Mapping Semantics of CORBA IDL and GIOP to Open Core Protocol for Portability and Interoperability of SDR Waveform Components

Grégory Gailliard, Hugues Balp and Michel Sarlotte
Thales Communications S.A.
Colombes, France
{firstname.lastname}@fr.thalesgroup.com

François Verdier
ETIS Lab – UMR CNRS 8051
Cergy-Pontoise, France
verdier@ensea.fr

Abstract

Patterns, middlewares and frameworks have been used for decades in software architecture to address the main problems encountered today by the MPSoC and NoC communities: heterogeneity of languages, programming models, simulation/execution environments, interaction semantics and communication protocols. A complete semantics mapping of CORBA Interface Definition Language (IDL) and General Inter-ORB Protocol (GIOP) on the Open Core Protocol (OCP) has been investigated for hardware components. This mapping is generic, highly configurable and illustrated through our target application: Software Defined Radio.

1. Introduction

Complex embedded systems like MPSoC are composed of heterogeneous elements ranging from CPU cores to hardware accelerators communicating through various media: buses or NoC. These computation and communication units have different specifications and implementation languages, simulation/execution environments, interaction semantics and communication protocols.

For decades, software middlewares address these issues thanks to standard *Interface Definition Languages (IDL)*, data encoding rules and message passing protocols. The *Common Object Request Broker Architecture* (**CORBA**) specifies an abstract language to define functional/business interfaces called **IDL** and an abstract communication protocol called **GIOP** (*General Inter-ORB Protocol)*. IDL enables *portability* of service definition, while GIOP permits *interoperability* of communications between distributed software objects and components. The key point in the object-oriented approach is that a function call or *method invocation* is equivalent to a message passing.

Our target application is Software Defined Radio (SDR) with QoS and hard real-time constraints. *Software Communication Architecture* (**SCA**) [1] and CORBA Component Model (CCM) are possible component models. SCA is a software architecture framework allowing separation between application waveform components and radio platforms. SCA is based on CORBA to provide interoperability and portability. However, CORBA implementations are not well adapted to the hardware domain, even if CORBA concepts are still relevant.

From an industrial point of view, technical choices must be based on standards to guarantee products life cycle and no vendor locking. In this paper, a configurable mapping from CORBA IDL to VHDL, Verilog or SystemC is thus proposed though a technology enabler: the Open Core Protocol (OCP). OCP is a non-proprietary hardware interface specification independent to any *Hardware Description Language* (**HDL**). One important point is to preserve interaction semantics and QoS requirements of applications in an end-to-end manner. The goal is that IDL becomes a HW/SW neutral definition, which describes explicitly the business interfaces of components.

This paper will be organized as follows: first, advantages of a transition from object to component model will be presented. Then an overview of CORBA will be given to describe IDL and GIOP in more details. The next section will be dedicated to OCP features. Afterwards, a mapping from CORBA semantics to the OCP ones is proposed. A realistic example inspired from an existing application will be used all along this paper to illustrate the proposed approach.

2. From object to component model

The component approach is natural in hardware. Instead of applying the object-oriented approach from software to hardware [5], we try to unify the component-oriented approach to hardware and software [7].

Consider a modem with a bit-symbol mapper. It maps an input stream of bits onto an output stream of I-Q symbols. This mapping may be based on constellations ranging from BPSK to QAM depending on the

978-3-9810801-3-1/DATE08 © 2008 EDAA

modulation mode. In an object-oriented approach, the data types used by the mapper could be defined in CORBA IDL 2.0 as:

```
// Data types definition
enum Mode{ BPSK_MODE, QPSK_MODE, QAM_MODE};
typedef  octet           Bit;
typedef  sequence<Bit>   BitSeq;
struct Symbol{
  short I;
  short Q; };
typedef sequence<Symbol> SymbolSeq;
```

Because the "bit" type doesn't exist, we choose a byte or *octet*. A **sequence** is a one-dimensional array with a bounded or unbounded size. An interleaver could push bits to the mapper, pushing itself symbols to a modulator. These services could be specified as follows:

```
// Business interfaces definition
interface BitStream {
  oneway void pushBit(in BitSeq bits); };
interface SymbolStream {
  void pushSymbol(in SymbolSeq sym); };
```

By default, the **oneway** keyword tells that the call or *invocation* semantics are best effort, the delivery of the call is not guaranteed. The **in, out** and **inout** keywords specify in which direction a parameter is sent. The modulation mode could be configured with:

```
interface Configure {
  void setTxMode(in Mode mode)};
```

Finally, the mapper object should implement the services described previously. It could provide a read-only identifier used by a deployment tool. Note that IDL doesn't provide a write-only keyword.

```
interface Mapper : Configure, BitStream {
  readonly attribute unsigned short id; };
```

The added value of the component model compared to the object model is that the interactions and dependencies between entities become explicit. A component is a reusable and composable building block. It *provides* and *requires* services through *ports*, which are bind to well-defined interfaces. In a component-oriented approach, the mapper should be defined in CORBA IDL 3.0 as:

```
component Mapper {
  readonly attribute    unsigned short id;
  provides Configure    config_in_port;
  provides BitStream    bit_in_port;
  uses     SymbolStream symbol_out_port;};
```

IDL 3.0 syntax is intuitive, only some additional hardware needed keywords would be necessary. In Figure 1, the equivalent UML diagrams are represented. CCM specifies how IDL 3.0 keywords are mapped to *equivalent interfaces* in IDL 2.0.

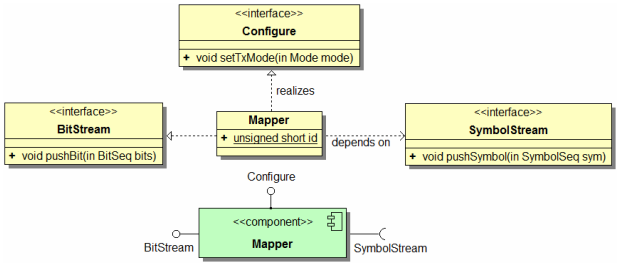

Figure 1: UML diagrams equivalent to IDL

A standard mapping exists from IDL to C++, but not from IDL to the SystemC language. Such a mapping is possible as shown in [1].

A brief overview of CORBA is given in the following to describe its underlying architecture.

3. CORBA overview

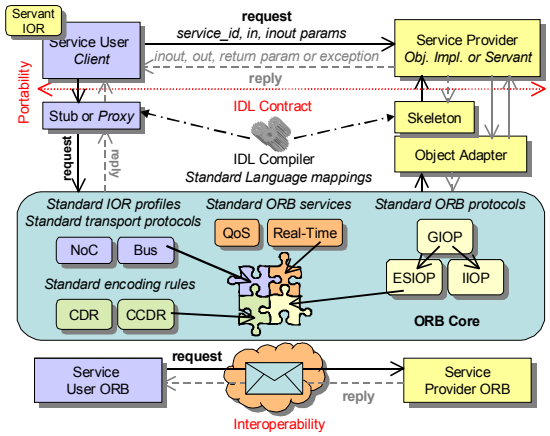

Figure 2: CORBA middleware framework

An *Object Request Broker* (**ORB**) is a logical bus allowing transparent accesses to services provided by distributed objects. A CORBA Object is identified by an *Interoperable Object Reference* (**IOR**) containing an Object ID. This IOR is similar to a visit card listing all the means to contact an object implementation or *servant*.

A **Stub** is a glue between a client and the ORB. It acts as a local surrogate or *remote proxy* to the servants and provides a Static Invocation Interface (SII). It encodes or *marshals* the input parameters of an operation call into a GIOP request message. It then de-serializes or unpacks the reply message to present operation results and exceptions to the client. A **Skeleton** acts as a glue between a servant and the ORB. It demarshals binary request packets into native data types and calls the operation on the servant. Then it packs results and exceptions in the reply returned to the client.

As it can be seen in Figure 2, an **IDL compiler** *automatically* generates stub and skeleton codes from the interfaces definition in IDL based on standard mappings

to software programming languages (C, C++, Java…). Section 6 will show that this mapping is also possible for VHDL, Verilog and SystemC thanks to OCP.

3.1. General Inter-ORB Protocol (GIOP)

GIOP is the native protocol of CORBA ORBs. GIOP is an abstract protocol, which has to be mapped onto concrete transport protocols. For instance, *Internet Inter-ORB Protocol* (**IIOP**) is a mapping of GIOP on TCP-IP. An *Environment-Specific Inter-ORB Protocol* (**ESIOP**) provides interoperability between CORBA ORBs and other middlewares through legacy/custom protocols [10].

GIOP contains three specifications: *Common Data Representation* (**CDR**), messages format and transport layer requirements. CDR specifies the bijective mapping from IDL data types to a byte stream. Primitive types are aligned on their natural boundaries implying padding. Other encoding rules can be proposed in ESIOPs like Compact CDR [10]. GIOP defines eight messages to send *Requests*, receive *Replys*, locate objects and manage logical connections. GIOP requires that transport layer is connection and byte stream oriented. Multiple independent requests for different objects may share the same connection. Clients may have multiple pending requests, which are ordered thanks to a *request_id*.

The *Open Management Group* (**OMG**) specifies a multitude of common services like Event, Log or Name service and specialized services like Security, Real-Time, QoS, Streams or Messaging. Lightweight CORBA profiles have been standardised like CORBA for embedded (CORBAe) or Lightweight CCM.

CORBA is not well suited to hardware because of its software history. However, CORBA provides an open framework into which optimized ESIOPs could be proposed and standardized by the OMG. After an overview of middleware architecture thanks to CORBA, the OCP layered approach will be presented.

4. On-chip communication architecture

4.1. Bus-centric vs core-centric approach

In on-chip communication architectures, two integration approaches of IP cores are traditionally considered. In the bus-centric approach, the IP core interfaces shall conform to a proprietary bus specification like AMBA or CoreConnect. This approach is not satisfactory for a standard mapping from IDL to HDL because of IPs portability. In the core-centric or *socket* approach, the IP core interface shall conform to a hardware interface specification like VCI, AXI or OCP. Socket based design allows to decouple IP cores from each others and from the on-chip interconnect. This fosters reuse and portability of IP cores. This approach is compatible with our goal focused on business logic.

We choose OCP as the socket interface for an IDL-to-HDL mapping because it is the only complete and configurable non-proprietary interface. Its semantics are generic enough to map to any bus or interface protocols.

4.2. Open Core Protocol (OCP)

OCP promotes a socket based layered SoC architecture. Interestingly, this communication architecture reminds distributed systems one like CORBA. A bus wrapper interface module acts *natively* as a stub. It consists of a slave entity and a bus initiator, which convert the OCP requests to the bus protocol. The first one acts as a proxy and the second one as an adapter. A kind of skeleton receives the requests and achieves the dual work. This socket approach allows communication transparency as it can be seen in Figure 3.

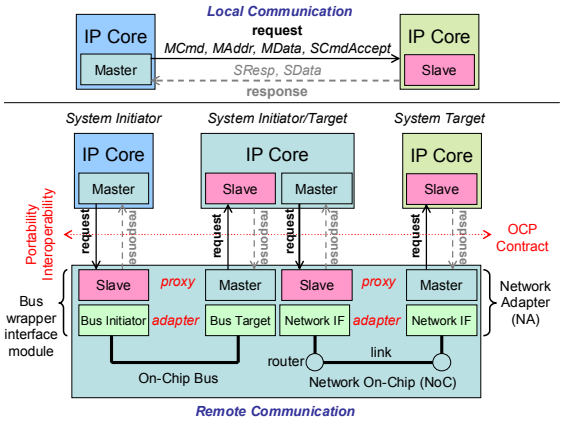

Figure 3: Communication transparency via OCP

The OCP standard has well defined protocol semantics. Its layered hierarchical protocol is made of transactions, transfers and phases. A transaction is composed of a set of read/write transfers. A transfer have three phases: request, response and optional data handshake. There are three groups of signals: dataflow, sideband and test signals. The dataflow signals are composed of basic signals and extensions: simple, burst, tag and thread. Only the main OCP features considered in the mapping are presented hereafter.

The basic signals provide signals for address and data bus, request and data acknowledge. MCmd signals encodes eight commands: basic write, basic read, exclusive read, linked read, non posted write, conditional write and broadcast. Write commands can return a response with **posted** or **non-posted semantics**. In non-posted writes, a response is sent after write completion at final target. Blocking or **locked synchronization** required by semaphores is based on exclusive read and posted or non-posted write. Non-blocking or **lazy synchronization** is also supported by linked read and conditional write.

The simple extension adds support to partial words transfers thanks to byte enables and in-band information in one of the three OCP phases.

The tag extension enables concurrent transfers within a single shared flow of control. Tags act as request identifiers and allow out-of-order responses. The thread extension enables concurrent transfers within independent flows of control. Whereas transfers within a single thread shall be ordered unless tags are used, transfers within different threads have no ordering constraints.

Connections establish end-to-end logical links between a system initiator and a system target in a global scope. A contrario, threads have a local scope in the point-to-point logical link between a master and a slave entity. Connections are identified by the connection ID signals. The OCP specification proposes a networking analogy: thread ID refers to the data link layer, whereas connection ID refers to the transport/session layer.

OCP defines native profiles to describe OCP interfaces for block data flow or register access, and bridging profiles for AHB like bus and AXI like read/write channels. An OCP profile activates and configures protocol features like phases acknowledge, signal data width or numbers of tags/threads.

OCP provides also a Transaction Level Modeling (TLM) in SystemC and defines four transaction levels. The layer 3 is bus protocol agnostic thanks to generic transactions. The layer 2 models OCP transactions with OCP profiles. In layer 1, OCP transfers are cycle true but faster than RTL. The Layer-0 corresponds to Register Transfer Level (RTL).

Our proposition is based on previous works, which are briefly presented in the following.

5. Related work

The Integrated Circuit ORB (ICO) engine [4] implements a hardware CORBA ORB. Read/write primitives on components registers and memories are described in CORBA IDL and are used to build GIOP messages transferred via FIFO-like interfaces. MultiFlex [6] comprises a system IDL compiler supporting a neutral data representation, hardware message passing engines acting as stubs/skeletons and a hardware ORB engine to manage inter-object communication. The Object-Oriented Communication Engine (OOCE) [5] is based on the IDL and encoding rules of another CORBA like middleware called Internet Communication Engine (ICE). OOCE also integrates hardware stubs/skeletons. The Modem Hardware Abstraction Layer (MHAL) API [2] specifies message formats and a communication/routing API for SCA based SDRs. This could constitute part of an ESIOP.

A container-component approach is proposed in [7]. Components have a control interface and "business" interfaces bind to a subset of IDL. Local services are provided by containers. Hardware implementations of components produce and consume messages through unidirectional ports. For each component port, there is one port for requests and one port for replies. An OCP profile is provided for each port category. The mapping from IDL to OCP is not systematic depending on interface type. Control operations are mapped as reads on OCP address signals in thread 0, configuration operations on regular reads/writes in thread 1 and business operations on OCP Control or Status signals.

An object-to-RTL mapping is proposed in [3]. For each operation, there is one input signal for request, one output signal for reply and several input/output data ports. It is simple, systematic and network/bus protocol independent. However it uses a basic and custom interface without control flow instead of a scalable and standard interface like OCP.

Our contribution consists in leveraging and improving these approaches with a generic mapping model supported by CORBA and OCP concepts.

6. Mapping CORBA to OCP semantics

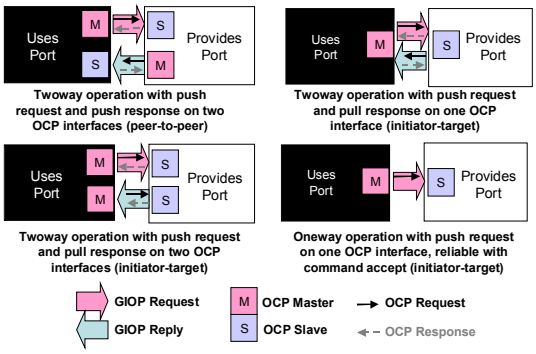

Figure 4: Different interaction models

To guarantee coherent HW/SW programming models, CORBA invocation semantics can be mapped on OCP commands. In oneway invocations, an initiator sends writes without response because no reply is expected. Reliable oneway invocations can be distinguished from best effort ones with the SCmdAccept signal. For synchronous twoway invocations, an initiator sends writes with response to wait on target reply. In a push model (Figure 4), a target master entity sends OCP commands to return a reply to an initiator slave entity. For a pull model, an initiator master entity sends OCP commands to obtain a reply from a target slave entity. Interrupts may also be used. CORBA exceptions can be raised with OCP error signaling mechanisms.

Operation invocations are interpreted as argument transfers. Typically, `in` corresponds to writes, `out` to reads in a pull model or writes in a push model, and `inout` as a combination of both. The namespace formed by IDL modules, interfaces and operations maps to a distributed and virtual address space. When configured, OCP address signals encode explicitly logical or physical

addresses of arguments. Otherwise, these addresses are implicitly determined by the order of transfers.

Concurrency and synchronization [5][6] are mapped on OCP locked and lazy synchronizations. For instance, the *Mutex* interface of Real-Time CORBA can be *natively* associated with exclusive reads and non posted writes. Broadcast could be used for the publish-subscribe model.

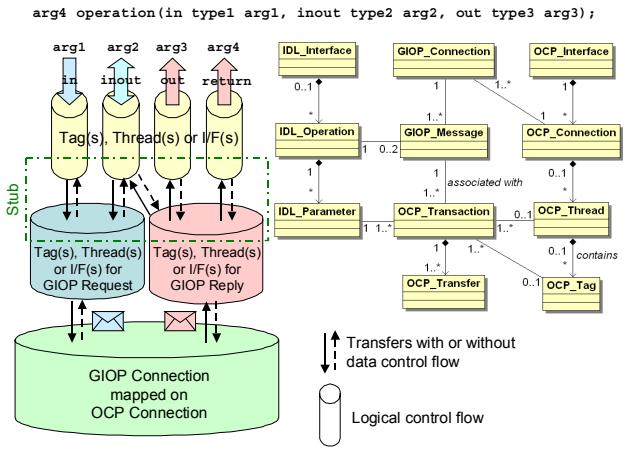

Figure 5: Generic model of the mapping

OCP flexibility provides various solutions for the transfers of parameters. Figure 5 illustrates the generic model formalizing the whole set of mapping solutions offered by the OCP transfers abstraction model. Similar to CORBA threading policies, the type and the number of control flows must be considered. A tag, a thread or an OCP interface can be assign by parameter(s), operation(s) or interface(s). For instance, an IDL interface can be associated to an OCP interface, an operation to a thread and each parameter to a burst identified by tags. To reduce the mapping solution space, one OCP port per interface can be chosen in accordance with CORBA and CCM. This implies that operations declared in the same IDL interface will share the same OCP profile. Like OCP packing rules, `in` parameters are aggregated or padded with byte enables, whereas `out` parameters are splitted or stripped depending on encoding rules to build messages. For an invocation, parameters transfers are ordered, but may be interleaved in the stub with other invocations. Control flow for each parameter may be handled thanks to SRespInfo or SThreadBusy signal and may be propagated on connection to guarantee end-to-end QoS.

Tag(s), thread(s) or OCP interface(s) can be used to transfer messages. GIOP request and reply phases correspond to two or more OCP transactions. GIOP connections are mapped on OCP connections between stub/skeleton pairs. Address alignment and endianness can be explicitly configured in OCP profiles.

An IDL-to-HDL mapping must be highly flexible to satisfy various performance/cost tradeoffs. OCP answers to this need. It formalizes protocols required by IP cores interfaces thanks to profiles acting as a "hardware IDL". Possible overheads only depend on profile definition, configuration and implementation as explained in [8].

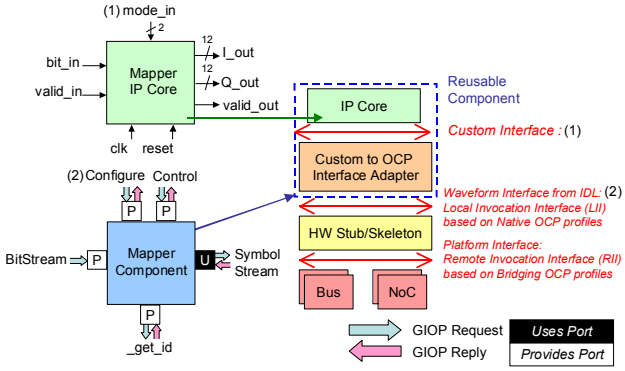

Figure 6: Example of IDL to OCP mapping

As it can be seen in Figure 6, our mapping proposition is based on two categories of OCP interfaces for stubs and skeletons: *Local Invocation Interface* (**LII**) provided by the business (waveform) component and *Remote Invocation Interface* (**RII**) provided by the platform. This allows a separation of concerns between transfers of operations arguments and transfers of messages. Both interfaces are configured by OCP profiles.

The LII corresponds to a "waveform socket", which maps IDL interfaces on OCP interfaces. Figure 7 shows the OCP interfaces for the mapper component of section 2. The RII corresponds to a "platform socket", which is the interface to the underlying communication infrastructure: bus or NoC. RII for bus is based on standard OCP bridging profiles. RII for NoC may be write/read FIFOs with control flow to send/receive lightweight GIOP or ESIOP messages. These messages can be encapsulated in OCP data signals or each field can be mapped on dedicated OCP signals.

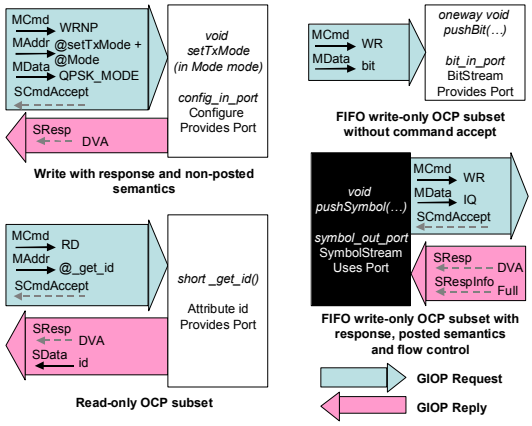

Figure 7: Details on Mapper component ports

This approach is generic and not dedicated to SDR. In the SDR case, the IDL interfaces may be relatively simple without requiring complex data types, and restricted to some OCP subsets as illustrated in Figure 7 and in [8].

OCP Transaction Level Models allow to evaluate mapping tradeoffs and to successively refine OCP interfaces using the same OCP profiles. The common HW/SW specification, simulation and design flow is based on *Model Driven Architecture* (**MDA**) [1]. The *Platform Independent Model* (**PIM**) is specified in UML and IDL; the *Platform Specific Model* (**PSM**) in HW/SW languages: C, SystemC and HDLs. Some experimental results of our approach are presented in the next section.

7. Experimental results

The IDL to HDL mapping can be seen as a refinement problem in SystemC TLM. We validated some aspects of the proposition in OCP TLM on a representative example inspired from an OFDM modem. The sample waveform is composed of a controller which broadcasts configuration invocations on several components: a MAC component, a Convolutional Coder, a BitInterleaver and a bit-to-symbol Mapper, as it can be seen in Figure 8.

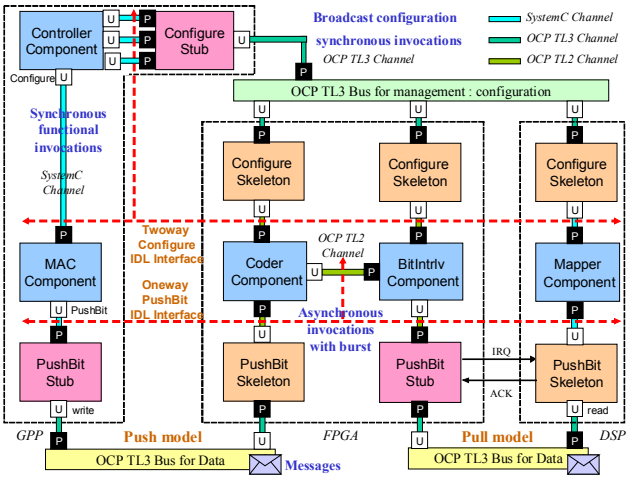

Figure 8: Mapping illustration using OCP TLM

After invocation of `setTxMode` operation, a stream of bits is sent and received from the MAC to the Mapper components by calling `pushBit` on their ports. Synchronous and asynchronous method invocations have been modeled in SystemC. For components targeted in a FPGA, these are non-posted and posted writes in OCP TL2. For local invocations between software components, this is achieved thanks to Interface Method Call (IMC) on SystemC channels [1]. The `BitSequence` *`in`* parameter is sent as one burst. A push and pull model has been implemented. In the push model, the MAC stub writes on the Coder skeleton. For the pull model, the BitInterleaver stub generates an interruption and the Mapper skeleton reads the bit stream. Messages can be ever encapsulated or mapped onto OCP TLM channels.

The Coder component has been refined in VHDL RTL and simulated. The configuration OCP port implements a write-only OCP subset to configure the encoding rate. Two input and output stream data flow ports implement the FIFO write-only OCP subset without SCmdAccept. Each OCP port is a VHDL module, whose interface signals are configured by generics defined in a VHDL package that contains all the OCP profile parameters. The Coder IP core has been reused from the existing OFDM modem. The three OCP ports make the adaptation between IDL derived OCP interfaces and the custom HDL interfaces. This hardware reuse approach is strictly similar to software CORBA wrappers developed to reuse legacy code in automatically generated stubs/skeletons.

8. Conclusion & future works

In this paper, a mapping from CORBA middleware semantics to Open Core Protocol semantics has been proposed. Thanks to OCP, our mapping is flexible and configurable to address from simple to complex interfaces. This mapping allows to raise the abstraction level of interactions between HW/SW components. The use of two industrial standards, CORBA and OCP, will foster automation as shown in [9]. We plan to apply and refine this mapping by implementing our example with the SCA Core Framework and the Lightweight CCM framework in the scope of the ITEA SPICES project.

9. References

[1] G. Gailliard et al., "Transaction Level Modelling of SCA Compliant Software Defined Radio Waveforms and Platforms PIM/PSM", DATE'07, April 2007.

[2] JPEO JTRS, Modem Hardware Abstraction Layer (MHAL) API, Version 2.11.1, May 2007.

[3] J. Barba et al., "Light-Weight Communication Infra-structure for IP Integration", IPSOC'06, December 2006.

[4] F. Humcke, "Making FPGAs "First Class" SCA Citizens", SDR Forum Technical Conference, November 2006.

[5] J. Barba et al., "OOCE: Object-Oriented Communication Engine for SoC Design", 10th EUROMICRO Conference on Digital System Design (DSD '07), August 2007.

[6] P. G. Paulin et al., *Parallel Programming Models for a Multi-Processor SoC Platform applied to Networking and Multimedia*, IEEE Transactions on VLSI Journal, Vol. 14, pp. 667-680, July 2006.

[7] JPEO JTRS, "Extension for component portability for Specialized Hardware Processors (SHP) to the JTRS SCA Specification", v3.1, March 2005.

[8] J. Noseworthy and J. Kulp "Standard Interfaces for FPGA Components", Milcom'07, October 2007.

[9] J. Noseworthy and J. Kulp, "Code Generation for SCA Components Running on FPGAs", Milcom'06, October 2006.

[10] S. Lankes et al., *Design and performance of a CAN-based connection-oriented protocol for Real-Time CORBA*, Journal of Systems and Software, Vol. 77, pp.37-45, July 2005.