Middleware for DSPs and FPGAs

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

OMG Real-time and Embedded Workshop 2005

© Objective Interface Systems, Inc. 2005
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)
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
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
### Comparing the Technologies

<table>
<thead>
<tr>
<th>Processing Device</th>
<th>Speed</th>
<th>Power</th>
<th>Initial Cost</th>
<th>Cost per Unit</th>
<th>Design Investment Longevity</th>
<th>Skill Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPP</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>DSP</td>
<td>1 – 1.8</td>
<td>0.5</td>
<td>2</td>
<td>0.8 – 3</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>FPGA</td>
<td>3 – 20</td>
<td>0.25 – 2</td>
<td>4</td>
<td>2 – 4</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>ASIC</td>
<td>3 – 50</td>
<td>0.1 – 0.3</td>
<td>20</td>
<td>0.1 – 0.2</td>
<td>--</td>
<td>---</td>
</tr>
</tbody>
</table>

© Objective Interface Systems, Inc. 2005
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)
Comparisons

Middleware for DSPs, FPGAs, & ASICs

- **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
  - FPGA: number of logical units
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
Middleware Benefits for DSP and FPGA Designers

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

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

**Processing Mobility:**
Moving Functions Independently

- Function A: GPP → DSP
- Function B: GPP → FPGA
- Function C: FPGA → DSP

© Objective Interface Systems, Inc. 2005
Custom
Transport level (HAL-C)
Real-time CORBA
High-Assurance CORBA
SCA-289 Component/Container Model
## Applicability of Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>GPP</th>
<th>DSP</th>
<th>GPP core on FPGA</th>
<th>FPGA</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom/Proprietary</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Transport (HAL-C)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Real-time CORBA</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>High Assurance CORBA</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SCA-289 Comp/Cont Model</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

© Objective Interface Systems, Inc. 2005
Custom Built Approach

Middleware for DSPs, FPGAs, & ASICs

- 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
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
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
Middleware for DSPs, FPGAs, & ASICs

Real-time CORBA

- 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
High Assurance CORBA

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

<table>
<thead>
<tr>
<th>Approach</th>
<th>Portable Logic</th>
<th>Portable Logic Shell</th>
<th>Portable Inter-logic Comm</th>
<th>Middleware FPGA Footprint (Slices/LEs)</th>
<th>Middleware Memory Footprint (kilobytes)</th>
<th>Execution Overhead (kilocycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom/Proprietary</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>varies</td>
<td>varies</td>
<td>varies</td>
</tr>
<tr>
<td>Transport (HAL-C)</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td></td>
<td>0.2 – 5 (+ xport)</td>
<td>0.1 – 2</td>
</tr>
<tr>
<td>Real-time CORBA</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>45 – 100 Clnt + Svr (+ xport)</td>
<td>1.8 – 20</td>
</tr>
<tr>
<td>High Assurance CORBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15 – 35 (+ xport)</td>
<td>0.9 – 4</td>
</tr>
<tr>
<td>SCA-289 Comp/Cont Approach</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20 – 35 (+ xport) 0.3/cpnt</td>
<td>0.1 – 3</td>
</tr>
</tbody>
</table>
Last Slide
Backup Slides

More information on
SCA-289 Component/Container Model
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
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
Simple FM3TR Receiver

Middleware for DSPs, FPGAs, & ASICs

Component implementations in Containers

RPL component

- Tuner RF to IF A/D
- Digital Down-converter
- MSK Demodulator
- FM3TR Decoder
- CVSD Decoder
- Audio Output

Soft components

- Digital Down-converter
- MSK Demodulator
- FM3TR Decoder

FPGA Container

PowerPC+CORBA+POSIX Container

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 accelerator on DSP)
- RCC components easily port and wrap into GPP environments.
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
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
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
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)