IP-XACT User Guide

Accellera IP-XACT Working Group

September 2023

Copyright © 2023 Accellera Systems Initiative Inc. All rights reserved.
Accellera Systems Initiative, 8698 Elk Grove Blvd. Suite 1, #114, Elk Grove, CA 95624, USA

Notices

Accellera Systems Initiative Standards documents are developed within Accellera Systems Initiative (Accellera) and its Technical Committee. Accellera develops its standards through a consensus development process, approved by its members and board of directors, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of Accellera and serve without compensation. While Accellera administers the process and establishes rules to promote fairness in the consensus development process, Accellera does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an Accellera Standard is wholly voluntary. Accellera disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other Accellera Standard document.

Accellera does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or suitability for a specific purpose, or that the use of the material contained herein is free from patent infringement. Accellera Standards documents are supplied "**AS IS**."

The existence of an Accellera Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of an Accellera Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change due to developments in the state of the art and comments received from users of the standard. Every Accellera Standard is subjected to review periodically for revision and update. Users are cautioned to check to determine that they have the latest edition of any Accellera Standard.

In publishing and making this document available, Accellera is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is Accellera undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other Accellera Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of Accellera, Accellera will initiate reasonable action to prepare appropriate responses. Since Accellera Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, Accellera and the members of its Technical Committee and Working Groups are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of Accellera Standards are welcome from any interested party, regardless of membership affiliation with Accellera. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Accellera Systems Initiative
8698 Elk Grove Blvd. Suite 1, #114
Elk Grove, CA 95624
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

existence or validity of any patent rights in connection therewith. Accellera shall not be responsible for identifying patents for which a license may be required by an Accellera Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Accellera is the sole entity that may authorize the use of Accellera-owned certification marks and/or trademarks to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use must be granted by Accellera, provided that permission is obtained from and any required fee, if any, is paid to Accellera. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained from Accellera. To arrange for authorization please contact Lynn Garibaldi, Executive Director, Accellera Systems Initiative, 8698 Elk Grove Blvd. Suite 1, #114, Elk Grove, CA 95624, phone (916) 760-1056, e-mail lynn@accellera.org.

Suggestions for improvements to the IP-XACT User Guide are welcome. They should be sent to the group's email Reflector:

ip-xact@lists.accellera.org

The current IP-XACT Working Group web page is:

https://accellera.org/activities/working-groups/ip-xact

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Contents

1. Introduction 1

1.1 Motivation 1

1.2 Audience 1

2. Background 2

3. IEEE 1685-2022 Explained 3

3.1 Basic Topics 6

3.1.1 Component 7

3.1.2 Design and Design Configuration 10

3.1.3 Bus and Abstraction Definition 16

3.1.4 Component Bus Interfaces and Design Interconnections 19

3.1.5 Component Memory Maps and Registers 25

3.1.6 Component Address Spaces and Bus Interface Bridges 31

3.1.7 Type definitions 35

3.2 Advanced Topics 37

3.2.1 Advanced Elements 38

3.2.2 Conditional Elements 46

3.2.3 Parameter Passing 48

3.2.4 Modes and Mode References 53

3.2.5 Structured Ports 54

3.2.6 Tight Generator Interface 58

4. Use Models 61

4.1 Typical Use Models 61

4.1.1 Packaging 61

4.1.2 Assembly 62

4.2 Advanced Use Models 63

4.2.1 Data Exchange Between Tools 63

4.2.2 Proprietary Tool Flows 63

5. Evolution of the Standard 65

5.1 Motivation of each Release 65

5.2 Key Elemental Differences between Adjacent Releases 65

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

1. Introduction

1.1 Motivation

The two main existing sources of information regarding the IP-XACT standard are the actual document defining the standard *(IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows)*, and the XML schema files that define the syntax of the standard. The IEEE document is required to be a normative description of the standard. The XML schema files contain some documentation, but are primarily a definition of the standard in a machine readable format. Neither of these information sources provides a user perspective nor any meaningful usage details. The primary motivation of this document is to fill this gap of missing user oriented documentation regarding the IP-XACT standard.

1.2 Audience

The primary audience for this document is anyone looking to gain an increased understanding of the IPXACT standard with a focus on its practical usage. The content is applicable to IP developers, IP integrators, and tool developers. It is likely more useful for those new to the standard, but does include coverage of advanced topics that might be useful to more experienced users.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

2. Background

Standards are typically created to provide a consistent means of defining information in a specific domain. The IP-XACT standard is no different in this regard. It defines a standard way to describe key details about an IP, such that users of the IP, both people and tools, can access the information in a consistent and potentially automated fashion.

When an IP consumer receives an IP from an internal or external IP provider, the hand-off typically occurs via a large volume of files documenting different views of the IP. RTL source code, documentation, simulation models, and synthesis constraints are common examples of the types of views delivered, but this list is far from complete, and the set of views delivered for different IPs can vary widely. The IP-XACT standard does not attempt to define which views should be provided, nor how they should be organized. The focus of the standard is to act as an electronic data book - its main function is to "document what's there." The most commonly used data documented via the IP-XACT standard falls into the following top-level categories.

— Document key modeling details of an IP such as top-level port names.
— Provide pointers to where different views of the IP exist within the delivered image.
— Document the configuration and interconnection of systems of IPs modeled using IP-XACT.

Documenting the IP modeling details directly in the IP-XACT file allows for access to critical information about an IP without requiring a search for the containing file or parsing of that file to gather the information. Modeling information includes details like top-level ports, logical interfaces, and a detailed description of the memory map.

Providing pointers to where different views reside within an IP image enables the IP user to know which views are present and where they reside without requiring standardization of file names or directory structures. This approach is used to document the location of views such as synthesis constraints, RTL source code, and simulation models, among others.

Documenting how systems of IPs are configured and connected enables the use of automation by design tools to drive the development of systems of components modeled using IP-XACT.

The IP-XACT standard provides for a consistent, machine readable description of an individual IP or system of interconnected IPs. The fact that IP descriptions are written to adhere to a specific set of syntax and semantic rules, as defined in the standard, means that design tools and scripts can leverage these "electronic data books" for documentation generation, test generation, system assembly, script generation for tools flows, or any other operation that typically requires knowledge about the key details defining IP components.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

3. IEEE 1685-2022 Explained

The IP-XACT standard provides XML schemas for different types of XML documents. The different document types are **component**, **design**, **design configuration**, **bus definition**, **abstraction definition**, **type definitions**, **abstractor**, **generator chain**, and **catalog**.

The purpose of a **component** is to enable the (re-)use of an IP through IP-XACT without the need for information from the implementation files. To this end, a **component** documents aspects such as

- the interfaces of an IP such as parameters, registers, ports, and grouping of ports into bus interfaces,
- the views of an IP such as RTL and TLM descriptions, and
- the files implementing each view, such as Verilog, VHDL, and SystemC files.

The purpose of a **design** is to describe the structure of a hierarchical IP and enable generation of views related to logical interconnect (e.g., system memory map) and physical interconnect (e.g., structural HDL). A **design** can be referenced in a **component** view. The combination of **component** and **design** enables the implementation and the (re-)use of hierarchical IP through IP-XACT. A **component** can reference multiple designs. A **design** documents an internal structure of a **component** by describing

- instances of sub-components that are used to implement a component,
- parameter values for component instances, and
- connections between component instances.

The purpose of a **design configuration** is to configure a **design** for a particular purpose by selecting an appropriate combination of views for its component instances. Abstractors, that perform communication abstraction conversion, may be needed on interconnections between component instances for which views have been selected that have a mismatch in communication abstraction (e.g., RTL versus TLM). A **design** can be associated with multiple design configurations. A **design configuration** documents a configuration of a design by describing

- views that are used for component instances and
- abstractor instances that are used for communication abstraction conversion on interconnections.

The purpose of a **bus definition** and an **abstraction definition** is to describe aspects of a hardware communication protocol. Bus and abstraction definitions are referenced in component bus interfaces to indicate which interface uses which protocol and which component ports implement that protocol. Two bus interfaces can be connected if and only if they reference the same bus definition (or if they reference compatible bus definitions; the meaning of compatible is explained in IEEE Std. 1685-2022). If two connected bus interfaces reference different abstraction definitions, then an **abstractor** is required on the interconnection to perform communication abstraction conversion. Abstractor instances are described in design configurations. A **bus definition** documents properties of a hardware communication protocol that are independent of the representation of the protocol such as

- if direct connections between initiator and target interfaces are supported for a protocol and
- if the IP-XACT address calculations apply to a protocol to map target memory maps into initiator address spaces.

An **abstraction definition** documents a representation of a hardware communication protocol in terms of the logical ports and their properties such as direction and number of bits for initiator and target interfaces.

The purpose of **type definitions** is to enable re-use of memory map related elements by providing definitions for access policies, enumerations, register fields, registers, register files, address blocks, banks, memory maps, and memory remaps. **Type definitions** can be instantiated in other type definitions and in components as external type definitions. Elements in external type definitions can be referenced to describe (internal) memory maps or memory map elements.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

The purpose of an **abstractor** is to describe communication abstraction conversion between connected bus interfaces. The required conversion depends on the views of the component instances in a design configuration. Hence, abstractors are instantiated in design configurations for the purpose of modeling or simulating a mixed-abstraction design. An **abstractor** documents IP for communication abstraction conversion and is a specialization of **component** with its own document type.

The purpose of a **generator chain** is to describe flows that are enabled by IP-XACT. A **generator chain** documents a sequence of generators. A generator is a software tool, e.g., a script or an executable, that implements a flow step by using, creating, or manipulating information described in IP-XACT files. A **generator chain** documents such a sequence of flow steps by describing for each generator in the chain

- a URL to access the tool and
- input argument names and values to be provided to the tool.

The purpose of a **catalog** is to manage collections of IP-XACT files by documenting the location of IP-XACT files and the identifiers of the IP-XACT elements documented in those files. For each of the mentioned IPXACT document types, a **catalog** documents — a URL to an IP-XACT file and

- the identifier of the element that is described in that IP-XACT file.

To demonstrate these IP-XACT concepts on a HDL example, we use the Inter-IC Sound (IIS or I2S) example shown in Figure 1.

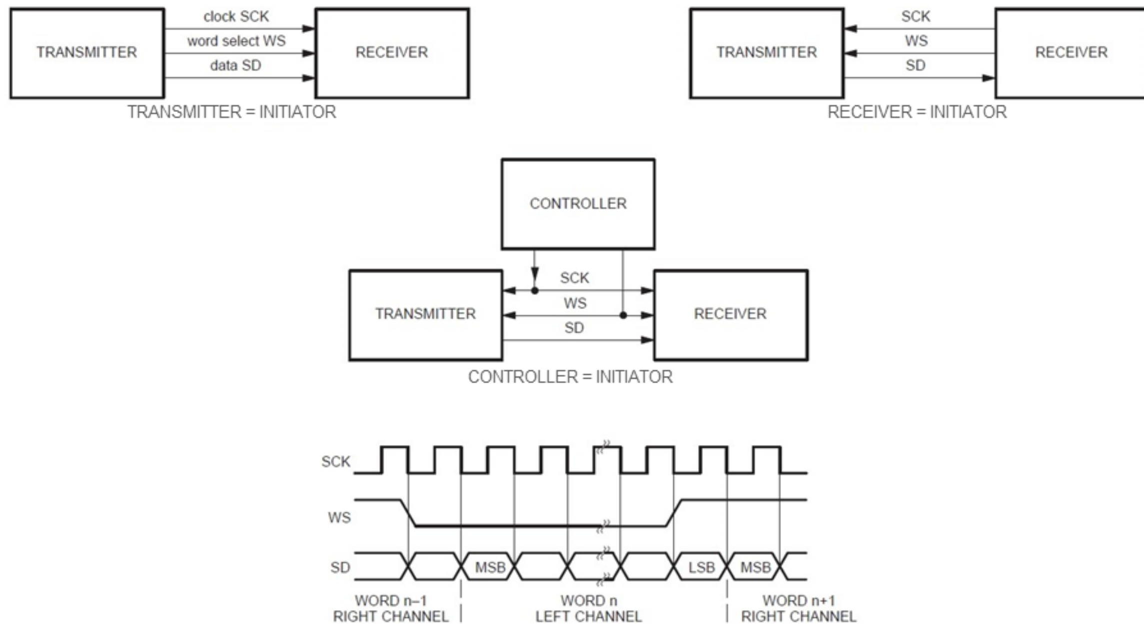

Figure 1 Basic I2S protocol and topologies

The basic topologies contain the following components:

- Transmitter in initiator mode,
- Transmitter in target mode,
- Receiver in initiator mode,
- Receiver in target mode, and
- Controller.

An I2S protocol consists of three signals named `SCK` (serial clock), `WS` (word select), and `SD` (serial data). A component that generates the `SCK` and `WS` signals is an initiator; a component that receives those signals is a target. For each of the mentioned components, we have Verilog module declarations as shown in Example 1, Example 2, Example 3, Example 4, and Example 5.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 1 Module initiator_transmitter

```
module initiator_transmitter(sck, ws, sd);
output wire sck;
output wire ws;
output wire sd;
endmodule
```

Example 2 Module target_transmitter

```
module target_transmitter(sck, ws, sd);
input  wire sck;
input  wire ws;
output wire sd;
endmodule
```

Example 3 Module initiator_receiver

```
module initiator_receiver(sck, ws, sd);
output wire sck;
output wire ws;
input  wire sd;
endmodule
```

Example 4 Module target_receiver

```
module target_receiver(sck, ws, sd);
input  wire sck;
input  wire ws;
input  wire sd;
endmodule
```

Example 5 Module controller

```
module controller(sck, ws);
output wire sck;
output wire ws;
endmodule
```

Each Verilog module is described as a single view in a single component. For instance, IP-XACT component initiator_transmitter contains a view describing the location of the Verilog file initiator_transmitter.v, containing the module declaration, as well as information on the contents of that Verilog file such as the HDL language Verilog and the module name initiator_transmitter. The IP-XACT component also describes the ports with their names and directions.

The three I2S topologies shown in Figure 1 in which the transmitter, receiver, and controller are initiators, can be represented in Verilog as shown in Example 6, Example 7, and Example 8, respectively.

Example 6 Module transmitter_is_initiator

```
module transmitter_is_initiator;

wire u_initiator_transmitter_sck_sig;
wire u_initiator_transmitter_ws_sig;
wire u_initiator_transmitter_sd_sig;

initiator_transmitter u_initiator_transmitter (
      .sck( u_initiator_transmitter_sck_sig ),
      .ws(  u_initiator_transmitter_ws_sig  ),
      .sd(  u_initiator_transmitter_sd_sig  )
      );
target_receiver u_target_receiver (
      .sck( u_initiator_transmitter_sck_sig ),
      .ws(  u_initiator_transmitter_ws_sig  ),
      .sd(  u_initiator_transmitter_sd_sig  )
      );

endmodule
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 7 Module receiver_is_initiator

```
module receiver_is_initiator;

wire u_initiator_receiver_sck_sig;
wire u_initiator_receiver_ws_sig;
wire u_target_transmitter_sd_sig;

initiator_receiver u_initiator_receiver (
      .sck( u_initiator_receiver_sck_sig  ),
      .ws(  u_initiator_receiver_ws_sig   ),
      .sd(  u_target_transmitter_sd_sig )
      );
target_transmitter u_target_transmitter (
      .sck( u_initiator_receiver_sck_sig  ),
      .ws(  u_initiator_receiver_ws_sig   ),
      .sd(  u_target_transmitter_sd_sig )
      );

endmodule
```

Example 8 Module controller_is_initiator

```
module controller_is_initiator;

wire u_controller_sck_sig;
wire u_controller_ws_sig;
wire u_target_transmitter_sd_sig;

controller u_controller (
      .sck( u_controller_sck_sig ),
      .ws(  u_controller_ws_sig  )
      );
target_transmitter u_target_transmitter (
      .sck( u_controller_sck_sig       ),
      .ws(  u_controller_ws_sig        ),
      .sd(  u_target_transmitter_sd_sig )
      );
target_receiver u_target_receiver (
      .sck( u_controller_sck_sig       ),
      .ws(  u_controller_ws_sig        ),
      .sd(  u_target_transmitter_sd_sig )
      );

endmodule
```

These module declarations are also described as views in the IP-XACT components transmitter_is_initiator, receiver_is_initiator, and controller_is_initiator, respectively. The internal structure of these three modules is described in three IP-XACT designs. The design for transmitter_is_initiator instantiates components initiator_transmitter and target_receiver and connects the `sck`, `ws`, and `sd` ports of component instances u_initiator_transmitter and u_target_receiver. The design configuration for that design describes the views that are selected for those component instances from which the module name can be derived for each component instance. Both the design and the design configuration can be referenced from the view in the component transmitter_is_initiator. As a result, that view can be used to generate the module transmitter_is_initiator in Verilog. The exact details are explained in the following sections.

3.1 Basic Topics

This section explains the IP-XACT document types **component**, **design**, **design configuration**, **bus definition**, **abstraction definition**, and **type definitions**. These document types are presented first in line with the introduction given in the previous section. Subsequently, more details of **component** and **design** are presented in the following sections addressing component bus interfaces, design interconnections between bus interfaces, component memory maps and registers, and component address spaces and bus interface bridges

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

to explain how IP-XACT unifies logical interconnect (system memory map) and physical interconnect (structural HDL) in a single description.

3.1.1 Component

In this section, component basics are explained, i.e.,

- file sets with files, and
- model with instantiations, views, and ports.

The explanation is organized as follows. A **component** documents one or more implementations of an IP. Examples of such implementations are different Verilog module declarations describing the interface as closed box view for synthesis, the behavioral code as golden reference view for simulation and verification, the synthesizable code as input for synthesis, and the gate-level netlist as output from synthesis. These different implementations are available in different files. The location of these files is documented in different file sets. The different file sets are associated with different instantiations that describe the meta-data extracted from those files such as module names. The different instantiations are associated with different views of the component such that each view corresponds to one implementation. The component ports are described once. The component ports can be tailored to specific views if needed, for instance to add language-specific type information.

For the explanation, we re-use the Verilog module named initiator_transmitter of Example 1, but with a slightly different interface to explain parameters and wire types as shown in Example 9. For simplicity, we only explain the interface view here; the other views can be described in a similar manner.

Example 9 Module initiator_transmitter with parameter.

```
module initiator_transmitter(sck, ws, sd);

output wire sck;
output wire ws;
output reg  sd;

parameter my_param = 0;

endmodule
```

The code of this module is assumed to be located in the following file.

```
<workdir>/data/ip_lib/initiator_transmitter/INTERFACE/initiator_transmitter.v
```

Example 10 shows a possible IP-XACT component description for this module.

Example 10 Component initiator_transmitter with module parameter

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>initiator_transmitter</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>interface</ipxact:name>
        <ipxact:componentInstantiationRef>hdl-interface</ipxact:componentInstantiationRef>
      </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:name>hdl-interface</ipxact:name>
        <ipxact:language>verilog</ipxact:language>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
        <ipxact:libraryName>initiator_transmitterlib</ipxact:libraryName>
        <ipxact:moduleName>initiator_transmitter</ipxact:moduleName>
        <ipxact:moduleParameters>
          <ipxact:moduleParameter parameterId="my_param" resolve="user" type="longint">
            <ipxact:name>my_param</ipxact:name>
            <ipxact:value>0</ipxact:value>
          </ipxact:moduleParameter>
        </ipxact:moduleParameters>
        <ipxact:fileSetRef>
          <ipxact:localName>fs-interface</ipxact:localName>
        </ipxact:fileSetRef>
      </ipxact:componentInstantiation>
    </ipxact:instantiations>
    <ipxact:ports>
      <ipxact:port>
        <ipxact:name>sck</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>out</ipxact:direction>
        </ipxact:wire>
      </ipxact:port>
      <ipxact:port>
        <ipxact:name>ws</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>out</ipxact:direction>
        </ipxact:wire>
      </ipxact:port>
      <ipxact:port>
        <ipxact:name>sd</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>out</ipxact:direction>
          <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
              <ipxact:typeName>reg</ipxact:typeName>
              <ipxact:viewRef>interface</ipxact:viewRef>
            </ipxact:wireTypeDef>
          </ipxact:wireTypeDefs>
        </ipxact:wire>
      </ipxact:port>
    </ipxact:ports>
  </ipxact:model>
  <ipxact:fileSets>
    <ipxact:fileSet>
      <ipxact:name>fs-interface</ipxact:name>
      <ipxact:file>
        <ipxact:name>../INTERFACE/initiator_transmitter.v</ipxact:name>
        <ipxact:fileType>verilogSource</ipxact:fileType>
        <ipxact:logicalName>initiator_transmitterlib</ipxact:logicalName>
      </ipxact:file>
    </ipxact:fileSet>
  </ipxact:fileSets>
</ipxact:component>
```

Every IP-XACT description starts with the type that is specified using a container element. In this example, the container element is **component**. Subsequently, the identifier for the type is specified using four elements: **vendor**, **library**, **name**, and **version** (VLNV). In this example (see Example 11), the values of these elements are accellera.org, i2s, initiator_transmitter, and 1.0, respectively. Typically, the value of the element **vendor** indicates the organization owning the module and the value of the element **name** indicates the name of the module.

Example 11 Component vendor, library, name, and version (VLNV)

```
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>initiator_transmitter</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
</ipxact:component>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Next, we focus on the file set description. File sets are located in the container element **fileSets**. Multiple file sets can be described using the container element **fileSet**. Each **fileSet** element has a **name** to identify it and a list of files. Each **file** has a **name**. The value of that name element is a path to a source file, in this case the file containing the verilog module definition. The path can be a relative path with respect to the location of the IP-XACT XML file or an absolute path possibly using an environment variable. In this example (see Example 12), we assume that the IP-XACT file is located in the following directory.

```
<workdir>/data/ip_lib/initiator_transmitter/METADATA
```

Hence, the value of the element **name** in element **file** is ../INTERFACE/initiator_transmitter.v. For reuse, it is advised to use relative paths or paths with environment variables. Each **file** also has a **fileType** and a **logicalName**. The **fileType** indicates the type of the file. The **logicalName** indicates the library in which the file is compiled.

Example 12 Component fileSets

```
<ipxact:fileSets>
  <ipxact:fileSet>
    <ipxact:name>fs-interface</ipxact:name>
    <ipxact:file>
      <ipxact:name>../INTERFACE/initiator_transmitter.v</ipxact:name>
      <ipxact:fileType>verilogSource</ipxact:fileType>
      <ipxact:logicalName>initiator_transmitterlib</ipxact:logicalName>
    </ipxact:file>
  </ipxact:fileSet>
</ipxact:fileSets>
```

A **fileSet** is used to describe the files that implement a **componentInstantiation**. The container element **instantiations** contains a **componentInstantiation** element that describes the information that is needed to use the Verilog module declaration of this **component**. The **componentInstantiation** element has a **name** to identify the element. The **language** indicates the hardware description language that is used. The **libraryName** indicates in which library the module is compiled. The **moduleName** indicates the name of the module. The **moduleParameters** is a container element for multiple **moduleParameter** elements. Each **moduleParameter** describes a parameter of the module. The **name** element of the **moduleParameter** is equal to the name of the HDL parameter. The **value** element of the **moduleParameter** is equal to the value of the HDL parameter. The HDL parameter value is a default value; likewise the **moduleParameter** value is a default value. The actual value of this **moduleParameter** can be set if the module is instantiated by using a reference to the value of attribute **parameterId** as shown later. Finally, the **fileSetRef** element is a list of references to local names of file sets. Each reference is contained in a **localName** element. In this example (see Example 13), the **fileSet** named fs-interface implements the **componentInstantiation** named hdl-interface.

Example 13 Component instantiations

```
  <ipxact:instantiations>
    <ipxact:componentInstantiation>
      <ipxact:name>hdl-interface</ipxact:name>
      <ipxact:language>verilog</ipxact:language>
      <ipxact:libraryName>initiator_transmitterlib</ipxact:libraryName>
      <ipxact:moduleName>initiator_transmitter</ipxact:moduleName>
      <ipxact:moduleParameters>
        <ipxact:moduleParameter parameterId="my_param" resolve="user" type="longint">
          <ipxact:name>my_param</ipxact:name>
          <ipxact:value>0</ipxact:value>
        </ipxact:moduleParameter>
      </ipxact:moduleParameters>
      <ipxact:fileSetRef>
        <ipxact:localName>fs-interface</ipxact:localName>
      </ipxact:fileSetRef>
    </ipxact:componentInstantiation>
  </ipxact:instantiations>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Multiple instantiations can be grouped together in a **view**. The container element **views** can have multiple **view** elements. Each **view** has a **name** to identify the element. A **view** can reference a **componentInstantiation**, a **designInstantiation**, and a **designConfigurationInstantiation**. The last two elements are described in Section 3.1.2. In our example (see Example 14), there is a **componentInstantiationRef** element that references the **componentInstantiation** named hdl-interface.

Example 14 Component views

```
<ipxact:views>
  <ipxact:view>
    <ipxact:name>interface</ipxact:name>
    <ipxact:componentInstantiationRef>hdl-interface</ipxact:componentInstantiationRef>
    </ipxact:view>
</ipxact:views>
```

Finally, a **component** contains **ports**. Each **port** element has a **name** to identify the element. Typically, the value of the **name** element is equal to the HDL port name. Furthermore, a **port** has a **wire**, **transactional**, or **structured** element depending on whether the HDL port is a signal-level port, a transaction-level port, or a structured port that is a composition using structures, unions, or SystemVerilog interfaces. In this example, there are **wire** ports. A **wire** port has a **direction** indicating the direction of the port. Possible values are in, out, inout, and phantom. The value phantom is discussed later in Section 3.1.4. The other values directly match with directions in HDLs. A **wire** port can also have a container element **wireTypeDefs** that contains type definitions for that **wire** port. Type definitions are needed if the port type is different from the default port type (wire for Verilog). In the example (see Example 15), the **port** named `sd` is of type reg. For this reason, it contains a **wireTypeDef** indicating the **typeName** and the **viewRef** indicating the view for which the **typeName** applies.

Example 15 Component ports

```
<ipxact:ports>
  <ipxact:port>
    <ipxact:name>sck</ipxact:name>
    <ipxact:wire>
      <ipxact:direction>out</ipxact:direction>
    </ipxact:wire>
  </ipxact:port>
  <ipxact:port>
    <ipxact:name>ws</ipxact:name>
    <ipxact:wire>
      <ipxact:direction>out</ipxact:direction>
    </ipxact:wire>
  </ipxact:port>
  <ipxact:port>
    <ipxact:name>sd</ipxact:name>
    <ipxact:wire>
      <ipxact:direction>out</ipxact:direction>
      <ipxact:wireTypeDefs>
        <ipxact:wireTypeDef>
          <ipxact:typeName>reg</ipxact:typeName>
          <ipxact:viewRef>interface</ipxact:viewRef>
        </ipxact:wireTypeDef>
      </ipxact:wireTypeDefs>
    </ipxact:wire>
  </ipxact:port>
</ipxact:ports>
```

This completes the description of basic component concepts for now. Additional basic concepts such as component bus interfaces, memory maps, and address spaces are explained later in Sections 3.1.4, 3.1.5, and 3.1.6, respectively.

3.1.2 Design and Design Configuration

In this section, design basics are explained, i.e.,

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

— component instances with configurable elements and
— ad hoc connections.

Additionally, design configuration basics are explained, i.e., view configurations.

The explanation is organized as follows. A **design** documents an internal structure of an IP. An example is a structural description between module instances in Verilog. A **design** describes the module instances as component instances. A **design** does not describe the module declarations that should be used for those component instances. This is described by a **design configuration** by configuring a view for each component instance. A **design** also describes component parameter values for component instances. If a component has view-specific parameters, then those parameter values are described in a **design configuration**. Both cases are described in Section 3.2.3 explaining parameter passing. This example contains view-specific **moduleParameters**. Finally, a **design** describes connections between component instances. This section describes connections between ports of component instances. Section 3.1.4 describes connections between bus interfaces of component instances.

Let's re-use the Verilog module named target_receiver as shown before in Example 4 and in Example 16.

Example 16 Module target_receiver

```
module target_receiver(sck, ws, sd);
input wire sck;
input wire ws;
input wire sd;
endmodule
```

The code of this module is assumed to be located in the following file.

```
<workdir>/data/ip_lib/target_receiver/INTERFACE/target_receiver.v
```

An IP-XACT component description can be created for this module in the same way as for the initiator_transmitter module.

The initiator_transmitter module and the target_receiver module are used to create the module named transmitter_is_initiator in Example 17 which was shown before, but is slightly modified due to the parameter in the initiator_transmitter module.

Example 17 Module transmitter_is_initiator

```
module transmitter_is_initiator;

wire u_initiator_transmitter_sck_sig;
wire u_initiator_transmitter_ws_sig;
wire u_initiator_transmitter_sd_sig;

initiator_transmitter #(
       .my_param (1)
       )
u_initiator_transmitter (
      .sck( u_initiator_transmitter_sck_sig ),
      .ws(  u_initiator_transmitter_ws_sig  ),
      .sd(  u_initiator_transmitter_sd_sig  )
      );
target_receiver u_target_receiver (
      .sck( u_initiator_transmitter_sck_sig ),
      .ws(  u_initiator_transmitter_ws_sig  ),
      .sd(  u_initiator_transmitter_sd_sig  )
      );

endmodule
```

Example 18 shows a possible IP-XACT component description for this module.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 18 Component transmitter_is_initiator

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>transmitter_is_initiator</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>rtl</ipxact:name>
        <ipxact:componentInstantiationRef>hdl-rtl</ipxact:componentInstantiationRef>
        <ipxact:designInstantiationRef>hdl-rtl_design</ipxact:designInstantiationRef>
        <ipxact:designConfigurationInstantiationRef>hdl-rtl_design_configuration
        </ipxact:designConfigurationInstantiationRef>
      </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:name>hdl-rtl</ipxact:name>
        <ipxact:language>verilog</ipxact:language>
        <ipxact:libraryName>transmitter_is_initiatorlib</ipxact:libraryName>
        <ipxact:moduleName>transmitter_is_initiator</ipxact:moduleName>
        <ipxact:fileSetRef>
          <ipxact:localName>fs-rtl</ipxact:localName>
        </ipxact:fileSetRef>
      </ipxact:componentInstantiation>
      <ipxact:designInstantiation>
        <ipxact:name>hdl-rtl_design</ipxact:name>
        <ipxact:designRef vendor="accellera.org" library="i2s" name="transmitter_is_initiator_rtl"
          version="1.0"/>
      </ipxact:designInstantiation>
      <ipxact:designConfigurationInstantiation>
        <ipxact:name>hdl-rtl_design_configuration</ipxact:name>
        <ipxact:designConfigurationRef vendor="accellera.org" library="i2s"
          name="transmitter_is_initiator_rtl_cfg" version="1.0"/>
      </ipxact:designConfigurationInstantiation>
    </ipxact:instantiations>
  </ipxact:model>
  <ipxact:fileSets>
    <ipxact:fileSet>
      <ipxact:name>fs-rtl</ipxact:name>
      <ipxact:file>
        <ipxact:name>../RTL/transmitter_is_initiator_rtl.v</ipxact:name>
        <ipxact:fileType>verilogSource</ipxact:fileType>
        <ipxact:logicalName>transmitter_is_initiatorlib</ipxact:logicalName>
      </ipxact:file>
    </ipxact:fileSet>
  </ipxact:fileSets>
</ipxact:component>
```

This IP-XACT component description is similar to the IP component descriptions for the initiator_transmitter and target_receiver modules. The only difference is that it contains a **designInstantiation** and a **designConfigurationInstantiation**. Both elements have a **name** to identify the elements. Both elements also have a reference to a design and a design configuration, respectively, using the **vendor**, **library**, **name**, **version** identifier. These references refer to an IP-XACT design and an IP-XACT design configuration that are described in other files. These descriptions are discussed in the next paragraphs. The new **designInstantiation** and a **designConfigurationInstantiation** are referenced from the **view** named rtl.

An IP-XACT design describes the structure of a component in terms of instances of sub-components and the connectivity between those instances. A possible IP-XACT design description for the transmitter_is_initiator module of Example 17 is shown in Example 19.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 19 Design transmitter_is_initiator_rtl

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>transmitter_is_initiator_rtl</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_initiator_transmitter</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="initiator_transmitter"
        version="1.0"/>
    </ipxact:componentInstance>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_target_receiver</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="target_receiver" version="1.0"/>
    </ipxact:componentInstance>
  </ipxact:componentInstances>
  <ipxact:adHocConnections>
    <ipxact:adHocConnection>
      <ipxact:name>u_initiator_transmitter_sck_u_target_receiver_sck</ipxact:name>
      <ipxact:portReferences>
        <ipxact:internalPortReference componentInstanceRef="u_initiator_transmitter" portRef="sck"/>
        <ipxact:internalPortReference componentInstanceRef="u_target_receiver" portRef="sck"/>
      </ipxact:portReferences>
    </ipxact:adHocConnection>
    <ipxact:adHocConnection>
      <ipxact:name>u_initiator_transmitter_ws_u_target_receiver_ws</ipxact:name>
      <ipxact:portReferences>
        <ipxact:internalPortReference componentInstanceRef="u_initiator_transmitter" portRef="ws"/>
        <ipxact:internalPortReference componentInstanceRef="u_target_receiver" portRef="ws"/>
      </ipxact:portReferences>
    </ipxact:adHocConnection>
    <ipxact:adHocConnection>
      <ipxact:name>u_initiator_transmitter_sd_u_target_receiver_sd</ipxact:name>
      <ipxact:portReferences>
        <ipxact:internalPortReference componentInstanceRef="u_initiator_transmitter" portRef="sd"/>
        <ipxact:internalPortReference componentInstanceRef="u_target_receiver" portRef="sd"/>
      </ipxact:portReferences>
    </ipxact:adHocConnection>
  </ipxact:adHocConnections>
</ipxact:design>
```

This IP-XACT description starts with the container element **design**. Subsequently, the identifier for the type is specified using the four elements: **vendor**, **library**, **name**, and **version**. In this example (see Example 20), the values of these elements are accellera.org, i2s, transmitter_is_initiator_rtl, and 1.0, respectively.

Example 20 Design vendor, library, name, and value

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>transmitter_is_initiator_rtl</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
</ipxact:design>
```

The element **componentInstances** is a container element for all component instances in the **design** as shown in Example 21. Each **componentInstance** has an **instanceName** to identify the instance. The **instanceName** matches with the HDL instance name. Furthermore, a **componentInstance** has a **componentRef** that references a **component** using the **vendor**, **library**, **name**, **version** identifier. The **componentRef** describes the (component) type of the instance.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 21 Design componentInstances

```
<ipxact:componentInstances>
  <ipxact:componentInstance>
    <ipxact:instanceName>u_initiator_transmitter</ipxact:instanceName>
    <ipxact:componentRef vendor="accellera.org" library="i2s" name="initiator_transmitter"
      version="1.0"/>
  </ipxact:componentInstance>
  <ipxact:componentInstance>
    <ipxact:instanceName>u_target_receiver</ipxact:instanceName>
    <ipxact:componentRef vendor="accellera.org" library="i2s" name="target_receiver" version="1.0"/>
  </ipxact:componentInstance>
</ipxact:componentInstances>
```

The element **adHocConnections** is a container element for all ad hoc connections in the **design**. Each **adHocConnection** has a **name** to identify the element. Furthermore, each **adHocConnection** has **portReferences** which is a container element for internal and external port references. An **internalPortReference** is a reference to a **port** of a **componentInstance**. An **externalPortReference** is a reference to a port at the boundary of the **design**. Typically, this port is described in the **component** that contains a **designInstantiation** referencing this design. An **externalPortReference** is not shown in this example (see Example 22).

Example 22 Design adhocConnections

```
<ipxact:adHocConnections>
  <ipxact:adHocConnection>
    <ipxact:name>u_initiator_transmitter_sck_u_target_receiver_sck</ipxact:name>
    <ipxact:portReferences>
      <ipxact:internalPortReference componentInstanceRef="u_initiator_transmitter" portRef="sck"/>
      <ipxact:internalPortReference componentInstanceRef="u_target_receiver" portRef="sck"/>
    </ipxact:portReferences>
  </ipxact:adHocConnection>
  <ipxact:adHocConnection>
    <ipxact:name>u_initiator_transmitter_ws_u_target_receiver_ws</ipxact:name>
    <ipxact:portReferences>
      <ipxact:internalPortReference componentInstanceRef="u_initiator_transmitter" portRef="ws"/>
      <ipxact:internalPortReference componentInstanceRef="u_target_receiver" portRef="ws"/>
    </ipxact:portReferences>
  </ipxact:adHocConnection>
  <ipxact:adHocConnection>
    <ipxact:name>u_initiator_transmitter_sd_u_target_receiver_sd</ipxact:name>
    <ipxact:portReferences>
      <ipxact:internalPortReference componentInstanceRef="u_initiator_transmitter" portRef="sd"/>
      <ipxact:internalPortReference componentInstanceRef="u_target_receiver" portRef="sd"/>
    </ipxact:portReferences>
  </ipxact:adHocConnection>
</ipxact:adHocConnections>
```

An IP-XACT design configuration describes the configuration of a design in terms of view configurations for component instances. Example 23 shows a possible IP-XACT design configuration description for the transmitter_is_initiator module specifying for each component instance the component view that is used to describe the component instance module name. These module names match the module names used in Example 17.

Example 23 DesignConfiguration transmitter_is_initiator_rtl_cfg

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:designConfiguration xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>transmitter_is_initiator_rtl_cfg</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:designRef vendor="accellera.org" library="i2s" name="transmitter_is_initiator_rtl"
    version="1.0"/>
  <ipxact:viewConfiguration>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
    <ipxact:instanceName>u_initiator_transmitter</ipxact:instanceName>
    <ipxact:view viewRef="interface">
      <ipxact:configurableElementValues>
        <ipxact:configurableElementValue referenceId="my_param">1</ipxact:configurableElementValue>
      </ipxact:configurableElementValues>
    </ipxact:view>
  </ipxact:viewConfiguration>
  <ipxact:viewConfiguration>
    <ipxact:instanceName>u_target_receiver</ipxact:instanceName>
    <ipxact:view viewRef="interface"/>
  </ipxact:viewConfiguration>
</ipxact:designConfiguration>
```

This IP-XACT description starts with the container element **designConfiguration**. Subsequently, the identifier for the type is specified using the four elements: **vendor**, **library**, **name**, and **version**. In this example (see Example 24), the values of these elements are accellera.org, i2s, transmitter_is_initiator_rtl_cfg, and 1.0, respectively.

Example 24 DesignConfiguration vendor, library, name, and version

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:designConfiguration xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>transmitter_is_initiator_rtl_cfg</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
</ipxact:designConfiguration>
```

The next element is **designRef** (see Example 25). This is an optional element to reference a **design** indicating to which design this design configuration applies. It is optional because the **component view** element can contain both a **designInstantiation** and a **designConfigurationInstantiation** which also indicates that the referenced **designConfiguration** applies to the referenced **design**.

Example 25 DesignConfiguration designRef

```
  <ipxact:designRef vendor="accellera.org" library="i2s" name="transmitter_is_initiator_rtl"
    version="1.0"/>
```

A design can contain multiple **viewConfiguration** elements (see Example 26). A **viewConfiguration** element describes which **component view** applies to which **componentInstance**. Each **viewConfiguration** contains an **instanceName** and a **viewRef**. The **instanceName** references a **componentInstance** by its **instanceName** element to indicate to which **componentInstance** the **viewConfiguration** applies. The **viewRef** references a **view** by its **name** that exists in the **component** of the referenced **componentInstance**. The **view** element can have **configurableElementValues**. Each **configurableElementValue** has an attribute **referenceId** of which the value must match with a **parameterId** that is defined in the referenced component. The value of the **configurableElementValue** describes the actual value of the referenced parameter for this instance.

Example 26 DesignConfiguration viewConfigurations

```
  <ipxact:viewConfiguration>
    <ipxact:instanceName>u_initiator_transmitter</ipxact:instanceName>
    <ipxact:view viewRef="interface">
      <ipxact:configurableElementValues>
        <ipxact:configurableElementValue referenceId="my_param">1</ipxact:configurableElementValue>
      </ipxact:configurableElementValues>
    </ipxact:view>
  </ipxact:viewConfiguration>
  <ipxact:viewConfiguration>
    <ipxact:instanceName>u_target_receiver</ipxact:instanceName>
    <ipxact:view viewRef="interface"/>
  </ipxact:viewConfiguration>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Although not shown in this example, a **design componentInstance componentRef** can have **configurableElementValues** similar to a **designConfiguration viewConfiguration view**. The **viewConfiguration view configurableElementValues** apply to parameters defined in **componentInstantation** and **designConfigurationInstantiation**. The **componentInstance componentRef configurableElementValues** apply to all parameters that are defined outside of **componentInstantation** and **designConfigurationInstantiation**.

This completes the description of basic design and design configuration concepts. Additional concepts such as interconnections are described in Section 3.1.4.

3.1.3 Bus and Abstraction Definition

Bus and abstraction definitions define interface types for hardware communication protocols. To explain these concepts, we use the example of the Inter-IC Sound (IIS or I2S) protocol and topologies introduced in Figure 1. Recall that the I2S protocol consists of three signals named `SCK` (serial clock), `WS` (word select), and `SD` (serial data). However, these signals only exist in a signal-level representation of the I2S protocol. To enable multiple representations of the same interface, for instance at different levels of abstraction, an interface type definition is split into a bus definition and one or more abstraction definitions. A bus definition describes properties of the bus, while an abstraction definition describes properties of the bus representation.

Example 27 shows a possible bus definition for I2S.

Example 27 BusDefinition I2S

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:busDefinition xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>I2S</ipxact:name>
  <ipxact:version>1.1</ipxact:version>
  <ipxact:description>I2S (Inter-IC Sound) bus definition</ipxact:description>
  <ipxact:directConnection>true</ipxact:directConnection>
  <ipxact:isAddressable>false</ipxact:isAddressable>
</ipxact:busDefinition>
```

This IP-XACT description starts with the container element **busDefinition**. Subsequently, the identifier for the type is specified using the four elements: **vendor**, **library**, **name**, and **version**. In this example, the values of these elements are accellera.org, i2s, I2S, and 1.1, respectively. Additional elements shown in this example **busDefinition** are **description**, **directConnection**, and **isAddressable**. The element **description** provides a human-readable description for the **busDefinition**. The element **directConnection** indicates if direct connections from initiator to target bus interfaces are allowed. For many buses, direct connections are allowed. For asymmetric buses, such as AHB, the value of **directConnection** can be set to false to indicate that an initiator cannot be connected directly to a target. The element **isAddressable** indicates if the bus is addressable meaning the memory maps of targets are mapped in the address spaces of initiators according to the IP-XACT addressing scheme that is explained in the IEEE 1685 standard. For serial buses, such as I2S, this is not the case and the value of **isAddressable** is false.

Example 28 shows a possible abstraction definition for I2S.

Example 28 AbstractionDefinition I2S

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:abstractionDefinition xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
<ipxact:library>i2s</ipxact:library>
<ipxact:name>I2S_rtl</ipxact:name>
<ipxact:version>1.1</ipxact:version>
<ipxact:busType vendor="accellera.org" library="i2s" name="I2S" version="1.1"/>
<ipxact:ports>
  <ipxact:port>
    <ipxact:logicalName>SCK</ipxact:logicalName>
    <ipxact:description>Continuous Serial Clock</ipxact:description>
    <ipxact:wire>
      <ipxact:qualifier>
        <ipxact:isClock>true</ipxact:isClock>
      </ipxact:qualifier>
      <ipxact:onInitiator>
        <ipxact:presence>required</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>out</ipxact:direction>
      </ipxact:onInitiator>
      <ipxact:onTarget>
        <ipxact:presence>required</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
      </ipxact:onTarget>
    </ipxact:wire>
  </ipxact:port>
  <ipxact:port>
    <ipxact:logicalName>WS</ipxact:logicalName>
    <ipxact:description>Word Select</ipxact:description>
    <ipxact:wire>
      <ipxact:onInitiator>
        <ipxact:presence>required</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>out</ipxact:direction>
      </ipxact:onInitiator>
      <ipxact:onTarget>
        <ipxact:presence>required</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
      </ipxact:onTarget>
    </ipxact:wire>
  </ipxact:port>
  <ipxact:port>
    <ipxact:logicalName>SD_IN</ipxact:logicalName>
    <ipxact:description>Serial Data from other device</ipxact:description>
    <ipxact:wire>
      <ipxact:qualifier>
        <ipxact:isData>true</ipxact:isData>
      </ipxact:qualifier>
      <ipxact:onInitiator>
        <ipxact:presence>optional</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
      </ipxact:onInitiator>
      <ipxact:onTarget>
        <ipxact:presence>optional</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>out</ipxact:direction>
      </ipxact:onTarget>
    </ipxact:wire>
  </ipxact:port>
  <ipxact:port>
    <ipxact:logicalName>SD_OUT</ipxact:logicalName>
    <ipxact:description>Serial Data to other device</ipxact:description>
    <ipxact:wire>
      <ipxact:qualifier>
        <ipxact:isData>true</ipxact:isData>
      </ipxact:qualifier>
      <ipxact:onInitiator>
        <ipxact:presence>optional</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>out</ipxact:direction>
      </ipxact:onInitiator>
      <ipxact:onTarget>
        <ipxact:presence>optional</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
        </ipxact:onTarget>
      </ipxact:wire>
    </ipxact:port>
  </ipxact:ports>
</ipxact:abstractionDefinition>
```

This IP-XACT description starts with the container element **abstractionDefinition**. Subsequently, the identifier for the type is specified using the four elements: **vendor**, **library**, **name**, and **version**. In this example (see Example 29), the values of these elements are accellera.org, i2s, I2S_rtl, and 1.1, respectively. The postfix "_rtl" is typically used in the **name** to indicate that the abstraction definition is a signal-level representation.

Example 29 AbstractionDefinition vendor, library, name, and version

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:abstractionDefinition xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>I2S_rtl</ipxact:name>
  <ipxact:version>1.1</ipxact:version>
  ...
</ipxact:abstractionDefinition>
```

Additional elements in an **abstractionDefinition** are **busType** and **ports**. The element **busType** is a reference to a bus definition using the **vendor**, **library**, **name**, **version** identifier to indicate to which bus definition this abstraction definition applies to (see Example 30).

Example 30 AbstractionDefinition busType

```
  <ipxact:busType vendor="accellera.org" library="i2s" name="I2S" version="1.1"/>
```

The element **ports** is a container element for multiple **port** elements. A **port** has a **logicalName** to identify the element. Often, a **logicalName** value matches the logical name of a signal in the definition of a communication protocol such as AMBA AHB. A **port** can have a **description** to provide a human-readable description for the **port**. Next, a **port** must have a **wire** element or a **transactional** element to indicate if the **port** is a signal-level port or a transaction-level port, respectively. In this example (see Example 31), all ports are wire ports. The **wire** element can contain a **qualifier** element to indicate if the **wire port** is a clock port, reset port, address port, or data port using the elements **isClock**, **isReset**, **isAddress**, or **isData**, respectively. Note that in the IEEE 1685-2022 standard additional qualifier element values have been introduced. Finally, the **wire port** can contain **onInitiator**, **onTarget**, and **onSystem** elements that describe properties for the **port** in initiator, target, and system bus interfaces, respectively. The properties that can be described are **presence**, **width**, **allBits**, and **direction**. The **presence** element can take values required, optional, and illegal indicating if the **port** is a required, optional, or illegal element in a bus interface port map. The **width** element describes the number of bits in the **port**. If **width** is not specified, then the number of bits in the port is not constrained. If **width** is specified and the value of the **allBits** element equals true, then all bits in the port shall be mapped. The **direction** element can take values in, out, and inout indicating the direction of the **port**.

Example 31 AbstractionDefinition ports

```
    <ipxact:port>
      <ipxact:logicalName>SCK</ipxact:logicalName>
      <ipxact:description>Continuous Serial Clock</ipxact:description>
      <ipxact:wire>
        <ipxact:qualifier>
          <ipxact:isClock>true</ipxact:isClock>
        </ipxact:qualifier>
        <ipxact:onInitiator>
          <ipxact:presence>required</ipxact:presence>
          <ipxact:width>1</ipxact:width>
          <ipxact:direction>out</ipxact:direction>
        </ipxact:onInitiator>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
        <ipxact:onTarget>
          <ipxact:presence>required</ipxact:presence>
          <ipxact:width>1</ipxact:width>
          <ipxact:direction>in</ipxact:direction>
        </ipxact:onTarget>
      </ipxact:wire>
    </ipxact:port>
```

The example **abstractionDefinition** contains four ports named `SCK`, `WS`, `SD_IN`, and `SD_OUT`. The direction indication in the names of ports `SD_IN` and `SD_OUT` are indicated from the point of view of an initiator interface. For an initiator interface, `SD_IN` has direction in and `SD_OUT` has direction out. For a target interface, `SD_IN` has direction out and `SD_OUT` has direction in. The other ports `SCK` and `WS` have direction out on initiators and direction in on targets. Figure 1 showing the basic I2S protocol and topologies does not differentiate between signals `SD_IN` and `SD_OUT`. That is because signal `SD` always describes serial data going from transmitter to receiver independent of the roles of initiator and target.

3.1.4 Component Bus Interfaces and Design Interconnections

The bus and abstraction definition introduced in the previous section are applied on the initiator_transmitter, target_receiver, and transmitter_is_initiator components that have been discussed earlier.

The component initiator_transmitter of Example 10 is extended with a bus interface shown in Example 32.

Example 32 Component busInterfaces

```
  <ipxact:busInterfaces>
    <ipxact:busInterface>
      <ipxact:name>I</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="i2s" name="I2S" version="1.1"/>
      <ipxact:abstractionTypes>
        <ipxact:abstractionType>
          <ipxact:abstractionRef vendor="accellera.org" library="i2s" name="I2S_rtl" version="1.1"/>
          <ipxact:portMaps>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>SCK</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>sck</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>WS</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>ws</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>SD_OUT</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>sd</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
          </ipxact:portMaps>
        </ipxact:abstractionType>
      </ipxact:abstractionTypes>
      <ipxact:initiator/>
    </ipxact:busInterface>
  </ipxact:busInterfaces>
```

The element **busInterfaces** is a container element for **busInterface** elements. A **busInterface** has a name to identify the element. In the example, the bus interface is named I. A **busInterface** has a **busType** referencing a **busDefinition** using the four elements **vendor**, **library**, **name**, and **version** identifier. It also has

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

abstractionTypes which is a container element for **abstractionType** elements. An **abstractionType** has an **abstractionRef** referencing an **abstractionDefinition** using the four elements **vendor**, **library**, **name**, and **version** identifier. Furthermore, an **abstractionType** has **portMaps** containing **portMap** elements. Each **portMap** contains a **logicalPort** and a **physicalPort** indicating a mapping between the named logical port and the named physical port. This example contains the following port maps:

- `SCK` maps to `sck`,
- `WS` maps to `ws`, and
- `SD_OUT` maps to `sd`.

Finally, a **busInterface** from this component definition contains an **initiator**, **target**, or **system** element to indicate the interface mode of the **busInterface** and the **abstractionDefinition** referenced by the **busInterface**. The interface mode can also be mirrored which is indicated by the elements **mirroredInitiator**, **mirroredTarget**, and **mirroredSystem**. For bus interfaces with a mirrored interface mode, the **direction** of each **logicalPort** is mirrored, i.e., in becomes out and out becomes in.

Similarly, the component target_receiver can be extended with a target bus interface named T containing the following port maps:

- `SCK` maps to `sck`,
- `WS` maps to `ws`, and
- `SD_OUT` maps to `sd`.

With these bus interfaces, the **design** of the component transmitter_is_initiator shown in Example 19 can be rewritten using connections between bus interfaces as shown in Example 33. Such connections are called **interconnections**. Each **interconnection** has a **name** to identify the element. Furthermore, it has multiple **activeInterface** elements indicating the end-points of the **interconnection**. An **activeInterface** references the component instance bus interfaces by using a **componentInstanceRef** attribute indicating the **componentInstance instanceName** and a **busRef** attribute indicating the **busInterface name**.

Example 33 Design transmitter_is_initiator_rtl with interconnections

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>transmitter_is_initiator_rtl</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_initiator_transmitter</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="initiator_transmitter"
        version="1.0">
      </ipxact:componentRef>
    </ipxact:componentInstance>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_target_receiver</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="target_receiver" version="1.0"/>
    </ipxact:componentInstance>
  </ipxact:componentInstances>
  <ipxact:interconnections>
    <ipxact:interconnection>
      <ipxact:name>u_initiator_transmitter_I__u_target_receiver_T</ipxact:name>
      <ipxact:activeInterface componentInstanceRef="u_initiator_transmitter" busRef="I"/>
      <ipxact:activeInterface componentInstanceRef="u_target_receiver" busRef="T"/>
    </ipxact:interconnection>
  </ipxact:interconnections>
</ipxact:design>
```

The connectivity between the ports of component instances is determined by connecting the physical ports that are mapped to the same logical port in all interconnected bus interfaces. A component port can be mapped

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

to multiple logical ports in one bus interface. A component port can also be mapped in multiple bus interfaces. In this way, complex connectivity can be created using bus interfaces.

To complete the basic I2S topologies, we describe new components initiator_receiver and target_transmitter. The component initiator_receiver has ports `sck`, `ws`, and `sd` with directions out, out, and in, respectively. These ports are mapped in an initiator bus interface named I as follows:

- `SCK` maps to `sck`,
- `WS` maps to `ws`, and
- `SD_IN` maps to `sd`.

The component target_transmitter has ports `sck`, `ws`, and `sd` with directions in, in, out, respectively. These ports are mapped in a target bus interface named T as follows:

- `SCK` maps to `sck`,
- `WS` maps to `ws`, and
- `SD_IN` maps to `sd`.

With these components, the hierarchical component receiver_is_initiator is created using the design description in Example 34.

Example 34 Design receiver_is_initiator_rtl with interconnections

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>receiver_is_initiator_rtl</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_initiator_receiver</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="initiator_receiver"
        version="1.0"/>
    </ipxact:componentInstance>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_target_transmitter</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="target_transmitter"
        version="1.0"/>
      </ipxact:componentInstance>
  </ipxact:componentInstances>
  <ipxact:interconnections>
    <ipxact:interconnection>
      <ipxact:name>u_initiator_receiver_I__u_target_transmitter_T</ipxact:name>
      <ipxact:activeInterface componentInstanceRef="u_initiator_receiver" busRef="I"/>
      <ipxact:activeInterface componentInstanceRef="u_target_transmitter" busRef="T"/>
    </ipxact:interconnection>
  </ipxact:interconnections>
</ipxact:design>
```

Finally, there is the challenge of describing the last I2S topology containing a component controller as initiator. This component controller has ports `sck` and `ws`, both with direction out. It has an initiator bus interface named I with the following port maps:

- `SCK maps to sck, and`
- `WS maps to ws.`

The challenge is to reuse the descriptions of components target_transmittter and target_receiver and to describe the connectivity between controller, transmitter, and receiver. To this end, a new component named bridge is created that uses phantom ports. A phantom port is a **component port** that has **direction** phantom. This direction value indicates that the IP-XACT component port has no HDL port equivalent. The component

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

bridge has ports sck, ws, and sd that all have direction phantom. It has a target interface T and two initiator interfaces I1 and I2 with the following port maps.

— For target bus interface T, `SCK` maps to `sck` and `WS` maps to `ws`.
— For initiator bus interface I1, `SCK` maps to `sck`,`WS` maps to `ws`, and `SD_IN` maps to `sd`.
— For initiator bus interface I2, `SCK` maps to `sck`, `WS` maps to `ws`, and `SD_OUT` maps to `sd`.

Since the component bridge has no physical ports, an HDL netlister will not instantiate the component in the generated HDL netlist. Rather, it will generate the connectivity described by the bridge that results from mapping the phantom ports to multiple bus interfaces as shown in Figure 2. Example 35 shows the corresponding IP-XACT component description with phantom ports and a **componentInstantiation** element containing an **isVirtual** element with value true to indicate that its component instances should not be netlisted.

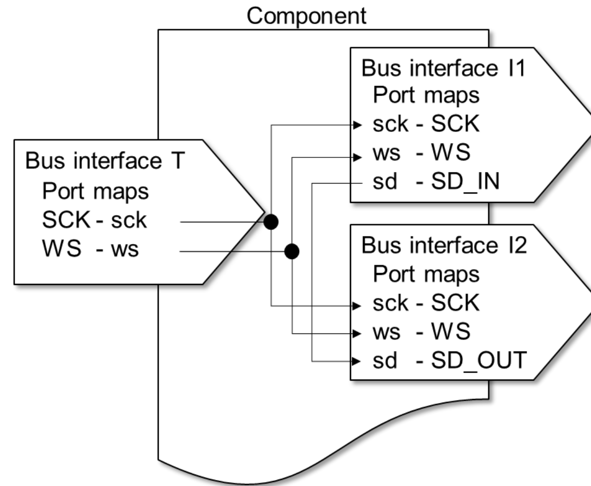

Figure 2 Graphical illustration of HDL connectivity resulting from component bridge port maps of phantom ports

Example 35 Component with phantom ports

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>bridge</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:busInterfaces>
    <ipxact:busInterface>
      <ipxact:name>T</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="i2s" name="I2S" version="1.1"/>
      <ipxact:abstractionTypes>
        <ipxact:abstractionType>
          <ipxact:abstractionRef vendor="accellera.org" library="i2s" name="I2S_rtl" version="1.1"/>
          <ipxact:portMaps>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>SCK</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>sck</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>WS</ipxact:name>
              </ipxact:logicalPort>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
          <ipxact:physicalPort>
            <ipxact:name>ws</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
      </ipxact:portMaps>
    </ipxact:abstractionType>
  </ipxact:abstractionTypes>
  <ipxact:target/>
</ipxact:busInterface>
<ipxact:busInterface>
  <ipxact:name>I1</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="i2s" name="I2S" version="1.1"/>
  <ipxact:abstractionTypes>
    <ipxact:abstractionType>
      <ipxact:abstractionRef vendor="accellera.org" library="i2s" name="I2S_rtl" version="1.1"/>
      <ipxact:portMaps>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>SCK</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>sck</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>WS</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>ws</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>SD_IN</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>sd</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
      </ipxact:portMaps>
    </ipxact:abstractionType>
  </ipxact:abstractionTypes>
  <ipxact:initiator/>
</ipxact:busInterface>
<ipxact:busInterface>
  <ipxact:name>I2</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="i2s" name="I2S" version="1.1"/>
  <ipxact:abstractionTypes>
    <ipxact:abstractionType>
      <ipxact:abstractionRef vendor="accellera.org" library="i2s" name="I2S_rtl" version="1.1"/>
      <ipxact:portMaps>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>SCK</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>sck</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>WS</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>ws</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>SD_OUT</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>sd</ipxact:name>
          </ipxact:physicalPort>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
          </ipxact:portMap>
        </ipxact:portMaps>
      </ipxact:abstractionType>
    </ipxact:abstractionTypes>
    <ipxact:initiator/>
  </ipxact:busInterface>
</ipxact:busInterfaces>
<ipxact:model>
  <ipxact:views>
    <ipxact:view>
      <ipxact:name>virtual</ipxact:name>
      <ipxact:componentInstantiationRef>hdl-virtual</ipxact:componentInstantiationRef>
    </ipxact:view>
  </ipxact:views>
  <ipxact:instantiations>
    <ipxact:componentInstantiation>
      <ipxact:name>hdl-virtual</ipxact:name>
      <ipxact:isVirtual>true</ipxact:isVirtual>
    </ipxact:componentInstantiation>
  </ipxact:instantiations>
  <ipxact:ports>
    <ipxact:port>
      <ipxact:name>sck</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>phantom</ipxact:direction>
      </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
      <ipxact:name>ws</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>phantom</ipxact:direction>
      </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
      <ipxact:name>sd</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>phantom</ipxact:direction>
      </ipxact:wire>
    </ipxact:port>
  </ipxact:ports>
</ipxact:model>
</ipxact:component>
```

The design for component controller_is_initiator can now be described in Example 36.

Example 36 Design controller_is_initiator_rtl with interconnections

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>controller_is_initiator_rtl</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_controller</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="controller" version="1.0"/>
    </ipxact:componentInstance>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_bridge</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="i2s" name="bridge" version="1.0"/>
    </ipxact:componentInstance>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_target_transmitter</ipxact:instanceName>
        <ipxact:componentRef vendor="accellera.org" library="i2s" name="target_transmitter"
          version="1.0"/>
    </ipxact:componentInstance>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_target_receiver</ipxact:instanceName>
        <ipxact:componentRef vendor="accellera.org" library="i2s" name="target_receiver"
          version="1.0"/>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
    </ipxact:componentInstance>
  </ipxact:componentInstances>
  <ipxact:interconnections>
    <ipxact:interconnection>
      <ipxact:name>u_controller_I__u_bridge_T</ipxact:name>
      <ipxact:activeInterface componentInstanceRef="u_controller" busRef="I"/>
      <ipxact:activeInterface componentInstanceRef="u_bridge" busRef="T"/>
    </ipxact:interconnection>
    <ipxact:interconnection>
      <ipxact:name>u_bridge_I1__u_target_transmitter_T</ipxact:name>
      <ipxact:activeInterface componentInstanceRef="u_bridge" busRef="I1"/>
      <ipxact:activeInterface componentInstanceRef="u_target_transmitter" busRef="T"/>
    </ipxact:interconnection>
    <ipxact:interconnection>
      <ipxact:name>u_bridge_I2__u_target_receiver_T</ipxact:name>
      <ipxact:activeInterface componentInstanceRef="u_bridge" busRef="I2"/>
      <ipxact:activeInterface componentInstanceRef="u_target_receiver" busRef="T"/>
    </ipxact:interconnection>
  </ipxact:interconnections>
</ipxact:design>
```

The HDL netlist for component controller_is_initiator will be similar to Example 37.

Example 37 Module controller_is_initiator_rtl

```
module controller_is_initiator;

wire u_controller_sck_sig;
wire u_controller_ws_sig;
wire u_target_transmitter_sd_sig;

controller u_controller (
      .sck( u_controller_sck_sig ),
      .ws(  u_controller_ws_sig  )
      );
target_transmitter u_target_transmitter (
      .sck( u_controller_sck_sig       ),
      .ws(  u_controller_ws_sig        ),
      .sd(  u_target_transmitter_sd_sig )
      );
target_receiver u_target_receiver (
      .sck( u_controller_sck_sig       ),
      .ws(  u_controller_ws_sig        ),
      .sd(  u_target_transmitter_sd_sig )
      );

endmodule
```

The application of phantom ports as explained above is universal. It can be applied for many types of interconnects including:

- interrupt slot assignment, where a phantom port can be used to route interrupt requests;
- clock and reset distribution, where phantom ports can be used to route clock and reset sources;
- daisy chaining for JTAG and test buses, where phantom ports can be used to route serial data.

This way of working clearly separates the roles of IP provider and IP integrator. The role of the IP provider is to provide IP-XACT component descriptions that target reuse. Hence, bus interfaces should be defined to identify complete hardware interfaces such that all ports are available in the interface to implement protocol conversions or protocol abstractions. The role of the IP integrator is to create integration-specific components with phantom ports to describe designs, thereby avoiding ad hoc connections. This way of working enables mixed-abstraction designs.

3.1.5 Component Memory Maps and Registers

The I2S bus definition used so far is an example of a non-addressable bus definition. Here, we switch to an addressable bus definition to make the link to component memory maps and registers. As an example, we use a **busDefinition** that has the four elements **vendor**, **library**, **name**, and **version** with the values accellera.org,

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

amba3, APB3, and 1.0, respectively, and the **isAddressable** element value set to true. There is an associated **abstractionDefinition** with a **vendor**, **library**, **name**, and **version** identifier equal to accellera.org, amba3, APB3_rtl, and 1.0. The **ports** of the **abstractionDefinition** are described according the AMBA3 APB specification. We refer to those ports using the standard signal names `PCLK`, `PRESETn`, and so on.

If a **component busInterface** in **target** mode references an addressable **busDefinition**, then the bus interface must either reference a **component memoryMap** or contain **bridges**. The concept of **bridges** is discussed in Section 3.1.6. Here, the concept of **memoryMaps** is explained. For this purpose, the **component** description in Example 38 is used which describes only one register to limit the amount of space used.

Example 38 Component with target busInterface and memoryMap

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>ip</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:busInterfaces>
    <ipxact:busInterface>
      <ipxact:name>Target</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="APB3" version="1.0"/>
      <ipxact:target>
        <ipxact:memoryMapRef memoryMapRef="RegisterMap"/>
      </ipxact:target>
    </ipxact:busInterface>
  </ipxact:busInterfaces>
  <ipxact:memoryMaps>
    <ipxact:memoryMap>
      <ipxact:name>RegisterMap</ipxact:name>
      <ipxact:addressBlock>
        <ipxact:name>ControlSpace</ipxact:name>
        <ipxact:baseAddress>'h0</ipxact:baseAddress>
        <ipxact:range>'h1000</ipxact:range>
        <ipxact:width>32</ipxact:width>
        <ipxact:accessPolicies>
          <ipxact:accessPolicy>
        <ipxact:access>read-write</ipxact:access>
           </ipxact:accessPolicy>
        </ipxact:accessPolicies>
        <ipxact:register>
          <ipxact:name>STAT</ipxact:name>
          <ipxact:description>Status register. Collection of Status flags including interrupt status
before enabling
          </ipxact:description>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:size>32</ipxact:size>
          <ipxact:field>
            <ipxact:name>RXFIFO_NE</ipxact:name>
            <ipxact:description>RX-FIFO Not Empty. This interrupt capable status flag indicates  the
RX-FIFO status and associated interrupt status before the enable stage. The flag can only be
implicitly cleared by reading the RXFIFO_DAT register
            </ipxact:description>
            <ipxact:bitOffset>0</ipxact:bitOffset>
            <ipxact:bitWidth>1</ipxact:bitWidth>
            <ipxact:resets>
              <ipxact:reset>
                <ipxact:value>'h0</ipxact:value>
                <ipxact:mask>'h1</ipxact:mask>
              </ipxact:reset>
            </ipxact:resets>
            <ipxact:fieldAccessPolicies>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
              <ipxact:fieldAccessPolicy>
                <ipxact:access>read-only</ipxact:access>
              </ipxact:fieldAccessPolicy>
            </ipxact:fieldAccessPolicies>
            <ipxact:enumeratedValues>
              <ipxact:enumeratedValue usage="read">
                <ipxact:name>EMPTY</ipxact:name>
                <ipxact:description>RX-FIFO empty</ipxact:description>
                <ipxact:value>0</ipxact:value>
              </ipxact:enumeratedValue>
              <ipxact:enumeratedValue usage="read">
                <ipxact:name>NOT_EMPTY</ipxact:name>
                <ipxact:description>RX-FIFO not empty.</ipxact:description>
                <ipxact:value>1</ipxact:value>
              </ipxact:enumeratedValue>
            </ipxact:enumeratedValues>
          </ipxact:field>
          <ipxact:field>
            <ipxact:name>RXFIFO_OVFL</ipxact:name>
            <ipxact:description>RX-FIFO Overflow. This interrupt capable status flag indicates  an
overflow error and associated interrupt status before the enable stage. The flag can only be
explicitly cleared by writing 1 to the flag.
            </ipxact:description>
            <ipxact:bitOffset>1</ipxact:bitOffset>
            <ipxact:bitWidth>1</ipxact:bitWidth>
            <ipxact:resets>
              <ipxact:reset>
                <ipxact:value>'h0</ipxact:value>
                <ipxact:mask>'h1</ipxact:mask>
              </ipxact:reset>
            </ipxact:resets>
            <ipxact:fieldAccessPolicies>
              <ipxact:fieldAccessPolicy>
                <ipxact:access>read-write</ipxact:access>
                <ipxact:modifiedWriteValue>oneToClear</ipxact:modifiedWriteValue>
              </ipxact:fieldAccessPolicy>
            </ipxact:fieldAccessPolicies>
            <ipxact:enumeratedValues>
              <ipxact:enumeratedValue usage="read">
                <ipxact:name>NO_OVFL</ipxact:name>
                <ipxact:description>no overflow</ipxact:description>
                <ipxact:value>0</ipxact:value>
              </ipxact:enumeratedValue>
              <ipxact:enumeratedValue usage="read">
                <ipxact:name>OVFL</ipxact:name>
                <ipxact:description>overflow error</ipxact:description>
                <ipxact:value>1</ipxact:value>
              </ipxact:enumeratedValue>
              <ipxact:enumeratedValue usage="write">
                <ipxact:name>NO_EFFECT</ipxact:name>
                <ipxact:description>no effect</ipxact:description>
                <ipxact:value>0</ipxact:value>
              </ipxact:enumeratedValue>
              <ipxact:enumeratedValue usage="write">
                <ipxact:name>CLEAR</ipxact:name>
                <ipxact:description>clear flag</ipxact:description>
                <ipxact:value>1</ipxact:value>
              </ipxact:enumeratedValue>
            </ipxact:enumeratedValues>
          </ipxact:field>
          <ipxact:field>
            <ipxact:name>RXSTATE</ipxact:name>
            <ipxact:description>RX state. This field indicates the state of the receiver.
            </ipxact:description>
            <ipxact:bitOffset>2</ipxact:bitOffset>
            <ipxact:bitWidth>2</ipxact:bitWidth>
            <ipxact:resets>
              <ipxact:reset>
                <ipxact:value>'h0</ipxact:value>
                <ipxact:mask>'h3</ipxact:mask>
              </ipxact:reset>
            </ipxact:resets>
            <ipxact:fieldAccessPolicies>
              <ipxact:fieldAccessPolicy>
                <ipxact:access>read-only</ipxact:access>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
            </ipxact:fieldAccessPolicy>
          </ipxact:fieldAccessPolicies>
          <ipxact:enumeratedValues>
            <ipxact:enumeratedValue usage="read">
              <ipxact:name>IDLE</ipxact:name>
              <ipxact:description>Idle state</ipxact:description>
              <ipxact:value>0</ipxact:value>
            </ipxact:enumeratedValue>
            <ipxact:enumeratedValue usage="read">
              <ipxact:name>BUSY</ipxact:name>
              <ipxact:description>Busy state</ipxact:description>
              <ipxact:value>1</ipxact:value>
            </ipxact:enumeratedValue>
            <ipxact:enumeratedValue usage="read">
              <ipxact:name>SYNC</ipxact:name>
              <ipxact:description>Sync state</ipxact:description>
              <ipxact:value>2</ipxact:value>
            </ipxact:enumeratedValue>
          </ipxact:enumeratedValues>
        </ipxact:field>
        <ipxact:field>
          <ipxact:name>reserved0</ipxact:name>
          <ipxact:displayName>RESERVED</ipxact:displayName>
          <ipxact:description>reserved. Read value undefined. Should be written 0.
          </ipxact:description>
          <ipxact:bitOffset>4</ipxact:bitOffset>
          <ipxact:bitWidth>28</ipxact:bitWidth>
          <ipxact:resets>
            <ipxact:reset>
              <ipxact:value>'h0</ipxact:value>
              <ipxact:mask>'h0</ipxact:mask>
            </ipxact:reset>
          </ipxact:resets>
          <ipxact:fieldAccessPolicies>
            <ipxact:fieldAccessPolicy>
              <ipxact:access>read-only</ipxact:access>
              <ipxact:reserved>true</ipxact:reserved>
            </ipxact:fieldAccessPolicy>
          </ipxact:fieldAccessPolicies>
        </ipxact:field>
      </ipxact:register>
    </ipxact:addressBlock>
    <ipxact:addressUnitBits>8</ipxact:addressUnitBits>
  </ipxact:memoryMap>
 </ipxact:memoryMaps>
</ipxact:component>
```

The **busInterface target** element contains an additional **memoryMapRef** element with an attribute **memoryMapRef** that references a **memoryMap** element by name (see Example 39). This indicates that the referenced **memoryMap** is addressable through the referencing **busInterface**.

Example 39 Component busInterface with memoryMapRef

```
    <ipxact:busInterface>
      <ipxact:name>Target</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="APB3" version="1.0"/>
      <ipxact:target>
        <ipxact:memoryMapRef memoryMapRef="RegisterMap"/>
      </ipxact:target>
    </ipxact:busInterface>
```

The referenced **memoryMap** is defined in the **component memoryMaps** container element (see Example 40). A **memoryMap** has a **name** to identify the element. Furthermore, a **memoryMap** can contain different types of memory map elements: **addressBlock**, **bank**, and **subspaceMap**. The **addressBlock** element is explained in the next paragraph. The other two elements are not discussed. Finally, a **memoryMap** has **addressUnitBits** that describes the number of bits of an address increment between two consecutive addressable units in the **memoryMap**. If an **addressUnitBits** element is not described, then its value defaults to 8 indicating a byte addressable **memoryMap**.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 40 Component memoryMaps

```
<ipxact:memoryMaps>
  <ipxact:memoryMap>
    <ipxact:name>RegisterMap</ipxact:name>
     ...
    <ipxact:addressUnitBits>8</ipxact:addressUnitBits>
  </ipxact:memoryMap>
</ipxact:memoryMaps>
```

An **addressBlock** describes a single, contiguous block of memory in a **memoryMap** (see Example 41). It has a **name** to identify the element. Furthermore, an **addressBlock** has a **baseAddress**, a **range**, and a **width** to describe the location of the **addressBlock** in the **memoryMap**. A **baseAddress** describes the starting address of the **addressBlock** expressed in **addressUnitBits** from the containing **memoryMap**. A **range** describes the number of addressable units in the **addressBlock**. A **width** describes the maximum number of bits that can be accessed in a single transfer into the **addressBlock**. Finally, an **addressBlock** can have an **accessPolicies** element, that contain multiple **accessPolicy** and **access** elements, and multiple **register** and **registerFile** elements. An **access** element can take the values read-write, read-only, write-only, read-writeOnce, writeOnce, and no-access. The **access** indicates the accessibility of data in the **addressBlock**. If an **access** element is not described, then its value defaults to read-write. The **register** and **registerFile** elements are discussed in the next paragraphs.

Example 41 Component addressBlock

```
<ipxact:addressBlock>
  <ipxact:name>ControlSpace</ipxact:name>
  <ipxact:baseAddress>'h0</ipxact:baseAddress>
  <ipxact:range>'h1000</ipxact:range>
  <ipxact:width>32</ipxact:width>
  <ipxact:accessPolicies>
    <ipxact:accessPolicy>
      <ipxact:access>read-write</ipxact:access>
     </ipxact:accessPolicy>
  </ipxact:accessPolicies>
  ...
</ipxact:addressBlock>
```

A **register** element describes the software interface to a register (see Example 42). It has a **name** to identify the element. It can have a **description** to provide a human-readable description. A **register** can have an **array** element describing the dimensions and stride of the register. If **array** is not described, then the register is a scalar. Furthermore, a **register** has an **addressOffset** that describes the location of the **register** expressed in **addressUnitBits** as offset to the starting address of the containing **addressBlock** or the containing **registerFile**. The **size** of a **register** describes the number of bits in the register. A register **size** cannot exceed the **width** of a containing **addressBlock**. A **register** can also contain an **accessPolicies** element describing the accessibility of data in the register similar to access of an **addressBlock**. A **register** has **field** elements that are discussed in the next paragraph.

Example 42 Component register

```
        <ipxact:register>
          <ipxact:name>STAT</ipxact:name>
          <ipxact:description>Status register. Collection of Status flags including interrupt status
before enabling
          </ipxact:description>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:size>32</ipxact:size>
          ...
        </ipxact:register>
```

A **field** element describes one or more bits of a register (see Example 43). A **field** has a **name** to identify the element. A **field** can have an **array** element describing the dimensions and stride of the field. If **array** is not described, then the field is a scalar. A field can have a **description** to provide a human-readable description. The **bitOffset** describes the starting bit of the field expressed in number of bits. The **bitWidth** describes the

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

number of bits in the field. A field can have a **resets** element containing multiple **reset** elements. Each **reset** has a **value** describing the reset value of the bits in the field and a **mask** describing which bits in the field have a defined reset value. In a **mask**, a bit value of 1 describes that the bit reset value is defined and a bit value of 0 describes that the bit reset value is not defined. A **reset** can have a **resetTypeRef** attribute describing the type of reset. If **resetTypeRef** is not described, then its value defaults to HARD which is a pre-defined reset type. A **field** can also contain a **fieldAccessPolicies** element describing the accessibility of data in the field similar to access of a **register**. Finally, a **field** can contain **enumeratedValues** which are explained in the next paragraph.

Example 43 Component register field

```
          <ipxact:field>
            <ipxact:name>RXFIFO_NE</ipxact:name>
            <ipxact:description>RX-FIFO Not Empty. This interrupt capable status flag indicates  the 
RX-FIFO status and associated interrupt status before the enable stage. The flag can only be 
implicitly cleared by reading the RXFIFO_DAT register
            </ipxact:description>
            <ipxact:bitOffset>0</ipxact:bitOffset>
            <ipxact:bitWidth>1</ipxact:bitWidth>
            <ipxact:resets>
              <ipxact:reset>
                <ipxact:value>'h0</ipxact:value>
                <ipxact:mask>'h1</ipxact:mask>
              </ipxact:reset>
            </ipxact:resets>
            <ipxact:fieldAccessPolicies>
              <ipxact:fieldAccessPolicy>
                <ipxact:access>read-only</ipxact:access>
              </ipxact:fieldAccessPolicy>
            </ipxact:fieldAccessPolicies>
          </ipxact:field>
```

The **enumeratedValues** element is a container element for **enumeratedValue** elements (see Example 44). An **enumeratedValue** describes a value of the containing **field**. An **enumeratedValue** can have an attribute **usage** describing the usage of the **enumeratedValue** which can take the values read, write, and read-write. Furthermore, an **enumeratedValue** has a **name** to identify the element. It can have a **description** to provide a human-readable description. An **enumeratedValue value** describes the value.

Example 44 Component register field enumeratedValues

```
            <ipxact:enumeratedValues>
              <ipxact:enumeratedValue usage="read">
                <ipxact:name>EMPTY</ipxact:name>
                <ipxact:description>RX-FIFO empty</ipxact:description>
                <ipxact:value>0</ipxact:value>
              </ipxact:enumeratedValue>
              <ipxact:enumeratedValue usage="read">
                <ipxact:name>NOT_EMPTY</ipxact:name>
                <ipxact:description>RX-FIFO not empty.</ipxact:description>
                <ipxact:value>1</ipxact:value>
              </ipxact:enumeratedValue>
            </ipxact:enumeratedValues>
```

Finally, an **addressBlock** can contain **registerFile** elements which are used to group **register** elements (see Example 45). A **registerFile** has a **name** to identify the element. A **registerFile** can have an **array** element describing the dimensions and stride of the register file. If **array** is not described, then the register file is a scalar. A register file can have a **description** to provide a human-readable description. A **registerFile addressOffset** describes the location of the **registerFile** expressed in **addressUnitBits** as offset to the starting address of the containing **addressBlock** or the containing **registerFile**. A **registerFile range** describes the number of addressable units in the **registerFile** similar to **addressBlock range**. A **registerFile** can contain **register** and **registerFile** elements. Hence, **registerFile** elements can be nested arbitrarily deep.

Example 45 Component registerFile

```
        <ipxact:registerFile>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
      <ipxact:name>CHANNEL</ipxact:name>
      <ipxact:description>Channel Descriptor Registers</ipxact:description>
      <ipxact:array>
        <ipxact:dim>4</ipxact:dim>
      </ipxact:array>
      <ipxact:addressOffset>'h200</ipxact:addressOffset>
      <ipxact:range>'h10</ipxact:range>
       ...
    </ipxact:registerFile>
```

3.1.6 Component Address Spaces and Bus Interface Bridges

The **component memoryMaps** introduced in the previous section are used to determine global memory maps for **componentInstances** of a **design**. Global memory maps are not described explicitly. Rather, they are computed from the design topology by positioning a **memoryMap** in **addressSpaces**. We use the **component** description for a CPU shown in Example 46 to illustrate the concept of address spaces.

Example 46 Component with initiator busInterface and addressSpace

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>cpu</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:description>Central Processing Unit.</ipxact:description>
  <ipxact:busInterfaces>
    <ipxact:busInterface>
      <ipxact:name>AHB</ipxact:name>
      <ipxact:description>AHB interface provides system access.</ipxact:description>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteInitiator" version="1.0"/>
      <ipxact:initiator>
        <ipxact:addressSpaceRef addressSpaceRef="AS">
          <ipxact:baseAddress>'h0</ipxact:baseAddress>
        </ipxact:addressSpaceRef>
      </ipxact:initiator>
    </ipxact:busInterface>
  </ipxact:busInterfaces>
  <ipxact:addressSpaces>
    <ipxact:addressSpace>
      <ipxact:name>AS</ipxact:name>
      <ipxact:range>'h100000000</ipxact:range>
      <ipxact:width>32</ipxact:width>
      <ipxact:segments>
        &lt;ipxact:segment&gt;
          <ipxact:name>Code</ipxact:name>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:range>'h20000000</ipxact:range>
        &lt;/ipxact:segment&gt;
        &lt;ipxact:segment&gt;
          <ipxact:name>SRAM</ipxact:name>
          <ipxact:addressOffset>'h20000000</ipxact:addressOffset>
          <ipxact:range>'h20000000</ipxact:range>
        &lt;/ipxact:segment&gt;
        &lt;ipxact:segment&gt;
          <ipxact:name>Peripheral</ipxact:name>
          <ipxact:addressOffset>'h40000000</ipxact:addressOffset>
          <ipxact:range>'h20000000</ipxact:range>
        &lt;/ipxact:segment&gt;
        &lt;ipxact:segment&gt;
          <ipxact:name>ExternalRAM</ipxact:name>
          <ipxact:addressOffset>'h60000000</ipxact:addressOffset>
          <ipxact:range>'h40000000</ipxact:range>
        &lt;/ipxact:segment&gt;
        &lt;ipxact:segment&gt;
          <ipxact:name>External</ipxact:name>
          <ipxact:addressOffset>'hA0000000</ipxact:addressOffset>
          <ipxact:range>'h40000000</ipxact:range>
        &lt;/ipxact:segment&gt;
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
        &lt;ipxact:segment&gt;
          <ipxact:name>Private</ipxact:name>
          <ipxact:addressOffset>'hE0000000</ipxact:addressOffset>
          <ipxact:range>'h100000</ipxact:range>
        &lt;/ipxact:segment&gt;
        &lt;ipxact:segment&gt;
          <ipxact:name>Vendor</ipxact:name>
          <ipxact:addressOffset>'hE0100000</ipxact:addressOffset>
          <ipxact:range>'h1FF00000</ipxact:range>
        &lt;/ipxact:segment&gt;
      </ipxact:segments>
      <ipxact:addressUnitBits>8</ipxact:addressUnitBits>
      <ipxact:localMemoryMap>
        <ipxact:name>PPB</ipxact:name>
        <ipxact:addressBlock>
          <ipxact:name>PrivateInt</ipxact:name>
          <ipxact:baseAddress>'hE0000000</ipxact:baseAddress>
          <ipxact:range>'h40000</ipxact:range>
          <ipxact:width>32</ipxact:width>
          <ipxact:accessPolicies>
            <ipxact:accessPolicy>
               <ipxact:access>read-write</ipxact:access>
            </ipxact:accessPolicy>
          </ipxact:accessPolicies>
        </ipxact:addressBlock>
        <ipxact:addressBlock>
          <ipxact:name>PrivateExt</ipxact:name>
          <ipxact:baseAddress>'hE0040000</ipxact:baseAddress>
          <ipxact:range>'hC0000</ipxact:range>
          <ipxact:width>32</ipxact:width>
          <ipxact:accessPolicies>
            <ipxact:accessPolicy>
               <ipxact:access>read-write</ipxact:access>
            </ipxact:accessPolicy>
          </ipxact:accessPolicies>
        </ipxact:addressBlock>
      </ipxact:localMemoryMap>
    </ipxact:addressSpace>
  </ipxact:addressSpaces>
</ipxact:component>
```

This **busInterface** references an addressable bus definition to make the link to a **component addressSpace** (see Example 47). The **busDefinition** has the four elements **vendor**, **library**, **name**, and **version** identifier with values accellera.org, amba3, AHBLiteInitiator, and 1.0, respectively. If a **component busInterface** in **initiator** mode references an addressable bus interface, then the bus interface must reference a **component addressSpace** by name in the **initiator** element. This reference is described in the attribute **addressSpaceRef** of the element **addressSpaceRef**. This element also contains a **baseAddress** describing the start address of the address space for the containing **busInterface**.

Example 47 Component busInterface addressSpaceRef

```
    <ipxact:busInterface>
      <ipxact:name>AHB</ipxact:name>
      <ipxact:description>AHB interface provides system access.</ipxact:description>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteInitiator" version="1.0"/>
      <ipxact:initiator>
        <ipxact:addressSpaceRef addressSpaceRef="AS">
          <ipxact:baseAddress>'h0</ipxact:baseAddress>
        </ipxact:addressSpaceRef>
      </ipxact:initiator>
    </ipxact:busInterface>
```

The **component addressSpaces** element is a container element for **addressSpace** elements (see Example 48). An **addressSpace** has a name to identify the element. Furthermore, it has **addressUnitBits**, **range** and **width** elements. The **addressUnitBits** element describes the number of bits of an address increment between two consecutive addressable units in the **addressSpace**. If **addressUnitBits** is not described, then its value defaults to 8, indicating a byte-addressable **addressSpace**. The **range** describes the number of addressable units of the **addressSpace**. The **width** describes the maximum number of bits that can be accessed in a single transfer in

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

the **addressSpace**. An **addressSpace** can have **segments** and a **localMemoryMap** which are discussed in the next paragraphs.

Example 48 Component addressSpace

```
    <ipxact:addressSpace>
      <ipxact:name>AS</ipxact:name>
      <ipxact:range>'h100000000</ipxact:range>
      <ipxact:width>32</ipxact:width>       ...
      <ipxact:addressUnitBits>8</ipxact:addressUnitBits>       ...
    </ipxact:addressSpace>
```

An **addressSpace** segments element is a container element for multiple segment element (see Example 49). Each segment describes a portion of the **addressSpace**. A segment has a name to identify the element. It has an **addressOffset** describing the address offset of the segment with respect to the start address of the **addressSpace** expressed in addressing units. Furthermore, it has a range describing the number of addressable units of the segment.

Example 49 Component addressSpace segment

```
        &lt;ipxact:segment&gt;
          <ipxact:name>Code</ipxact:name>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:range>'h20000000</ipxact:range>
        &lt;/ipxact:segment&gt;
```

An **addressSpace localMemoryMap** describes a memory map that is visible only in the containing **addressSpace** (see Example 50). A **localMemoryMap** has a **name** to identify the element. Furthermore, it can contain **addressBlock** and **bank** elements similar to a **component memoryMap**.

Example 50 Component addressSpace localMemoryMap

```
      <ipxact:localMemoryMap>
        <ipxact:name>PPB</ipxact:name>
        ...
      </ipxact:localMemoryMap>
```

To illustrate the concept of **busInterface transparentBridge** elements, a second component is used describing an AHB bus fabric. This component has two target bus interfaces to connect to the CPU and DMA components and four initiator bus interfaces to connect to the ROM, RAM, DMA, and APB peripherals as shown in Example 51.

Example 51 Component busahb

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>busahb</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:description>AHB interconnect.</ipxact:description>
  <ipxact:busInterfaces>
    <ipxact:busInterface>
      <ipxact:name>toCPU</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteInitiator" version="1.0"/>
      <ipxact:target>
        <ipxact:transparentBridge initiatorRef="toROM"/>
        <ipxact:transparentBridge initiatorRef="toRAM"/>
        <ipxact:transparentBridge initiatorRef="toDMA_S"/>
        <ipxact:transparentBridge initiatorRef="toAPB"/>
      </ipxact:target>
    </ipxact:busInterface>
    <ipxact:busInterface>
      <ipxact:name>toDMA_M</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteInitiator" version="1.0"/>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
      <ipxact:target>
        <ipxact:transparentBridge initiatorRef="toRAM"/>
        <ipxact:transparentBridge initiatorRef="toAPB"/>
      </ipxact:target>
    </ipxact:busInterface>
    <ipxact:busInterface>
      <ipxact:name>toROM</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteTarget" version="1.0"/>
      <ipxact:initiator>
        <ipxact:addressSpaceRef addressSpaceRef="AS_ROM">
          <ipxact:baseAddress>'h0</ipxact:baseAddress>
        </ipxact:addressSpaceRef>
      </ipxact:initiator>
    </ipxact:busInterface>
    <ipxact:busInterface>
      <ipxact:name>toRAM</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteTarget" version="1.0"/>
      <ipxact:initiator>
        <ipxact:addressSpaceRef addressSpaceRef="AS_RAM">
          <ipxact:baseAddress>'h20000000</ipxact:baseAddress>
        </ipxact:addressSpaceRef>
      </ipxact:initiator>
    </ipxact:busInterface>
    <ipxact:busInterface>
      <ipxact:name>toDMA_S</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteTarget" version="1.0"/>
      <ipxact:initiator>
        <ipxact:addressSpaceRef addressSpaceRef="AS_DMA">
          <ipxact:baseAddress>'h40000000</ipxact:baseAddress>
        </ipxact:addressSpaceRef>
      </ipxact:initiator>
    </ipxact:busInterface>
    <ipxact:busInterface>
      <ipxact:name>toAPB</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteTarget" version="1.0"/>
      <ipxact:initiator>
        <ipxact:addressSpaceRef addressSpaceRef="AS_APB">
          <ipxact:baseAddress>'h40001000</ipxact:baseAddress>
        </ipxact:addressSpaceRef>
      </ipxact:initiator>
    </ipxact:busInterface>
  </ipxact:busInterfaces>
  <ipxact:addressSpaces>
    <ipxact:addressSpace>
      <ipxact:name>AS_ROM</ipxact:name>
      <ipxact:range>'h20000000</ipxact:range>
      <ipxact:width>32</ipxact:width>
    </ipxact:addressSpace>
    <ipxact:addressSpace>
      <ipxact:name>AS_RAM</ipxact:name>
      <ipxact:range>'h20000000</ipxact:range>
      <ipxact:width>32</ipxact:width>
    </ipxact:addressSpace>
    <ipxact:addressSpace>
      <ipxact:name>AS_DMA</ipxact:name>
      <ipxact:range>'h1000</ipxact:range>
      <ipxact:width>32</ipxact:width>
    </ipxact:addressSpace>
    <ipxact:addressSpace>
      <ipxact:name>AS_APB</ipxact:name>
      <ipxact:range>'h1000</ipxact:range>
      <ipxact:width>32</ipxact:width>
    </ipxact:addressSpace>
  </ipxact:addressSpaces>
</ipxact:component>
```

A **transparentBridge** is part of a **busInterface target** element (see Example 52). An addressable target must contain one or more **transparentBridge** elements or a **memoryMapRef** element. The **memoryMapRef** element describes the fact that a **component MemoryMap** can be accessed through the referencing **busInterface**. It has been discussed in Section 3.1.5. A **transparentBridge** element describes the fact that a **component busInterface** bridges to another **busInterface** element by referencing the **busInterface name**. The referenced **busInterface** must be an **initiator**. In the example, the **transparentBridge** elements describe

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

that a CPU connected to bus interface toCPU can access ROM, RAM, DMA, and APB peripherals. However, a DMA controller connected to bus interface toDMA_M can only access RAM and APB peripherals. A **transparentBridge** is called *transparent* to indicate that addresses entering at a target interface exit at an initiator interface without any modification.

Example 52 Component busInterface transparentBridge

```
<ipxact:busInterface>
  <ipxact:name>toCPU</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteInitiator" version="1.0"/>
  <ipxact:target>
    <ipxact:transparentBridge initiatorRef="toROM"/>
    <ipxact:transparentBridge initiatorRef="toRAM"/>
    <ipxact:transparentBridge initiatorRef="toDMA_S"/>
    <ipxact:transparentBridge initiatorRef="toAPB"/>
  </ipxact:target>
</ipxact:busInterface>
```

The decoding of addresses entering at target interfaces is described by the **busInterface initiator** elements and **addressSpace** elements. In the example (see Example 53), **busInterface** toRAM references **addressSpace** AS_RAM and locates AS_RAM at offset 'h20000000 in the CPU **addressSpace** AS. More specifically, AS_RAM is located in the **addressSpace segment** SRAM. The **range** of AS_RAM is 'h20000000, hence, AS_RAM occupies locations 'h20000000 to 'h3FFFFFFF which fits in **segment** SRAM. All addresses entering at the target interfaces in the range 'h20000000 to 'h3FFFFFFF are forwarded to **busInterface** toRAM since the start address of that range is described by its **busInterface initiator baseAddress** with value 'h20000000 and the end address of that range is described by that start address plus the referenced **addressSpace range** with the value 'h20000000 minus one addressable unit.

Example 53 Component busInterface address ranges

```
<ipxact:busInterface>
  <ipxact:name>toRAM</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteTarget" version="1.0"/>
  <ipxact:initiator>
    <ipxact:addressSpaceRef addressSpaceRef="AS_RAM">
      <ipxact:baseAddress>'h20000000</ipxact:baseAddress>
    </ipxact:addressSpaceRef>
  </ipxact:initiator>
</ipxact:busInterface>

<ipxact:addressSpace>
  <ipxact:name>AS_RAM</ipxact:name>
  <ipxact:range>'h20000000</ipxact:range>
  <ipxact:width>32</ipxact:width>
</ipxact:addressSpace>
```

3.1.7 Type definitions

The **typeDefinitions** top-level element has been introduced in the IEEE 1685-2022 standard with the objective to support reusability and maintainability of memory-related meta-data. It allows users to create definitions once and reuse their instances wherever required. It also supports configurability of these definitions through parameters. Effectively, a **typeDefinitions** element is the central placeholder which allows users to refer to any of these definitions through **externalTypeDefinitions** and **typeDefinitionsRef** elements in components.

A **typeDefinitions** element lists definitions of field access policies, enumerations, fields, registers, register files, address blocks, banks, memory maps, and memory remaps. Another **typeDefinitions** element can be referenced in a **typeDefinitions** using an **externalTypeDefinitions** element allowing users to achieve nesting of **typeDefinitions**.

For further explanation, consider an example with a **memoryMapDefinition** element which is described inside the **memoryMapDefinitions** element which, in turn, is described inside a **typeDefinitions** element.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 54 describes one **memoryMapDefinition** with an **addressBlock** element having a single **register** and a **field**.

Example 54 memoryMapDefinitions

```
<ipxact:memoryMapDefinitions>
  <ipxact:memoryMapDefinition>
  <ipxact:name>top_chip_mem</ipxact:name>
    <ipxact:addressBlock>
      <ipxact:name>ControlSpace</ipxact:name>
      <ipxact:baseAddress>'h0</ipxact:baseAddress>
      <ipxact:range>'h100</ipxact:range>
      <ipxact:width>32</ipxact:width>
      <ipxact:register>
        <ipxact:name>STAT</ipxact:name>
        <ipxact:addressOffset>'h0</ipxact:addressOffset>
        <ipxact:size>32</ipxact:size>
        <ipxact:field>
          <ipxact:name>RXFIFO_NE</ipxact:name>
          <ipxact:bitOffset>'h0</ipxact:bitOffset>
          <ipxact:bitWidth>1</ipxact:bitWidth>
        </ipxact:field>
      </ipxact:register>
    </ipxact:addressBlock>
    <ipxact:addressUnitBits>AUB</ipxact:addressUnitBits>
  </ipxact:memoryMapDefinition>
</ipxact:memoryMapDefinitions>
```

As shown above, the **memoryMapDefinitions** element contains one or more **memoryMapDefinition** elements. A **memoryMapDefinition** element follows the schema details of **memoryMap**, hence, it has an **addressBlock** with **register** and **field** elements inside. In the example, we have not specified the **addressUnitBits** as a constant number, rather we have specified it as an expression using the **parameterId** attribute of a parameter described later inside **typeDefinitions**. Let's assume this **parameterId** value to be AUB for now.

Next, we encapsulate the above **memoryMapDefinitions** in a **typeDefinitions** in Example 55.

Example 55 typeDefinitions

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:typeDefinitions xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>TypeDefCentral</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:displayName>Central Type Definitions</ipxact:displayName>
  <ipxact:shortDescription>Example typeDefinitions</ipxact:shortDescription>
  <ipxact:description>Example typeDefinitions used in IP-XACT standard 1685-2022.</ipxact:description>
  <ipxact:memoryMapDefinitions>
    <ipxact:memoryMapDefinition>
      <ipxact:name>top_chip_mem</ipxact:name>
      <ipxact:addressBlock>
        <ipxact:name>ControlSpace</ipxact:name>
        <ipxact:baseAddress>'h0</ipxact:baseAddress>
        <ipxact:range>'h100</ipxact:range>
        <ipxact:width>32</ipxact:width>
        <ipxact:register>
          <ipxact:name>STAT</ipxact:name>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:size>32</ipxact:size>
          <ipxact:field>
           <ipxact:name>RXFIFO_NE</ipxact:name>
           <ipxact:bitOffset>'h0</ipxact:bitOffset>
           <ipxact:bitWidth>32</ipxact:bitWidth>
          </ipxact:field>
        </ipxact:register>
      </ipxact:addressBlock>
      <ipxact:addressUnitBits>AUB</ipxact:addressUnitBits>
    </ipxact:memoryMapDefinition>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
  </ipxact:memoryMapDefinitions>
  <ipxact:parameters>
    <ipxact:parameter parameterId="AUB" prompt="Address Unit Bits" type="shortint" resolve="user">
      <ipxact:name>AUB</ipxact:name>
      <ipxact:value>8</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
</ipxact:typeDefinitions>
```

In addition to the **memoryMapDefinitions** element there is a parameter described inside **typeDefinitions** with **name** AUB and **parameterId** AUB. The parameters element allows users to specify one or more parameters and they can be used inside definitions to achieve configurability at the time of referring to the **typeDefinitions** inside component. Now, the **typeDefinitions** element described in Example 55 is ready to be used inside component elements. As the **typeDefinitions** named TypeDefCentral is described outside the component, it can be used inside the component using **externalTypeDefinitions** element and **typeDefinitionsRef** element using the VLNV of the concerned **typeDefinitions** as attributes. Example 56 describes a **component**.

Example 56 Component with externalTypeDefinitions

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>comp_a</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:typeDefinitions>
    <ipxact:externalTypeDefinitions>
      <ipxact:name>centralTypes</ipxact:name>
      <ipxact:typeDefinitionsRef vendor="accellera.org" library="ug" name="TypeDefCentral"
        version="1.0">
        <ipxact:configurableElementValues>
          <ipxact:configurableElementValue referenceId="AUB">16
          </ipxact:configurableElementValue>
        </ipxact:configurableElementValues>
      </ipxact:typeDefinitionsRef>
    </ipxact:externalTypeDefinitions>
  </ipxact:typeDefinitions>
  <ipxact:memoryMaps>
    <ipxact:memoryMap>
      <ipxact:name>chip_mem</ipxact:name>
      <ipxact:memoryMapDefinitionRef typeDefinitions="centralTypes">top_chip_mem
      </ipxact:memoryMapDefinitionRef>
    </ipxact:memoryMap>
  </ipxact:memoryMaps>
</ipxact:component>
```

Example 56 illustrates how **typeDefinitions** are referenced inside a **component**. The referenced **typeDefinitions** enables the use of the **memoryMapDefinition**, described in the **typeDefinitions** named TypeDefCentral, inside a **memoryMap** element. The **memoryMapDefinitionRef** element is used for this purpose along with **externalTypeDefinitions name** as an attribute of this element and name of the **memoryMapDefinition** as the value of this element.

Similarly, other type definitions can also be described inside **typeDefinitions** and then used inside components.

3.2 Advanced Topics

This section first explains the remaining IP-XACT document types **abstractor**, **generator chain**, and **catalog**. Subsequently, more details of conditional elements, parameter passing, and the Tight Generator Interface are presented.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

3.2.1 Advanced Elements

3.2.1.1 Abstractor

Abstractors are elements that describe IP blocks that implement conversion between two different abstraction definitions. They are similar to components because they describe IP blocks. They are different from components because their description is tailored to the function of abstraction conversion. Abstractors are instantiated in design configurations, whereas components are instantiated in designs. The reason for this is that design configurations describe which views are used for which component instances, hence, required abstraction conversions between component instances is specific to design configurations.

The explanation of abstractors is organized as follows. First, the concept of components with multiple views at different levels of abstraction is described. Then those components are instantiated in a design and multiple design configurations are used to configure that design to generate multiple netlists. Finally, the details of abstractors and the use of abstractor instances in design configurations are described.

Figure 3 illustrates a component with an RTL view and a TLM view and a bus interface that contains two view-specific port maps. The component bus interface references a bus definition named MyBus and two abstraction definitions named MyBusAbs_rtl and MyBusAbs_tlm. The references to those abstraction definitions and the associated port maps depend on a view reference, meaning that the abstraction definition reference and the port map must be applied on a given component instance if and only if the referenced view is configured through a design configuration view configuration for that component instance. Also, the component ports are made view-specific through a view reference in a port. The blue and yellow parts in Figure 3 indicate the descriptions of the component that are specific for the RTL and TLM view, respectively. Example 57 shows the IP-XACT description of that **component**.

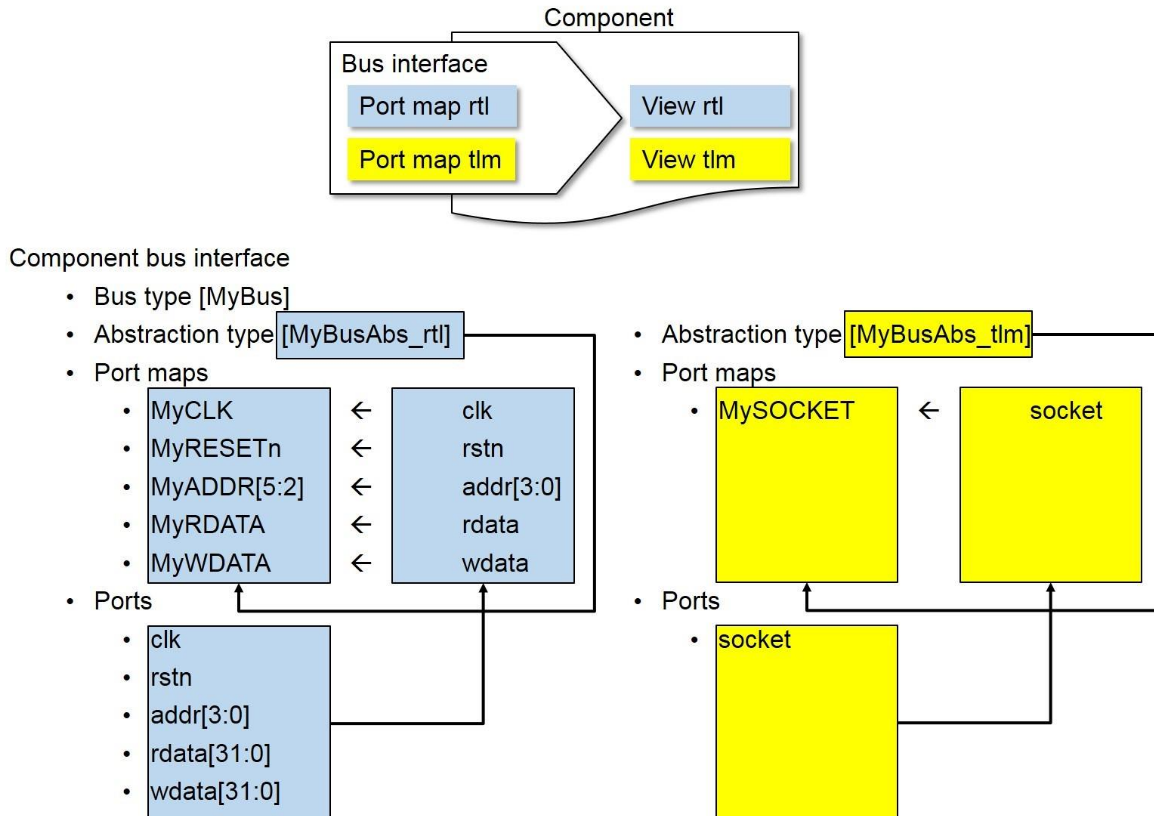

Figure 3 Graphical representation of a component with RTL and TLM view and view-specific port maps

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 57 Component with RTL and TLM view and view-specific port maps

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>comp</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:busInterfaces>
    <ipxact:busInterface>
      <ipxact:name>busif</ipxact:name>
      <ipxact:busType vendor="accellera.org" library="ug" name="MyBus" version="1.0"/>
      <ipxact:abstractionTypes>
        <ipxact:abstractionType>
          <ipxact:viewRef>rtl</ipxact:viewRef>
          <ipxact:abstractionRef vendor="accellera.org" library="ug" name="MyBusAbs_rtl"
            version="1.0"/>
          <ipxact:portMaps>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>MyCLK</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>clk</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>MyRESETn</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>rstn</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>MyADDR</ipxact:name>
                <ipxact:range>
                  <ipxact:left>5</ipxact:left>
                  <ipxact:right>2</ipxact:right>
                </ipxact:range>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>addr</ipxact:name>
                <ipxact:partSelect>
                  <ipxact:range>
                    <ipxact:left>3</ipxact:left>
                    <ipxact:right>0</ipxact:right>
                  </ipxact:range>
                </ipxact:partSelect>
              </ipxact:physicalPort>
            </ipxact:portMap>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>MyRDATA</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>rdata</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
            <ipxact:portMap>
              <ipxact:logicalPort>
                <ipxact:name>MyWDATA</ipxact:name>
              </ipxact:logicalPort>
              <ipxact:physicalPort>
                <ipxact:name>wdata</ipxact:name>
              </ipxact:physicalPort>
            </ipxact:portMap>
          </ipxact:portMaps>
        </ipxact:abstractionType>
        <ipxact:abstractionType>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
        <ipxact:viewRef>tlm</ipxact:viewRef>
        <ipxact:abstractionRef vendor="accellera.org" library="ug" name="MyBusAbs_tlm"
          version="1.0"/>
        <ipxact:portMaps>
          <ipxact:portMap>
            <ipxact:logicalPort>
              <ipxact:name>mySOCKET</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
              <ipxact:name>socket</ipxact:name>
            </ipxact:physicalPort>
          </ipxact:portMap>
        </ipxact:portMaps>
      </ipxact:abstractionType>
    </ipxact:abstractionTypes>
    <ipxact:initiator/>
  </ipxact:busInterface>
</ipxact:busInterfaces>
<ipxact:model>
  <ipxact:views>
    <ipxact:view>
      <ipxact:name>rtl</ipxact:name>
    </ipxact:view>
    <ipxact:view>
      <ipxact:name>tlm</ipxact:name>
    </ipxact:view>
  </ipxact:views>
  <ipxact:ports>
    <ipxact:port>
      <ipxact:name>clk</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:wireTypeDefs>
          <ipxact:wireTypeDef>
            <ipxact:viewRef>rtl</ipxact:viewRef>
          </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
      </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
      <ipxact:name>rstn</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:wireTypeDefs>
          <ipxact:wireTypeDef>
            <ipxact:viewRef>rtl</ipxact:viewRef>
          </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
      </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
      <ipxact:name>addr</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>out</ipxact:direction>
        <ipxact:vectors>
          <ipxact:vector>
            <ipxact:left>3</ipxact:left>
            <ipxact:right>0</ipxact:right>
          </ipxact:vector>
        </ipxact:vectors>
        <ipxact:wireTypeDefs>
          <ipxact:wireTypeDef>
            <ipxact:viewRef>rtl</ipxact:viewRef>
          </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
      </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
      <ipxact:name>rdata</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:vectors>
          <ipxact:vector>
            <ipxact:left>31</ipxact:left>
            <ipxact:right>0</ipxact:right>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
          </ipxact:vector>
        </ipxact:vectors>
        <ipxact:wireTypeDefs>
          <ipxact:wireTypeDef>
            <ipxact:viewRef>rtl</ipxact:viewRef>
          </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
      </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
      <ipxact:name>wdata</ipxact:name>
      <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:vectors>
          <ipxact:vector>
            <ipxact:left>31</ipxact:left>
            <ipxact:right>0</ipxact:right>
          </ipxact:vector>
        </ipxact:vectors>
        <ipxact:wireTypeDefs>
          <ipxact:wireTypeDef>
            <ipxact:viewRef>rtl</ipxact:viewRef>
          </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
      </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
      <ipxact:name>socket</ipxact:name>
      <ipxact:transactional>
        <ipxact:initiative>provides</ipxact:initiative>
        <ipxact:transTypeDefs>
          <ipxact:transTypeDef>
            <ipxact:viewRef>tlm</ipxact:viewRef>
          </ipxact:transTypeDef>
        </ipxact:transTypeDefs>
      </ipxact:transactional>
    </ipxact:port>
  </ipxact:ports>
 </ipxact:model>
</ipxact:component>
```

This **component** differs from earlier components because the **port** and **abstractionType** elements contain a **viewRef** element. For ports that **viewRef** element is part of the **wireTypeDef**, **transTypeDef**, or **structTypeDef** elements. A **viewRef** element value defines in which view the containing **port** and containing **abstractionType** must be applied. Hence, in this example, the wire port and their port maps apply to view rtl and the transactional port and its port map apply to view tlm. If a port has no **viewRef** elements in its **wireTypeDef**, **transTypeDef**, or **structTypeDef** elements, then that port exists in all views. If a port has **viewRef** elements in its **wireTypeDef**, **transTypeDef**, or **structTypeDef** elements, then that port exists only in the referenced views. Hence, in Example 57, port named socket only exists in the tlm view and all other ports only exist in the rtl view of the component.

Figure 4 shows a design containing two component instances I and J. The components have views rtl and tlm as explained above. Their bus interfaces are connected with interconnection C. There are four design configurations to configure the design in four different ways by selecting either the rtl or the tlm view for each of the component instances. If a design configuration results in a mixed-abstraction configuration containing both rtl and tlm views, then abstraction conversion is required on interconnection C. For this reason, an abstractor must be instantiated on interconnection C. Abstractor instance A translates a signal-level initiator bus interface to a transaction-level target bus interface. Abstractor instance B translates a transaction-level initiator bus interface to a signal-level target bus interface. The four design configurations can be used to generated four different netlists as shown in Figure 5. The abstractor instance is part of the generated netlist.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

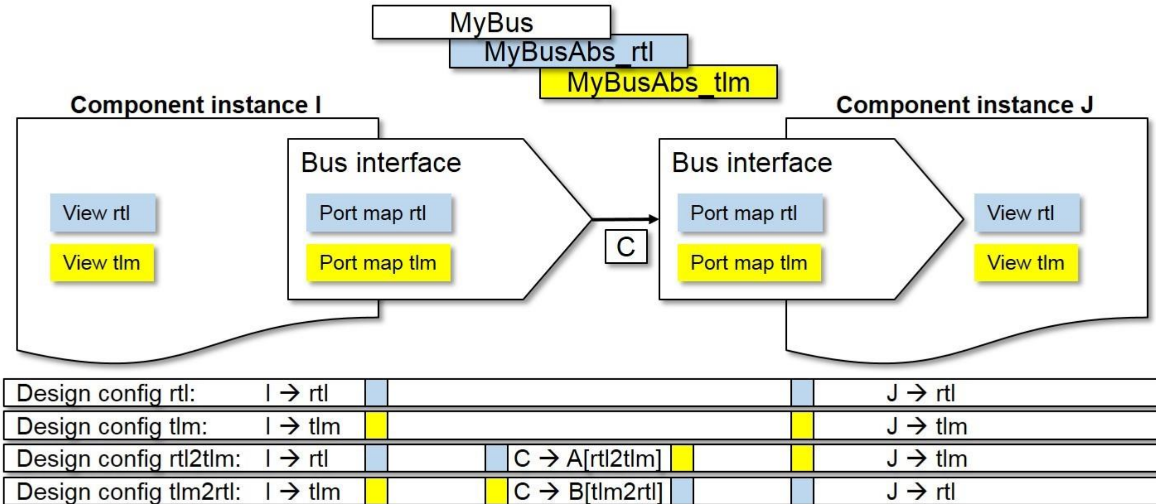

Figure 4 Design containing component instances I and J and four designConfigurations that configure views RTL and TLM for component instances I and J; two design configurations contain abstractor instances A and B performing RTL-to-TLM and TLM-to-RTL abstraction conversion on interconnection C

Design config rtl:	`I.clk → J.clk`
	`I.rstn → J.rstn`
	`I.addr[3:0] → J.addr[3:0]`
	`I.rdata[31:0] → J.rdata[31:0]`
	`I.wdata[31:0] → J.wdata[31:0]`	

Design config tlm:	`I.socket → J.socket`	

Design config rtl2tlm:	`I.clk → A.clk`
	`I.rstn → A.rstn`
	`I.addr[3:0] → A.addr[3:0]`
	`I.rdata[31:0] → A.rdata[31:0]`
	`I.wdata[31:0] → A.wdata[31:0]`
		`A.socket → J.socket`

Design config tlm2rtl:	`I.socket → B.socket`	`B.clk → J.clk`
		`B.rstn → J.rstn`
		`B.addr[3:0] → J.addr[3:0]`
		`B.rdata[31:0] → J.rdata[31:0]`
		`B.wdata[31:0] → J.wdata[31:0]`

Figure 5 Four netlists generated from the four design configurations containing signal-level and transaction-level port bindings

Example 58 shows an **abstractor** describing an IP block that converts the AHBLite target protocol from a TLM2 generic payload representation to an RTL representation.

Example 58 Abstractor AHBLiteTarget_tlm2gp_to_rtl

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:abstractor xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>AHBLiteTarget_tlm2gp_to_rtl</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:abstractorMode>direct</ipxact:abstractorMode>
  <ipxact:busType vendor="accellera.org" library="amba3" name="AHBLiteTarget" version="1.0"/>
  <ipxact:abstractorInterfaces>
    <ipxact:abstractorInterface>
      <ipxact:name>tlm</ipxact:name>
      <ipxact:abstractionTypes>
        <ipxact:abstractionType>
          <ipxact:abstractionRef vendor="accellera.org" library="amba3" name="AHBLiteTarget_tlm"
            version="1.0"/>
          <ipxact:portMaps>
            ...
          </ipxact:portMaps>
        </ipxact:abstractionType>
      </ipxact:abstractionTypes>
    </ipxact:abstractorInterface>
    <ipxact:abstractorInterface>
      <ipxact:name>rtl</ipxact:name>
      <ipxact:abstractionTypes>
        <ipxact:abstractionType>
          <ipxact:abstractionRef vendor="accellera.org" library="amba3" name="AHBLiteTarget_rtl"
            version="1.0"/>
          <ipxact:portMaps>
            ...
          </ipxact:portMaps>
        </ipxact:abstractionType>
      </ipxact:abstractionTypes>
    </ipxact:abstractorInterface>
  </ipxact:abstractorInterfaces>
  <ipxact:model>
    ...
  </ipxact:model>
  <ipxact:fileSets>
    ...
  </ipxact:fileSets>
</ipxact:abstractor>
```

The container element **abstractor** indicates that the IP-XACT XML file describes an **abstractor**. Subsequently, the identifier for an abstractor is specified using the four elements: **vendor**, **library**, **name**, and **version**. In this example, the values of these elements are accellera.org, ug, AHBLiteTarget_tlm2gp_to_rtl, and 1.0, respectively. The element **abstractorMode** can have four values: direct, initiator, target, or system. The value describes the interface modes to which the abstractor applies.

— Direct, indicates conversion from a bus interface with mode initiator to a bus interface with mode target.
— Initiator, indicates conversion from a bus interface with mode initiator to a bus interface with mode mirroredInitiator.
— Target, indicates conversion from a bus interface with mode mirroredTarget to a bus interface with mode target.
— System, indicates conversion from a bus interface with mode system to a bus interface with mode mirroredSystem.

The element **busType** references a **busDefinition** element, using the four elements **vendor**, **library**, **name**, and **version** identifier, indicating for which **busDefinition** the **abstractor** describes abstraction conversion. The element **abstractorInterfaces** is a container element for exactly two **abstractorInterface** elements. The first **abstractorInterface** element is applied to a bus interface describing the source of the abstraction conversion, in the interface mode as indicated by the **abstractorMode** element value. The second **abstractorInterface** element is applied to a bus interface describing the target of the abstraction conversion, in the interface mode as indicated by the **abstractorMode** element value. Each **abstractorInterface** element contains an **abstractionRef** element referencing an **abstractionDefinition** element, using the four elements **vendor**, **library**, **name**, and **version** identifier, describing the abstraction of the bus at that interface. Each **abstractorInterface** element also contains a **portMaps** element containing the logical-to-physical port

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

mapping identical to the **component busInterface** elements. Also, the **model** and **fileSets** elements in **abstractor** are identical to the elements in **component**.

Abstractor instances are described in **interconnectionConfiguration** elements within **designConfiguration** elements as shown in Example 59. An **interconnectionConfiguration** element references a **design interconnection** element in the design that is referenced by the **designRef** element in the **designConfiguration**, indicating which **interconnection** is configured with abstractor instances. An **interconnectionConfiguration** element contains one or more **abstractorInstances** elements. Each **abstractorInstances** element can have an **interfaceRef** element indicating to which endpoint of the **interconnection** it applies, by referencing a component instance and bus interface by name. If no **interfaceRef** element is described, then it applies to all endpoints of the interconnection. Furthermore, an **abstractorInstances** element has an **abstractorInstance** element which describes a sequence of abstractor instances that implement the abstraction conversion to the indicated endpoint(s).

Example 59 DesignConfiguration interconnectConfiguration

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:designConfiguration xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>design_cfg</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:designRef vendor="accellera.org" library="ug" name="design" version="1.0"/>
  <ipxact:interconnectionConfiguration>
    <ipxact:interconnectionRef>C</ipxact:interconnectionRef>
    <ipxact:abstractorInstances>
      <ipxact:interfaceRef componentRef="J" busRef="T"/>
      <ipxact:abstractorInstance>
        <ipxact:instanceName>u_AHBLiteTarget_tlm2gp_to_rtl</ipxact:instanceName>
        <ipxact:abstractorRef vendor="accellera.org" library="ug" name="AHBLiteTarget_tlm2gp_to_rtl"
          version="1.0"/>
        <ipxact:viewName>SystemC</ipxact:viewName>
      </ipxact:abstractorInstance>
    </ipxact:abstractorInstances>
  </ipxact:interconnectionConfiguration>
  <ipxact:viewConfiguration>
    <ipxact:instanceName>I</ipxact:instanceName>
    <ipxact:view viewRef="TLM"/>
  </ipxact:viewConfiguration>
  <ipxact:viewConfiguration>
    <ipxact:instanceName>J</ipxact:instanceName>
    <ipxact:view viewRef="RTL"/>
  </ipxact:viewConfiguration>
</ipxact:designConfiguration>
```

An **abstractorInstance** element has an **instanceName** to identify the instance. The **instanceName** matches the HDL instance name. Furthermore, an **abstractorInstance** element has an **abstractorRef** that references an **abstractor** using the **vendor**, **library**, **name**, **version** identifier. The **abstractorRef** describes the (abstractor) type of the instance. The **abstractorRef** element can have **configurableElementValues**. Finally, an **abstractorInstance** element has a **viewName** element that references a **view** by its **name** that exists in the **abstractor** referenced by the **abstractorRef** element.

3.2.1.2 Generator Chain

Generator chains describe tools that operate on IP-XACT XML documents to enable design environments to run those tools based on the defined flows in the chains. The location of tools and tool input values are described in IP-XACT generator chain documents. Example 60 shows a generator that traverses a design hierarchy taking a **component vendor**, **library**, **name**, and **version** identifier and a **component view name** as input.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 60 GeneratorChain

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:generatorChain xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>DesignHierarchyTraversal</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:generator>
    <ipxact:name>TraverseDesignHierarchy</ipxact:name>
    <ipxact:parameters>
      <ipxact:parameter parameterId="vendor" resolve="user" prompt="Component vendor">
        <ipxact:name>vendor</ipxact:name>
        <ipxact:value>accellera.org</ipxact:value>
      </ipxact:parameter>
      <ipxact:parameter parameterId="library" resolve="user" prompt="Component library">
        <ipxact:name>library</ipxact:name>
        <ipxact:value>ipxact</ipxact:value>
      </ipxact:parameter>
      <ipxact:parameter parameterId="name" resolve="user" prompt="Component name">
        <ipxact:name>name</ipxact:name>
        <ipxact:value>component</ipxact:value>
      </ipxact:parameter>
      <ipxact:parameter parameterId="version" resolve="user" prompt="Component version">
        <ipxact:name>version</ipxact:name>
        <ipxact:value>1.0</ipxact:value>
      </ipxact:parameter>
      <ipxact:parameter parameterId="view" resolve="user" prompt="Component view name">
        <ipxact:name>view</ipxact:name>
        <ipxact:value>my_view</ipxact:value>
      </ipxact:parameter>
    </ipxact:parameters>
    <ipxact:generatorExe>../../../../../org.accellera.ipxact.generators.DesignHierarchyTraversal.class
    </ipxact:generatorExe>
  </ipxact:generator>
</ipxact:generatorChain>
```

The container element **generatorChain** indicates that the IP-XACT XML file describes a **generatorChain**. Subsequently, the identifier for a generator chain is specified using the four elements: **vendor**, **library**, **name**, and **version**. In this example, the values of these elements are accellera.org, ug, DesignHierarchyTraversal, and 1.0, respectively. A **generatorChain** element contains one or more **generator** elements describing a sequence of generators. Each **generator** element has a **name** to identify the element. Furthermore, it can have a **parameters** element which is a container element for one or more **parameter** elements. Each **parameter** element describes a generator input. Finally, a **generator** element has a **generatorExe** element indicating the location of the generator executable or script.

3.2.1.3 Catalog

Catalogs describe the location and the **vendor**, **library**, **name**, and **version** identifier of other IP-XACT top-level elements in order to manage collections of IP-XACT files. Catalogs are organized in terms of top-level element types. Example 61 shows a catalog describing the location and identifier of a bus definition and an abstraction definition.

Example 61 Catalog

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:catalog xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>my_catalog</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
  <ipxact:busDefinitions>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="i2s" name="I2S" version="1.1"/>
      <ipxact:name>accellera.org/i2s/I2S/1.1/xml/I2S.xml</ipxact:name>
    </ipxact:ipxactFile>
  </ipxact:busDefinitions>
  <ipxact:abstractionDefinitions>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="i2s" name="I2S_rtl" version="1.1"/>
      <ipxact:name>accellera.org/i2s/I2S/1.1/xml/I2S_rtl.xml</ipxact:name>
    </ipxact:ipxactFile>
  </ipxact:abstractionDefinitions>
</ipxact:catalog>
```

The container element **catalog** indicates that the IP-XACT XML file describes a **catalog**. Subsequently, the identifier for a catalog is specified using the four elements: **vendor**, **library**, **name**, and **version**. In this example, the values of these elements are accellera.org, ug, my_catalog, and 1.0, respectively. Furthermore, a **catalog** element can contain **catalogs**, **busDefinitions**, **abstractionDefinitions**, **components**, **abstractors**, **designs**, **designConfigurations**, and **generatorChains** container elements. In this example, only **busDefinitions** and **abstractionDefinitions** elements are shown. All these container elements have an **ipxactFile** element describing the four elements **vendor**, **library**, **name**, and **version** identifier of the top-level element and the **name** indicating the location of the file containing the description of the top-level element.

3.2.2 Conditional Elements

Conditional elements have been removed in the IEEE 1685-2022 standard to reduce complexity of the standard and associated IPs and tools. To support backward compatibility of conditional elements, Accellera provides an IP-XACT vendor extension; see https://www.accellera.org/XMLSchema/IPXACT/1685-2022-VE-1.0. This section explains conditional elements in IEEE 1685-2014 and how they are supported with vendor extensions in IEEE 1685-2022.

In IEEE 1685-2014, many IP-XACT elements have an **isPresent** sub-element to support conditional existence. The value of **isPresent** is a Boolean expression in terms of parameters. If the expression evaluates to true, then the encapsulating IP-XACT element is considered to be included in the containing document. If the expression evaluates to false, then the encapsulating IP-XACT element is considered to be excluded from the containing document. This section contains examples for a conditional register and for a conditional port.

In Example 62, the **register** element named `STAT` has an **isPresent** element with value 'my_param>12' indicating that the **register** element has to be treated as if not present in the containing document if and only if the expression 'my_param>12' evaluates to false. In the expression, the term 'my_param' is a parameter that is defined in the same component as the register as shown in the example. The parameter is user-resolvable meaning that its actual value can be set using a **design componentInstance componentRef configurableElementValue** when the component is instantiated.

Example 62 Register isPresent (IEEE 1685-2014)

```
        <ipxact:register>
          <ipxact:name>STAT</ipxact:name>
          <ipxact:isPresent>my_param>12</ipxact:isPresent>
          <ipxact:description>Status register. Collection of Status flags including interrupt status
before enabling
          </ipxact:description>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:size>32</ipxact:size>
          ...
        </ipxact:register>

  <ipxact:parameters>
    <ipxact:parameter parameterId="my_param" resolve="user" type="longint">
      <ipxact:name>my_param</ipxact:name>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
      <ipxact:value>0</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
```

In IEEE 1685-2022, the **isPresent** element has been removed from the standard. It is supported with an Accellera vendor extension as shown in Example 63. The vendor extension uses a container element in the **accellera** namespace. In this example the container element is named **register** because the vendor extension applies to the **register** element. The container element contains an **isPresent** element in the **accellera-cond** namespace.

Example 63 Register isPresent (IEEE 1685-2022)

```
        <ipxact:register>
          <ipxact:name>STAT</ipxact:name>
          <ipxact:description>Status register. Collection of Status flags including interrupt status
before enabling
          </ipxact:description>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:size>32</ipxact:size>
          ...
          <ipxact:vendorExtensions>
            <accellera:register>
              <accellera-cond:isPresent>my_param>12</accellera-cond:isPresent>
            </accellera:register>
          </ipxact:vendorExtensions>
        </ipxact:register>

  <ipxact:parameters>
    <ipxact:parameter parameterId="my_param" resolve="user" type="longint">
      <ipxact:name>my_param</ipxact:name>
      <ipxact:value>0</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
```

Example 64 shows an IEEE 1685-2014 example with an **isPresent** element inside a port element. The **port** element named `my_port` has an **isPresent** element with value 'ip_config == "FPGA"' indicating that the **port** element has to be treated as if not present in the containing document if and only if the expression 'ip_config == "FPGA"' evaluates to false. In the expression, the term 'ip_config' is a parameter that is defined in the same component as the port as shown in the example. The parameter is user-resolvable meaning that its actual value can be set using a **design componentInstance componentRef configurableElementValue** when the component is instantiated.

Example 64 Port isPresent (IEEE 1685-2014)

```
      <ipxact:port>
        <ipxact:name>my_port</ipxact:name>
        <ipxact:isPresent>ip_config == "FPGA"</ipxact:isPresent>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
        </ipxact:wire>
      </ipxact:port>

  <ipxact:parameters>
    <ipxact:parameter parameterId="ip_config" resolve="user" type="string">
      <ipxact:name>ip_config</ipxact:name>
      <ipxact:value>FPGA</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
```

Example 65 shows the same example in IEEE 1685-2022. The **isPresent** element has been moved into a vendor extension that is encapsulated in a container named **port** because the vendor extension applies to the **port** element.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 65 Port isPresent (IEEE 1685-2022)

```
      <ipxact:port>
        <ipxact:name>my_port</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
        </ipxact:wire>
        <ipxact:vendorExtensions>
          <accellera:port>
            <accellera-cond:isPresent>ip_config == "FPGA"</accellera-cond:isPresent>
          </accellera:port>
      </ipxact:vendorExtensions>
    </ipxact:port>

  <ipxact:parameters>
    <ipxact:parameter parameterId="ip_config" resolve="user" type="string">
      <ipxact:name>ip_config</ipxact:name>
      <ipxact:value>FPGA</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
```

3.2.3 Parameter Passing

In IP-XACT, **parameter** elements within a top-level element can be defined to allow their value to be overridden by a referencer of that top-level element. Parameters that can be overridden are identified by having their resolve attribute set to either “user” or “generated”. When another element references a top-level element, such as in a **design componentInstance componentRef** element, values can be provided to override the default values specified by the referenced element. Configurable references to top-level elements are of type **configurableLibraryRefType**, such as **design componentInstances componentInstance componentRef** or **component busInterfaces busInterface busType**. The elements of type **configurableLibraryRefType** can include a **configurableElementValues** element that contains the parameters to override and the parameter values to override with in the referenced element.

Example 66 shows an example of parameter passing in Verilog. Module A and Module P have parameter pA and pB, respectively. Module A instantiates module B and passes the value of parameter pA onto the value of parameter pB using the expression pA*4+7.

Example 66 Parameter passing in Verilog

```
module B();
parameter pB = 1;
endmodule

module A();
parameter pA = 3; B #( .pB( (pA*4)+7 ) ) u_B();
endmodule
```

Example 67 shows the same example in IP-XACT. Component A and component B contain **componentInstantiation** hdl-rtl with **moduleParameter** pA and pB, respectively. The value of pA is equal to the value of **component parameter** param_A1 and the value of pB is equal to the value of **component parameter** param_B. These dependencies have been introduced to demonstrate that parameter passing is supported on components and designs independent of **designConfiguration viewConfiguration** elements. Components can have additional parameters that are not necessarily related to HDL parameters. In this example, **component** A has a second **parameter** param_A2. Furthermore, **component** A references parameterized **design** A_design in a **designInstantiation**. The value of **design parameter** param_A3 is set with a **configurableElementValue** using an expression param_A1*param_A2 which passes the value of the component parameters onto the value of the design parameter. **Design** A_design instantiates **component** B and the value of **component parameter** param_B is set with a **configurableElementValue** using expression param_A3+7 which passes the value of the design parameter onto the value of the component parameter. **View** rtl of **component** A references the **component componentInstantiation**, **designInstantiation**, and **designConfigurationInstantiation**, thereby configuring the IP-XACT design hierarchy in line with the

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Verilog module hierarchy containing parameters pA and pB. As a result, the expressions in terms of IP-XACT parameters param_A1, param_A2, param_A3, and param_B can be translated into expressions in terms of Verilog parameters pA and pB.

Example 67 Parameter passing in IP-XACT

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>B</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>rtl</ipxact:name>
        <ipxact:componentInstantiationRef>hdl-rtl</ipxact:componentInstantiationRef>
      </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:name>hdl-rtl</ipxact:name>
        <ipxact:language>Verilog</ipxact:language>
        <ipxact:moduleName>B</ipxact:moduleName>
        <ipxact:moduleParameters>
          <ipxact:moduleParameter>
            <ipxact:name>pB</ipxact:name>
            <ipxact:value>param_B</ipxact:value>
          </ipxact:moduleParameter>
        </ipxact:moduleParameters>
      </ipxact:componentInstantiation>
    </ipxact:instantiations>
  </ipxact:model>
  <ipxact:parameters>
    <ipxact:parameter parameterId="id_B" resolve="user">
      <ipxact:name>param_B</ipxact:name>
      <ipxact:value>1</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
</ipxact:component>

<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>A</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>rtl</ipxact:name>
        <ipxact:componentInstantiationRef>hdl-rtl</ipxact:componentInstantiationRef>
        <ipxact:designInstantiationRef>hdl-rtl_design</ipxact:designInstantiationRef>
        <ipxact:designConfigurationInstantiationRef>hdl-rtl_design_cfg
        </ipxact:designConfigurationInstantiationRef>
     </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:name>hdl-rtl</ipxact:name>
        <ipxact:language>Verilog</ipxact:language>
        <ipxact:moduleName>A</ipxact:moduleName>
        <ipxact:moduleParameters>
          <ipxact:moduleParameter>
            <ipxact:name>pA</ipxact:name>
            <ipxact:value>param_A1</ipxact:value>
          </ipxact:moduleParameter>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
        </ipxact:moduleParameters>
      </ipxact:componentInstantiation>
      <ipxact:designInstantiation>
        <ipxact:name>hdl-rtl_design</ipxact:name>
        <ipxact:designRef vendor="accellera.org" library="ug" name="A_design" version="1.0">
          <ipxact:configurableElementValues>
            <ipxact:configurableElementValue referenceId="id_A3">param_A1*param_A2
            </ipxact:configurableElementValue>
          </ipxact:configurableElementValues>
        </ipxact:designRef>
      </ipxact:designInstantiation>
      <ipxact:designConfigurationInstantiation>
        <ipxact:name>hdl-rtl_design_cfg</ipxact:name>
        <ipxact:designConfigurationRef vendor="accellera.org" library="ug" name="A_design_cfg"
          version="1.0">
        </ipxact:designConfigurationRef>
      </ipxact:designConfigurationInstantiation>
    </ipxact:instantiations>
  </ipxact:model>
  <ipxact:parameters>
    <ipxact:parameter parameterId="id_A1" resolve="user">
      <ipxact:name>param_A1</ipxact:name>
      <ipxact:value>3</ipxact:value>
    </ipxact:parameter>
    <ipxact:parameter parameterId="id_A2" resolve="user">
      <ipxact:name>param_A2</ipxact:name>
      <ipxact:value>4</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
</ipxact:component>

<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>A_design</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_B</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="ug" name="B" version="1.0">
        <ipxact:configurableElementValues>
          <ipxact:configurableElementValue referenceId="id_B">param_A3+7
          </ipxact:configurableElementValue>
        </ipxact:configurableElementValues>
      </ipxact:componentRef>
    </ipxact:componentInstance>
  </ipxact:componentInstances>
  <ipxact:parameters>
    <ipxact:parameter parameterId="id_A3" resolve="user">
      <ipxact:name>param_A3</ipxact:name>
      <ipxact:value>1</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
</ipxact:design>

<?xml version="1.0" encoding="UTF-8"?>
<ipxact:designConfiguration xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>A_design_cfg</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:designRef vendor="accellera.org" library="ug" name="A_design" version="1.0"/>
  <ipxact:viewConfiguration>
    <ipxact:instanceName>u_B</ipxact:instanceName>
    <ipxact:view viewRef="rtl"/>
  </ipxact:viewConfiguration>
</ipxact:designConfiguration>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

In addition to top-level elements, parameters within a **component model componentInstantiation** and **designConfigurationInstantiation** element can also be configured by the **designConfiguration viewConfiguration view** element. The configuration of the values of view-specific parameters are limited to parameters defined with the scope of that referenced view. In other words, a **designConfiguration viewConfiguration view** element cannot override a value within the referenced component that is not defined within the referenced **componentInstantiation** and **designConfigurationInstantiation** in the referenced **view**.

Example 68 shows a modification of Example 67 illustrating view-specific parameter passing. The **moduleParameter** pB in **component** B is now user-resolvable rather than dependent on a **component parameter**. As a result, the value of **moduleParameter** pB must be set in a **configurableElementValue** element in a **designConfiguration viewConfiguration**. Similar to parameter passing from component to design, parameters can be passed from component to design configuration. Here, **designConfiguration** A_design_cfg has a **parameter** param_A3 that is used to set the value of **moduleParameter** pB to param_A3+7. The value of **designConfiguration parameter** param_A3 is set in a **configurableElementValue** using an expression param_A1*param_A2 in the **component designConfigurationInstantiation**.

Example 68 View-specific parameter passing in IP-XACT

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>B</ipxact:name>
  <ipxact:version>2.0</ipxact:version>
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>rtl</ipxact:name>
        <ipxact:componentInstantiationRef>hdl-rtl</ipxact:componentInstantiationRef>
      </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:name>hdl-rtl</ipxact:name>
        <ipxact:language>Verilog</ipxact:language>
        <ipxact:moduleName>B</ipxact:moduleName>
        <ipxact:moduleParameters>
          <ipxact:moduleParameter parameterId="id_B" resolve="user">
            <ipxact:name>pB</ipxact:name>
            <ipxact:value>1</ipxact:value>
          </ipxact:moduleParameter>
        </ipxact:moduleParameters>
      </ipxact:componentInstantiation>
    </ipxact:instantiations>
  </ipxact:model>
</ipxact:component>

<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>A</ipxact:name>
  <ipxact:version>2.0</ipxact:version>
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>rtl</ipxact:name>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
        <ipxact:componentInstantiationRef>hdl-rtl</ipxact:componentInstantiationRef>
        <ipxact:designInstantiationRef>hdl-rtl_design</ipxact:designInstantiationRef>
        <ipxact:designConfigurationInstantiationRef>hdl-rtl_design_cfg
        </ipxact:designConfigurationInstantiationRef>
      </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:name>hdl-rtl</ipxact:name>
        <ipxact:language>Verilog</ipxact:language>
        <ipxact:moduleName>A</ipxact:moduleName>
        <ipxact:moduleParameters>
          <ipxact:moduleParameter>
            <ipxact:name>pA</ipxact:name>
            <ipxact:value>param_A1</ipxact:value>
          </ipxact:moduleParameter>
        </ipxact:moduleParameters>
      </ipxact:componentInstantiation>
      <ipxact:designInstantiation>
        <ipxact:name>hdl-rtl_design</ipxact:name>
        <ipxact:designRef vendor="accellera.org" library="ug" name="A_design" version="2.0"/>
      </ipxact:designInstantiation>
      <ipxact:designConfigurationInstantiation>
        <ipxact:name>hdl-rtl_design_cfg</ipxact:name>
        <ipxact:designConfigurationRef vendor="accellera.org" library="ug" name="A_design_cfg"
          version="2.0">
          <ipxact:configurableElementValues>
            <ipxact:configurableElementValue referenceId="id_A3">param_A1*param_A2
            </ipxact:configurableElementValue>
          </ipxact:configurableElementValues>
        </ipxact:designConfigurationRef>
      </ipxact:designConfigurationInstantiation>
    </ipxact:instantiations>
  </ipxact:model>
  <ipxact:parameters>
    <ipxact:parameter parameterId="id_A1" resolve="user">
      <ipxact:name>param_A1</ipxact:name>
      <ipxact:value>3</ipxact:value>
    </ipxact:parameter>
    <ipxact:parameter parameterId="id_A2" resolve="user">
      <ipxact:name>param_A2</ipxact:name>
      <ipxact:value>4</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
</ipxact:component>

<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>A_design</ipxact:name>
  <ipxact:version>2.0</ipxact:version>
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_B</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="ug" name="B" version="2.0"/>
    </ipxact:componentInstance>
  </ipxact:componentInstances>
</ipxact:design>

<?xml version="1.0" encoding="UTF-8"?>
<ipxact:designConfiguration xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>A_design_cfg</ipxact:name>
  <ipxact:version>2.0</ipxact:version>
  <ipxact:designRef vendor="accellera.org" library="ug" name="A_design" version="2.0"/>
  <ipxact:viewConfiguration>
    <ipxact:instanceName>u_B</ipxact:instanceName>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
    <ipxact:view viewRef="rtl">
      <ipxact:configurableElementValues>
        <ipxact:configurableElementValue referenceId="id_B">param_A3+7
        </ipxact:configurableElementValue>
      </ipxact:configurableElementValues>
    </ipxact:view>
  </ipxact:viewConfiguration>
  <ipxact:parameters>
    <ipxact:parameter parameterId="id_A3" resolve="user">
      <ipxact:name>param_A3</ipxact:name>
      <ipxact:value>1</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
</ipxact:designConfiguration>
```

3.2.4 Modes and Mode References

Components can contain user-defined **modes** that describe internal operating modes of component. Each **mode** element that has a **name** that describes a symbolic name for the operating mode and a **condition** that describes a Boolean expression that indicates when the mode is active in the encapsulating component. The Boolean expression is of type ***unresolvedUnsignedBitExpression***, meaning that it may not be possible to resolve the value of the expression inside the IP-XACT Design Environment. The reason for that is that the expression can contain terms of which the values can only be determined at run-time, such as the value of an input port and the value of a register field.

Example 69 show a **component** containing two **modes**. The first **mode** is named FUNC. It contains a **portSlice** element that is named testmode and references a component **port** named test_mode. The name of the **portSlice** is referenced in the value of the **condition** element. The special function $ipxact_port_value takes the **portSlice name** value as argument and returns an unresolved bit expression that represents the value of port test_mode at run-time. The condition states that mode FUNC is active if the value of port test_mode equals 0. Similarly the second mode named TEST is active if the value of port_test mode equals 1. Mode conditions do not need to be mutually exclusive. Hence, multiple modes can be active at any given time.

Example 69 Component modes

```
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>component_modes</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:modes>
    <ipxact:mode>
      <ipxact:name>FUNC</ipxact:name>
      <ipxact:portSlice>
        <ipxact:name>testmode</ipxact:name>
        <ipxact:portRef portRef="test_mode"/>
      </ipxact:portSlice>
      <ipxact:condition>$ipxact_port_value(testmode) == 0</ipxact:condition>
    </ipxact:mode>
    <ipxact:mode>
      <ipxact:name>TEST</ipxact:name>
      <ipxact:portSlice>
      <ipxact:name>testmode</ipxact:name>
      <ipxact:portRef portRef="test_mode"/>
        </ipxact:portSlice>
       <ipxact:condition>$ipxact_port_value(testmode) == 1</ipxact:condition>
    </ipxact:mode>
  </ipxact:modes>
  <ipxact:model>
    <ipxact:ports>
      <ipxact:port>
        <ipxact:name>test_mode</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
        </ipxact:wire>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
      </ipxact:port>
    </ipxact:ports>
  </ipxact:model>
</ipxact:component>
```

The component modes can be referenced in **accessPolicy** elements. Such elements are present in **addressBlock**, **registerFile**, and **register** elements. An **accessPolicy** element can contain zero or more **modeRef** elements meaning that other sub-elements inside an **accessPolicy** element only apply in the referenced modes. If an **accessPolicy** element does not contain a **modeRef** element, then its sub-elements apply when no other **accessPolicy** element in the encapsulating **accessPolicies** element applies.

Example 70 shows a **register** with an **accessPolicies** element that contains two **accessPolicy** elements. The first **accessPolicy** element references mode FUNC meaning that the **access** element value read-only only applies in mode FUNC. The **modeRef** element has a **priority** attribute which shall have a unique value with the containing **accessPolicies** element. If multiple modes are active, the **modeRef** element with the lowest numerical **priority** value shall have the highest priority. The second **accessPolicy** element does not contain **modeRef** elements. Hence, it applies when mode FUNC is not active.

Example 70 Register access policies

```
        <ipxact:register>
          <ipxact:name>STAT</ipxact:name>
          <ipxact:description>Status register. Collection of Status flags including interrupt status
before enabling
          </ipxact:description>
          <ipxact:addressOffset>'h0</ipxact:addressOffset>
          <ipxact:size>32</ipxact:size>
          <ipxact:accessPolicies>
            <ipxact:accessPolicy>
              <ipxact:modeRef priority="1">FUNC</ipxact:modeRef>
              <ipxact:access>read-only</ipxact:access>
            </ipxact:accessPolicy>
            <ipxact:accessPolicy>
              <ipxact:access>read-write</ipxact:access>
            </ipxact:accessPolicy>
          </ipxact:accessPolicies>
          ...
        </ipxact:register>
```

Mode references describe operating mode specific addressing and access into memory maps and they may occur in the following elements:

- initiator/addressSpaceRef
- target/memoryMapRef
- memoryRemap
- alternateRegister
- addressBlock/accessPolicies
- bank/accessPolicies
- register/accessPolicies
- alternateRegister/accessPolicies
- registerFile/accessPolicies
- field/fieldAccessPolicies
- accessRestrictions

3.2.5 Structured Ports

Structured ports describe compositions of wire ports in terms of structures, unions, and SystemVerilog interfaces. Only SystemVerilog interfaces that describe structural connectivity are supported. SystemVerilog interfaces that contain behavior are not supported.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 71 shows a SystemVerilog model with an output port named `myPort` that has a structure named `myStruct` as type. The structure is packed and has two fields named `mySubPort1` and `mySubPort2`.

Example 71 Module port using structure

```
 typedef struct packed {
  logic [11:0] mySubPort1;
  logic  [3:0] mySubPort2;
} myStruct;

module m(output myStruct myPort);
endmodule
```

Example 72 shows the description of port `myPort` in IP-XACT using a **structured** element inside a **port** element.

Example 72 Component structured port with struct

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>m</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:ports>
      <ipxact:port>
        <ipxact:name>myPort</ipxact:name>
          <ipxact:structured packed="true">
            <ipxact:struct direction="out"/>
            <ipxact:subPorts>
              <ipxact:subPort>
                <ipxact:name>mySubPort1</ipxact:name>
                  <ipxact:wire>
                  <ipxact:direction>out</ipxact:direction>
                  <ipxact:vectors>
                    <ipxact:vector>
                    <ipxact:left>11</ipxact:left>
                    <ipxact:right>0</ipxact:right>
                  </ipxact:vector>
                </ipxact:vectors>
              </ipxact:wire>
            </ipxact:subPort>
            <ipxact:subPort>
              <ipxact:name>mySubPort2</ipxact:name>
              <ipxact:wire>
                <ipxact:direction>out</ipxact:direction>
                <ipxact:vectors>
                  <ipxact:vector>
                    <ipxact:left>3</ipxact:left>
                    <ipxact:right>0</ipxact:right>
                  </ipxact:vector>
                </ipxact:vectors>
              </ipxact:wire>
            </ipxact:subPort>
          </ipxact:subPorts>
          <ipxact:structPortTypeDefs>
            <ipxact:structPortTypeDef>
              <ipxact:typeName>myStruct</ipxact:typeName>
            </ipxact:structPortTypeDef>
          </ipxact:structPortTypeDefs>
        </ipxact:structured>
      </ipxact:port>
    </ipxact:ports>
  </ipxact:model>
</ipxact:component>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

The **structured** element has a Boolean attribute **packed** to indicated if a structure is packed or unpacked. The element **struct** indicates that the structured port is a struct. The attribute **direction** describes the direction of the port. The container element **subPorts** described the field of the structure. Each **subPort** element describes one field. The **name** element describes the name of the field. The **subPort** element can contain another **subPort** element to describe nested structures or a **wire** element to describe a leaf of the structure. In this example, each **subPort** element contains a **wire** element. The element **structPortTypeDefs** describes the type names for the port for different views. In this case there is one **structPortTypeDef** element that applies to all views since there are no **viewRef** elements. The **typeName** element describes the name of the struct.

For port types that are unions, the IP-XACT description is similar. The only difference is that the element **struct** is replaced by the element **union** to indicate that the composite type is a union rather than a struct.

To explain port types that are SystemVerilog interfaces, we use Example 73 containing SystemVerilog interface named `itr`, two modules named `ip1` and `ip2` that use the modports named `mod1` and `mod2` of the SystemVerilog interface, and module named `top` that instantiates and connects instances of `ip1` and `ip2`.

Example 73 SystemVerilog interface with modports

```
interface itr( input clk );
  logic x,y;
  modport mod1( input x, clk );
  modport mod2( output y );
endinterface

module ip1( itr.mod1 z );
endmodule

module ip2( itr.mod2 z );
endmodule

module top(input clk);
  itr my_if(.clk(clk));
  ip1 u_ip1(.z(my_if));
  ip2 u_ip2(.z(my_if));
endmodule
```

Example 74 shows the description of module `ip1`. The **component** has a **port** named `z`. That port contains a **structured** element with sub-element **interface** to indicate that the port describes a SystemVerilog interface. The type of **port** `z` is `itr.mod1` so the **subPort** elements describe the arguments of modport `mod1`. The first argument is `x` which is a logic with direction inout inside the SystemVerilog interface. The direction in of argument `x` inside modport `mod1` is not relevant for the IP-XACT description. The second argument is `clk` which is an input port of the SystemVerilog interface. To indicate that `clk` is a port of the interface the **attribute** isIO of the **subPort** element has value true. The **structPortTypeDef** element contains a **typeName** element to describe the name of the interface and a **role** element to describe the name of the modport.

Example 74 Component structured port for modport mod1

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>ip1</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:ports>
      <ipxact:port>
        <ipxact:name>z</ipxact:name>
          <ipxact:structured packed="true">
            <ipxact:interface/>
            <ipxact:subPorts>
              <ipxact:subPort>
                <ipxact:name>x</ipxact:name>
                <ipxact:wire>
```


Copyright © 2023 Accellera Systems Initiative. All rights reserved.

```
                  <ipxact:direction>inout</ipxact:direction>
                </ipxact:wire>
              </ipxact:subPort>
              <ipxact:subPort isIO="true">
                <ipxact:name>clk</ipxact:name>
                  <ipxact:wire>
                  <ipxact:direction>in</ipxact:direction>
              </ipxact:wire>
            </ipxact:subPort>
          </ipxact:subPorts>
          <ipxact:structPortTypeDefs>
            <ipxact:structPortTypeDef>
              <ipxact:typeName>itr</ipxact:typeName>
              <ipxact:role>mod1</ipxact:role>
            </ipxact:structPortTypeDef>
          </ipxact:structPortTypeDefs>
        </ipxact:structured>
      </ipxact:port>
    </ipxact:ports>
  </ipxact:model>
</ipxact:component>
```

Similarly, Example 75 shows the description of module `ip2` with port `z` of type `itr.mod2`.

Example 75 Component structured port for modport mod2

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>ip2</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:ports>
      <ipxact:port>
        <ipxact:name>z</ipxact:name>
          <ipxact:structured packed="true">
            <ipxact:interface/>
            <ipxact:subPorts>
              <ipxact:subPort>
                <ipxact:name>y</ipxact:name>
                <ipxact:wire>
                  <ipxact:direction>inout</ipxact:direction>
                </ipxact:wire>
              </ipxact:subPort>
          </ipxact:subPorts>
          <ipxact:structPortTypeDefs>
            <ipxact:structPortTypeDef>
              <ipxact:typeName>itr</ipxact:typeName>
              <ipxact:role>mod2</ipxact:role>
            </ipxact:structPortTypeDef>
          </ipxact:structPortTypeDefs>
        </ipxact:structured>
      </ipxact:port>
    </ipxact:ports>
  </ipxact:model>
</ipxact:component>
```

If a SystemVerilog interface has parameters, then these parameters are described in a **typeParameters** element inside a **structPortTypeDef** element.

Example 76 shows an IP-XACT description of module top with component instances `u_ip1` and `u_ip2`. The **design** contains an **adHocConnection** between port `z` of instance `u_ip1` and port `z` of instance `u_ip2`. There is a second **adHocConnection** between port `clk` of instance `u_ip1` and the top port `clk`. This **adHocConnection** contains an **internalPortReference** with sub-element **subPortReference** that references the **subPort** `clk` of **port** `z` in **component** `ip1`.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

Example 76 Design connecting structured ports

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-2022"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>ug</ipxact:library>
  <ipxact:name>top_design</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_ip1</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="ug" name="ip1" version="1.0"/>
    </ipxact:componentInstance>
     <ipxact:componentInstance>
      <ipxact:instanceName>u_ip2</ipxact:instanceName>
      <ipxact:componentRef vendor="accellera.org" library="ug" name="ip2" version="1.0"/>
    </ipxact:componentInstance>
 </ipxact:componentInstances>
  <ipxact:adHocConnections>
    <ipxact:adHocConnection>
      <ipxact:name>u_ip1_z_u_ip2_z</ipxact:name>
      <ipxact:portReferences>
        <ipxact:internalPortReference componentInstanceRef="u_ip1" portRef="z">
        </ipxact:internalPortReference>
        <ipxact:internalPortReference componentInstanceRef="u_ip2" portRef="z">
        </ipxact:internalPortReference>
      </ipxact:portReferences>
    </ipxact:adHocConnection>
    <ipxact:adHocConnection>
      <ipxact:name>u_ip1_z_clk_clk</ipxact:name>
      <ipxact:portReferences>
        <ipxact:internalPortReference componentInstanceRef="u_ip2" portRef="z">
          <ipxact:subPortReference subPortRef="clk"/>
        </ipxact:internalPortReference>
        <ipxact:externalPortReference portRef="clk"></ipxact:externalPortReference>
      </ipxact:portReferences>
    </ipxact:adHocConnection>
  </ipxact:adHocConnections>
</ipxact:design>
```

Note that **subPortReference** elements can also be used to connect **subPort** elements inside **structured** ports that describe structs and unions.

3.2.6 Tight Generator Interface

The Tight Generator Interface (TGI) provides an API to query, modify, create, and delete IP-XACT XML documents residing in an IP-XACT compliant design tool. The standard defines this API in terms of SOAP messages in order to make it programming language neutral. The SOAP messages are defined in the IEEE 1685-2014 standard. The purpose of the TGI is to abstract from direct XML document manipulation and enable a client/server architecture between IP-XACT design tools and third-party TGI generators.

The TGI uses the concept of handles to objects. Handles are called identifiers (IDs). Objects are entities that can be described in IP-XACT XML documents. There are two classes of IDs, namely unconfigured IDs and configured IDs. An unconfigured ID provides access to an object type, i.e., an uninstantiated object with default parameter values. A configured ID provides access to an object instance, i.e., an instantiated object with default parameter values that may have been replaced with actual, instance-specific parameter values. Access through an unconfigured ID returns unconfigured IDs, while access through a configured ID returns configured IDs. Configured IDs can be converted to their unconfigured IDs, but only for top-level IP-XACT elements bus definition, abstraction definition, component, abstractor, design, design configuration, and generator chain, as well as parameters. Unconfigured IDs cannot be converted to configured IDs. However,

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

unconfigured IDs can provide access to objects that instantiate other objects and that are referenced through configured IDs as follows.

- — A component externalTypeDefinitions instantiates parameterized type definitions.
- — A component bus interface instantiates a parameterized bus definition.
- — A component bus interface can instantiate one or more parameterized abstraction definitions.
- — A component design instantiation instantiates a parameterized design.
- — A component design configuration instantiation instantiates a parameterized design configuration.
- — A design component instance instantiates a parameterized component.
- — A design configuration abstractor instance instantiates a parameterized abstractor.
- — A type definitions externalTypeDefinitions instantiates parameterized type definitions.

Note that configurable elements in a design configuration view configuration configure the **componentInstantiation** and **designConfigurationInstantiation** of a component instance.

Example 77 shows the usage of TGI with a Tcl programming interface, illustrating how to traverse component memory maps in order to access registers in address blocks. The Tcl namespace tgi:: encapsulates the TGI API SOAP or REST messages in Tcl procedures. This Tcl API and its implementation are assumed to be provided by an IP-XACT compliant design tool that supports TGI generator development.

Example 77 TGI generator in Tcl querying registers

```
# Get unconfigured ID for the component with the given VLNV
set componentID [ tgi::getID [ list "accellera.org" "myLib" "myComponent" "1.0" ] ]

# Get the memory maps of the component
set memoryMapIDs [ tgi::getComponentMemoryMapIDs $componentID ]

# Walk each memory map
foreach memoryMapID $memoryMapIDs {

  # Get the memory map elements of the memory map
  set memoryMapElementIDs [ tgi::getMemoryMapElementIDs $memoryMapID ]
  # Walk each memory map element
  foreach memoryMapElementID $memoryMapElementIDs {

    # Get the type of the memory map element
    set type [ tgi::getMemoryMapElementType $memoryMapElementID ]

    # Check if the memory map element is an address block
    if { [ string compare $type "addressBlock" ] == 0 } {

      # Get the registers of the address block
      set registerIDs [ tgi::getAddressBlockRegisterIDs $memoryMapElementID ]

      # Walk each register
      foreach registerID $registerIDs {

        # Get the register name, description, and address offset
        set registerName [ tgi::getName $registerID ]
        set registerDescription [ tgi::getDescription $registerID ]
        set registerOffset [ tgi::getRegisterAddressOffset $registerID ]
      }
    }
  }
}
```

This style of programming can be used to traverse design hierarchies and access component instances and their configurable element values. Also, the view configuration of the component instances can be retrieved from design configurations in a given design hierarchy. The unconfigured component and designs can be retrieved from the configured component and design instances. In this way, TGI can be used to develop generators to create proprietary views from IP-XACT XML documents.

TGI can be used also to develop generators to create IP-XACT XML documents from proprietary views. Example 78 shows the use of TGI to create a component register description in an address block of 4K 32bit

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

words. All string arguments describing the register properties are typically extracted or derived from a proprietary view or user input. A typical example is to parse register descriptions from a spreadsheet to create an IP-XACT register description.

Example 78 TGI generator in Tcl creating registers

```
# Create a new component with the given VLNV and get its unconfigured ID
set componentID [ tgi::createComponent [ list "accellera.org" "myLib" "myComponent" "1.0" ] ]

# Create a new memory map in the component with the given memory map name set memoryMapID
[ tgi::addComponentMemoryMap $componentID "myMemoryMap" ]

# Create a new address block in the memory map with the given address block name, base address,
# range, and width
set addressBlockID [ tgi::addMemoryMapAddressBlock $memoryMapID "myAddressBlock" "'h0" "'h1000" "32" ]

# Create a new register in the address block with the given register name, address offset, and size,
# and create a new field in the register with the given field name, bit offset, and bit width set
registerID [ tgi::addAddressBlockRegister $addressBlockID "reg" "'h0" "32" "field1" "0" "8" ]

# Set the description of the register
tgi::setDescription $registerID "This is my register description."

# Create a new field in the register with the given field name, bit offset, and bit width
set fieldID [ tgi::addRegisterField $registerID "field2" "8" 24 ]

# Set the description of the field
tgi::setDescription $fieldID "This is my second field description"
```

This way of TGI programming or scripting can be used to create complete design hierarchies. An example of such usage is the creation of an IP-XACT design hierarchy for a configurable hierarchical IP block. The configuration values for such a block are taken as input by the TGI generator creating the IP-XACT component, design, and design configuration files.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

4. Use Models

The IP-XACT standard provides the means for consistent, machine-readable descriptions of non-hierarchical and hierarchical IP that can be used in many ways to improve design processes. Typical use models include packaging, that is often applied in the context of non-hierarchical IP, and assembly, that is applied in the context of hierarchical IP. These use models are supported by IP-XACT compliant design environments offered by IP and EDA vendors such as the ones listed on the Accellera IP-XACT ecosystem webpage. More advanced use models include the use of vendor extensions in IP-XACT documents to describe additional, non-standard information and the use of TGI generators to automate proprietary flows.

4.1 Typical Use Models

This section covers typical use models related to packaging and assembly.

4.1.1 Packaging

In the context of IP-XACT, the term *packaging* means creating an IP-XACT component XML document. Typically, that document describes what is available in terms of IP deliveries for that component and meta-data of those deliveries such as port names, register descriptions, and file locations. There are different approaches to IP packaging.

One approach is that configurable IP blocks are configured through an IP configuration tool. The IP configuration tool accepts IP configuration parameters and generates views for the configured IP block, including an IP-XACT description. Another approach is that IP-XACT descriptions are created from existing views of configured IP blocks. For example, HDL code can be parsed to generate IP-XACT component ports and fileset, and SystemRDL code can be parsed to generate IP-XACT register descriptions. In both approaches, the resulting IP-XACT component describes a configured IP block with resolved parameter values and evaluated expressions.

IP-XACT supports expressions for all value types. This enables the description of configurable IP blocks in IP-XACT because all values can be described as expressions in terms of IP configuration parameters. In earlier versions of the standard, it was not always possible to store the expression itself in XML documents and the value of the evaluated expression had to be stored instead for given values of the configuration parameters. A consequence of supporting expressions is that a significant part of the semantic consistency rules, e.g., overlapping registers, can be checked only after the values of the parameters occurring in the expressions have been resolved.

IP-XACT component descriptions can be used for many purposes including:

- IP transfer
- Component documentation
- Register testing.

4.1.1.1 IP Transfer

Enabling IP exchange between different parties is one of the main goals of the IP-XACT standard. In this use model, IP-XACT component descriptions are produced by IP providers and consumed by IP users. The IP-XACT standard defines the syntax and semantics of such descriptions, hence, there is no ambiguity in the handover and no need for conventions, such as standard directory structures. Furthermore, the IP-XACT XML exchange format enables automated processing such that IP users can easily process IP deliveries in their own

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

third-party or proprietary tool flows. It also enables automated processing for IP providers such that they can easily create and verify IP deliveries in their own flows.

In many configurable IP flows, the IP configuration process results in IP deliveries containing configured IP including an IP-XACT component description. This description can be used by the IP user to integrate the configured IP into a System-on-Chip. The IP-XACT component description can be used also by the IP provider in the IP configuration flow to generate configured IP views such as documentation, HDL interface, software register abstraction layer, and UVM register model, e.g., using TGI generators.

4.1.1.2 Component Documentation

Traditional component documentation, e.g., data sheets, can be generated easily from IP-XACT component descriptions in a variety of formats and styles. The big advantage of IP-XACT is that it standardizes the format of data sheets in a machine-readable format. Machine processing can be used to translate this format in an organization-specific format. Furthermore, if the IP-XACT documents have been used in design and verification flows, then they are accurate, consistent, up-to-date, and a good source of information to generate documentation. Documentation flows are not limited to single components. Complete design hierarchies can be traversed and global memory maps can be computed to generate parts of System-on-Chip documentation.

4.1.1.3 Register Testing

IP-XACT components containing register descriptions can be used to automate register testing. The metadata describes both the register data layout and, to some extent, the effect of actions on register data such as the fact that writing a 1 to a bit will clear that bit after the write action. Hence, generated register tests can include testing of data layout, access properties, and effects of read and write actions on register fields. Common approaches for register testing are UVM-based approaches and software-driven approaches. Both approaches generate register transactions through bus interfaces towards register blocks and test if a register block implementation matches its IP-XACT register description.

For hierarchical components, the design hierarchy can be traversed and global memory maps can be calculated; in these global memory maps, component registers are mapped into CPU address spaces in order to generate UVM register models and software memory maps to perform System-on-Chip register testing.

4.1.2 Assembly

IP assembly defines a means for creating an IP-XACT design XML document in order to create a new hierarchical IP block, subsystem, or system. There are different approaches to IP assembly.

One approach is that available IP-XACT components are instantiated in a design. The parameter values of these component instances are set and the component instances are connected. The interface of the design can be described by an IP-XACT component that references the design. The views of the component instances can be selected in a design configuration that references the design. The new IP-XACT component can be instantiated in other designs. In this way, a design hierarchy can be created in a bottom-up way. Another approach is that the interface of the hierarchical IP-XACT component is described before the IP-XACT design of that component is assembled. In this way, a design hierarchy can be created in a top-down way. Both approaches can be combined. Note that a component may reference multiple designs and a design may be referenced by multiple design configurations. In other words, a component may be implemented by multiple design topologies and a design topology may have multiple design configurations.

The physical connectivity, e.g., as structural RTL or TLM description, as well as the logical connectivity, e.g., as system memory map description, can be generated from IP-XACT design hierarchies. First, the logical connectivity and addressing can be described to construct the system memory map for all addressable initiators in a System-on-Chip. Next, physical connectivity can be introduced by adding component ports and ports maps that map component ports onto logical ports. Component ports can be wire ports for RTL connectivity

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

and transactional ports for TLM connectivity. Ideally, system memory map, TLM connectivity, and RTL connectivity are generated from the same IP-XACT design hierarchy.

An additional well-known use model of IP-XACT design hierarchies is design hierarchy transformation. In this use model, the IP-XACT design hierarchy is transformed into another design hierarchy before the structural RTL or TLM description is generated in a target language. Since IP-XACT is language neutral, design hierarchy transformations can be applied in combination with different target languages.

4.2 Advanced Use Models

This section covers advanced use models related to data exchange between tools and proprietary tool flows.

4.2.1 Data Exchange Between Tools

An added value of IP-XACT is that it standardizes the XML document format that is exchanged between parties and tools. If the standard does not support the description of particular pieces of information, then this information can be entered in such XML documents as IP-XACT vendor extensions. Vendor extensions are well-defined locations in IP-XACT XML documents where information can be added that cannot be described in the IP-XACT XML schema. The limitation is that this information can only be interpreted and processed by tools that understand its meaning. However, IP-XACT tools that do not understand a vendor extension need to leave the information intact.

4.2.2 Proprietary Tool Flows

The Tight Generator Interface (TGI) provides an interface to IP-XACT compliant design environments to process IP-XACT documents. The use of TGI ensures that generators operate in any IP-XACT design environment. TGI generators can be used for generation of IP-XACT meta-data as well as generation from IP-XACT meta-data.

- Generation of IP-XACT meta-data includes component packaging and design assembly. TGI generators can read content in specific formats and create IP-XACT meta-data from that content. Examples are:
- generators that read HDL files or table formats to create module names, model parameters, ports, file sets, and bus interfaces;
- generators that read SystemRDL files or table formats to create registers;
- generators that read configuration values of a configurable IP configuration to create a (hierarchical) IP-XACT component describing the configured IP;
- generators that read HDL files or table formats to create component instances, parameter values, and connections;
- generators that read design partitioning information to perform meta-data manipulation for design hierarchy transformation or cell insertion for cells such as clamps, level shifters, clock domain crossers, reset synchronizers, mixed-signal connect modules, and mixed-abstraction transactors.

Generation from IP-XACT meta-data includes netlisting. TGI generators read IP-XACT meta-data and create content in specific formats. Examples are:

- generators to create human-readable data sheets, e.g., rendered from DITA or HTML formats;
- generators to create software register abstraction layers and memory maps, e.g., in C/C++ or ARM CMSIS SVD formats;
- generators to create ESL register implementations and interconnect, e.g., in SystemC format;
- generators to create RTL register implementations and interconnect, e.g., in Verilog or VHDL formats;
- generators to create UVM register models, e.g., in SystemVerilog format;

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

— generators to create file lists and compilation scripts, e.g., in Make, Tcl, or EDA vendor tool formats.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

5. Evolution of the Standard

This section describes the evolution of the standard based on different releases. First, an overview of the different releases is given together with the main motivation for each release. Next, the main differences between the releases are summarized.

5.1 Motivation of each Release

So far seven different versions of the IP-XACT standard have been released. The first four version have been released by The Spirit Consortium which merged into the Accellera Systems Initiative in 2010. The last three versions have been released by IEEE. The versions and release dates are listed below.

- IP-XACT 1.0, December 2004
- IP-XACT 1.1, June 2005
- IP-XACT 1.2, April 2006
- IP-XACT 1.4, March 2008
- IEEE Std. 1685-2009, December 2009
- IEEE Std. 1685-2014, June 2014
- IEEE Std. 1685-2022, September 2022

The aim of the IP-XACT versions 1.0, 1.1, and 1.2 was to support Register Transfer Level descriptions of IP blocks. The aim of IP-XACT version 1.4 was to extend the description of IP blocks to Transaction Level Model descriptions. The first IEEE version of the standard supported enhanced register descriptions. The second revision of the IEEE standard added a lot of new features including conditionality, extended parameter propagation through hierarchical components, view-specific port maps, SystemVerilog expression language, and full schema coverage in TGI. The most recent version of the standard removes conditionality and adds new features such as type definitions, structure ports, and power domains.

5.2 Key Elemental Differences between Adjacent Releases

The early versions of the IP-XACT standard, i.e., 1.0, 1.1, and 1.2, were focused primarily on support for RTL based design. IP-XACT 1.1 added support for modeling of synthesis constraints and the notion of a generator interface that could query and configure the XML via a Loose Generator Interface (LGI) supporting updates sent via XML files, or a Tight Generator Interface (TGI) supporting updates sent via a SOAP interface. IPXACT 1.2 added a consistent model for hierarchical designs, replacing the prior **componentInstances** element with the notion of components referencing designs which instantiate components.

IP-XACT 1.4 introduced TLM IP support. In order to facilitate TLM support, several changes were made amongst others to component signals and bus definitions. Component signals were replaced by component ports. Two type of component ports were introduced: wire ports and transactional ports. The component wire ports contained the functionality of the original component signals. The component transactional ports were introduced to describe TLM1 and TLM2 style transactional ports. Bus definitions were split into bus and abstraction definitions. One bus definition can be used in combination with multiple abstraction definitions. In this way, it is possible to describe multiple abstractions of the same bus, e.g., an RTL abstraction and a TLM abstraction. Each component bus interface references a bus and, optionally, an abstraction definition. Two bus interfaces can be connected if they reference the same bus definition. However, the referenced abstraction definitions may be different. In order to describe the translation between two abstraction definitions, a new top-level element was introduced called an **abstractor**. An abstractor is a specialized component that describes the meta-data of a module or entity that translates from one abstraction to another abstraction. Abstractors are not instantiated in designs like regular components, rather they are instantiated in design configurations on specific interconnections that reference bus interfaces with different abstractions. The underlying concept is to keep the IP-XACT design topology independent of the views that are selected for the

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

component instances. IP-XACT 1.4 also defined the TGI as part of the standard. The TGI was introduced in IP-XACT 1.1 in order to support a standard client/server model between IP-XACT design environments and third-party generators. The idea is that TGI generators can be executed remotely from design environments in order to configure components such as bus fabrics and to integrate these components in designs. Also, the TGI provides abstraction from direct XML file manipulation using XSL transformations in order to make generators less dependent on the details of a particular IP-XACT version. The TGI has been defined in terms of SOAP messages in order to be programming language neutral.

IEEE Std. 1685-2009 enhanced register descriptions. Register files were introduced to support nested register descriptions. Also, the **modifiedWriteValue** and **readAction** elements were introduced to describe the modification of register fields after a write or read action, respectively. The register and field access types were extended with **writeOnce** and **read-writeOnce**. The enumerated value usage attribute was introduced to enable restricting of enumerated values to read, write, and read-write. Also, the concept of alternate registers was introduced. Furthermore, address space segments were introduced in order to be able to split address spaces in multiple segments. The TGI was modified to support identifiers for unconfigured and configured components meaning that it was possible to access the default values of parameters as well as the actual values of parameters while querying component meta-data.

IEEE Std. 1685-2014 was a major revision of the standard introducing a lot of new features and changes. One objective of this version is to achieve the goal of a single design topology supporting switching of RTL and TLM views in different design configurations. Although TLM support was introduced in IP-XACT 1.4, switching of RTL and TLM views was not possible because component wire and transactional ports were not view specific. Also, the bus interface port map was not view specific. The view-specific component ports and port maps have been introduced in this version. Another new feature is the ability to propagate parameters through design hierarchies. To this end, designs, design configurations, bus definitions, and abstraction definitions have been augmented with parameters. Component parameter values can be propagated to referenced designs, design configurations, bus definitions, and abstraction definitions. Design and design configuration parameters can be propagated to referenced components. To simplify expressions in terms of parameters for end users, the expression language was changed from XPATH to SystemVerilog. All values, e.g., for parameters, base addresses, offsets, widths, sizes, and so on, can be written as expressions. Yet another major feature is conditionality. Many elements, such as ports, registers, bus interfaces, and interconnections, have been made conditional, meaning their presence can depend on the value of a Boolean expression in terms of the earlier mentioned parameters. As a consequence of this, the semantic consistency rules have been categorized into a set that can be checked before all expression values have been elaborated and a set can be checked only after all expression values have been elaborated. Finally, the TGI has been extended to cover the complete XML schema. Earlier versions of the standard provide limited functionality in the TGI in the sense that the functionality does not cover the complete XML schema. So, there were limitations with respect to editing of XML documents. The latest version IEEE 1685-2014 supports full XML document editing.

IEEE Std. 1685-2022 is the latest revision of the standard. A new top-level element named **typeDefinitions** was introduced to describe definitions for memory map related elements such as registers and address blocks. These type definitions can be (re-)used in components. Furthermore, the elements for component remap states and alternate groups have been replaced by more generic elements for component modes which describe the operating modes of a component. The memory remapping and access towards the register field bits depend on the component modes. For this reason, the definition of the access element has changed and the access of register field bits needs to be computed using the access elements of the encapsulating memory map elements. Also, access elements support a new value **no-access**. Another feature related to memory maps is multi-dimensional array and stride support for address blocks, register files, registers, and register fields. Also, index variables have been introduced into arrays which can be used to reference specific array elements in access handles, descriptions, short descriptions, and display names. Register field aliasing and broadcasting can now be described. In the area of connectivity, some new features have been added also. First, connectivity based on structs, unions, and SystemVerilog interfaces is now supported with **structured** ports. This is a new type of ports in addition to **wire** and **transactional** ports. Second, analog- and mixed-signal connectivity is

Copyright © 2023 Accellera Systems Initiative. All rights reserved.

supported with **signalType** and **domainType** descriptions in wire ports. Finally, abstraction definition ports have been extended with the Boolean element **match** that indicates that logical ports shall be connected on all ends of an interconnection, and a new Boolean attribute **allBits** that indicates that all logical bits shall be connected. A completely new feature in this revision of the standard is the description of power domains in components and component ports and the binding of power domains in design component instances. For TGI, REST is now also a standard transport layer in addition to SOAP. Conditional elements have been removed in this revision.

Copyright © 2023 Accellera Systems Initiative. All rights reserved.