FPGA에서 SCA컴포넌트 개발을 지원하는 하드웨어 ORB

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

A Hardware ORB for Supporting the SCA-based Component Development in FPGA

Myung-Nam Bae*, Byung-Bog Lee*, Ae-Soon Park**, In-Hwan Lee*,
Nae-Soo Kim* ***Regular Members***

요 약

SCA는 단일 단말 플랫폼에 여러 무선체계를 운용하기 위해 제안되었고, 소프트웨어 컴포넌트들에 대해 플랫폼 독립성을 보장하기 위해 코바 미들웨어를 채택하고 있다. 최근, 여러 이유로 소프트웨어 컴포넌트에서 로직 수준으로 재구현 요구가 확대됨에 따라, 코바 미들웨어는 FPGA를 포함한 하드웨어 보드에 대한 독립성을 추가로 보장하여야 한다. 이에 따라, 하드웨어 보드에 의존적인 특성들을 추상화하고, 컴포넌트에 대한 IDL기반 연동 인터페이스를 제공할 수 있는 하드웨어 미들웨어의 필요성이 대두되었다. 본 논문에서는 FPGA용 ORB인 HAO의 개발에 대해 기술하였으며, 구체적으로는 하드웨어 보드에 대한 독립성을 보장하기 위한 local transport, 그리고 GPP용 ORB와 동일하게 다른 컴포넌트와 코바 IDL에 의한 연동을 제공하기 위한 HAO Core를 포함한다.

현재, HAO는 평균 2,900 로직셀 크기의 초경량 ORB로 구성되었으며, 소프트웨어 컴포넌트 대비 수십 배의 성능 개선을 보였다. 이를 통해, SCA기반의 시스템 구축에 있어서, 그 개발 영역을 소프트웨어 뿐만 아니라 FPGA 로직까지 자연스럽게 확장할 수 있게 되었다.

Key Words : CORBA, FPGA, SCA, Hardware Components, SDR.

ABSTRACT

SCA is proposed in order to operate various wireless systems in the single terminal platforms and uses the CORBA middleware to guarantee the platform-independence for software components. As the reconstruction demand is expanded in the software component to the logic level to many reasons, CORBA has to guarantee the independence of hardware on board. Accordingly, the characteristics depending on hardware board is abstracted. And the IDL-based interworking interface about the component has to be provided. In this paper, we described about local transport for guaranteeing the independency on the hardware board and the HAO Core for providing a coupling by the CORBA IDL identically with the other component.

HAO produced at 2,900 logic cell size in average and provided the performance of the tens times than the software component. Through the use of HAO in the SCA-based development environment, it was naturally expanded to not only the software area but also the FPGA logic.

※ 본 연구는 지식경제부 및 IITA의 신성장동력핵심기술개발사업(2005-S-404-33), 국토해양부 지능형국토정보기술혁신사업(06국토정보C01)의 연구비지원에 의해 수행되었습니다.
* 한국전자통신연구원 USN전송기술연구팀 ({mnbae, bblee40, ihlee, nskim}@etri.re.kr)
** 한국전자통신연구원 차세대이동단말연구팀 (aspark@etri.re.kr)
논문번호 : KICS2008-09-421, 접수일자 : 2008년 9월 25일, 최종논문접수일자 : 2009년 2월 10일

Ⅰ. 서 론

단일 플랫폼에서 여러 무선체계(waveform)간의 상호운용과 재구성을 보장하기 위한 객체지향 분산 프레임워크 기술로 SCA(Software Communications Architecture)가 제안되었다. 무선통신 환경에서 SCA는 코바(CORBA, Common Object Request Broker Architecture) 미들웨어를 사용함으로써, 프레임워크 내의 모든 컴포넌트들은 코바 ORB/GIOP를 통해 플랫폼에 의존하지 않고 상호 독립적으로 연동될 수 있다.

현재, 플랫폼내 핵심 부품인 범용 프로세서(이하 GPP, General Purpose Processor)가 비약적인 발전을 이루고 있으나, GPP상의 컴포넌트들이 FPGA (Field Programmable Gate Array)에서의 구현이 확대되고 있다. FPGA 프로그래밍의 경우, 매우 저수준에서 이루어져야 하고 설계시에도 패브릭(fabric)의 특정 자원들과 밀결합되어야 하는 등의 어려움이 있다. 하지만, 이러한 문제에도 불구하고, FPGA는 개별 컴포넌트들에 대해 문맥 교체(context switching)등과 같은 제약을 받지 않는 병행처리를 보장함으로써 성능을 극대화할 수 있다[1,2].

현재의 SCA에서 컴포넌트를 로직으로 구현할 때 여러 고려가 필요하다. 첫째, GPP에는 FPGA상의 로직과 연동하기 위한 장치 드라이버가 필요하다. 장치 드라이버는 보드상의 GPP와 FPGA간의 세부적인 제어 메커니즘에 의존성을 갖게 되며, 개발자에 의해 직접 제공되어야 한다. 둘째, 기존 컴포넌트가 가지는 IDL(Interface Definition Language) 인터페이스를 갖는 새로운 어댑터 컴포넌트를 구현하고, GPP에 추가로 적재하여야 한다. 어댑터는 장치 드라이버를 통해, 수신한 요구와 데이터를 FPGA상의 로직에 전달한다. 셋째, 로직은 보드의 데이터/제어 메커니즘에 따라 시그널을 수신하고 처리한다. 이러한 과정에서 볼 때, 현재의 방식은 GPP에 불필요한 컴포넌트(어댑터)가 탑재되어야 하며, GPP상의 장치와 FPGA상의 로직은 플랫폼의 보드 구성에 의존성을 갖게 된다는 단점이 있다.

이러한 문제를 해결하기 위해, 개발자는 로직 구현 과정에서 칩이나 보드 의존적인 부분들을 분리하여 개발할 수 있어야 하며, 동시에 프레임워크에서 표준으로 사용하는 CORBA IDL을 준수하는 컴포넌트와 로직간의 데이터 교환 및 변환 체계를 제공하여야 한다. 이러한 요구사항은 FPGA에서 CORBA ORB의 기능과 역할을 제공하는 미들웨어인 하드웨어 ORB(이하 HAO, Hardware ORB)를 통해 달성될 수 있다. 현재, HAO는 Virtex 4에서 개발되었으며, 평균 2,900여 로직 셀(logic cell)이하의 크기를 가진다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련연구로서 SCA와 코바 미들웨어에 대해 기술하며, 3장에서는 FPGA를 타깃으로 개발된 코바 ORB인 HAO의 기능과 역할, 그리고 여러 이슈에 대해 기술하도록 한다. 4장에서는 HAO의 성능 분석과 평가를 수행하며, GPP상의 컴포넌트 운영과 이에 대한 로직 수준의 재구현 결과를 비교하도록 한다.

Ⅱ. 관련연구

SCA는 JTRS(Joint Tactical Radio System) JPEO (Joint Program Executive Office)에서 제안되었으며, 소프트웨어 컴포넌트의 이식성을 제공하고 여러 무선체계간의 재구성 능력을 보장할 수 있는 통신을 위한 소프트웨어 구조이다. SCA는 분산 객체 모델의 산업 표준인 코바를 기반으로 하고 있다. 코바는 컴포넌트 개발과정에서 통신체계, 개발언어, 운영체제에 대한 독립성을 제공하며, 프레임워크는 코바의 사용을 통해 컴포넌트의 재사용성과 상호운용성을 보장한다[3,4].

코바는 분산환경(즉, GPP들과 FPGA들로 구성되는)에서 컴포넌트(서버와 클라이언트)에 대한 위치 독립성 및 통신 방식에 대한 추상화를 제공한다. 이를 통해, 클라이언트는 코바 통신 체계를 사용하여 서버가 존재하는 프로세서의 위치에 무관하게 서버와 통신할 수 있다.

코바는 ORB(Object Request Broker)와 IDL (Interface Definition Language)로 대표되는 두 요소로 구성된다. ORB는 클라이언트의 호출을 받은 후, 이 요청을 구현한 서버를 찾고, 이후 매개 변수와 함께 서버의 메소드를 호출하며, 그 결과를 다시 클라이언트에게 반환하는 제반 역할을 수행한다. IDL은 서버와 클라이언트간에 통신하는 방법을 정의한 것이다. 즉, 클라이언트는 IDL로 기술한 인터페이스 정의(ORB가 인식하는 요청과 매개변수들)에 따라 ORB를 통해 서버에게 요구를 제시하면 서버는 반드시 이에 응답하도록 된다.

한편, 무선체계의 규모와 복잡성이 커짐에 따라, 무선 개발자들은 동일한 시간에 다수의 대형 무선체계를 관리해야만 한다. SDR(Software Defined

Radio)에서는 이러한 처리 성능을 만족시키기 위해 GPP, DSP, FPGA의 조합을 사용한다[2]. 즉, 플랫폼에서 엄격한 실시간 처리, 다양한 하드웨어 자원의 활용, 서비스 품질 개선을 위해서는 하드웨어 로직을 직접 사용해야 할 필요성이 있으며, 이들 하드웨어 로직은 VHDL이나 Verilog와 같은 HDL(Hardware Description Language)로 코딩되어야 한다[5]. 하지만, 현재의 SCA 규격에 따르면, 프레임워크에서 GPP 상의 소프트웨어 컴포넌트와 FPGA상의 서버 로직간의 상호 연동은 어댑터 방식을 권고하고 있다[7]. 어댑터 방식에서는, GPP에 서버 로직에 대한 접근용 어댑터를 별로도 두고, 클라이언트는 어댑터와 코바 통신을 통해 요청하고 응답을 받는다. 어댑터는 GPIO(General Purpose Input/Output) 할당과 같은 특정 하드웨어 의존적 구현을 직접 사용하며, 수신한 요청은 FPGA상의 서버에 전달하고 처리 결과를 반환받는다. 어댑터는 이 결과를 다시 코바 통신 체계를 통해 클라이언트에 제공한다. 이러한 방식의 단점은 해당 단말의 하드웨어 자원을 직접 사용함으로 인해 단말의 하드웨어 구성에 의존성을 갖게 된다는 것이다. 결국, 하드웨어 환경의 변경에 따라 하드웨어 컴포넌트의 재사용이 어렵고 프레임워크내 컴포넌트들간의 상호운용성 보장이 가능하지 않게 된다. 또한, 어댑터를 부가적으로 사용함으로 인해 하드웨어 로직의 성능 개선, 서비스 품질 개선 등의 효과를 충분히 발휘할 수 없다. 다른 측면으로, GPP상의 어댑터 대신에 MicroBlaze와 같은 FPGA상의 소프트 코어에 운영체제와 미들웨어의 포팅을 통해 하드웨어 컴포넌트와 로직 수준에서 연동하는 방식도 고려되고 있지만, 결국, 하드웨어 컴포넌트 연동에 대한 추가의 오버헤드라는 점에서 문제 해결이 아니고, 또한 소프트 프로세서에 대한 비용과 규모 측면의 부담으로 크게 선호되지 않고 있다[6].

최근 들어, JTRS JPEO는 DSP와 FPGA에 대한 컴포넌트의 융통성을 보다 확대하기 위해, M-HAL (Modem Hardware Abstraction)을 통해 부분적으로 해결하고자 하는 노력을 진행하고 있다. 하지만, M-HAL은 배포가 제한되어 있으며, 이 기술에 대한 세부적인 부분이 공개되지 않고 있다. 따라서, 이들은 범용 용도의 솔루션이 아니다. 또한 이러한 방식은 SCA의 근간인 COTS의 적극적인 활용 취지에도 일치하지 않으며, 융통성있는 설계 성능의 보장이 어렵고 개발 및 배포의 비용을 감소시키는 것도 아니다. 이에 따라, 보다 근본적으로 시스템의 성능을 향상하고 폭 넓은 하드웨어 프로세서를 대상으로 컴포넌트에게 융통성을 제공하기 위한 고성능의 off-the-shelf 기술을 필요로 하게 되었다. 이러한 새로운 기술의 중요한 부분은 특수화된 프로세서를 위한 표준기반 코바 미들웨어를 제공하는 것이다[2,8].

본 논문은 FPGA에 적용할 수 있는 코바 미들웨어인 HAO에 대해 기술한다. HAO는 개발자에게 보드에서 하드웨어 의존적인 부분에 대한 추상화를 제공하며, 가변성이 필요한 부분에 대해서는 미리 정의된 설정 파라미터로 전달 받아 재구성하도록 한다. 또한, HAO는 FPGA외부와 표준 코바 통신 프로토콜인 GIOP(General Inter-ORB Protocol)를 통해 메시지를 송수신하고, 이 메시지의 해석을 통해 목적 하드웨어 컴포넌트를 식별하고 필요한 데이터 변환을 수행하며, 최종적으로 하드웨어 컴포넌트에 전달하는 과정을 포함한다. 한편, 프레임워크에서 컴포넌트간 연동에 사용되는 데이터/연산은 VHDL(VHSIC Hardware Description Language)과 같은 HDL의 데이터와 호출체계로 사상되어야 한다. 모든 하드웨어 컴포넌트는 GPP상의 컴포넌트와 동일하게 IDL로 정의하도록 하고, 이로부터 데이터 변환 및 하드웨어 컴포넌트 식별을 위한 과정을 갖도록 구조화된다.

이러한 HAO와 IDL2VHDL 컴파일러를 통해, SCA기반 컴포넌트 개발과정에서 GPP와 FPGA상의 모든 컴포넌트들은 IDL 인터페이스에 기반한 상호운용성 보장이 가능하다.

Ⅲ. 하드웨어 ORB(HAO)

FPGA 특성상, GPP상의 소프트웨어 컴포넌트와 달리 컴포넌트의 생성과 동적 연결(dynamic binding) 같은 가변성의 지원이 어려우며, 구현 성능도 효율적이지 않다. 이러한 특성으로 인해, 하드웨어 컴포넌트는 IDL 컴파일 시점에 생성과 연결이 미리 고정된다.

이 장에서는 이러한 특성에 의해 범용의 ORB에 비해 차별화되어야 하는 역할과 HAO의 개괄적인 구조에 대해 기술한다.

3.1 코바 ORB의 기능적 계층

코바는 컴포넌트간 연동 과정을 몇 단계의 계층적 구성을 통해 달성되도록 정의하고 있다. 첫째로, 모든 컴포넌트는 POA(Portable Object Adapter) 계층화를 통해 그룹화 될 수 있으며, 다양한 검색 및 연동 관련된 전략(policy)의 적용이 가능하다. 둘째,

ORB는 (동일 혹은 다른) 프로세서상의 다른 ORB와의 상호 연동을 책임지며, POA를 통해 해당 ORB에 연결된 모든 컴포넌트를 관리한다. 셋째, ORB들간의 요구와 데이터는 GIOP를 통해 이루어진다. 이때, GIOP는 운영체제에 각기 탑재된 ORB간에는 보편적으로 소켓(socket)을 사용하여 구현된 IIOP로 구체화된다. 하지만, 본 논문에서 기술하는 HAO와의 연동을 위해, GIOP는 추가로 보드상의 물리적인 연결인 시스템 버스의 제어를 통해 구체화될 것이다. 그림 1은 통상적으로 컴포넌트간 연동 과정을 보인다.

스켈리톤(Skeleton)은 IDL로 부터 생성되며, 서버 구현(server impl)에 대한 일종의 접근 인터페이스이다. 즉, 스켈리톤은 서버 구현이 제공하는 모든 오퍼레이션에 대한 EPV(Entry Point Vector)를 가지고 있으며, 외부 메시지로부터 요구된 오퍼레이션 이름을 EPV로 사상하기 위한 기능과 외부 메시지에 포함된 데이터에 대한 변환 체계를 갖고 있다. 반면에, 서버 구현은 모든 오퍼레이션에 대응하여 실제적으로 수행되어야 하는 기능을 구체화한 것으로, 개발자에 의해 직접 구현되어야 한다. 스터브(Stub)는 스켈리톤과 연계하여, 실제 구체화된 서버 구현을 호출하기 위한 인터페이스를 제공한다. 즉, 그림에서 client1은 스터브를 사용하여 서버 구현의 구체화 방식이나 위치에 대한 고려없이 호출가능하다. 유사하게, 원격의 클라이언트(client2)가 서버 구현(server impl)을 사용하고자 한다면, ORB(Other ORB)는 GIOP(Other GIOP)를 통해 ORB와 상호 연동할 수 있으며, 이와 관련된 사항은 클라이언트(client2 impl)에게 감춰진다. 이때, 두 GIOP는 TCP/IP나 시스템버스와 같은 물리적인 연결을 사용해 상호 연결된다.

추가로, 스터브/스켈리톤은 소프트웨어 버스를 통한 클라이언트와 서버간 연동에 필요한 모든 인터

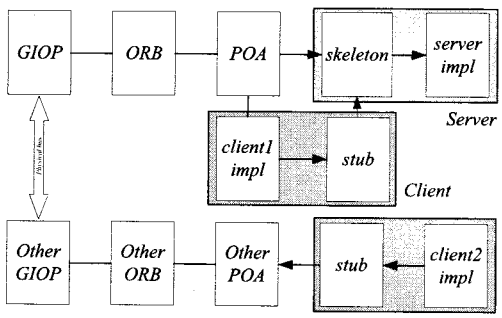


그림 1. ORB 서비스의 계층적 역할

페이스를 제공한다. 구체적으로, 바이트 순서(data endian), 주소 정렬(address align), 전달 메시지 작성/전달/해석 등의 역할을 포함한다. 간단히 정리하면, 서버는 클라이언트의 요구를 처리하는 일종의 서비스 구현이다. 반면에, 클라이언트는 전형적으로 서버의 서비스를 사용해 애플리케이션을 구현한다.

3.2 HAO의 구조

HAO는 FPGA에서 구체화된 일종의 ORB이다. 따라서, 기본적으로 코바에서 제안하는 계층적 구조를 그대로 수용하고 있다. 하지만, GPP상의 ORB와 달리 실행 시점에 POA의 계층구조를 새로이 설정하거나 변경하는 동적인 처리를 제한하고 있다. 즉, HAO는 범용 ORB의 역할에서 동적인 기능(컴포넌트의 동적인 생성, 새로운 ORB 추가 등)은 수행할 수 없다(실제 통신 환경에서는 엄격한 시간제약으로 동적으로 필요한 메모리나 쓰레드를 초기에 모두 생성하고 풀(pool)로 유지한다는 점에서, 통신 소프트웨어 개발 과정에서 큰 제약은 아니다). 따라서, HAO와 하드웨어 컴포넌트는 FPGA에 다운로드되기 전에 바인딩이 결정되어야 하며, 본 연구에서는 이러한 기능들을 대부분 IDL 컴파일 단계에서 수행한다(이에 대한 구체적인 사항은 [9,10]을 참조).

HAO는 세 계층으로 구분할 수 있다. 첫째는 범용 GIOP의 하위 부분으로 물리적인 연결 영역인 Local transport이다. Local transport는 HAO가 적용될 보드의 환경에 의존적인 부분을 포함하며, 최종적으로 보드 독립적인 부분과 분리하기 위해 구성되었다. 두 번째는 범용 ORB에서 GIOP-ORB-POA를 포함하는 HAO Core 영역이다. 이 영역은 통상적으로 개별적인 보드 환경이나 응용 로직을 포함하는 하드웨어 컴포넌트와 독립적인 부분이고 파라미터화되어 있어, 부분적으로 IDL2VHDL 컴파일러를 통해 재구성된다. 세번째는 하드웨어 컴포넌트 부분으로 범용 ORB에서 서버 구현부분이다. 이 부분은 스켈리톤과 개발자에 의해 응용 관점으로 직접 작성되는 순수한 로직의 조합으로 구성된다.

3.2.1 Local Transport

Local transport는 HAO Core와 적용되는 보드 환경에 대한 의존성을 제거하기 위해 필요하다.

그림 2는 local transport의 전형적인 구조이며, 네 개의 블록으로 구성된다.

Local transport의 기본적인 기능은 FPGA외부로부터 요구 메시지를 수신하고, 응답 메시지를 송신하

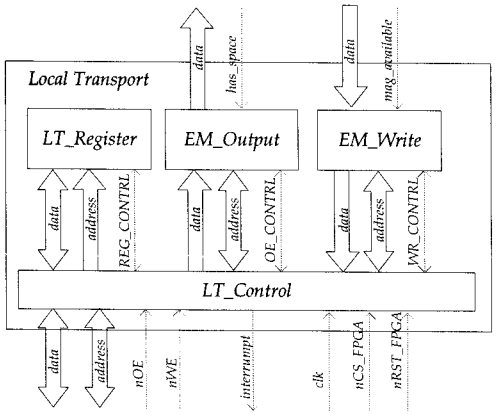


그림 2. Local Transport 블록 구조

는 것이다. EM_output은 외부로부터 GIOP 메시지를 수신하고 이를 local transport와 HAO Core간 중간 버퍼인 FIFO 블록 메모리로 전달한다. EM_write는 HAO Core로부터 받은 응답 GIOP 메시지를 외부에 전달한다. 시스템 버스를 통해 송수신되는 GIOP 메시지는 보드의 확장 메모리(Extended memory) 혹은 FPGA내 블록 메모리에 버퍼링되어야 하며, 버퍼 유지에 필요한 메모리 맵과 인덱스에 대한 정보 등은 lt_register에 포함된다. LT_control은 보드 구성으로부터 GIOP 메시지 송수신에 필요한 버스와 시그널의 제어를 포함한다. 이러한 제어는 보드 구성에 의존적이기 때문에, 보드 구성 변경시 설정 파라미터에 의해 버스의 크기와 시그널의 타이밍 등은 새로이 설정되어야 한다.

3.2.2 HAO core

HAO Core는 로직으로 구현된 전형적인 코바 ORB이다. Local transport와의 인터페이스는 수신 메시지를 전달하기 위한 FIFO 블록 메모리(MSG_Queue)와 송신 메시지를 받기 위한 버퍼로 정의하였다. 이 방식은 물리적인 주클럭의 차이를 해소하고 Local transport-HAO core간의 병행성을 높이기 위한 것이다. HAO Core는 local transport를 통해 수신한 GIOP 메시지를 해석하고, 수행되어야 할 하드웨어 컴포넌트를 인가하며, 하드웨어 컴포넌트에게 데이터를 전달한다. 이후, 그 결과를 수신하는 역할을 수행한다.

HAO core의 블록 구조는 그림 3과 같다. 그림에서, GIOP Interpreter는 GIOP 메시지에 대한 분석을 수행한다. 분석을 통해 대상 객체ID와 오퍼레이션 이름을 Logic selector에 전달하며, Logic

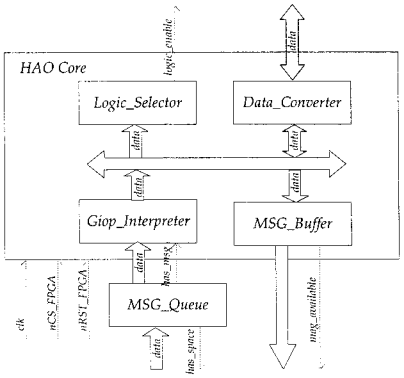


그림 3. HAO core 블록 구조

selector는 이들에 대한 해석을 통해 목적 하드웨어 컴포넌트를 식별하고 이를 인가한다. 하드웨어 컴포넌트를 식별하는 과정에서 상속(inheritance)과 같은 여러 이슈들에 대한 고려를 담당한다. 또한, GIOP 메시지에 포함된 매개변수 데이터들은 Data converter를 통해 추출되고 하드웨어 컴포넌트에 전달된다. 이때, 하드웨어 컴포넌트와 HAO Core간의 버스연결은 최적화되어야 하며, 중첩된 데이터 구조(structure, sequence, array 등)와 공유 속성에 대한 버스 구성 및 설정 등을 담당한다. MSG buffer는 송신할 GIOP 메시지를 구성하기 위해 임시적으로 사용되며, HAO Core에서 송신 처리는 수신 처리에 비해 우선하므로 소규모 버퍼로 구성된다.

3.2.3 하드웨어 컴포넌트

하드웨어 컴포넌트(Hardware Component)는 HAO Core와 인가 시그널과 데이터 버스로 연결된다. 하드웨어 컴포넌트는 IDL2VHDL 컴파일러에 의해 자동 생성된 Hardware logic과 개발자가 직접 작성해야 하는 Developer logic으로 구성된다.

Hardware logic은 수신한 GIOP 데이터를 VHDL 데이터로 변환하여 FPGA 사용자가 개발한 developer

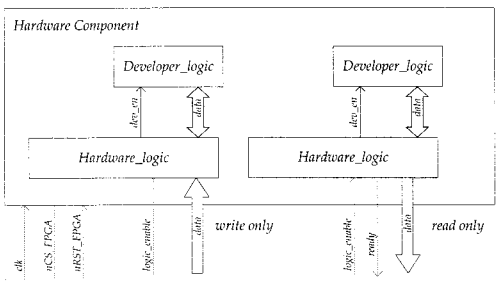


그림 4. 하드웨어 컴포넌트 블록 구조

logic에 전달하고, 해당 로직에 전원을 인가한다. hardware logic과 HAO Core간의 데이터 버스는 두 가지 방식을 함께 사용한다. 모든 고정길이의 데이터는 바이트 정렬(alignment)로 조정하여 버스에 직접 포함하며, 가변길이 데이터는 별도의 블록 메모리에 저장하고 그 주소와 길이만 버스에 포함하여 한 클럭 주기내에 전달한다. 이는 라우팅 부하를 최소화하기 위해 HAO Core와 하드웨어 컴포넌트간의 버스가 최적화되어야 하기 때문이다.

한편, 하드웨어 컴포넌트의 경우, FPGA내 존재하는 또 다른 하드웨어 컴포넌트에 대한 호출은 두 가지 방식이 가능하다. 먼저, 통상적으로 ORB를 통한 전송을 사용하는 방식이 있으며, 다른 방식으로, 목적 하드웨어 컴포넌트에 대한 직접적인 인가를 통해서도 가능하다. 두 가지 방식이 모두 가능하나, 전자의 경우 GIOP 메시지를 구성하고, 메시지 버퍼의 대기시간, GIOP 수신후 해석 등의 과정에 많은 클럭이 낭비될 수 있다. 따라서, 지극히 높은 수준의 독립성이 요구되지 않는다면, 원본 하드웨어 컴포넌트에서 직접 인가하는 방식이 보다 효율적이다.

Ⅳ. 처리 시나리오

이 장에서는 전형적인 코바 처리 과정을 통해 HAO의 역할과 하드웨어 컴포넌트와의 연계에 대해 구체적으로 기술하도록 한다.

코바에서 컴포넌트간 연동은 다음의 단계로 설명할 수 있다. 첫째, HAO는 ORB간 전달 체계를 인식할 수 있어야 한다. 그 결과, HAO는 ORB간 표준 전달 체계인 GIOP를 지원하며, 시스템 버스와 같은 하드웨어 자원을 직접 사용하여 구체화한다. 이외 하드웨어 의존성을 갖는 모든 부분들은 HAO를 통해 파라미터화되어야 한다. 이를 통해, SCA의 모든 컴포넌트는 하드웨어 의존성을 갖지 않도록 추상화할 수 있다. 둘째, HAO는 전달 체계를 통해 수신한 클라이언트의 요구가 실제 처리되어야 하는 하드웨어 컴포넌트를 결정하고, 필요한 데이터를 준비한 후 이들에 전달한다. 셋째, HAO에 의해 유일하게 인가된 하드웨어 컴포넌트는 전달받은 데이터를 hardware logic용 데이터(예, VHDL 데이터)로 변환한다. 넷째, 하드웨어 개발자가 작성한 hardware logic에 이 데이터를 전달하고 인가하여 직접 수행된다. 마지막으로, 수행 결과는 역순으로 최초 요구 컴포넌트에 전달된다.

4.1 컴포넌트로의 변환

코바 환경에서 개발자는 요청과 응답 관점으로 인터페이스를 정의한다. 인터페이스 정의는 표준인 IDL을 사용한다. 모든 코바 컴포넌트간의 연동 규격은 코바 IDL을 통해 기술되어야 하기 때문이다[4]. 본 논문에서 제안한 HAO에 탑재될 하드웨어 컴포넌트의 연동 규격도 코바 IDL로 정의된다.

정의된 인터페이스는 IDL2VHDL 컴파일러에 의해 HAO Core와 하드웨어 컴포넌트내 추가될 VHDL 코드들로 변환된다. 변환된 VHDL 코드들은 Logic selector, Data converter, 그리고 hardware logic에 부분적으로 삽입된다. 또한, 컴파일 과정에서 생성된 최적화 및 각종 설정 정보들은 설정 파라미터(Configuration Parameter)를 통해 HAO에 제공된다(이에 대한 자세한 설명은 [9]를 참조). 이들은 모두 HAO Core에 추가되어 합성된다.

IDL2VHDL 컴파일러에 의한 IDL로부터 VHDL로의 기본적인 변환은 표 1과 같다.

IDL에서 VHDL로의 변환 규칙에서 'interface'는 VHDL의 'entity(A)'로 변환된다. 'operation'은 A의 하위 'entity'로 변환되며, 반환 값의 유무와 파라미터의 입출력 형태(in, out, inout)에 따라 한 개 혹은 두 개의 하위 'entity'로 나뉘어 변환되어 각각 Hardware logic에 포함된다. 'attribute'는 'entity'내의 레지스터 혹은 독립된 메모리 블록을 제어하는 하위 'entity'로 변환될 수 있다(5절 참조). 이들은 HDL 합성 도구에 의해 합성되고 변환된 후 FPGA 이미지로 생성된다.

한편, FPGA의 특성상, IDL의 컴파일 과정에서 'interface'의 개수와 바인드될 HAO(소프트웨어 컴포넌트의 경우에는 생성 시점에 결정) 등이 미리 결정되어야 한다. 즉, 하드웨어 컴포넌트에 대한 식별 처리가 필요하며, IDL 컴파일 과정에서 해결되어야 한다. 이에 따라, 보드내 모든 FPGA내 각 HAO에 대해 가상 식별자를 부여하고 이를 바탕으로 모든 하드웨어 컴포넌트에 대해 정적으로 IOR(Interoperable Object Reference)을 할당한다. IOR은 연결 가능한 모든 하드웨어 컴포넌트를 유일하게 구분할 수 있는 식별자이다. 따라서, 하드웨어 컴포넌트에 대한

표 1. IDL to VHDL 변환 규칙

IDL	VHDL
Interface	Entity(A)
Attribute, Operation	Entity(A의 sub entity 혹은 A와 동등한 수준의 entity)
parameter	Entity간 데이터 버스
호출	logic 인가시그널

표 2. 하드웨어 컴포넌트의 객체식별자 구성방법

객체식별자 영역	
인터페이스 식별자	인스턴스 식별자

요구 메시지에는 해당 하드웨어 컴포넌트의 IOR로부터 '객체식별자'를 추출하여 포함하여야 한다. 객체식별자 크기는 IDL 컴파일 과정에서 모든 하드웨어 컴포넌트를 대상으로 고정된 최소 비트로 결정되고 설정 파라미터를 통해 HAO에 제공된다. IDL2VHDL 컴파일러에 의해 생성되는 하드웨어 컴포넌트의 객체식별자는 표 2와 같이 구성된다.

'인터페이스 식별자'는 FPGA에 활성화될 전체 하드웨어 컴포넌트를 범위로 갖는 비트열(bit sequence)로 구성되며, '인스턴스 식별자'는 하드웨어 컴포넌트중 최대 중복 개수를 범위로 갖는 비트열(bit sequence)로 구성된다. IDL2VHDL 컴파일러는 이러한 최소화된 객체식별자의 크기와 생성 규칙 등을 설정 파라미터를 통해 HAO에 제공한다. 예를 들어, 인터페이스 정의내 8개의 'interface'와 각각 2개의 중복 생성이 허용된다면, 표 2의 '객체식별자'는 5비트 크기로 설정된다. 만일, 모든 'interface'에 중복생성이 허용되지 않는다면, 4비트 크기만을 필요로 한다. 이를 통해 제어 버스 크기를 최소화할 수 있다. 이들 정보들은 설정 파라미터로서 HAO에 제공된다.

이외에도 IDL2VHDL 컴파일러에 의해 생성된 Logic selector는 'operation'에 반환 결과의 유무와 파라미터의 입출력 형태에 따라, 필요한 로직을 인가하고 순서를 제어하는 로직을 생성한다. 예를 들어, 반환값이 있다면, 결과의 반환을 요청하는 hardware logic을 추가로 인가하고 반환할 때 까지 대기한다(그림 4).

Data converter의 경우에는 Logic selector에 의해 결정된 하드웨어 컴포넌트로(혹은 로 부터) 전달할(혹은 수신한) 데이터로 변환하기 위한 코드를 포함한다. 데이터의 변환은 HAO와 하드웨어 컴포넌트간의 데이터 버스를 최적화하기 위해 필요하다(이에 대한 설명은 4절에서 설명).

하드웨어 개발자는 설정 파라미터와 이러한 블록들에 대한 고려없이, 오직 hardware logic을 통해 전달받은 VHDL 데이터와 제어 시그널만을 참조하여 요청된 요구에 대응하여 실제로 수행해야할 기능인 developer logic의 개발만을 담당한다. 작성된 developer logic은 이전에 생성된 hardware logic과 함께 하드웨어 컴포넌트로 구성되고, 자신이 바인드되어야 하는 HAO Core와 버스를 통해 통합된다. 이후, 합성과 FPGA에 다운로드 과정을 거쳐 코바 영역내에 포함된다. 즉, 로직으로 구현하여 HAO에 탑재된 서버 컴포넌트로서 역할을 수행하게 된다.

4.2 GIOP에 의한 요구 메시지 구성

하드웨어 컴포넌트에 대한 요구 메시지는 ORB간 표준 전달 프로토콜인 GIOP 형식으로 구성된다. 요구 메시지는 시스템 버스를 통해 전달되므로, 모든 메시지는 보드상의 메모리(혹은 FPGA내 블록 메모리)에 버퍼링되며, 개별 HAO는 Local transport를 통해 자신의 메모리 영역에 포함된 메시지를 추출한다. Local transport는 할당 메모리의 주소, 주클럭, 버스 제어와 같은 모든 보드 의존적인 특성들을 추상화한다. 따라서, HAO Core는 local transport를 오직 GIOP 메시지에 대한 입출력 블록으로 본다.

표 3은 전형적인 요구 메시지로, GIOP 1.0 규격에 따른 예이다.

요구 메시지는 HAO Core에서 부분적인 해석이 수행된다. 요구 메시지에서 '객체 식별자' (01000000)는 하드웨어 컴포넌트를 구분하기 위해 IDL2VHDL 컴파일러에 의해 할당된 ID이며, '연산 이름'("echo")은 하드웨어 컴포넌트 내의 특정 hardware logic을 인가하는데 사용될 것이다. '데이터'들은 hardware logic이 인가된 후 developer logic의 처리과정에서 필요로 하는 데이터 값들이다(즉, "echo"의 파라미터들로 100, "input").

4.3 하드웨어 컴포넌트 식별 및 버스 구성

Logic selector는 local transport를 통해 수신한 요구 메시지에 포함된 '객체식별자'와 '연산이름'으로부터 수치화된 식별자를 생성하여, 최종적으로 해당 연산의 기능을 구현한 실제 하드웨어 컴포넌트내 hardware logic을 인가(enable)한다.

표 3. GIOP Request 메시지 구성

요구메시지					
47	49	4F	50		"GIOP"
01	00	01	00		1.0, little_endian, req.
34	00	00	00		길이(=52)
00	00	00	00		Service context
E0	F1	12	00		Request #(=1241560)
01	00	00	00		Response expected
04	00	00	00	0x01000000	객체식별자(Object key)
05	00	00	00	"echo"	길이(=5), 연산이름
00	00	00	00		Default principal
64	00	00	00		데이터(short=100)
06	00	00	00	"input"	데이터(str="input")

표 4. 설정 로직의 인터페이스 구조

내용	크기
인가 시그널	객체식별자 + 연산식별자(bit 단위)
축소된 파라미터 데이터	전체 파라미터 크기 합(byte단위) + 복합 타입에 대한 고려

또한, 인가된 hardware logic은 수행에 필요한 파라미터들을 추출하여야 한다. 요구 메시지내 데이터들은 CDR(Common Data Representation)에 의해 바이트 정렬(endian), 주소 정렬(address align)되어 있다[4]. 따라서, 다수의 패딩 데이터가 포함되어 있어, 이러한 패딩 데이터 제거가 필요하다. Data Converter는 이러한 데이터 변환 과정에서 데이터를 최소 크기로 구성된 데이터 버스를 통해 hardware logic에 전달되어야 한다. 이는 FPGA의 제한된 자원(전원, bus, multiplexer 등)들에 대해 사용량을 최소화하기 위해 필요하다. 이를 위해, IDL2VHDL 컴파일러는 인터페이스에 정의된 전체 오퍼레이션을 대상으로 인가 시그널, 그리고 파라미터들에 대한 바이트 단위의 최적화와 이에 따라 데이터 축소를 수행하는 데이터 변환과정을 제공한다. 통상적으로, 제어 및 데이터 버스에 대해 표 4의 정보를 제공한다.

표 4에서 '객체식별자'와 '연산식별자'는 설정 파라미터에 의해 정의된 비트단위의 크기이며, 대상 HAO상의 모든 '인가 시그널'에 대해 공통이다.

한편, 인가 시그널은 코바 IDL의 상속과 같은 오퍼레이션의 가변성을 고려하여야 한다. HAO에서는 가용한 오퍼레이션이 IDL 컴파일 과정에서 미리 고정되어야 하기 때문에, 상속에서 오버로딩은 지원하지 않으며 오버라이딩에 대해서만 허용하도록 제한된다. '축소된 파라미터 데이터'는 전체 파라미터의 바이트 단위의 크기의 합이며, 앞의 예인 경우, 6바이트(short길이(2) + 문자열이 저장된 블록 메모리 주소(2) + 길이(2))이다.

그림 5는 HAO Core에서 요구 메시지의 처리과정이다.

Logic selector는 객체식별자와 오퍼레이션의 해석을 통해, hardware logic을 인가하기 위한 제어 버스인 logic_enable이 설정된다. hardware logic의 실제적인 인가에 앞서, data converter는 logic_enable에 따라 버퍼로부터 메시지내 해당 파라미터들에 대한 추출 및 변환이 먼저 수행되어야 한다. 데이터 변환 과정에서 복합 타입의 경우 추가의 블록 메모리 접근이 필요할 수 있어, 보다 긴 주기의 클럭이 필요하다. 이후, logic_enable은 데이터 변환 과정이 완료되는 시점에 해당 hardware logic을 실제로 인가한다.

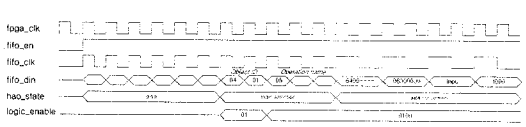


그림 5. HAO Core에서 요구 메시지 처리

4.4 하드웨어 컴포넌트와의 데이터 교환

앞서 예와 같이, SCA는 컴포넌트간의 데이터 교환에 복합 타입(composite type)을 사용되도록 정의하였다. 복합 타입은 IDL 컴파일 시점에 크기와 범위를 알 수 없다. 따라서, 하드웨어 컴포넌트가 복합 타입의 파라미터를 수신해야 할 경우, Data Converter는 자체의 블록 메모리인 DataBuffer에 가변 길이의 복합 타입 데이터를 저장하고, 이후 하드웨어 컴포넌트에 의해 접근할 수 있도록 한다.

변환된 파라미터들은 데이터 버스(data_bus)를 통해 하드웨어 컴포넌트로 전달된다. 크기와 범위가 결정되어 있는 기본 타입 데이터는 해석과 동시에 data_bus에 설정되지만, 복합 타입의 가변 길이 데이터는 메모리 블록인 DataBuffer에 저장하고 data_bus에는 주소와 크기만을 싣는다.

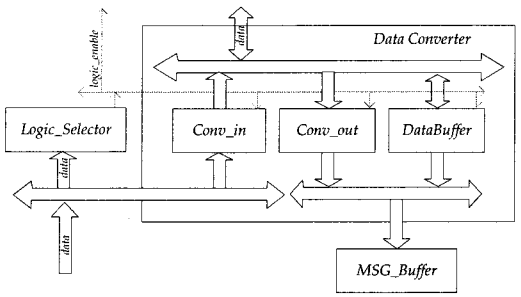


그림 6. 데이터 변환기

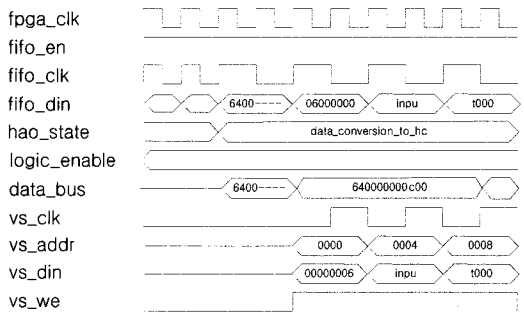


그림 7. 데이터 변환기의 타이밍 다이어그램

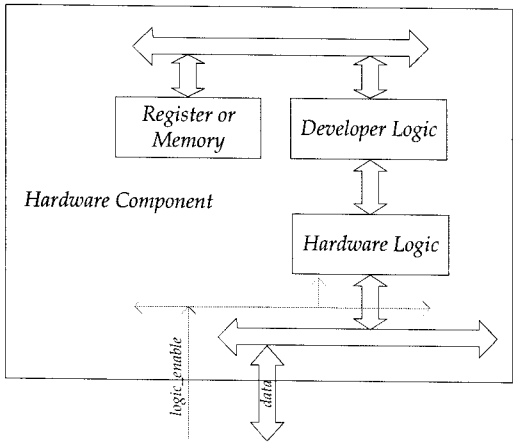


그림 8. 공유 구조를 갖는 하드웨어 컴포넌트 구성

4.5 하드웨어 컴포넌트에서 처리

하드웨어 컴포넌트는 HAO Core와 제어 시그널(logic_enable)과 데이터 버스(data_bus)를 통해 연결된다. 이때, 버스에 실린 '데이터들'은 통상적으로 VHDL이나 Verilog를 사용하여 구현되는 개발 로직에서 수행될 수 있는 시그널 데이터 혹은 필요에 따라 IDL-VHDL간 데이터 사상 규칙에 의해 변환되어야 한다. hardware logic은 이에 필요한 전처리과정을 수행한다. 이후, 일종의 서버 구현인 developer logic에게 제어와 변환된 데이터가 전달되며, 미리 설계된 기능과 역할을 수행하게 된다. 한편, 하드웨어 컴포넌트 내에 공유되어야 하는 특성들이 존재할 수 있다. 'interface'에 정의된 'attribute'가 대표적인 예이며, 모든 'operation'에 의해 공유되어야 한다.

HAO에서 공유 특성은 기본적으로 하드웨어 컴포넌트의 내부 register로 구체화된다. 이 register를 사용하는 'operation'의 구체화인 developer logic은 레지스터와 연결되도록 추가로 확장하여 재정의하여야 한다. register의 구축에 있어, FPGA에서 제공하는 다양한 메모리 구성 방식을 사용할 수 있으며, 이에 대한 선택적인 적용이 가능하다. 기본적으로, IDL2VHDL 컴파일러는 이에 대한 템플릿을 작성해 주지만, 이에 대한 구체적인 활용을 위해 개발자에 의해 보완되어야 한다. 또한, 오퍼레이션 상속과 마찬가지로, 공유특성에 대해서도 상속에 대한 고려가 필요하다. IDL2VHDL 컴파일러는 상속관계가 존재하는 'interface'들을 모두 하나의 주 VHDL 블록(Entity)으로 구성하는 방법을 통해 공유 특성의 상속을 구현한다(이에 대한 전략 및 선택에 대한 구체적인 방법은 [9]를 참조).

표 5. 응답 메시지 구성

응답메시지					
47	49	4F	50		"GIOP"
01	00	01	01		1.0, little_endian, reply
1C	00	00	00		길이(=28)
00	00	00	00		Service context
E0	F1	12	00		Request #(=1241560)
00	00	00	00		No exception
C8	00	00	00		Return data(=200)
07	00	00	00	"output"	Return data(="output")

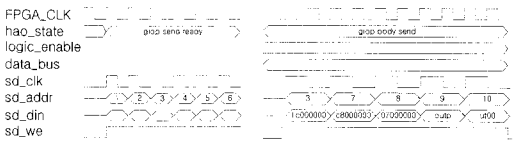


그림 9. 응답 메시지 처리 과정

이후, developer logic의 수행이 완료된 후, 결과 시그널들은 다시 HAO Core를 통해 송신하기 위해 '데이터들'로 변환되어야 하는 후처리가 필요하다. 즉, 수신과 마찬가지로 복합 타입의 데이터가 있다면, 이들 데이터를 DataBuffer에 저장하고, data_bus에는 이들 데이터가 존재하는 데이터에 DataBuffer의 주소와 크기를 싣고, 즉시 데이터 반환 준비 완료 시그널을 활성화한다(그림 6).

4.6 GIOP 응답

HAO Core가 반환 준비 완료 시그널을 수신하면, GIOP에 의해 정의된 인터페이스에 따라 응답 메시지를 구성하게 된다. 표 5는 표 3의 요구 메시지에 대한 응답 메시지 구조이다.

요구 메시지에서 응답이 요구된 경우(response_expected=0x01), 응답 메시지는 두 단계로 구성된다. 응답 메시지의 헤더를 구성하는 전반부에, 메시지의 페이로드를 구성하는 후반부로 나뉘어 구성된다. 페이로드의 길이는 가변적일 수 있기 때문에 후반부에서 수정된다.

응답 메시지는 HAO Core와 local transport를 통해 보드상의 확장 메모리에 적재되며, 상대 ORB에 의해 수신된다. 응답이 요구되지 않는 경우에는 이 과정을 생략한다.

V. 성능평가

이 장에서는 GIOP 요구 메시지에 대해, HAO를 통한 하드웨어 컴포넌트로 부터 응답 메시지 도착까

지를 처리 시간으로 하여 HAO의 성능을 평가한다.

5.1 시험 환경

HAO의 성능은 GPP와 FPGA를 함께 탑재하고 있는 상용 보드를 사용하였다. GPP는 PXA272 processor (520MHz)이며 Linux(Kernel v2.6.x)을 사용한다. 아울러, GPP에 통신용 고성능 코바인 UniORB[11]을 탑재하고, 추가로 보드의 시스템 버스를 통해 FPGA상의 HAO와 메시지 송수신을 위한 장치 드라이버가 함께 탑재된다.

메시지의 송수신을 위한 메시지 버퍼는 보드상의 SRAM(4MB, 70ns)을 사용한다. 버퍼는 환형큐로 유지되며, 송신 버퍼와 수신 버퍼는 별도로 구성된다. 각 메시지 버퍼는 2MB의 크기로 구성되었다.

FPGA는 Virtex-4의 보급형 모델인 XC4VLX60을 사용하며, 보드상의 SRAM(4MB, 70ns)과 32bit 시스템 버스를 통해 메시지 송수신하도록 local transport가 재구성되었다.

HAO는 Local transport를 통해, 송신메시지가 있을 경우 송신메시지를 송신 메시지 버퍼(SRAM)에 저장하고, GPP GPIO를 통해 인터럽트를 발생시킨다. 송신 메시지에 대한 처리가 종료한 후, Local transport를 통해 수신 메시지의 존재유무를 검사한다. GPP가 시스템버스를 통해 송신메시지를 처리하고 있다면, 다음 시스템버스 유휴시에 메시지의 수신 처리가 진행된다.

5.2 평가 방법

트랜잭션은 IOR을 인식하고 있는 하드웨어 컴포넌트 성능을 계산하기 위해, 파라미터 값 설정, FPGA상의 하드웨어 컴포넌트 호출, 결과값 검사, 처리시간을 누적하는 기본 단위이다. 성능 평가는 트랜잭션을 반복적으로 수행하고, 처리 시간의 평균을 해당 트랜잭션의 성능으로 한다.

하나의 트랜잭션은 하나의 기능만을 평가 대상으로 하며, HAO내 하드웨어 컴포넌트는 GPP상의 소프트웨어 컴포넌트와의 1:1연동만을 평가의 대상으로 한다(즉, HAO내 하드웨어 컴포넌트에서 또 다른 하드웨어 컴포넌트와의 연동은 평가에서 제외한다). 이는 하나의 하드웨어 컴포넌트 처리를 위해, 순수하게 HAO에서 소요되는 시간만을 평가 지표로 하기 위해서이다.

본 성능 평가에서는 커널레벨에서 UniORB[11]상의 두 소프트웨어 컴포넌트(클라이언트와 서버)간 트랜잭션 처리 성능, 그리고 GPP상의 컴포넌트와 FPGA상의 하드웨어 컴포넌트간 트랜잭션 처리 성능을 비교하도록 한다.

표 6. 성능 평가 환경

모듈	항목	내용
GPP	Processor	RVPXA272FC0_520(Intel XScale), RISC, 32bit, 520MHz
	GPIO	MBREQ, MBGNT
FPGA	Chip	XC4VLX60-FG668, Logic cells; 55,904(약 5M gate)
	pkg.	FG668(448 I/O)
	clock	Main OSC 50MHz(20 ns)
Base board	RAM	K6F1616T6B-F x 2ea(samsung), 4MB, 32bit, 70ns

5.3 성능 평가 결과

본 평가는 GPP상의 ORB와 HAO간 1:1컴포넌트 연동으로 시험하였다. GPP상의 서버 소프트웨어 컴포넌트에 대한 성능 측정은 서버측에서 요구 메시지를 수신한 시점부터 서버 소프트웨어 컴포넌트가 GIOP 응답 메시지 구성을 완료한 시점(응답 메시지 전송전)까지 이다. 반면에, HAO의 성능을 보다 엄격하게 측정하기 위해, 하드웨어 컴포넌트에 대한 성능 측정은 GPP상에 적재된 장치 드라이버에서 송신 메시지 버퍼(SRAM)에 요구 메시지를 송신하기 전부터 수신 메시지 버퍼(SRAM)으로부터 응답 메시지 수신을 완료한 시점까지이다.

데이터에 대한 다양한 타입의 조합이 가능하다. 따라서 SCA 코어 프레임워크에서 사용되는 모든 호출 인터페이스를 사용하여 시험하였으며, 여러 사례 중에서 가장 대표적인 사례들에 대해 정리하였다.

다음 표는 앞서 언급한 구간의 성능을 정리한 내용이다. 트랜잭션은 기본 타입과 복합 타입의 데이터를 전달하고, 동일 데이터를 다시 수신하도록 구성되었다.

표에 따르면, 기본 타입의 고정길이 데이터 처리에 대한 성능 개선 효과에 비해 복합 타입의 가변길이 데이터가 포함된 경우에 보다 큰 향상요인이 있음을 알 수 있다. 이는 C++의 언어적 특성으로 복합 타입에 대한 마샬링/언마샬링 오버헤드가 로직으로의 구현에 비해 큰 것이 원인이다. 또한, 객체

표 7. HAO의 처리 성능 비교표(단위: us)

파라미터	GPP ORB	HAO	비고
long(1), string(1)	2,110	80	26.39
char(1), short(1), long(1), long long(1)	2,311	90	25.68
str(1), sequence(2)	2,400	70	34.30

식별자의 구조 최적화와 오퍼레이션의 해석 과정의 단순화를 통해, 컴포넌트 검색과 활성화에 소요되는 오버헤드를 크게 감소시킨 결과이다.

한편, 이러한 처리 성능에 대한 추가의 성능 요인이 있다. 보편적으로 실제 코바 환경에서, 하드웨어 컴포넌트는 역시 다른 하드웨어 컴포넌트에게 처리 요구를 보내고 그 결과를 자신의 처리에 사용한다는 점에서, 내제된 하드웨어 컴포넌트에서 오는 추가의 처리 성능상의 이점은 보다 크다. 따라서, 실제 SCA기반 개발에서 HAO용 하드웨어 컴포넌트의 처리 성능은 소프트웨어 컴포넌트와 하드웨어 컴포넌트간의 1:1성능 비교에 비해 훨씬 큰 추가 향상 요인을 가질 것이다. 현재, SCA 미들웨어 관점에서 이에 대한 정량적 평가가 진행되고 있다. 또한, HAO는 GPP용 ORB에 비해, 개별 하드웨어 컴포넌트들이 상호 경쟁없이 물리적으로 병행하여 수행될 수 있고, 계산 모델에 따라 종료 시간을 정확히 예측할 수 있어, 엄격한 실시간 처리 등이 요구되는 응용에 적합하다[11].

VI. 결 론

SCA는 JTRS JPEO에 의해 하나의 플랫폼에서 여러 무선체계를 운용하기 위해 제안되었으며, 소프트웨어 컴포넌트들에 대해 플랫폼 독립성을 보장하기 위해 코바 미들웨어를 채택하고 있다. 한편, 단말에서 소프트웨어 컴포넌트가 FPGA상의 로직 레벨에서 재구현 요구가 확대됨에 따라, 코바 수준에서 FPGA를 포함한 하드웨어 보드에 대한 독립성의 보장이 요구되었다. 이에 따라, 하드웨어 보드에 의존적인 특성들을 추상화하고, 소프트웨어 컴포넌트에 대한 IDL기반 연동 인터페이스를 제공할 수 있는 하드웨어 미들웨어의 필요성이 대두되었다.

이를 위해, 본 논문에서는 FPGA에서 로직 수준으로 구현한 ORB인 HAO의 개발에 대해 기술하였으며, (1) HAO는 하드웨어 보드에 대한 독립성을 보장하는 local transport와 (2) GPP용 ORB와 동일하게 코바 IDL에 의한 연동을 제공하는 HAO Core로 구성된다. (3) HAO는 평균 2,900 로직셀 크기의 초경량 ORB로 구성되었다. 또한, (4) 소프트웨어 컴포넌트 대비 수십배의 성능 개선을 보였으며, 하드웨어 컴포넌트에 대한 다양한 타이밍 분석과 충돌 해결, GIOP 처리 등의 기능을 보장하고 있다.

무선체계의 개발과정에서, HAO의 활용은 기존 방식에 비해 많은 장점을 제공할 수 있다. 모든 컴포넌트는 컴포넌트의 위치와 인터페이스 변경에 대한 어떠한 고려도 없이 개발중(혹은 배포후)의 어느 시점에서도 분할되어 시험되고 통합될 수 있다. 또한, FPGA 컴포넌트에 대한 접근 인터페이스가 제공되므로 어댑터가 더 이상 필요하지 않으며, 단지 컴포넌트의 고유 기능 개선에만 집중할 수 있다. 다음으로, 특정 GPP 컴포넌트를 FPGA 컴포넌트로의 전환도 용이하며, 단지 마이그레이션될 코바 오퍼레이션에 대한 특정 구현(C++ 혹은 VHDL)에만 집중할 수 있다는 점이다. 결론적으로, HAO의 활용을 통해, 무선체계에 대한 신뢰성과 기존 알고리즘로직간의 호환성을 증가시킬 수 있으며, 표준기반 구조를 통해 이들은 고성능, 저전력, 작은 footprint 등의 장점을 달성할 수 있다.

참 고 문 헌

[1] Yu Helen, "Build and Optimize a MicroBlaze Soft-Processor System Your Way," Embedded Magazine, Xilinx, San Jose, CA pp. 50-52, Sept. 2005.

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

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

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

[5] Xilinx, Virtex-4 FPGA User Guide(UG070) V2.5, June 2008.

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

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

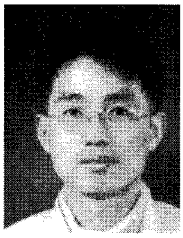
[8] S. Aslam-Mir, "ICO : Integrity Circuit ORB," PrismTech White paper, 2006.

[9] 배명남, 이병복, 박애순, 이인환, 김내수, "SCA에서 C++/VHDL 구현 독립성을 보장하기 위한 코바 미들웨어 확장", 대한전자공학회 논문지, 제46권 TC편, 2009.

〔10〕 장종현, 이동길, 한치문, "개방형 통신 시스템을 위한 고성능, 고신뢰성 CORBA 플랫폼에 관한 연구", 대한전자공학회 논문지, 제41권 TC편, 제 2호, pp. 19-29, 2004.

〔11〕 Eui Hyun Paik, Jong Hyun Jang, Tae IL Kim, and Hyeong Ho Lee, "UniORB: A Priority-Based Middleware for Communication System," PDPTA2003, vol. 4, Las vegas, Nevada, pp. 1557-1560, June 2003.

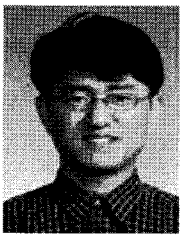
배 명 남 (Myung-Nam Bae) 정회원



1991년 2월 전북대학교 전산통계학과 (학사)
1993년 2월 전북대학교 전산통계학과 (석사)
1998년 2월 전북대학교 전산통계학과 (박사)
1998년~현재 한국전자통신연구원 USN전송기술연구팀 선임연구원

<관심분야> SDR, 통신미들웨어, 개방형 플랫폼, 임베디드 하드웨어, 무선전송

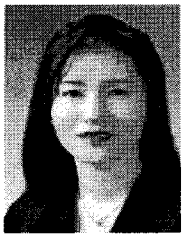
이 병 복 (Byung-Bog Lee) 정회원



1991년 2월 호원대학교 전자계산학과 (학사)
1993년 7월 전북대학교 전산통계학과 (석사)
2005년 3월~현재 고려대학교 전산학과 (박사 과정)
1993년 7월~현재 한국전자통신연구원 USN전송기술연구팀 책임연구원

<관심분야> 이동통신단말기 시스템, 임베드디 시스템 개발 및 실행환경, 무선 센서네트워크 전송기술

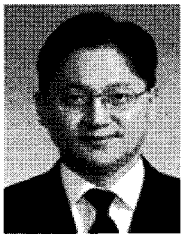
박 애 순 (Ae-Soon Park) 정회원



1987년 2월 충남대학교 전자계산학과 (학사)
1997년 8월 충남대학교 전자공학과 (석사)
2001년 8월 충남대학교 컴퓨터공학과 (박사)
1998년~현재 한국전자통신연구원 차세대이동단말연구팀장(책임연구원)

<관심분야> 4세대 이동통신, 이동통신망, 이동성관리, 이동단말 기술

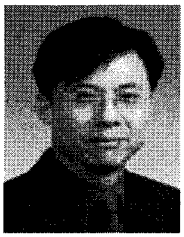
이 인 환 (In-Hwan Lee) 정회원



1988년 2월 한양대학교 전기공학과 (학사)
1990년 2월 한양대학교 전기공학과 (석사)
2007년 3월~현재 한양대학교 전자컴퓨터통신공학과 (박사 과정)
1990년~1993년 (주)동아전기연구원
1993년~현재 한국전자통신연구원 USN전송기술연구팀 책임연구원

<관심분야> RFID/USN, 무선통신

김 내 수 (Nae-Soo Kim) 정회원



2000년 2월 한남대학교 컴퓨터공학과 (박사)
1986년~1990년 국방과학 연구소
1990년 3월~현재 한국전자통신연구원 USN전송기술연구팀장(책임연구원)

<관심분야> RFID/USN, 위성통신, 컴퓨터네트워크