2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

IP-XACT Demystified

An In-Depth Training on the IEEE 1685-2022 IP-XACT Standard

accellera
SYSTEMS INITIATIVE

IP-XACT is an XML-based IEEE standard that defines a standardized schema for Packaging, Integrating, and Reusing IP within Tool Flows

The Spirit Consortium

- 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

Accellera Systems Initiative

- IEEE Std. 1685-2014 June 2014
- IEEE Std. 1685-2022 September 2022

Target Audience

- IP developers and integrators
- IP providers
- Verification engineers
- Tool developers
- Startup companies
- Graduate students and researchers
- New users or users of earlier IP-XACT versions

What Problems We Solve

- IP packaging and configuration
- SoC connectivity and assembly
- Design data exchange across teams and tools
- Auto-generation of TB (incl UVM RAL)
- Auto-generation of documentation and metadata collaterals
- Memory maps and registers
- Vendor extensibility

Outline

- Connectivity use cases & tool flows
- Connectivity IP-XACT documents

- Registers use cases
- Registers IP-XACT documents

- Low power and other topics

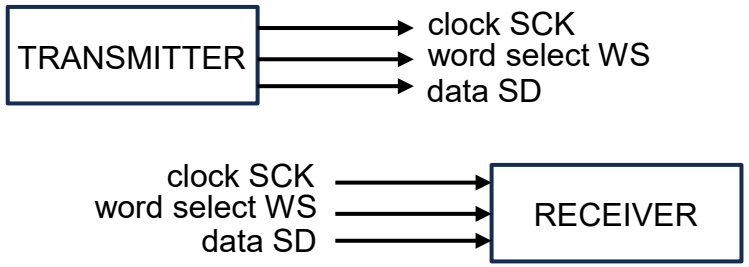
Connectivity
Use Cases & Tool Flows

Example Use Case: Top-Level Autogeneration

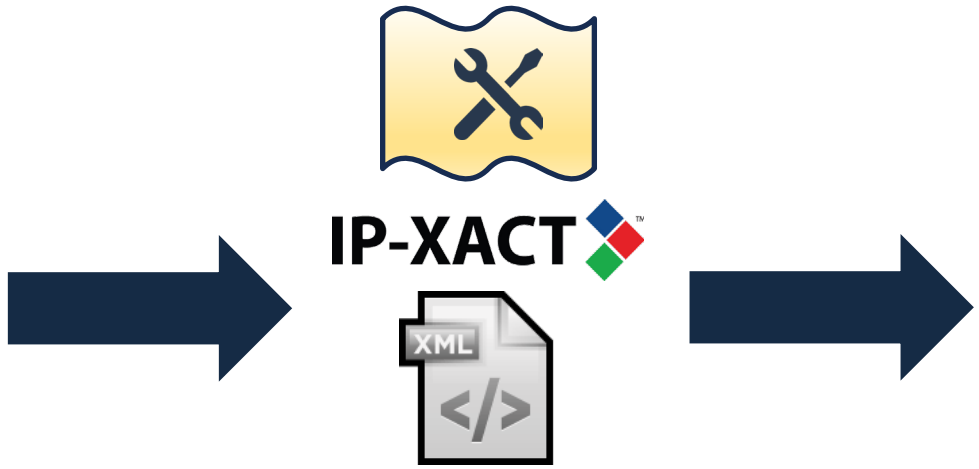
Tool Input

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

module receiver (sck, ws, sd);
  input wire sck;
  input wire ws;
  …
```



IP-XACT Tool



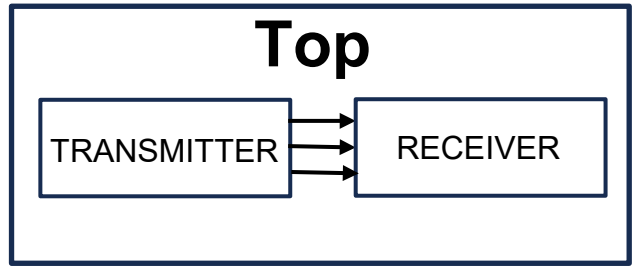
TGI or tool-specific APIs

```
tgi.createDesign()
tgi.addComponent...()
…
```

```
module top ();
  wire sck;
  wire ws;
  wire sd;

  transmitter u_transmitter
          (sck, ws, sd);

  receiver u_receiver
          (sck, ws, sd);
endmodule
```

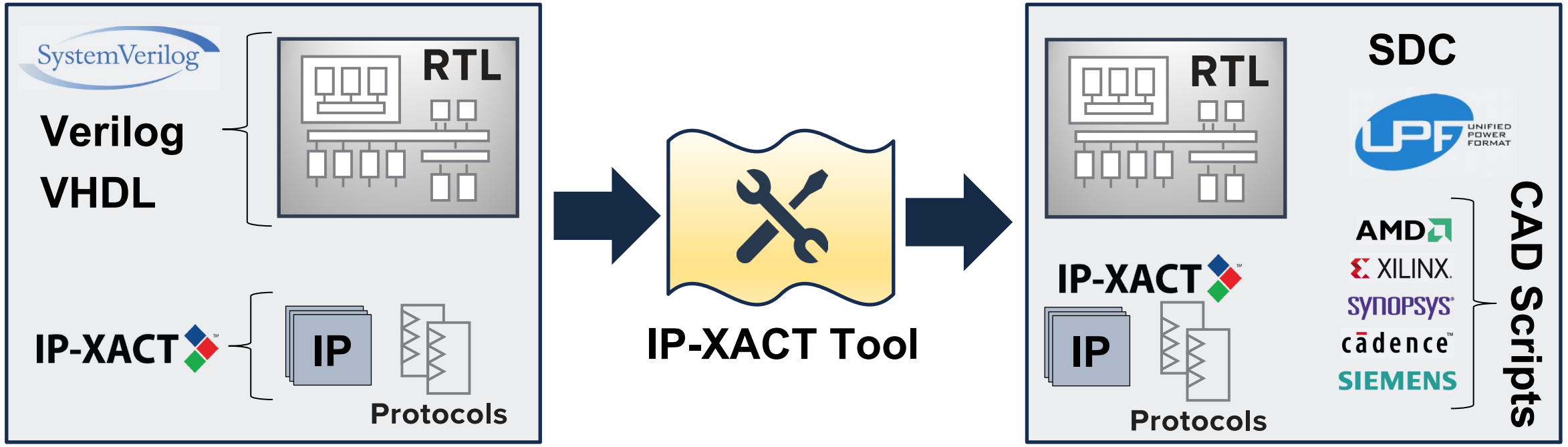


Overview of Connectivity Use Cases

- Vendors package their RTL into IP-XACT
- Exchange their IPs using this standardized format
- Integrate IPs back into RTL through automated IP-XACT tool flows
- Additionally: RTL hierarchy transformations, UPF, CAD Script, …

→ ***Consistent and reusable SoC assembly reduces manual effort, eliminates integration errors, and accelerates design cycles.***

Overview of Connectivity Use Cases

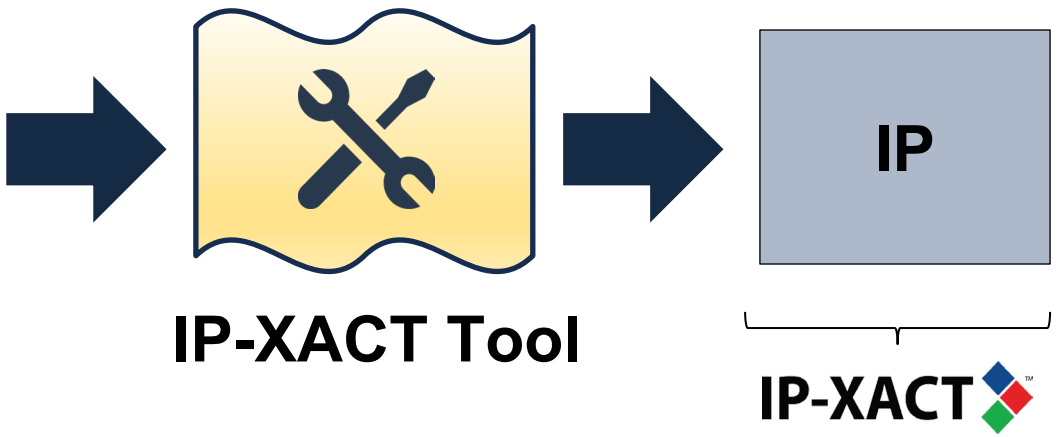


Source	Transform	Output	Meaning
RTL	→	IP-XACT	IP Packaging
IP-XACT	→	RTL	IP Integration
RTL	→	RTL	Hierarchy Transform (via IP-XACT)

IP Packaging: Generating IP-XACT Metadata

```
module top();
  wire sck, ws, sd;
  parameter my_param = 0;

// hier omitted in packaging
  transmitter u_transmitter
          (sck, ws, sd);
  receiver u_receiver
          (sck, ws, sd);
...
endmodule
```



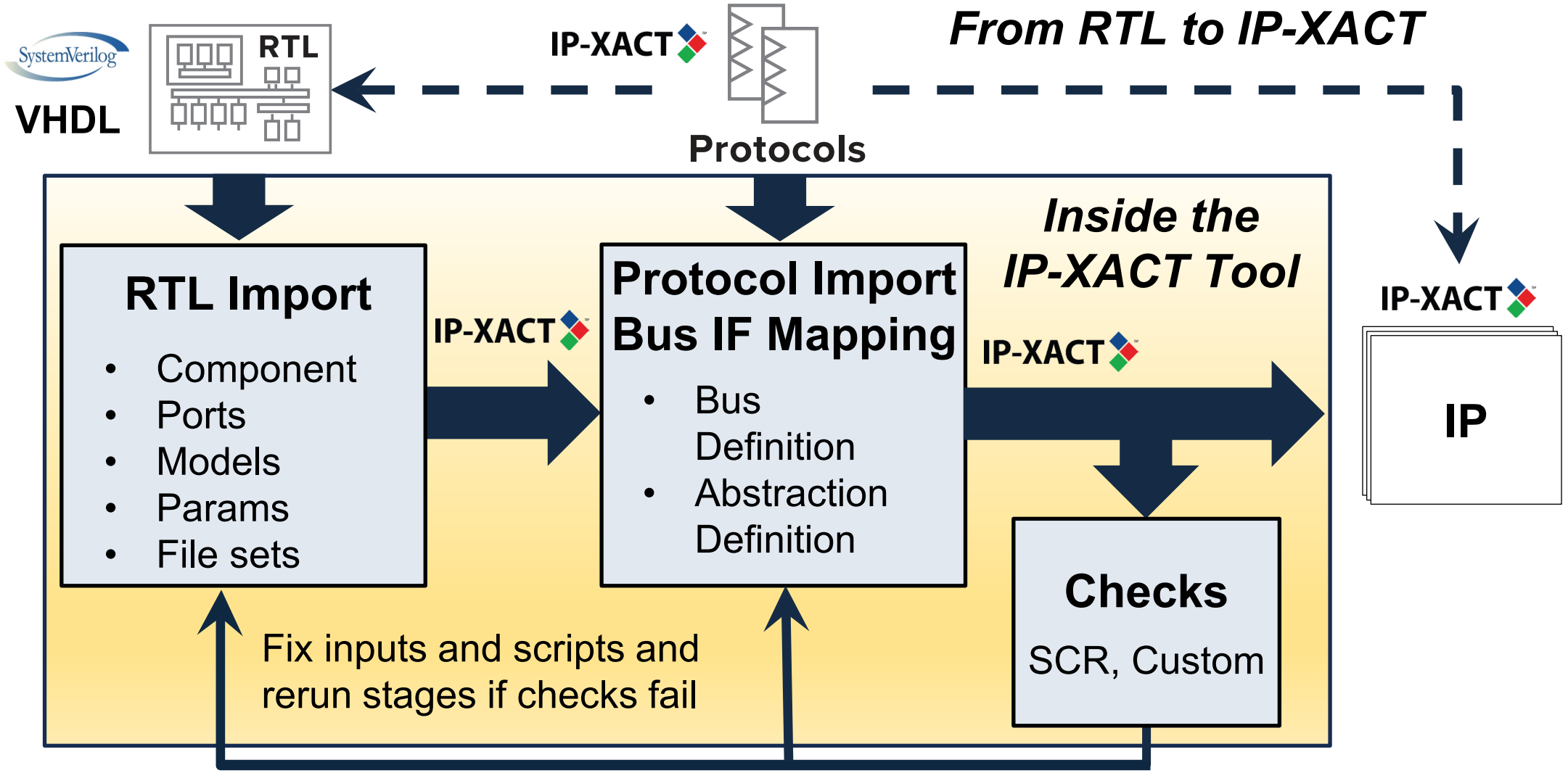
- Packaging generates IP-XACT for the **interface** of the IP
  - Used as a standard exchange format between organizations
  - Often called “Leaf Packaging”
- Packaging the **internal design hierarchy** is also possible
  - IP-XACT tools can use this to transform RTL hierarchies (covered in next slides)

IP-XACT Component Excerpt

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component …>
  <ipxact:vendor>accellera.org</ipxact:vendor>
  …
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>interface</ipxact:name>
        …
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:moduleName>initiator_transmitter…
        <ipxact:moduleParameters>
            <ipxact:name>my_param</ipxact:name>
            <ipxact:value>0</ipxact:value>
```

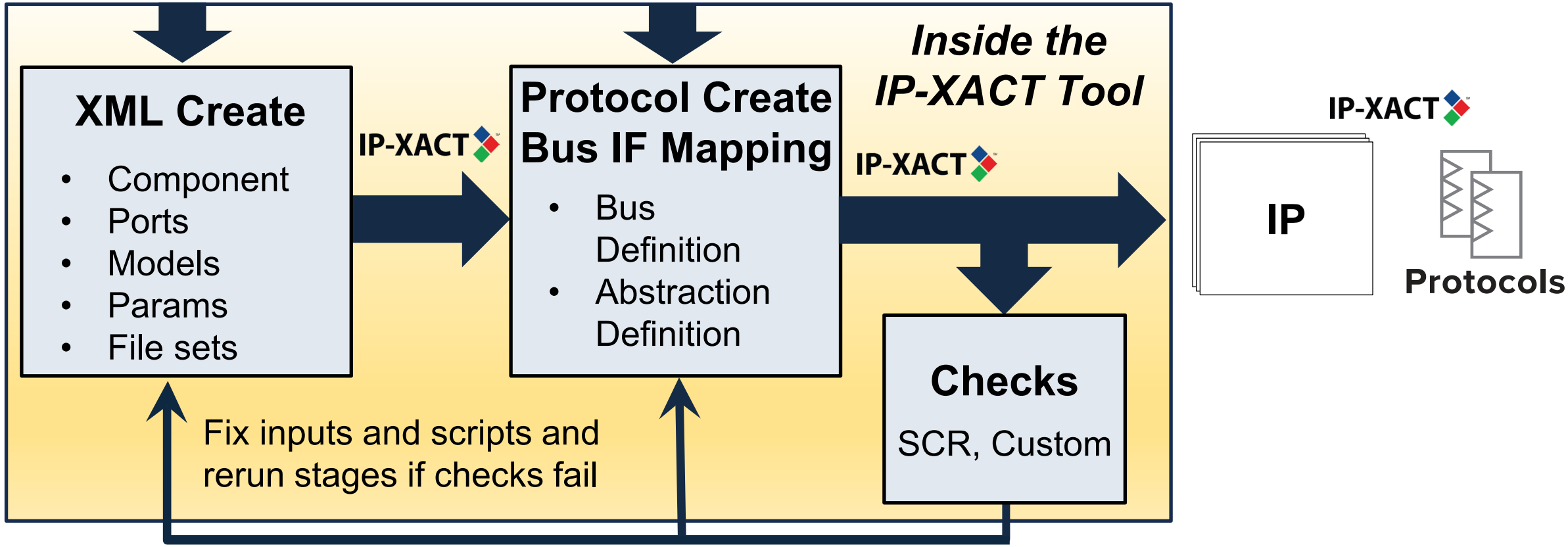
```
    <ipxact:ports>
      <ipxact:port>
        <ipxact:name>sck</ipxact:…>
        <ipxact:wire>
          <ipxact:direction>out</ipx…>
        </ipxact:wire>
    </ipxact:ports>
  </ipxact:model>
  <ipxact:fileSets>
    <ipxact:fileSet>
      <ipxact:name>fs-interface</…>
      <ipxact:file>
        <ipxact:name>../src.v</…>
        <ipxact:fileType>verilogSource<
        …
      </ipxact:file>
    </ipxact:fileSet>
  </ipxact:fileSets>
</ipxact:component>
```

IP Packaging Tool Flow



IP Packaging Tool Flow

No RTL or XML input!
The tool generates IP-XACT internally using APIs

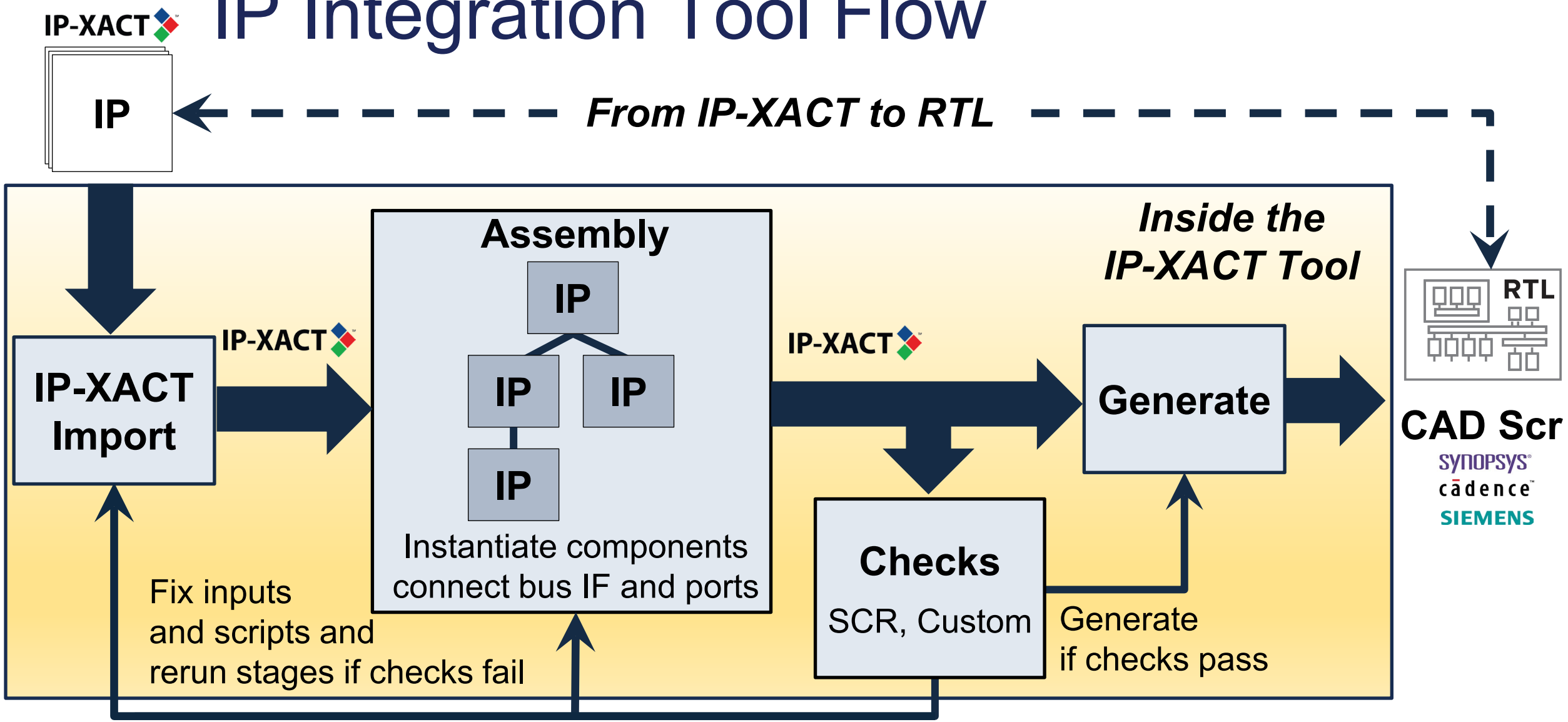


IP Integration With IP-XACT

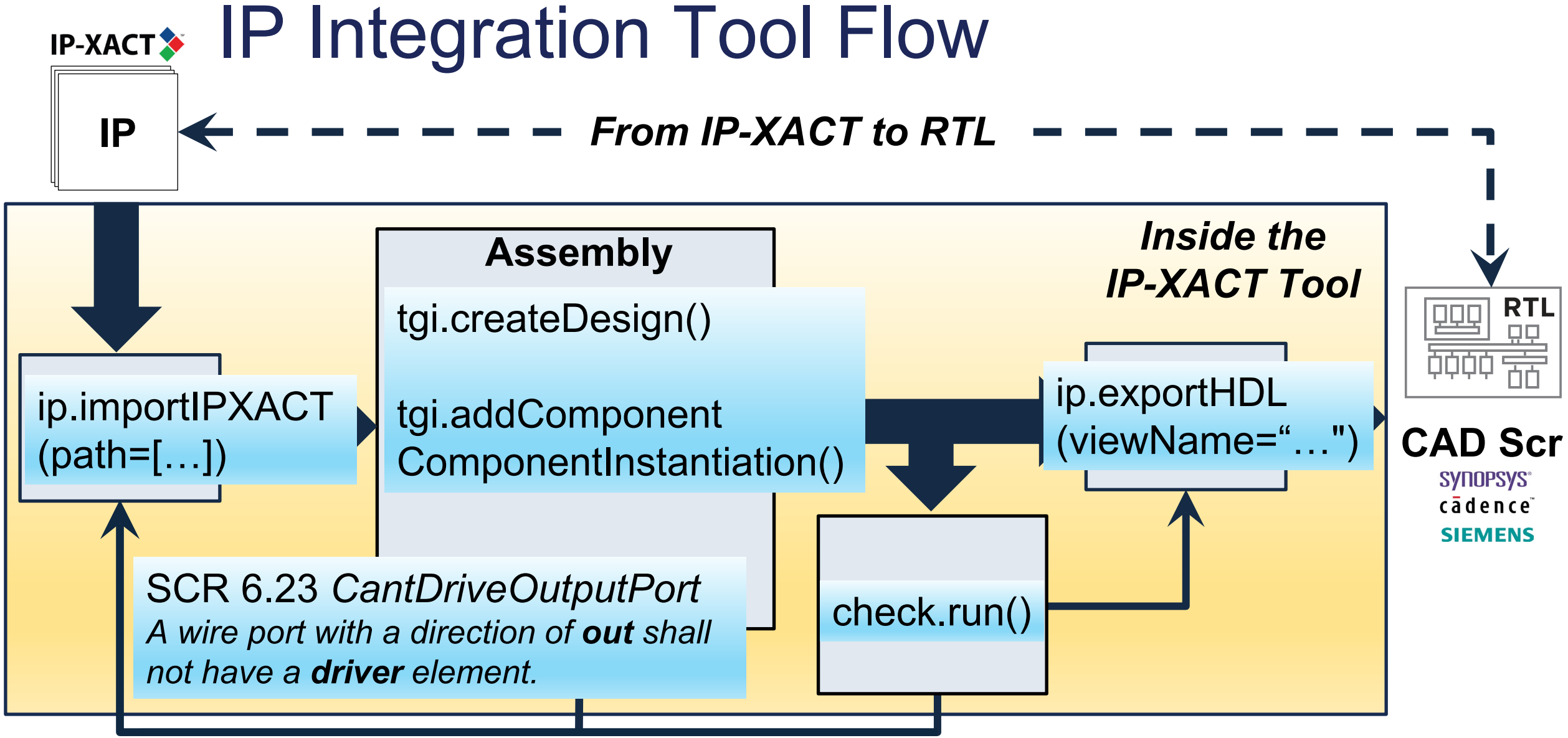
What is IP Integration?

- The process of assembling multiple IP blocks into a system or subsystem
- Requires consistent interface descriptions, parameterization, and connectivity
- Traditionally manual, error-prone, and time-consuming

IP Integration Tool Flow



IP Integration Tool Flow



RTL Hierarchy Transformations

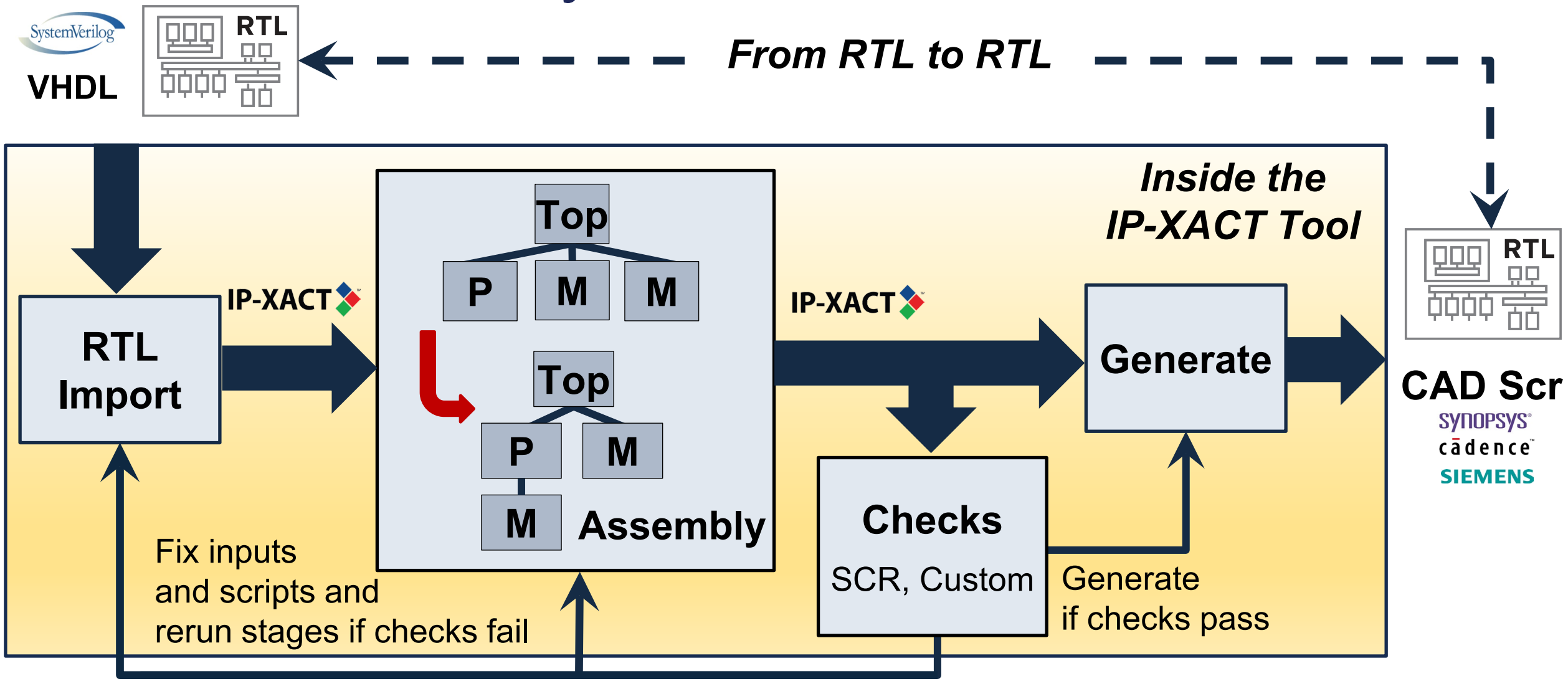
What is RTL Hierarchy Transformation?

- A structural rewrite of RTL

- Common use cases include:
  - Flattening or deepening hierarchy
  - Reorganizing modules
  - Regrouping blocks into new top-levels
  - Inserting generated modules (wrappers, etc.)
  - Cleaning or normalizing inconsistent RTL structures

- IP-XACT is used as the “structured database”

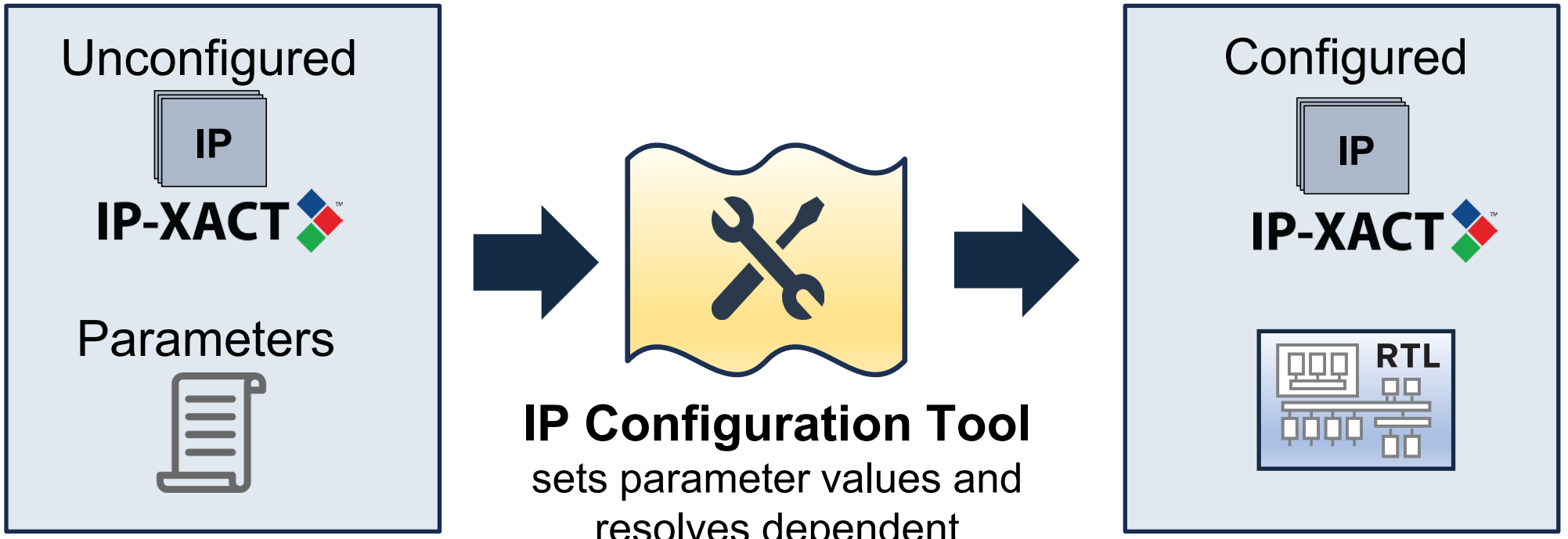
RTL Hierarchy Transformations



Available IP-XACT Tools

Tool Vendor	Type
Agnisys IDS-Integrate, IDS-Batch	Commercial
Arm Socrates	Commercial
Arteris Magillem Connectivity Magillem Registers	Commercial
EDAUtils Baya, IP-XACT 1685 Solution RTL utilities, translators, and parsers	Commercial
Synopsys coreBuilder / coreAssembler / coreConsultant	Commercial
Tampere University of Technology Kactus	Open-Source

IP Configuration Use Case



- Parameters can describe:
  - Bus widths
  - FIFO depths
  - Feature enables/disables
  - …
- The configured IP-XACT can be used as input to IP Integration flows

IP-XACT Before and After Configuration

```
<ipxact:moduleParameter dataType="int" parameterId="WR_FIFO_DEPTH1"type="longint">
 <ipxact:name>WR_FIFO_DEPTH1</ipxact:name>
 <ipxact:value>(2*(2*(4+2)))</ipxact:value>
</ipxact:moduleParameter>
<ipxact:port>
<ipxact:name>port_in</ipxact:name><ipxact:vec
  <ipxact:left>1+0b100000*0+(18*WR_FIFO_DEPT
? 0b011 : ... )))) </ipxact:left>
  <ipxact:right>0</ipxact:right></ipxact:vector>
</ipxact:port>
<ipxact:port>
<ipxact:name>port_out</ipxact:name><ipxact:v
  <ipxact:left>1+0b100000*0+(18*WR_FIFO_DEPT
? 0b011 : ... )))) </ipxact:left>
  <ipxact:right>0</ipxact:right></ipxact:vector>
</ipxact:port>
```

```
<ipxact:name>port_in</ipxact:name>
   <ipxact:vector>
     <ipxact:left>12</ipxact:left>
     <ipxact:right>0</ipxact:right>
   </ipxact:vector>
</ipxact:port>
<ipxact:port>
  <ipxact:name>port_out</ipxact:…
   <ipxact:vector>
     <ipxact:left>7</ipxact:left>
     <ipxact:right>0</ipxact:right>
   </ipxact:vector>
```

Recap Quiz

1. What is the main purpose of using IP-XACT in connectivity use cases?

A. To simulate RTL

B. To standardize IP metadata and automate design assembly

C. To visualize timing diagrams

D. To replace HDL entirely

Recap Quiz

1. What is the main purpose of using IP-XACT in connectivity use cases?

A. To simulate RTL

B. To standardize IP metadata and automate design assembly

C. To visualize timing diagrams

D. To replace HDL entirely

2. True or False: IP-XACT packaging *must* always include internal hierarchy.

Recap Quiz

1. What is the main purpose of using IP-XACT in connectivity use cases?
A. To simulate RTL
B. To standardize IP metadata and automate design assembly
C. To visualize timing diagrams
D. To replace HDL entirely

2. True or False: IP-XACT packaging *must* always include internal hierarchy.

3. True or False: IP-XACT tools can generate RTL *from* IP-XACT design descriptions.

Connectivity IP-XACT 1685-2022 Documents

XML Basics and IP-XACT

XML element

The basic building block of an XML document. It has a start tag, content, and an end tag:

```
<tagName>content</tagName>
```

Elements contain elements, text, or attributes.

XML attributes

Elements have attributes that give extra information about elements:

```
<book id="B101" genre="Technology"/>
```

Hierarchy and nesting

XML elements form tree-like structures where parent elements are **containers** of child elements, which themselves can have children.

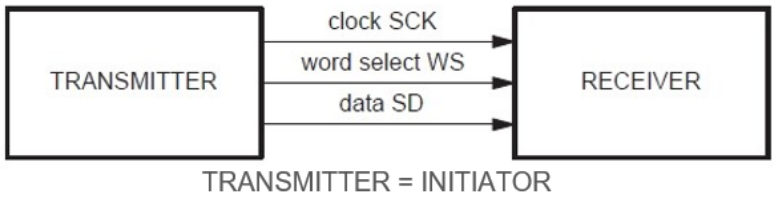
```
<book id="B101" genre="Technology">
   <title>XML Basics</title>
   <author>E. Hemingway</author>
   <publisher>TechPress</publisher>
   <year>2022</year>
</book>
```

IP-XACT is an XML-based standard (IEEE 1685-2022) that defines a standardized schema for describing IP blocks and designs.

IP-XACT Component Overview

../src/initiator_transmitter.v

```
module initiator_transmitter
                    (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



TRANSMITTER = INITIATOR

- An IP-XACT component is a fundamental top-level IP-XACT component
- 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.
- A component must document:
  - Interfaces of an IP
  - Parameters of an IP
  - Files that implement the IP
  - …
  - VLNV for identification
- All elements above are children of the component

IP-XACT Component Overview

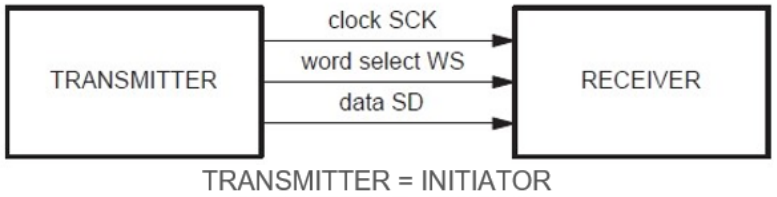
Identify the Comp

VLNV
vendor: accellera.org,
library: i2s,
name: initiator_transmitter,
version: 1.0

Component Container

../src/initiator_transmitter.v

```
module initiator_transmitter
                  (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



TRANSMITTER = INITIATOR

- **VLNV** is a child of the **component** container
- IP-XACT uses VLNV to identify any top-level object, not just components

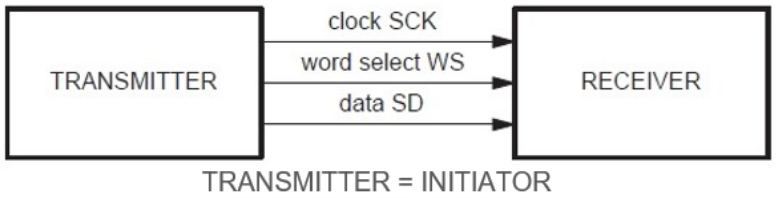
IP-XACT Component Overview

VLNV
vendor: accellera.org,
library: i2s,
name: initiator_transmitter,
version: 1.0

Configurable Properties
Optional, implementation-independent parameters passed around the hier.

../src/initiator_transmitter.v

```
module initiator_transmitter
                    (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



TRANSMITTER = INITIATOR

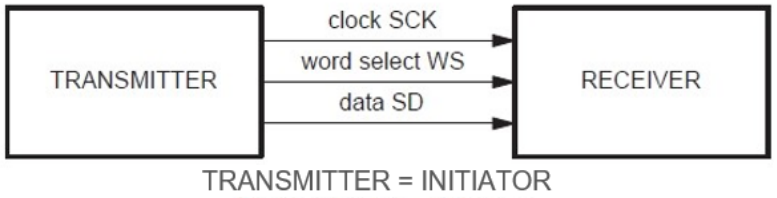
Component Container

- Another child of the component container is the **configurable parameters**
- Configurable parameters can be linked to RTL parameters **(my_param)** by using parameterId's
- IP Configuration tools only configure the configurable parameters

IP-XACT Component Overview

../src/initiator_transmitter.v

```
module initiator_transmitter
                    (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



VLNV
vendor: accellera.org,
library: i2s,
name: initiator_transmitter,
version: 1.0

Configurable Properties
Optional, implementation-independent parameters passed around the hier.

Bus Interfaces (covered later)

Capture module Intf

Ports
sck: out, wire
ws: out, wire
sd: out, wire

Model Container

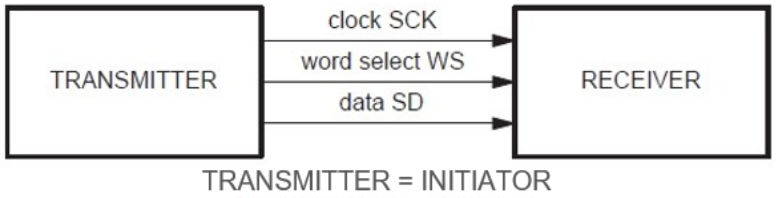
Component Container

- **Bus Interfaces** is child of **Component**
- **Model** is child of **Component**
- **Ports** is child of **Model**
- Ports can also be structured (SV struct, union, interface)

IP-XACT Component Overview

../src/initiator_transmitter.v

```
module initiator_transmitter
                    (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



TRANSMITTER = INITIATOR

VLNV
vendor: accellera.org,
library: i2s,
name: initiator_transmitter,
version: 1.0

Configurable Properties
Optional, implementation-independent parameters passed around the hier.

Bus Interfaces (covered later)

Model Container

Ports
sck: out, wire
ws: out, wire
sd: out, wire

Component Container

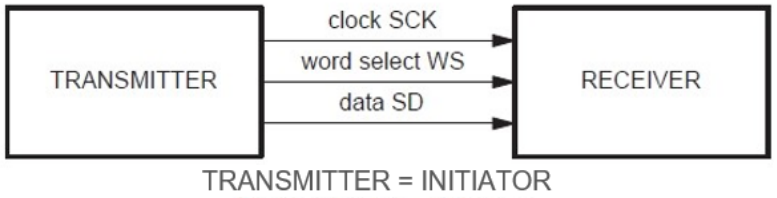
Views element
binds to a specific implementation
(interface, behavioral, synthesis, etc)

Views and **Ports** are children of **model** container

IP-XACT Component Overview

../src/initiator_transmitter.v

```
module initiator_transmitter
                    (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



VLNV
vendor: accellera.org,
library: i2s,
name: initiator_transmitter,
version: 1.0

Configurable Properties
Optional, implementation-independent parameters passed around the hier.

Bus Interfaces (covered later)

Model Container

Ports
sck: out, wire
ws: out, wire
sd: out, wire

Component Container

Implementation Binding

view → componentInstantiation - - - - → Impl-specific RTL params (my_param)

hdl_interface ↓ fs_interface

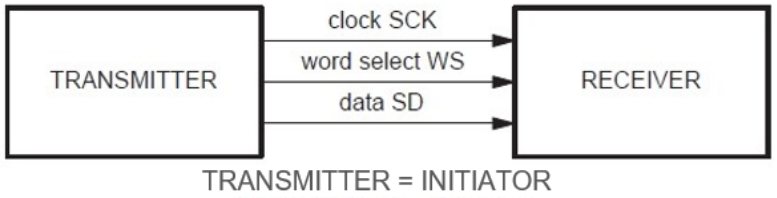
(../src/initiator_transmitter.v) fileSet

IP-XACT tools utilize **fileSets** to generate CAD scripts

IP-XACT Component Overview

../src/initiator_transmitter.v

```
module initiator_transmitter
                    (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



VLNV
vendor: accellera.org,
library: i2s,
name: initiator_transmitter,
version: 1.0

Configurable Properties
Optional, implementation-independent parameters passed around the hier.

Bus Interfaces (covered later)

Model Container

Ports
sck: out, wire
ws: out, wire
sd: out, wire

Component Container

viewRef

Implementation Binding

view → componentInstantiation - - - - - → Impl-specific RTL params (my_param)

hdl_interface | fs_interface

(../src/initiator_transmitter.v) fileSet

Ports reference the view(s) on which they apply.
If no viewRef, applies to all views.

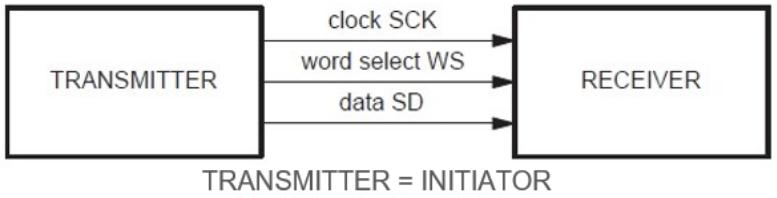
Component XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component …>
  <ipxact:vendor>accellera.org</ipxact:vendor>
  …
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>interface</ipxact:name>
        …
    </ipxact:views>
    <ipxact:instantiations>
      <ipxact:componentInstantiation>
        <ipxact:moduleName>initiator_transmitter…
        <ipxact:moduleParameters>
            <ipxact:name>my_param</ipxact:name>
            <ipxact:value>0</ipxact:value>
```

```
      <ipxact:ports>
        <ipxact:port>
          <ipxact:name>sck</ipxact:…>
          <ipxact:wire>
            <ipxact:direction>out</ipx…>
          </ipxact:wire>
      </ipxact:ports>
    </ipxact:model>
    <ipxact:fileSets>
      <ipxact:fileSet>
        <ipxact:name>fs-interface</…>
        <ipxact:file>
          <ipxact:name>../src.v</…>
          <ipxact:fileType>verilogSource<
          …
        </ipxact:file>
      </ipxact:fileSet>
    </ipxact:fileSets>
</ipxact:component>
```

Hierarchical Component

```
module initiator_transmitter
                  (sck, ws, sd);
  output wire sck;
  output wire ws;
  output wire sd;
  parameter my_param = 0;
endmodule
```



VLNV
vendor: accellera.org,
library: i2s,
name: initiator_transmitter,
version: 1.0

Configurable Properties
Optional, implementation-independent parameters passed around the hier.

Bus Interfaces (covered later)

Ports
sck: out, wire
ws: out, wire
sd: out, wire

Model Container

Component Container

viewRef

Implementation Binding

view → componentInstantiation - - - - - → Impl-specific RTL params (my_param)

hdl_interface

fs_interface

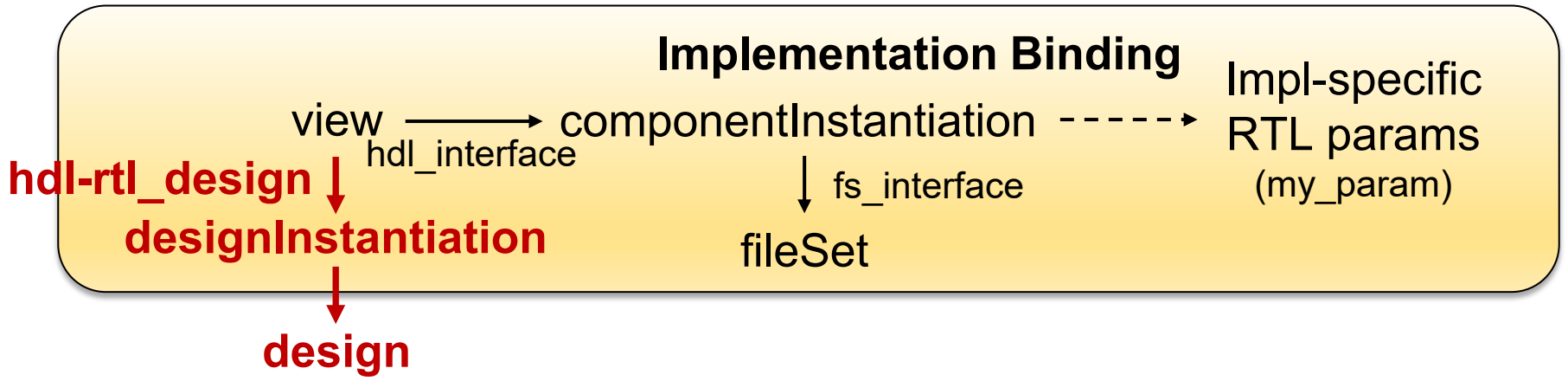
(../src/initiator_transmitter.v) fileSet

...

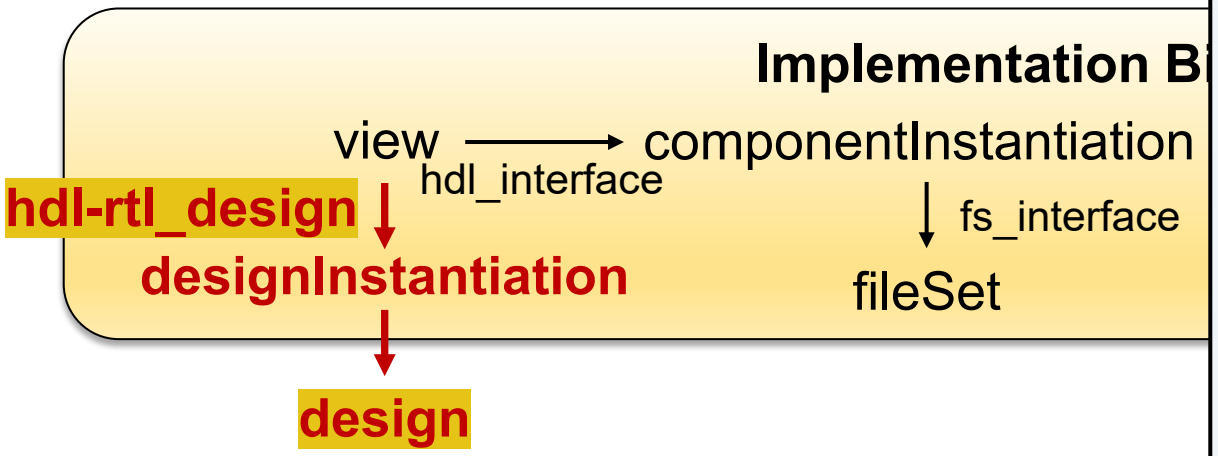
Design container captures internal RTL structure

Hierarchical vs Leaf Components

- **Leaf Component**: A component that **does not contain** any views that reference an IP-XACT design
- **Hierarchical Component**: A component that has one or more views that reference an IP-XACT design description



Hierarchical Component Reference to Design



- A component can declare many designInstantiation's, but a view can only select one instantiation at a time
- transmitter_is_initiator is the top component that instantiates initiator_transmitter

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:component …>
  <ipxact:vendor>accellera.org</i…>
  <ipxact:library>i2s</ipxact:library>
  <ipxact:name>transmitter_is_initiator</…
  <ipxact:version>1.0</ipxact:version>
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:designInstantiationRef>
                  hdl-rtl_design </ipxact..
    …
    <ipxact:instantiations>
      <ipxact:designInstantiation>
        <ipxact:name>hdl-rtl_design</…>
        <ipxact:designRef vendor="accellera…"
            library="i2s"
            name="transmitter_is_initiator_rtl"
            version="1.0"/>
```

IP-XACT Design

```
module transmitter_is_initiator;
  wire u_init_xmitter_sck_sig;
  …
  initiator_transmitter #(
  .my_param (1)
  )
  u_initiator_transmitter (
   .sck(u_init_xmitter_sck_sig),
   .ws(…),
   .sd(..)
);
endmodule
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design …>
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:name>transmitter_is_initiator_rtl</ipx…
  <ipxact:componentInstances>
    <ipxact:componentInstance>
      <ipxact:instanceName>u_initiator_transmitter<…
      <ipxact:componentRef vendor="accellera.org"
                           library="i2s"
                           name="initiator_transmitter"
                           version="1.0">
      </ipxact:componentRef>
  <ipxact:interconnections>
    <ipxact:interconnection>
      <ipxact:name>u_init_xmitter_sck_sig</ipx…
      …
</ipxact:design>
```

Example component of previous slides

IP-XACT Design

```
module transmitter_is_initiator;
  wire u_init_xmitter_sck_sig;
  …
  initiator_transmitter #(
  .my_param (1)
  )
  u_initiator_transmitter (
   .sck(u_init_xmitter_sck_sig),
   .ws(…),
   .sd(..)
);
endmodule
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ipxact:design …>
 <ipxact:vendor>accellera.org</ipxact:vendor>
 <ipxact:name>transmitter_is_initiator_rtl</ipx…
 <ipxact:componentInstances>
  <ipxact:componentInstance>
       <spirit:configurableElementValues>
          <spirit:configurableElementValue
             spirit:referenceId="my_param">1
          </spirit:configurableElementValue>
       </spirit:configurableElementValues>
</ipxact:design>
```

Recap Quiz

1. What is the VLNV, and why is it important?

A. A type of netlist
B. A unique identifier for any IP-XACT object
C. A verification library
D. A synthesizable template

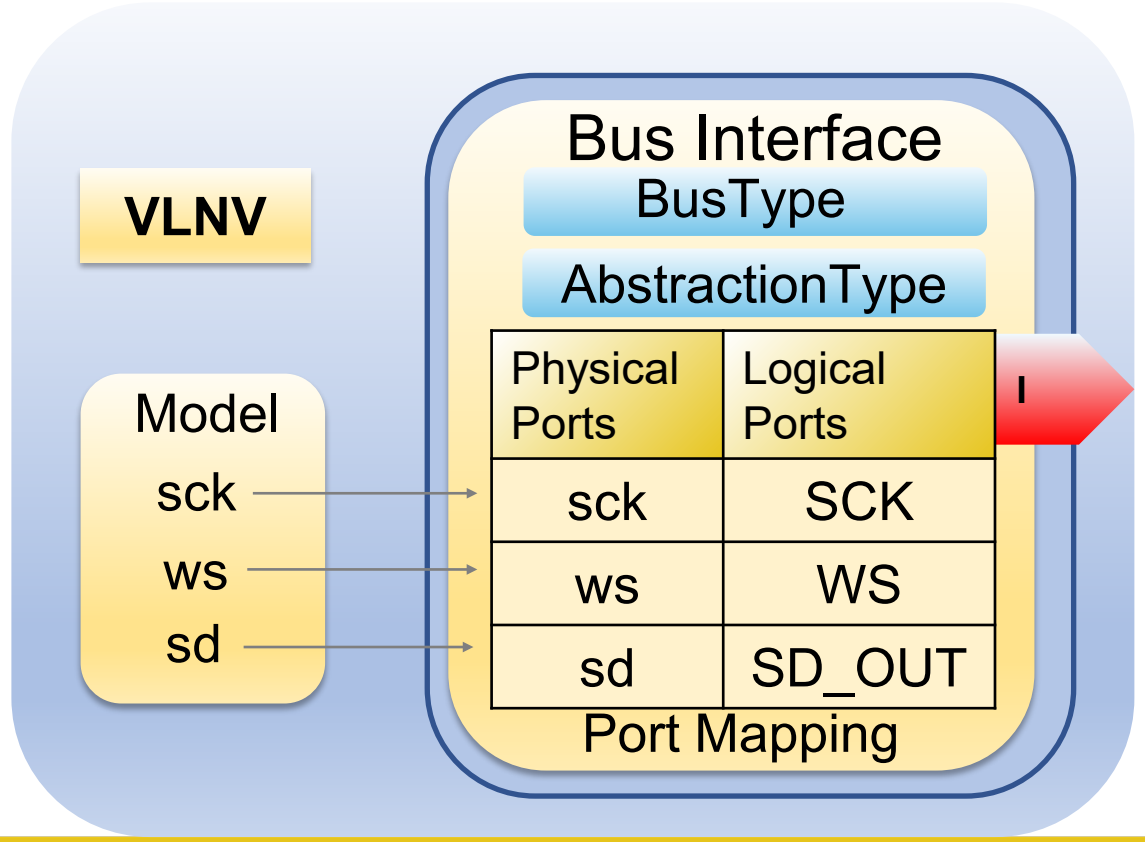
2. What does the “model” element of an IP-XACT component contain?

A. Ports, views, parameters, and file sets
B. Power grid data
C. Memory maps only
D. Only the VLNV

Bus Interfaces

Component Bus Interface

ipxact **component** **busInterfaces** **busInterface**



- A busInterface has a name element, which is used to specify name of the interface
  - A busInterface has a **busType and abstractionType** referencing a **busDefinition and abstractionDefinition** using the VLNV
  - It defines a memory map associated with the interface (if target).
  - Provides mapping between physical ports and logical ports.
  - The bus interface has 7 modes.

Bus Interface Modes

- The Bus Interface has 7 modes as listed below

1. Initiator	2.Target	3.Mirrored Initiator
4. Mirrored Target	5. System	6. Mirrored System
7. Monitor		

- **Initiator:** An initiator interface is one that initiates transactions.
- **Target:** A target interface is one that responds to transactions. It has a memory map reference that points to information about the range of registers, memory, or other address blocks accessible through this target interface.
- For more details about the different bus interface modes, you can refer to section 6.7.4 Interface modes in the IP-XACT 2022 Standard.

Bus Definition

ipxact **busDefinition**

directConnection
Connection types:
True = direct initiator-target;
False = Target– mirrored
Target/initiator- mirroredInitiator

I2S

VLNV

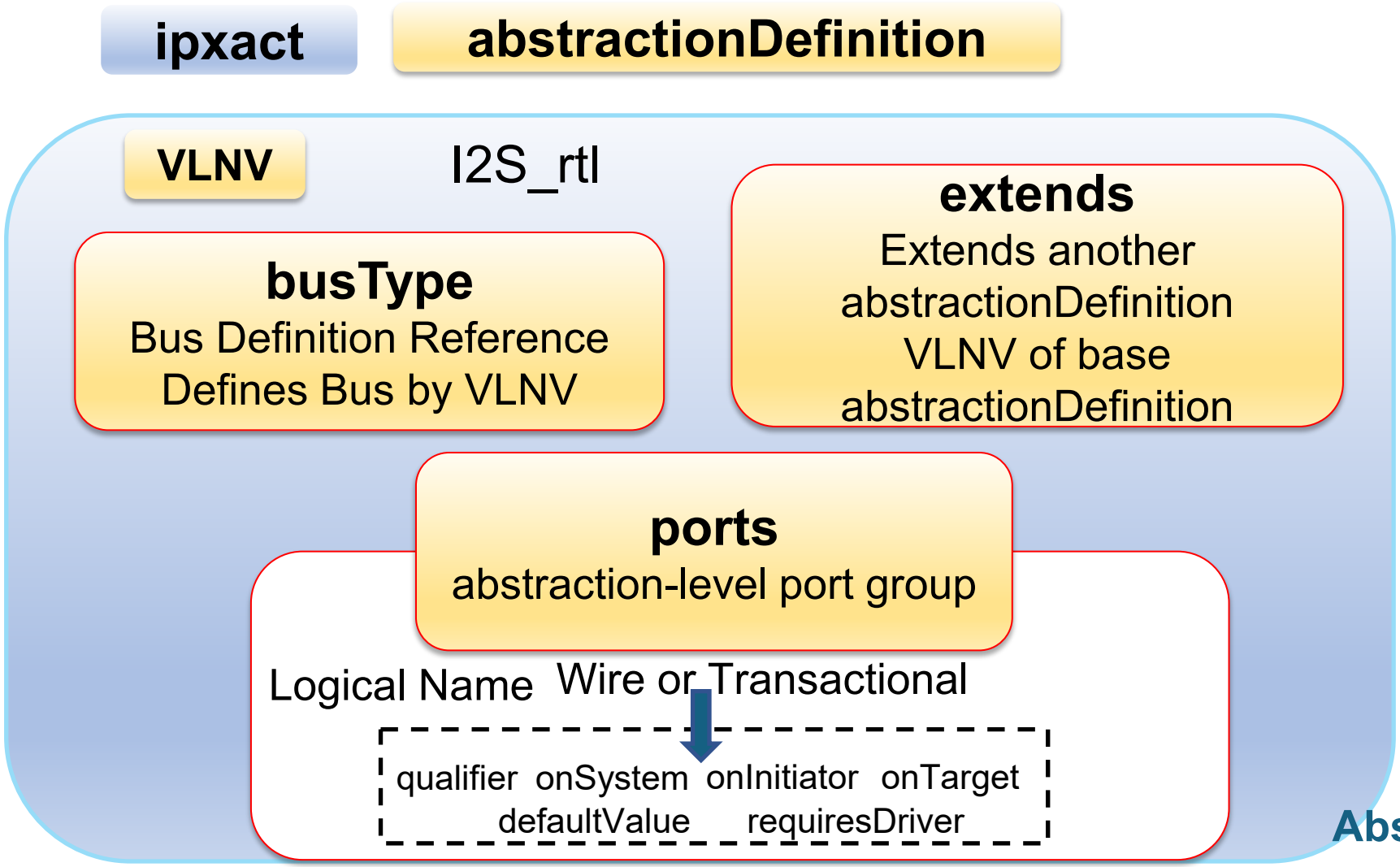
extends
Extends another
busDefinition?
VLNV of base busDefinition

isAddressable
Addressing Info:
True = interfaces + memory map;
False = no addressing

- It defines interface types for hardware communication protocols and high-level aspects
- Additional elements are **directConnection and isAddressable**.
- It can be extended from another bus definition

Bus Definition Container

Abstraction Definition

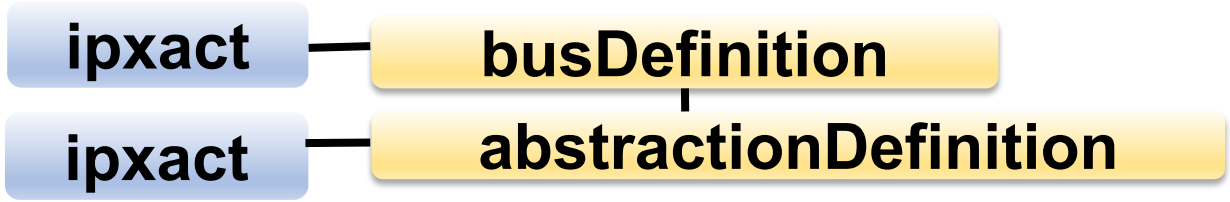


IP-XACT Abstraction Definition contains

- **wirePort** elements inside onSystem, onInitiator, or onTarget within a wire
- **wirePort** details port appearance in busInterface, including width and direction

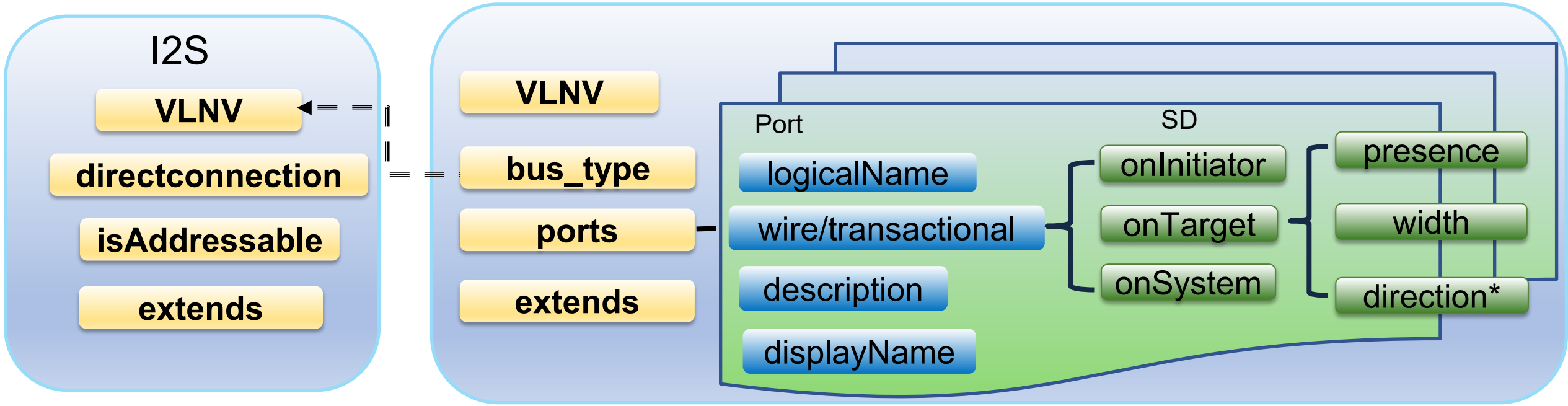
Abstraction Definition Container

Abstraction and Bus Definition



Bus Definition

Abstraction Definition



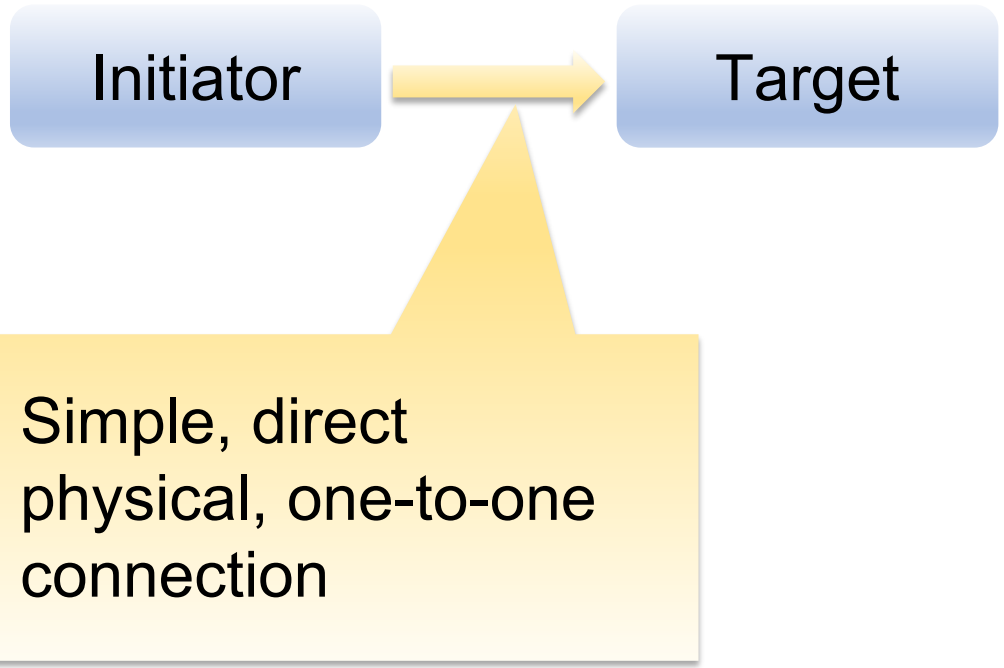
*direction: The three possible values are in, out, or inout.(default value is out.)

a) **allLogicalDirectionsAllowed** (optional; type: *boolean*; default: **false**) attribute defines whether the port may be mapped to a port in an **abstractionDefinition** with a different direction. See 5.3.

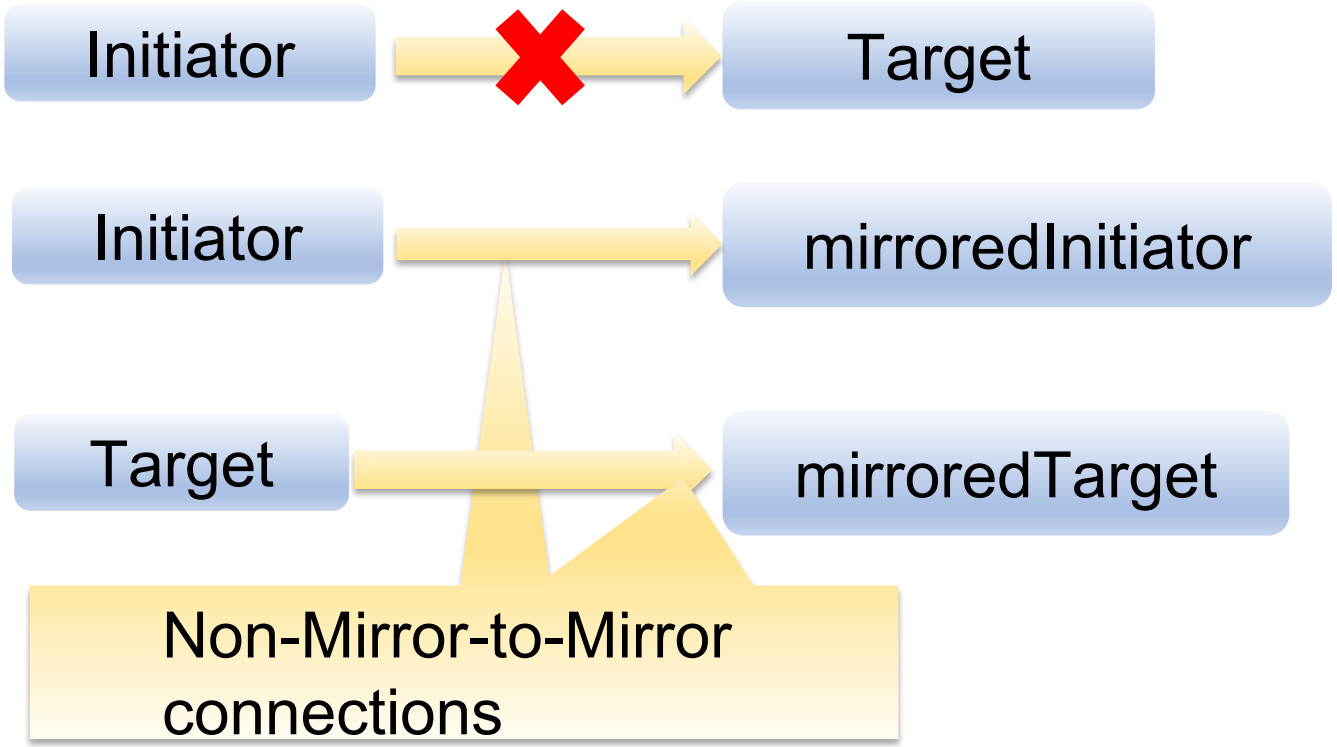
Bus Definition

ipxact **busDefinition** **directconnection**

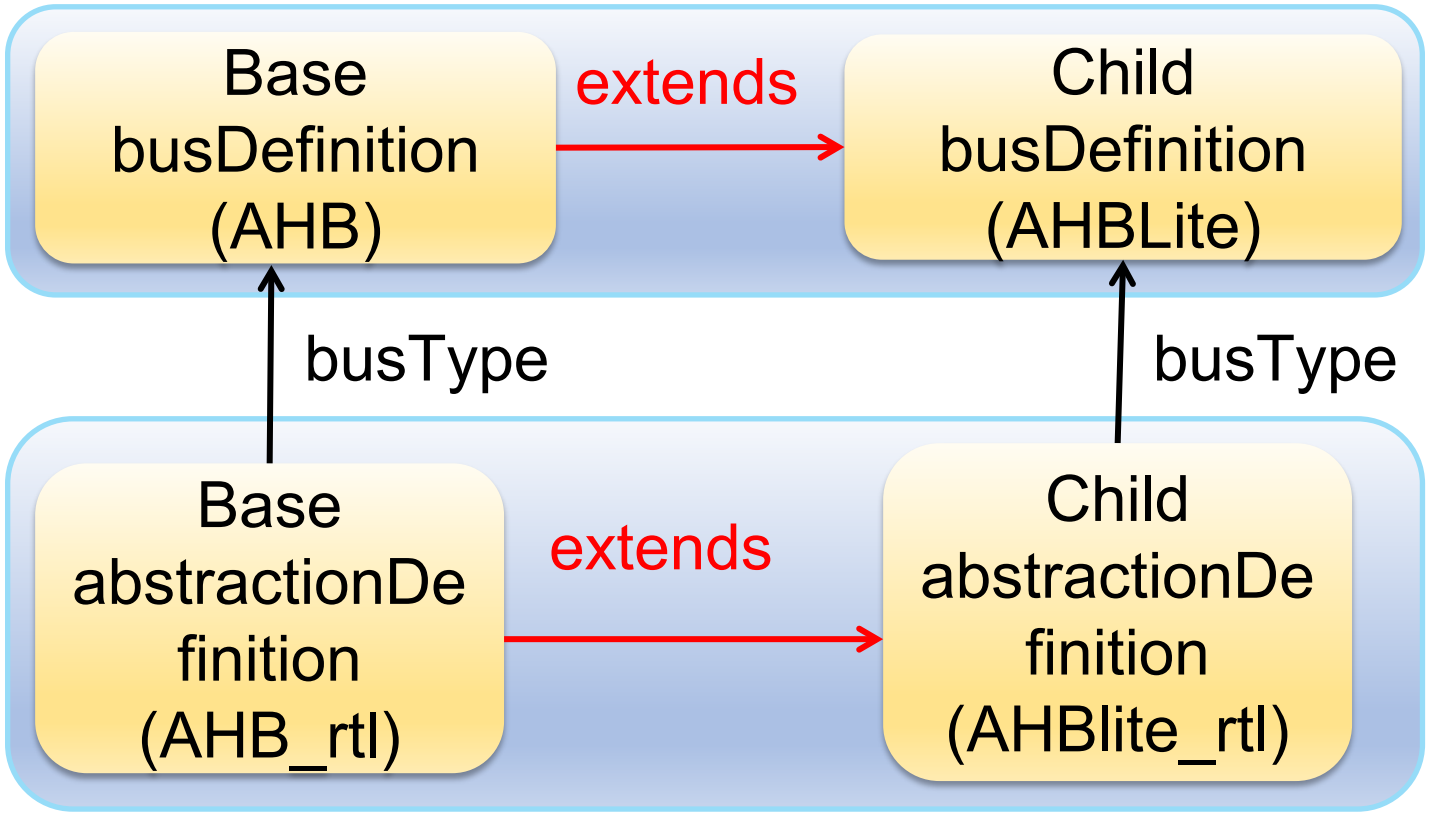
directconnection: True



directconnection: False



Extends in Abstraction and Bus Definition



- Bus definitions can extend another bus
- Child bus inherits compatibility from base bus
- Only same family buses can connect
- Extending bus must redefine everything
- Each bus has a matching abstraction
- Child abstraction extends base abstraction

Recap Quiz

1. What is the VLNV, and why is it important?

A. A type of netlist
B. A unique identifier for any IP-XACT object
C. A verification library
D. A synthesizable template

2. What does the “model” element of an IP-XACT component contain?

A. Ports, views, parameters, and file sets
B. Power grid data
C. Memory maps only
D. Only the VLNV

Connectivity Topics We Did Not Cover

1. TLM

…

Connectivity Topics We Did Not Cover

1. TLM

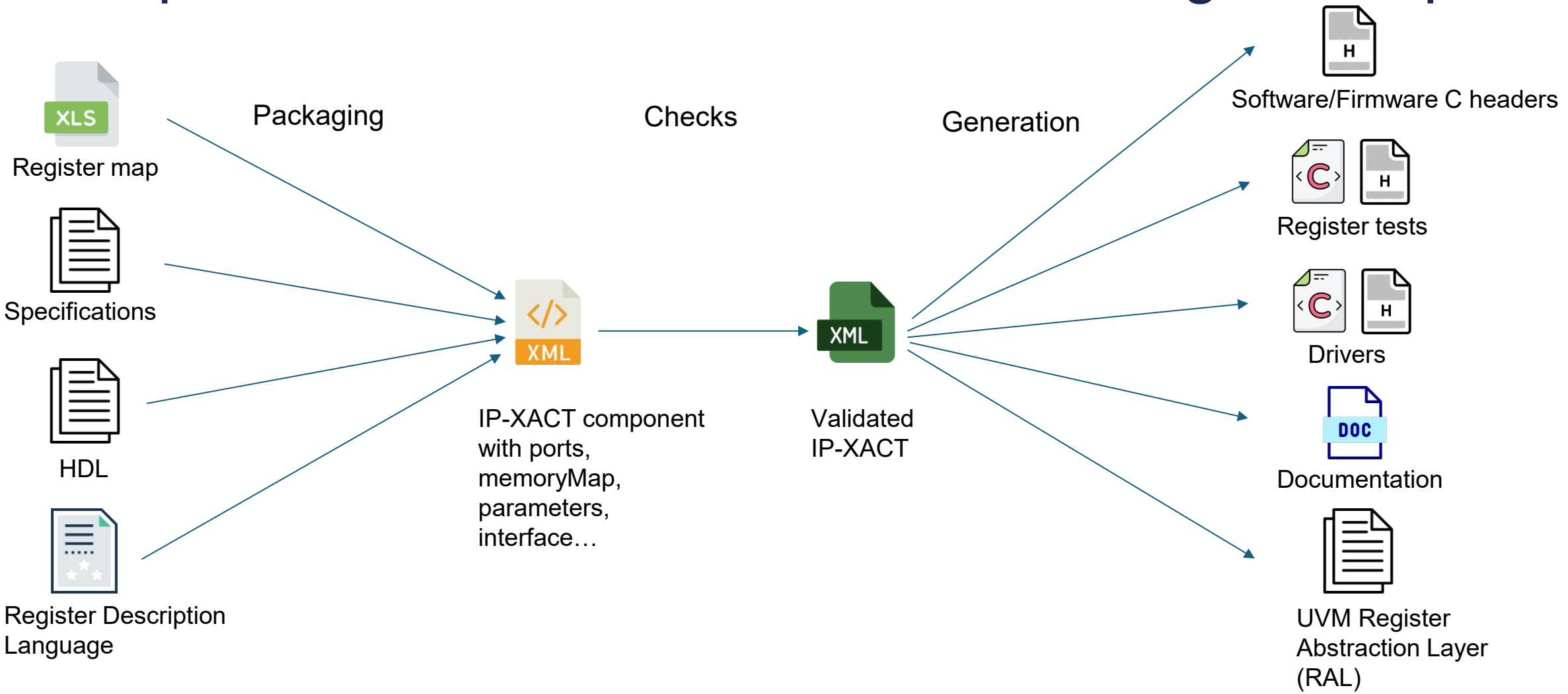
…

Hardware/Software Interface (HSI)

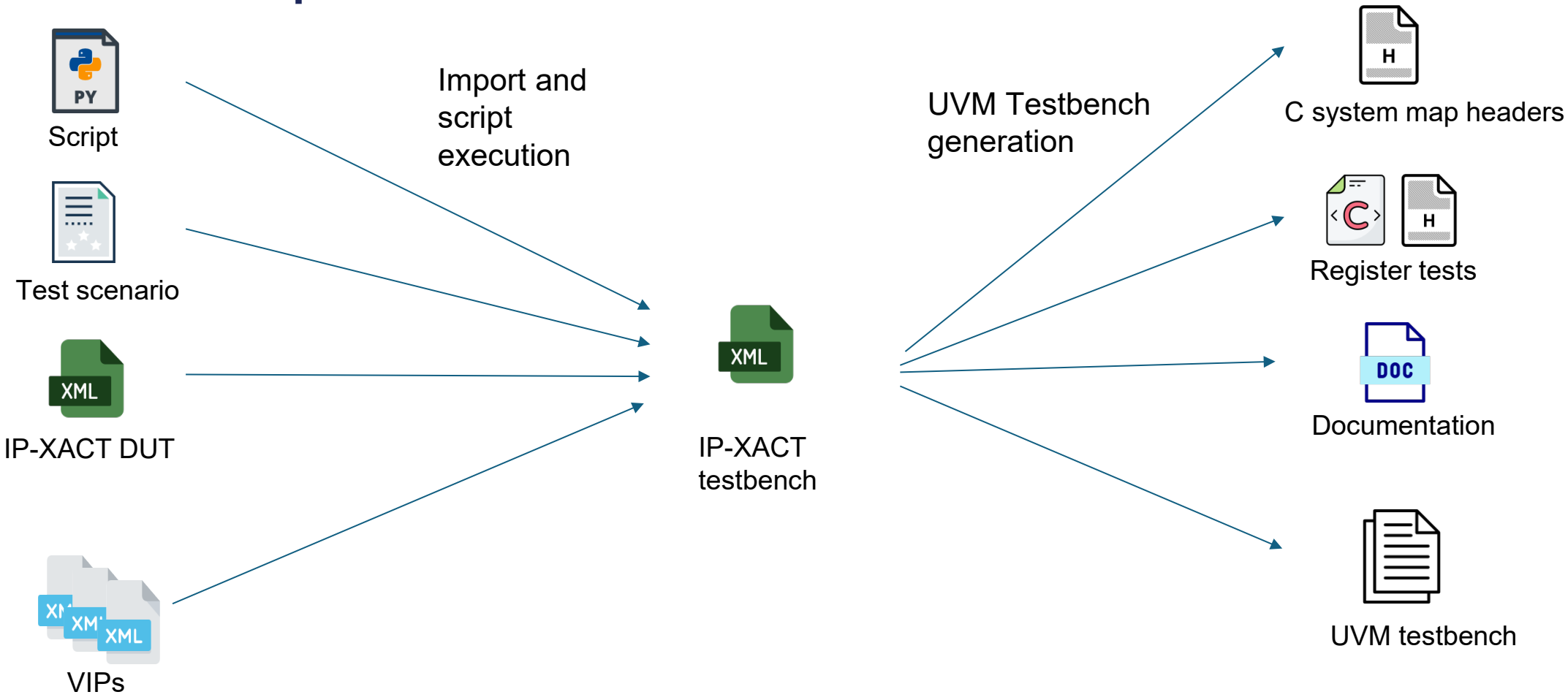
What is HSI ?

- HSI defines how the SW talks to the HW, for example:
  - Peripheral configuration and functionality
  - CPUs and peripherals interactions
- IP-XACT supports HSI:
  - At IP level: memory map
  - At system level: system map
- IP-XACT enables:
  - For IP providers: standard definition of memory map
  - For architects: standard definition of system map
  - From these definitions: automatic generation of firmware, drivers, verification environment, documentation…

Example: automated IP Control / Status register exports



Example: automated verification flow



Focus on typeDefinitions and accessHandles

Feature	New in 2022?	Enhanced vs 2014 ?	Benefits
typeDefinitions	✓		Factorize address map info in common file: - Increase reuse - Reduce XML and generated file size - Reduce manual errors when capturing address map
accessHandles		✓	Mapping of IP-XACT register elements to HDL internal signals: - More robust solution vs IP-XACT 2014: support views, arrays, slices, expressions - Improved generation of UVM RAL from IP-XACT

What is "typeDefinitions"?

- New root file in IP-XACT 2022
- Central placeholder for memory-related object metadata:
  - Field access policies
  - Enumerations
  - Fields, registers, register files, address block, bank
  - Memory maps and memory remaps…
- Configurable using SystemVerilog expressions
- Support dependencies on views, modes, and reset-types
- Component refers to typeDefinitions using externalTypeDefinitions element

Example: typeDefinitions

Unique identifier of typeDefinitions (VLNV)

List of modes

bitField width is configured by FIELD_WIDTH parameter with id = FIELD_WIDTH_id

myControlField bitField definition

Mode-specific access:
- read-write in debugMode
- write-only otherwise

```
<ipxact:typeDefinitions>
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>tutorial</ipxact:library>
  <ipxact:name>fieldDefinitions</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:modes>
    <ipxact:mode>
      <ipxact:name>debugMode</ipxact:name>
    </ipxact:mode>
    …
  </ipxact:modes>
  <ipxact:fieldDefinitions>
    <ipxact:fieldDefinition>
      <ipxact:name>myControlField</ipxact:name>
      <ipxact:bitWidth>FIELD_WIDTH_id</ipxact:bitWidth>
      …
      <ipxact:fieldAccessPolicies>
        <ipxact:fieldAccessPolicy>
          <ipxact:access>write-only</ipxact:access>
        </ipxact:fieldAccessPolicy>
        <ipxact:fieldAccessPolicy>
          <ipxact:modeRef priority="1">debugMode</ipxact:modeRef>
          <ipxact:access>read-write</ipxact:access>
        </ipxact:fieldAccessPolicy>
      </ipxact:fieldAccessPolicies> …
```

Example: component

Instantiates and configures the referenced typeDefinitions in the component

Binds debugMode mode of typeDefinitions to DEBUG mode of the component

Use myControlField bitfield definition from fieldTypeWidth32 typeDefinitions instead of redefining all its properties

```
<ipxact:component>
 …
 <ipxact:typeDefinitions>
  <ipxact:externalTypeDefinitions>
   <ipxact:name>fieldTypeWidth32</ipxact:name>
   <ipxact:typeDefinitionsRef vendor="accellera.org" library="tutorial" name="fieldDefinitions" version="1.0">
    <ipxact:configurableElementValues>
     <ipxact:configurableElementValue referenceId="FIELD_WIDTH_id">32</ipxact:configurableElementValue>
    </ipxact:configurableElementValues>
   </ipxact:typeDefinitionsRef>
   <ipxact:modeLinks>
    <ipxact:modeLink>
     <ipxact:externalModeReference modeRef="debugMode"/>
     <ipxact:modeReference modeRef="DEBUG"/>
    </ipxact:modeLink>
   </ipxact:modeLinks>
 </ipxact:externalTypeDefinitions>
 …
 </ipxact:typeDefinitions>
 …
 <ipxact:memoryMaps>
  <ipxact:memoryMap>
     …
    <ipxact:field>
     <ipxact:name>sourceAddress</ipxact:name>
     <ipxact:bitOffset>0</ipxact:bitOffset>
     <ipxact:fieldDefinitionRef typeDefinitions="fieldTypeWidth32">myControlField</ipxact:fieldDefinitionRef>
    </ipxact:field>…
```

What is "accessHandles"?

- Define HDL path to the internal signals
- Complete HDL path is distributed across the memoryMap hierarchy
- Different types of access handles:
  - Simple accessHandle for the different memory map containers (bank, register file, register, alternateRegister)
  - Sliced accessHandle for the actual storage cells
  - Port accessHandles
- Can be configurable and view-specific
- Support multiple slices and arrays

Example: accessHandle

Access handle for a register array element

Access handle for a bitField array element

```
<ipxact:component>
  …
   <ipxact:register>
     <ipxact:name>RX_ERROR_Reg</ipxact:name>
     <ipxact:accessHandles>
       <ipxact:accessHandle force="true">
         <ipxact:slices>
          <ipxact:slice>
            <ipxact:pathSegments>
              <ipxact:pathSegment>"$sformatf("rx_error_reg_%0d",$ipxact_index_value(i)</ipxact:pathSegment>
            </ipxact:pathSegments>
          </ipxact:slice>
        </ipxact:slices>
       </ipxact:accessHandle>
     </ipxact:accessHandles>
     <ipxact:array>
       <ipxact:dim indexVar="i">8</ipxact:dim>
     </ipxact:array>
    …
     <ipxact:field>
      <ipxact:name>lane_errors</ipxact:name>
      <ipxact:accessHandles>
       <ipxact:accessHandle force="true">
       <ipxact:slices>
         <ipxact:slice>
          <ipxact:pathSegments>
            <ipxact:pathSegment>"$sformatf("_lane_errors_%0d_%0d",  $ipxact_index_value(k),
$ipxact_index_value(j)) </ipxact:pathSegment>
          </ipxact:pathSegments>
        …
```

Generated UVM RAL

```
this.clear_hdl_path();
    this.add_hdl_path("top.path_to.register_bank");
    foreach (this.RX_ERROR_Reg[i0]) begin
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 7, 3), 31, 1, 1);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 7, 2), 30, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 7, 1), 29, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 7, 0), 28, 1, 0);
      ...
                  ...
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 1, 3), 7, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 1, 2), 6, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 1, 1), 5, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 1, 0), 4, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 0, 3), 3, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 0, 2), 2, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 0, 1), 1, 1, 0);
      this.RX_ERROR_Reg[i0].add_hdl_path_slice($sformatf("rx_error_reg_%0d_lane_errors_%0d_%0d", i0, 0, 0), 0, 1, 0);
    end
  endfunction: build
```

Low Power and Other Topics

Power Domain

set_design_top design_b

set_scope .

create_power_domain PD_Top -include_scope

create_power_domain PD_a -elements {comp_a}

create_power_domain PD_b -elements {comp_b}

```
<ipxact:powerDomains>
  <ipxact:powerDomain>
    <ipxact:name>PD_Top</ipxact:name>
  </ipxact:powerDomain>
</ipxact:powerDomains>

<ipxact:powerDomains>
  <ipxact:powerDomain>
    <ipxact:name>PD_a</ipxact:name>
  </ipxact:powerDomain>
</ipxact:powerDomains>

<ipxact:powerDomains>
  <ipxact:powerDomain>
    <ipxact:name>PD_b</ipxact:name>
  </ipxact:powerDomain>
</ipxact:powerDomains>
```

Sample UPF

```
# Create supply ports for each domain

create_supply_port VDD_A -domain PD_a

create_supply_port VDD_B -domain PD_b
```

```
<ipxact:port>
  <ipxact:name>VDD_A</ipxact:name>
  <ipxact:powerConstraint>
    <ipxact:powerDomainRef> PD_a
    </ipxact:powerDomainRef>
  </ipxact:powerConstraint>
</ipxact:port>

<ipxact:port>
  <ipxact:name>VDD_B</ipxact:name>
    <ipxact:powerConstraint>
      <ipxact:powerDomainRef> PD_b
      </ipxact:powerDomainRef>
    </ipxact:powerConstraint>
</ipxact:port>
```

IP-XACT Design Mapped to Sample UPF

```
<ipxact:componentInstance>
  <ipxact:instanceName>u_apb_ip</ipxact:instanceName>
    <ipxact:powerDomainLinks>
      <ipxact:powerDomainLink>
       <ipxact:externalPowerDomainReference>PD_Top</ipxact:externalPowerDomainReference>
 <ipxact:internalPowerDomainReference>PD_APB_IP</ipxact:internalPowerDomainReference>
       </ipxact:powerDomainLink>
    </ipxact:powerDomainLinks>
</ipxact:componentInstance>
```

Conclusion

IPXACT 2022 (Key Takeaways)

- Establishes a unified, tool-agnostic standard for IP description and integration
- Reduces ambiguity in registers, interfaces, and memory maps
- Enables consistent automation across RTL, UVM, firmware, and documentation
- Improves reuse and scalability for large, multi-IP SoC projects
- Strengthens interoperability across vendors and internal teams
- Minimizes manual edits, integration errors, and version drift
- Supports modern abstraction and configurable IP packaging
- Equips teams to build predictable, maintainable, automation-ready flows