MIDDLEWARE FOR DSPS AND FPGAS

William Beckwith (Objective Interface Systems, Herndon, VA, USA; bill.beckwith@mail.ois.com)
Victor Giddings (Objective Interface Systems, Herndon, VA, USA; victor.giddings@mail.ois.com)

ABSTRACT

System designers have historically wrestled with the determination of what portion of their logic is appropriate for each type of specialized processor. The typical approach is an educated guess or, worse yet, an unconscious repeat of the design pattern from a previous generation. Multiple tensions drive and constrain the assignment of application logic to each specialized processor or GPP. To minimize deployment cost while achieving the required current and future functionality, system designers must balance the following factors:

- Initial ramp up cost,
- Period to amortize the initial research and development,
- Cost per deployed unit,
- Power consumption,
- Legacy intellectual property,
- Existing engineering skills and paradigms and
- Performance.

The appropriate communications middleware allows system designers to construct flexible, maintainable systems that can accommodate the widest possible range of waveform computing loads while maintaining economic goals for a target platform cost profile.

This paper will compare and contrast multiple approaches to using communications middleware on specialized devices.

1. INTRODUCTION

According to WikiPedia, *middleware* “consists of software agents acting as an intermediary between different application components. It is used most often to support complex, distributed applications. The software agents involved may be one or many.”[1] The rise of middleware for general purpose computing is well known, and has been extended to embedded and real-time systems. In the Software Defined Radio domain, appropriate middleware allows system designers to construct more flexible and maintainable systems that can accommodate the widest possible range of signal processing computing loads while maintaining economic goals for a target deployment cost profile. This paper compares and contrasts multiple approaches to extending the applicability of middleware concepts to specialized devices, including Digital Signal Processors (DSPs), Field Programmable Gate Arrays (FPGAs), and Application Specific Integrated Circuits (ASICs).

2. MIDDLEWARE GOALS

Our goals for the DSP, FPGA, and ASIC middleware are as follows. First and foremost is the necessity to achieve the required functionality for the system, both in terms of current requirements and future requirements. Second, the system must achieve the performance profile required for the target user. A frequent goal of middleware is to achieve portability by isolating developed application logic from the surrounding platform environments. Interoperability, the ability for two components of logic to communicate over a common media, is frequently useful to accommodate independent development of the components. Incremental migration is the capability to migrate developed logic a piece at a time. Processing mobility means that processing logic can be moved from one type of specialized device to another. Of course, with any system it is important to minimize the cost of developing the system and minimize the time it takes to complete a correct system. Another key cost factor is the cost of the deployed hardware. It is important to minimize this deployed cost to maximize return on investment.

In achieving these middleware goals, there are important considerations to balance. System designers must balance the following factors:

- Performance
- Power consumption
- Cost per deployed unit
- Legacy intellectual property/hardware architectures
- Existing engineering skills

 Copyright © 2005 SDR Forum. All Rights Reserved

- Existing engineering paradigms
- Initial ramp up cost
- Period to amortize the initial research and development

3. PROCESSING DEVICE TYPES

The scope of this paper addresses analyzing for types of processing devices: GPP, DSP, FPGA, and ASIC.

Table 1 is intended to be representative of the types of tradeoffs that system designers are faced with. The table helps exemplify that each type of device has its place in the system architects bag of tools, given the wide variance in speed, power, cost, and human factors issues.

Table 1: Processing Device Tradeoffs

Processing Device	Speed	Power	Initial Cost	Cost per Unit	Design Investment Longevity	Skill Avail-ability
GPP	**1**	**1**	**1**	**1**	+++	+++
DSP	**1 – 1.8**	**0.5**	**2**	**0.8 – 3**	+	+
FPGA	**3 – 20**	**0.25 – 2**	**4**	**2 – 4**	+	-
ASIC	**3 – 50**	**0.1 – 0.3**	**20**	**0.1 – 0.2**	--	---

Of course in each of these device types has a very wide range of processing capability. Each row in the table should be considered in its entirety when comparing two other rows in the table. For example the cost per unit of a small FPGA would typically be less than a large GPP.

It is the basic nature of computing resources that two fundamental rules can be observed. First, processors absorb algorithmic complexity, meaning that the more complex the algorithms the more processing capability is required. Second algorithmic complexity will typically expand to outstrip the available processing capability.

Over the course of time the natural solution to a given computing problem will have a natural migration. Initially an ASIC is the only option to solve some algorithms given a severe constraint on computing power per watt or per pound.
As the speed of FPGA processors improves over time a FPGA may provide sufficient computing capability for an algorithm given the original power or weight constraint. Subsequently a DSP processor will provide sufficient computing capability for the original constraints. Eventually the algorithm can be processed on general purpose processors (GPP) within the constraints.

An example of three algorithms currently in different stages of this natural migration is drawn from the audio/video domain. A state of the art GPP will have sufficient processing capability to implement an audio music codec. However, even a low-end video codec will require a DSP currently. And a high-end video codec such as HDTV will require an FPGA or ASIC to properly implement the algorithm in a reasonable form factor for a small embedded system.

The ramp up costs for these different devices differs quite a bit. For the GPP, DSP, and FPGA that ramp up cost is simply the cost of developing the logic. However, for an ASIC, the ramp up cost is the cost of developing the logic plus approximately $10 million.

The financial benefit of a given research and development effort to produce a new capability is not limited to the revenue produced by the products that it is used for. The financial benefits also include residual reusable intellectual property. One key benefit to a GPP is that the software can survive many generations of hardware. The result is a long period of over which the initial research and development expenses can be amortized. The initial research and development expenses for developing the software for a DSP are typically shorter than that for a GPP. There are two causes for this reduced amortization period. First, the architecture and capabilities of DSPs tend to change in a way that obsoletes previous implementations of an algorithm. Second, there is considerable innovation in DSP signal processing algorithms which render previous algorithms obsolete.

FPGA processors have an even shorter payback period for the initial research and development expenses. That payback is typically bound to the FPGA family. ASIC processors are typically redesigned each new generation. This is typically driven by the high ramp up costs of developing each new ASIC revision.

The relevant metric for measuring the size required to absorb a given algorithms complexity differs for each device type: For a GPP and a DSP, size is the object code and memory used during the execution of that object code. For a FPGA and an ASIC, size is the number of transistors needed to execute the algorithm in hardware. Frequently FPGA processors are discussed in "logical units".

Execution time criticality has a different scale on each specialized device type. A performance critical embedded systems developer will consider each microsecond as

 Copyright © 2005 SDR Forum. All Rights Reserved

critical on a GPP processor and a DSP processor. Correspondingly, on an FPGA, tens of nanoseconds are critical. On an ASIC, ones of nanoseconds are critical.

The difficulty of developing the logic to realize a particular algorithm differs between the device types. GPP processors have a wide array of available tools and are the easiest to develop on. DSP processors are more constrained specialized environments with fewer tools. There are a variety of tools that a hardware engineer may use to implement a particular algorithm on a FPGA or ASIC processor. While various software conversion tools exist, most logic is developed using hardware description languages such as Verilog and VHDL.

Development resource also differs according to processor type. Software engineers familiar with general purpose processors are readily available. Those familiar with DSP familiarity are less common, but it is possible to convert a good software engineer in about six months. FPGA developers are less common still, and tend to be hardware engineers by training. Developers of ASICs are rare these days and are usually specially-experienced hardware engineers.

The programming languages available to developers, a key determinant of development productivity, differ between the processing classes. GPP developers have a number of languages to choose from, including: C, C++, Java, and Ada. DSP code is generally C, with C++ becoming more available. FPGA & ASIC descriptions are generally in Verilog or VHDL.

It is important to understand the security implications of specialized devices. GPP processors typically contain memory management units that allow for hardware enforced boundaries between applications. DSP processors typically have no memory management units and thus provide little boundaries to contain a erroneous or subverted applications. FPGA and ASIC processors provide no architecturally imposed separation other than the natural time and space separation of hardware blocks that are not directly connected.

4. MIDDLEWARE FOR DSP, FPGA, AND ASIC DESIGNERS

We can now elaborate the expected benefits of middleware for these processing device types. In addition to middleware used within a particular processing device type, there are additional benefits in middleware that bridges between different processing device types.

Middleware technology deals with connecting pieces of a system, whether software components or hardware blocks. Useful DSP, FPGA, and ASIC middleware technology would provide a component definition and container technology for each of these processing device types, as well as an inter-container communications solution. The middleware would allow developers to rewrite the contents of a container for a different processing capability without changing the contents of the rest of the existing container, i.e., the rest of the system. This *processing mobility* of moving functions independently greatly increases incremental system evolution capacity. For example, a function could be moved from an expensive ASIC to an FPGA as the result of increased FPGA processing power, without changing the interfaces between components of the system. Also, a function in a GPP can be moved in an FPGA to support higher capacity or lower power models of a device. Of course, there are always limits on the isolation that component technologies can attain; system-level properties such as a timing and throughput will have to be re-analyzed after any change.

There have recently been proposals to extend middleware concepts and technologies into these processing device types. In the remainder of this paper, we will discuss the merits of five candidates:

- Custom-built
- Transport level (HAL-C)
- Real-time CORBA
- High-Assurance CORBA
- SCA-289 Component/Container Model

Table 2 shows the range of applicability of each of these candidate middleware technologies to the processing device types described above. It also contains a column for a hybrid type of processing device. Most of the FPGA vendors have products that are capable of hosting one or more full general purpose microprocessors in a fraction of the gates available. This allows immediate availability of some of the technologies currently available only on GPPs on these FPGAs.

The following subsections discuss the tradeoffs in these middleware technologies.

Proceeding of the SDR 05 Technical Conference and Product Exposition. Copyright © 2005 SDR Forum. All Rights Reserved

Table 2: Applicability of Candidate Middleware Technologies

Approach	GPP	DSP	GPP core on FPGA	FPGA	ASIC
Custom/ Proprietary	✓	✓	✓	✓	✓
Transport (HAL-C)	✓	✓	✓	✗	✗
Real-time CORBA	✓	✓	✓	✗	✗
High Assurance CORBA	✓	✓	✓	✗	✗
SCA-289 Comp/Cont Model	✓	✓	✓	✓	✓

4.1 Custom Built Approach

"Roll you own" middleware is still popular in some domains and, until recently, was the only viable option for the processing device types other than the GPP. This approach is applicable to all the processing device types. It has some decided technical advantages. Since it can be developed for the specific processing devices to be deployed, it is potentially very fast and small in footprint, and can be tightly focused on the requirement current for the project.

It suffers a number of disadvantages because of its tight focus, however. A specific project may not have the time or money to develop optimal performance and time. The project management must always trade off time to market against these other market distinguishers. The solution is likely to be highly specific to the hardware chosen for the project; it may not be reusable on the vendor's next model in the same product line. The proprietary solution will also lock you into the supplying vendor. Because the custom solution is not interoperable, the ability to incrementally migrate parts of the solution to new technologies is precluded. The custom-developed infrastructure will be more expensive than the cost of middleware, and make it more difficult to control the lifecycle costs of the product. The questionable reusability of the custom infrastructure yields problems when attempts to extend it are made, and result in a short period over which the development investment is paid back.

4.2 Transport Level Approach

Version 3.0 of the Software Communication Architecture (SCA) developed under the Joint Tactical Radio System program includes a Hardware Abstraction Layer – Connectivity (HAL-C). HAL-C "specifies a hardware platform-independent means for communication between software components running on specialized hardware"[2]. HAL-C is exemplary of the transport-level approach; it specifies how the bits of a message are transported between components in the different processing device types.

Such approaches are applicable to GPPs and to DSPs, but break down when FPGAs are included. (The specification of HAL-C for FPGAs is generally regarded as being too underspecified to be useful.)

Transport-level approaches to middleware offer a standard API for inter-device communication, and provides some portability for both transport implementations and applications that use the transport.

They do not provide message formatting, and thus no interoperability between components. They also do not incorporate zero-copy interfaces, which impacts their use for intra-device communication and adversely affects performance. The limitations of this approach preclude processing mobility. Finally, there is no specification of inter-component timing.

4.3 Real-Time CORBA

There have been proposals to extend specialized CORBA implementations into DSPs. The Real-Time CORBA specification [3] added features to the CORBA standard to support predictable remote invocations across a distributed system. The specification is supported by a number of implementations that also support the Minimum CORBA specification, a specification that subsets the features of CORBA to reduce complexity and footprint. It is possible to employ implementations that take less than 100K of object code for both client and server support.

These implementations could be used on GPPs, on GPP cores within FPGAs, and in DSPs. Based on a well defined foundation of OMG specification, they would provide device location transparent logic, i.e., the logic for a component could be located and re-located across any of the supported processing device types without affecting the other components of the system. The OMG standards basis of these implementations would ease integration of the specialized devices with GPP-based components of the application. The components developed for this technology will contain code that is partially portable to and from a GPP; the interfaces to the components' functionality will be portable, but the implementation logic will depend on DSP-

specific processing libraries. The communication between components, since it is standards-based will be interoperable.

The drawbacks of this approach come from trying to impose a GPP-developed paradigm onto a different processing device type. The object-oriented, multi-threaded CORBA paradigm is unfamiliar to most DSP engineers. DSP engineers are using DSPs in order to wring the best performance out of the hardware. Thus, DSP-traditional footprint and performance concerns can become a cultural barrier, even with a small, fast DSP ORB. Most proponents of this approach focus on the footprint of the ORB core and ignore the footprint of the code generated from the IDL. The code can be quite large; an experiment with the SCA Core Framework yielded between 30K and 250K lines of generated code.

4.4 High-Assurance CORBA

The High-Assurance CORBA effort that is ongoing in the OMG is being conducted by Objective Interface Systems and Rockwell Collins. It is an effort to define a subset of Minimum CORBA for use in systems that are subject to the requirements of safety certification, such as those that are certified to DO-178B [4]. One of the outcomes of this specification effort will be more robust, easier to use language mappings. There will also be enhancements to apply formal methods to IDL, e.g., in the form of pre-conditions and post-conditions. Tthe users of this technology will be better able to build correct systems on first attempt. This effort should result in a specification that would allow implementation with further reduced footprint.

Thus, this approach has the same applicability as the previous approach: GPPs, GPP cores on FPGAs, and DSPs. It also has the same advantages but sometimes to greater degree: ORB core written to this subset will be even smaller in footprint. The implementations, because of the safety certification considerations, will have better testability and robustness. This correctness in infrastructure should yield quicker deployment.

However, in addition to the disadvantages of the previous approach, this approach is not yet available. The development of the specification is still in progress, and products supporting the specification will follow after the specification. This approach still requires the DSP engineer to adopt the object-oriented coding paradigm of CORBA. There is no specification of inter-component timing that allows analysis of performance and timing correctness.

4.5 SCA-289 Component/Container Model

Change proposal #289 [3], a candidate for SCA 3.1, includes a proposal for a component and container model for three classes of processing systems:

- GPP Class: normal first class component environments or CORBA enabled environments. In this processing class existing component specifications are suitable.
- RCC Class: Resource-Constrained C language environments. When C is available, but CORBA is not suitable. For example, DSPs or Microcontrollers or RISC cores with limited memory
- RPL Class: RTL-Programmable-Logic environments When an RTL language is available (VHDL/Verilog), but C is not available or not suitable. This includes FPGAs and ASICs.

This proposal is comprehensive and includes all of the previously enumerated processing device types. It provides both inter-container and inter-device interface definition, portability of components at source level, replace-ability across technologies. It addresses the separation of concerns between platform providers and component authors, allows resource efficiency and performance, and has minimal impact on existing component models.

The primary shortcoming of this proposal is that it does not address a specification of inter-component timing.

CONCLUSION

A summary of this analysis of tradeoffs along with some quantization of the expected overheads is offered in Table 3. Each of the approaches above are represented by a row in the table. The "Portable Logic" column shows that none of these approaches offers portability across the different processing device types: FPGA "code" will be different than DSP code will be different than GPP code. The second column, "Portable Logic Shell" indicates whether the definition of the interfaces is portable across the different processing device types. OMG's IDL is used as the specification of inter-component interfaces in the last three approaches. The column labeled "Portable Inter-logic Comm" reveals whether communication paths between different components of the system are portable, i.e., whether the migration of a component from one processing device type to another (e.g., GPP to DSP) would be accommodated by the infrastructure. The last three columns indicate the expected overhead of the middleware infrastructure in terms of FPGA slice or Logic Elements, memory footprint, and clock cycles, respectively.

Table 3: Evaluation of Candidate Middleware Technologies

Approach	Portabl e Logic	Portable Logic Shell	Portable Inter-logic Comm	Middleware FPGA Footprint (Slices/LEs)	Middleware Memory Footprint (kilobytes)	Execution Overhead (kilocycles)
Custom/ Proprietary	✗	✗	✗	varies	Varies	varies
Transport (HAL-C)	✗	✗	✓	✗	0.2 – 5 (+ xport)	0.1 – 2
Real-time CORBA	✗	✓	✓	✗	45 – 100 Clnt + Svr (+ xport)	1.8 – 20
High Assurance CORBA	✗	✓	✓	✗	15 – 35 (+ xport)	0.9 – 4
SCA-289 Comp/Cont Approach	✗	✓	✓	80 – 120	20 – 35 (+ xport) 0.3/cpnt	0.1 – 3

REFERENCES

[1] "Middleware",Wikipedia, http://en.wikipedia.org/wiki/Middleware

[2] "Specialized Hardware Supplement to the Software Communication Architecture (SCA) Specification", JTRS-5000 SP, V3.0, 27 August 2004, http://jtrs.army.mil/documents/sca_documents/V3.0/SCA%20Specialized%20Hardware%20Supplement%203.0.pdf

[3] "Real-Time CORBA Specification", Version 1.2, OMG Document formal/2005-01-04, http://www.omg.org/technology/documents/formal/real-time_CORBA.htm

[4] "Software Considerations in Airborne Systems and Equipment Certification", DO-178B, RTCA Inc., 1140 Connecticut Avenue, N.W., Suite 1020, Washington, DC 20036

[5] Kulp, J., et al, "Portable Waveform Components for Specialized Hardware". http://jtrs.army.mil/documents/sca_documents/V3.1_workshop/CP_289%20Component_PortabilityJTRSWKSHOP3.ppt

Proceeding of the SDR 05 Technical Conference and Product Exposition. Copyright © 2005 SDR Forum. All Rights Reserved

Middleware for DSPs and FPGAs

Bill Beckwith
Objective Interface Systems, Inc.
http://www.ois.com info@ois.com

SDR Forum Technical Meeting 2005

© Objective Interface Systems, Inc. 2005

Overview

- **The appropriate *middleware* allows system designers**
  - to construct flexible, maintainable systems
  - that can accommodate the widest possible range of signal processing computing loads
  - while maintaining economic goals for a target deployment cost profile

- **This presentation will compare and contrast multiple approaches to using middleware on specialized devices (DSPs, FPGAs and ASICs)**

© Objective Interface Systems, Inc. 2005

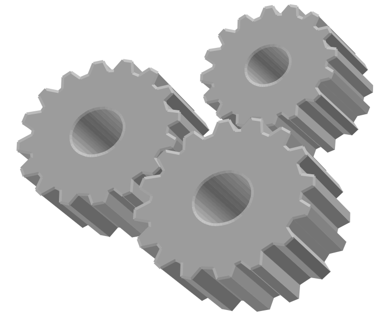
Goals for DSP, FPGA, and ASIC Middleware

- **Achieve the required functionality**
  - Current requirements
  - Future requirements
- **Achieve the required performance**
- **Portability**
- **Interoperability**
- **Incremental migration**
- **Processing mobility**
- **Minimize development cost and time-to-completion**
- **Maximize return on investment**
- **Minimize deployment cost**

© Objective Interface Systems, Inc. 2005

Considerations While Achieving Middleware Goals

- **System designers must balance the following factors**
  - Performance
  - Power consumption
  - Cost per deployed unit
  - Legacy intellectual property/hardware architectures
  - Existing engineering skills
  - Existing engineering paradigms
  - Initial ramp up cost
  - Period to amortize the initial research and development



© Objective Interface Systems, Inc. 2005

Comparing the Technologies

Processing Device	Speed	Power	Initial Cost	Cost per Unit	Design Investment Longevity	Skill Availability
GPP	1	1	1	1	+++	+++
DSP	1 – 1.8	0.5	2	0.8 – 3	+	+
FPGA	3 – 20	0.25 – 2	4	2 – 4	+	-
ASIC	3 – 50	0.1 – 0.3	20	0.1 – 0.2	--	---

© Objective Interface Systems, Inc. 2005

Nature of Computing Resources

- **Two rules**
  - Processors absorb algorithmic complexity
  - Algorithmic complexity expands to outstrip processing capability

- **Solutions to computing problems have natural migration**
  - ASIC	the only option for maximum compute power per watt or lb
  - FPGA	when solution allows more watts or lbs
  - DSP	when solution can exist on specialized signal processing hw
  - GPP	when solution can exist on general purpose hw

- **Examples of solutions currently constrained by computing resource**
  - GPP Ok	audio music codec
  - DSP Ok	low end video codec
  - FPGA/ASIC Ok	high-end video codec (HDTV)

© Objective Interface Systems, Inc. 2005

Comparisons

- **Ramp up cost**
  - GPP: Logic development + $0
  - GPP: Logic development + $0
  - FPGA: Logic development + $0
  - ASIC: Logic development + $10,000,000
- **Period to amortize the initial research and development (longevity of IPR assets)**
  - GPP: long; can survive many generations of hardware
  - DSP: medium (bound to DSP family)
  - FPGA: short-to-medium (bound to FPGA family)
  - ASIC: short, typically bound to the lifetime of that chip run
- **Size**
  - GPP & DSP: object code and memory use size
  - FGPA: number of logical units

© Objective Interface Systems, Inc. 2005

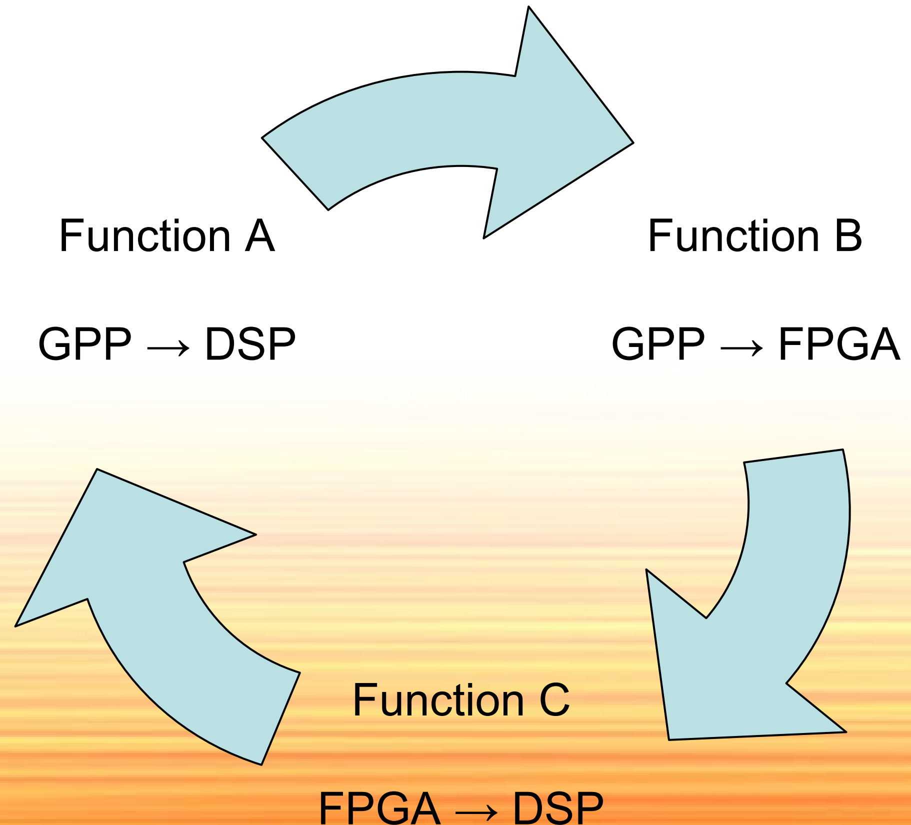
Comparisons (cont'd)

- **Performance**
  - GPP & DSP: 1's of microseconds critical
  - FPGA: 10's of nanoseconds critical
  - ASIC: 1's of nanoseconds critical
- **Logic development**
  - GPP: easy to develop, lots of tools
  - DSP: more constrained environment, fewer tools
  - FPGA & ASIC: some swr conversion tools, mostly VHDL & Verilog
- **Engineering resources**
  - GPP: software engineers widely available
  - DSP: less common, but can convert a good swr engineer in six months
  - FPGA: less common, hardware engineers by education
  - ASIC: rare these days, special experience hardware engineers
- **Languages**
  - GPP: many, C, C++, Java, Ada, …
  - DSP: C, C++
  - FPGA & ASIC: Verilog & VHDL
- **Partitioning Support**
  - GPP: MMUs provide partitioning and separation
  - DSP: typically no partitioning or separation (there are exceptions)
  - FPGA & ASIC
    - no architecturally imposed separation
    - hw blocks can separate by design if carefully constructed

© Objective Interface Systems, Inc. 2005

- **Inter-container communication solution allows:**
  - Rewrite a container for a different processing capability
  - Without changing the rest of the existing containers

Processing Mobility: Moving Functions Independently



➢ **Incremental system evolution – *Processing Mobility***
  - Each container can migrate/integrate independent of other containers
  - Still need system-level timing analysis

© Objective Interface Systems, Inc. 2005

Catalog of Approaches

- Custom
- Transport level (HAL-C)
- Real-time CORBA
- High-Assurance CORBA
- SCA-289 Component/Container Model

© Objective Interface Systems, Inc. 2005

Applicability of Approaches

Approach	GPP	DSP	GPP core on FPGA	FPGA	ASIC
Custom/ Proprietary	✓	✓	✓	✓	✓
Transport (HAL-C)	✓	✓	✓	✗	✗
Real-time CORBA	✓	✓	✓	✗	✗
High Assurance CORBA	✓	✓	✓	✗	✗
SCA-289 Comp/Cont Model	✓	✓	✓	✓	✓

© Objective Interface Systems, Inc. 2005

Custom Built Approach

- **Applicable anywhere**
- **Advantages**
  - Potentially fast
  - Potentially small
  - Tightly focused on current requirements
- **Disadvantages**
  - May not have time or $ to develop optimal performance
  - May not have time or $ to develop optimal size
  - Typically hardware specific—DMS issues over system life
  - Proprietary solution—vendor lock-in, also no interfaces to standardize
  - Not interoperable, can't incrementally migrate
  - More expensive to develop
  - More difficult to control lifecycle costs
  - Questionable reusability
    - Problem to extend
    - Short payback period

© Objective Interface Systems, Inc. 2005

Transport Level Approach

- **Characteristics**
  - HAL-C in SCA 3.0
- **Scope**
  - GPP
  - DSP
- **Advantages**
  - Standard API for inter-device communication
  - Provides some portability for:
    - Transport implementations
    - Applications that use the transport
- **Disadvantages**
  - Doesn't provide messaging format
    - No interoperability
  - Not zero-copy
    - Prevents use for intra-device communication
    - Not optimal
  - Doesn't allow for processing mobility
  - No specification of inter-component timing

© Objective Interface Systems, Inc. 2005

SCA-289 Component/Container Model

- **Characteristics**
  - SCA change proposal #289
  - Candidate for SCA 3.1
- **Scope**
  - GPP and GPP core on FPGA
  - DSP
  - FPGA
  - ASIC
- **Advantages**
  - Provides both inter-container and inter-device interface definition
  - Portability of components at source level
  - Replace-ability across technologies
  - Separation of concerns between platform provider and component author
  - Resource efficiency and performance
  - Minimal impact/changes required on existing component models
- **Disadvantages**
  - No specification of inter-component timing

© Objective Interface Systems, Inc. 2005

- **Characteristics**
  - Specialized ORBs for DSPs
  - Much less than 100K object code for both client and server support
- **Scope**
  - GPP and GPP core in FPGA
  - DSP
- **Advantages**
  - Well defined foundation, easy to integrate with GPP, OMG standard
  - Device-location-transparent logic
  - Code partially portable to/from GPP
    - Interfaces to functionality are portable
    - Implementation logic will depend on DSP-specific signal processing libs
  - Interoperable communications
- **Disadvantages**
  - Unfamiliar paradigm to most DSP engineers
  - Traditional footprint and performance concerns can become a cultural barrier (despite small, fast DSP ORBs!)
  - ORBs require some knowledge of object-oriented design
  - Generated code from IDL can be quite large (eg. Core Framework IDL)
  - No specification of inter-component timing

© Objective Interface Systems, Inc. 2005

- **Characteristics**
  - Still being defined by OMG (Objective Interface and Rockwell-Collins)
  - A subset of Minimum CORBA
  - Designed to support safety certification efforts (DO-178B)
  - More robust mappings to languages
  - Formal methods enhancements to IDL (better correctness)
- **Scope**
  - GPP and GPP core in FPGA
  - DSP
- **Advantages**
  - Same as Real-time CORBA plus:
    - Much smaller ORBs
    - Better testability and robustness
    - Correctness infrastructure makes for quicker deployment
- **Disadvantages**
  - Specification in progress
  - Still requires knowledge of O-O
  - No specification of inter-component timing

© Objective Interface Systems, Inc. 2005

Middleware Characteristics

Approach	Portable Logic	Portable Logic Shell	Portable Inter-logic Comm	Middleware FPGA Footprint (Slices/LEs)	Middleware Memory Footprint (kilobytes)	Execution Overhead (kilocycles)
Custom/ Proprietary	✘	✘	✘	varies	varies	varies
Transport (HAL-C)	✘	✘	✓	✘	0.2 – 5 (+ xport)	0.1 – 2
Real-time CORBA	✘	✓	✓	✘	45 – 100 Clnt + Svr (+ xport)	1.8 – 20
High Assurance CORBA	✘	✓	✓	✘	15 – 35 (+ xport)	0.9 – 4
SCA-289 Comp/Cont Approach	✘	✓	✓	80 – 120	20 – 35 (+ xport) 0.3/cpnt	0.1 – 3

© Objective Interface Systems, Inc. 2005

Last Slide

© Objective Interface Systems, Inc. 2005

Backup Slides

More information on SCA-289 Component/Container Model

© Objective Interface Systems, Inc. 2005

SCA-289 Component/Container Concepts

- **Component:**
  - unit of deployable functionality
  - independently defined unit of functionality of an application
- **Application:**
  - One or more components deployed as a unit to perform interesting work for the user/client
  - A configured and interconnected set of one or more components
- **Container:**
  - the immediate runtime environment in which a component instance executes
  - the provider of any local runtime services or APIs to components
  - the local invoker/controller/manager of the component
- **Class (a.k.a. which Component Implementation Framework):**
  - A particular language/API model to which components are written

© Objective Interface Systems, Inc. 2005

SCA-289 Component/Container Classes

- **Different types of processors require different ways of writing components.**
  - No “one language/API fits all”, especially when performance sensitive
- **GPP Class: normal first class component environments**
  - CORBA enabled environments
  - Existing component specifications are suitable
- **RCC Class: Resource-Constrained C language environments**
  - When C is available
  - When CORBA is not suitable
  - E.g. DSPs or Microcontrollers or RISC cores with limited memory
- **RPL Class: RTL-Programmable-Logic environments**
  - When RTL language is available (VHDL/Verilog)
  - When C is not available or not suitable
  - E.g. FPGAs and ASICs

© Objective Interface Systems, Inc. 2005

SCA-289 Component/Container System

Distributed/Embedded System/Platform

Processor X

ContainerA

Component instance1

Component instance2

ContainerB

Component instance3

Processor Y

ContainerC

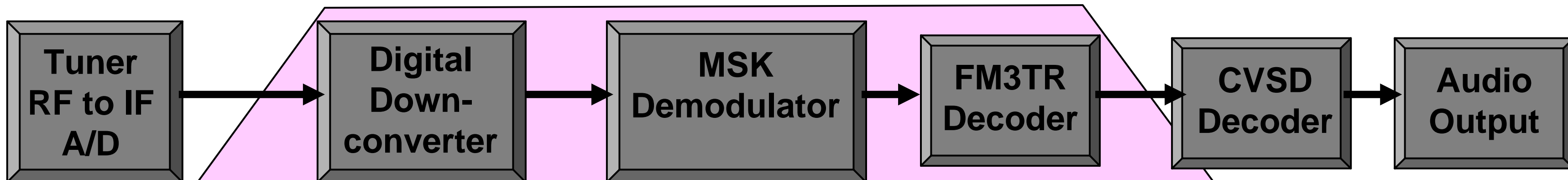
Component instance4

Component instance5

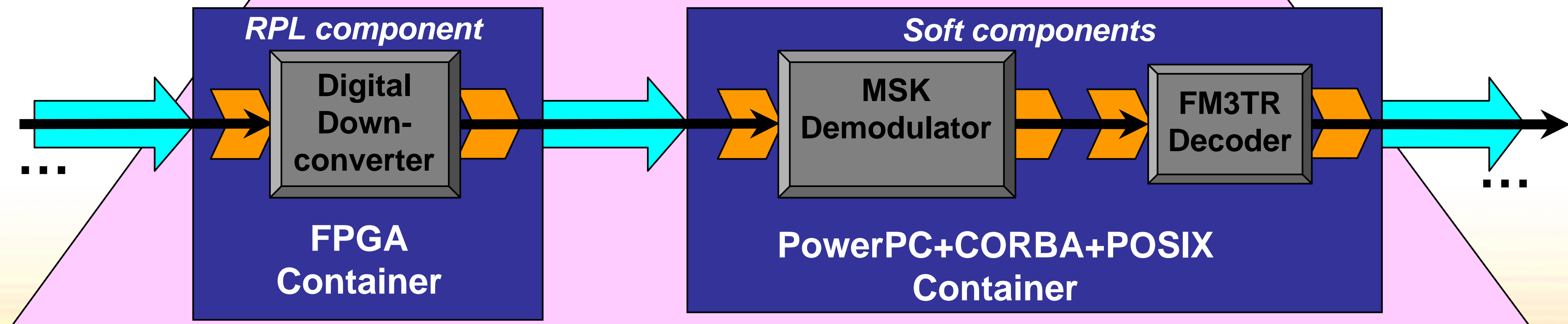
Component instance6

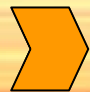
© Objective Interface Systems, Inc. 2005

Simple FM3TR Receiver

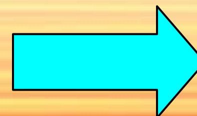


Component implementations in Containers





Components talk to their containers. Important interface for portability of components. APIs used by component authors.



Containers talk to containers. Important interface for interoperability/plug&play of containers (e.g. boards). Protocols/networks/busses.

Communication between components, conveyed by their containers

© Objective Interface Systems, Inc. 2005

SCA-289 Component/Container Portability Goals

- **A component implementation can move to same class of container ("like for like"), recompiling source**
  - RPL component written in VHDL ports between FPGA families or to an ASIC.
- **Use of platform/processor/container-specific features impedes portability.**
- **Portable "reference implementations" can be tweaked to use special features (e.g. Viterbi acccelerator on DSP)**
- **RCC components easily port and wrap into GPP environments.**

© Objective Interface Systems, Inc. 2005

SCA-289 Component/Container Replace-ability Goal

- **Enable Changes in technology/processor class with no impact on the rest of the application (other components)**
  - Change a filter from FPGA to (new faster) DSP
  - Change a modem from DSP to (new faster) GPP
  - Increase data rate requiring switch to (new faster) FPGA.
- **Enable simple addition of component implementations to existing components**
  - Both CCM & SCA support multiple implementations in a component package.
  - Allow adding FPGA implementation to component with GPP implementation without impacting application
- **Implies opaque interoperability between all classes of component implementations**

© Objective Interface Systems, Inc. 2005

Resource Efficiency and Performance Goals

- **Minimize “tax” for portability**
- **Minimize “tax” for interoperability**
- **Enable appropriately small footprint**
  - Satisfy the fanatics
- **Enable full performance usage of inter-processor hardware interconnections**
  - busses, networks, fabrics, NICs
- **Enable full performance for collocated component instances**
- **Enable statically pre-combinations of component implementations**
- **Enable zero copy operation**
  - To inter-processor interconnects
  - Between collocated components
  - Between input and output of a component

© Objective Interface Systems, Inc. 2005

SCA-289 Component/Container Component Interfaces

- **Management interfaces**
  - Generic for deployment, configuration, introspection, lifecycle
  - SCA has CF::Resource as exposed external interface
    - No container/local interface
  - CCM has:
    - CCMObject as exposed external interface
    - EnterpriseComponent and SessionComponent as local container-to-component base interface
    - CCMContext and SessionContext as local component-to-container interface
- **Inter-component interfaces**
  - IDL-defined user and provider ports
  - CCM specializes event ports and stream ports
- **Local O/S APIs**
  - CCM says nothing
  - SCA defines POSIX profile

© Objective Interface Systems, Inc. 2005

RPL: RTL-Programmable Logic environments

- **All interfaces are OCP**
  - An open standard for how "IP Cores" are connected.
  - Independent of VHDL vs. Verilog
  - A range of performance options
- **Management interface**
  - Simplified from (CCM or SCA) component model
  - Initialize/start/stop/release/test on one OCP "thread"
  - Configure read/write on second OCP "thread"
- **Intercomponent interface**
  - Burst read/write transactions on OCP-port
    - One OCP port per IDL port per direction
  - Implementation chooses master or slave role
  - Implementation chooses FIFO or random access style
- **Local interfaces**
  - Clocks and local memory access (several styles)

© Objective Interface Systems, Inc. 2005