PRISMTECH
Productivity Tools & Middleware

CORBA for DSP & FPGA – synthesizing an “SCA machine”

Andrew Foster
Middleware Product Manager
PrismTech Corporation

Agenda

1. Key challenges in the SDR domain
2. Optimised middleware for SDR
3. Standardizing the signal processing chain
   - Middleware support for DSP & FPGA
4. Addressing SDR accidental complexities
5. PrismTech approach to waveform portability
6. Conclusions

The Challenges

- **Build a better JTRS radio than previous legacy set**
  - Change radio engineers focus from a first generation software integration problem into one that is focussed on optimising the radio and achieving the original SCA goals
- **SDR infrastructure and in particular the middleware will have to be improved for the following reasons:**
  - Enable the radio to be much more highly optimised - smaller latencies, lower MIPS overhead, reduced power consumption, all of these enabling longer running, less power hungry smaller form factor sets
  - Concerns about heavy weight middleware and SCA infrastructure are inhibiting the take up of an Open architecture SDR standard in the commercial space
  - More highly optimised middleware will facilitate the adoption of CORBA running across the radios signal process chain, including DSP and FPGA architectures
- **Improve waveform portability – current waveforms are not very portable.** Key to improving waveforms portability :
  - Better tools for building waveforms
  - Greater standardization of the software interfaces that integrate the signal processing chain

Optimized Middleware for SDR

1. Optimization considerations for soft radio designer
2. Anatomy of the GIOP protocol
3. High performance IDL
4. CDR streams and IDL

Optimisation considerations for soft-radio designer

- Use of CORBA Anys
- Use of Anys vs Unions
- Use of CORBA structs – typecode concerns
- Efficient layout of CORBA structs
- Naming methods in CORBA interfaces (GIOP)
- Naming objects in the server (OIDs)
- Use of Custom Transports

Anatomy of GIOP

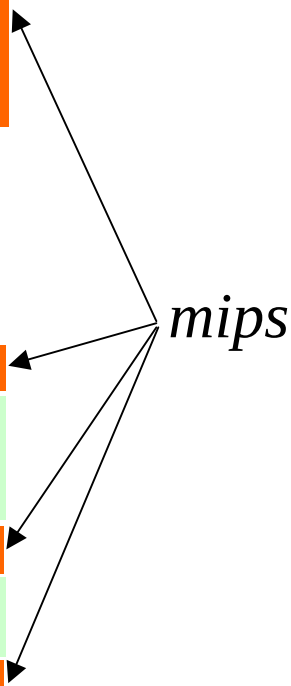
- CORBA IDL interface

```
interface I
{
    void f( );
};
```

Example use of CORBA & GIOP

CORBA Request – wire traffic for that simple call void f(void)

```
0x47, 0x49, 0x4F, 0x50,                 // 'G' 'I' 'O' 'P'
0x01, 0x00,                             // GIOP version '1' '0'
0x01,                                   // flags
0x00,                                   // msg type  (one of 8 )
0x28, 0x00, 0x00, 0x00,                 // msg size

/* RequestHeader */
0x00, 0x00, 0x00, 0x00,                 // svc context
0x00, 0x00, 0x00, 0x00,                 // request id
0x01,                                   // response_expected
0x00, 0x00, 0x00,                       // padding
0x09, 0x00, 0x00, 0x00,                 // object key size
0x00, 0x00, 0x00, 0x00, 0x00,           // object key body
        0x00, 0x00, 0x00, 0x00,
0x04, 0x31, 0x00,                       // padding
0x00, 0x00, 0x00, 0x02,                 // operation name size
0x66, 0x00,                             // operation name ('f' '\0')
0x00, 0x00,                             // padding
0x00, 0x00, 0x00, 0x00                  // principal
```

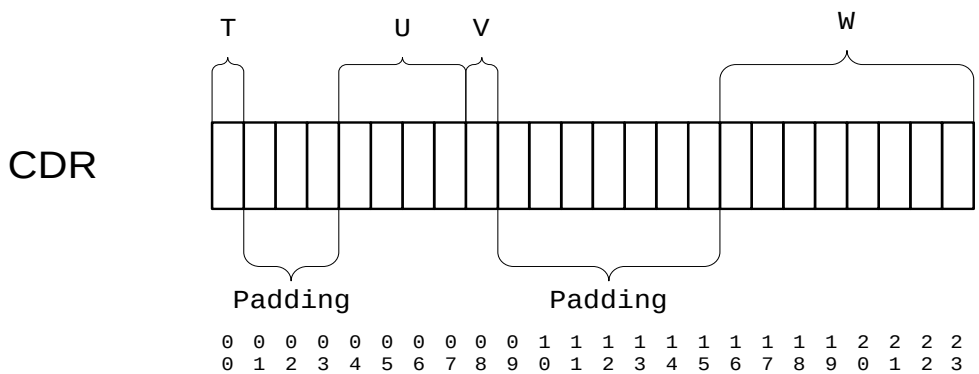


► High Performance IDL - Super optimized marshalling – intelligently

e.g. IDL structure that yields inefficient CDR layout.

- Padding required to align data per CDR rules

```
struct S {
  boolean  T;
  long     U;
  char     V;
  double   W;
};

-- same CDR layout as --

void Func(in boolean T, in long U, in char V, in double W);
```

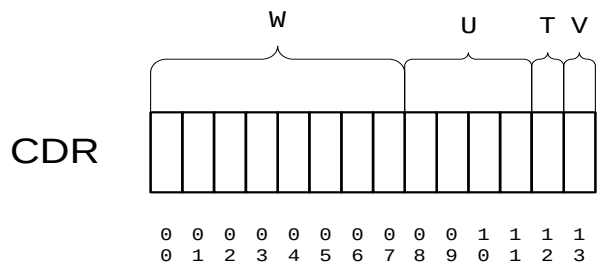


▶ CDR streams

IDL structure organization that yields optimal CDR layout.

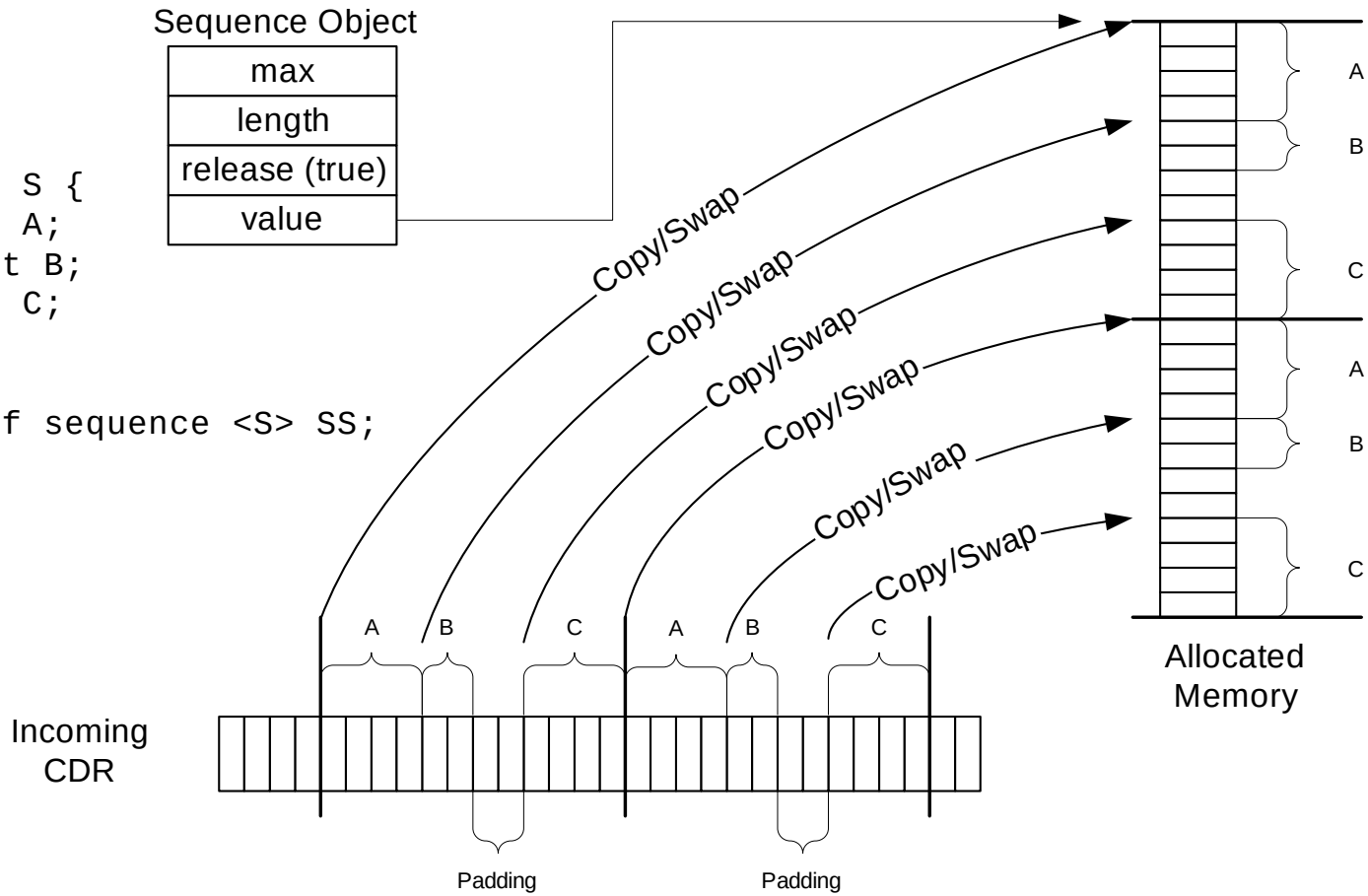
- **Structure elements rearranged to eliminate need for padding.**

```
struct S {
  double   W;
  long     U;
  boolean  T;
  char     V;
};

-- same CDR layout as --

void Func(in double W, in long U, in boolean T, in char V);
```



- Non optimized allocation and copying of a sequence of structs
  - Memory allocation and copying required normally – inefficient use of ORB

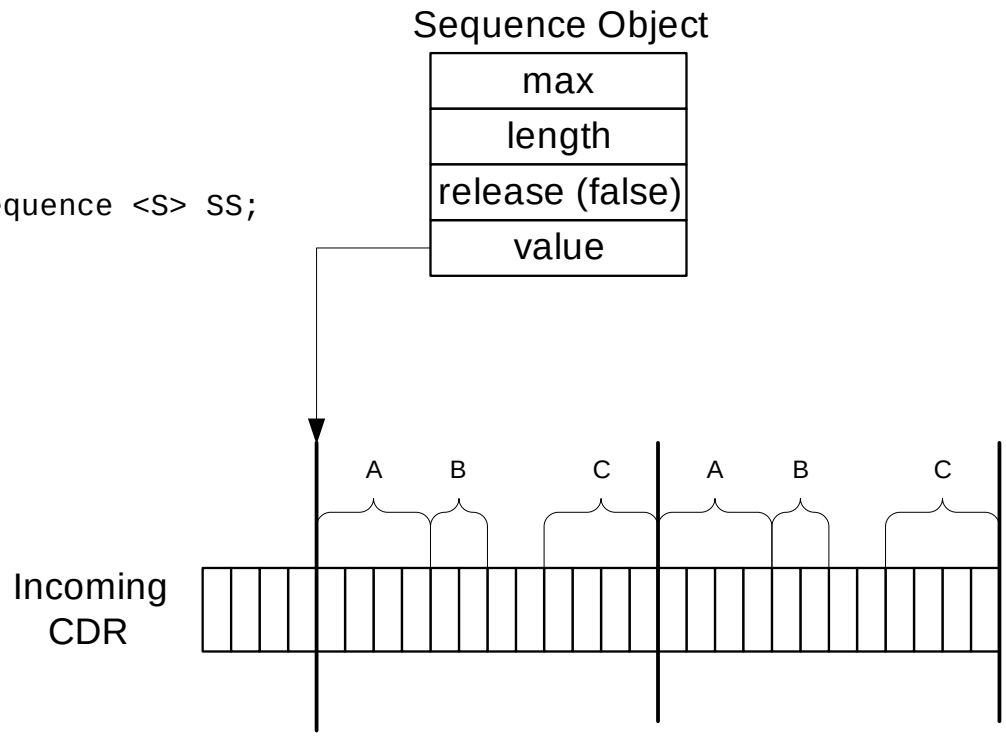
```
struct S {
  long A;
  short B;
  long C;
};

typedef sequence <S> SS;
```



► Optimized reference to the CDR buffer of a sequence of structs

- To a avoid a costly allocation and copy, the rules for alignment, padding and endianness of the compiler and processor must all match the corresponding attributes for the OMG standard CDR rules.
- Fortunately, these rules often do match and the optimization can be applied.

```
struct S {
  long A;
  short B;
  long C;
};

typedef sequence <S> SS;
```



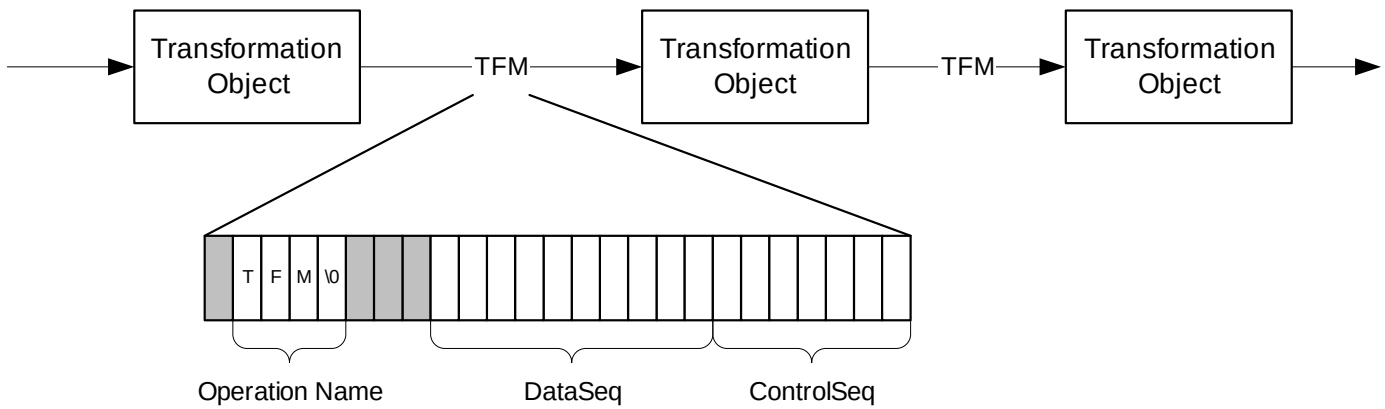
CORBA in the real-time payload chain

- **Operation name 'TFM' is optimal for the wire – three characters plus the null termination character fit nicely into a single DWord of the CDR stream.**

```
IDL: void oneway TFM(in DataSeq Data, in ControlSeq Control);
```

```
Implementation Pseudo-code:

void TFM(DataSeq Data, ControlSeq Control)
{
  // Perform transformation on Data either in-place or new copy

  TransformedData = Transform(Data);

  pNext -> TFM(TransformedData, Control)
}
```

signal processing chain….

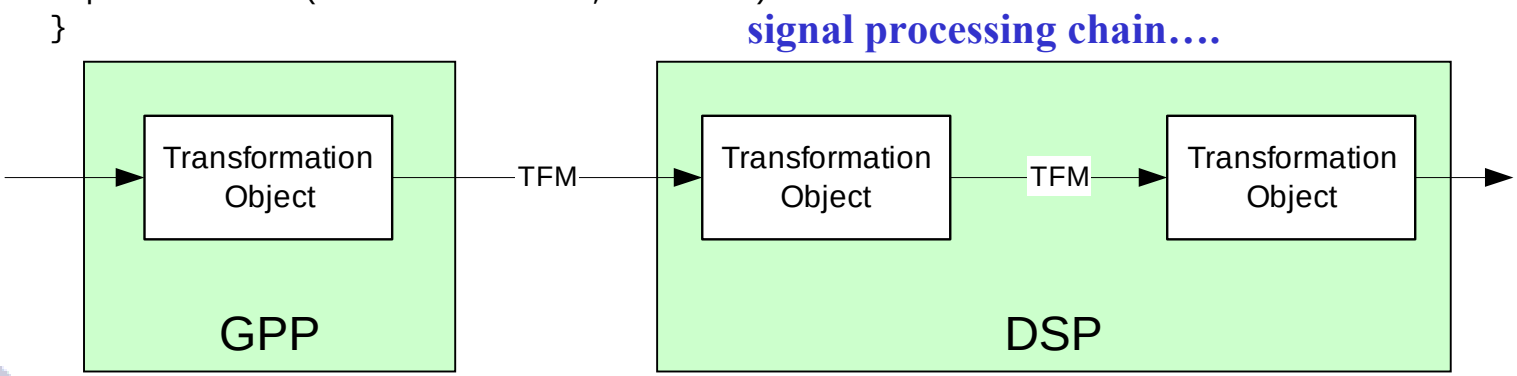


► CORBA in the real-time payload chain

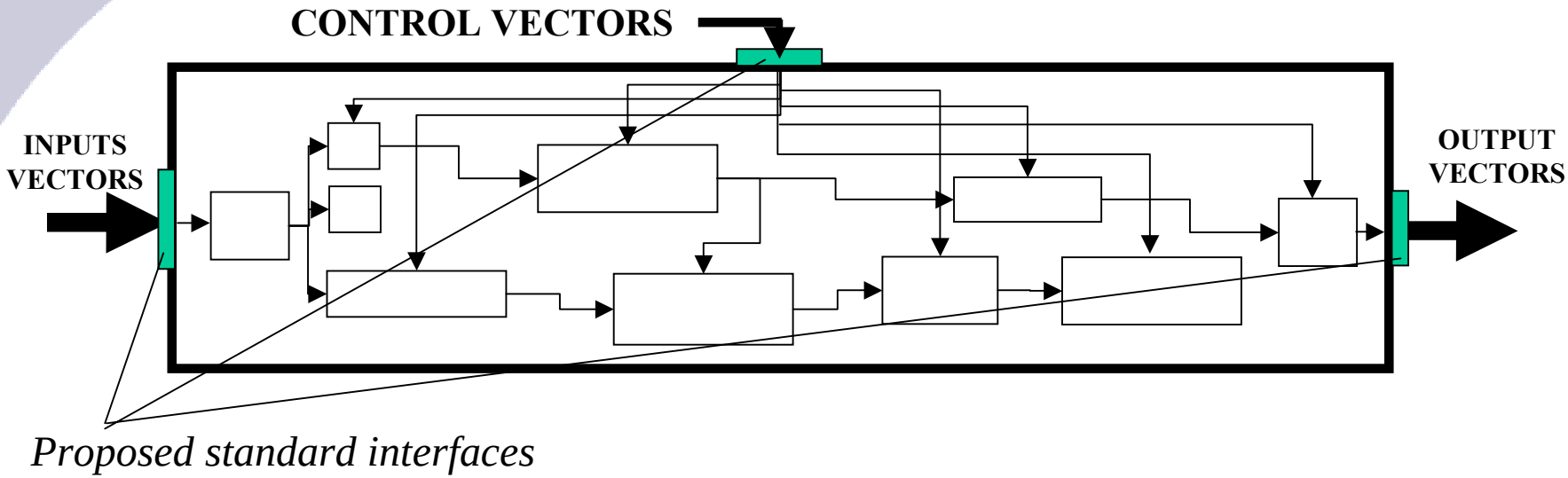
- **Operation name 'TFM' is optimal for the wire – three characters plus the null termination character fit nicely into a single DWord of the CDR stream.**

```
IDL: void oneway TFM(in DataSeq Data, in ControlSeq Control);
```

```
Implementation Pseudo-code:

void TFM(DataSeq Data, ControlSeq Control)
{
  // Perform transformation on Data either in-place or new copy

  TransformedData = Transform(Data);

  pNext -> TFM(TransformedData, Control)
}
```



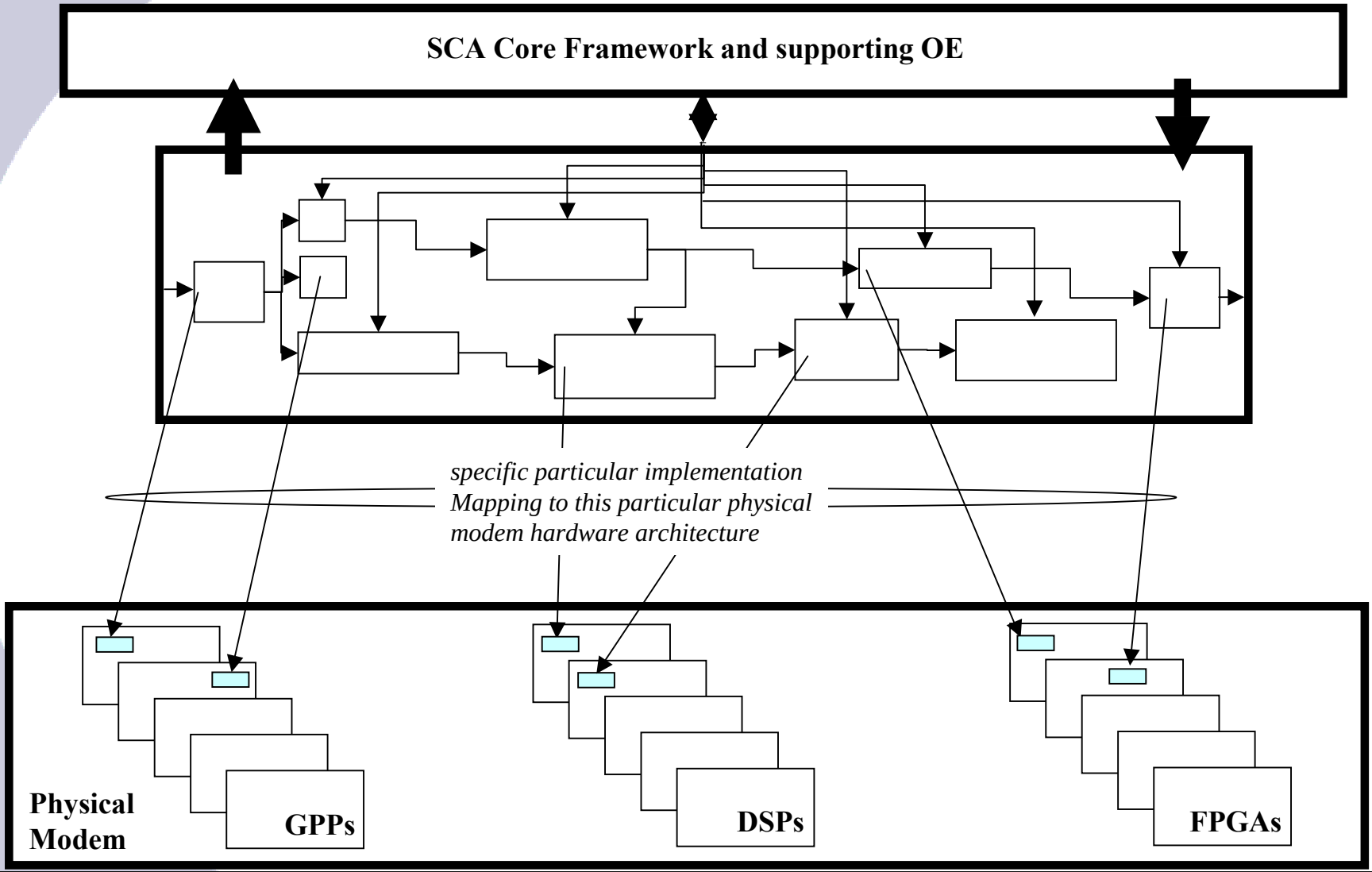
An SCA 3.0 Waveform Model



Any waveform can be modelled as SCA suggests described in IDL – proposed lightweight CORBA interfaces –

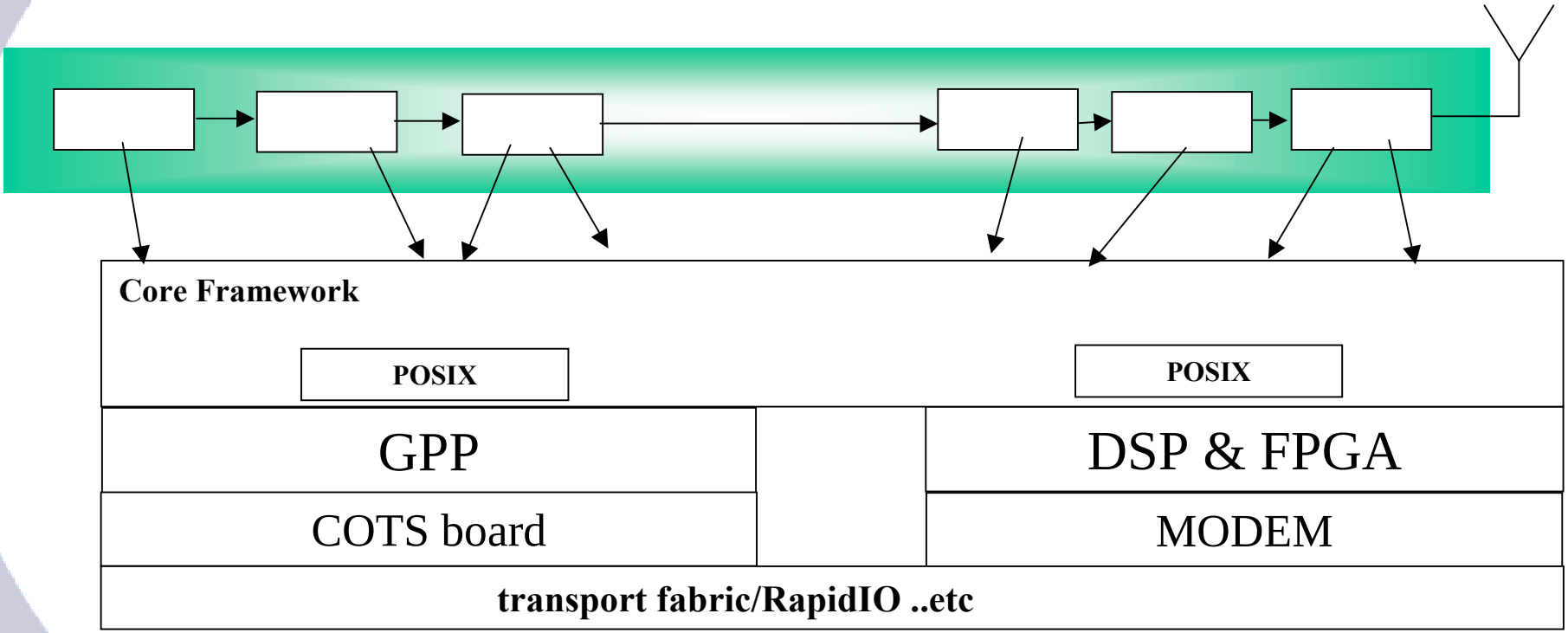
1. **Mod/De ulators**
2. **En/De coders**
3. **Mux/Demux**
4. **Up/down sampler/ Decimators etc**
5. **Note control inputs to these blocks are also shown**
6. **Note – No hardware mapping is shown – only canonical waveform described in terms of its signal processing chain and mathematical operations**

Physical Hardware Mapping



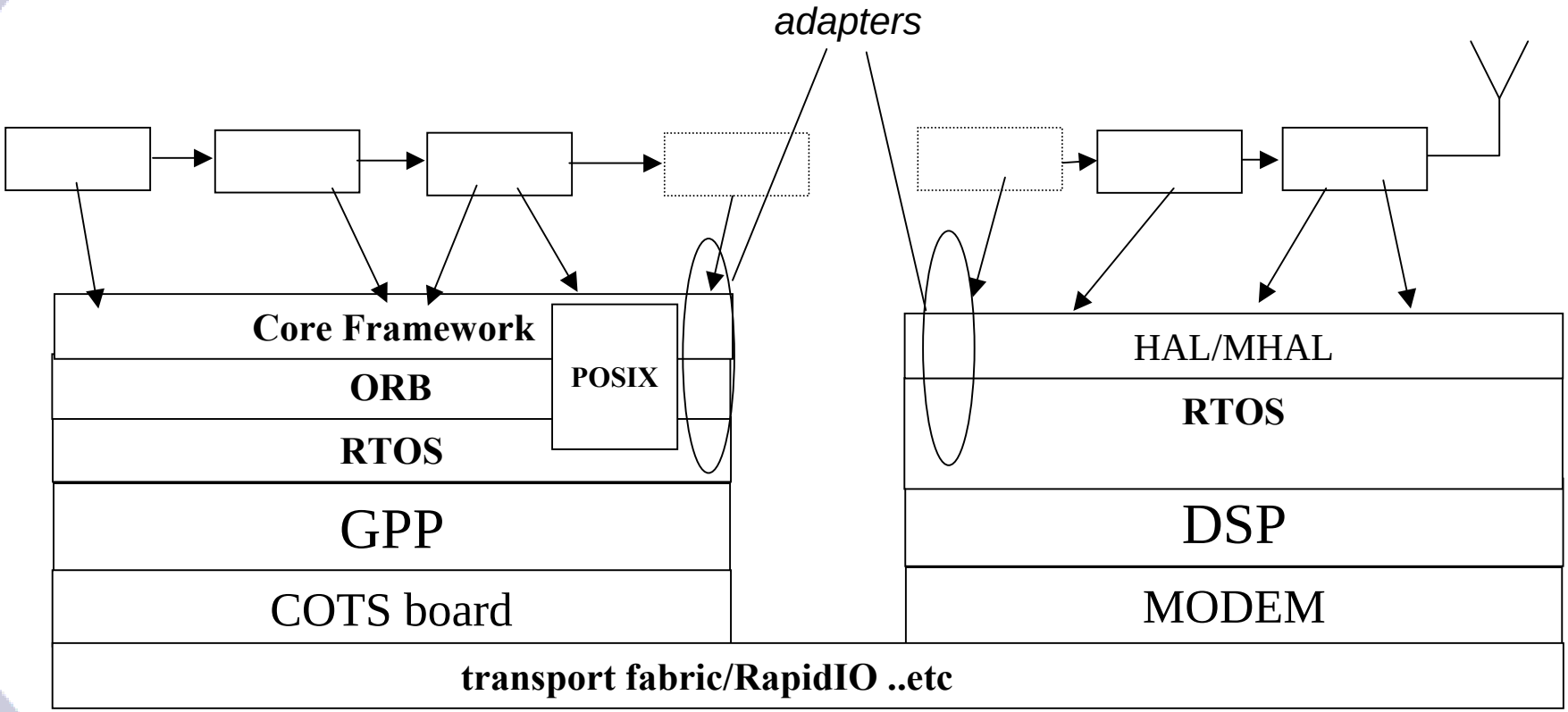
Idealized Device Model

Pure canonical signal processing chain of SDR device

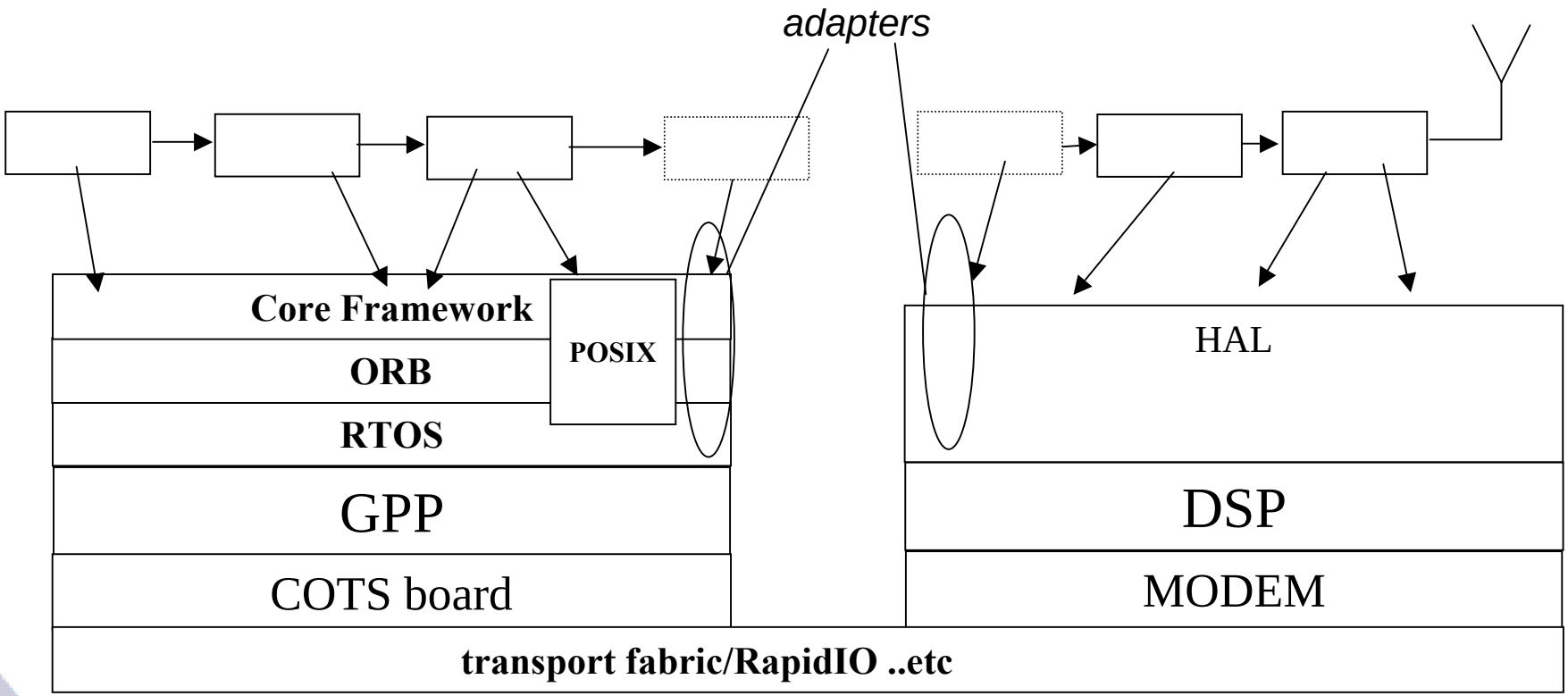


DSP Device Models

Signal processing chain of SDR device

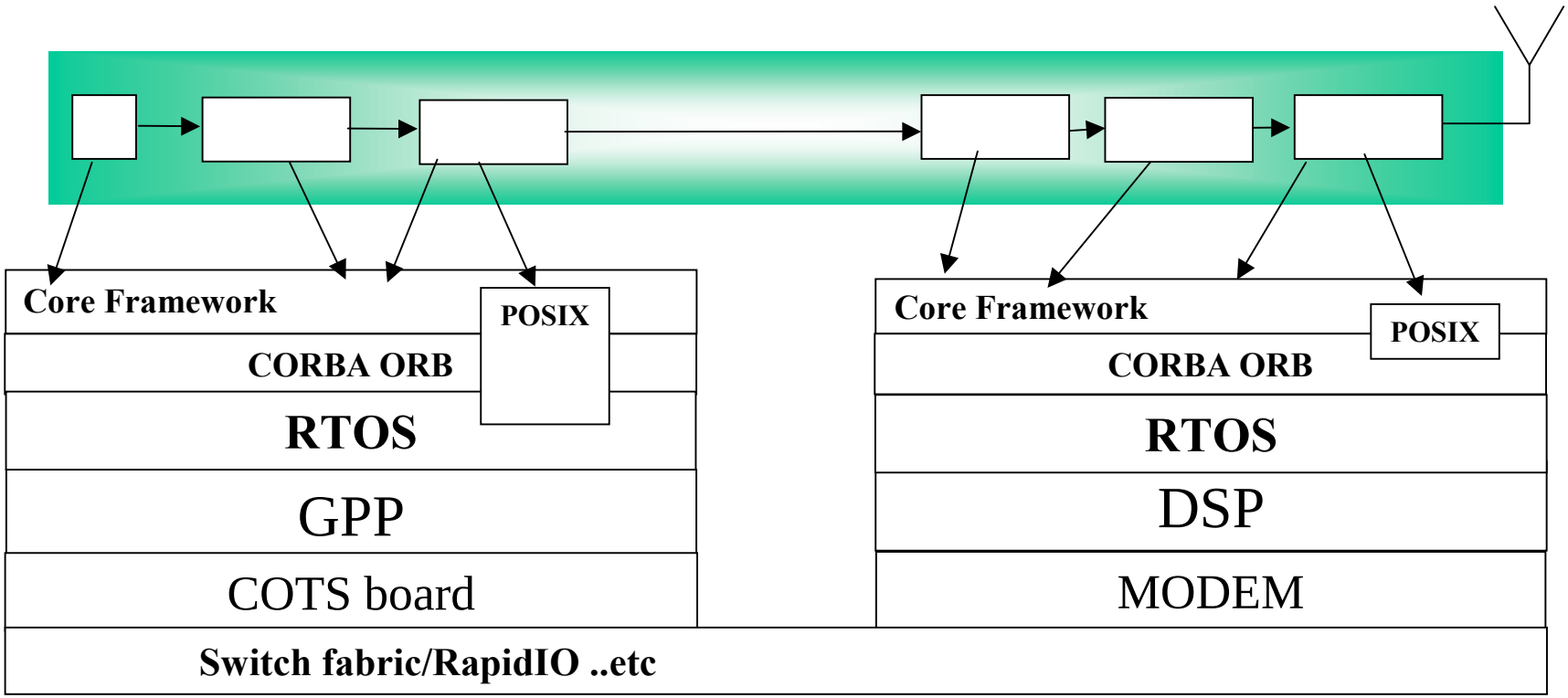


Signal processing chain of SDR device



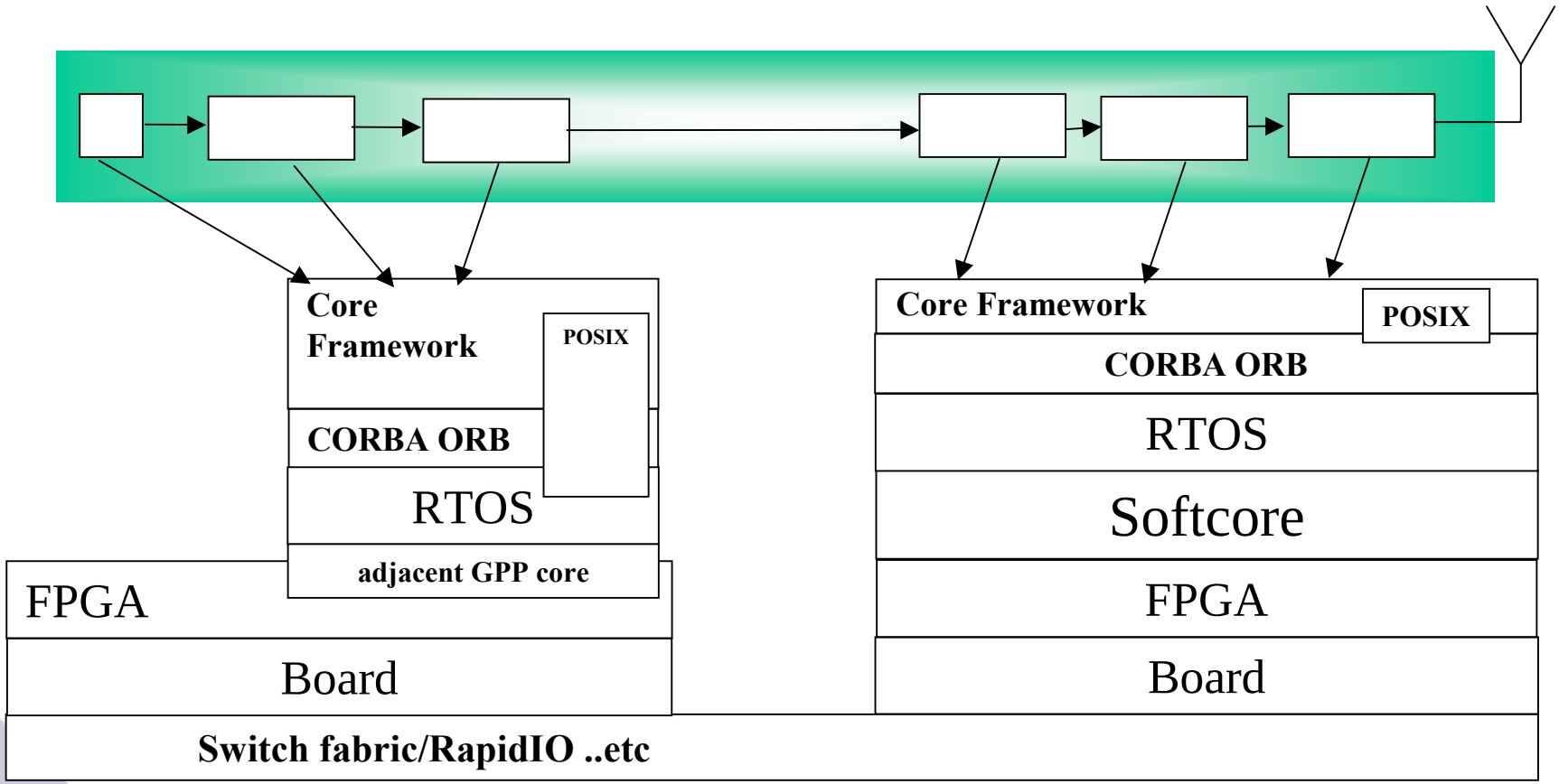
FPGA Device Models

Pure canonical signal processing chain of SDR device

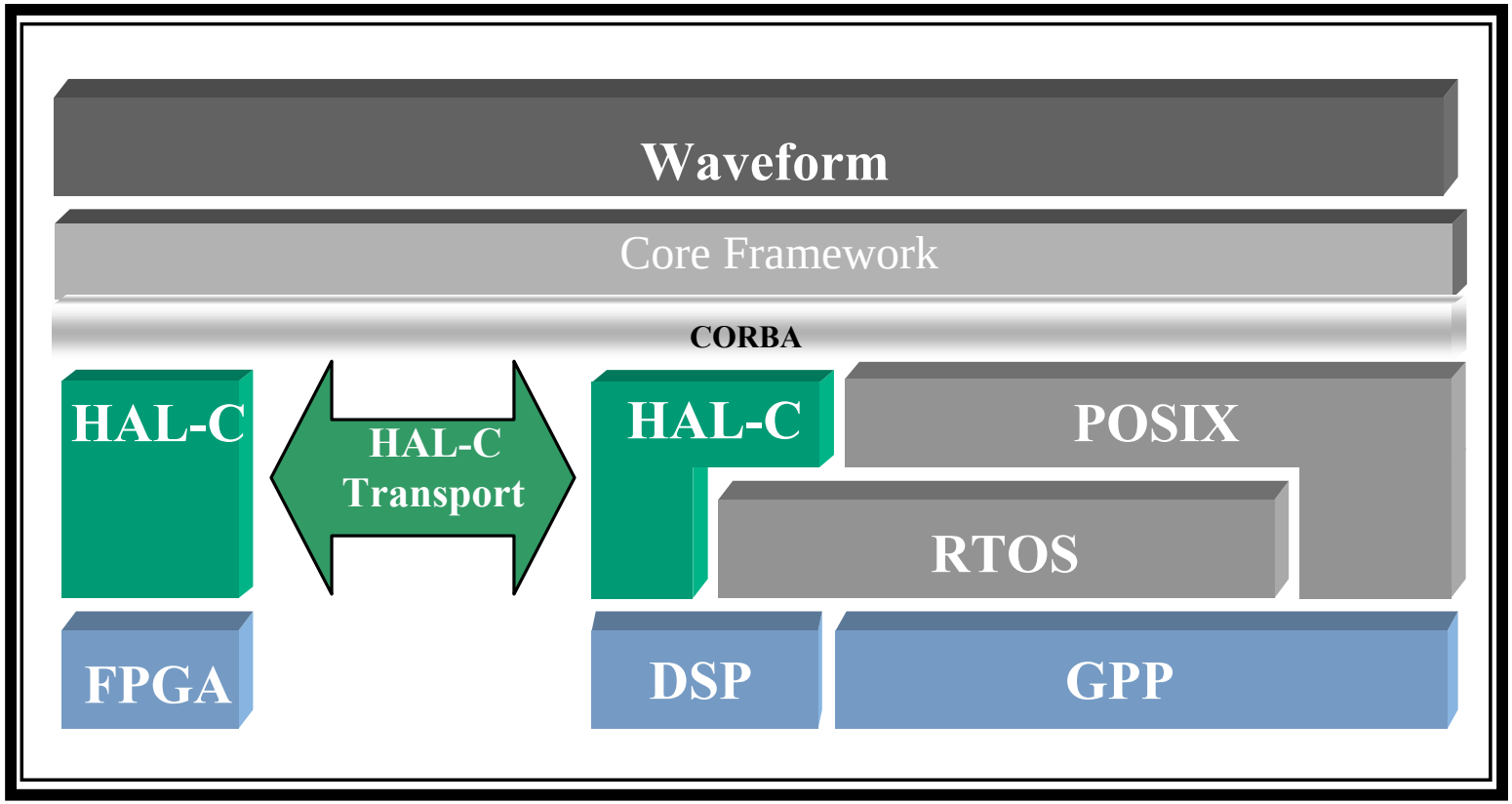


FPGA Device Models

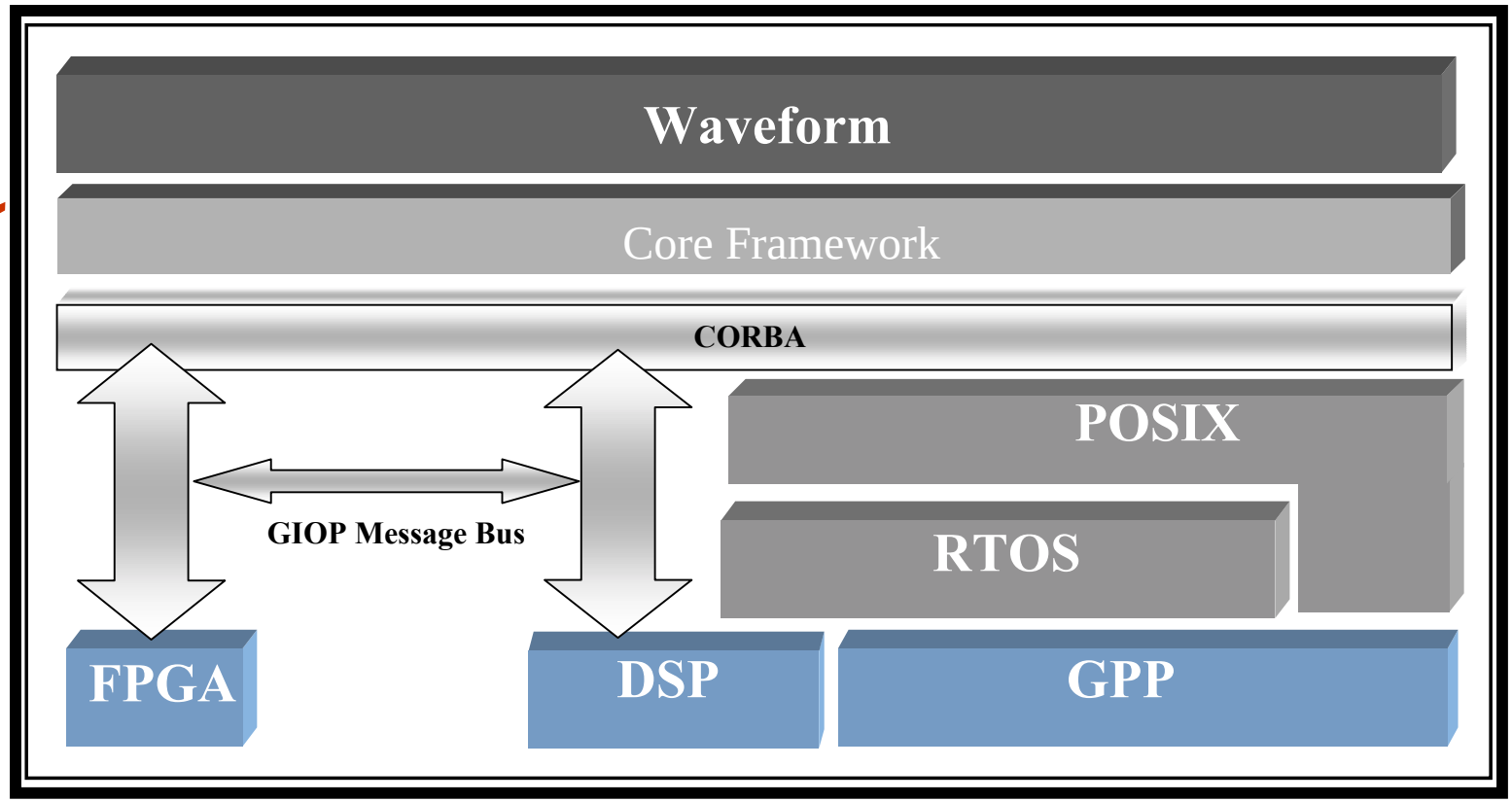
Pure canonical signal processing chain of SDR device



SCA 3.0 Device Model



PrismTech Proposed Device Model – SCA Machine



Standardized CORBA interfaces enabling SCA machine

Why Do We Think It's Possible ?

- Use of a low MIPS consuming CORBA implementation makes possible use of CORBA in places that were 'forbidden' to ORBs before
- The SCA machine idea is in keeping with the OMGs model of a canonical soft-radio device using the notion of PIM and PSMs

Future Benefits

- Waveform portability – radically improved
- Waveform independence from underlying physical radio topology
- Easier to achieve SCA compliance overall
- Some degree of independence from waveform language of implementation
- Greater flexibility to optimize and relocate tasks between the different processor cores underlying.
- Greater capability to more flexibly use the shared resource model proposed by Mercury – re: Mike Kosmicki at the SCA layers.
- Increased developer productivity as strict enforcement of the use of SCA interfaces only everywhere in the radio
- Will facilitate easier integration with next generation modelling tools – providing vastly levels of automatic code generation
- Will allow middleware vendors such as PrismTech to supply much of this technology as COTS –
- No need for developers to write custom non portable codecs in order to support underlying HALs – GIOP used everywhere !

SDR Accidental Complexities

- Frederick Brooks identified in his article "**No Silver Bullet**"
  - Systems contain ***inherent*** and ***accidental*** complexities
- **Inherent Complexities**
  - Those found within the problem domain itself
  - eg. Modulation, Demodulation, FEC, QoS, …
- **Accidental Complexities**
  - Those accompanying the production of software but can be eliminated
  - SDR Framework Complexity
    - Large number of interfaces
    - Difficult to understand how they work together
    - Complex implementation technologies (C++, XML, CORBA)

SDR Accidental Complexities

- **Accidental Complexities** (continued)
  - Tight coupling to particular vendors/technologies/hardware
  - Rediscovery and reinvention of SDR component design techniques and pattern languages
  - Lack of support for re-usability
  - Excessive low level details
    - E.g.– what exceptions to throw under what conditions
  - Tight coupling between operating elements
    - ie. portability of the implementation
  - Lack of tools

Eliminating Accidental Complexities

- Elimination of the accidental complexities allows developers to focus on the inherent complexities – **i.e. on the radio and waveform and not on CORBA and C++**

- We need to address these accidental complexities by leveraging:
  - ***Techniques***
  - ***Tools***
  - ***Turnkey solutions***
  - ***Significantly greater degrees of training***

Techniques

- **Separation of Responsibilities and Concerns**
  - SDR Frameworks purposely separate interfaces
    - Allowing waveform development to proceed in parallel to radio platform development
- **Domain Analysis**
  - Identify commonalities and variabilities of a domain
  - Provide re-usable solutions for common aspects
  - Provide hooks to allow seamless integration of variabilities
- **Support development in the Problem Space vs. the Solution Space**
  - Domain Specific Modeling
  - Domain Specific Programming

Benefits of Separation

- **SDR users can identify the correct aspect to support their role**
  - Waveform Developers
  - Radio Platform Developers
  - Radio Service Developers
- **Testing and Integration is eased**
  - As coupling is significantly decreased
- **When separation is not clearly established**
  - Users arbitrarily couple the aspects and confuse their roles
  - They over-emphasize of one aspect over another
  - And here's an example…

Over-emphasis On Radio Platform Development

- **Many organizations overly concerned themselves with radio platform development (ie. Core Frameworks)**
  - WF developers arbitrarily create a dependency between having a framework and getting started with development
- **WF development can be more complex than radio platform development**
  - As the waveform themselves are very complex
  - And development needs to start early…
- **Even when organizations start WF development early they provide little to no support for WF developers**
  - No waveform development tools
  - No waveform testing support
  - The integration of waveform and platform that follows may require a some amount of code modification.

PrismTech Approach To Waveform Portability

- **Increase overall SDR development productivity via**
  - Graphical development tools
  - Automatic code generation
  - High performance middleware
  - Intuitive SCA Core Framework Technologies (SCA on DSP and FPGA direct)
  - Increased developer training and mentoring to promote awareness of the issues in SDR – just like the early days of OO and C++ - we need more awareness campaigns
- **Protect past customer investments**
  - Support/maintenance for existing SW infrastructures and waveform technology.
  - Smooth migration path to future products and standards

Example - Spectra Modeling Tools

- **Graphical Domain Specific Modeling Tool**
- Used to Capture Component and Assembly High Level Design
- Provide Constraint Checking
  - Correctness checks performed at modeling time vs. runtime
- **Automatically generate:**
  - The source code needed by waveform developers to interface to the radio platform
    - Allow waveform developers to focus on waveform logic
    - Not on skeletal infrastructure code
  - The descriptors needed by the radio platform infrastructure
  - The unit tests needed to test waveform component functionality and compliance to the platform
  - The testing infrastructure to test the waveform stand alone (without need for the radio platform)
  - The high level documentation needed support design documentation

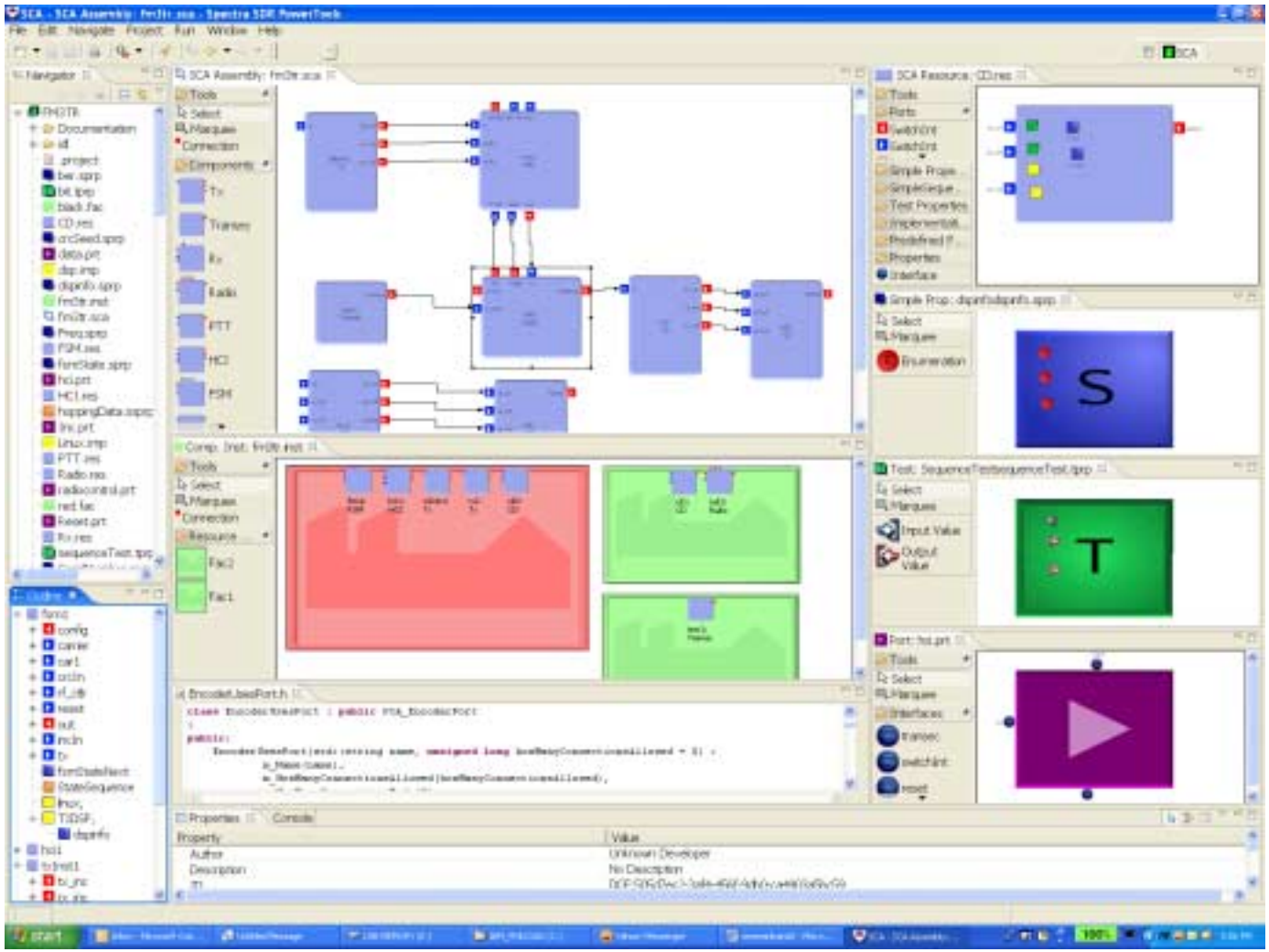
PrismTechs Approach To Helping Organizations Achieve Waveform Portability

- Provide an effective set of waveform modeling tools
  - Integration with industry standard tools – e.g. Matlab, Simulink, ModelSim etc.
  - Standards support for different SDR technology bases – e.g for SCA
    - Allows CF::Resource Declaration
    - Allows Assembly Declaration and Component Connections
- Automatic Generation of
  - CF::Resource Code in C++
  - CF::Resource Unit Tests
  - Surrogate ApplicationFactory capable of deploying Assembly
    - Lightweight, Robust, and SCA-compliant
    - Supports Waveform Testing

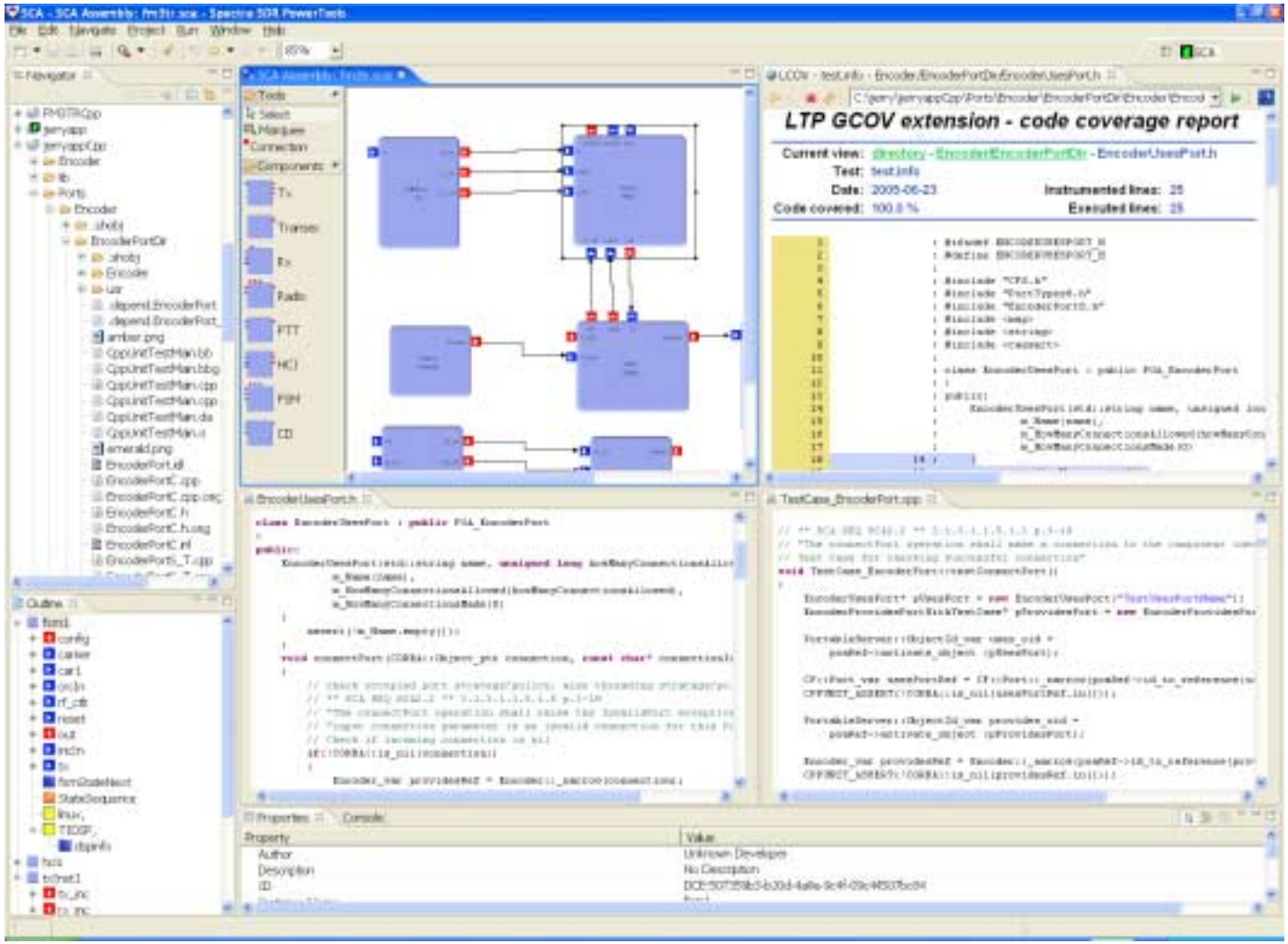
Targeting Multiple H/W Environments

- General Purpose Processors
- Digital Signal Processors
- RTL (Resistor Transistor Logic)
  - Field Programmable Gate Arrays and ASICs
- Running GIOP protocol between all hardware
  - ORBs on GPPs/DSPs and GIOP CODEC on FPGA
- Within RTL devices, following direction laid out by upcoming SCA 3.1
  - Provide tools for software developers on GPP, DSP FPGA and ASICs

Assembly Modeling and Exploring Deployment Over The Radio's Topology



Automatically Generating The Component Code and Measuring Test Code Coverage



Conclusions

- **CPU optimized CORBA middleware to support components and effectively push data through the defined interfaces of the soft-radio model**
  - Characterized by minimum CPU cycles per KB of data marshalled
  - Will enable support for standardized DSP and FPGA interfaces – increasing waveform portability
- **Waveform portability requires a multi-level component architecture regardless of technology mapping used – e.g. MDA based UML to IDL and SCA or some other model**
- **Tool support is crucial to support an effective and efficient component architecture with a standards based approach**
- **Waveform synthesis requires care**
  - keep it in component form – models
  - keep it decoupled from the complexities of the underlying platforms
  - More emphasis on tools to generate slim, MIPS sensitive code