2009-ARC-181／2009-EMB-11 (9)
2009/1/14

Object-Oriented Programming and Testing Environment for an FPGA Using CORBA/GIOP Protocol

CORBA/GIOPを用いたFPGA向けオブジェクト指向プログラミング・テスト環境

Takeshi Ohkawa (大川 猛)
National Institute for Advanced Industrial Science and Technology (AIST),
Information Technology Research Institute (ITRI)
Umezono 1-1-1, Tsukuba Central 2
Tsukuba, Ibaraki
+81-29-861-9146
ohkawa-takeshi@aist.go.jp

Kenji Toda (戸田 賢二)
National Institute for Advanced Industrial Science and Technology (AIST),
Information Technology Research Institute (ITRI)
Umezono 1-1-1, Tsukuba Central 2
Tsukuba, Ibaraki
+81-29-861-5875
k-toda@aist.go.jp

ABSTRACT
A small software and a circuit system are implemented on a Xilinx Spartan 3E FPGA, which handles CORBA (Common Object Request Broker Architecture) / GIOP (General Inter ORB Protocol) message with small runtime memory footprint (about below 16K Bytes) in small circuit size (about below 1500 LUTs). The small circuit system, which is named "ORB Engine", is composed of MicroBlaze soft-core processor provided by Xilinx, UART (Universal Asynchronous Receiver and Transceiver) controller and Interrupt controller. Connecting the ORB Engine's UART port to the PC's serial port, user's circuit resided on the FPGA can be operated from the PC by a simple remote method call. The ORB Engine interprets a method-call request from the PC, set the parameter value to the user's circuit, retrieves the output from the user's circuit and sends the result back. The ORB Engine is an interface for a circuit in an FPGA, easily understood by software engineers. The whole system works as an object-oriented programming and testing environment. The proposed system contributes to design quality of FPGAs in embedded systems.

Keywords
ORB, Object Request Broker, GIOP, FPGA, Testing, Distributed Object

1. INTRODUCTION

1.1 FPGA's Role in Embedded System

FPGA (Field Programmable Gate Array) has become a popular and standard platform to implement a desired logic into a hardwired circuit. By using FPGA, fine customization of an embedded system is enabled by its programmability.

FPGA vendor Xilinx is providing a product named "EDK (Embedded Development Kit) [1]" as an FPGA development environment for embedded systems. In the EDK suite, there are two development environments; "Platform Studio" for hardware design by connecting IP components and "Platform SDK (Software Development Kit)" for software design using C/C++ language in Eclipse [2] environment. The Platform SDK is capable to configure the software in conjunction with the hardware configuration, such as memory mapped address, properly.

Moreover, the logic design can be done even in a high level language of ANSI-C or C-like language, for example, Impulse-C [3]. They spread out the possibility of FPGA-based embedded system design. Ordinary software engineers are expected to be involved into the embedded system design using FPGA.

Meanwhile, micro-controller chips have most generic ready-made interfaces on-chip with very low cost compared to FPGA. For example, a USB port, a Universal Asynchronous Receiver and Transceiver (UART: PC's serial port), CAN, PCI, A/D, D/A. Converter and so on. The micro-controller chips are generally more popular than FPGA in constructing embedded systems.

Embedded systems in the area of advanced applications, however, often require special interface. For example, a rotary encoder is used in robots in order to detect the rotation angle of an axis. Usually it is not covered by the standard micro-controller chips. In the rotary encoder, phase A and phase B signals are used to detect the rotation angle from the state transition of the two signals. If the transitions of the signals are slow, software on a micro-controller also can detect, however, if the speed of transition is high, software would miss tracking the signal. Therefore, a signal edge detection circuit is always dedicated to detect signals watching. In such cases, the traditional way is utilizing the external counter IC; however, FPGA can load every special interface on-chip to build the system.

FPGA in embedded system has an important role in the era of fine customization, in order to add high value to the system,

1.2 What is good for Designing and Testing FPGA for Embedded System?

Development of an embedded system which is composed of a circuit and software needs a "good" programming and testing environment. It is because the circuit and the software are related tightly and affect each other. Especially recent system with multiple processors are making the programming and testing much difficult.

Before going into the detail, we need to define the term "good" is for whom? Assume there are three kinds of engineers or programmers concerned with the FPGA project: A. Hardware Engineer, B. Embedded System Specialist and C. Software Engineer (Fig. 1).

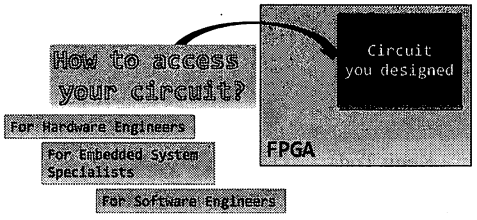

Fig. 1 the suitable method differs among the type of engineers to access the circuit in the FPGA

Next, think that an engineer has a circuit programmed on an FPGA. Candidates of the method to access the circuit are listed below,

1. JTAG interface: to read and write the in-circuit signal register with a special debugging environment, which can treat JTAG.
2. Memory mapped register: connect a CPU directly to the circuit through processor BUS and make the circuit register accessible from the CPU.

Hardware engineers and embedded specialists may choose 1. It is the lowest-level abstraction and is a good method for the early stage of development. Difficult timing problems should be treated at this level. Software engineers have less concern to this level and there is no need to understand the details of the circuits.

Embedded specialists may choose 2. Hardware engineers, too. Software engineers are asked to use this method, too. It is thought to be acceptable to the software engineers; however, progressive software engineers think that C/C++ is a low-level language. Their real intention is that writing a program should be done in highly productive programming languages such like Java, Python or another script language.

Until here, it has become obvious that a good method for accessing the circuit differs among the type of engineer or programmer, and by the abstraction level. Another point is that software engineers has good practices of building software, however, once you are designing an embedded system, C/C++ is still the maximum choice.

As productive design methodology, an object-oriented programming (for example, Java, Ruby) and testing (for example, JUnit) are widely accepted methodologies in the PC domain and the enterprise server domain.

1.3 Utilize Object Request Broker Middleware for FPGA Access

In addition to the programming languages, distributed object middleware like CORBA (Common Object Request Broker Architecture) makes the platform boundary transparent and therefore improves the productivity. For example, an object written in C++ running on a server can be called from a client program on PC written in Java through GIOP (General Inter Object Protocol) over TCP/IP network.

CORBA is at a neutral position from programming languages. IDL (Interface Definition Language) is used to define the interface of the object (Fig. 2). The objects with an interface defined by IDL can interpret the object messages each other, using common protocol (Inter-Operability), even the objects are implemented in any different programming language. Therefore, even if a new language appeared, if the new language adapt to CORBA, all languages adapted to CORBA can understand each other.

Because the platforms of embedded systems are versatile, CORBA is essentially useful for connecting PC to an embedded system / an embedded system to an embedded system. However, it is usually implemented to use TCP/IP socket. And the memory footprint is large up to several mega bytes. Many CORBA implementations for the PC as a target server, with rich functionality is large in size. They are roughly 2 Mbytes to 8 Mbytes (in case of OmniORB [4], MICO [5] and Java[8][9][10]).

Therefore, the authors think that the key is a very lightweight CORBA implementation, which can run on FPGA with limited resource.

```
module robot {
        interface Arm {
                void init();
                unsigned short getAngle();
                void setTorque(in long newValue, out long current);
        };
};
```

Fig. 2 an example of IDL code

2. CONCEPT PROPOSAL

2.1 "ORB Engine" as an Interface of a Circuit

The authors would like to propose a method to access a circuit in an FPGA based on the previous discussion. The purpose is to provide a good tool which would be welcomed from software engineers.

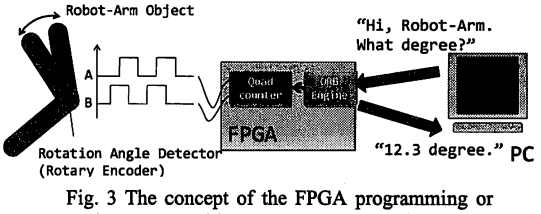

Fig. 3 The concept of the FPGA programming or testing system with ORB Engine using object level message communication

Fig. 3 shows the concept of "ORB Engine", our proposal. The circuit "Quad counter" as an example in the figure is driven by the object method call level message from PC through ORB Engine. That means FPGA programming or testing is done using object level message communication. During the development of embedded systems, instant question for a component is useful. For example, asking Robot-Arm object its angle by sending a message, "Hi, Robot-Arm. What degree?". Then the object will reply "12.3 degree" as shown in the figure.

2.2 Benefit of using CORBA for the Circuit Control

There are many benefits of using CORBA to control the circuit in FPGA.

Firstly, you can use a number of programming languages to control the circuit. Currently there are 10 IDL to language mapping; C, C++, Java, Lisp, Python and so on.

Secondly, various tools for PC software development can be utilized for the circuit control. Especially, a unit test tool JUnit can be used for the circuit function test.

The biggest benefit is that the circuit can have an object oriented interface. You can define an arbitrary interface you need. As the interface is same, the circuit can be replaced to another one. If necessary, you can replace your hardwired circuit into a software model, at the various phase of the development.

Another attraction is that some types of FPGA have dynamic (partial) reconfiguration feature. Dynamic replace of the circuit is possible, without causing the software version mismatch problem.

3. SYSTEM DESIGN

3.1 Purpose of Implementation

To estimate the reasonable size of hardware and software, a working implementation is inevitable. The aim is to make an minimum size system, but answer to CORBA/GIOP messages.

3.2 The Choice of Transportation Channel

Until now, CORBA is developed for TCP/IP on the Ethernet. However, the choice of TCP is not always the best solution.

To satisfy the above noted purpose, the simplest UART is chosen for the implementation. The benefits of using simple UART interface are low-cost, easy-to-use and not-too-slow as allowable.

Fig. 4 shows the detailed architecture of the client/server system. Client is made on PC and Server is made on FPGA. This research's main target is the server on the FPGA, however, to control the server on the FPGA, the client on PC must be prepared.

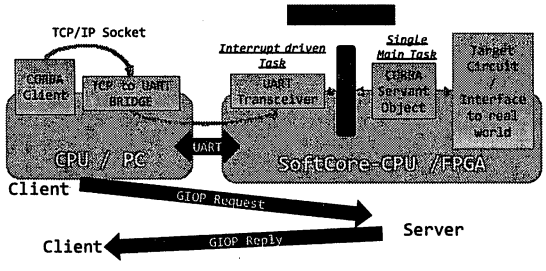

Fig. 4 Detailed CORBA Client / Server system model using UART as a transport channel

3.3 FPGA Design

ORB engine is implemented on Xilinx FPGA Spartan3E-500 as in Fig. 5. The resource usage is shown in Table I. The ORB Engine has only 4 external pins: clk, rst, rx and tx. The message comes from the rx pin and interpreted within the MicroBlaze processor, and makes interaction with the IP core and returns data to tx.

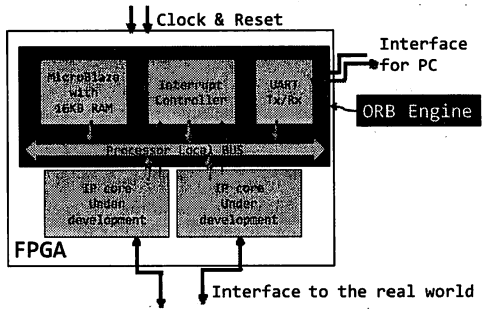

Fig. 5 FPGA Design done in the Xilinx EDK platform studio environment (PLB for Processor Local BUS)

Table 1. Resource usage of the FPGA design (ORB Engine part only)

Resource	Used	Capacity (Spartan3e500)	Ratio (%)
4-input LUTs	1467	9312	15%
IOBs	4	232	1%
Block RAMs	8	20	40%

3.4 FPGA side Software Design

The ORB engine software is implemented in C language using Platform SDK environment. It is not depend on any operating system. Some standard functions and device drivers are from Xilinx. The minimum runtime footprint is about 12KB (the detailed value is available in the following table. 3), in case that there is only one object of an interface with one method, and there is no error handler.

As shown in Fig. 4, there are only two tasks runs on the CPU. One is a interrupt driven task which takes data from the UART transceiver to the shared ring buffer. The other is the main task to interpret the GIOP message stored in the shared ring buffer. Because the UART controller has a FIFO (First-In First-Out) buffer only with the depth of 16, the first task is needed not to overflow the buffer.

3.5 PC side Software Design

At first, "TCP to UART" bridge server is implemented to pass thru the data received from the TCP port to the UART port.

In CORBA, the location of the servant object is designated in various ways. IOR (Interoperable Object Reference) is the most standard way to do it. In our system, CORBA client should make a connection to the "TCP to UART" bridge at localhost. For that purpose, the IOR is generated to designate the listening port of the "TCP to UART" bridge at PC. Using the architecture, an ordinary CORBA can be used without modification to communicate with the ORB Engine after UART channel.

4. EVALUATION

4.1 Experimental Setup

In order to evaluate the MicroBlaze ORB Engine, experimental setups are prepared.

We used the DOPG's interoperability test suite [7] for the measurement. (DOPG stands for Distributed Object Promotion Group). It is originally intended to evaluate the interoperability of various CORBA products. In this experiment, the IDL file of "dopgrm11.idl" is used for the round trip delay time. The remote method call latency measurement is done using the op1 method of the rf11 interface. The IDL of op1 is shown in the Fig. 6

```
interface rm11 {
        short op1(      in short argin,
                        out short argout,
                        inout short arginout);
};
```

Fig. 6 IDL declaration used in the experiments

The version of OmniORB is 4.0.7, MICO is 2.3.12, JavaIDL is JDK1.6.0. OmniORB and MICO are used for a C++ mapping and JavaIDL is used for a Java mapping of IDL.

Experimental setups #1 to #3 are ordinary CORBA usages through TCP/IP connection. Setups #4 to #7 are through UART

The UART's baud-rate is fixed to 115.2 KHz. In UART protocol, the data transfer is done by a 8-bit character in this case. Before the send start, 1bit start bit is added, and after the character 1bit stop bit is added. That is, 10bit is needed to send a character.

#5 and #7 uses TCP to UART bridge process to pass thru the data from TCP port to UART serial port as explained in Fig. 4.

#4 and #6 uses directly modified UART output is used. It means that the source code tcpConnection.cpp of OmniORB is directly

Table 2. Experimental Setup

#	Client	TCP to UART	Transportation	UART to TCP	Server
1	OmniORB (C++)	-	TCP/IP	-	OmniORB (C++)
2	MICO (C++)	-	TCP/IP	-	MICO (C++)
3	JavaIDL (Java)	-	TCP/IP	-	JavaIDL (Java)
4	OmniORB (C++)	Direct modify	UART	Direct modify	OmniORB (C++)
5	OmniORB (C++)	Bridge	UART	-	MicroBlaze ORB engine
6	OmniORB (C++)	Direct modify	UART	-	MicroBlaze ORB engine
7	JavaIDL (Java)	Bridge	UART	-	MicroBlaze ORB engine

4.2 GIOP Messaging Behavior

At first, message transaction between the client and the server is observed. Fig. 7 shows the transaction during the two remote method calls of op1 method. At the first time, Locate Request is issued to ensure the servant object existence. After that, Locate Reply is returned, which says "Object is here". Next, Request is issued from client. Then the op1 operation is executed at the server side, and returns Reply message. At the second time op1 call, no Locate Request message is sent. Because the messaging behavior at the first method call is different from the second one, the round-trip delay measurement is done for the second call.

In addition, the actual data amounts transmitted is counted. In the case of TCP/IP transport shows 66 bytes more data is transmitted. This is due to the TCP/IP header, while UART transmit the bare data.

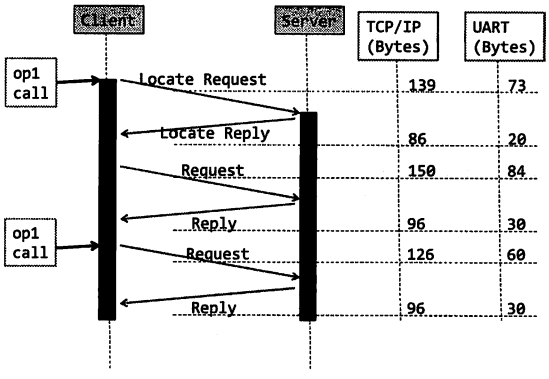

Fig. 7 The GIOP messages exchanged during two op1 calls

4.3 Round-trip Delay

The measured round-trip delay time is shown in Fig. 8.

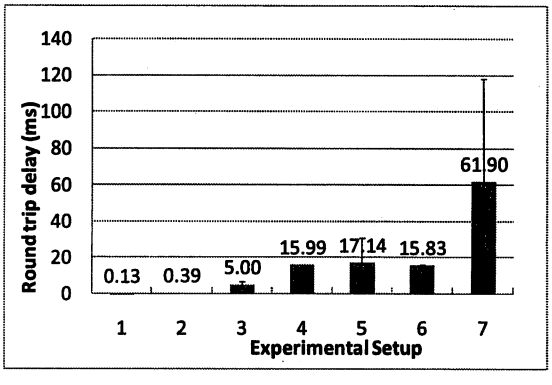

Fig. 8 Round trip delay at a remote method call (average of 4 times remote call)

All the measurement is done for remote method call between separated PCs / FPGA board. Each PC equips a 1000Base-T

Ethernet interface card. OmniORB exhibits the fastest round trip delay among the three full spec CORBA implementations with TCP/IP transportation (#1 - #3).

#4 is a reference setup in order to measure the latency of PC's serial port. It has about 16 ms delay; even a modern PC is used.

#5 and #6 are the round trip delays of the MicroBlaze ORB Engine servant objects. They are almost same latency as #4. That means most of the delay is due to the PC's serial port or the device driver for it. The difference between #5 and #6 is small. The "TCP to UART" bridge caused no serial delay at this point.

#7 shows the latest delay. In case of #7, the bridge and the JavaIDL client is running on the same PC, however, in #5, the bridge and the OmniORB client are running on different PC.

Anyway, the CORBA/IIOP on UART is much slow than TCP/IP, however, the FPGA can behave like CORBA servant. And the delay is enough allowable for the debugging control purpose.

4.4 Memory footprint

Table 3 shows the comparison of memory footprint between the full spec CORBA implementation - OmniORB-4.0.7 and the ORB Engine on MicroBlaze. This shows that the ORB Engine is made from totally different way of implementation. The ORB Engine doesn't understand the entire CORBA message. Only necessary part is implemented. That enables the very small footprint.

Table 3. Comparison of Memory Footprint

OmniORB-4.0.7-i386

Item	Size	
libomniDynamic4.so.0	2,673,871	Bytes
libomniORB4.so.0	1,833,970	Bytes
libomnithread.so.3	27,723	Bytes
Runtime footprint:		
Client	4,032	KBytes
Server	3,984	KBytes

ORB Engine MicroBlaze

Item	Size	
main	10,232	Bytes
Runtime Stack	1,024	Bytes
Server	11	KBytes

5. FUTURE PLANS

5.1 Versatile Transportation Channels

There are possibilities to choose other transportation channels. The candidates are Real-time-Ethernet, FlexRay, CAN, LIN, and Rocket I/O. Because their circuit implementations are mostly provided as a proven and ready-to-use IP cores, the implementation will not take so long time.

5.2 Acceleration of CORBA

In order to accelerate CORBA operation, there is a possibility to use this ORB Engine as a co-processor of a main processor. CORBA's processing has less parallelism. A dedicated MicroBlaze processor for ORB processing as reported in this paper may be enough.

5.3 Security Issue

For a debug purpose, the ORB Engine can be detached prior to the release; however, if the ORB Engine is embedded in a product, the security should be taken into account.

5.4 Error Recovery

The UART cannot recover communication error. For a critical mission, choosing another reliable communication channel is better.

5.5 Code Generator (IDL Compiler)

Currently, the message decoding and encoding are done by manually. That is the location of parameter data is specified by manual. The operation to write some code can be automated using the information from IDL.

5.6 Design shrinking

Instead of the MicroBlaze processor, smaller PicoBlaze processor may be used to drastically reduce the LUT count.

6. DISCUSSION

6.1 "ORB Engine" introduces the Iterative Development into FPGA design

What happens if the "ORB Engine" concept is introduced to the FPGA development in embedded system?

FPGA can be programmed many times. This advantage of "Re-programmability" will change the hardware design process and flow. Current FPGA design is still done in the hardware culture. In recent years, however, the importance of software in embedded systems tends to become larger. Therefore the introduction of software culture will lead to productivity gains.

First change is that the FPGA circuit designers can release small and tentative first release of working implementation quickly. (The term "release" means making his design available to others.) Then he can release the extended features and the performance improvements step by step.

The conventional hardware (ASIC) designer's method is firstly to decide the overall framework, next to do functional partitioning and module partitioning, and then assign a task to each engineer. Therefore the time for the release is relatively long and difficult to estimate the progress.

Additionally, the FPGA circuit designers will be able to accept the customer's changing needs of the market. Because FPGA is re-programmable, while ASIC has no flexibility, designers can realize the idea into processor based on customer's needs agile. Despite the fact that FPGAs are "physically" freely programmable, by the same problem as in software, you cannot change because the failure caused by the change is unpredictable.

One of the answers concluded by software engineering is "Integration of Testing and Implementation." In other words, demanded features are described as a test program. And implementation is done to pass the test program. This "Test First Method" has been recognized as a method which can contribute to improving the quality. Accordingly, "refactoring" is also actively done, which is to improve the implementation and the quality of the source code without changing the interface.

The corresponding development tools to the methodology are widely used in software development. For example, Java has an open source unit test tool "JUnit". For test-driven development (test-first development), of course, for the traditional waterfall

development, it is used effectively. Furthermore, "CUnit" is for C language, "CPPUnit" is for C++ language, "xUnit" is for another language... The effectiveness of test is recognized widely.

However, the FPGA designs for the "integration testing and implementation," the concept of the pervading hard to say. For FPGAUnit, there are no signs of emergence.

The ideal development and testing environment for FPGA in embedded system is; runtime is minimal in memory foot-print. And the test is done in rich development environment where highly productive programming languages and tools.

The ORB Engine may promote the testing of FPGA in software manner.

7. RELATED TECHNOLOGIES

"IONA technology [11]" is a major provider of CORBA products for an embedded market. Their minimum memory footprint is 100Kbytes for client and 150Kbytes for server. However they must be much decent implementation than us.

Prismtech [12] is providing FPGA version of CORBA for military applications and "Objective interface systems" is providing OrbExpress[13]. They both are selling very small footprint CORBA for General Purpose Processors and FPGA version. The performance of the FPGA version is said 100 times faster than GPP version. Their target is mainly SDR (Software Defined Radio) application.

8. CONCLUSION

In this paper, small CORBA/GIOP handling software and circuit implemented on an FPGA are presented. The software runs on Xilinx MicroBlaze soft-core processor. The implementation uses the simplest interface, UART (Universal Asynchronous Receiver and Transceiver) which is connected to PC's serial port. The benefits of using simple UART interface are low-cost, easy-to-use and not-too-slow, and very commonly used.

At first, the software object on the soft-core processor can be easily controlled from the PC. In the case of multiple processors, you can add debugging "physical" UART ports as many as you want because of its low cost. And the designed circuit on the FPGA under development can also be directly controlled from the PC through the soft-core processor.

Finally, a cross development environment in the object-oriented testing manner can be built. The proposed system contributes to add a freedom of choice of programming and testing language, for the wide range of embedded systems.

9. ACKNOWLEDGMENTS

This study was supported by Industrial Technology Research Grant Program in 2007 from New Energy and Industrial Technology Development Organization (NEDO) of Japan.

10. REFERENCES

[1] Xilinx EDK http://www.xilinx.com/

[2] Eclipse http://www.eclipse.org/

[3] Impulse-C http://www.impulsec.com/

[4] OMG (The Object Management Group) http://www.omg.org/

[5] OmniORB http://omniorb.sourceforge.net/

[6] MICO http://www.mico.org/

[7] DOPG (Distributed Object Promotion Group) http://www.dopg.gr.jp/en/

[8] "Memory Utilization Analysis of Java Middleware for Distributed Real-time and Embedded System", R.Qu, S.HIRANO and T.OHKAWA, Parallel and Distributed Computing and Systems (PDCS 2006), November 13-15, 2006,

[9] "Java 2 RMI and IDL Comparison", M. B. Juric and I. Rozman. Java Report, 5(2):36~48, Feb. 2000.

[10] "More Efficient Object Serialization", M. Philippsen, B. Haumacher, IPPS/SPDP Workshops, 1999

[11] IONA Technology http://www.iona.com/

[12] PrismTech http://www.prismtechnologies.com/

[13] ObjectiveInterfaceSystems http://www.ois.com/