마지막 단계로 배치배선 알고리즘은 전체 레이아웃을 압축함으로써 불필요한 영역을 제거한다[13].

Ⅳ. 실험 결과

실험 결과는 아키텍쳐 생성과 디버깅으로 나누어 설명한다. 아키텍처를 생성하려면 먼저 프로젝트가 정의되고 HDL 파일들이 프로젝트에 등록되어야 한다. 그러면 파싱 툴이 프로젝트에 등록된 HDL 파일들을 파싱한다. 객체 나무와 인스턴스 나무 구조는 그림 6과 같이 그래픽으로 생성된다. 설계자의 요구에 의해 각 모듈의 구조를 나타내는 논리회로도는 그림 7와 같이 생성된다. 모듈들은 계층구조를 갖기 때문에 계층 간의 연결선을 따라서 상위와 하위의 계층구조를 추적할 수 있다.

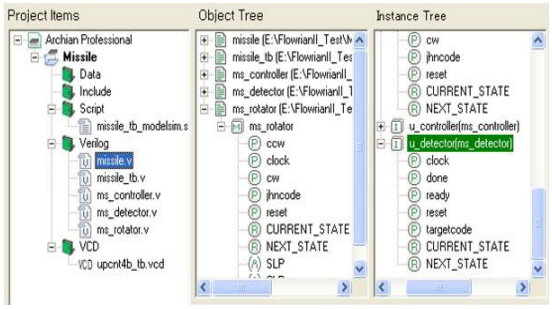


그림 6. 자동 생성된 객체 나무와 인스턴스 나무
Fig. 6 Generation of object tree and instance tree

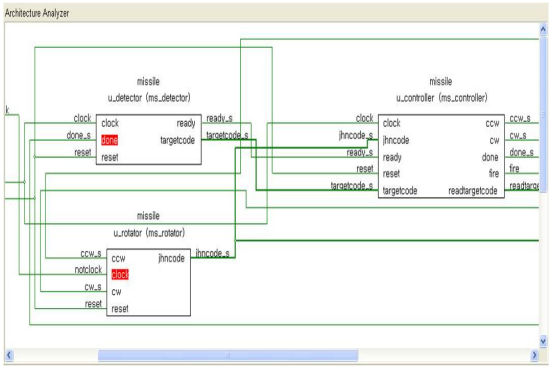


그림 7. 자동 생성된 아키텍처의 논리 블럭도
Fig. 7 Architecture schematic automatic generation

디버깅을 위해서 우선 기본적인 구문 오류는 HDL 컴파일러에 의해 제거되어야만 정상적인 파싱에 의해 구조가 생성될 수 있다. 어떤 한 설계 표현에서 하나의 설계 객체(시그널이나 단자 또는 인스턴스)가 선택되면 다른 설계 표현에 존재하는 대응되는 동일한 설계 객체가 자동으로 하이라이트 된다. 그림 8은 인스턴스 나무 구조에서 'fire' 시그널을 더블 클릭 했을 때 아키텍쳐 논리회로도에 나타난 'fire' 시그널 배선과 SPD 에 나타나 심볼에 연결된 'fire' 배선, 그리고 Verilog 코드에서 'fire' 가 정의된 구문이 모두 하이라이트 되어 있다. 만약 파형분석기가 활성화 되어 있으면 'fire' 파형 또한 하이라이트 된다.

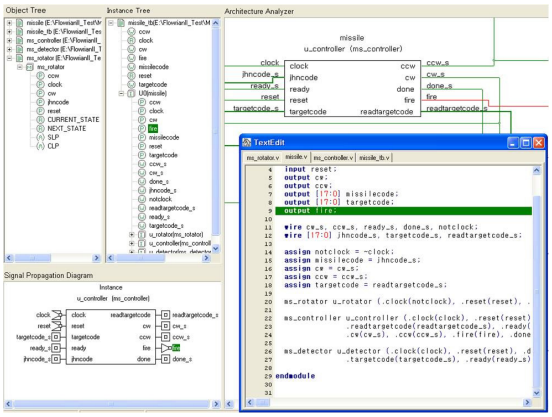


그림 8. 다른 설계 표현에 나타난 디버그 하이라이팅
Fig. 8 Debug highlighting on different design views

V. 결 론

본 논문에서 기술한 Archian은 Verilog HDL이나 VHDL 코드에 구현된 디지털 회로의 아키텍처를 효율적으로 분석하고 디버깅 할 수 있는 ECAD 소프트웨어이다. HDL 코드를 파싱함으로써 6가지의 다른 설계 표현이 그래픽으로 자동 생성되어 각각의 창에 나타난다. 이들은 객체 나무, 인스턴스 나무, SPD, 아키텍처 논리회로도, 시뮬레이션 파형 그리고 HDL 텍스트 코드이다. 구현된 설계 표현에서 한 객체를 더블 클릭하면 다음 설계 표현에 존재하는 동일한 객체를 찾아 하이라이팅 한다.

이러한 디버깅 기능은 설계자가 HDL 설계를 쉽게 이해할 뿐만 아니라 HDL 설계에 존재할 수 있는 버그를 효율적으로 찾을 수 있도록 돕는다. Archian은 (주)시스템 센트로이드에서 제품으로 마무리하는 작업이 진행 중이다. Archian은 앞으로 SystemVerilog 언어를 지원하도록 기능을 확장할 계획이다.

REFERENCES

[1] Hyung-Lae Roh, “Driving forces & the technological challenges for SOC development of tomorrow”, Signal Processing Systems, Aug. 2003.

[2] Yuan-Long Jeang, Liang-Bi Chen, Yi-Ting Chou, Hsin-Chia Su, “An embedded in-circuit emulator generator for SOC platform”, *Field-Programmable Technology (FPT),* Dec. 2003.

[3] Wooseung Yang, Moo-Kyeong Chung, “Current Status and Challenges of SoC Verification for Embedded Systems Market”, *IEEE International SOC Conference,* pp.213-216, Sept. 2003.

[4] Min Pan, Chu C, “IPR: An Integrated Placement and Routing Algorithm,” Design Automation Conference, pp. 59-62, 2007.

[5] Chih-Wen Hsueh, Tien-Fu Chen, “Development of architecture and software technologies in high-performance low-power SoC design”, *Embedded and Real-Time Computing Systems and Applications,* pp.475-480, Aug. 2005.

[6] Ruan, A.W, Li, C.Q, Song, Z.J., Chen, J., Deng, L.X., Hou, H.C. Liao, Y.B., “The third generation verification technology based SOC debugging", *Computational Problem-Solving (ICCP),* pp. 109-114, Oct. 2011.

[7] Park, S.-Y., Lee, J. and Hariri,S., “A multithreaded message-passing system for high performance distributed computing applications,” in *Distributed Computing Systems,* pp. 258-265, 1998.

[8] Choe, Y.K., Agrawal, D.P., Green, C.R. “A hierarchical message mechanism for distributed systems software”, *Distributed Computing Systems in the 1990s,* 1988.

[9] Jung, M., Biersack, E.W., "A component-based architecture for software communication systems", *Engineering of Computer Based Systems,* 2000.

[10] Jie Liang, Mao Lin Huang, “Highlighting in Information Visualization A Survey”, *Information Visualisation (IV),* July 2010.

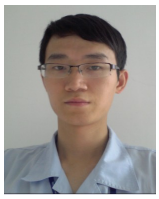
[11] Sherman, Alan T.(Alan Theodore). Springer-Verlag, "VLSI placement and routing: the PI project", *Springer,* 1989.

[12] Koschella, James J., "A placement/interconnect channel router: cutting your PI into slices", *B.S. thesis, Department of Computer Science,* MIT, 1981.

[13] Sechen, Carl, "VLSI placement and global routing using simulated annealing", *Kluwer Academic Publishers,* 1988.

문대철(Dai-tchul Moon)

약력
1987 고려대학교 전자공학과 공학박사
1994 North Carolina State Univ. 연구교수
2005 Minnesota State Univ. Duluth 객원교수
1984~현재 호서대 정보통신공학과 교수
※관심분야 : DSP 및 영상신호처리 칩 설계, SoC 설계, 임베디드 시스템 설계, VLSI 신호처리



Xie Cheng

약력
2010 Ningbo University 공학사
2011~현재 호서대학교 정보통신공학과 공학석사
※관심분야 : DSP 및 영상신호처리 칩 설계, SoC 설계, 임베디드 시스템 설계, EDA 응용 소프트웨어 개발

박인학(In-hag Park)

약력
1992 INPG Microelectronics 공학박사
1982~1985 한국전자기술연구소 연구원
1985~2000 한국전자통신연구소(ETRI) 책임연구원(실장)
2000~ 현재 (주)시스템센트로이드 대표이사
※관심분야 : 웹기반 CAD 시스템 개발, EDA 응용 소프트웨어 개발, 임베디드 시스템 설계, 반도체 IP 설계