PRISMTECH
Productivity Tools & Middleware

Real-time & Embedded Systems Workshop July 2007

Using a Hardware ORB to Facilitate Seamless Communication with FPGAs in DRE Systems

Fred Humcke
Senior Hardware Architect
PrismTech Corporation

SDR/SCA Overview

- SDR is gaining ground in military and commercial markets
- SDR adoption is being driven by the SCA
  - The SCA is a mature specification
  - SCA compliance is being required on more and more military platforms
  - Companies using the SCA are starting to report that it does save waveform porting time
- The industry is moving closer to producing the first true software defined radios
- Adoption has been slowed by some preconceptions and misconceptions about the SCA that may no longer be valid

SDR/SCA Misconceptions

- SCA operating environment components too large
  - They take up valuable system resources such as memory
- CORBA is too low in performance
  - The TCP/IP stack adds too much overhead for simple data transfer
- The SCA is complex and difficult (expensive) to learn and utilize
- There is no SCA solution for hardware objects
  - waveform components implemented on silicon devices such as FPGAs and ASICs

We've Heard This Before

- These generic problems are similar to issues encountered when moving to a higher level of abstraction
  - Assembly to C
  - Schematic capture to VHDL
- How did C/VHDL overcome these obstacles??
  - A strong driver to push it along
  - Enablers to make it practical
- The Driver was a strong business case based on portability
  - Write C once and run it on many machines
  - Write VHDL once and synthesize it on many platforms
- The enablers were technology breakthroughs

Generic Enablers

- Increases in processor performance
- Increases in memory size and density
- Smaller software footprint
- More efficient software processing tools
  - compilers, synthesizers
- High level tools to remove complexity from user
  - operating system, emulation and debugging environments
  - Simulators, automatic place and route

The Result

- The Driver:
  - Strong business case centered around portability, reuse and maintenance
- The Enabler:
  - Technological breakthroughs that resulted in ease of use and reduced penalty to performance
- The Result:
  - Adoption of the higher level of abstraction
  - The use of C and VHDL became widespread

SDR/SCA Enablers

- Small footprint embedded ORBs for GPPs and DSPs
  - ORBs that require KBs of memory compared to MBs of memory
- Small footprint second generation OE
  - An entire SCA OE middleware suite consisting of ORB, ORB services and core framework require less memory than ORBs of just a few years ago
- Domain specific SDR development tools
  - Allows development at higher levels of abstraction
  - Provides automatic SCA code generation
  - Reduces the complexity of the SCA
- CORBA enablers implemented in hardware
  - CORBA/SCA communications are brought directly onto hardware platforms such as FPGAs and ASICs

A New Era

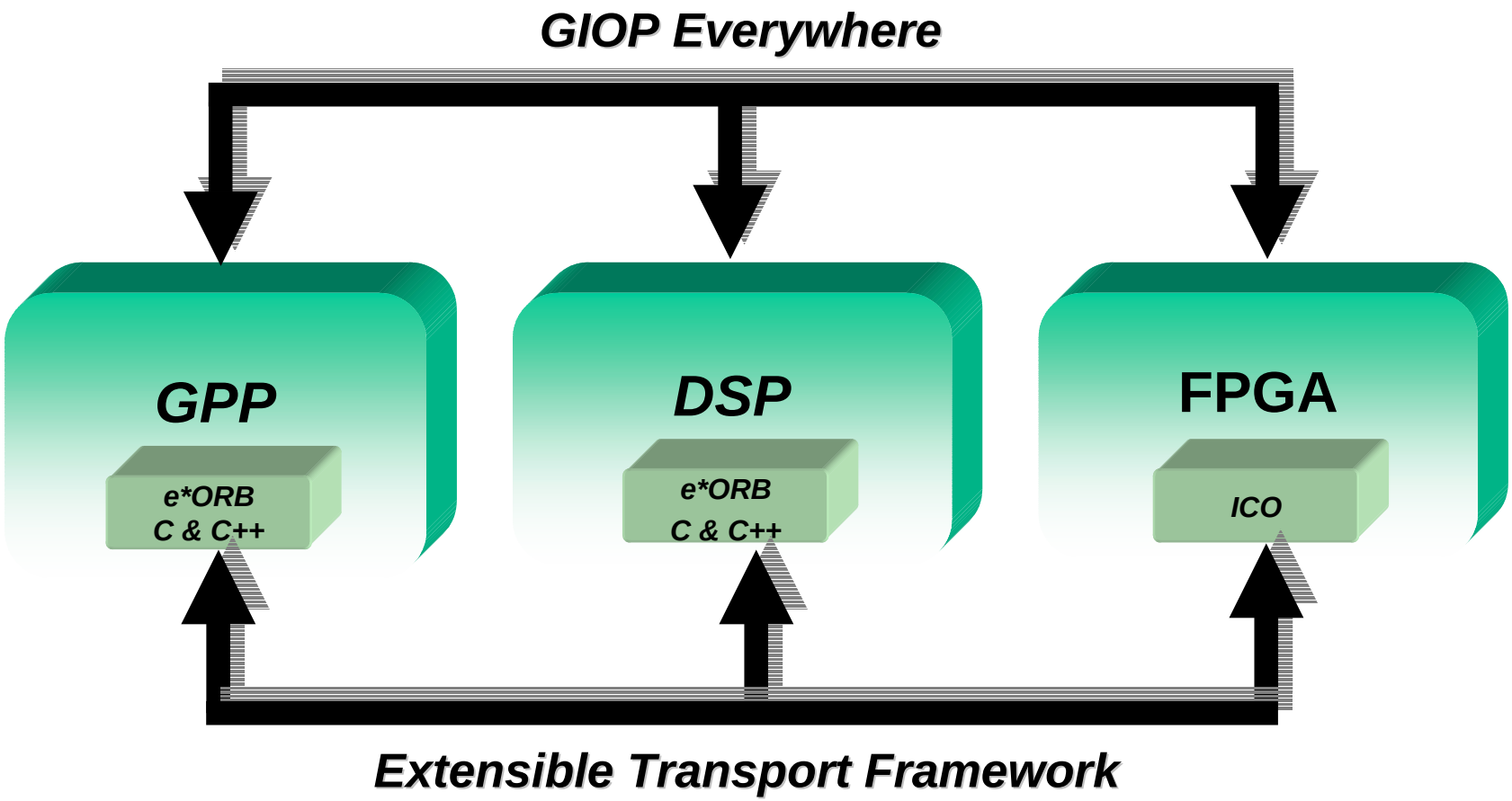
- The strong business case for the SCA and the development of technological enablers have brought SDR to the edge of a new era of industry wide acceptance

- The industry now has the critical innovations needed to allow ubiquitous SDR adoption in military and commercial markets

A Key Enabler

- The goal
  - Realize the cost savings of portability by moving the SCA as close to the antenna as possible.
- Small software footprint allows the SCA to run on embedded GPPs and DSPs
- The next obstacle for the SCA to cross lies in the interface between embedded processors and the digital radio hardware
- A key enabler is needed to bring the SCA to waveform components implemented on silicon devices

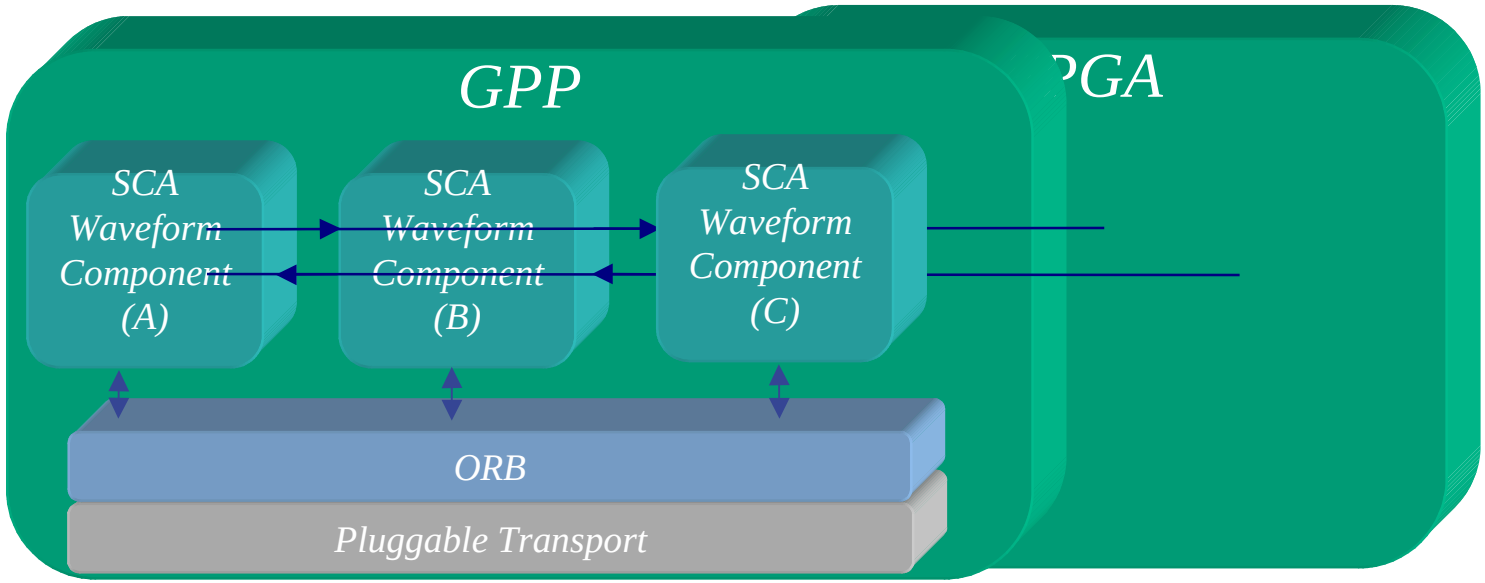
PrismTech's Embedded Middleware Strategy



The first standards-based, high-performance, low-footprint, fully-interoperable COTS middleware solution that can be deployed across multiple processor types, including GPP, DSP, & FPGA environments

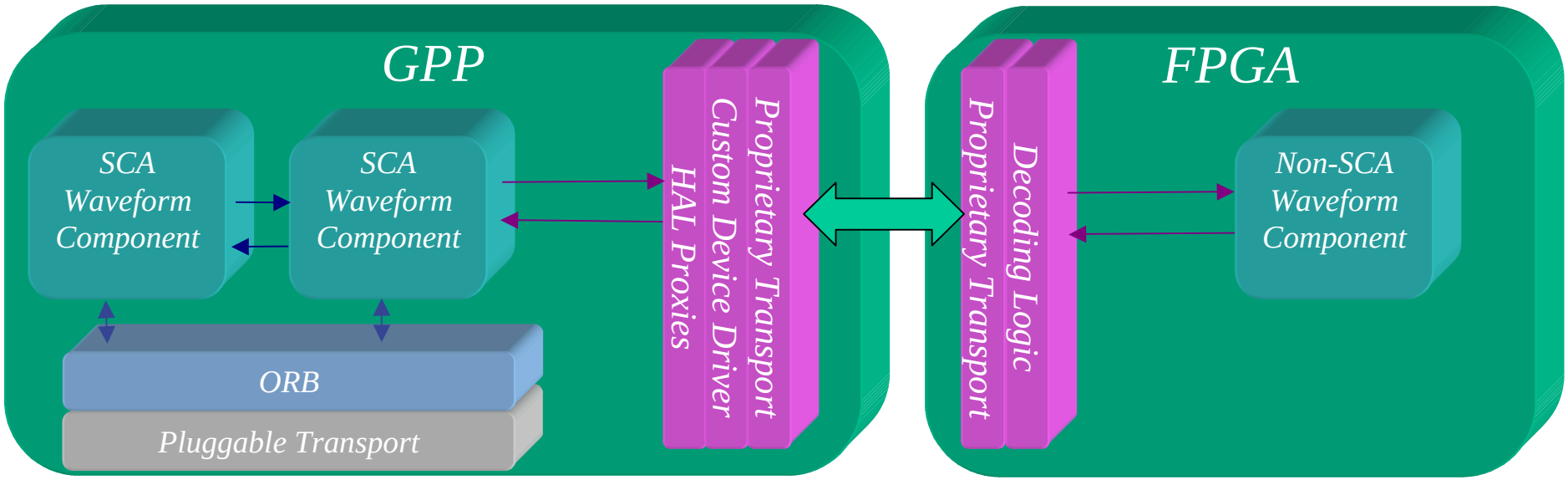
What We Want to Do

- Migrate certain waveform components into FPGA implementations… while still maintaining SCA compatibility



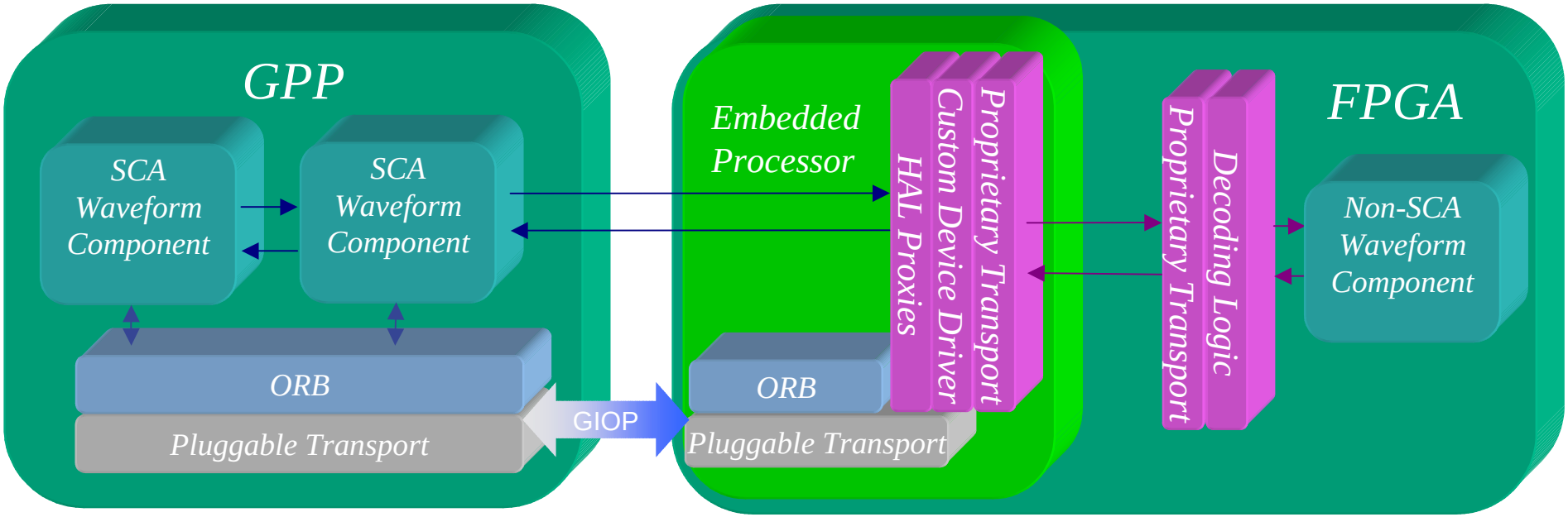
Historical Approaches

- Attempts to implement this resulted in non-SCA compliant components in the FPGA and added complexity for the radio developer



Other Attempts

- Other attempts to implement this have only moved the problem internal to the FPGA. The result is still non-SCA compliant components in the FPGA and a likley increase in added transport overhead

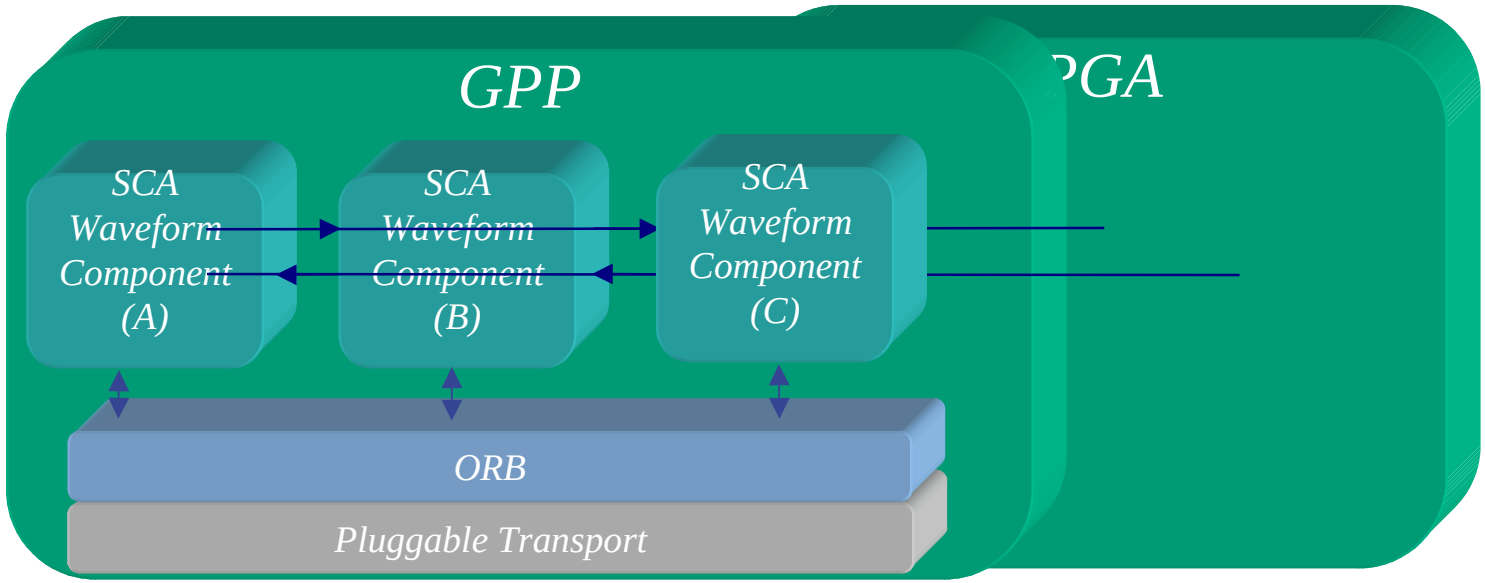


It’s Too Complicated!

- Current implementations are too complex
- They force the radio developer to become involved with middleware
- They don’t scale
  - Must be redone for each new SCA application implementation
- The resulting implementations contribute to the misconceptions about the SCA
- Its time to step back…

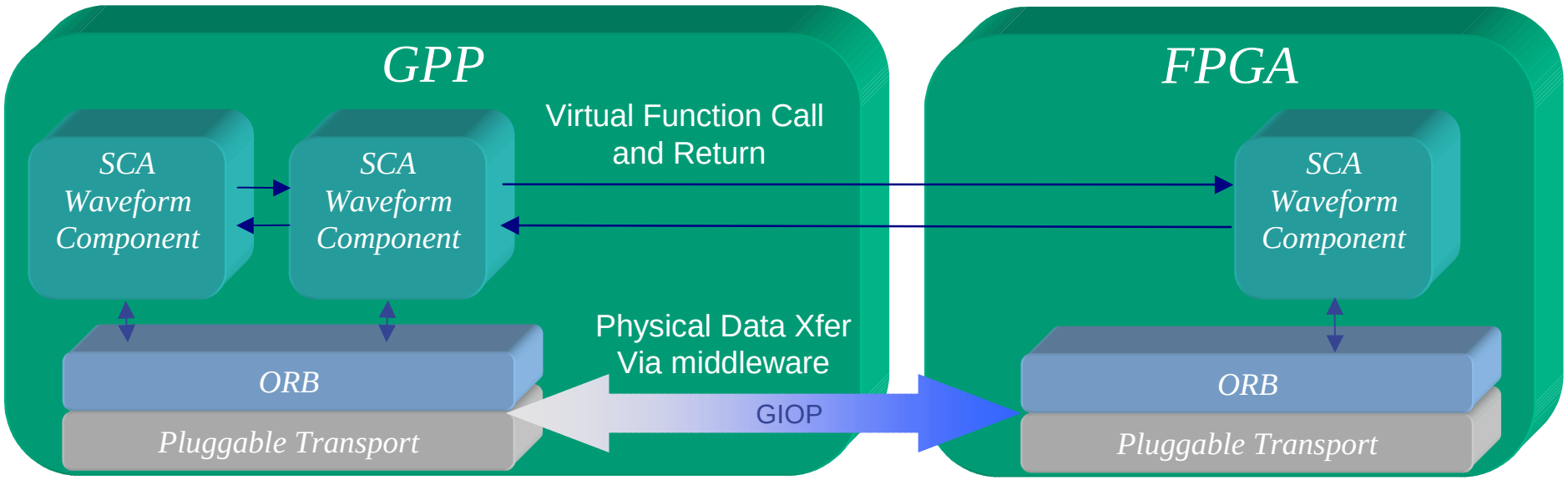
What We Want to Do

- Migrate certain waveform components into FPGA implementations… while still maintaining SCA compatibility



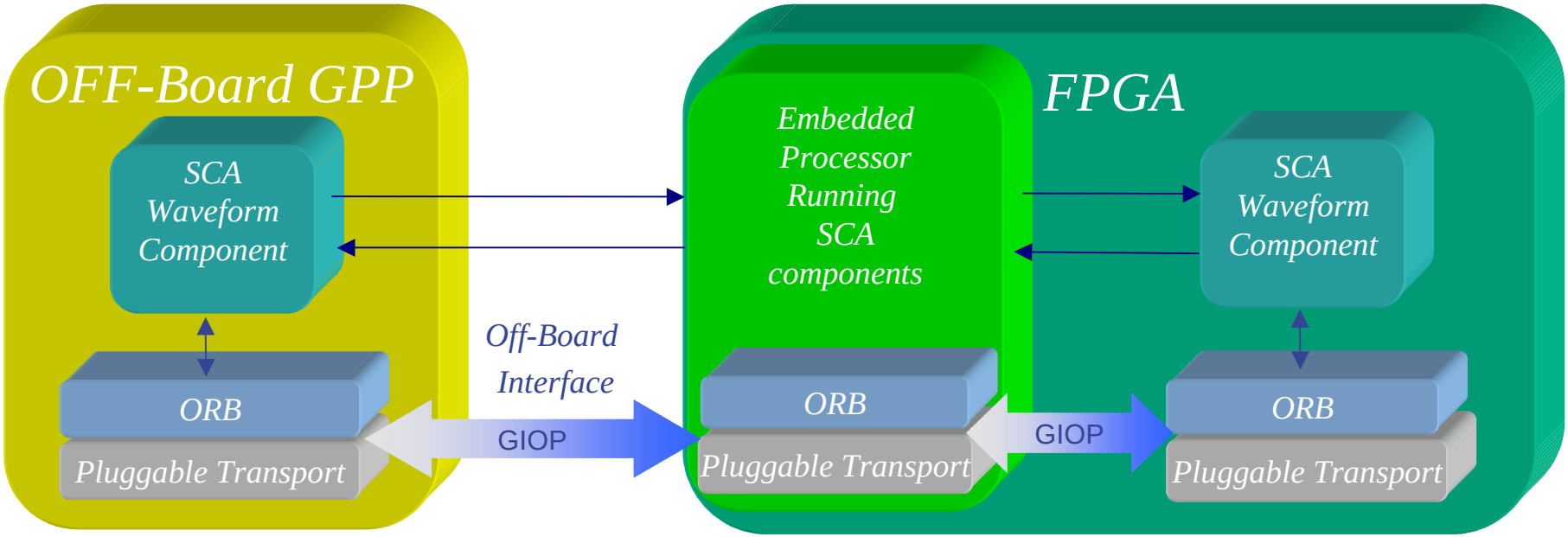
What if…

- Using an ORB, SCA compliance is maintained and overhead is reduced



What about Single Chip Solutions

- For System on a Chip (SoC) solutions, both the embedded processor and off-board processors can communicate, via an ORB with waveform components implemented in hardware; thus, SCA compliance is maintained.



Benefits of Hardware ORB

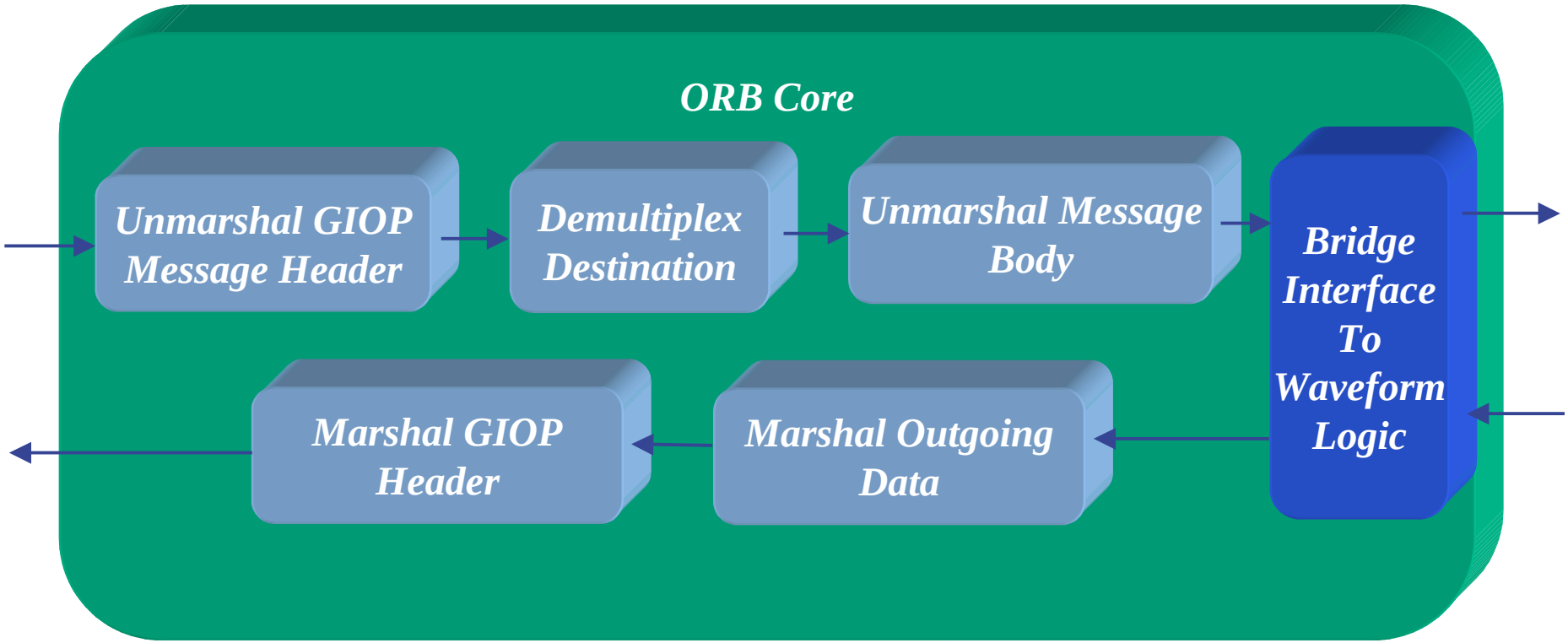
- A hardware implementation of a CORBA ORB
  - Eliminates HAL proxies and overhead
  - Eliminates the need for embedded processor in an FPGA for the sole purpose of meeting SCA compliance
    - Saves valuable FPGA resources and does not require chips from the high end of the cost range.
    - Solves the SCA compliance problem instead of moving it from the external bus to the embedded processor's local bus
  - A hardware ORB is a perfect fit for SoC solutions
    - Maintains SCA compliance across all waveform components of the system
    - Ensures portability and reusability of all waveform components

What Does A Hardware ORB need to look like?

- Can't be as complex as a software ORB
  - No need to implement the entire CORBA standard
  - All data types are not necessary to support SDR development
- Can't be as large as a software ORB
  - Can't just take software ORB and turn it into VHDL
  - Must minimize gate count
- Can't be as slow as a software ORB

What's In a Hardware ORB?

- It is a Generic Inter-ORB Protocol (GIOP) processor consisting of several different functional blocks

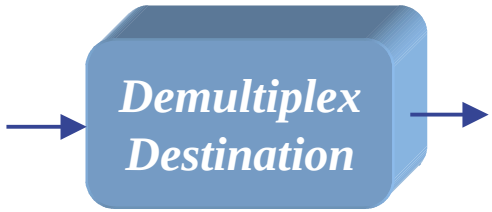


Message Header Processing

- The ORB must process the header fields and remove padding for different types of GIOP messages
  - A Request Message
    - GIOP Message Header Fields
    - GIOP Request Header Fields
    - Request Body
  - A Reply Message
    - GIOP Message Header
    - GIOP Reply Header
    - Reply body

Determine the Destination

- The ORB must demultiplex the destination header fields in order to determine the waveform logic being accessed
  - Object Key
  - Operation Name
- This is similar to the address decode function in typical hardware interfaces

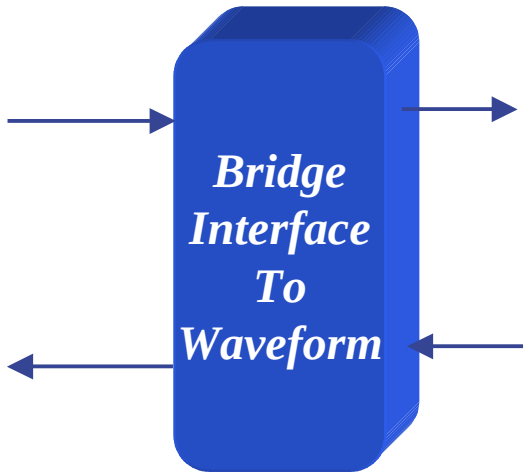


Build the Output

- For GIOP messages that require a write, the ORB must unmarshal the message body
  - Pack the bytes of the message body into data words that match the destination width
  - Select the required write handshaking signals
  - Remove padding bytes between words
- If the GIOP message is a read, the required read handshaking signal must be selected

Interface to Waveform Logic

- The bridge interface maps the ORB's handshaking signals and data busses to the waveform logic interface
  - Similar to the Stubs and skeletons of a software ORB
  - A standard such as OCP could be used for the bridge



Build Outgoing GIOP Message

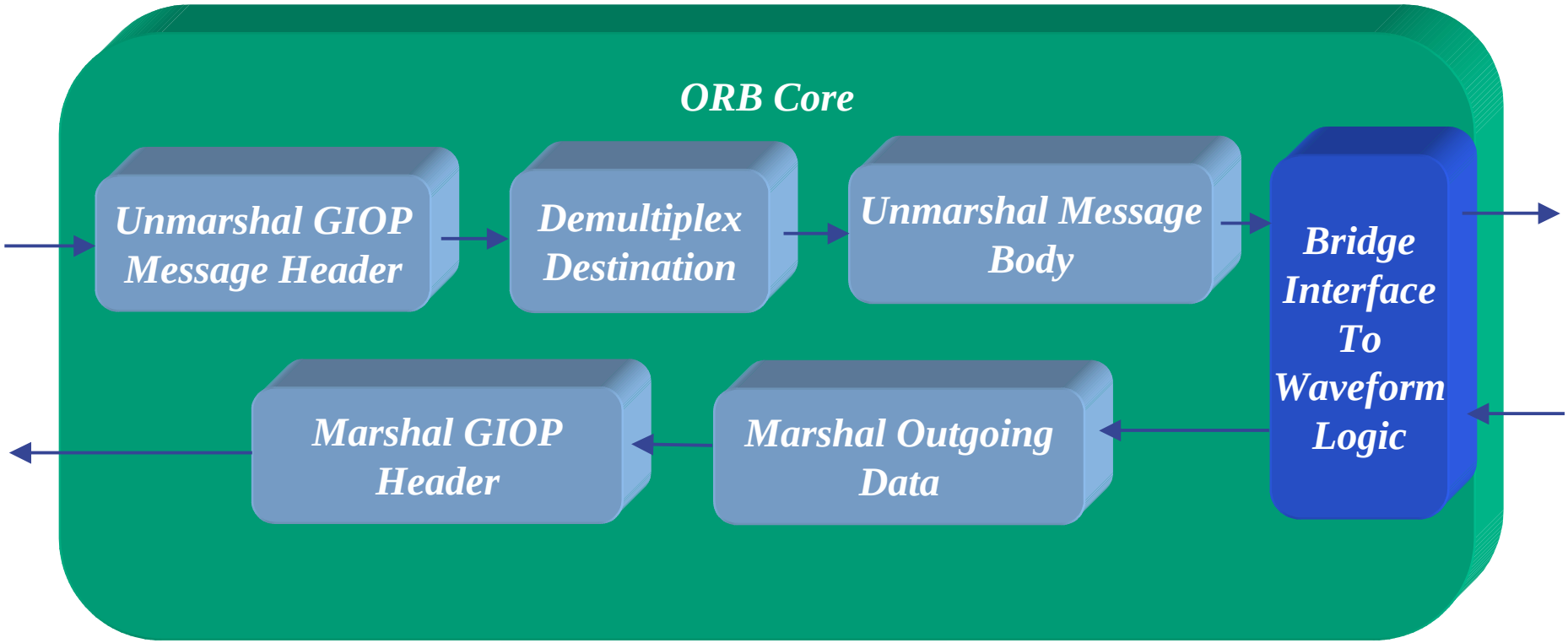
- For GIOP messages that require a read, the ORB must marshal data from the waveform logic
  - Pack the received words into the return message body
  - Add padding bytes between words

Build The Outgoing GIOP Header

- The ORB must generate header fields and padding for different types of GIOP messages
  - A Request Message
    - GIOP Message Header Fields
    - GIOP Request Header Fields
    - Request Body
  - A Reply Message
    - GIOP Message Header
    - GIOP Reply Header
    - Reply body

Basic Internal Blocks of a Hardware ORB

- The internal blocks of the ORB provide support for typical GIOP message types



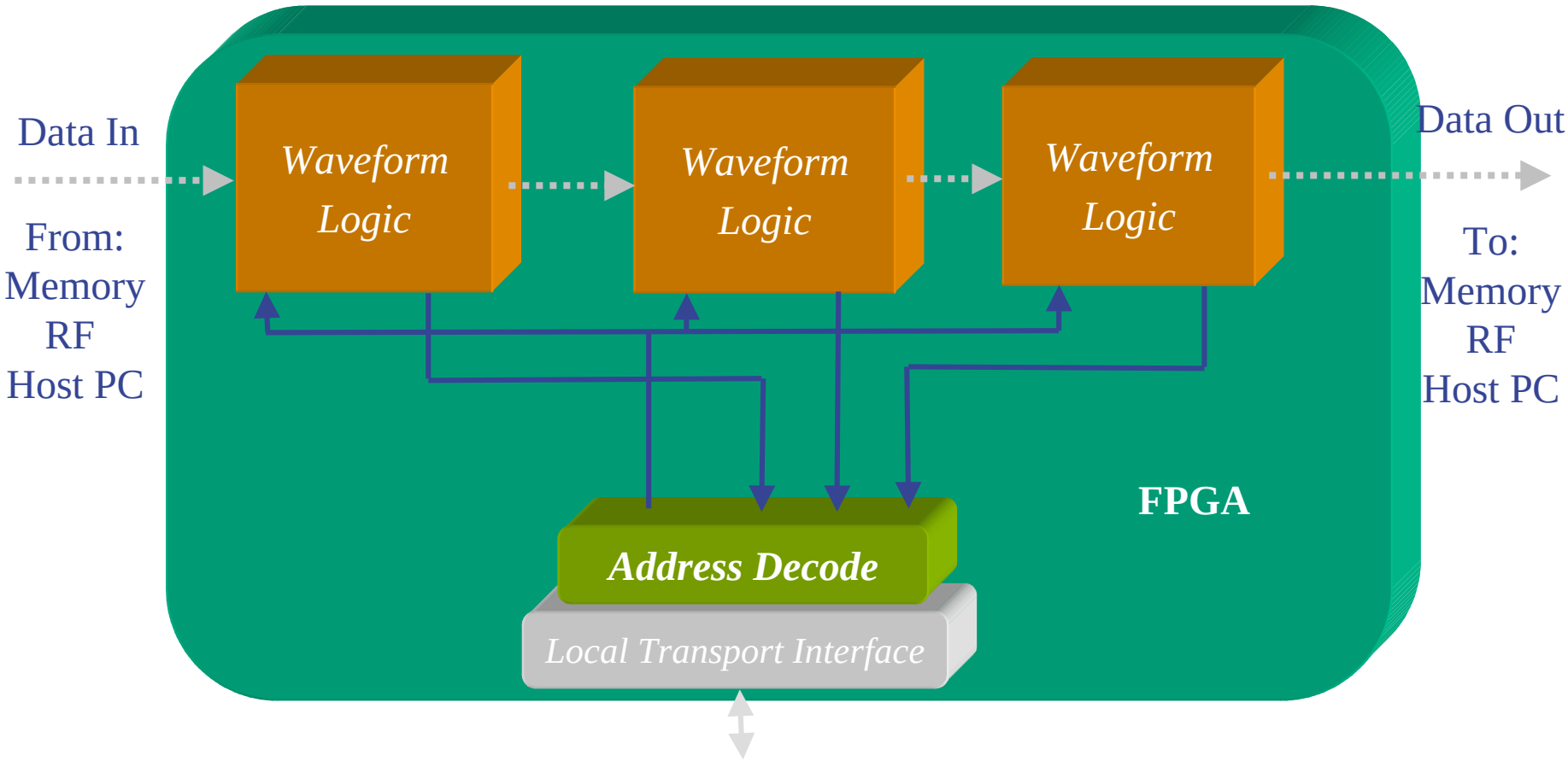
PrismTech's ICO

- PrismTech has developed a hardware implementation of an ORB
- It is called the Integrated Circuit ORB (ICO)

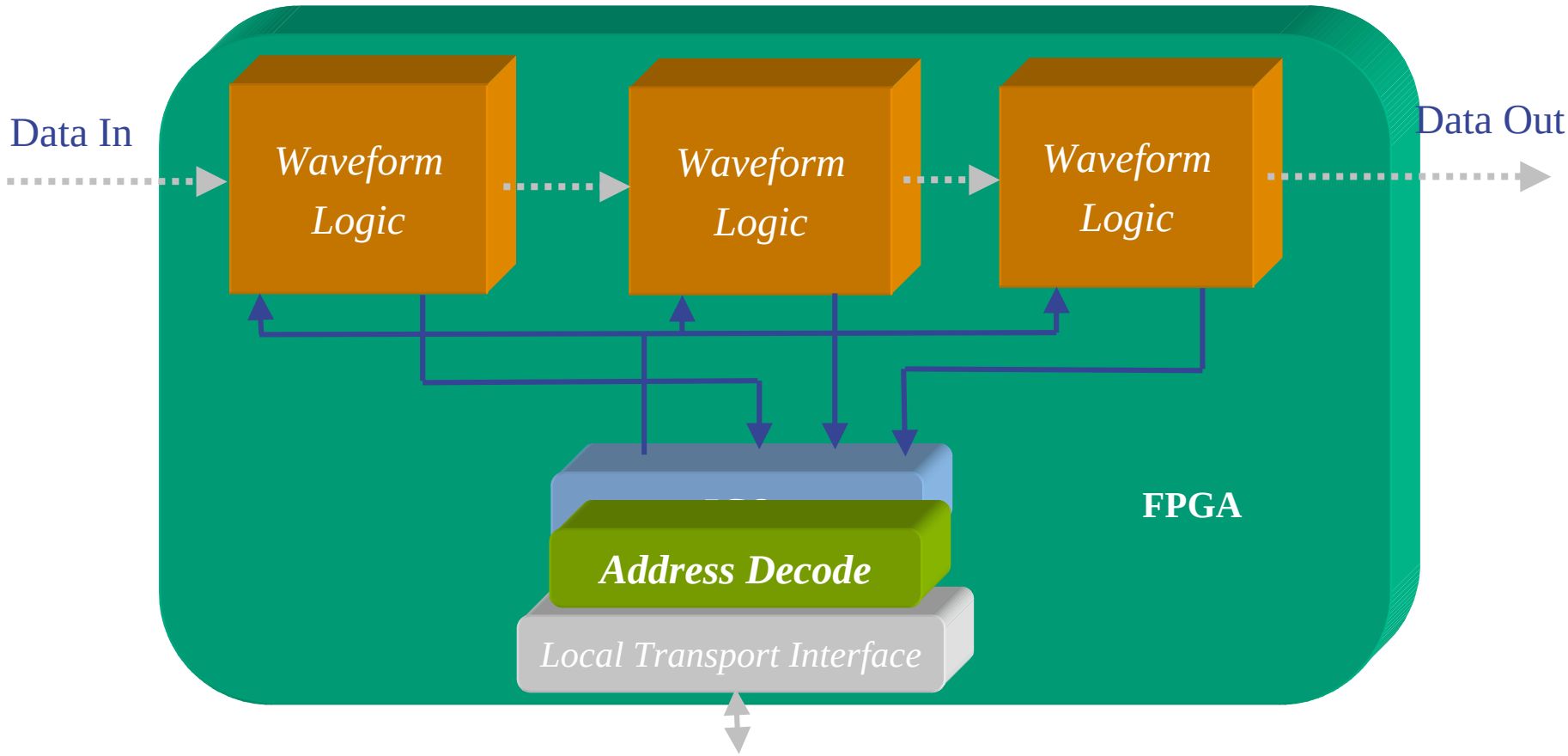
PRISMTECH
Productivity Tools & Middleware

ICO in the FPGA Design

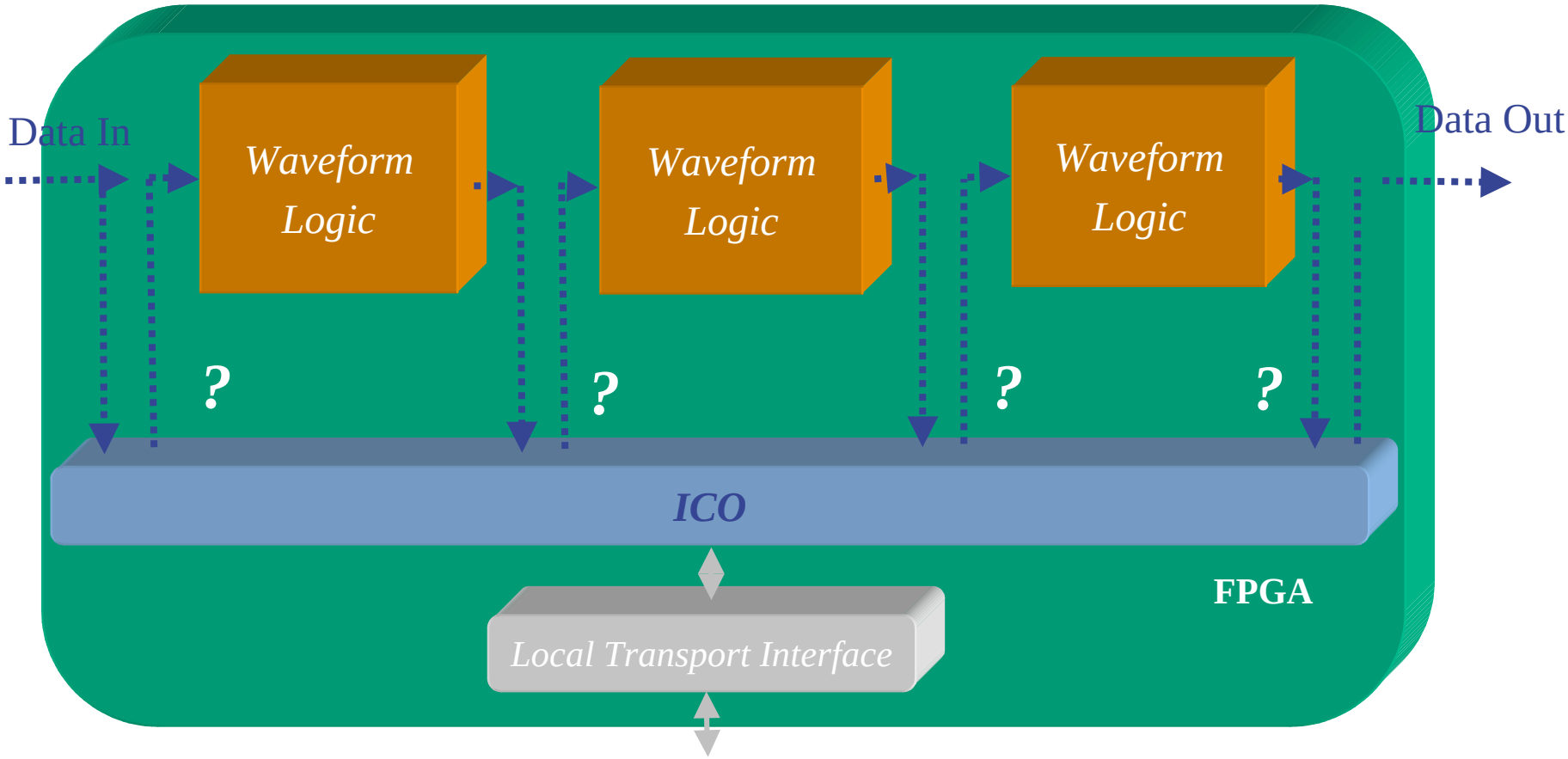
Generic FPGA Data Flow Example



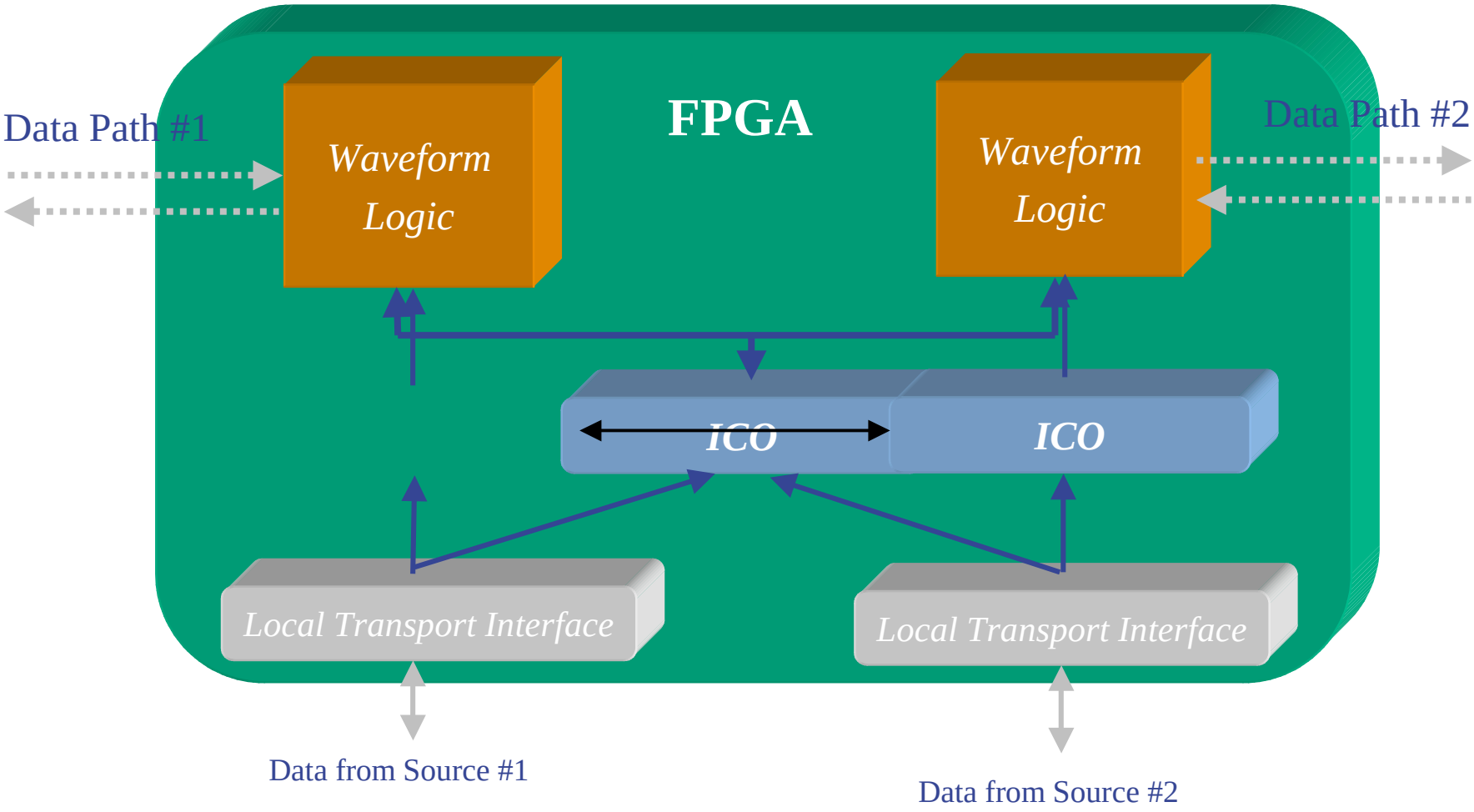
ICO FPGA Data Flow Example



ICO and Inter/Intra Chip Communication



ICO and Parallel Processing



ICO In FPGA Summary

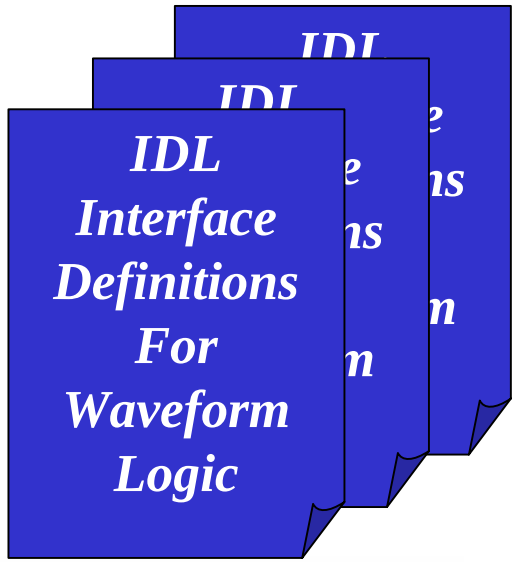
- ICO may be used for inter and intra board communications
- System performance requirements will determine the nature of its deployment in FGPA designs
  - The data speed
  - The burstiness of the data
  - The ultimate portability of the interface
- The system considerations are analogous to Assembly vs. C language choices
  - Assembly is still used where high performance is critical and C is used for portability

PRISMTECH
Productivity Tools & Middleware

The ICO Design Flow

ICO Design Flow

- The radio developer defines component to be implemented in H/W
- IDL definitions of the component interfaces are made



IDL Basics

- The Interface Definition Language (IDL) is independent of any particular programming language
  - enables interfaces to be defined generically
- Components of CORBA-based distributed applications can be written in different languages
  - Java, C, C++ or VHDL
- IDL must be translated into the appropriate programming language
  - Language constructs are defined by CORBA's IDL mapping for that language
  - PrismTech has developed a compiler for IDL to VHDL

Writing IDL for FPGA Components

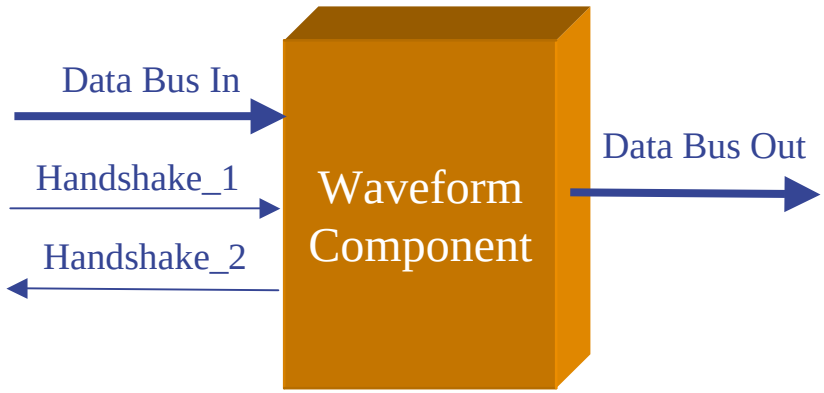
- IDL defines the logical function to be performed
- A CORBA object's interface defines the operations the object provides and how a client interacts with the object
- The interface consists of a list of operations (functions) and their I/O parameters

```
Generic IDL Interface
return_type operation_name (direction data_type parameter_name)

Interface SimpleObject
{
   viod operation1 (in unsigned long p1);
   void operation2 (out unsigned long p1);
};
```

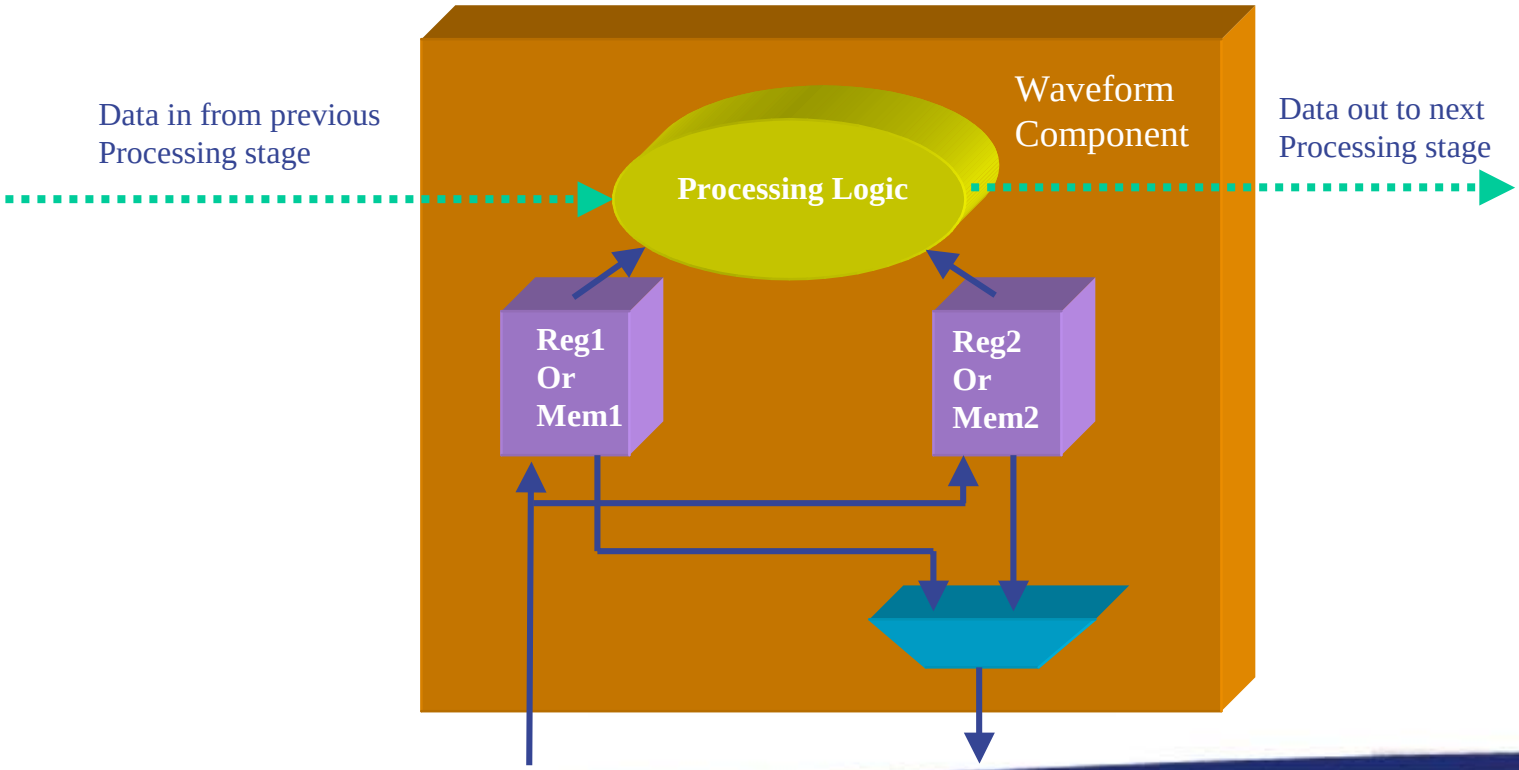
Writing IDL for FPGA Components

- Hardware physical interfaces typically define the physical nature of the I/O connections of the block
  - Data bus widths, handshaking signals, etc.
- This is not what we want for IDL!
  - IDL should define the functions that this block performs and the I/O parameters these functions require



Writing IDL for FPGA Components

- An IDL compiler maps the parameters defined in the IDL to physical memory elements *inside* the component.
- For example, the same elements that would require an address in the address map.



Writing IDL for FPGA Components

- One might be tempted to write an IDL interface that maps to each memory element being read and written
- However, this is not a logical definition of the functionality

```
Interface SimpleObject2
{
   void wrReg1 (in unsigned long p1);
   void rdReg1 (out unsigned long p1);
   void wrReg2 (in unsigned long p1);
   void rdReg2 (out unsigned long p1);
};
```

Writing IDL for FPGA Components

- Additionally, IDL gives the flexibility to read or write multiple parameters of a functional block
  - Maps to reading or writing multiple registers at the same time

```
Interface SimpleObject3
{
   void wrBoth (in unsigned long p1, in unsigned long p2);
   void rdBoth (out unsigned long p1, out unsigned long p2);
};
```

Writing IDL for FPGA Components

- Or, IDL gives the flexibility to read *and* write multiple parameters of a functional block
  - Maps to reading and writing multiple registers at the same time

```
Interface SimpleObject
{
   void wrRdReg1 (inout unsigned long p1);
   void wrRdReg2 (inout unsigned long p1);
};
```

Writing IDL for FPGA Components

- The designer can send or read a stream of data using a sequence

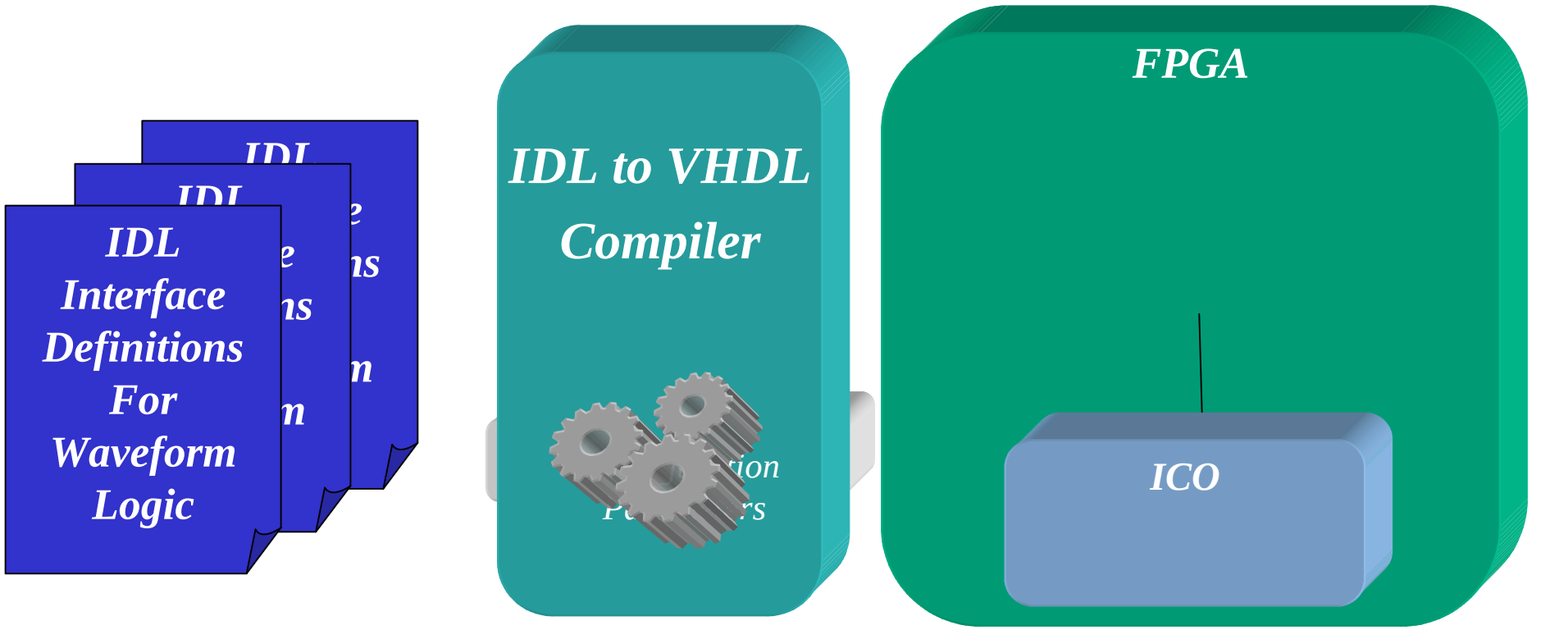
```
Interface SimpleObject
{
   void wrLotsofData (in sequence p1);
};
```

CORBA Data Types Supported by ICO

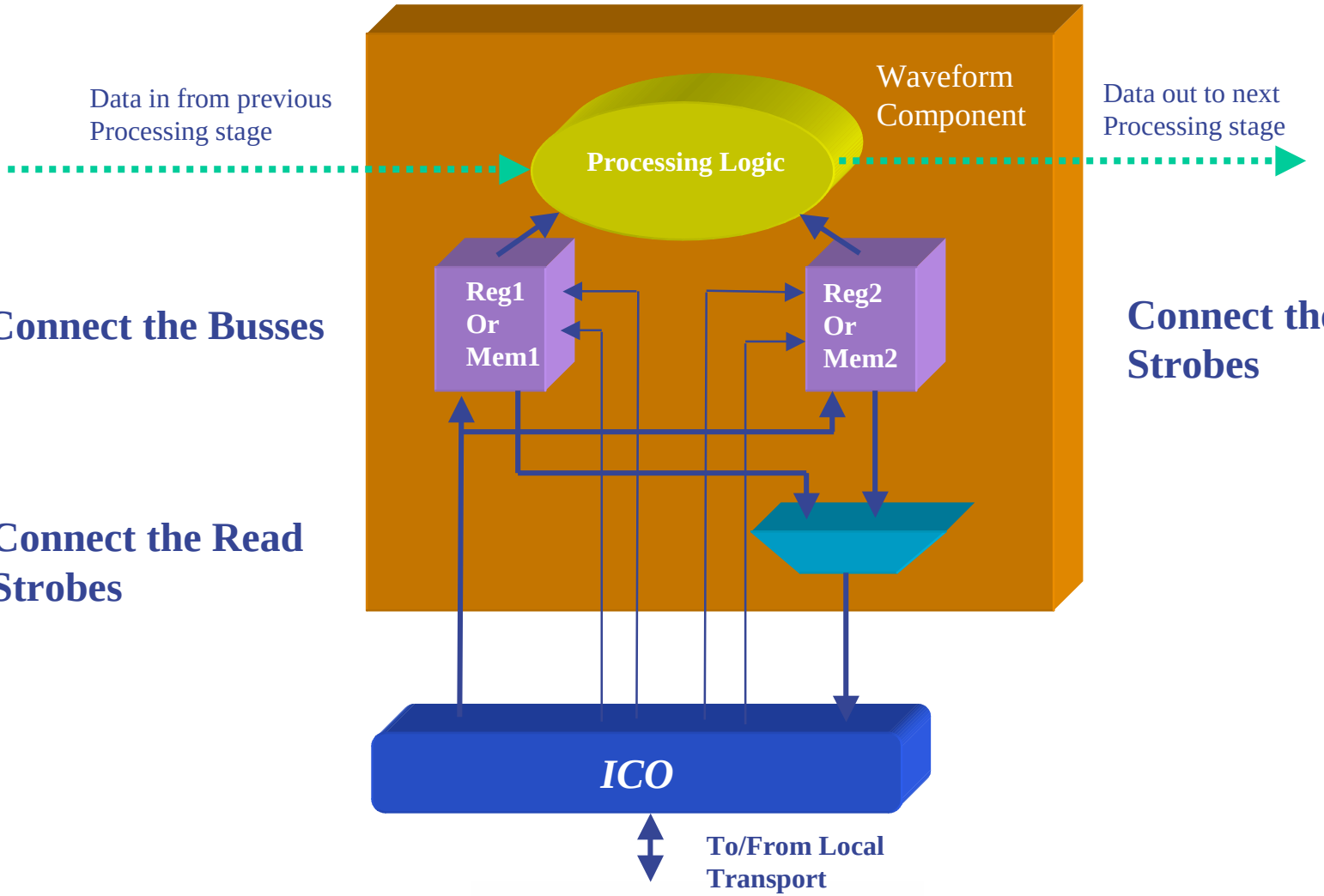
- CORBA data types supported by ICO (so far)
  - Simple Data types
    - Char, octet,
    - Unsigned short
    - Unsigned long
    - Unsigned long long
  - Complex Data types
    - Strings of simple types
    - Sequence of simple types
    - Any of simple type
    - Structures of simple types, strings and anys
- CORBA standard supports additional complex data types not typically need in FPGA processing

ICO Design Flow

- The IDL to VHDL compiler generates configuration parameters and a VHDL binding for the ICO



Connecting ICO to FPGA Components



Connect the Busses

Connect the Write Strobes

Connect the Read Strobes

PRISMTECH
Productivity Tools & Middleware

Designing SCA compliant components using ICO

ICO Support for the SCA

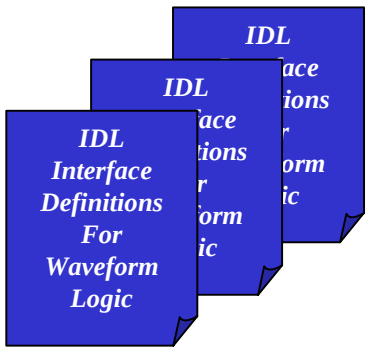
- ICO supports the CORBA constructs required for the SCA Resource and Device Interface
  - Testable Object
    - runTest
  - Resource
    - Start, Stop
  - Life Cycle
    - Initialize
    - Release Object
  - Port
    - Connect
    - Disconnect
  - Property Set
    - Configure
    - Query
  - Port Supplier
    - Get(port)

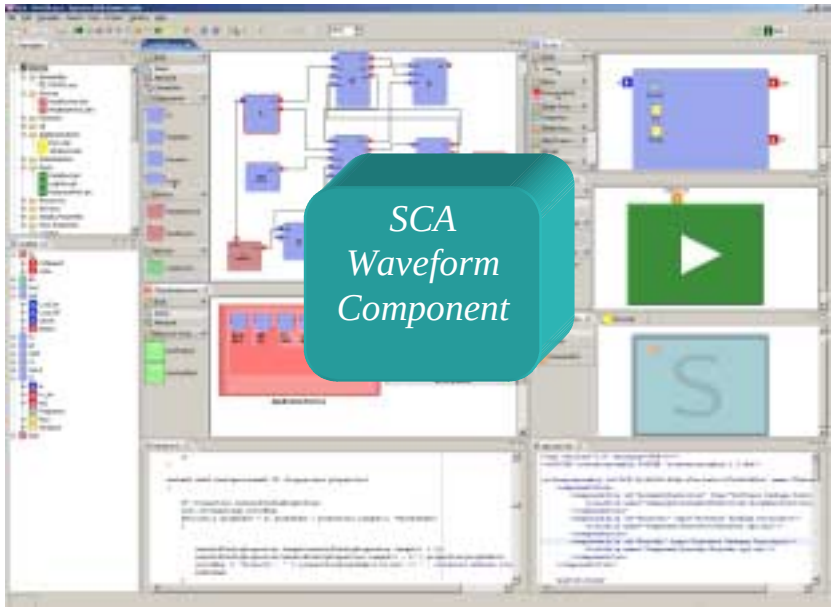
SCA Devices

- The SCA executableDevice type is probably not appropriate for FPGAs as there is no code being executed
- The loadableDevice type must initially be handled by an external processor that stands in for ICO
- Reconfigurable FPGAs allow the loadableDevice component to reside on the FPGA and dynamically load other parts of the FPGA

MDE Tools Can Greatly Improve Developer Productivity

- Model Driven Engineering tools can be used to generate a VHDL SCA waveform component container

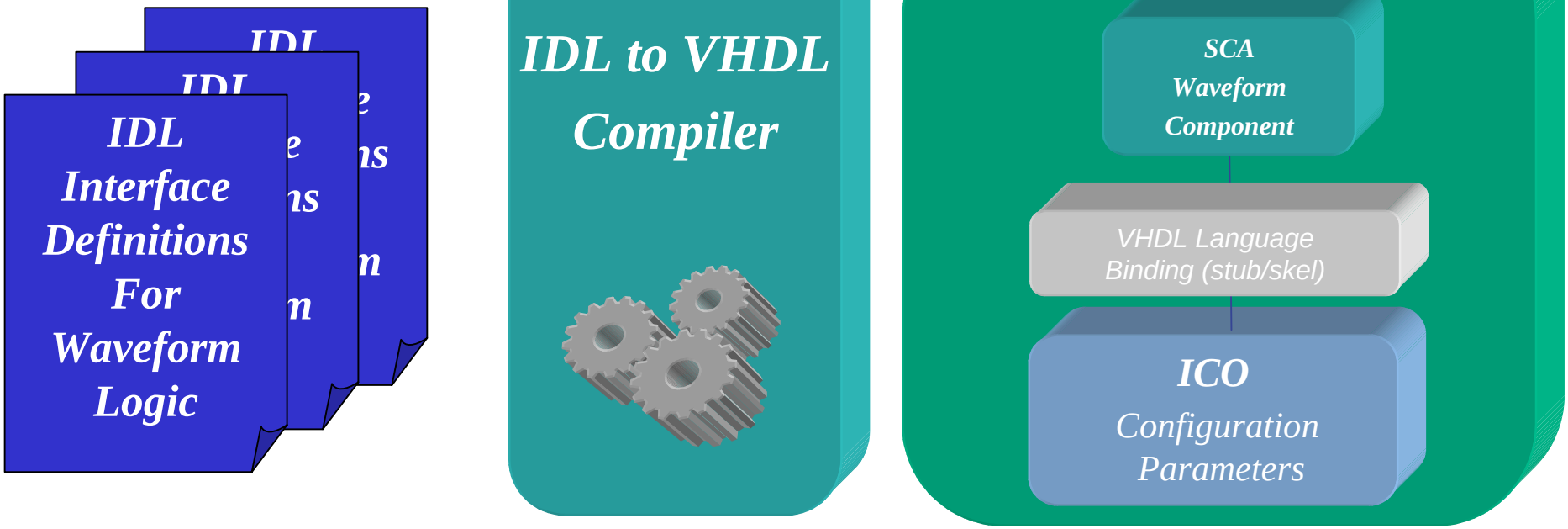




VHDL
Source
Code

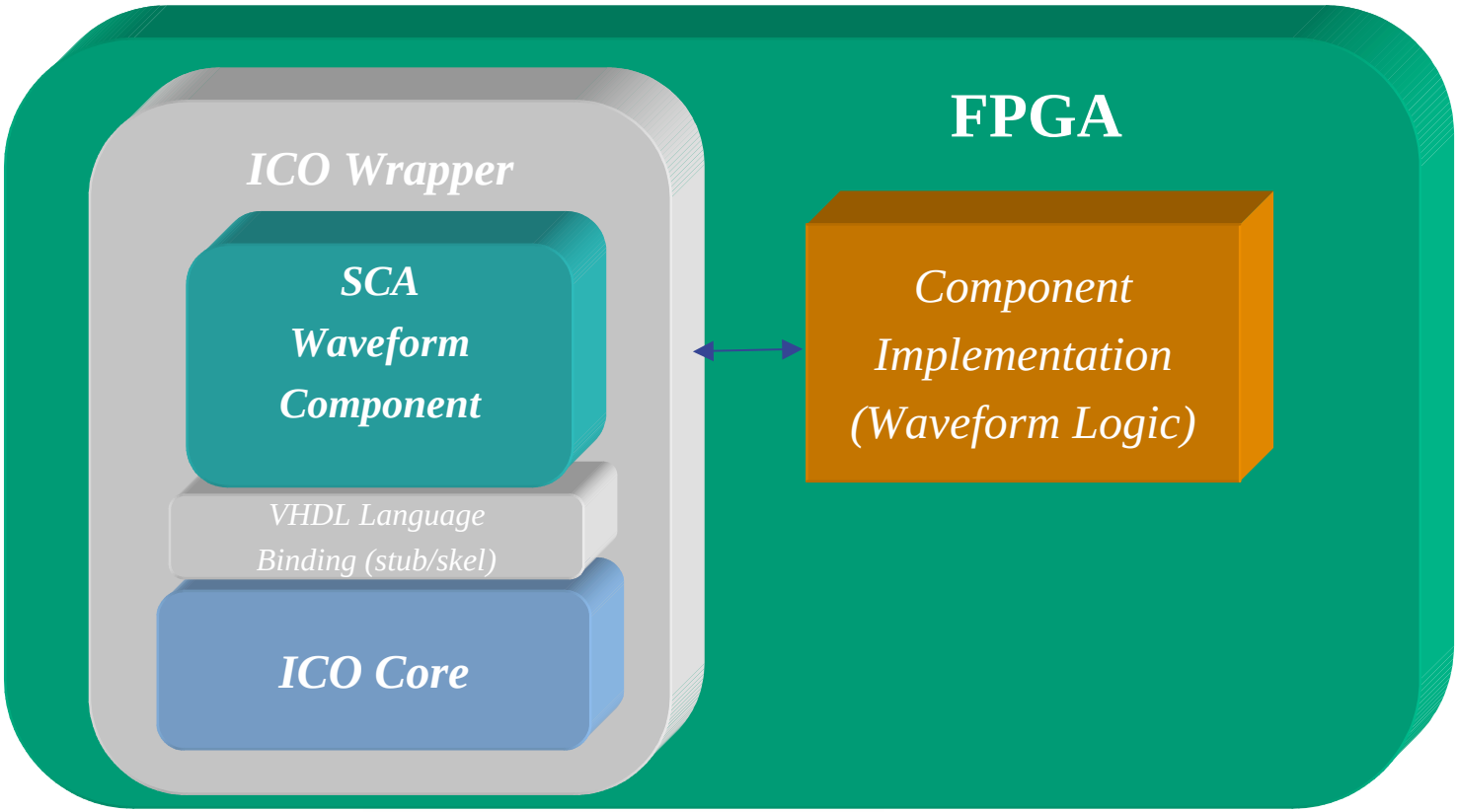
ICO Design Flow

- Connect ICO to the generated SCA waveform component



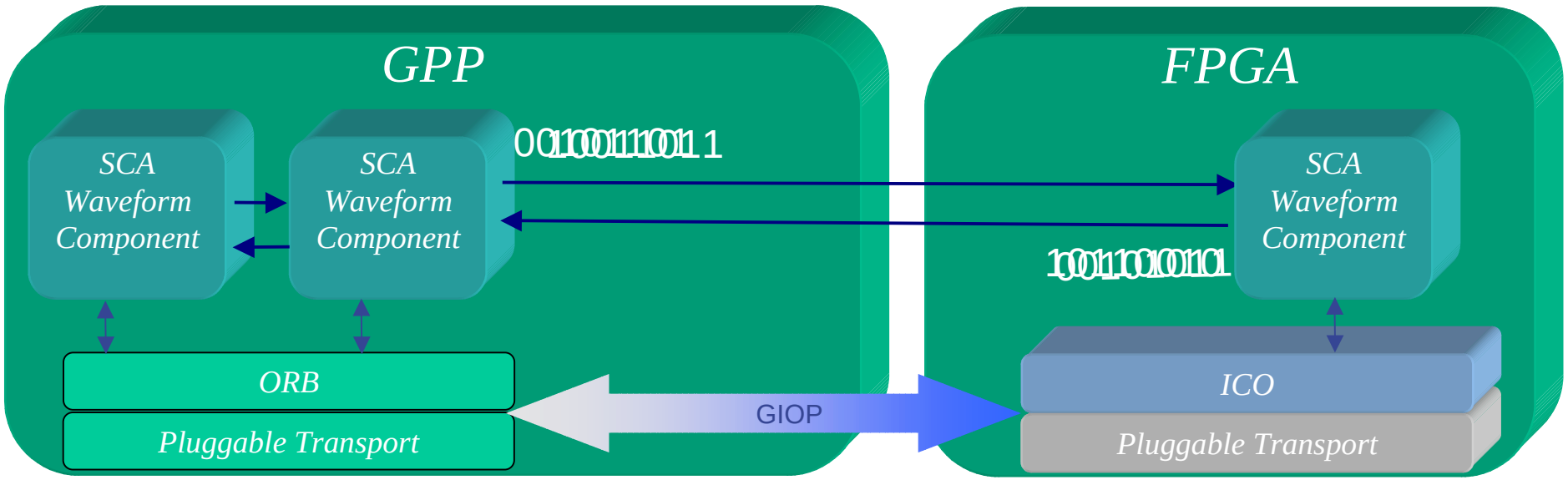
ICO Design Flow

- The embedded waveform logic is then connected to the ICO Wrapper which contains the VHDL SCA waveform component



Using ICO…

- Providing seamless SCA compliance to the FPGA with all location transparency benefits of CORBA



Why use ICO?

- CORBA message processing is executed directly in H/W
  - 100x faster than in S/W
- Eliminates the need for S/W proxies/adapters on GPPs
  - Reduces overhead, latency & Increases throughput
- Eliminates the need for complex hardware abstraction layer protocols improving application portability
- Supports direct access to SCA components running on H/W
  - Modeling tools can auto-generate VHDL delivering SCA “components” in H/W
- Applications in security-related areas where the assurance of large software applications (such as ORBs) is suspect
- Supports vision of SCA architectural consistency across all aspects of the SDR

No More Misconceptions

- ICO removes barriers to FPGA use in SDR
- ICO and other enablers remove the complexity of the SCA from the radio developer and place it where it belongs – in the middleware
- The SDR and embedded development communities now have tools and technology available for the SCA that are similar to those which allowed C and VHDL to become widespread solutions

PRISMTECH
Productivity Tools & Middleware

Question & Answers

Contact Information?

For additional information please visit:

- **www.prismtech.com**

or contact Fred Humcke

- **fh@prismtech.com**