



Deliverable – D2

Asynchronous IP Packaging: Specifications

Grant Agreement No:	214364
Project acronym:	GALAXY
Project title:	GALS InterfAce for CompleX Digital System Integration
Funding Scheme:	STREP
Date of latest version of Annex I against which the assessment will be made:	23.10.2007.
Contractual Date of Delivery to the EC:	30. Apr. 08
Actual Date of Delivery to the EC:	
Author(s):	Lilian Janin (UNIMAN)
Participant(s):	UNIMAN
Work Package:	WP2
Security:	Public
Nature:	Report
Version:	2
Total number of pages:	25

Abstract:

This document describes the specifications of the ASIP (Asynchronous/Synchronous IP) format, a file format for packaging IPs with mixed synchronous-asynchronous interfaces.

The ASIP format is able to describe hardware/software entities at multiple levels of abstraction in multiple languages (SystemC, C, Verilog, VHDL, Balsa, gate-level netlist, etc.) with both synchronous and asynchronous interfaces. It contains enough information to transparently convert signals between varying levels of abstraction, enabling a transparent co-simulation of IPs at different levels of abstraction, the visualisation of signals at levels of abstraction independently of the simulated level of abstraction, and the exploitation of transaction level during hardware-software co-design and in the visualisation. ASIP also aims at enabling various IPs to be interconnected seamlessly in a point-to-point manner or via a network-on-chip and providing a way to package synchronous IPs in an asynchronous container for inclusion in GALS systems.

Keyword list: GALS, asynchronous, IP, packaging



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

Function	Responsibility	Date	Signature
Written by:	Lilian Janin	17 Apr 2008	
Checked by:	Members of GALAXY Consortium		
Approved by:	-		

Reserved to EC

Approved by:			
---------------------	--	--	--





GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

CHANGE RECORDS

<i>ISSUE</i>	<i>DATE</i>	<i>§ : CHANGE RECORD</i>	<i>AUTHOR</i>
1	17-Apr-08	1 st version	Lilian Janin
2	24-Apr-08	2 nd version: applied corrections suggested by partners	Lilian Janin



BIBLIOGRAPHIC RECORD

Project Number:	214364 GALAXY
Project Title:	GALAXY
Deliverable Type:	Report
Deliverable Number:	D2
Contractual Date of Delivery:	31. Apr. 2008
Actual Date of Delivery:	
Title of Deliverable:	Asynchronous IP Packaging: Specifications
Work package contributing to the Deliverable:	WP2
Authors:	Lilian Janin
Abstract	<p>This document describes the specifications of the ASIP (Asynchronous/Synchronous IP) format, a file format for packaging IPs with mixed synchronous-asynchronous interfaces.</p> <p>The ASIP format is able to describe hardware/software entities at multiple levels of abstraction in multiple languages (SystemC, C, Verilog, VHDL, Balsa, gate-level netlist, etc.) with both synchronous and asynchronous interfaces. It contains enough information to transparently convert signals between varying levels of abstraction, enabling a transparent co-simulation of IPs at different levels of abstraction, the visualisation of signals at levels of abstraction independently of the simulated level of abstraction, and the exploitation of transaction level during hardware-software co-design and in the visualisation. ASIP also aims at enabling various IPs to be interconnected seamlessly in a point-to-point manner or via a network-on-chip and providing a way to package synchronous IPs in an asynchronous container for inclusion in GALS systems.</p>
Keywords	GALS, asynchronous, IP, packaging
Confidentiality Level	Public
Name of Client:	EC
Distribution List:	GALAXY, EC, internet
Authorised by:	
Issue:	2
Document ID:	D2
Total Number of Pages:	25
Contact Details:	Lilian.janin@manchester.ac.uk



TABLE OF CONTENTS

1	INTRODUCTION.....	6
2	REFERENCES.....	7
2.1	ACRONYMS	7
3	REQUIREMENTS	8
4	THE ASIP FORMAT	10
4.1	XML SCHEMA	10
4.2	DIFFERENCES WITH SPIRIT CONSORTIUM'S IP-XACT FORMAT	11
4.3	FUTURE EXTENSIONS	11
5	ASIP FORMAT STRUCTURE.....	14
5.1	TOP-LEVEL DESCRIPTION	14
5.2	COMPONENTS.....	14
5.3	INTERFACE(S).....	16
5.3.1	Hardware interface	16
5.3.2	Software interface	16
5.4	IP IMPLEMENTATION(S)	17
5.4.1	Sub-circuit	17
5.4.2	Single element.....	18
5.5	TRANSACTORS BETWEEN HARDWARE AND SOFTWARE INTERFACES	18
5.6	ASYNCHRONOUS PROTOCOLS	19
5.7	GLOBAL TYPES	19
APPENDIX	20	
A	APPENDIX A - ASIP XML SCHEMA.....	20

LIST OF FIGURES

Figure 1:	Block diagram of an ASIP component	15
-----------	--	----



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

1 INTRODUCTION

This document describes the specifications of the ASIP (Asynchronous/Synchronous IP) format, a file format for packaging IPs with mixed synchronous-asynchronous interfaces.

The ASIP format is able to describe hardware/software entities at multiple levels of abstraction in multiple languages (SystemC, C, Verilog, VHDL, Balsa, gate-level netlist, etc.) with both synchronous and asynchronous interfaces. It contains enough information to transparently convert signals between varying levels of abstraction, enabling a transparent co-simulation of IPs at different levels of abstraction, the visualisation of signals at levels of abstraction independently of the simulated level of abstraction, and the exploitation of transaction level during hardware-software co-design and in the visualisation. ASIP also aims at enabling various IPs to be interconnected seamlessly in a point-to-point manner or via a network-on-chip and providing a way to package synchronous IPs in an asynchronous container for inclusion in GALS systems.

Further in this project, ASIP is intended to serve as a foundation for the exchange of asynchronous IPs on the internet, following a model similar to opencores.org. We will try to avoid the problems usually associated with the opencores model: outdated documentation, absence of quality rating, absence of coding standards, portability issues between hardware targets, missing simulation scripts



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

2 REFERENCES

2.1 ACRONYMS

AMBA	Advanced Microcontroller Bus Architecture
ASIP	Asynchronous-Synchronous IPs packaging format
CGP	CHAIN Gateway Protocol
DE	Design Environment
ESL	Electronic System Level
GALS	Globally Asynchronous Locally Synchronous
IP	Intellectual Property
IP-XACT	SPIRIT consortium's IP packaging format
NoC	Network on Chip
OCP	Open Core Protocol
SoC	System on Chip
SPIRIT	http://www.spiritconsortium.org
VHDL	VHSIC Hardware Description Language
XML	eXtensible Markup Language



3 REQUIREMENTS

The design of the ASIP IP packaging format is based on the following requirements.

- **File format**

- AsipReq1.1:* Format working with control version systems, i.e. with the line-by-line ASCII 'diff' tool. Definitely not a binary format
- AsipReq1.2:* Descriptions following the format must be sufficiently clear as to be human-readable
- AsipReq1.3:* Format "compatible" with SPIRIT, to leave the possibility for future integration
- AsipReq1.4:* Versioning: When the file format evolves over time, designs described using different versions of this format need to be identifiable and upgradable.

- **Components**

- AsipReq2.1:* Support for module hierarchy: A component can be a hierarchical object or a leaf object. Leaf components do not contain other components, while hierarchical components contain other sub-components. This can be recursive by having hierarchical components that contain hierarchical components.
- AsipReq2.2:* Versioning: components will evolve. They need to be assigned version numbers used for their identification, in order to prevent old designs from breaking when a referenced component evolves.

- **Interfaces**

- AsipReq3.1:* Support for both synchronous and asynchronous interfaces
- AsipReq3.2:* Support for software PV interface
- AsipReq3.3:* Although actual buses are synchronous, our format must be capable of support for any particular bus (IP-XACT 1.2.2)
- AsipReq3.4:* Ability to instantiate large interfaces such as AMBA/OCP interfaces by using globally defined names or types.

- **Inter-component connectivity: wires, channels, buses and NoCs**

- AsipReq4.1:* Support for point-to-point connections
- AsipReq4.2:* Support for hardware-hardware, hardware-software and software-software connections
- AsipReq4.3:* Support for networks-on-chip

- **Implementations/Views**

- AsipReq5.1:* The IP format will be able to refer to hardware/software entities at multiple levels of abstraction in multiple languages (SystemC, C, Verilog, VHDL, Balsa, gate-level netlist,...)
- AsipReq5.2:* Support for either source code or encrypted netlists
- AsipReq5.3:* Support XML catalogue listing the views available, their type and their source files (IP-XACT 3.1.5)

- **Abstractions**

- AsipReq6.1:* The IP format will contain enough information to transparently convert signals between varying levels of abstraction



- **Tools: Generic**

AsipReq7.1: Separation of logical and graphical information (IP-XACT 1.1.5): The schema must separate logical and graphical information. Minimally, the schema should define only logical information about the design; it should not contain information about how that information is to be presented in a UI. If the schema contains diagrammatic information, it must be kept separate from the logical information. It is critical to separate content from presentation.

AsipReq7.2: Execution platform independence (IP-XACT 1.2.3): The schema must be execution platform independent. The schema should support the ability to re-use the design on multiple execution platforms

- **Tools: Design Environment and Visualisation**

AsipReq8.1: Support for visualisation of signals at levels of abstraction independently of the simulated level of abstraction,

- **Tools: Simulation**

AsipReq9.1: Format should refer to the preferred simulators for each implementation

AsipReq9.2: Support for automatic generation of simulation code for sub-systems described in our format

AsipReq9.3: Support for automatic generation of co-simulation interfaces

AsipReq9.4: Support for transparent co-simulation of IPs at different levels of abstraction

AsipReq9.5: Support for exploitation of transaction level during hardware-software co-design and in the visualisation.

- **IP library support**

AsipReq10.1: It must be possible to create, maintain and distribute libraries of IPs (IP-XACT 3.2.1)

AsipReq10.2: Avoid the problems usually associated with the opencores model: outdated documentation, absence of quality rating, absence of coding standards, portability issues between hardware targets, missing simulation scripts

AsipReq10.3: Versioning: IP libraries should include version numbers in order to prevent old designs from breaking when a library evolves.



4 THE ASIP FORMAT

Based on the above-mentioned requirements we have defined the ASIP packaging format.

ASIP is based on XML (*AsipReq 1.1, 1.2 and 1.3*). As a convention, files described using the ASIP format should use the extension `.asip.xml`, in order to keep the compatibility with existing generic XML tools (editors, viewers, ...), and to reflect the ASIP name.

ASIP describes a component at three levels:

- interface(s) - at different levels of abstraction
- implementation(s) - in different languages or as a sub-system made of other IPs
- adapters between interfaces, described as transactors

Interfaces are able to refer to hardware elements (wires, asynchronous channels) and software function calls, for either hardware-software design or high level abstraction of functionalities.

Multiple implementations of the same component can refer to associated file sets at different levels of abstraction and/or in different languages. They can also describe a hierarchical sub-system in a language-independent manner, by referring to a set of other IPs and by describing their port-to-port connections. This format is intentionally oriented towards simulation (and debugging) of IPs. For this reason, components also include the information required to compile the source files, transform them, simulate them and exploit the simulation traces.

Adapters (also called “transactors”) between hardware and software interfaces enable the transparent conversion of signals at different levels of abstraction, for use during simulation, visualisation or even synthesis.

Some replication appears in our schema: the data types of a component interface, its ports, and the channels connected to them are identical. This allows tools to access the information more directly. This also allows the design environment to manipulate unconnected elements and to automatically suggest links between elements sharing the same description. Finally, this replication also enables checking the validity of the information.

Buses or NoCs with memory maps can be described by using software abstractions, allowing the normal point-to-point connections between components to be abstracted away and later instantiated with “invisible” buses or NoCs in between.

4.1 XML SCHEMA

The XML Schema for ASIP is provided in Appendix A. As our format is similar to SPIRIT’s IP-XACT format, we had 3 choices for its description:

- Copy-pasting those lines from the SPIRIT description which are similar to our format (but that could lead to license problems)

For example, by keeping in mind that *componentInstances*, *interconnections* and *internalConnections* are elements described in the IP-XACT schema, we could integrate them in our model by directly importing IP-XACT’s schema into ours:



```
<xs:element ref="ourSchema:componentInstances"/>  
<xs:element ref="ourSchema:interconnections" minOccurs="0"/>  
<xs:element ref="ourSchema:internalConnections" minOccurs="0"/>  
<xs:element ref="ourSchema:ourOwnElements" minOccurs="0"/>
```

- Starting from IP-XACT, and applying some extensions to support our asynchronous types, and some restrictions to remove what we don't want to support yet.

```
<xs:schema ..ourSchema...>  
  <xs:import "http:spirit/design.xsd">  
    <xs:restriction weDontNeed SpiritSchema:hierConnections>  
    <xs:extension ourSchema:ourOwnElements>
```

- Keeping our format as we originally described it, and maintaining a list of differences with IP-XACT and some XLD converters to transform IPs between ASIP and IP-XACT.

```
<xs:element ref="ourSchema:components"/>  
<xs:element ref="ourSchema:channels"/>  
<xs:element ref="ourSchema:wires"/>  
<xs:element ref="ourSchema:ourOwnElements"/>
```

This is the route we decided to take to avoid any licensing issue, until our tools get more refined and could support the SPIRIT format more extensively.

4.2 DIFFERENCES WITH SPIRIT CONSORTIUM'S IP-XACT FORMAT

IP-XACT's main objective is the packaging of IPs in order to build electronic systems interconnecting packaged IPs via buses. The ability to describe hierarchical sub-components makes it possible to describe the internals of IPs for use in the Design Environments, but is only a side benefit. On the other hand, our IP format is aimed at describing electronic modules (synchronous or asynchronous) in a hierarchical manner from the high-level IP view down to the lowest possible level, in order to perform mixed-mode simulation and visualisation of signal activity as well at the transaction level than at the transistors level. Being able to build the final system of interconnected IPs described with our format is of course necessary for the simulation of the system, but the integration with buses and NoCs is not the main objective.

IP-XACT was created with SoC design in mind rather than general hardware design. It targets interconnects based on bus structures, and makes design composition without a bus awkward: component interfaces are always using bus interfaces and memory maps are omnipresent. Our format is targeting point-to-point connections between modules, with an abstraction mechanism which allows "virtual" point-to-point connections to actually implement intermediate NoCs.

Regarding the design of asynchronous circuits, it is possible to describe asynchronous channels with IP-XACT as a set of wires grouped as a bus interface with the ability to specify a protocol. Each type of channel (single-rail 4-phase, dual-rail, etc.) could therefore be a specific bus interface. It is however unclear whether the asynchronous properties useful to our DE tools would be able to be described in SPIRIT without using tricks such as string properties parsed a-posteriori.

As the integration with the SPIRIT format is not critical to the success of this project, we decide to postpone it to the end of the project, when we will have a set of tools able to demonstrate the usefulness and necessity of each asynchronous property and asynchronous construct.

4.3 FUTURE EXTENSIONS

This section provides a list of features that have not been integrated in the ASIP format. Those features are bringing more complexity to the schema, while the benefits they provide to the design tools are still uncertain. Therefore we decided to keep them on hold until our tools from WP5 are implemented.



- Hierarchical channels

In the same way as hierarchical components can describe a sub-system of components interconnected by channels and wires and hide the complexity in the hierarchy, hierarchical channels can describe a sub-system of 1 input channel and 1 output channel connecting components in between.

Hierarchical channels might be useful during visualisation, as they provide the ability to hide adapter components: those components doing protocol translations between their input and their output. At many levels of abstraction, only the data is important during debugging. Hierarchical channels can abstract away the components acting on the control part and not modifying the data.

- Extension of *AsipReq3.2*: versioning

New versions of a component should indicate if they are bug fixes or changes in functionalities. A design referencing a component which has seen a bug fix should use the new version of the component, while a design referencing a component which has been functionally modified should keep using the previous version.

- Support for parametric IP configuration

Although the current ASIP schema can use verilog modules with parameters, the ASIP components themselves cannot be parameterised. This will become a limitation when designing libraries of components and will need to be addressed.

- Design constraints

Include timing constraints/information, in order to let the Design Environment automatically suggest synchronous-asynchronous converters.

This was one of the original ideas behind the DE, which was supposed to automatically suggest the best kind of adapters between synchronous and asynchronous domains of the GALS system, based on some specified properties of the synchronous components. However, we have not managed to isolate a proper list of properties able to automate efficiently the process. It seems that too many properties are required for the DE to “guess” the designers preferred choice, and it is therefore more efficient to let the designer choose himself the appropriate component in a list rather than make him fill in lists of properties.

We are however keeping a degree of automatism in the choice of the asynchronous protocol converter: linking a single-rail port to a dual-rail port should suggest the appropriate adapter automatically.

- Asynchronous protocols

Asynchronous protocols (described in Section 5.6) are currently described in ASIP as a number of data and control wires. For simplicity, the full encoding of values is not taken into account in our format. For example, a code representing the value 0 by the bits “101”, value 1 by “110” and 2 by “011” could not be specified so precisely: only the fact that 3 wires are used to represent the data can currently be specified. By going through various case studies and examples, we will need to establish if such a description would bring any benefits.

- Complex types

Linked to the previous point, current types are limited to the number of wires in use. For debugging purposes, we need to represent more complex types such as structures of channels and wires.



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

- Library visibility options

All components defined inside a library are visible to the user. We will need to add some visibility flags, to indicate if a component is local to a library, but shouldn't be directly used by designers.



5 ASIP FORMAT STRUCTURE

5.1 TOP-LEVEL DESCRIPTION

The top-level of an ASIP description provides general information about the file, which can be a library of components or a full project. The difference between the two being that a project contains an additional reference to a component designated as the top level. This ASIP description contains the following information:

- ASIP format version
- Project title
- Project version number and description
- An optional reference to the top-level component and implementation.

The ASIP format version satisfies *AsipReq1.4*, allowing the ASIP schema to evolve over time while keeping its compatibility with existing tools. Automatic XML conversions between versions are possible.

The project title is a string chosen by the designer.

The project version number and description is following a scheme repeated at different places in the ASIP format: project version, library version and component version. ASIP doesn't impose a particular version numbering scheme. The designer can chose whatever fits him best: major.minor numbering, date, year of release, code names, etc., but some guidelines will be defined during task 3.4 for the sharing of libraries via a website.

The last item identifies the component and implementation at the top-level of the project. The component is identified by a string of the form "library name::component name:version number", where the library name and the version number are optional. If this reference is not present, the file is considered as a library.

Example:

```
<ASIP-description>
  <header>
    <ASIP-version>0.1</ASIP-version>
    <project-title>my great hardware-software project</project-title>
  </header>

  <version>
    <version-number>2.3</version-number>
    <description>Version 2.3 now supports: ... </description>
  </version>

  <main-circuit>
    <component>myLibrary::top_design:1.0</component>
    <implementation>fpga</implementation>
  </main-circuit>

  (Components declarations)*
</ASIP-description>
```

5.2 COMPONENTS

Component descriptions expose information about:



- Component name
- Version and description
- Interfaces
- Implementations
- Transactors

The component name is, as previously, of the form form “library name::component name”, with the library name (and the “::”) being optional.

The component version and description is identical to the project’s version group described in the previous paragraph. The groups describing the component’s interfaces, implementations and transactors are each described in the following sections.

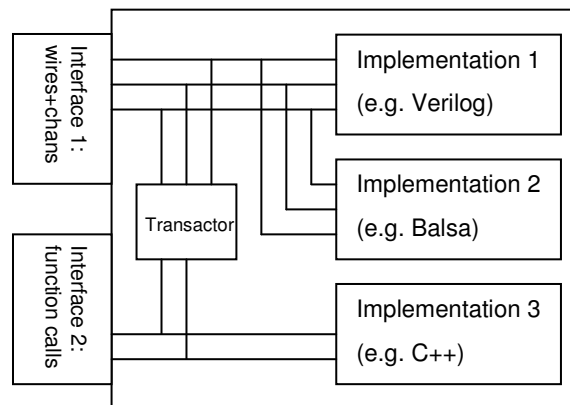


Figure 1: Block diagram of an ASIP component

Example:

```
<ASIP-component name="my_process">
  <version>
    <version-number>1.0</version-number>
    <description>...</description>
  </version>

  <interfaces>
    ...
  </interfaces>

  <parameters>
    ...
  </parameters>

  <implementations>
    ...
  </implementations>

  <transactors>
    ...
  </transactors>
</ASIP-component>
```



5.3 INTERFACE(S)

Asynchronous IPs described using the ASIP format are expected to expose one unique hardware interface to the outside world. However, the possibility of having multiple levels of abstraction can give a component two different interfaces: one hardware interface and one software interface.

5.3.1 Hardware interface

The hardware interface of an asynchronous component could be simply described as a set of wires. In this respect, synchronous formats such as the SPIRIT IP packaging format could be successfully used to describe asynchronous IPs. However, the asynchronous IPs described in this manner require many additional specifications regarding the grouping of wires belonging to the same asynchronous channel and timing information related to them. This extra information can be specified in a more concise manner by using asynchronous constructs and properties. Direct use of asynchronous elements also brings many benefits for visualisation and analysis tools.

Our format describes hardware interfaces as a set of wires and asynchronous channels. Each channel can be specified with a different asynchronous protocol, data width and direction. The protocol, defined by the *code-width*, *data-blocks* and *array-size* attributes, is described in Section 5.6.

Example:

```
<interfaces>
  <ports>
    <port name="activate" type="channel"
          sense="passive" direction="none_sync"/>
    <port name="image_in" type="channel"
          sense="passive" direction="push" code-width="9+1"/>
    <port name="wire_out" type="net"
          direction="output" width="20"/>
  </ports>
</interfaces>
```

5.3.2 Software interface

At the levels of abstraction where the interface cannot be represented with the hardware interface model (PV level) and for software modules during hardware-software co-design, the software interface is described as a set of function calls:

- name
- arguments with a direction and a number of bits

The mapping between software arguments and hardware wires and channels is described in the implementation of the component (Section 5.4.2).

Example:

```
<interfaces>
  <functions>
    <function name="process2_image" hw-addr="0x4110800">
      <port type="function arg" direction="input" name="control" type2="unknown"
            hw-addr="0x4110800-0x4110801"/>
      <port type="function arg" direction="input" name="img_in" type2="1024 bits"
            hw-addr="0x4110000-0x411007F"/>
      <port type="function arg" direction="output" name="img_out" type2="1024 bits"
            hw-addr="0x4110080-0x41100FF"/>
    </function>
  </functions>
</interfaces>
```



5.4 IP IMPLEMENTATION(S)

Multiple implementations of a component are necessary for iterative design at different levels of abstraction, in different languages, and for mixed-mode simulation.

ASIP accepts two kinds of implementations:

- *Sub-circuit*: The component is described as list of sub-components interconnected by a list of wires and channels (described in the same way as the IP interface). This allows the main IP to be described as a hierarchy (or tree) of modules.
- *Single element*: The component is a leaf in the tree. Its description includes: the language used for its description (Verilog, Balsa or C), a list of source files and the specification of a direct mapping between the module's arguments in this particular language and the wires/channels of the hardware interface.

5.4.1 Sub-circuit

A simple list of components linked together by channels and wires. The connections between the main component's ports and the sub-components are integrated in the same list of channels and wires, where the parent component is designated by an empty *comp-id* field.

Example:

```
<implementation type="sub-circuit" name="default">
  <components>
    <component id="interface" impl="default">chain_ROM_if_256</component>
    <component id="I1" impl="default">rom256x32_8</component>
  </components>
  <channels>
    <channel is-port="true" name="c" direction="push" code-width="5+1" array-size="2">
      <channel-port type="initiator" comp-id="" port-name="c"/>
      <channel-port type="target" comp-id="interface" port-name="c"/>
    </channel>
    <channel is-port="true" name="r" direction="push" code-width="5+1" array-size="2">
      <channel-port type="initiator" comp-id="interface" port-name="r"/>
      <channel-port type="target" comp-id="" port-name="r"/>
    </channel>
  </channels>
  <wires>
    <wire is-port="true" name="nreset">
      <connection comp-id="interface" port-name="nreset"/>
    </wire>
    <wire is-port="true" name="delay_control" width="3">
      <connection comp-id="interface" port-name="delay_control"/>
    </wire>
    <wire name="Q" width="32">
      <connection comp-id="interface" port-name="Q"/>
      <connection comp-id="I1" port-name="Q"/>
    </wire>
    <wire name="CLK">
      <connection comp-id="interface" port-name="CLK"/>
      <connection comp-id="I1" port-name="CLK"/>
    </wire>
    <wire name="CEN">
      <connection comp-id="interface" port-name="CEN"/>
      <connection comp-id="I1" port-name="CEN"/>
    </wire>
    <wire name="A" width="8">
      <connection comp-id="interface" port-name="A"/>
      <connection comp-id="I1" port-name="A"/>
    </wire>
  </wires>
</implementation>
```



5.4.2 Single element

The main role of the *single element* structure is to define the mapping between the component's interface (either hardware or software interface) and the targeted element (which can also be either a hardware or a software source code).

The following information is described:

- Associated file
- Procedure/module/function name in the associated file
- Arguments to ports mapping
- Simulation target

These 4 items are necessary and sufficient for the automatic generation of co-simulation interfaces and the selection and launch of the appropriate simulators.

The only important point to note is that tools are relying on the associated file's extension to determine its file type (verilog HDL, C++, etc.). This allows the DE to select the appropriate targets based on the file type.

Example:

```
<implementation type="single-element" name="default">
  <file>chain-structure.v</file>
  <procedure>chain_routecode_g3card</procedure>
  <args>
    <arg name="routeh" type="direct port connection"/>
    <arg name="routel" type="direct port connection"/>
    <arg name="routeeop" type="direct port connection"/>
    <arg name="tselect" type="direct port connection"/>
  </args>
  <target compatible-target-versions="" incompatible-target-versions="">VCS</target>
</implementation>
```

5.5 TRANSACTORS BETWEEN HARDWARE AND SOFTWARE INTERFACES

Transactors are pieces of code able to perform the translation of signals at different levels of abstraction.

By describing transactors in a language which can be executed (or efficiently simulated) transactors are able to be used by software tools manipulating signals at multiple levels of abstractions, allowing a transparent co-simulation of IPs at different levels of abstraction, and the visualisation of signals at levels of abstraction independently of the simulated level of abstraction.

By describing transactors in a language which can be synthesised, transactors are able to be used in the synthesised circuit, either for fabrication or for targeting hardware emulators.

The Balsa language can be simulated and synthesised, depending on the simulation target, and can be in-lined directly in the ASIP format.

ASIP transactors are converting signals between CSP and PV levels, and vice-versa. Other (hardware) levels of abstraction can be automated fro and to CSP by making use of the channels' protocol descriptions. The two types of transactors, PV to CSP and CSP to PV, refer to a list of CSP ports via the *involved-port* attributes, and to PV functions via the *involved-PV-function* attributes.

Example:

```
<transactor type="PV to CSP">
  <involved-port name="command_selection"/>
  <involved-port name="image_in"/>
  <involved-port name="image_out"/>
</transactor>
```



```
<involved-PV-function name="process_image"/>
<transactor-code
  language="Balsa"
  inline-code="command_selection &lt;- 1 ||
              image_in &lt;- img_in ||
              img_out &lt;- image_out"/>
</transactor>

<transactor type="CSP to PV">
  <involved-port name="command_selection"/>
  <involved-port name="image_in"/>
  <involved-port name="image_out"/>
  <involved-PV-function name="process_image"/>
  <involved-PV-function name="process2_image"/>
  <transactor-code
    language="Balsa"
    inline-code="select command_selection, image_in then case command_selection in 1:
                  image_out &lt;- EXTERN[process_image(image_in)] | 2: image_out &lt;-
                  EXTERN[process2_image(image_in)] end end"/>
  </transactor-code
</transactor>
```

5.6 ASYNCHRONOUS PROTOCOLS

Asynchronous protocol specifications are used in the description of asynchronous channels and ports. The current format describes the number of data wires, number of control wires per data, number of data blocks per control sets and number of data+control blocks by using 3 attributes: *code-width*, *data-blocks* and *array-size*.

Code-width is a string of the form "d+c", where d and c are respectively the numbers of wires used to represent the data and the control in the asynchronous encoding. For example, single-rail, dual-rail and 1-of-4 are respectively represented as "1+2", "2+1", "4+1". Depending of the encoding, one "block" can carry different numbers of bits, for example 1 for single-rail, 2 for dual-rail and 2 for 1-of-4. The number of "blocks" (and therefore the number of bits able to be encoded) making up the signal is given by the *data-groups* attribute. For example, an 8-bit data channel in single-rail, dual-rail and 1-of-4 is represented as "8x1+2", "4x2+1", "2x4+1". Finally, the *array-size* attribute is used to describe arrays of channels with independent control wires.

Example:

```
<channel name="... code-width="4+1" data-blocks="2" array-size="10"/> = 10 8-bit 1-of-4 channels
```

5.7 GLOBAL TYPES

Global code width definitions can be added at the top level of the ASIP file, just under the ASIP-description level. These are only useful at the moment for the Balsa→ASIP converter, in order to keep Balsa type names in the ASIP file.

Example:

```
<code-width-type-definitions>
  <code-width-type-def name="36-bit single-rail" value="36+2">
</code-width-type-definitions>
```



APPENDIX

A APPENDIX A - ASIP XML SCHEMA

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="arg">
    <xs:complexType>
      <xs:attribute name="channel-name" type="xs:NMTOKEN" use="optional" />
      <xs:attribute name="channel-field" type="xs:string" use="optional" />
      <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="type" type="xs:string" use="required" />
      <xs:attribute name="array-index" type="xs:integer" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="args">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="arg" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="ASIP-component">
    <xs:complexType>
      <xs:choice>
        <xs:element ref="desc" />
        <xs:element ref="implementations" />
        <xs:element ref="interfaces" />
        <xs:element ref="parameters" />
        <xs:element ref="transactors" />
      </xs:choice>
      <xs:attribute name="name" type="xs:ID" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ASIP-description">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="header" />
        <xs:element ref="main-circuit" />
        <xs:element ref="code-width-types-definitions" />
        <xs:element ref="ASIP-component" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="ASIP-version">
    <xs:complexType mixed="true" />
  </xs:element>

  <xs:element name="channel">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="channel-port" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="data-blocks" type="xs:integer" use="optional" />
      <xs:attribute name="array-size" type="xs:integer" use="optional" />
      <xs:attribute name="is-port" type="xs:boolean" use="optional" />
      <xs:attribute name="code-width" type="xs:string" use="optional" />
      <xs:attribute name="direction" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="none_sync" />
            <xs:enumeration value="pull" />
            <xs:enumeration value="push" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

```
</xs:element>

<xs:element name="channel-port">
  <xs:complexType>
    <xs:attribute name="port-name" type="xs:NMTOKEN" use="required" />
    <xs:attribute name="comp-id" type="xs:string" use="required" />
    <xs:attribute name="internal-name" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="i1" />
          <xs:enumeration value="i2" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="initiator" />
          <xs:enumeration value="target" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="code-width" type="xs:string" use="optional" />
    <xs:attribute name="direction" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="pull" />
          <xs:enumeration value="push" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="channel-with-converters">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="channel-port" maxOccurs="unbounded" />
      <xs:element ref="converter" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:ID" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="channels">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="channel" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="channels-with-converters">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="channel-with-converters" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="code-width-type-def">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="value" type="xs:NMTOKEN" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="code-width-types-definitions">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="code-width-type-def" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="component">
  <xs:complexType mixed="true">
    <xs:attribute name="impl" type="xs:string" use="optional" />
    <xs:attribute name="params2" type="xs:string" use="optional" />
    <xs:attribute name="id" type="xs:NMTOKEN" use="optional" />
  </xs:complexType>
```



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

```
</xs:element>

<xs:element name="components">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="component" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="connection">
  <xs:complexType>
    <xs:attribute name="port-name" type="xs:NMTOKEN" use="required" />
    <xs:attribute name="comp-id" type="xs:NMTOKEN" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="converter">
  <xs:complexType>
    <xs:attribute name="port-names" type="xs:string" use="required" />
    <xs:attribute name="conv-name" type="xs:NMTOKEN" use="required" fixed="BrzPassivatorPush" />
  </xs:complexType>
</xs:element>

<xs:element name="desc">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="directory">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="file">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="header">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ASIP-version" />
      <xs:element ref="title" />
      <xs:element ref="project-version" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="implementation">
  <xs:complexType mixed="true">
    <xs:choice>
      <xs:element ref="args" />
      <xs:element ref="channels" />
      <xs:element ref="channels-with-converters" />
      <xs:element ref="components" />
      <xs:element ref="desc" />
      <xs:element ref="directory" />
      <xs:element ref="file" />
      <xs:element ref="procedure" />
      <xs:element ref="function-calls" />
      <xs:element ref="target" />
      <xs:element ref="tool-options" />
      <xs:element ref="wires" />
    </xs:choice>
    <xs:attribute name="name" type="xs:NMTOKEN" use="optional" />
    <xs:attribute name="type" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="single-element" />
          <xs:enumeration value="sub-circuit" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="implementations">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="implementation" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

```
</xs:complexType>
</xs:element>

<xs:element name="interfaces">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ports" />
      <xs:element ref="functions" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="involved-port">
  <xs:complexType>
    <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="involved-PV-function">
  <xs:complexType>
    <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="main-circuit">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="component" />
      <xs:element ref="implementation" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="parameters" type="xs:string" />

<xs:element name="port">
  <xs:complexType>
    <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
    <xs:attribute name="width" type="xs:NMTOKEN" use="optional" />
    <xs:attribute name="sense" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="active" />
          <xs:enumeration value="passive" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="type2" type="xs:string" use="optional" />
    <xs:attribute name="type-to-circuit" type="xs:string" use="optional" />
    <xs:attribute name="code-width" type="xs:string" use="optional" />
    <xs:attribute name="array-size" type="xs:integer" use="optional" />
    <xs:attribute name="hw-addr" type="xs:NMTOKEN" use="optional" />
    <xs:attribute name="type" type="xs:string" use="required" />
    <xs:attribute name="direction" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="inout" />
          <xs:enumeration value="input" />
          <xs:enumeration value="none_sync" />
          <xs:enumeration value="output" />
          <xs:enumeration value="pull" />
          <xs:enumeration value="push" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="data-blocks" type="xs:integer" use="optional" />
  </xs:complexType>
</xs:element>

<xs:element name="ports">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="port" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="procedure">
  <xs:complexType mixed="true" />
</xs:element>
```



GALAXY

GALS InterfAce for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

```
<xs:element name="project-version">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="function">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="port" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
    <xs:attribute name="hw-addr" type="xs:NMTOKEN" use="optional" />
  </xs:complexType>
</xs:element>

<xs:element name="function-call">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="port" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="function-id" type="xs:NMTOKEN" use="required" />
    <xs:attribute name="hw-addr" type="xs:NMTOKEN" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="function-calls">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="function-call" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="functions">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="function" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="target">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="title">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="tool-option">
  <xs:complexType>
    <xs:attribute name="tool" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="balsa2breeze" />
          <xs:enumeration value="breeze-sim" />
          <xs:enumeration value="breeze2verilog" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="key" type="xs:string" use="required" />
    <xs:attribute name="value" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="tool-options">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tool-option" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="transactor">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="involved-port" maxOccurs="unbounded" />
      <xs:element ref="involved-PV-function" maxOccurs="unbounded" />
      <xs:element ref="transactor-code" />
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```



GALAXY

GALS InterfACE for CompleX Digital
SYstem Integration

Confid. Level: Public
Date : 24/04/2008
Issue: 1

```
</xs:element>

<xs:element name="transactor-code">
  <xs:complexType>
    <xs:attribute name="inline-code" type="xs:string" use="required" />
    <xs:attribute name="language" type="xs:NMTOKEN" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="transactors">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="transactor" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="wire">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="connection" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="floating_dangling" type="xs:NMTOKEN" use="optional" />
    <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
    <xs:attribute name="is-port" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="true" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="width" type="xs:NMTOKEN" use="optional" />
  </xs:complexType>
</xs:element>

<xs:element name="wires">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="wire" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```