논문 2009-46TC-6-8

SCA에서 C++/VHDL 구현 독립성을 보장하기 위한 미들웨어의 확장

(The Middleware Extension for guaranteeing the Implementation-Independency between C++ and VHDL)

배 명 남*, 이 병 복*, 박 애 순**, 이 인 환*, 김 내 수***

(Myungnam Bae, Byungbog Lee, Aesoon Park, Inhwan Lee, and Naesoo Kim)

요 약

본 논문에서는 SCA 기반 무선통신환경에 적합한 코바 미들웨어의 확장에 대해 기술한다. 제안한 미들웨어 확장은 모든 컴포넌트가 컴포넌트의 구현 방식에 대한 고려없이 상호 연동될 수 있도록 보장하며 기존 방식에 비해 성능 개선이 가능하다. 이러한 미들웨어 확장은 HAO와 IDL2VHDL 컴파일러, 그리고 ORBit의 개선을 포함한다. HAO는 FPGA 환경을 고려하여 로직수준에서 개발된 ORB이며, FPGA의 특성에 따라 일부 기능은 제한되었다. 추가로, IDL2VHDL 컴파일러는 CORBA IDL로부터 하드웨어 기술언어인 VHDL로의 매핑과 추가의 절차들을 제공한다. 마지막으로, ORBit는 HAO와 직접 연동할 수 있도록 개선된 GPP상의 코바 ORB이다.

Abstract

In this paper, we propose a CORBA middleware extension which is suitable to SCA based communication environment. The extensions guarantee the components to interconnect others without consideration about its implementation way and enables the developers to easily achieve the performance improvements in comparison to the existing methodology. This extension includes the HAO, the IDL2VHDL compiler, and the improvement of ORBit. The HAO is ORB implemented in logic level and is limited the some function according to the characteristic of FPGA. In addition, the IDL2VHDL compiler provides the mapping from CORBA IDL to VHDL, the VHSIC hardware description language, and the additional procedures for processing the component. Finally, the improved ORBit, CORBA ORB on GPP, can be direct connecting with the HAO on FPGA.

Keywords : CORBA Middleware, HAO, SCA, SDR, FPGA

* 정회원, 한국전자통신연구원 USN응용기술연구팀
(USN-based Application Technology Research Team, ETRI)
** 정회원, 한국전자통신연구원 차세대이동단말연구팀
(Next Generation Mobile Terminal Research Team, ETRI)
*** 정회원, 한국전자통신연구원 USN기반기술연구팀
(USN Basic Technology Research Team, ETRI)
※ 본 논문은 지식경제부/IITA의 IT신성장동력핵심기술개발사업(2005-S-404-33), 국토해양부 지능형국토정보기술혁신사업(06국토정보C01)의 연구비지원에 의해 수행되었습니다.
접수일자: 2009년1월2일, 수정완료일: 2009년6월17일

I. 서 론

무선통신환경에서, 하나의 플랫폼에 여러 무선체계에 대해 재사용, 상호운용, 그리고 재구성을 보장하기 위한 객체지향 분산 프레임워크 기술이 제안되었다. 대표적인 기술인 SCA (Software Communication Architecture)는 JTRS(Joint Tactical Radio System) JPEO(Joint Program Executive Office)에 의해 단일

플랫폼에서 여러 무선통신 체계간의 상호 연동성을 제공하는 유연한 소프트웨어 구조 개발을 목표로 제안되었다[1]. SCA는 플랫폼에 소프트웨어와 하드웨어의 이식성을 보장하고, 여러 주파수 밴드를 포함한 새로운 기술적 추가, 소프트웨어의 재사용성 및 확장성, 플랫폼의 재구성 능력을 보장할 수 있는 통신 구조이다. 이를 통해, 새로운 무선 환경의 확장과 추가 시에 비용과 시간을 절감할 수 있고, 새로운 기술과 쉽게 융합할 수 있어 진화된 시스템과의 호환성을 제공할 수 있다는 장점이 있다[2].

한편, 단말 구성에 있어서, 프로토콜 스택과 같은 소프트웨어 컴포넌트들이 탑재되는 범용 목적의 프로세서(이하 GPP, General Purpose Processor)가 빠르게 개선되고 있으나, 단말내 특수 목적 소프트웨어 컴포넌트들을 로직으로 구현하려는 시도가 점차 확대되고 있다. FPGA(Field Programmable Gate Array)에서 로직 수준의 구현은 엄격한 실시간 처리가 가능하며, 다수의 복잡한 알고리즘들을 병행해서 실행할 수 있기 때문이다[3]. 현재, SCA는 소프트웨어 컴포넌트와 FPGA에서 로직 수준에서 구현된 컴포넌트(이하, 하드웨어 컴포넌트)들과 미들웨어를 사용하지 않고 개별적이고 직접적으로 연동하는 방식을 권고하고 있다. 즉, SCA는 FPGA와 같은 하드웨어적 구현에 대한 독립성을 보장하지 못한다. 본 논문은 통신 시스템에 있어, 기존 SCA에 추가로 하드웨어적 구현에 독립성을 보장하도록 확장한 미들웨어 구조에 대해 기술한다.

이 구조에서, C++ 혹은 VHDL간의 구현 방식에 대한 상호 독립성과 운용성을 보장하기 위해, 1) 하드웨어 컴포넌트 개발 과정에서 칩(FPGA나 DSP)이나 보드 의존적인 부분들을 분리하여 포팅 작업을 최소화할 수 있는 추상화 체계와 2) 동시에 구현 방식(프로그래밍 코드 혹은 디지털 로직)에 대한 고려 없이 모든 컴포넌트들 간의 데이터 교환 및 변환 체계를 제공하여야 한다.

본 논문의 구성은 다음과 같다. Ⅱ장은 본 논문과 관련 주제들을 개략적으로 소개한다. Ⅲ장은 SCA가 하드웨어적 구현에 독립성을 갖도록 확장한 환경과 체계에 대해 기술한다. Ⅳ장에서는 IDL로부터 유도되는 컴포넌트의 구성에 대해 Ⅴ장에서는 이에 적용된 소프트웨어 ORB와 하드웨어 ORB에 대해 기술하며, Ⅵ장에서는 검증을 위한 시연과 결과를 기술한다. 마지막으로, 결론 및 향후 과제에 대해 기술한다.

Ⅱ. 관련 연구

SCA에서 무선통신 체계간의 독립성과 상호운용성은 미들웨어를 포함하는 코어 프레임워크 계층의 제공과 이의 관리에 따른 어플리케이션간 의존성 제거를 통해 달성될 수 있다. 미들웨어 계층은 공개적이고 상업적으로 활용되고 있는 코바[4], XML(eXtensible Markup Language)[5], 그리고 POSIX(Portable Operating System Interface)[6]를 사용한다. 코어 프레임워크는 미들웨어 계층을 바탕으로 잘 통제되고 안정된 방법으로 여러 무선 체계에 따른 단말의 환경 구성, 장치에 적재, 구성과 배포 등을 수행한다. 이를 통해, 주파수 밴드 변경, 진화된 통신 프로토콜 및 무선체계간 절체를 달성할 수 있다.

SCA의 미들웨어인 코바는 컴포넌트로부터 통신체계, 개발언어, 운영체제 등과 같은 환경 의존적인 부분을 분리하고, 분산 환경에서 서버 컴포넌트와 클라이언트 컴포넌트간의 위치 독립성 및 통신 방식에 대한 추상화를 제공한다. 이를 통해, 클라이언트 컴포넌트는 코바를 통해 서버 컴포넌트가 존재하는 칩(프로세서, FPGA 등)이나 물리적인 위치에 무관하게 통신할 수 있다.

현재, SCA는 소프트웨어 컴포넌트들간의 상호운용성에 대해서만 고려할 뿐 로직 수준에서 구현된 하드웨어 컴포넌트의 지원방식에 대해 고려하지 않고 있다[3,7]. 이는 SCA에서 채택된 코바 미들웨어가 FPGA와 같은 하드웨어에 대한 추상화를 보장하지 못한다는 한계로 인한 제약이다. 부분적으로, 로직 구현을 수용하기 위해[8]에서는 MicroBlaze와 같은 FPGA상의 소프트 코어에 운영체제와 미들웨어의 포팅을 통해 하드웨어 컴포넌트와 연동하는 방식도 고려되고 있지만, 결국, 하드웨어 컴포넌트에 대한 추가의 노드 제공이라는 점에서 성능과 복잡성 문제가 있고 또한 소프트 프로세서에 대한 비용과 규모 측면의 부담이 크다는 단점이 있어 적극 활용되진 않고 있다.

또한, [9]에서는 GPP상의 클라이언트 컴포넌트가 FPGA상의 하드웨어 로직과 통신하기 위해 어댑터를 사용한다. 이 방식에서는, GPP에 하드웨어 로직에 대한 접근을 위해 별도로 어댑터를 두고, 클라이언트 컴포넌트는 코바를 통해 이 어댑터에 요청하고 응답을 받는다. 어댑터는 GPIO(General Purpose Input/Output) 할당과 같은 특정 하드웨어 의존적 구현을 직접 사용하여 장치 드라이버 형태로 구현되며, 수신한 요청은 시그널 수준으로 하드웨어 로직에 전달하고 그 처리 결과를 받

는다. 어댑터는 이 결과를 다시 코바 통신 체계를 통해 클라이언트 컴포넌트에 제공한다. 이러한 방식의 단점은 하드웨어 자원을 직접 사용함으로 인해 단말의 하드웨어 구성에 의존성을 갖게 된다는 것이다. 결국, 하드웨어 환경의 변경에 따라 하드웨어 로직과 어댑터의 재사용이 어렵다. 또한, 어댑터를 사용함으로 인해 하드웨어 로직의 성능 개선 등의 효과가 반감된다.

최근, [10~11]에서는 로직 수준의 코바 ORB를 통해 FPGA상에 코바 개발 환경을 제공하려는 연구가 진행되고 있다. 이러한 방식은 수십 배 이상의 성능 개선과 안정적인 실행 구조를 제공한다는 점에서 점차 활용 분야와 적용 빈도가 확대되고 있다. 이러한 맥락에서 본 연구팀은 FPGA용 ORB인 HAO[7]를 개발한 바 있다.

HAO는 코바의 기본 통신 체계에서 사용되는 GIOP(General Inter-ORB Protocol) 메시지를 인식하고, 이의 해석을 통해 대상 하드웨어 컴포넌트를 식별하며 필요한 데이터 변환을 거쳐 목적 하드웨어 컴포넌트에 전달하는 과정을 포함한다. 이때, 하드웨어 컴포넌트는 VHDL(VHSIC Hardware Description Language)이나 Verilog와 같은 하드웨어 기술 언어의 데이터와 호출체계로 사상된다. 즉, 하드웨어 컴포넌트는 소프트웨어 컴포넌트와 동일하게 코바 IDL(Interface Definition Language)을 사용하여 정의하며, 개별 하드웨어 컴포넌트는 이를 바탕으로 고유의 데이터 변환 및 하드웨어 컴포넌트 식별을 위한 과정을 갖도록 구조화된다.

본 논문에서는 FPGA에 IP(Intellectual Property) 형태로 직접 적용할 수 있는 하드웨어 미들웨어인 HAO를 기반으로, 이의 활용을 통해 하드웨어 컴포넌트에 대한 재사용성과 상호운용성을 보장하는 과정을 제시하고, HAO로 인해 기존 SCA구조에서 추가로 요구되는 미들웨어 구조의 확장과 활용 체계에 대해 기술한다.

III. 구현독립적인 SCA 미들웨어 구성

SCA는 코바 미들웨어를 통해 개발언어, 운영체제 등과 같은 다양한 단말 환경에 대해 독립성을 보장하고 있다. 하지만, SCA는 하드웨어적 구현에 독립적이지 않다. 이 장에서는 SCA가 하드웨어 구현 독립성을 보장하기 위한 방법으로 FPGA상에 하드웨어 ORB와 IDL을 통한 하드웨어 컴포넌트 구성 수단을 제공하며, 하드웨어 컴포넌트의 적재에 필요한 도메인 프로파일에 대한 고려, 그리고 적재된 하드웨어 컴포넌트과 연동을 위한 IOR 교환 체계와 같이 개선된 기능을 기술한다.

1. 미들웨어 구성

GPP와 FPGA간 구현독립성을 보장하기 위한 미들웨어 구성은 다음과 같다. 전형적으로, GPP에는 운영체제 위에 코바, SCA CF(Core Framework), 그리고 이를 기반으로 서비스와 어플리케이션이 탑재된다. 기존 방식에 확장하여 FPGA에 적재될 하드웨어 컴포넌트와 이들과의 상호운용을 보장하기 위해, FPGA에는 GPP상의 코바와 동일한 역할을 수행하는 미들웨어 체계가 필요하다. 즉, FPGA에는 칩과 보드 의존적인 특성에 대한 투명성을 보장하는 Local Transport, 로직으로 구현된 ORB인 HAO, 그리고 여러 하드웨어 컴포넌트들로 구성될 어플리케이션 로직이 탑재된다(그림 1).

SCA 미들웨어의 논리적 관점인 IDL영역에서, 모든 컴포넌트는 구현 방식(소프트웨어적 혹은 하드웨어적)에 무관하게 컴포넌트간 인터페이스들의 모임인 논리적인 소프트웨어 버스 구조를 통해 상호 연결된다. 즉, 그림에서 모뎀 컴포넌트는 로직으로 구현되고 FPGA에 존재하지만, GPP상의 네트워크 컴포넌트는 모뎀 컴포넌트의 위치(FPGA)나 구현 방식(로직)에 대한 어떠한 고려도 없이 IDL에서 정의된 인터페이스에 따라 다른 소프트웨어 컴포넌트와의 연동 방식과 동일하게 상호연동될 수 있어야 한다.

구현 영역에서, 소프트웨어 컴포넌트와 하드웨어 컴포넌트간의 물리적인 연동은 소프트웨어 버스와 달리, 실제 시스템 버스와 같은 개별 보드 특성에 의존한다. 하지만, 이러한 보드 특성은 GPP와 FPGA상의 코바 미들웨어에 의해 추상화되어야 한다.

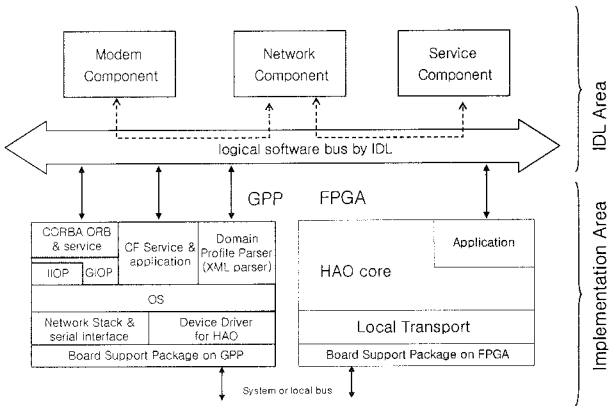


그림 1. 구현독립적인 미들웨어 구조
Fig. 1. Implementation-Independent Middleware Architecture.

2. 컴포넌트 변환 과정

하드웨어 컴포넌트의 지원을 위해 새로이 확장된 절차를 도식화하면 그림 2와 같다. 확장된 절차에서, IDL2VHDL 컴파일러는 IDL로 작성된 인터페이스 정의로 부터 VHDL 템플릿을 생성한다. 기본적으로 하드웨어 컴포넌트를 결정하고 이의 동작을 인가하기 위한 LS(Logic Selector) 블록과 IDL 데이터와 VHDL간 데이터를 변환하는 HC(Hardware Component) 블록이 포함된다. 이외에 HAO에 의해 고려되어야 할 블록 메모리들의 크기, 메모리 주소 등에 대한 정보를 설정 파라미터로 자동 생성한다. 이후, 하드웨어 개발자는 HC 블록의 서브 블록으로 하드웨어 컴포넌트의 고유 역할을 수행하는 DL 블록을 직접 개발하여야 한다. 이후, 이들 블록은 HAO와 함께 합성(synthesis)되고 MAP/P&R과정을 거쳐 FPGA에 적재된다. FPGA상에 적재된 하드웨어 컴포넌트는 전원의 인가와 함께 동작을 시작할 것이다.

추가로, 하드웨어 컴포넌트의 경우에도 인터페이스 정의로부터 생성되는 코드들은 언어 종속적인 부분과 언어 독립적이면서 하드웨어 칩과 보드 의존적인 부분으로 분리된다. 하지만, GPP에서 호출/응답과정이 스택상의 함수 호출구조를 통해 동적으로 이루어지는 반면에, FPGA에서는 로직으로 미리 고정되어 결정되어야 한다는 점에서 다르다. 즉, 하드웨어 컴포넌트와 연관되기 위한 버스 구조와 클럭에 대한 코드가 HAO내에 내장되어야 하고, 이에 따라 HAO의 버스 구조와 클럭에 대한 조정이 필요하다.

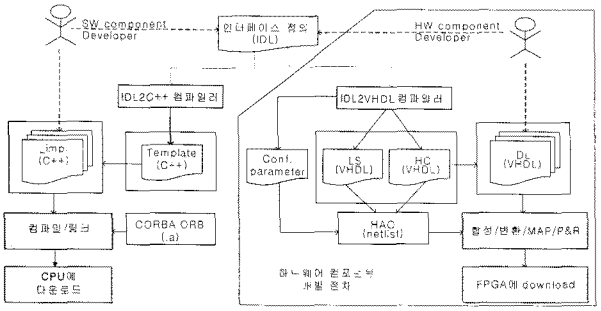


그림 2. 컴포넌트 개발 과정
Fig. 2. Component Development Process.

3. 도메인 프로파일 파서에 대한 고려

SCA는 모든 컴포넌트에 대해 XML로 작성된 도메인 프로파일을 포함하도록 규정하고 있다. 도메인 프로파일은 단말 운용에 필요한 컴포넌트의 논리적 위치, 상호 의존성, 고유 파라미터 특성들을 명세한 파일이다[11].

소프트웨어 컴포넌트와 마찬가지로, 하드웨어 컴포넌트 역시 초기화 및 설정을 위해 자신의 도메인 프로파일을 파싱해야 하며, 이로 인해 도메인 프로파일 파서가 필요하다. 현재, SCA에서 도메인 프로파일을 파싱하기 위한 도메인 프로파일 파서는 GPP의 코어 프레임워크에 내장된 형태로 존재한다.

하드웨어 컴포넌트의 도메인 프로파일 파싱을 위해, FPGA상에 새로운 도메인 프로파일 파서를 위치시키는 방식이 있지만, 파서를 로직으로 구현해야 한다는 점에서 이 방법은 미들웨어 구성에 적지 않은 부담을 주며, 효용성도 크지 않다. 또 다른 방식으로, GPP상의 코어 프레임워크를 경유하여 도메인 프로파일 파서를 공유하는 방법을 통해 보다 단순화할 수 있다. 하지만, FPGA상의 하드웨어 컴포넌트가 기존 도메인 프로파일 파서와의 직접적인 연동체계가 존재하지 않는다. 따라서 내장된 도메인 프로파일 파서에 접근하기 위해서는, 기존 코어 프레임워크에 추가의 규격이 정의되어야 한다는 어려움이 있다[12].

이를 해결하기 위해, 확장된 SCA 미들웨어 구조에서는 GPP상의 도메인 프로파일 파서를 코바 IDL로 정의하고 코바 컴포넌트로 재구현하였다. 즉, 기존 방식에서 코어 프레임워크는 도메인 프로파일 파서를 내부에 내장한 라이브러리로 가지는 반면, 확장된 구조에서는 도메인 프로파일 파서가 코바 IDL을 통해 접근할 수 있는 코바 컴포넌트로 구현되므로, 하드웨어 컴포넌트 역시 IDL에 의한 논리적 소프트웨어 버스를 통해 GPP상의 도메인 프로파일 파서를 사용하여 하드웨어 컴포넌트 자신의 프로파일을 파싱할 수 있다. 이러한 방식의 장점은 FPGA를 포함한 다중 프로세서에 도메인 프로파일 파서에 대한 중복 적재를 방지할 수 있고, 컴포넌트가 탑재된 위치에 무관하게 IDL을 통해 제한 없이 접근이 가능하다.

이를 위해, 도메인 프로파일 파서에 대한 모든 인터페이스는 IDL정의되어야 하며 이에 대한 구현이 제공되어야 한다. 이를 대한 구체적인 사례와 적용은 [12]를 통해 파악할 수 있다.

IV. IDL로부터 스켈리톤/스터브 생성

SCA에서 모든 컴포넌트는 코바를 통해 물리적으로 연결되며, 통신 방식이나 개발언어 등으로부터 독립성

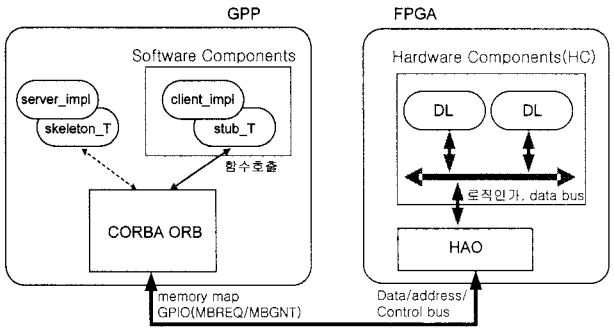


그림 3. 소프트웨어 및 하드웨어 컴포넌트
Fig. 3. Software Component and Hardware Component.

을 보장받고 있다. 소프트웨어 컴포넌트는 일련의 함수 호출과 사상 과정을 통해 코바 ORB와 연계되는 반면, 하드웨어 컴포넌트는 HAO를 통해 개발 로직으로의 데이터 버스와 해당 로직의 전원 인가를 위한 시그널 정의를 포함한다(그림 3).

이를 통해, 하위의 코바 ORB와 HAO는 컴포넌트간의 메시지 요구 전송과 결과 수신을 위한 체계를 보장한다. 소프트웨어 컴포넌트가 하드웨어 컴포넌트에게 요구 메시지를 보내는 경우를 고려해 보자. 소프트웨어 컴포넌트는 목적 하드웨어 컴포넌트 식별 정보와 처리 요구를 포함하는 메시지를 구성하여 코바 ORB에 제시한다. 코바 ORB는 HAO와 유지하고 있는 연결을 통해 GIOP 형식으로 요구 메시지를 전달한다. HAO는 요구 메시지에 포함된 정보의 해석을 통해 목적 하드웨어 로직을 찾아 인가하고, 데이터 전송을 통해 고유의 역할을 수행하도록 한다. 결과는 다시 HAO를 통해 GIOP 형식의 응답 메시지로 구성되어 송신된다. 코바 ORB는 수신한 결과를 호출한 소프트웨어 컴포넌트에 전달함으로서 요구가 완료된다.

이러한 과정을 통해, 소프트웨어 컴포넌트와 하드웨어 컴포넌트는 구현 방식에 대한 고려 없이 상호 독립적으로 연동이 가능하다. 코바 ORB/HAO와 GIOP를 통해 확장된 미들웨어 구조에서, 소프트웨어 컴포넌트와 하드웨어 컴포넌트의 구체화 방식에 대해 다음 절에서 기술한다.

1. 컴포넌트 인터페이스 정의

IDL은 컴포넌트들간의 논리적 연동 버스인 인터페이스를 정의하는데 사용된다. 소프트웨어 컴포넌트와 마찬가지로 하드웨어 컴포넌트의 경우에도, 논리적 버스를 통해 다른 컴포넌트와 연동하기 위해 자신의 인터페이스를 제공해야 하며, 또한 사용하고자 하는 다른 컴포넌트의 인터페이스 정의를 공유하여야 한다.

간단한 인터페이스 정의 예는 다음과 같다.

```
typedef sequence<long> LongSeq;
interface Echo {
  attribute long identifier;
  void echoInt(inout long,inout LongSeq, inout long);
  void echoLong(inout Long);
};
```

Echo 인터페이스는 각각의 컴파일러를 통해 C++ 혹은 VHDL 체계에 따라, 컴포넌트의 선택과 데이터 변환을 수행하는 IDL 컴파일러 영역과 컴포넌트의 고유 기능을 구현해야 하는 개발자 영역으로 나뉘어 생성된다. 하드웨어 컴포넌트의 경우, 하드웨어 개발자는 이에 따라 VHDL을 통해 인터페이스의 구체화 과정에서 실제 구현을 제공해야 한다.

2. 소프트웨어 컴포넌트

소프트웨어 컴포넌트의 경우, IDL2C++ 컴파일러는 인터페이스 정의로부터 소프트웨어 컴포넌트의 스터브(stub) 코드와 스켈리톤(skeleton) 코드를 생성한다(그림 4).

스켈리톤은 개발자가 구현한 서버가 ORB를 통해 응답하기 위해 필요한 구조와 선언을 포함하는 구현 템플릿이다. 반면에, 스터브는 구현된 서버에 요구하기 위해 클라이언트가 사용하기 위한 구조와 오퍼레이션 호출 구조를 제공한다. 이후, 개발자는 전형적으로 생성된 스켈리톤을 상속(inheritance)받고 이에 대한 구체화를 통해 소프트웨어 컴포넌트(Echo_impl)를 완성한다. 이들

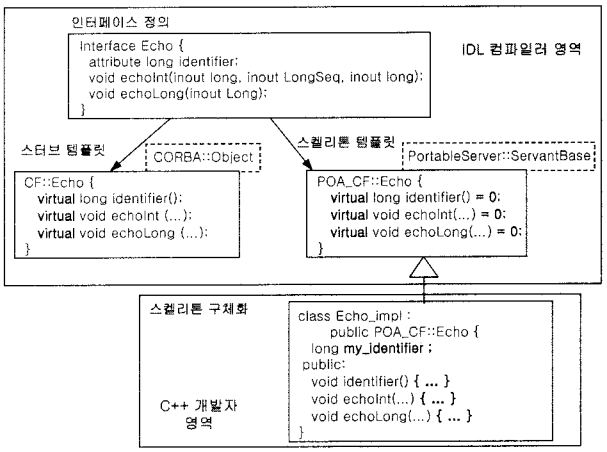


그림 4. 소프트웨어 컴포넌트
Fig. 4. Software Component.

은 C++컴파일러에 의해 컴파일 되고 GPP에 적재된 후 실행된다.

3. 하드웨어(로직) 컴포넌트

하드웨어 컴포넌트는 FPGA상에서 수행되어야 할 하드웨어 로직이다. 이들은 VHDL로 구체화되며, HAO의 식별과정을 통해 전원이 인가되면 이후 HAO로부터 전달받은 시그널과 데이터를 사용하여 역할을 수행한다.

P&R이후 하드웨어 컴포넌트가 FPGA에 적재되면, 물리적인 위치가 고정된다. 하드웨어 컴포넌트는 소프트웨어 컴포넌트와 달리 시그널과 데이터 교환을 위한 버스 구조와 타이밍 제약의 설정을 통해 구체화가 완성된다. 하드웨어 컴포넌트의 구체화는 템플릿의 고정된 버스와 시그널을 공유해서 사용해야 하므로, 그림 5와 같이 집성화에 의해 내부 서브 블록으로 이루어진다.

소프트웨어 컴포넌트는 하드웨어 컴포넌트를 사용하기 위해, 하드웨어 컴포넌트의 인터페이스 정의를 IDL2C++로 컴파일하여 얻어진 스터브를 사용한다. 이때, 스터브는 컴포넌트의 구현 방식에 관련 없이 IDL 정의로부터 GIOP 요구 메시지를 생성하는 방식은 동일하다.

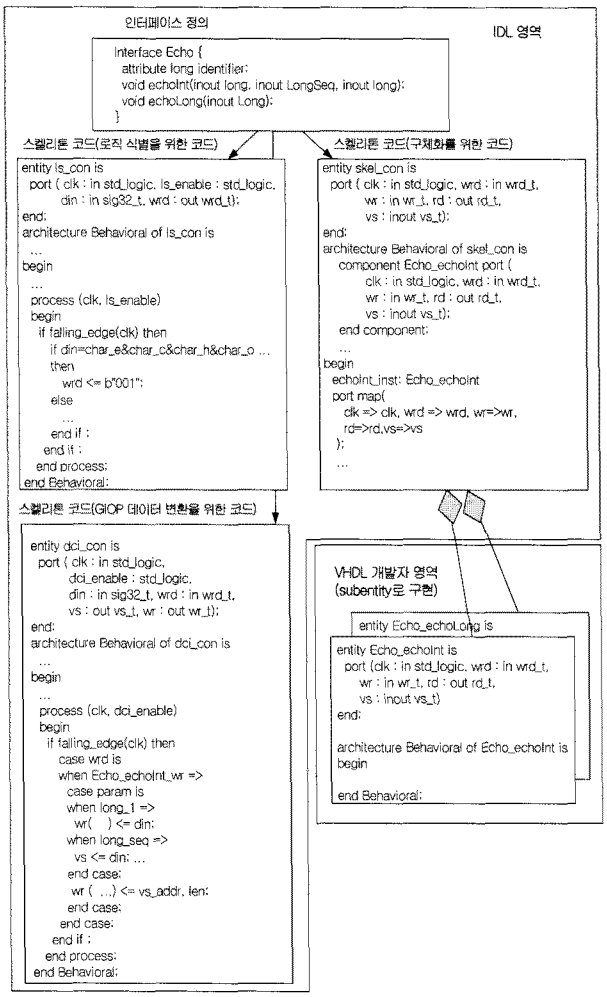


그림 5. 하드웨어 컴포넌트
Fig. 5. Hardware Component.

한편, 하드웨어 컴포넌트의 경우, FPGA내 존재하는 또 다른 하드웨어 컴포넌트에 대한 호출은 두 가지 방식으로 가능하다. 첫째, 소프트웨어 컴포넌트와 동일하게 ORB간 GIOP 메시지 전송 방식을 사용하는 방식이 있으며, 둘째, 원본 하드웨어 컴포넌트가 목적 하드웨어 컴포넌트를 직접 인가하는 방식도 가능하다. HAO에서 두 가지 방식이 모두 가능하나, 전자의 경우 GIOP 메시지를 구성하고, 메시지 버퍼의 대기시간, GIOP 수신 후 해석 등의 과정에 여러 오버헤드가 요구된다. 따라서 지극히 높은 수준의 독립성이 요구되지 않는다면, 원본 하드웨어 컴포넌트에서 직접 인가하는 방식이 보다 효율적이다. 이에 대한 선택은 전적으로 애플리케이션의 요구사항과 추구하는 설계 방향에 따라 결정될 것이다.

4. IDL 컴파일과 IOR의 할당

IDL2VHDL 컴파일러는 IDL2C++컴파일러와 달리, 몇 가지 추가 요구사항을 갖는다. FPGA 특성상, 합성과 P&R을 거쳐 FPGA에 적재하면, HAO를 포함하여 모든 컴포넌트들간의 물리적인 바인딩이 고정된다. 따라서 하드웨어 컴포넌트에 대한 추가의 동적 생성과 같은 기능은 효율적이지 않다. 또한 효율성 측면에서 HAO와의 잦은 연동은 조율되어야 한다. 이는 하드웨어 컴포넌트에서 기능을 수행하는 것에 비해, 시스템 버스를 점유하고 GIOP를 송수신하는 과정의 오버헤드가 더 클 수 있고 GPP의 장치 드라이버는 FPGA에 비해 비교적 큰 처리 단위로 스케줄링되기 때문이다.

이러한 특징들을 고려할 때, 모든 하드웨어 컴포넌트들은 IDL2VHDL에 의한 컴파일 시점에 IOR 설정 정보들을 미리 할당하는 것이 바람직하다. 이는 IDL의 컴파일 시점에, 특정 FPGA에 설정될 HAO와 연결될 하드웨어 컴포넌트가 물리적으로 결정되기 때문에 가능하다[7]. 현재, 하드웨어 컴포넌트용 IOR에는 {FPGA식별자, HAO식별자, 하드웨어컴포넌트 식별자}을 포함하도록 하고 있다.

이후, 생성된 하드웨어 컴포넌트의 IOR은 GPP상의 코바 네이밍 서비스에 등록하는 절차가 별도로 수행되어야 한다. 이를 통해, 하드웨어 컴포넌트에 대한 잦은

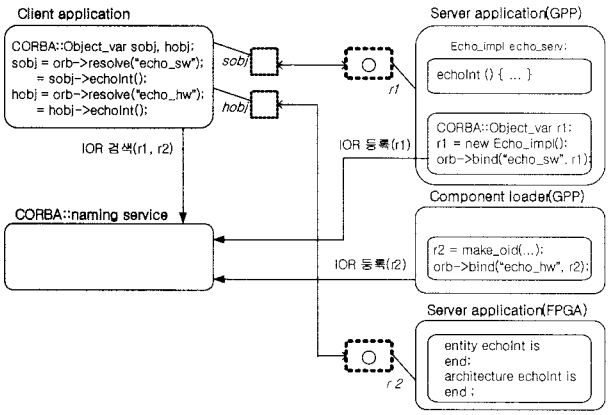


그림 6. 네이밍 서비스
Fig. 6. Naming Service.

연동을 가능한 축소하고 하드웨어 컴포넌트의 연동/초기화 과정을 보다 단순화할 수 있다.

네이밍 서비스는 SCA에서 운용되는 컴포넌트에 대한 식별과 연결을 위해 필요하다. SCA 환경에서 모든 컴포넌트는 기본적으로 네이밍 서비스를 통해서 인식되고 상호 연결될 수 있기 때문이다(그림 6).

모든 서버 컴포넌트는 스켈리톤 코드의 구체화를 완료하고, 자신을 식별할 수 있는 IOR과 함께 네이밍 서비스에 등록되어야 한다. 이후, 클라이언트는 목적 서버의 명칭을 사용하여 네이밍 서비스로부터 검색하며, 검색된 IOR을 통해 서버에 직접 요구 메시지를 송신하고 그 결과를 수신할 수 있다.

V. 코바 ORB의 확장

하드웨어 컴포넌트와 연동하고자 하는 소프트웨어 컴포넌트는 네이밍 서버에 존재하는 IOR을 참조하여, 목적 하드웨어 컴포넌트의 식별자와 요구 및 데이터를 적용하고, 장치 드라이버를 통해 메시지를 송신함으로서 FPGA상의 하드웨어 컴포넌트와 연동될 수 있다.

이 장에서는 이를 위한 코바 ORB의 적용 및 확장에 대해 기술한다.

1. ORBit와 orbitcpp

소프트웨어 컴포넌트는 코바 ORB를 통해 IDL에 의한 논리적 버스에 참여할 수 있다. 이 절에서는 코바 오픈 프로젝트를 사용하여 소프트웨어 미들웨어의 실제적인 적용 사례에 대해서 설명하고, 확장된 미들웨어 구조에 적용되기 위해 추가 고려된 기능과 개선 사항에 대해 기술한다.

본 논문에서 사용한 코바는 오픈 프로젝트에서 많이 사용되고 있는 ORBit 0.5.17[13]과 ORBitcpp 1.3.9[14]를 대상으로 하였다.

가. 최소 코바, C++ 변환, 그리고 네이밍 서비스

SCA 미들웨어는 제한된 환경에서 운용될 수 있도록 최소 코바(minimum CORBA)의 사용을 권고하고 있다[1]. ORBit은 전체 코바(full CORBA) 규격 지원을 목표로 작성되었기 때문에, 권고에 비해 많은 오버헤드를 포함하고 있다. 최소 코바 규격을 만족시키기 위해, [15]에 따라 28개의 인터페이스, 5개 인터페이스에서 37개의 오퍼레이션이 제외되었다. 결과로 얻어진 최소 코바는 크게 orb, idl, iiop와 같은 세 개의 라이브러리를 통해 제공되고 있으며, 37%크기로 경량화되었다.

SCA의 요소는 코바 컴포넌트이며, 이에 따라 코어 프레임워크의 모든 서비스와 어플리케이션 인터페이스 역시 IDL로 정의되었다. 이들은 IDL 컴파일러에 의해 변환되어야 한다. 본 연구에서는 ORBitcpp를 사용하여 SCA의 코어 프레임워크 IDL을 컴파일하고, 생성된 스켈리톤으로부터 코어 프레임워크를 구체화한 바 있다. 이를 통해, SCA 응용에 적용할 수 있는 ORBitcpp의 기능 검증을 수행하였다.

ORBitcpp는 ANSI 호환의 C++ 코드 생성을 보장하지만, 변환 규칙과 기능상의 개선이 부분적으로 요구되었다. 코어 프레임워크와 컴포넌트 개발과정에서 표 1과 같은 주요 문제점들이 발생되었고, 이에 대한 기능 보정이 이루어졌다. 아울러, IDL내 선언과 정의가 분리되지 않아 생기는 오류 등과 같은 문제점들이 함께 보정되었다.

표 1. ORBitcpp의 주요 개선항목
Table 1. Improvement of ORBit.

메모리관련	_var 표준 IDL 메모리 관리 정책 위반 (원본 데이터 손실)
Sequence	Ctor : 길이가 0인 경우 오류 동적 생성시 길이가 0인 경우 오류
C++ mapping	copy constructor, assign operator의 선언/구현 생략 등
Interface 추가	CORBA::TypeCode, CORBA::Current, CORBA::Any::type() 등 추가
Type mapping	struct내 Object, array 미지원
	wstring sequence 불허
IIOP 관련	로컬 IP의 적용을 불허

나. 시스템 버스를 사용하는 GIOP의 구체화

동일 ORB상에 탑재된 서버와 클라이언트 컴포넌트는 기본적으로 내부 함수 호출을 통해 연결된다. 반면에 다른 ORB상의 서버 컴포넌트 호출은 ORB 하위의 ORB간 통신 규격을 사용하여야 한다. GIOP는 ORB간 메시지 교환을 위한 통신 규격이다. 통상적으로 GIOP는 시스템에서 제공하는 물리적인 통신 수단으로 구체화되어야 하며, GPP에서 GIOP는 TCP/IP를 사용하는 IIOP로 구체화한다. 하지만, FPGA 환경에서 운영체제나 이더넷의 탑재는 매우 큰 제약이다. 대신에 HAO는 이더넷 IP core의 사용을 배제하고 시스템 버스와 직접 연결되도록 설계하였다. 따라서 ORBit는 구체화 수단으로 기존 IIOP 뿐만 아니라 시스템 버스를 제어하는 디바이스 드라이버를 추가로 제공한다(그림 7).

디바이스 드라이버는 두 영역으로 구성된다. GPP에서 볼 수 있는 GIOP 메시지 버퍼를 제어하기 위한 메모리 맵의 제공과 시스템 버스를 제어하는 시그널들에 대한 인식이다. 메모리 맵은 GIOP 메시지의 버퍼이며 물리적인 영역은 보드상의 확장 메모리(DPRAM, SRAM) 혹은 FPGA 내부 메모리일 수 있다. ORBit와 HAO는 시스템 버스를 통해 메시지를 송수신하므로 상호 경쟁적으로 시스템 버스의 점유를 시도할 수 있다. 이를 해결하기 위해, GPP의 GPIO(General Purpose Input/Output)의 부가 기능인 MBREQ와 MBGNT을 사용한다. HAO에 의해 생성된 MBREQ 시그널은 보드를 제어하는 CPLD를 거쳐 GPP에 시스템 버스 사용을 요청한다. GPP는 이 시그널 수신이후, 자신의 시스템 버스 사용을 종료한 후, MBGNT로 응답한다. 따라서 FPGA는 MBREQ을 통한 요청 후 MBGNT가 수신되면 시스템 버스를 사용할 수 있다.

이에 따라, 디바이스 드라이버는 ORBit의 ORB로부터 요구 GIOP 메시지를 수신하여, 시스템 버스와의 충돌없이 보드내 확장 메모리(혹은 FPGA내 메모리)에 송신할 수 있고 또한 HAO가 확장 메모리에 전송한 결과 GIOP 메시지를 수신하여 GPP상의 해당 ORB에 전달할 수 있다.

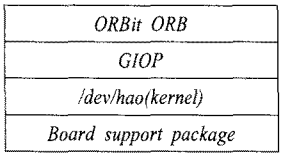


그림 7. GIOP 구체화
Fig. 7. GIOP Implementation.

2. Hardware ORB(HAO)

HAO는 FPGA상에 구현된 코바 미들웨어이다[7]. HAO는 기본적으로 GIOP 메시지를 수신하고, 해석한 후 하드웨어 컴포넌트의 처리 결과를 GIOP 메시지로 응답하는 절차를 제공한다. 이 장에서는 HAO에 개괄적인 구조와 하드웨어 컴포넌트에 대해 기술한다.

가. 보드 환경

단말에 따라 보드 환경의 구성은 매우 다양하다. 그림 8은 ORBit와 HAO가 보드상의 확장 메모리를 GIOP 메시지 버퍼로 사용하는 경우이다.

ORBit의 /dev/hao와 HAO의 Local transport는 상호간의 신뢰성있는 GIOP 메시지의 송수신을 담당한다. HAO core는 실장된 보드의 개별성에 독립적으로 수신된 GIOP 메시지를 해석하고 처리하며, 그 결과를 GIOP 메시지로 구성하여 반환하는 역할을 수행한다.

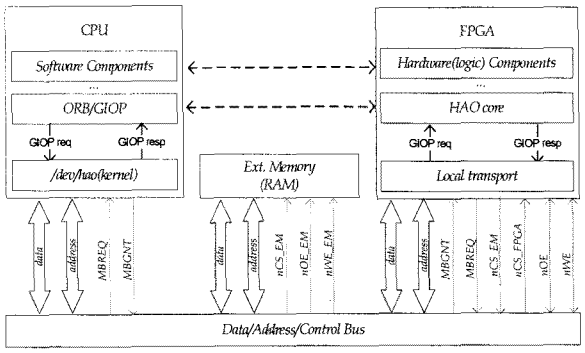


그림 8. ORBit와 HAO의 연동
Fig. 8. Interworking between ORBit and HAO.

나. Local Transport

Local transport는 HAO를 칩/보드 구성에 관련된 의존성으로부터 분리시킨다. 구체적으로, 보드 상의 자원인 시스템 버스와 메시지 버퍼링을 위한 메모리 관리에 대한 추상화를 지원한다. 기본적으로, 보드상의 여러 요소들과 MBREQ/MBGNT를 통해 경쟁적으로 시스템 버스에 대한 점유와 해제를 수행하며, 보드 상의 확장 메모리 혹은 FPGA내의 블록 메모리 등을 사용하여 GIOP 메시지에 대해 제어한다. 이때, 시스템 버스 대신에 ORBit의 IIOP와 연동되도록 변경하는 것과 같이 HAO의 물리적인 인터페이스가 변경된다면, 오직 Local transport만의 변경으로 적응할 수 있다.

그림 9는 Local transport의 블록 구조이다. 그림에서 내부 레지스터(register)는 메시지 버퍼에 사용되는 각

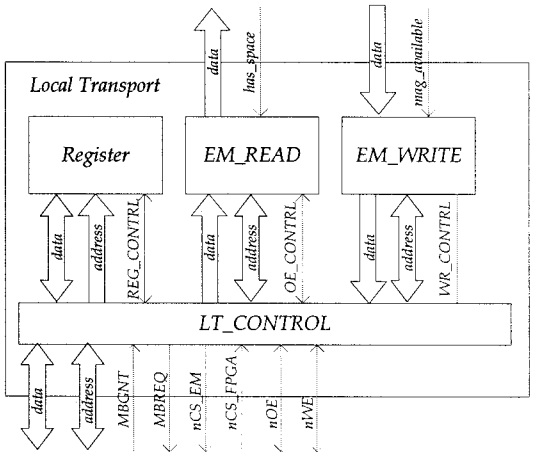


그림 9. Local Transport 블록도
Fig. 9. Local Transport block diagram.

종 설정 및 제어 정보들을 저장하며, 모든 메시지 송수신 과정에서 이 정보들이 사용된다. 내부 레지스터는 GPP에 설정된 메모리 맵을 통해 /dev/hao와 공유되며, 제어 시그널들로 FPGA 인가를 위한 nCS_FPGA, 데이터 추출을 위한 nOE, 데이터 전달을 위한 nWE, 그리고 데이터/주소 버스가 함께 사용된다.

외부 메모리에 존재하는 GIOP 메시지의 송수신을 위해 시스템 버스를 먼저 점유하여야 하며, 이를 위해 MBREQ를 요청한 후 MBGNT 시그널을 수신하면 처리를 시작한다.

EM_READ 블록은 확장 메모리로부터 GIOP 메시지를 수신하기 위한 블록이며, HAO Core내 GIOP 버퍼링의 여유가 있는 경우에만 수신을 시작한다. EM_WRITE 블록은 HAO Core에 의해 송신할 GIOP 메시지가 존재하는 경우, 이를 확장 메모리에 송신하는데 사용한다. FPGA내 하드웨어 컴포넌트는 소프트웨어 컴포넌트에 비해 상대적으로 처리 성능이 높기 때문에, Local transport에서 송신은 수신에 비해 보다 높은 우선순위를 갖는다. 그림 10은 GPP로부터 확장메모리에

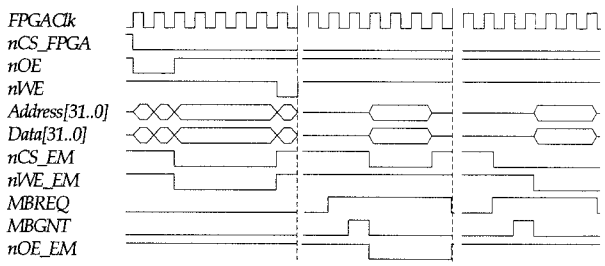


그림 10. GIOP 메시지 송수신 타이밍
Fig. 10. timing diagram for exchanging the GIOP message.

GIOP 메시지를 송신하는 단계(a), 송신 완료후 Local transport에 의해 송신된 GIOP 메시지를 읽는 단계(b), HAO에 의해 생성된 응답 GIOP 메시지를 다시 반환하는 단계(c)를 설명하는 타이밍 다이어그램이다.

첫 단계에서, GPP는 local transport내 레지스터 영역으로부터 확장 메모리에 구현된 큐(Queue)의 rear/front 주소를 읽는다. 큐에 여유 공간이 있을 경우, GPP로부터 확장 메모리를 쓰는 과정이 적용된다. 쓰기 과정이 완료한 후, 다시 큐의 front에 대한 변경을 수행한다. 두 번째 단계로, FPGA의 동작은 인가된 경우, Local transport는 큐의 rear/front에 대한 인식 후 확장 메모리로부터 GIOP 메시지를 수신한다. 수신된 메시지는 HAO Core에 전달된다. 세 번째 단계에서, HAO Core는 처리를 완료한 후 송신할 메시지를 확장 메모리에 쓰는 동작을 수행한다. 이때, MBREQ를 통해 시스템 버스에 대한 접근이 허용된 경우 수행한다.

한편, Local transport는 HAO Core와 두 개의 인터페이스로 연결된다. 보드로부터 수신한 GIOP 메시지를 송신하기 위한 FIFO 메모리 인터페이스와 HAO Core에 의해 생성한 GIOP 메시지를 수신하기 위한 인터페이스로 구성된다.

다. HAO core와 하드웨어 컴포넌트

HAO Core는 local transport를 통해 수신한 GIOP 메시지를 해석하고, 수행되어야 할 하드웨어 컴포넌트를 인가하고 데이터를 전달하며, 그 결과를 수신하는

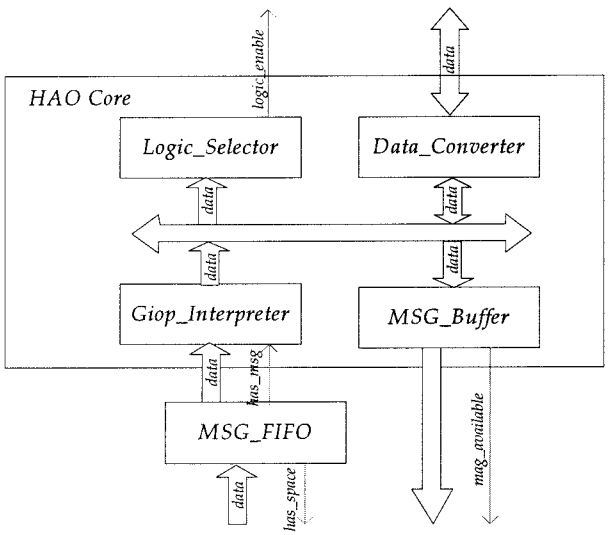


그림 11. HAO core 블록도
Fig. 11. HAO Core block diagram.

역할을 수행한다. 그림 11은 HAO core의 블록 구조도를 보인다. 블록 구조도에서, GIOP_Interpreter는 GIOP 메시지에 대한 분석을 수행한다. 분석을 통해 Logic Selector에 대상 객체 ID를 전달하며, 객체 ID에 대한 해석을 통해 대상 하드웨어 컴포넌트를 식별하고 이를 인가한다. 또한, 하드웨어 컴포넌트를 식별하는 과정에서 상속(inheritance)과 같은 여러 이슈들에 대한 고려를 담당한다. 또한, 메시지에 포함된 데이터들은 Data Converter를 통해 추출되어 이를 하드웨어 컴포넌트에 전달하게 된다. 하드웨어 컴포넌트와 HAO Core간의 버스연결은 최적화 되어야 하며, 중첩된 데이터 구조(structure, sequence, array 등)와 공유 속성(shared attribute)에 대한 버스연결 방식 등과 같은 다양성 해소를 담당한다. MSG_buffer는 송신할 GIOP 메시지를 구성하기 위해 임시적으로 사용되며, HAO Core에서 송신이 수신에 비해 우선순위를 가지므로 소규모 버퍼로 구성된다.

VI. 검 증

이 장에서는 소프트웨어 컴포넌트와 하드웨어 컴포넌트간의 연동 과정을 다룬다. 성능평가의 세부적인 방법과 절차는 [7]과 동일하게 적용되었다. 간단히 요약하면, 검증을 위해 GPP (PXA272)와 Xilinx FPGA 칩(XC4VLX60)을 함께 제공하는 상용 보드(SYSLAB 3000)를 사용하였다. PXA272에는 Linux(Kernel v2.6.x)와 함께 ORBit를 포팅하였고, 추가로 보드상의 SRAM을 통한 메시지 송수신을 위한 디바이스 드라이버가 함께 탑재되었다.

다음 GPP용 어플리케이션 예에서 클라이언트(소프트웨어 컴포넌트)는 IDL2C++를 사용하여 얻어진 스터브를 통해 하드웨어 컴포넌트 Echo 오퍼레이션(echoInt)의 실행을 요구하도록 구성되었다. 반면에 서버(하드웨어 컴포넌트)의 오퍼레이션 echoInt는 두 Long 값을 교환하고 sequence내 요소의 위치를 변경하도록 FPGA에서 로직으로 작성되었다.

```
CORBA::Long a=0xaaaaaaaa, c=0xcccccccc;
sequence<long> s(16);
s.length(2);
s[1] = 2;
s[2] = 3;

try {
  hobj->echoInt(a, x1, c);
} catch (CORBA::SystemExecption& e) {
  ...;
}
```

클라이언트에서 요청한 echoInt 오퍼레이션과 파라미터는 GIOP에 따라 메시지로 구성되고, 보드상의 확장 메모리(SRAM)내 ORBit2HAO 버퍼 영역에 저장된다. 저장된 메시지는 HAO에 의해 수신되어 처리된다.

서버에 대해, 세 개의 매개변수로 (0xaaaaaaaa, {2, 3}, 0xcccccccc)을 갖는 echoInt 오퍼레이션 호출시, 커널상의 /dev/hao에서 GIOP 메시지들에 대한 송수신 내용은 다음과 같다.

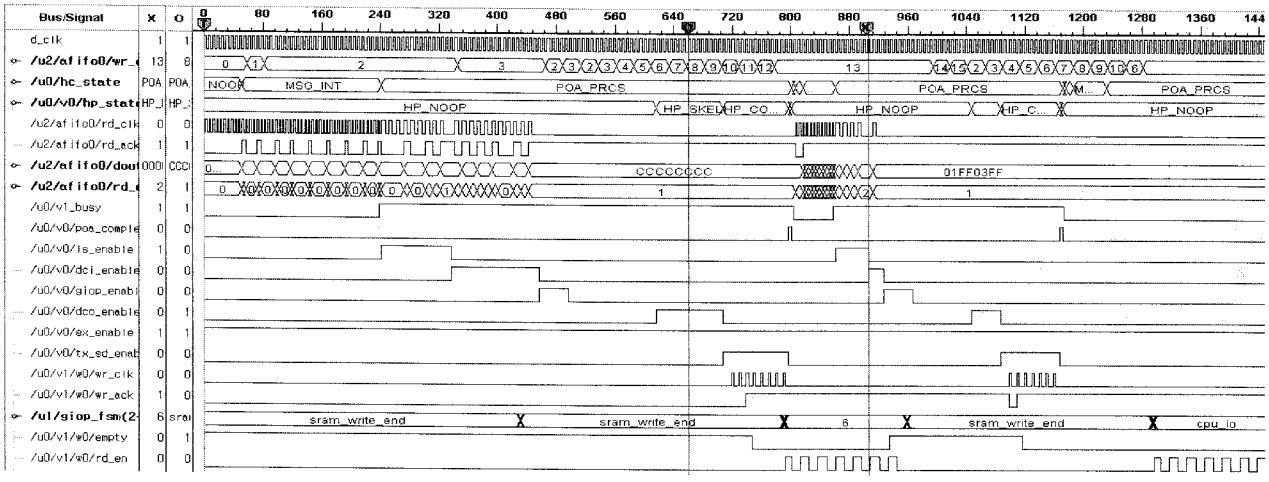


그림 12. Echo의 오퍼레이션(echoInt, echoLong) 처리에 대한 타이밍 다이어그램
Fig. 12. timing diagram for processing Echo operation(echoInt, echoLong).

```
[root@mercury hao_test]# ./echo_client
Using hao_ctrl.ko
<4>App_sram_write_base_virtual_address: 0xc8c00000
<4> - phys2virt address               : 0x29200000
<4>App_sram_read_base_virtual_address : 0xcd400000
<4> - phys2virt address               : 0x29000000
<4>send msg
<4> 47494f50 01000100 38000000
<4> 00000000 a0f11200 01000000
<4> 04000000 00010000 08000000 6563686f 496e7400 00000000
<4> aaaaaaaa 02000000 02000000 03000000 cccccccc
<4>receive msg
<4> 47494f50 01000101 20000000
<4> 00000000 a0f11200 01000000
<4> cccccccc 02000000 03000000 02000000 aaaaaaaa
```

디바이스 드라이버 상의 메모리 맵에 포함한 WBR(Write Base Register)은 0x29200000 주소로 접근 가능한 버퍼 제어 레지스터이며, 이를 통해 GIOP 메시지를 송신한다. 이때, 하드웨어 컴포넌트 echoInt는 각 매개변수 값을 교환한 후 반환하고 있다.

FPGA에서 echoInt에 대한 시그널/버스 수준의 처리 과정은 ChipScope를 통해 확인할 수 있다(그림 12). Local transport는 SRAM에서 수신한 GIOP 메시지를 HAO Core에 전달한다. HAO Core는 하드웨어 컴포넌트가 바인드되어 있는 POA를 인가(v1_busy)한다. 이후, 하드웨어 컴포넌트의 echoInt/echoLong 블록을 인가하기 위한 Logic selector 처리과정(is_enable), 해당 블록에 필요한 파라미터를 추출하고 시그널을 변환하는 Data converter 과정(dci_enable)을 거친 후 해당 블록이 처리된다. 블록의 처리가 완료된 후, 반환값 및 반환 파라미터를 처리하는 과정(dco_enable)이 수행된다. 이후, 송신을 위해, SRAM에 반환 GIOP 메시지를 전송하는 과정이 수행된다.

현재, ORBit의 모든 Testbench를 로직으로 구현한 각 하드웨어 컴포넌트는 HAO를 포함해서 평균 3,000여 로직 셀로 구성되고 있으며, 커널 레벨(/dev/hao)에서 ORBit의 성능과 비교하여 평균 30배 이상의 성능을 제공하고 있다.

Ⅶ. 결 론

본 논문은 SCA 환경에서 GPP와 FPGA를 대상으로 모든 컴포넌트들간의 상호운용성을 보장하기 위해 소프트웨어 미들웨어로서 ORBit를 적용하고 보완하였으며, 추가의 확장인 HAO와 그 연동 방식에 대해 기술하였다.

이를 위해, ORBit에서 필요한 디바이스 드라이버, 네이밍 서비스에 대한 여러 고려와 절차 등에 대해 기술하였다. 또한, 최소 코바의 규격을 준수하며 FPGA의 특성상 동적인 역할들을 제외한 소형화된 로직용 ORB인 HAO, 그리고 IDL 정의로부터 하드웨어 기술 언어인 VHDL로의 매핑을 제공하는 IDL2VHDL 컴파일러에 대해 간략히 기술하였다.

본 논문을 통해 제시한 ORBit과 HAO와의 연동, 컴포넌트간의 상호운용 방안, 도메인 프로파일 파싱에 대한 고려 등을 통해, 기존 SCA 환경에 비해 하드웨어 구현에 대한 독립성을 추가로 보장할 수 있다. 이의 활용을 통해, GPP와 FPGA, 하드웨어 보드에 대한 의존성없는 SCA기반 플랫폼에 적용할 수 있다. 현재, HAO는 검증과 성능 향상을 위한 개선이 진행되고 있다. 아울러, Xilinx의 대표적인 칩들에 대한 적용을 진행하고 있다.

참 고 문 헌

[1] Joint Tactical Radio Systems, "Software Communications Architecture Specification V2.2." Nov. 2002.

[2] J. Kulp, M. Bicer, L. Pucker, and G. Holt, "Portable Waveform Components for Specialized Hardware," JPO Portability Workshop, Jan. 2005.

[3] Joe Jacob, "CORBA for FPGA: The Missing Link for SCA Radios," COTS Journal, vol. 9, no. 1, pp. 30-33, Jan. 2007.

[4] Object Management Group, "The Common Object Request Broker Architecture: Core Specification Revision 3.0." Dec. 2002.

[5] W3C Recommendation: Extensible Markup Language(XML) 1.0, Feb. 1998.

[6] POSIX Std. 1003.13: Standardized Application Environment Profile-POSIX Realtime Application Support(AEP), 1998.

[7] 배명남, 이병복, 박애순, 이인환, 김내수, "FPGA에서 SCA 컴포넌트 개발을 지원하는 하드웨어 ORB", 한국통신학회 논문지, 제46편, 제3호(무선통신), pp. 185-196, 2009.

[8] John Huie, Price D'Antonio, Robert Pelt, and Brian Jentz, "Synthesizing FPGA Cores for Software Defined Radio," SDR Forum 2003, Nov. 2003.

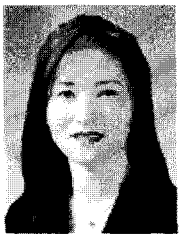
[9] Mark Hermeling, "Component-Based Support for FPGAs and DSPs in Software Defined Radio," SDR Forum 2006, Nov. 2006.

[10] S. Aslam-Mir, "ICO : Integrity Circuit ORB," PrismTech White paper, 2006.

[11] "ORBexpress FPGA," Objective Interface White paper, 2007.
[12] 배명남, 이병복, 박애순, 이인환, 김내수, "SCA에서 적응형 도메인 프로파일 파서의 구축 방법", 대한전자공학회논문지 제46권 CI편, 제1호, pp 103-111, 2009.
[13] ORBit Project, http://orbit.sourceforge.net/
[14] orbitcpp Project, http://orbitcpp.sourceforge.net/
[15] minimumCORBA: OMG Document orbos/98-05-13, May 19, 1998.

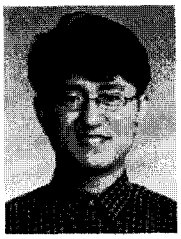
저 자 소 개

배 명 남(정회원)
1991년 전북대학교 전산통계학과 학사 졸업
1993년 전북대학교 전산통계학과 석사 졸업
1998년 전북대학교 전산통계학과 박사 졸업
1998년～현재 한국전자통신연구원 USN응용기술연구팀 책임연구원
<주관심분야 : SDR, 통신 미들웨어, 개방형 플랫폼, 임베디드 하드웨어, 무선통신>



박 애 순(정회원)
1987년 충남대학교 전자계산학과 학사 졸업
1997년 충남대학교 전자공학과 석사 졸업
2001년 충남대학교 컴퓨터과학과 박사 졸업
1988년～현재 한국전자통신연구원 차세대이동단말연구팀장(책임연구원)
<주관심분야 : 4세대 이동통신, 이동통신망, 이동성관리, 이동단말기술>

김 내 수(정회원)
2000년 한남대학교 컴퓨터공학과 박사 졸업
1986년～1990년 국방과학연구소
1990년～현재 한국전자통신연구원 RFID/USN연구부 USN기반기술연구팀 책임연구원
<주관심분야 : RFID/USN, 위성통신, 컴퓨터 네트워크>



이 병 복(정회원)
1991년 호원대학교 전자계산학과 학사 졸업
1993년 전북대학교 전산통계학과 석사 졸업
2005년～현재 고려대학교 전산학과 박사과정
1993년～현재 한국전자통신연구원 USN응용기술연구팀 책임연구원
<주관심분야 : 이동통신단말기 시스템, 임베디드 시스템 개발 및 실행환경, 무선 센서네트워크 전송기술>

이 인 환(정회원)
1988년 한양대학교 전기공학과 학사 졸업
1990년 한양대학교 전기공학과 석사 졸업
2007년～현재 한양대학교 전자컴퓨터통신공학과 박사과정
1990년～1993년 (주)동아전기 연구원
1993년～현재 한국전자통신연구원 USN응용기술연구팀 책임연구원
<주관심분야 : RFID/USN, 무선통신>