인터페이스 자동 생성을 위한 IP 재사용 환경 및 사례 연구

*이강희, *조영철, *서동관, *최기영, **정의영
*서울대학교 전기컴퓨터 공학부 설계 자동화 연구실, **삼성전자 LSI 사업부 CAE 센터
전화 : 02-880-1304 (ext. 311)

Environment of IP Reuse for Automatic Interface Generation and Case Study

*Ganghee Lee, *Youngchul Cho, *Dongkwan Suh, *Kiyoung Choi, **Eui-Young Chung

*Design Automation Lab. EECS of Seoul National Univ.,
**CAE Center, Samsung Electronics Co., Ltd.
E-mail : berean97@poppy.snu.ac.kr

Abstract

In the design of embedded SoCs, which is getting more and more complex, design reuse is known to be a good strategy to improve the designer's productivity significantly. Therefore, new design methodologies such as IP based design or Platform based design are becoming popular. In such design methodologies, IP integration - interconnection of IPs through on-chip communication network - is the main job of SoC design. And also important is IP authoring, which is the process of designing reusable IPs.

In this paper, we focus on the automatic interface generation environment for making IPs reusable. We present an approach of building a library of such IPs with automatically generated interfaces. We also present a case study to demonstrate how we can use the proposed environment.

I. 서론

SoC의 복잡도 및 제조 기술이 빠르게 증가/향상하고 있으나, 설계자들의 설계 생산성의 증가가 상대적으로 느려 설계 생산성 격차 문제가 심각하다. 이에 대한 극복 방안으로 설계 재사용에 대한 필요성이 절실해지고 있다[1]. 설계 재사용의 방법으로 많이 사용되는 것이 IP 기반 설계와 platform 기반 설계이다. 이러한 설계는 주어진 IP와 on-chip 통신 네트워크를 연결하는 일이 설계의 상당 부분을 차지한다. IP와 on-chip 통신 네트워크의 연결을 손쉽게 하는 방법은 통신 래퍼를 사용하는 것이다. 그러나 이제까지의 래퍼의 설계는 대체로 설계자가 수동적으로 해왔다. 이는 시간이 많이 걸리고 오류가 생기기 쉬운 작업들이다. 또한 이와 같이 설계자가 직접 래퍼를 설계하는 경우에는 정형화된 래퍼 구조가 없이 각 설계자의 의도에 따라 래퍼는 서로 다른 구조를 가지게 될 것이고, 이는 또다시 재사용의 문제를 어렵게 만드는 요인이 된다.

이러한 것을 해결하기 위해서 VSIA(Virtual Socket Interface Alliance)에서는 래퍼의 표준화 방안으로 모듈 어댑터나 채널 어댑터의 개념을 도입하였다. 여기서는 모든 virtual component(VC)들이 virtual component interface (VCI)를 가지고 있어서, master 쪽의 VCI initiator와 slave 쪽의 VCI target이 각각 master의 프로토콜을 VCI 통신 프로토콜로 변환하고 다시 slave의 프로토콜로 변환하는 작업을 하게 된다[6].

본 연구실에서도 이와 비슷한 개념의 포트 어댑터를 도입하여 래퍼를 자동 생성하는 도구를 개발하였다[7]. 특히 래퍼에 내부 버퍼를 두거나, 래퍼를 공유하는 등의 방법을 제시하였다. 이 래퍼 생성 도구(" RTAGen" 이라고 부름)는 최종 SoC 구조를 생성하기 위한 라이브러리가 모두 갖추어 졌다고 보고 생성을 시작하는데, SoC 설계를 하는 데 있어서 라이브러리 구축 또한 무시 못할 부분이다. 따라서 본 논문은 [7]에서 제시한 RTAGen이라는 도구를 활용하기 위해서 LibGen이라는 도구를 제시하고 이를 통해서 어떻게 라이브러리 환경을 구축 할 수 있는 가에 대해서 자세히 설명한다. 또한 이러한 환경 구축을 통해서 실제 예제를 다양한 SoC 구조로 구현하고 비교해 보았다.

이 논문은 다음과 같이 구성된다. 우선 2절에서는 본 연구실에서 개발 중인 SoC 설계를 위한 전체 flow에

대해서 살펴보고 특히 하드웨어 라이브러리에 기반을 둔 IP 통합 설계에 대해서 설명한다. 3절에서는 이 논문의 모체가 되는 [7]의 RTAGen에 대해서 간단히 살펴본다. 4절에서는 이러한 RTAGen에 사용될 라이브러리를 생성하기 위한 환경에 대해서 살펴보고 LibGen이라는 도구를 사용한 라이브러리 구축에 대해서 살펴본다. 5절에서는 이러한 두 가지 도구를 사용한 설계 예제를 보이고 6절에서 결론을 맺도록 하겠다.

II. SoC 설계 과정 및 IP 통합

1. SoC 설계 과정

그림 1의 설계 과정은 미국 UCI의 SpecC 그룹에서 연구한 시스템 수준에서의 통신 refinement[2]를 기반으로 하였으며 아직 개발 중에 있다. Kahn Process Network(KPN)을 기반으로 하여 SystemC로 기술한 예제를 입력으로 받아 본 연구실에서 개발한 분할 방법[5]을 통해 HMDG(Hierarchical Module Dependency Graph)를 생성한다. [5]의 논문은 C 코드를 입력으로 받아 profiling을 하고 성능을 예측하여 HW/SW 분할을 해주는 과정을 제시하고 있다. 그러나 이 논문에서는 C를 그 시작으로 하기 때문에 예제의 병렬성을 살리기가 힘들다. 또한 simulation을 하기 위해서는 상대적으로 속도가 느린 RTL cosimulation을 해야 했다. 그러한 이유로 위의 SoC 설계 flow는 병렬성을 잘 살린 KPN의 SystemC 표현을 입력으로 사용한다. 또한 RTL cosimulation을 하기 이전에 mapping 단계와 통신 refinement 과정 중에 TLM을 이용한 simulation을 하기 때문에 보다 빠른 설계 공간 탐색을 할 수 있다.

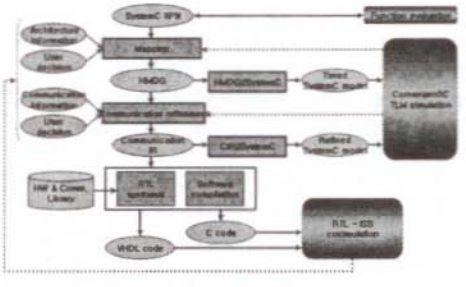

그림 1. SoC design flow.

이렇게 생성된 HMDG는 behavior에 대해서는 timed 모델이지만 통신에 대해서는 아직 추상화 수준이 높기 때문에 통신 refinement를 통해서 cycle-accurate한 CIR(Communication Intermediate Representation)을 생성한다. 이 모델은 behavior 뿐만 아니라 통신에서도 추상화 수준이 구체적이기 때문에 정확한 simulation 결과를 얻을 수 있다. Simulation의 결과는 mapping과 통신 refinement에 feedback을 주고 그 결과 주어진 constraint를 만족하는 SoC 구조를 찾을 때까지 같은 과정을 반복하게 된다.

위의 과정을 통해 SoC 구조가 정해지게 되면 refine된 SystemC 코드는 "RTL synthesis"와 "SW compilation" 단계를 거쳐서 VHDL로 기술된 최종 RTL 코드와 C로 기술된 SW 실행 코드로 바뀐다. 여기에서 RTL synthesis 과정에 IP 통합 과정이 포함된다.

2. IP 통합

그림 2는 이 논문에서 제안하는 SoC 설계를 위한 IP 통합 과정을 보여준다. 그림 1의 SoC 설계 과정으로부터 결정된 SoC 구조는 그림 3과 같이 XML로 표현 가능하다. XML은 유연성과 확장성이 뛰어나기 때문에 embedded 분야에서도 널리 사용되는 언어이다. 이 XML은 IP 통합 과정에 해당하는 RTL synthesis의 입력으로 들어간다. 이 논문에서는 특정 시스템 언어의 사용에 국한하지 않는 IP 통합 방법을 개발하기 위해 XML을 입력으로 받아 중간 표현인 SAD로 변환을 하고 이를 토대로 해서 라이브러리로부터 RTL architecture를 VHDL로 생성하는 IP 통합 과정에 대해서 설명한다.

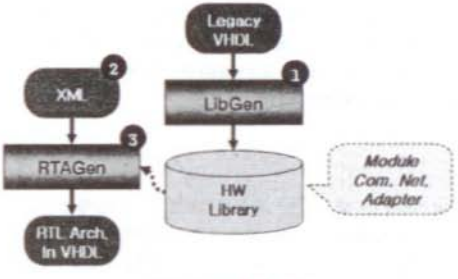

그림 2. IP 통합 과정.

그림 2에서 보듯이 SoC 설계를 위한 IP 통합과정은 3단계로 나뉘어져 있다. 첫 번째 단계는 라이브러리 생성기(LibGen)를 사용하여 설계자가 만든 하드웨어 IP를 라이브러리에 추가하는 단계이다. 두 번째 단계는 XML을 사용하여 설계자가 SoC 구조를 설계하는

단계이다. 마지막 단계는 SoC 구조 생성기(RTAGen)를 이용하여, 최종 SoC 구조를 VHDL로 자동 생성을 하는 단계이다.

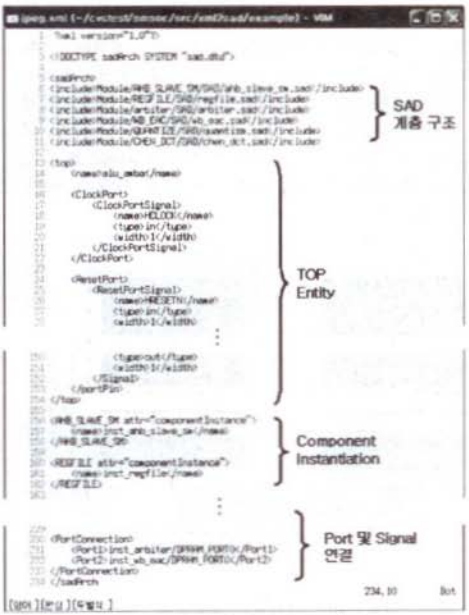

그림 3. XML 입력 (JPEG) 예제.

하드웨어 라이브러리는 IP 라이브러리, 통신 네트워크 라이브러리, IP와 통신 네트워크를 연결해 주는 포트 어댑터 라이브러리의 세 가지 종류로 나뉜다. 라이브러리 생성기(LibGen)는 설계자가 설계한 IP 또는 통신 네트워크를 입력으로 받아 하드웨어 라이브러리를 생성하고 설계자가 포트 어댑터를 설계하기 위한 환경을 제공한다. SoC 구조 자동 생성기(RTAGen)는 통신 래퍼 생성과 최종 SoC 구조 생성이라는 두 가지로 나뉜다. 통신 래퍼 생성은 XML을 입력으로 받아들여서, IP와 연결되는 포트 어댑터(port adapter)와 통신 네트워크와 연결되는 포트 어댑터를 포함하는 통신 래퍼를 생성하게 된다. 여기에서는 입력으로 받아들인 XML에서 추출한 정보를 이용하여 하드웨어 라이브러리에 있는 포트 어댑터를 선택, instantiation하고, 래퍼의 내부 버스 구조를 자동으로 구성하여 VHDL로 기술된 통신 래퍼를 생성해 준다. 최종 SoC 구조 생성에서는 라이브러리로부터 SoC 설계에 필요한 IP들과 통신 네트워크들을 가지고 와서(instantiation) 래퍼 생성기가 생성해준 통신 래퍼로 연결하여 그림 4와 같은 SoC 구조를 생성한다.

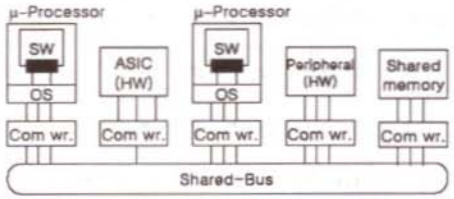

그림 4. Shared-bus architecture .

III. SoC 구조 생성기 (RTAGen)

본 연구실에서 개발한 SoC 구조 생성기[7]는 주어진 라이브러리로부터 그림 4와 같이 통신 래퍼를 사용하여 프로세서와 같은 IP들을 통신 네트워크에 연결해 주는 역할을 한다. 이 과정에서 사용되는 통신 래퍼는 IP와 연결되는 포트 어댑터(port adapter), 통신 네트워크와 연결되는 포트 어댑터, 그리고 래퍼 내부에서 포트 어댑터를 서로 연결해 주는 내부 버스 구조로 이루어져 있다.

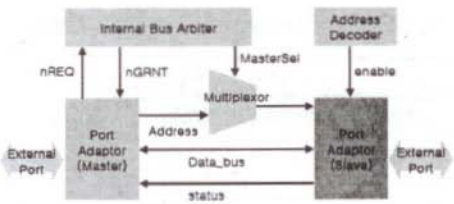

그림 5. 래퍼의 구조.

포트 어댑터는 그림 5와 같이 외부 포트의 프로토콜을 내부 포트의 프로토콜로 바꾸어 주는 역할을 한다. 포트 어댑터의 외부 포트는 IP 또는 네트 워크와 연결이 되며 포트의 연결 타입(access type)에 따라 master 포트 어댑터와 slave 포트 어댑터로 나뉜다. 이러한 어댑터 모델은 IP의 통신 프로토콜과 통신 네트워크의 통신 프로토콜을 변환하는 역할과 함께, IP와 통신 네트워크간의 일대일(1-to-1) 통신뿐만 아니라, 다대다(many-to-many) 통신까지 가능하게 한다. 예를 들어, IP에 통신을 위한 포트가 여러 개 연결될 수 있을 때(예, multi-port memory, DSP modules, etc.) 혹은 IP가 2개 이상의 통신 네트워크를 통해 통신을 할 때에도 통신 래퍼의 생성을 자동화할 수 있다.

그림 5에서 볼 수 있듯이 래퍼의 내부 버스는 우선 순위 기반의 공유 버스로 구현되어 있다. 즉, " Internal Bus Arbiter" 가 있어서 Master 포트 어댑터들로부터 *Request*를 받아서 우선 순위에 따라서 *Grant* 신호를 주고 " Address Decoder" 에 의해서 Slave 포트 어댑터를 enable 한다.

Ⅳ. 라이브러리 생성 환경

위에서 설명한 SoC 구조 생성기(RTAGen)는 라이브러리를 기반으로 한다. 그렇기 때문에 이러한 도구를 사용하기 위해서는 라이브러리를 먼저 구성해 주어야 할 필요가 있다. 이를 위해 이 논문에서는 LibGen이라는 라이브러리 생성기를 GUI를 사용하여 개발하였고, 이를 통해서 설계자의 IP를 라이브러리에 등록하는 과정을 자동화 하였다.
그림 2에서 볼 수 있듯이 라이브러리는 모듈과 통신 네트워크 그리고 어댑터의 세가지로 구성이 된다. 이 중에서 설계자는 IP 모듈과 통신 네트워크를 라이브러리에 등록할 수 있으며 어댑터는 등록된 IP나 통신 네트워크에 따라서 라이브러리 생성기가 자동으로 생성해 준다. 그림 6은 라이브러리 생성기(LibGen)에 IP를 추가하는 예제를 보여준다.

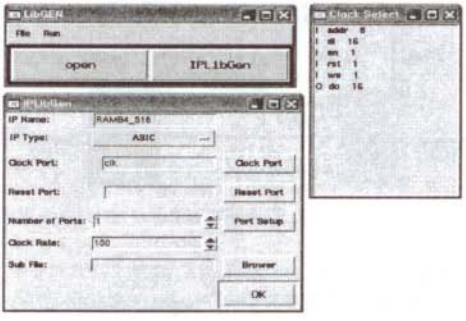

그림 6. 라이브러리리 생성기(LibGen) 예제.

LibGen은 다음과 같은 일을 한다. Legacy IP에 해당하는 VHDL 코드를 입력으로 받아 코드의 entity 부분을 파싱하여 라이브러리의 중간 저장 형태인 SAD의 일부를 생성한다. 여기서 SAD[7]는 System Architecture Description을 의미하는 것으로 Architecture의 구조를 표현한 라이브러리 포맷이다. VHDL을 파싱하여 알 수 없는 부분은 GUI환경을 통해 설계자의 입력을 받아 SAD를 완성할 수 있다. 생성된 SAD는 VHDL과 함께 라이브러리에 저장된다.
VHDL을 파싱하여 알 수 없는 정보에는 모듈의 타입, 네트워크의 타입, 포트 정보, 입력 clock rate 등이 있다. 모듈의 타입에는 CPU, DSP, HW 등이 있고 네트워크의 타입에는 priority-based network, TDMA network, dedicated network 등이 있다.
라이브러리는 아래의 그림 7에서처럼 점선으로 감싼 부분이 한꺼번에 저장되는 형태를 취한다. 즉, 등록하려는 IP와 포트 어댑터가 그룹으로 묶여서 저장이 되는 것이다. 이와 같이 되면 해당 IP의 인터페이스는 이 연구에서 제안하는 포트 어댑터의 프로토콜을 갖게 된다. 그러므로 이 인터페이스를 갖는 어떠한 IP라도 RTAGen을 통해서 바로 연결이 될 수 있는 것이다.

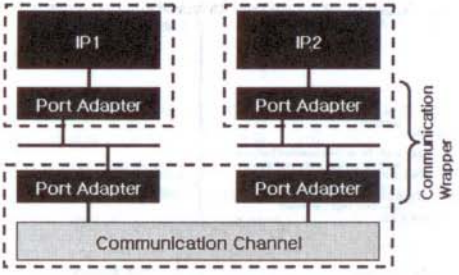

그림 7. 라이브러리 저장 형태.

이러한 방법으로 래퍼를 생성하게 되면 래퍼를 지나면서 겪는 지연이 생길 수가 있다. 이에 대한 최적화 연구는 [7]에서 이미 언급된 내용이다.
현재, 라이브러리에 IP를 등록하는 과정 중 통신 네트워크를 등록하는 과정은 자동화가 되어 있지 않다. 그 이유는 통신 네트워크의 경우에는 연결되려는 IP의 개수에 따라서 포트의 수가 달라지는 등, SoC 구조에 따라서 수정되어야 할 configuration이 다양하기 때문이다. 그래서 널리 사용되는 통신 네트워크인 AMBA[3], uNetwork[4] 등은 라이브러리에 미리 수동으로 등록하였다. 또한 ARM 프로세서와 같이 많이 사용되는 IP 역시 라이브러리에 미리 등록하였다. 설계자는 원한다면 이러한 라이브러리를 사용하여 최종 SoC 구조를 생성할 수 있다.

Ⅴ. 설계 예제

위에서 언급한 RTAGen과 LibGen 도구를 검증하기 위해 이 논문에서는 JPEG encoder를 예제로 사용하였다. 본 연구실에서 개발한 분할 도구[5]를 사용하여 JPEG을 분할 하였더니, DCT를 HW로 보내고 SW는 ARM 프로세서를 사용하는 것으로 결정되었다.

HW IP를 위해, ChenDCT 알고리즘을 VHDL로 설계한 기존 legacy IP를 LibGen을 사용해서 라이브러리에 등록하였다. ARM 프로세서는 seamless CVE에서 제공되는 ARM926 ISS를 사용하였다. 통신 네트워크로는 ARM 프로세서와 함께 가장 널리 사용되는 AMBA AHB를 사용하였다.

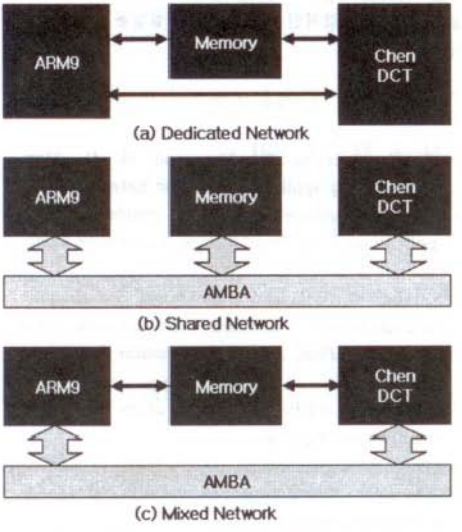

그림 8. 통신 변화에 따른 3가지 SoC 구조.

JPEG encoder를 실행하기 위한 이러한 3가지 IP들을 라이브러리로 가지고, RTAGen을 통해 그림 8과 같이 통신 변화에 따라 3가지 SoC 구조를 생성하였다. (a)는 프로세서와 IP들이 dedicated bus를 통해 각각 일대일로 연결된 구조이고, (b)는 모든 IP들이 공유버스를 통해서 연결된 구조이다. (c)는 통신 량이 많을 것으로 생각되는 부분은 직접 일대일로 연결하고 나머지 부분에 대해서는 공유 버스를 사용해서 연결한 모양을 보여준다.

그림 8 (b)의 공유 버스를 사용한 SoC 구조 내부를 좀 더 자세히 그리면 그림 9와 같다. ARM 프로세서는 AMBA와 인터페이스가 동일하기 때문에 버스에 바로 연결될 수 있다. 그러나 메모리나 Chen_DCT 같은 IP의 경우에는 직접 연결되지 못하고 래퍼를 통해 연결되었다. 특히 Chen_DCT의 경우에는 두 개의 래퍼가 함께 사용된 구조를 보여주고 있는데 이는 이 IP가 DCT 수행을 위한 제어는 ARM 프로세서로부터 받고 필요한 데이터는 메모리로부터 능동적으로 가져오는 구조로 되어있기 때문이다. 이에 따라서 포트는 master가 되기도 하고 slave가 되기도 해야 하기 때문에, master 포트 하나 slave 포트 하나로 구성되어 결국 2개의 래퍼로 연결되었다. 결과적으로, AMBA 버스에 2개의 master와 2개의 slave가 붙는 형태가 되었다.

각각의 래퍼는 VHDL 코드로 약 400라인으로 설계되었다. 이를 Xilinx ISE 6.1을 사용하고 VERTEX FPGA를 target으로 하여 synthesis를 하였을 때 전체 래퍼는 2200 gates의 area를 필요로 하였다. 이는 DCT(38000 gates)같은 IP의 area에 비하면 비교적 작은 수준이다. 이 래퍼를 지나면서 소요되는 사이클 단위의 지연은 없는데 이는 하나의 IP 포트당 하나의 네트워크 포트로 연결되는 구조로 설계 되었기 때문이다. 만약 많은 수의 포트 어댑터가 하나의 래퍼에 포함되고 내부에 버퍼 등을 가지게 된다면 지연이 발생할 수 있는데 이에 대한 내용은 [7]에서 다루었으므로 여기서는 생략한다.

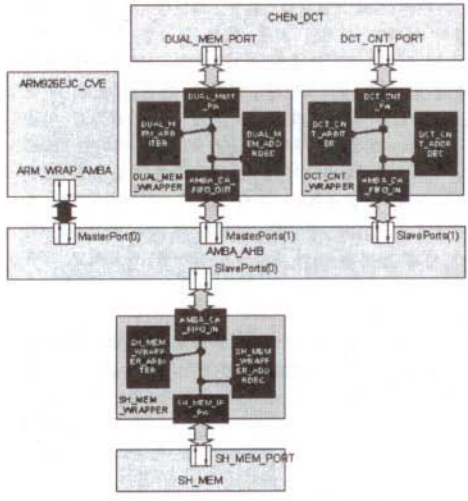

그림 9. 공유 버스와 래퍼를 이용한 SoC 구조.

각 IP에 연결되는 포트 어댑터의 behavior는 LibGen을 사용하여 직접 기술하였고 arbiter와 decoder를 내부 버스로 연결하여 전체 wrapper를 구성하는 것은 RTAGen을 사용하였다. 포트 어댑터를 설계할 때 IP의 인터페이스를 분석하는 것 만으로는 VHDL의 behavior를 자동으로 생성할 수가 없다. 이는 설계자가 직접 해야 하는데, 설계 경험이 있는 전문가라면 LibGen의 도움을 받아서 수 시간 안에 설계할 수 있다. 만약 VHDL로 약 400라인 정도인 래퍼를 이러한

도움 없이 설계한다면 수십 시간이 소요되는 작업이다. 또한 포트 어댑터를 한번 설계하고 나면 RTAGen을 사용하여 수초 안에 그림 8과 같은 여러 가지 구조를 새롭게 생성할 수 있다. 이는 SoC 구조가 바뀔 때마다 매번 래퍼를 새롭게 설계해야 하는 수십 시간의 작업을 한번 더 줄이는 결과를 가져올 수 있다.

결론적으로, 설계자는 RTAGen과 LibGen을 사용하여 빠른 시간 안에 SoC 구조를 설계할 수 있고 이를 통해 설계 생산성을 향상 시킬 수 있을 것이다. 또한 이러한 도구들을 통해서 설계된 SoC 구조는 정형화된 래퍼 구조를 가지게 되므로 다른 사용자에 의해서도 쉽게 이해될 수 있고 이는 유지 보수의 측면에서 또 다른 설계 생산성 향상을 가져 올 수 있다.

VI. 결론

본 논문은 SoC 설계를 위한 흐름을 제시하고, 이 과정의 마지막 단계인 최종 합성 가능한 VHDL 코드 생성 단계에서 IP 통합을 하기 위한 라이브러리 환경 구축에 대해서 설명하였다. 이러한 환경 구축을 위해 LibGen이라는 라이브러리 생성 도구를 만들었고 이를 통해 라이브러리를 구축한 후 RTAGen이라는 SoC 구조 생성기를 통해 JPEG 예제에 대해서 IP들과 통신 네트워크를 연결해주는 통신 래퍼 및 SoC 구조를 다양하게 VHDL로 생성해 보았다. 본 논문에서 제시한 SoC 설계 방법과 위의 두 가지 도구를 사용하면 오류 없이 빠른 시간 내에 IP 통합을 할 수 있는 환경을 갖출 수 있다.

아직까지 SoC 설계를 위한 이 과정은 100 % 자동화가 이루어지지 못했다. 4절에서 설명한 것과 같이 GUI를 통해서 사용자가 직접 수정해 주어야 하는 부분들이 존재한다. 앞으로는 이러한 부분들을 조금씩 고쳐서 사용자가 수고해야 하는 부분들을 줄일 필요가 있다.

또한 현재 예제로 사용한 JPEG은 래퍼의 구성에서 포트 어댑터가 모듈과 통신 네트워크에 각 하나씩만 연결된 구조를 가지기 때문에 다양한 성능과 면적 비교를 할 수가 없었다. 현재 더 복잡한 예제를 적용하여 IP에 2개 이상의 포트 어댑터가 사용 되고 이 포트 어댑터들이 하나의 래퍼로 구성되는 복잡한 구조를 RTAGen을 통해서 생성하고, 합성을 하여 성능과 면적의 직접적인 비교를 하는 실험을 진행 하고 있다.

참고문헌

[1] S. Vercauteren, B. Lin, and H. D. Man. "Constructing application specific heterogeneous embedded architectures from custom HW/SW applications," in *Proc. Design Automation Conf.*, June 1996.

[2] S. Abdi, D. Shin, and D.D. Gajski, " Automatic communication refinement for system level design," in *Proc. Design Automation Conf.*, June 2003.

[3] ARM Inc. *AMBATM specification (rev 2.0)*, http://www.arm.com/

[4] Sonics, Inc. *Sonics μNetworks. Technical Overview. Networking Technology Datasheets*, http://www.sonicsinc.com/.

[5] 한기성, 안용진, 이제형, 곽재화, 유준희, 최기영, "자동화 툴을 이용한 HW/SW 분할 및 사례 연구", *대한전자공학회 학술지*, May 2003.

[6] On-chip bus development working group, *Virtual Component Interface Standard (OCB 2 2.0)*, http://www.vsi.org/

[7] 소미영, 조영철, 이강희, 서동관, 최기영, " SoC 설계에서 하드웨어 통신 래퍼 자동 생성", *대한전자공학회 학술지*, October 2002.