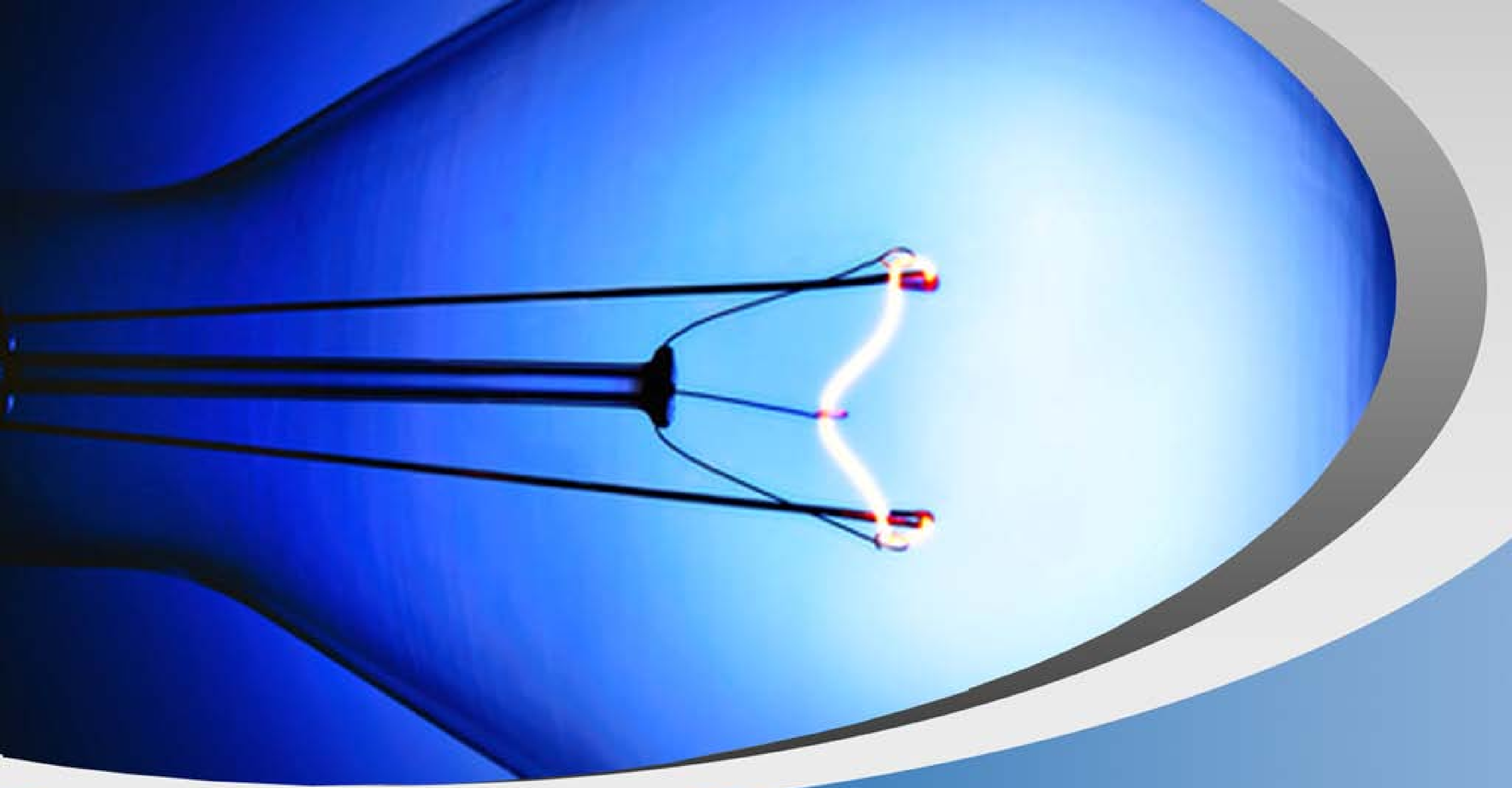


Improving Verification Efficiency Using IP-XACT

Members of IP-XACT Technical Committee

IP-XACT™

DVCon 2012 Design & Verification Conference & Exhibition

accellera SYSTEMS INITIATIVE

Improving Verification Efficiency Using IP-XACT

John A. Swanson

Synopsys

Agenda

- Why is the IP-XACT Accellera Systems Initiative working group looking into verification?
- What is IP-XACT
- How IP-XACT can be used in verification

Major Consumer Electronics Trends 2011

- Media tablets galore!
- It's smart everything!
- Everything is connected
  - 1 trillion connected devices, or 140 devices per person by 2013
- Video anywhere and anytime
  - Watch your shows on any device; And in 3D
- Smartphone, smart TV, smart grid, smart car
- The user experience is everything
  - Same user experience on any device

Connected Device

+

Connected TV/Home

+

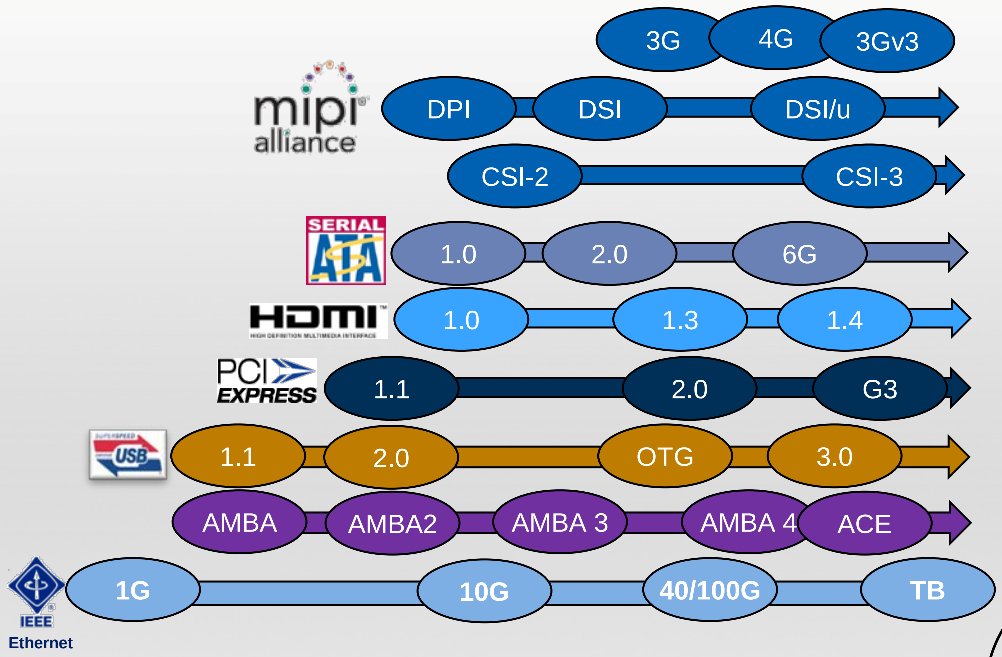
Connected Car

=

Connected Life

IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

Explosion in the Number of Protocols!

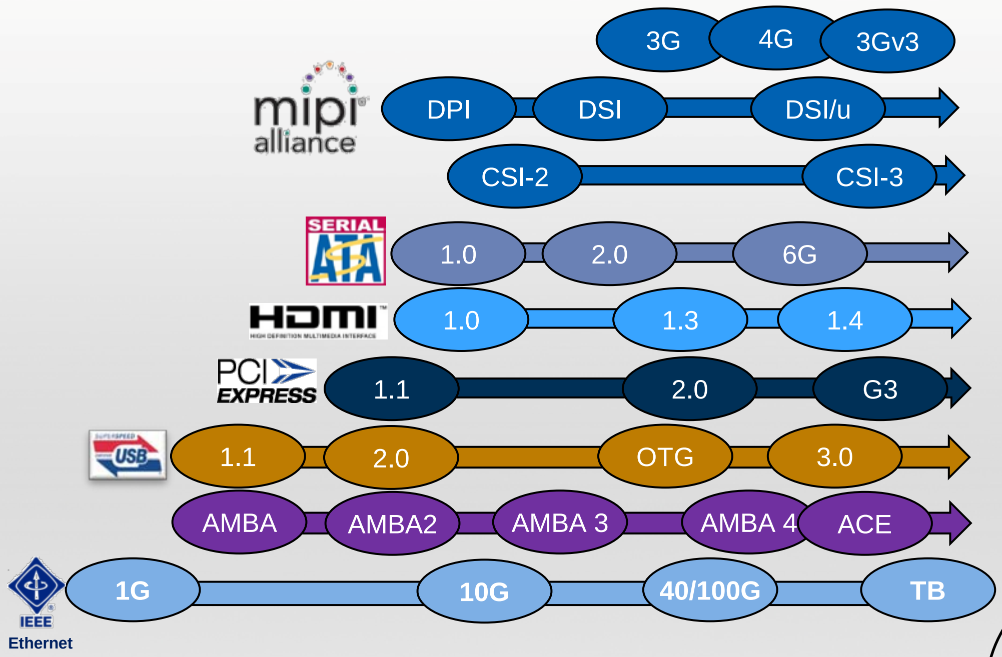


- **Rapidly increasing # of protocols on SoCs**
- **Highly optimized for end-user applications**
  - PCIe, USB 3.0, Ethernet, SDIO, SATA6G, OCP 3.0, AMBA AXI4, ACE, …

Consumers driving speed and features

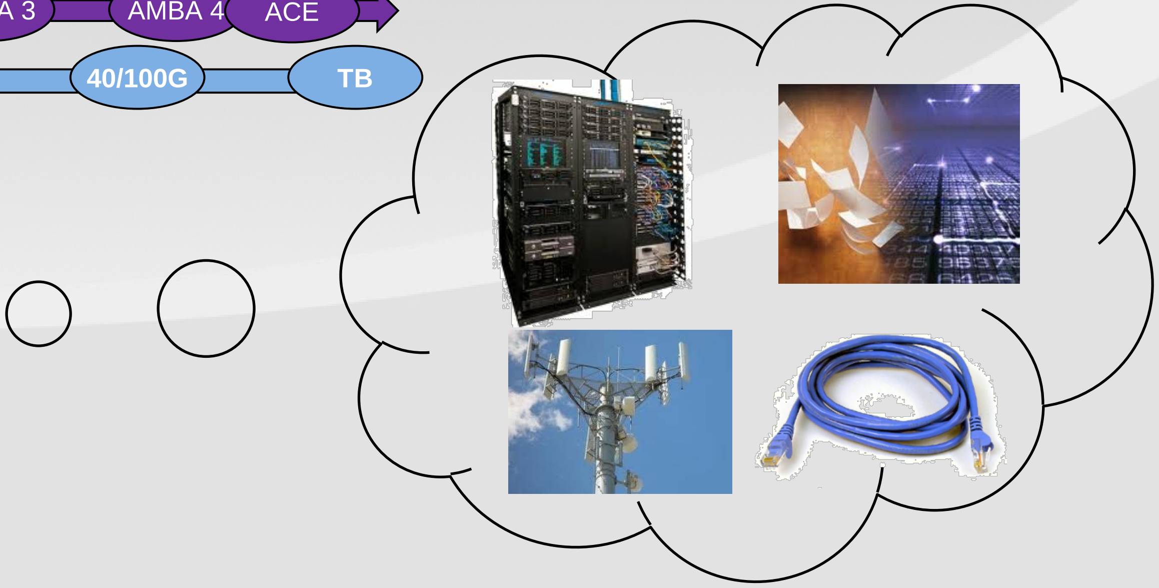
IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

Explosion in the Number of Protocols!



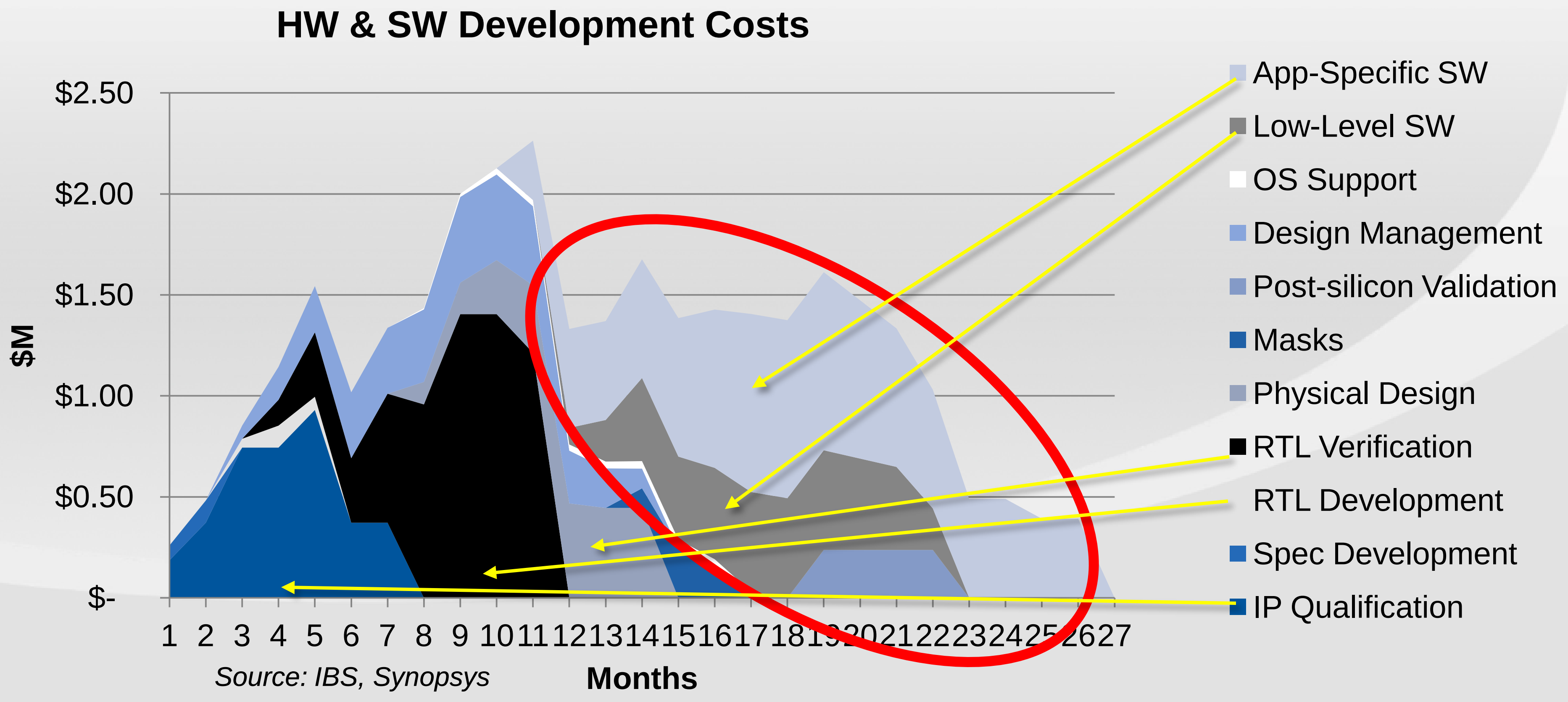
- **Rapidly increasing # of protocols on SoCs**
- **Highly optimized for end-user applications**
  - PCIe, USB 3.0, Ethernet, SDIO, SATA6G, OCP 3.0, AMBA AXI4, ACE, …

Companies keeping up with demand

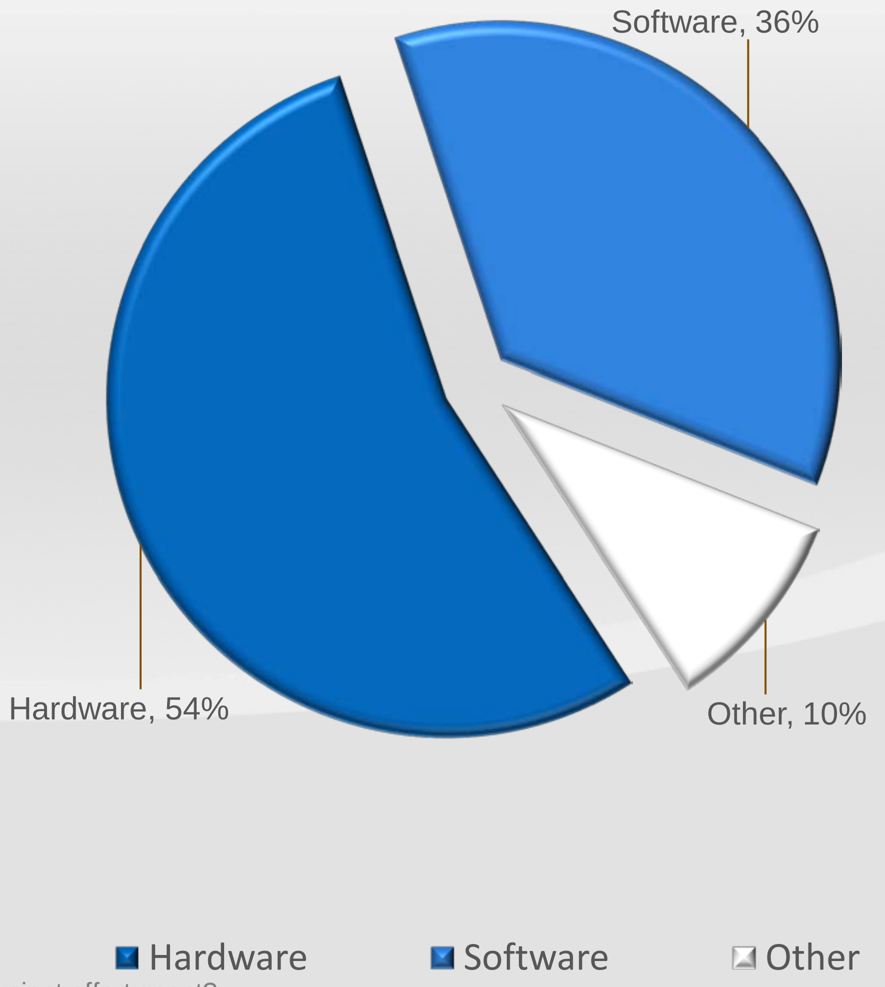


IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

SW is Half of Time-To-Market!



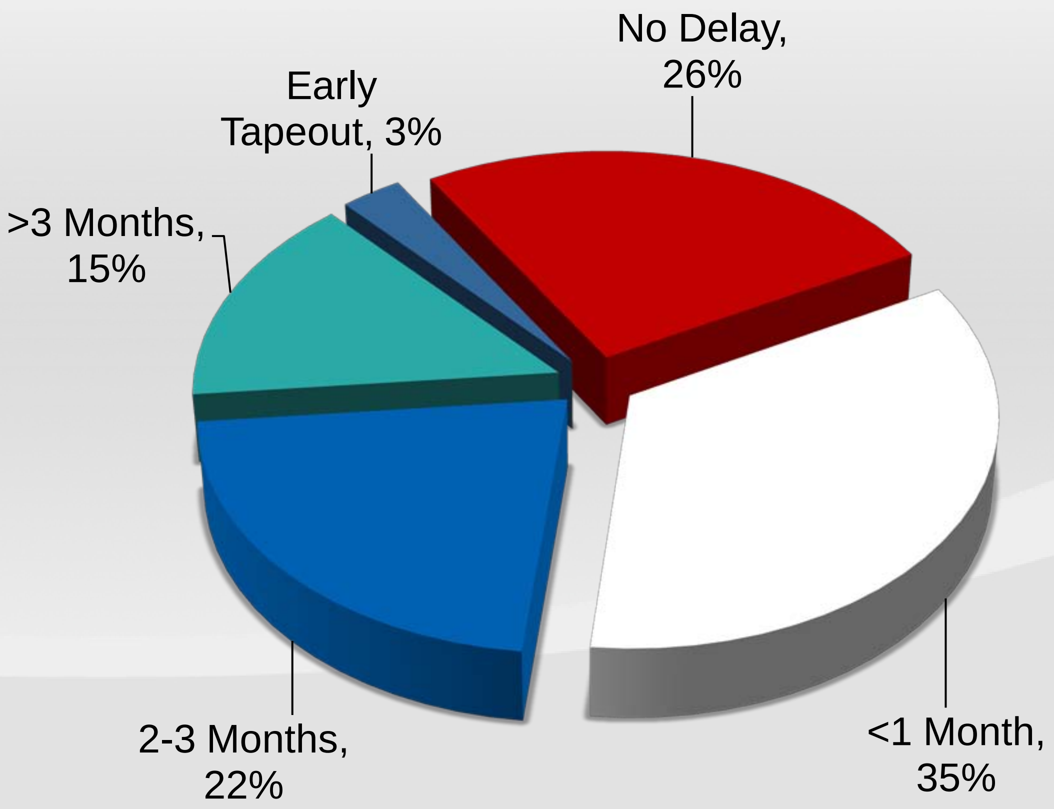
… And Approaching 50% of the Effort



Indicate the percentage of total project effort spent?
2010 N = 860; Margin of error = +/- 3%

Source: WW GUS Survey 2010

All of This Yields Incredible Schedule Pressure



For your most recently taped-out design, how did the tapeout date compare to the initial tapeout target date?
2010 N = 575; Margin of error = +/- 4%

"Don't know" and "N/A" responses are excluded from analysis.

2010

Source: WW GUS Survey 2010

And it's your Job to make sure it works!

Did I connect this core to my bus correctly?

Will this core do what I want?

Will I meet my performance goals ?

How can I be sure it works properly?

Will it work in my technology?

Can I use it in my own tool environment ?

So…

- **If you know how your components are configured**
  - Modes of operation
  - Register maps
  - Address space
  - Interface configurations
  - Language
  - …

You can streamline your verification flow!
using IP-XACT XML

Agenda

- Why is the IP-XACT Accellera Systems Initiative working group looking into verification?
- What is IP-XACT
- How IP-XACT can be used in verification

What is IEEE 1685-2009 (IP-XACT)

- An XML schema for language and vendor-neutral IP descriptions
- Includes a generator interface for “plug-in” functionality
- It has proven**:**
  - Low adoption costs
  - Value
  - The data needed to expand on for:
    - Verification
    - Software tools
    - Documentation
    - ...

IP-XACT provides the information that you would expect to find in a data book in an electronic tool independent format so you can use the data to enhance your companies productivity

The IP-XACT Specification

- Is design language neutral
- Is design tool neutral

Why is this so important?

- Is efficient
- Is proven
- Is built on the existing XML (W3C) standard
- Includes a standardized API for generator integration (TGI)
- Validated and released in accordance with the IEEE policies

Why is this Important?

Languages used by IC/SoC Designers

	Languages used to describe SOC design	Languages used to write SOC testbench & assertions
C++	44.4%	38.9%
VHDL	41.7%	33.3%
Verilog	38.9%	2.3%
C	36.1%	36.1%
SystemC	22.2%	30.6%
System Verilog	22.2%	27.8%
Netlists	8.3%	*
e	5.6%	5.6%
Open Vera	5.6%	2.8%
PSL	2.8%	2.8%
In-House Developed	5.6%	5.6%
Other	0.0%	0.0%
N/A	11.1%	8.3%

Source VDC Research 2009 Service Year Track 1: Embedded Software Engineering Market Technologies

* Not offered as a choice

% > 100% due to multiple languages used in a project

What is an XML Schema?

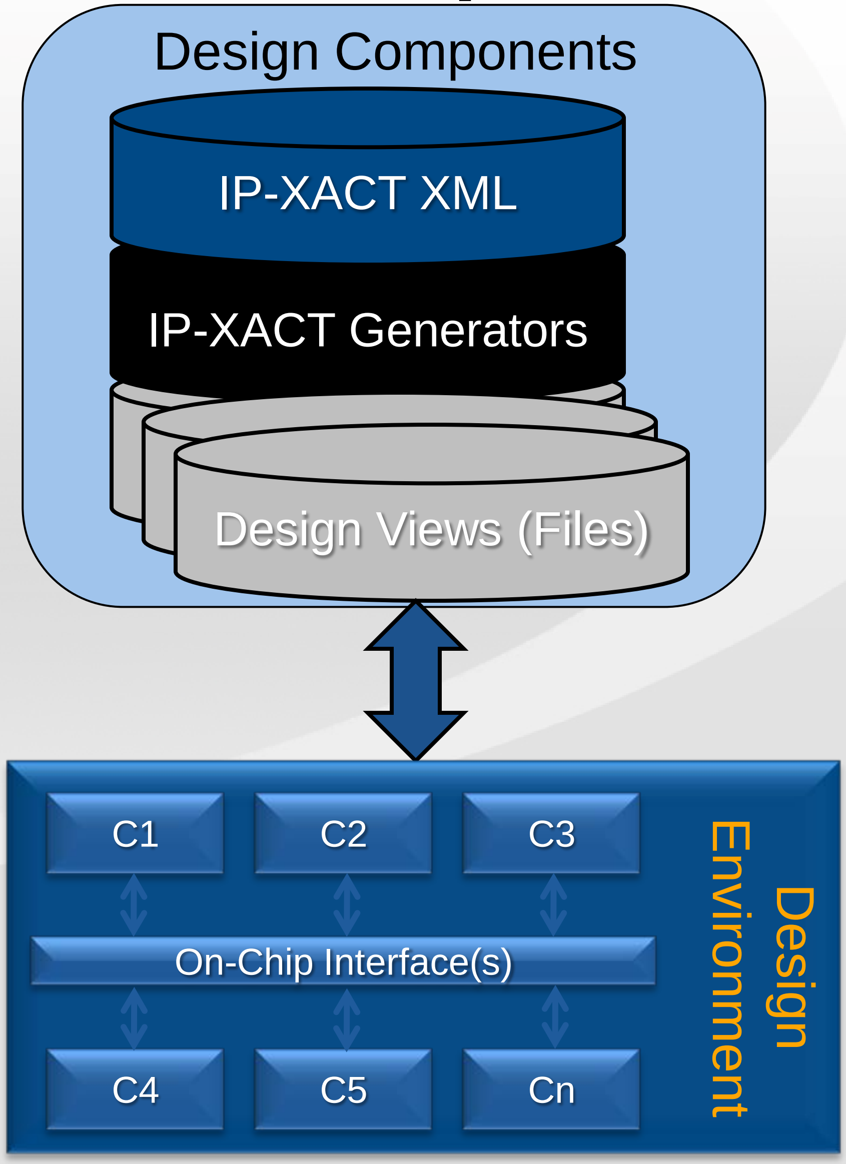
- **The purpose of a schema is to define the legal building blocks of an XML document**
  - It defines the document structure with a list of legal elements
- **An XML schema defines:**
  - Elements and attributes that can appear in a document
  - Which elements are child elements
  - The number and order of child elements
  - Whether an element is empty or can include text
  - Data types for elements and attributes
  - Default and fixed values for elements and attributes

XML 101

- **XML does *NOT DO ANYTHING***
  - XML was created to structure, store, and transport information
- **XML is just plain text**
  - XML is nothing special. It is just plain text. Software that can handle plain text can also handle XML.
  - However, XML-aware applications can handle the XML tags specially.
  - The functional meaning of the tags depends on the nature of the application.
- **With XML you invent your own tags**
  - XML has no pre-defined tags
  - XML is designed to allow things like… IP-XACT and XML Schema

IP-XACT: An XML Schema for Components

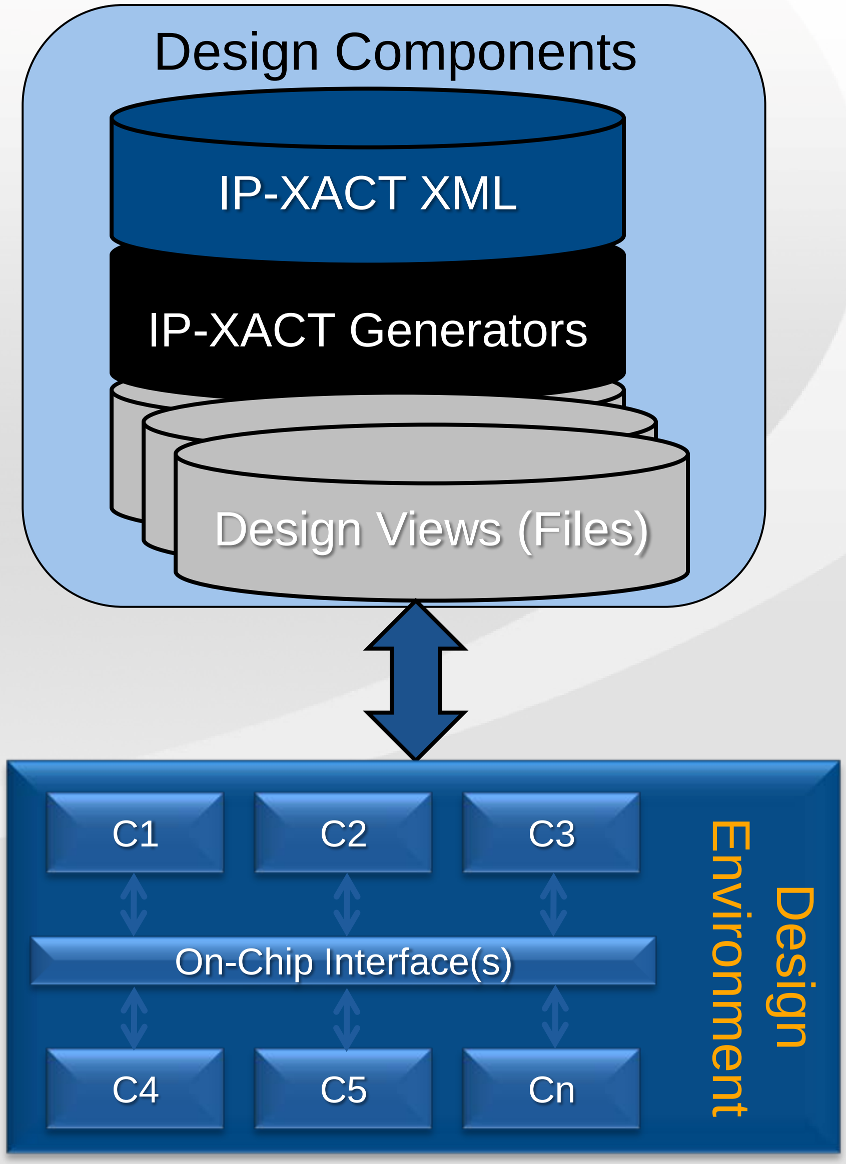
- **IP-XACT is an IEEE specification for documenting IP**
  - Enables automated design creation and configuration
  - Tool independent
  - Machine readable
- **Benefits**
  - Documentation of all aspects of IP using an XML databook format
  - Documentation of models in a quantifiable and language-independent way
  - Enables designers to deploy specialist knowledge in their design



IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

IP-XACT for Component Descriptions

- **Component XML describes**
  - Memory maps
  - Registers
  - Bus interfaces
  - Ports
  - Views (additional data files)
  - Parameters
  - Generators
  - File sets

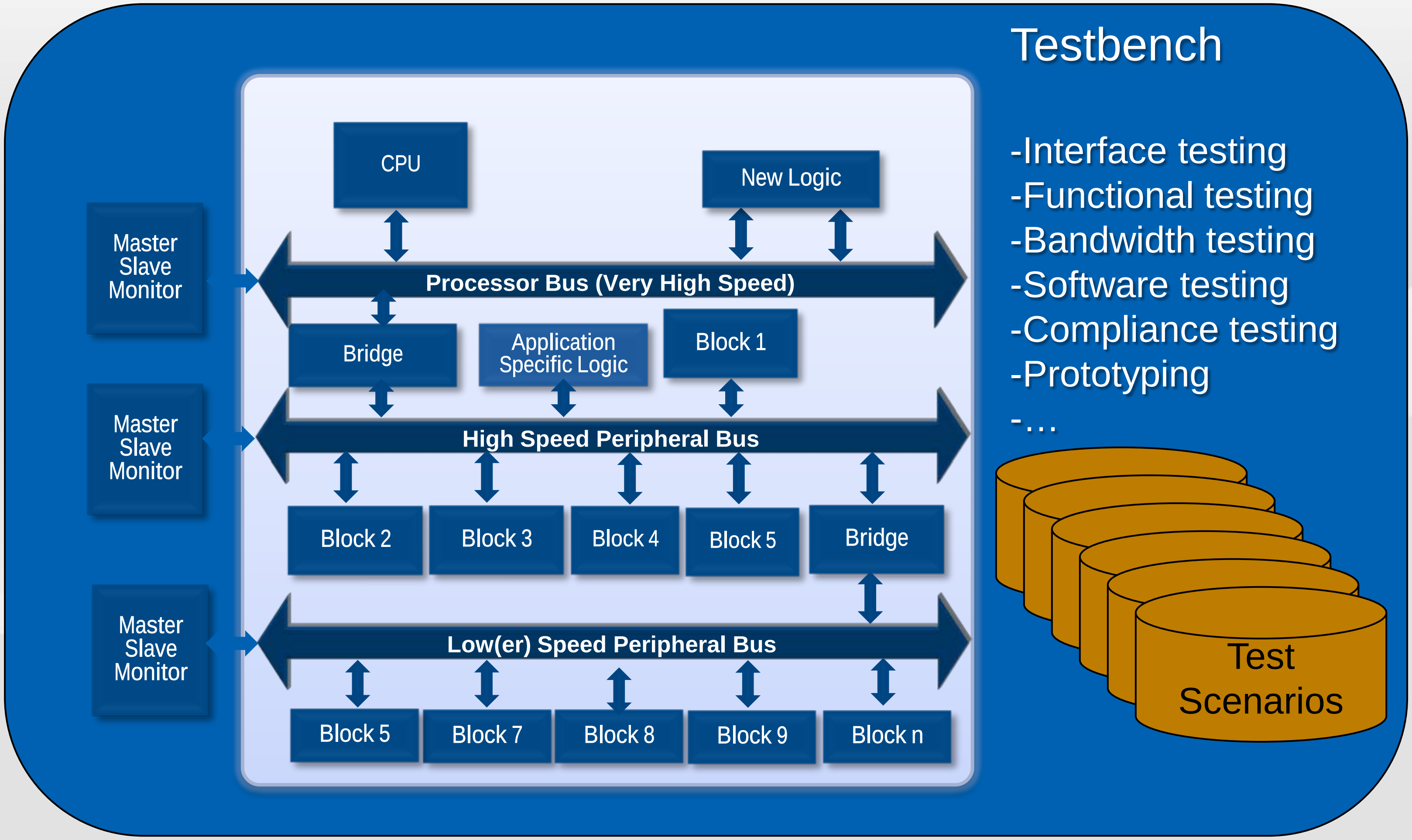


IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

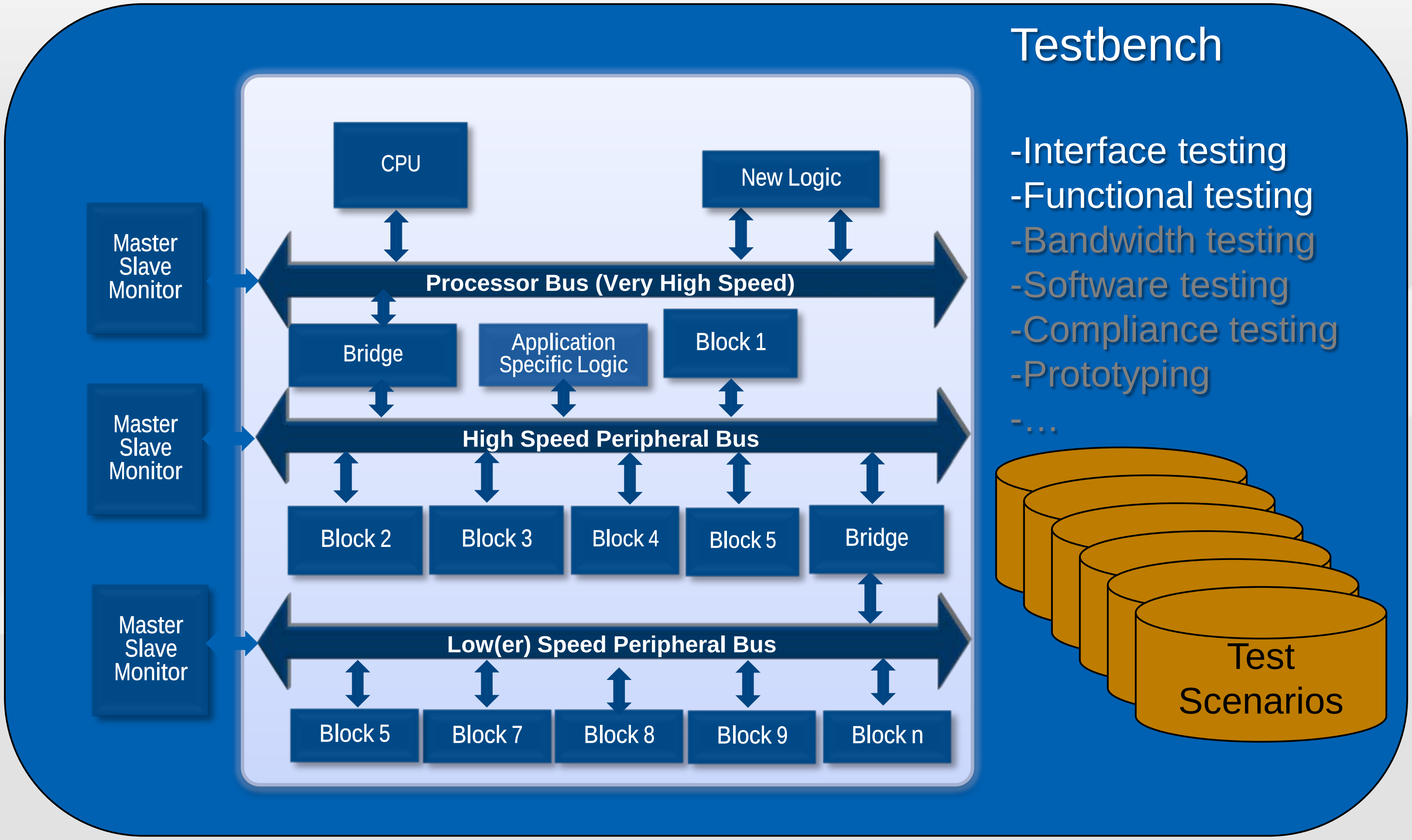
Agenda

- Why is the IP-XACT Accellera Systems Initiative working group looking into verification?
- What is IP-XACT
- **How IP-XACT can be used in verification**

The information to help build your Verification Environment



The information to help build your Verification Environment



-Interface testing
-Functional testing
-Bandwidth testing
-Software testing
-Compliance testing
-Prototyping
-…

Where to Connect the BFM?

What is the schema version?

```
<?xml version="1.0" encoding="utf-8" ?>
- <spirit:component xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009" …>
<spirit:vendor>ThirdParty</spirit:vendor>
<spirit:library>ThirdParty</spirit:library>
<spirit:name>Subsystem1</spirit:name>
<spirit:version>1.0</spirit:version>
- <spirit:busInterfaces>
- <spirit:busInterface>
<spirit:name>HCLK_0</spirit:name>
<spirit:description>Clock signal for AHB masters and slaves… </spirit:description>
<spirit:busType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb" spirit:version="r1p0" />
<spirit:abstractionType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb_rtl" spirit:version="r1p0" />
- <spirit:system>
<spirit:group>AHB_CLK</spirit:group>
</spirit:system>
- <spirit:portMaps>
- <spirit:portMap>
- <spirit:logicalPort>
<spirit:name>HCLK</spirit:name>
</spirit:logicalPort>
- <spirit:physicalPort>
<spirit:name>HCLK_hclk</spirit:name>
</spirit:physicalPort>
</spirit:portMap>
</spirit:portMaps>
-<spirit:vendorExtensions>
-…
```

Where to Connect the BFM?

Find the top-level of the DUT

```
<?xml version="1.0" encoding="utf-8" ?>
- <spirit:component xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009" …>
<spirit:vendor>ThirdParty</spirit:vendor>
<spirit:library>ThirdParty</spirit:library>
<spirit:name>Subsystem1</spirit:name>
<spirit:version>1.0</spirit:version>
- <spirit:busInterfaces>
- <spirit:busInterface>
<spirit:name>HCLK_0</spirit:name>
<spirit:description>Clock signal for AHB masters and slaves… </spirit:description>
<spirit:busType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb" spirit:version="r1p0" />
<spirit:abstractionType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb_rtl" spirit:version="r1p0" />
- <spirit:system>
<spirit:group>AHB_CLK</spirit:group>
</spirit:system>
- <spirit:portMaps>
- <spirit:portMap>
- <spirit:logicalPort>
<spirit:name>HCLK</spirit:name>
</spirit:logicalPort>
- <spirit:physicalPort>
<spirit:name>HCLK_hclk</spirit:name>
</spirit:physicalPort>
</spirit:portMap>
</spirit:portMaps>
-<spirit:vendorExtensions>
-…
```

Where to Connect the BFM?

Find the top-level interfaces

```
<?xml version="1.0" encoding="utf-8" ?>
- <spirit:component xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009" …>
<spirit:vendor>ThirdParty</spirit:vendor>
<spirit:library>ThirdParty</spirit:library>
<spirit:name>Subsystem1</spirit:name>
<spirit:version>1.0</spirit:version>
- <spirit:busInterfaces>
- <spirit:busInterface>
<spirit:name>HCLK_0</spirit:name>
<spirit:description>Clock signal for AHB masters and slaves… </spirit:description>
<spirit:busType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb" spirit:version="r1p0" />
<spirit:abstractionType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb_rtl" spirit:version="r1p0" />
- <spirit:system>
<spirit:group>AHB_CLK</spirit:group>
</spirit:system>
- <spirit:portMaps>
- <spirit:portMap>
- <spirit:logicalPort>
<spirit:name>HCLK</spirit:name>
</spirit:logicalPort>
- <spirit:physicalPort>
<spirit:name>HCLK_hclk</spirit:name>
</spirit:physicalPort>
</spirit:portMap>
</spirit:portMaps>
-<spirit:vendorExtensions>
-…
```

IP-XACT

DVCon 2012

accellera SYSTEMS INITIATIVE

Where to Connect the BFM?

Find the top-level interfaces(2)

```
…
- <spirit:busInterface>
<spirit:name>ex_i_ahb_AHB_Master</spirit:name>
<spirit:description>Bus Master side of the AHB. On this interface the 'consumer's are AHB masters…</spirit:description>
<spirit:busType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb" spirit:version="r1p0" />
<spirit:abstractionType spirit:vendor="amba.com" spirit:library="busdef.amba.amba2" spirit:name="ahb_rtl" spirit:version="r1p0" />
<spirit:mirroredMaster />
- <spirit:portMaps>
- <spirit:portMap>
- <spirit:logicalPort>
<spirit:name>HSIZE</spirit:name>
</spirit:logicalPort>
- <spirit:physicalPort>
<spirit:name>ex_i_ahb_AHB_Master_hsize</spirit:name>
</spirit:physicalPort>
</spirit:portMap>
- <spirit:portMap>
- <spirit:logicalPort>
<spirit:name>HBURST</spirit:name>
</spirit:logicalPort>
- <spirit:physicalPort>
<spirit:name>ex_i_ahb_AHB_Master_hburst</spirit:name>
</spirit:physicalPort>
</spirit:portMap>
- <spirit:portMap>
- <spirit:logicalPort>
<spirit:name>HGRANTx</spirit:name>
</spirit:logicalPort>
- <spirit:physicalPort>
<spirit:name>ex_i_ahb_AHB_Master_hgrant</spirit:name>
</spirit:physicalPort>
</spirit:portMap>
```

IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

What is inside the DUT?

Find the components

```
<?xml version="1.0" encoding="utf-8" ?>
- <spirit:design xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009" xmlns:xsi="http://www.w3.org/2001/...">
<spirit:vendor>ThirdParty</spirit:vendor>
<spirit:library>ThirdParty</spirit:library>
<spirit:name>design_Subsystem1</spirit:name>
<spirit:version>1.0</spirit:version>
- <spirit:componentInstances>
- <spirit:componentInstance>
<spirit:instanceName>i_ahb</spirit:instanceName>
<spirit:componentRef spirit:vendor="IPProvider" spirit:library="MyLibrary" spirit:name="i_ahb_NAME" spirit:version="2.10b" />
</spirit:componentInstance>
+ <spirit:componentInstance>
<spirit:instanceName>i_apb</spirit:instanceName>
<spirit:componentRef spirit:vendor="IPProvider" spirit:library="MyLibrary" spirit:name="i_apb_NAME" spirit:version="2.02b" />
</spirit:componentInstance>
- <spirit:componentInstance>
<spirit:instanceName>i_uart</spirit:instanceName>
<spirit:componentRef spirit:vendor="IPProvider" spirit:library="MyLibrary" spirit:name="i_uart_NAME" spirit:version="3.12c" />
</spirit:componentInstance>
</spirit:componentInstances>
+ <spirit:interconnections>
- <spirit:interconnection>
<spirit:name>i_ahb_AHB_Slave_0</spirit:name>
<spirit:activeInterface spirit:componentRef="i_apb" spirit:busRef="AHB_Slave" />
<spirit:activeInterface spirit:componentRef="i_ahb" spirit:busRef="AHB_Slave_0" />
</spirit:interconnection>
- <spirit:interconnection>
<spirit:name>i_apb_APB_Slave</spirit:name>
<spirit:activeInterface spirit:componentRef="i_uart" spirit:busRef="APB_Slave" />
<spirit:activeInterface spirit:componentRef="i_apb" spirit:busRef="APB_Slave" />
</spirit:interconnection>
</spirit:interconnections>
```

What is inside the DUT?

Find the connections between components (Interface Level)

```
<?xml version="1.0" encoding="utf-8" ?>
- <spirit:design xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009" xmlns:xsi="http://www.w3.org/2001/...">
<spirit:vendor>ThirdParty</spirit:vendor>
<spirit:library>ThirdParty</spirit:library>
<spirit:name>design_Subsystem1</spirit:name>
<spirit:version>1.0</spirit:version>
- <spirit:componentInstances>
- <spirit:componentInstance>
<spirit:instanceName>i_ahb</spirit:instanceName>
<spirit:componentRef spirit:vendor="IPProvider" spirit:library="MyLibrary" spirit:name="i_ahb_NAME" spirit:version="2.10b" />
</spirit:componentInstance>
+ <spirit:componentInstance>
<spirit:instanceName>i_apb</spirit:instanceName>
<spirit:componentRef spirit:vendor="IPProvider" spirit:library="MyLibrary" spirit:name="i_apb_NAME" spirit:version="2.02b" />
</spirit:componentInstance>
- <spirit:componentInstance>
<spirit:instanceName>i_uart</spirit:instanceName>
<spirit:componentRef spirit:vendor="IPProvider" spirit:library="MyLibrary" spirit:name="i_uart_NAME" spirit:version="3.12c" />
</spirit:componentInstance>
</spirit:componentInstances>
+ <spirit:interconnections>
- <spirit:interconnection>
<spirit:name>i_ahb_AHB_Slave_0</spirit:name>
<spirit:activeInterface spirit:componentRef="i_apb" spirit:busRef="AHB_Slave" />
<spirit:activeInterface spirit:componentRef="i_ahb" spirit:busRef="AHB_Slave_0" />
</spirit:interconnection>
- <spirit:interconnection>
<spirit:name>i_apb_APB_Slave</spirit:name>
<spirit:activeInterface spirit:componentRef="i_uart" spirit:busRef="APB_Slave" />
<spirit:activeInterface spirit:componentRef="i_apb" spirit:busRef="APB_Slave" />
</spirit:interconnection>
</spirit:interconnections>
```

What is inside the DUT?

Find the connections between components (AdHoc Connections)

```
- <spirit:adHocConnections>
- <spirit:adHocConnection>
<spirit:name>i_ahb_pause</spirit:name>
<spirit:internalPortReference spirit:componentRef="i_ahb" spirit:portRef="pause" />
<spirit:externalPortReference spirit:portRef="i_ahb_pause" />
</spirit:adHocConnection>
- <spirit:adHocConnection>
<spirit:name>i_apb_pclk_en</spirit:name>
<spirit:internalPortReference spirit:componentRef="i_apb" spirit:portRef="pclk_en" />
<spirit:externalPortReference spirit:portRef="i_apb_pclk_en" />
</spirit:adHocConnection>
- <spirit:adHocConnection>
<spirit:name>i_uart_dcd_n</spirit:name>
<spirit:internalPortReference spirit:componentRef="i_uart" spirit:portRef="dcd_n" />
<spirit:externalPortReference spirit:portRef="i_uart_dcd_n" />
</spirit:adHocConnection>
- <spirit:adHocConnection>
<spirit:name>i_uart_dsr_n</spirit:name>
<spirit:internalPortReference spirit:componentRef="i_uart" spirit:portRef="dsr_n" />
<spirit:externalPortReference spirit:portRef="i_uart_dsr_n" />
</spirit:adHocConnection>
```

What is inside the DUT?

Find the connections between components (Hierarchical Connections)

```
- <spirit:hierConnections>
- <spirit:hierConnection spirit:interfaceRef="SIO">
<spirit:interface spirit:componentRef="i_uart" spirit:busRef="SIO" />
</spirit:hierConnection>
+ <spirit:hierConnection spirit:interfaceRef="ex_i_ahb_AHB_Master">
<spirit:interface spirit:componentRef="i_ahb" spirit:busRef="AHB_Master" />
</spirit:hierConnection>
- <spirit:hierConnection spirit:interfaceRef="PRESETn">
<spirit:interface spirit:componentRef="i_uart" spirit:busRef="PRESETn" />
</spirit:hierConnection>
- <spirit:hierConnection spirit:interfaceRef="PCLK">
<spirit:interface spirit:componentRef="i_uart" spirit:busRef="PCLK" />
</spirit:hierConnection>
- <spirit:hierConnection spirit:interfaceRef="HRESETn_0">
<spirit:interface spirit:componentRef="i_ahb" spirit:busRef="HRESETn" />
-</spirit:hierConnection>
</spirit:hierConnections>
</spirit:design>
```

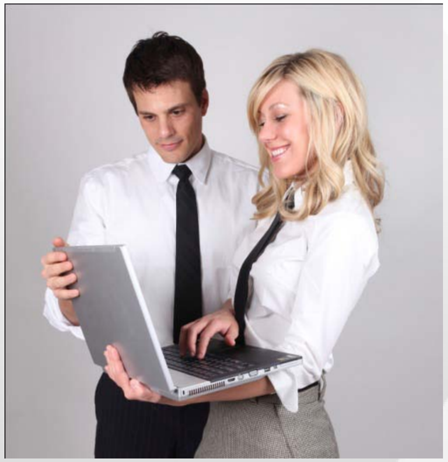
What is inside the DUT?

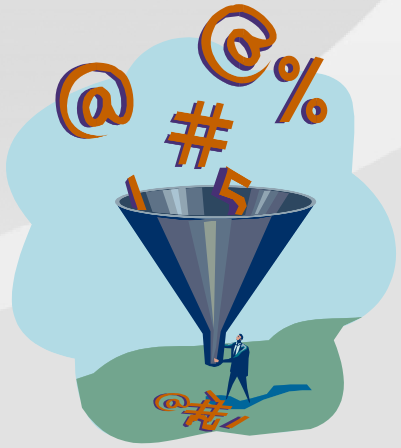
Find the Memory Map / Registers

```xml
<spirit:memoryMaps>
- <spirit:memoryMap>
<spirit:name>uart_memory_map</spirit:name>
- <spirit:addressBlock>
<spirit:name>uart_address_block</spirit:name>
<spirit:baseAddress>0x0</spirit:baseAddress>
<spirit:range>0x1000</spirit:range>
<spirit:width>32</spirit:width>
- <spirit:register>
<spirit:name>RBR</spirit:name>
<spirit:description>Receive Buffer Register, reading this register when the DLAB bit is zero… </spirit:description>
<spirit:addressOffset>0x0</spirit:addressOffset>
<spirit:size>32</spirit:size>
<spirit:volatile>false</spirit:volatile>
<spirit:access>read-only</spirit:access>
- <spirit:reset>
<spirit:value>0x0</spirit:value>
</spirit:reset>
- <spirit:field>
<spirit:name>rbr</spirit:name>
<spirit:description>Receive Buffer Register: This register contains the data byte received on the serial input port …</spirit:description>
<spirit:bitOffset>0</spirit:bitOffset>
<spirit:bitWidth>8</spirit:bitWidth>
<spirit:access>read-only</spirit:access>
</spirit:field>
- <spirit:field>
<spirit:name>RSVD_RBR_31to8</spirit:name>
<spirit:description>Reserved bits [31:8] - Read Only</spirit:description>
<spirit:bitOffset>8</spirit:bitOffset>
<spirit:bitWidth>24</spirit:bitWidth>
<spirit:access>read-only</spirit:access>
</spirit:field>
…
```

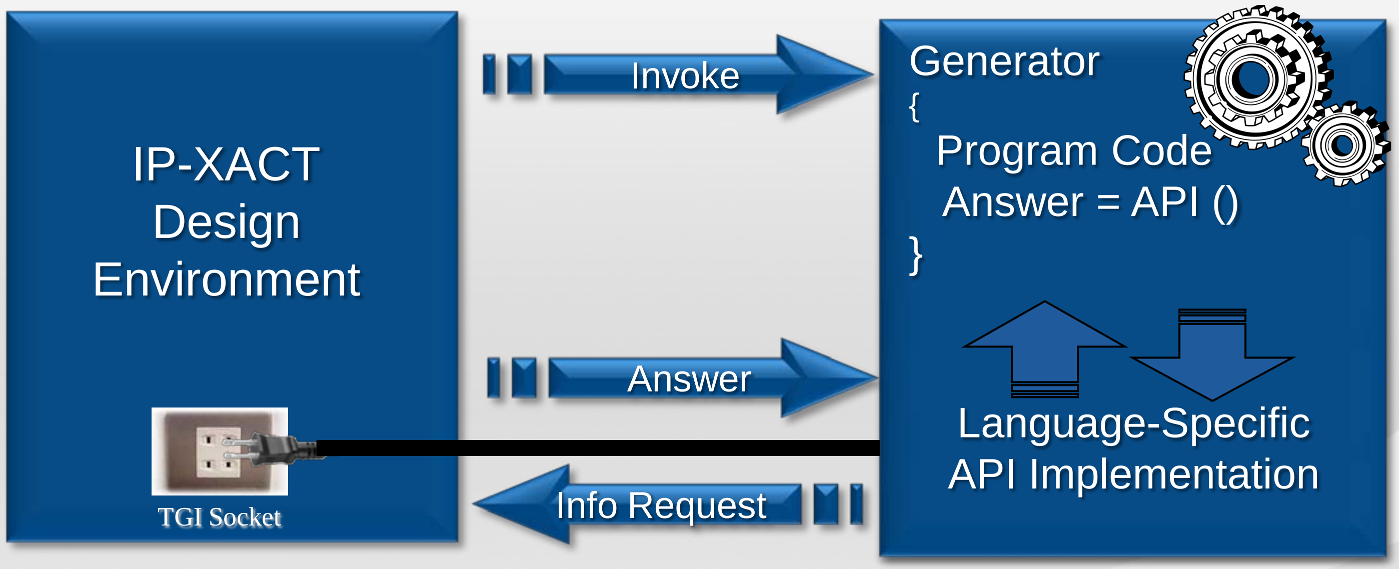
How do you do all this?

- **Vendors have tools**
- **All IP-XACT design environments must support the TGI**
  - You can expand the features of the design environment as you desire
  - You can easily parse the information
    - XML is ideal for parsing
  - Vendors have “things” to make your job easier





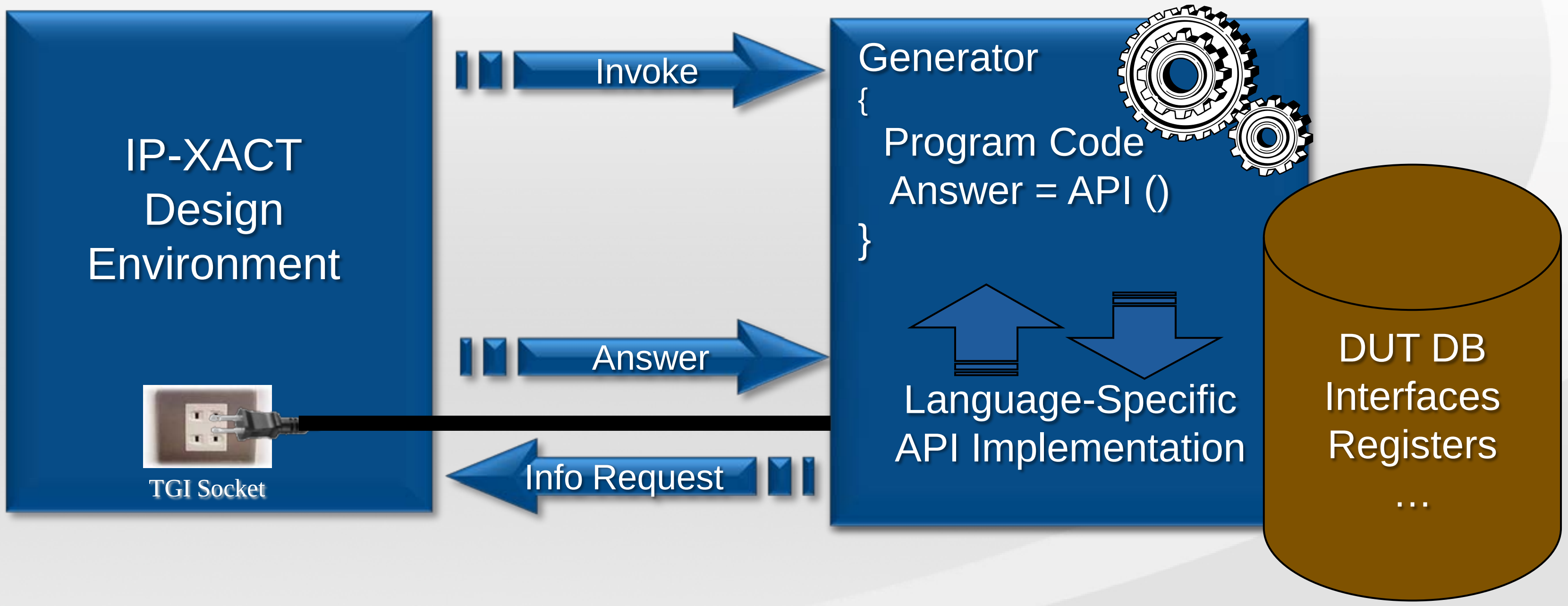
Generators – A Tool for Verification!



Generators are program modules that process IP-XACT XML data into 'something useful' for the design

Key portable mechanism for encapsulating specialist design knowledge
Enables designers to deploy specialist knowledge in their design

Generators – Assembly of the Testbench



Key portable mechanism for encapsulating specialist design knowledge

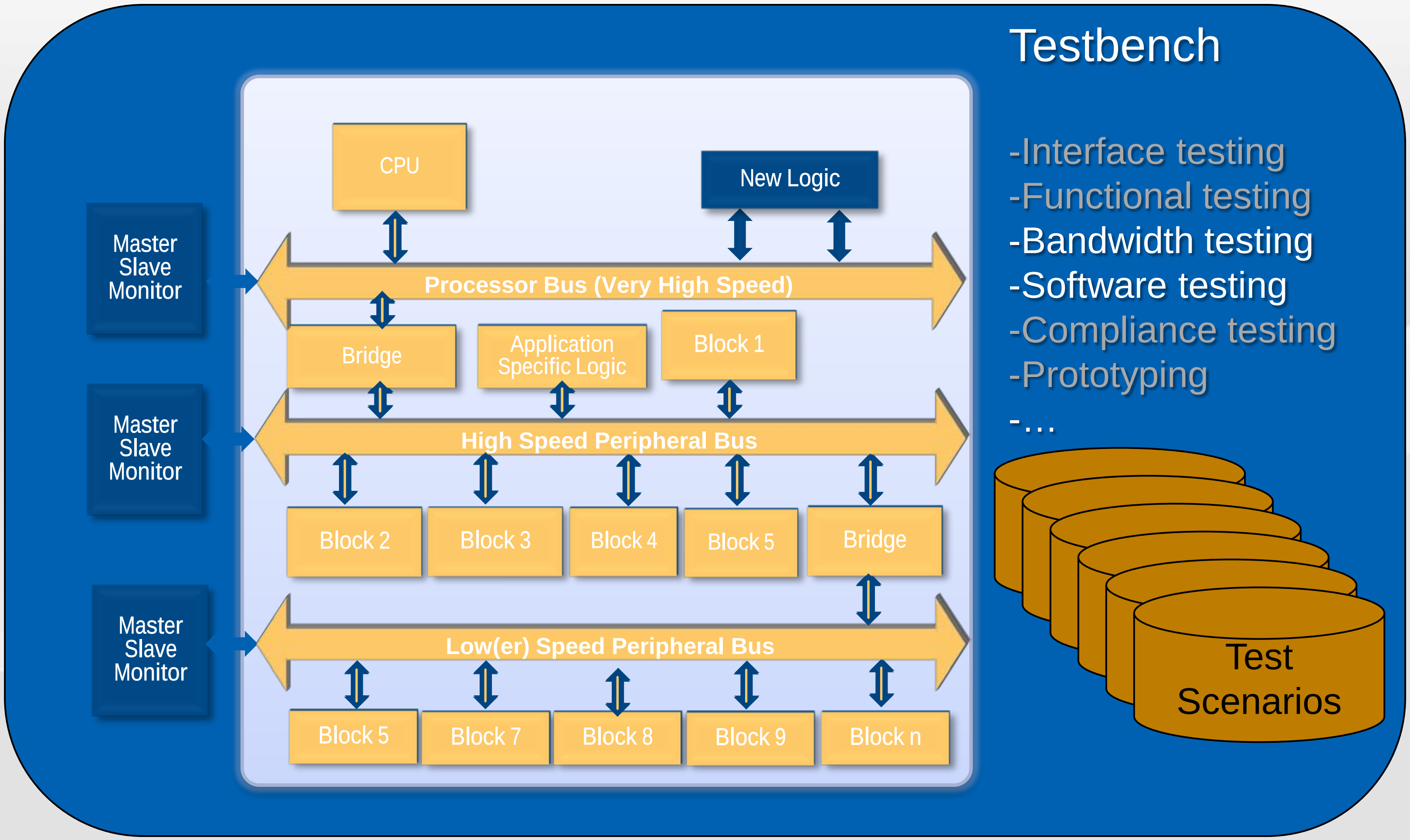
Enables designers to deploy specialist knowledge in their design

Like a testbench!

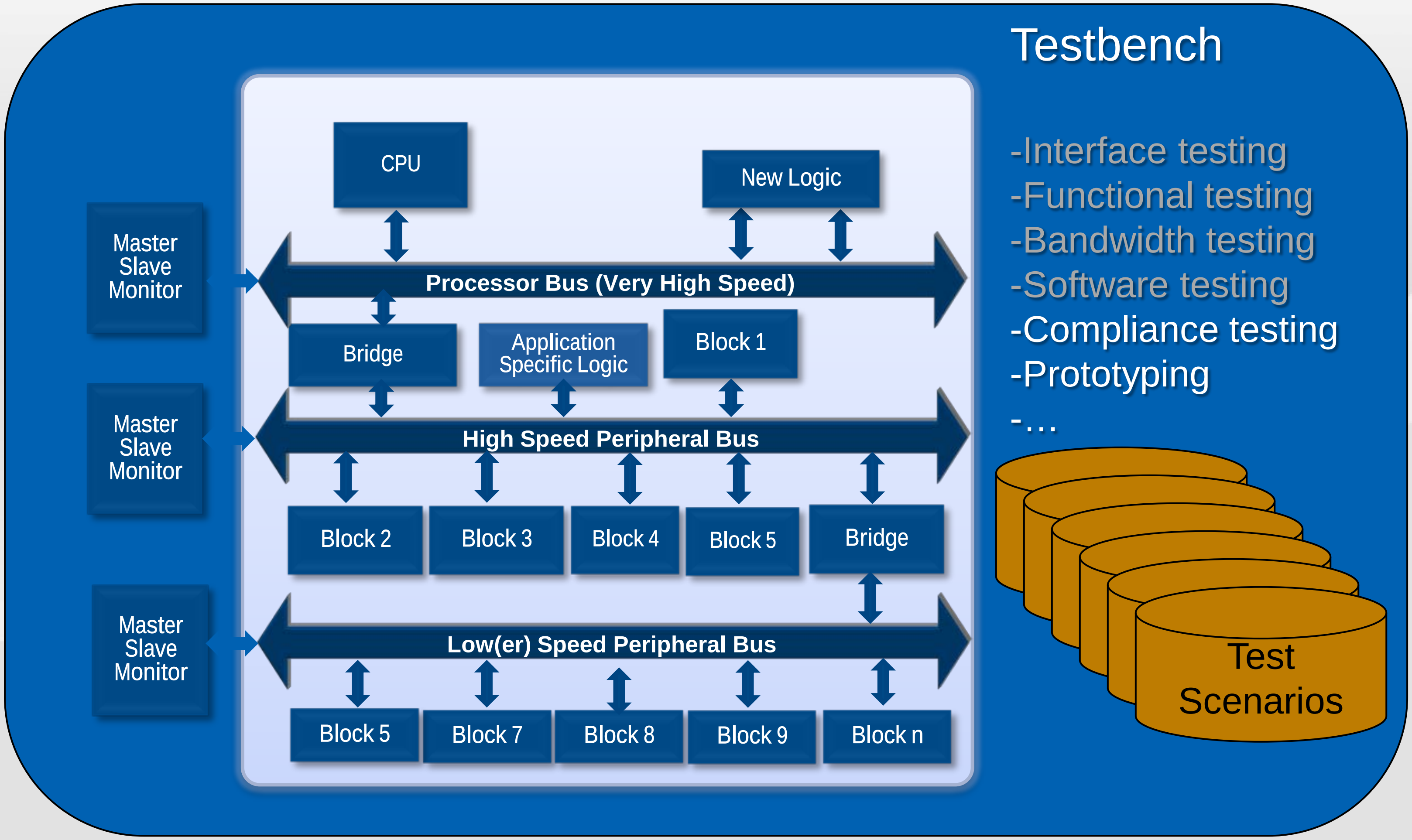
Where Are Generators Specified

- **Generators can be grouped into generator chains and invoked from the design environment**
  - Combining individual generators enables the creation of custom functionality, like *design specific verification tests*
  - Example: A generator chain may combine a generic HDL netlist generator with a simulator specific compilation command generator to build a custom command to run a simulation
    - You can also automate connection of VIP to monitor, interfaces, registers, etc.
    - You may also want to develop stimulus runs to validate your design
    - Set-up different views of components to run more “focused” verification. C/C++, behavioral HDL, etc…
- **Generators can also be attached to a component**
  - The activate only of the component is included in the design
  - Example: Check to see if a more up-to-date version of a component exists in a your companies RCS

Change Model Views



The information to help build your Verification Environment



Compliance Testing Options

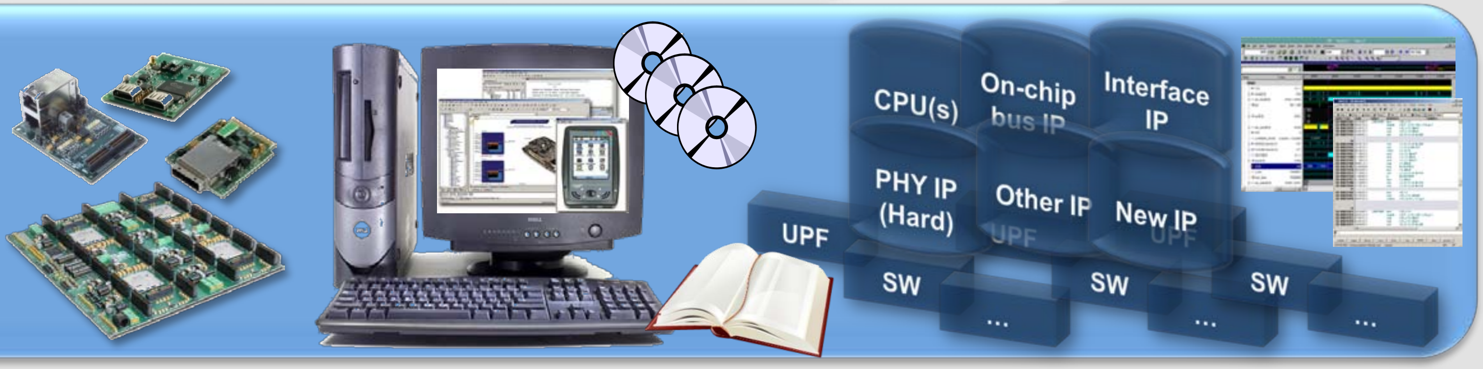
FPGA Prototype

Development System

Pre-Silicon

Post-Silicon

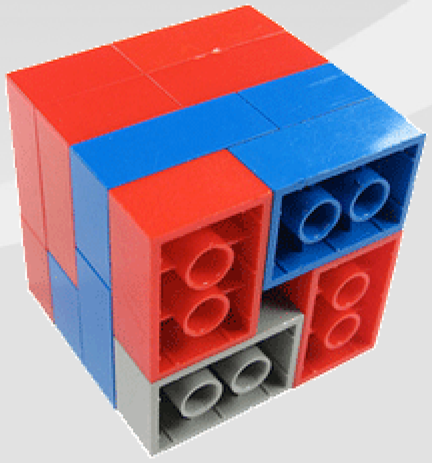
Development platforms

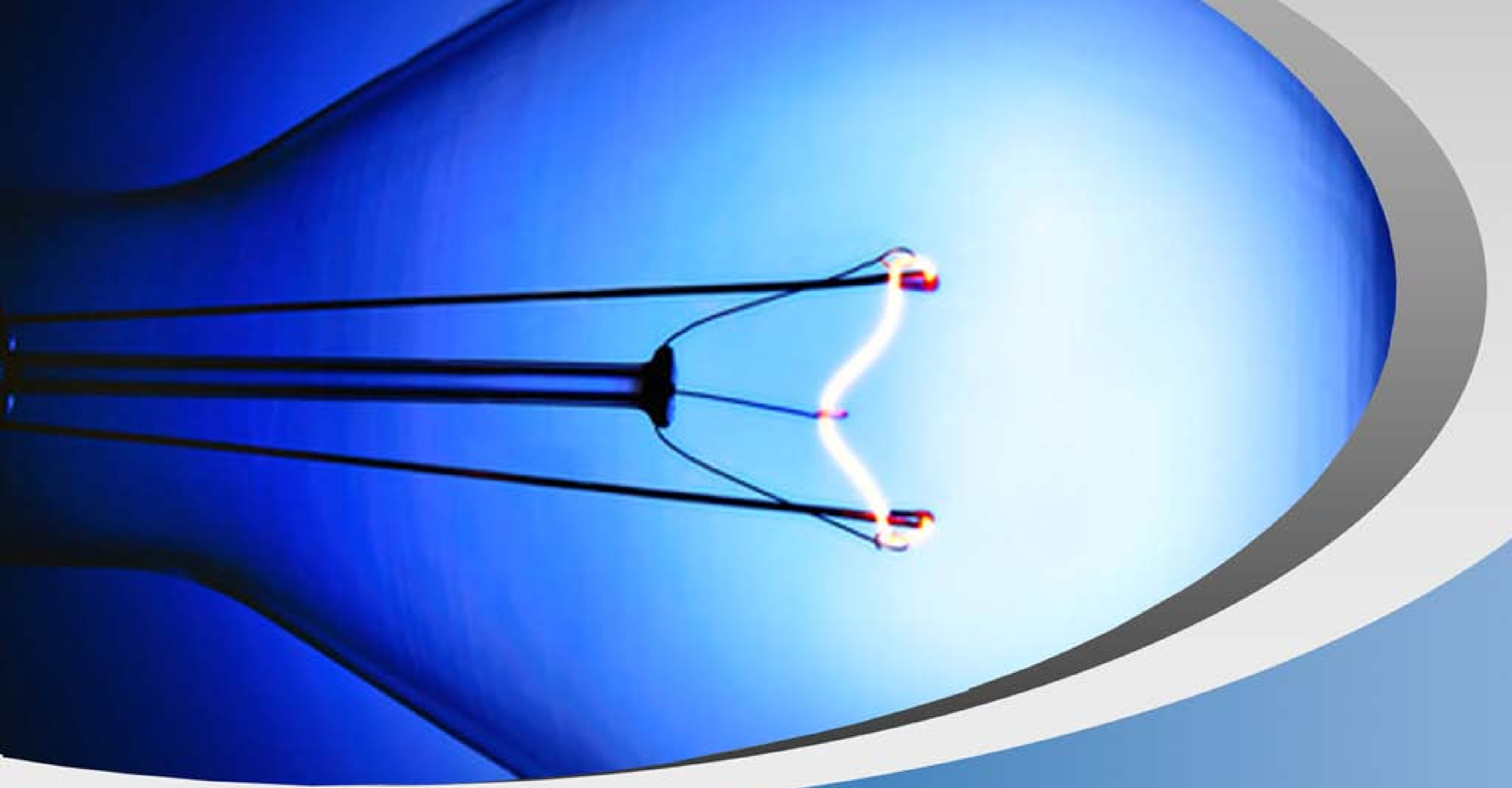


IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

In Summary

- **IP-XACT provides the information that you would expect to find in a data book**
  - An XML schema for language and vendor-neutral IP descriptions
  - Generator interfaces for plug-in functionality
  - Easy to process with scripts, etc...
  - It has
    - Low adoption costs
    - Proven value at large companies
    - The data needed to expand on
      - Hardware verification
      - Software verification
      - Application verification
      - ...





Verification Automation Improvement Using IP-XACT

Members of IP-XACT Technical Committee

IP-XACT™

DVCon 2012 Design & Verification Conference & Exhibition

accellera SYSTEMS INITIATIVE

Use Case: Verification Automation Improvement Using IP-XACT

Kamlesh Pathak

STMicroelectronics

Agenda

- Typical challenges in verification
- IP-XACT offerings for verification automation
- Applying IP-XACT for verification automation
- Overcoming challenges
- Conclusion
- Q&A

Typical Challenges in Verification

- **Developing testbench**
  - IP Integration needs knowledge of IP
  - Mechanisms for accommodating IP configuration
- **Use of multiple IP suppliers results in inconsistent IP verification views**
  - Methodology, testbenches, coverage data, etc. Industry standards late
- **Awareness and impact of IP implementation changes & known problems**
- **Concurrent IP development and SoC integration demands incremental maturity**
- **Difficult to debug complex interactions between IPs**

Typical Challenges in Verification

- **Register implementation**
  - Handling register descriptions at multiple places
  - Cost, productivity, quality
  - Coherency between design teams
- **Writing register test cases**
  - Large number of registers
- **Impact of changes in specification**
  - Additions/changes are problematic and error prone
- **Reuse IP test cases at top level**
  - Huge effort! What subset is needed for integration verification?

IP-XACT Offerings for Verification Automation

- **Single description for all information**
  - All representations generated from the single source
- **Current version of IP-XACT (IEEE 1685-2009) provides**
  - Metadata to describe components, designs
    - Interfaces, ports ,registers, bit-fields
    - Component instances, connections between components
    - Configurable attributes
  - Automated configuration of IPs
  - Automated composition, integration and configuration of verification environment
  - Automatic insertion of required transactors based on the abstraction
  - Parameters of design and verification components
  - Design configuration file
  - Easy interface for generators

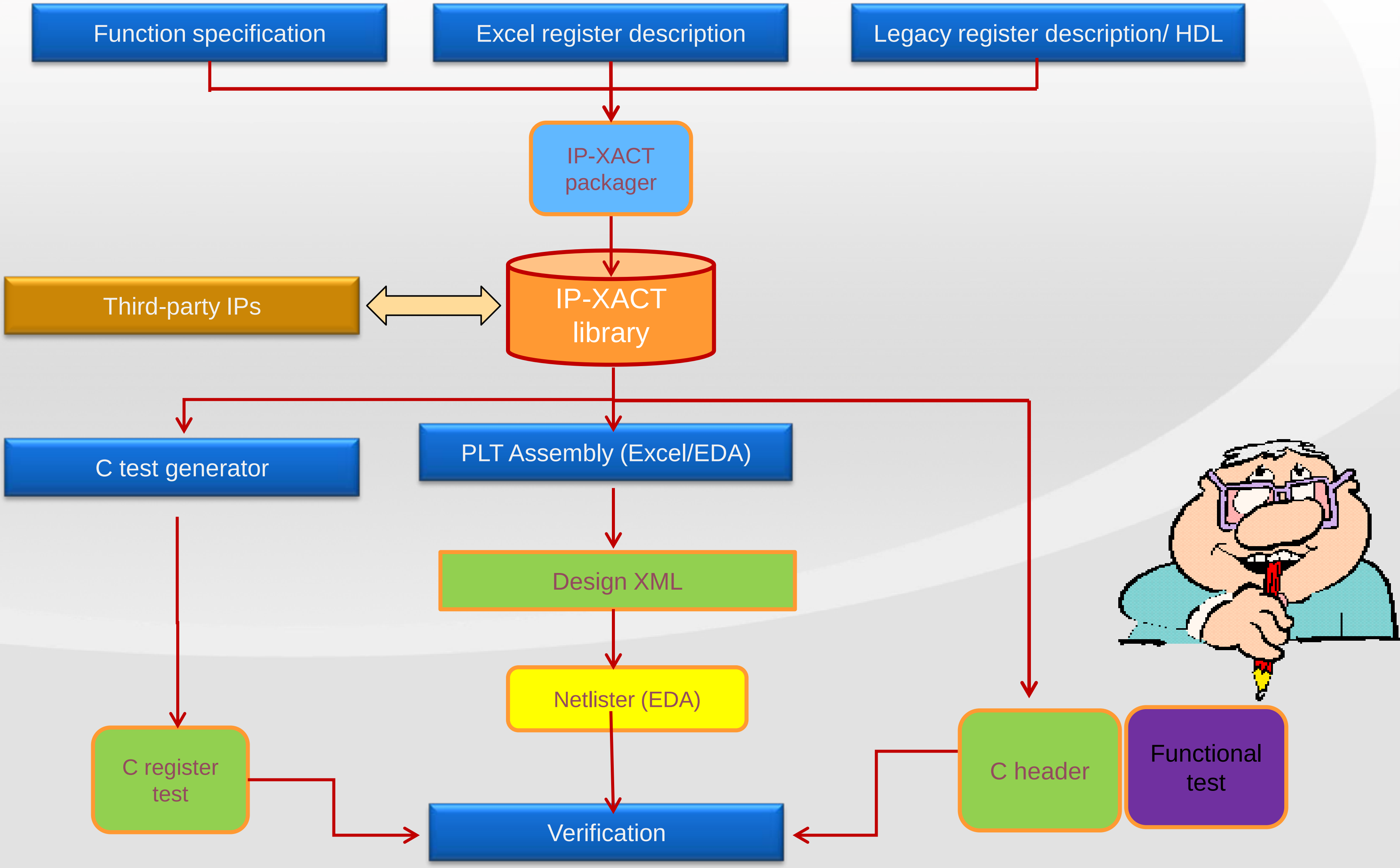
IP-XACT Offerings for Verification Automation

- **Supports verification components**
  - Monitor interfaces
  - White-box interfaces
  - Complete API for metadata exchange and database querying
  - Generator plug-ins support to enable automated configuration
- **Portability across multiple tools, multiple vendors, EDA**
- **Command line tools, GUI based tools, EDA**
- **Provides language and vendor independent description of the testbench configuration and connection to DUT**

Applying IP-XACT for Verification Automation

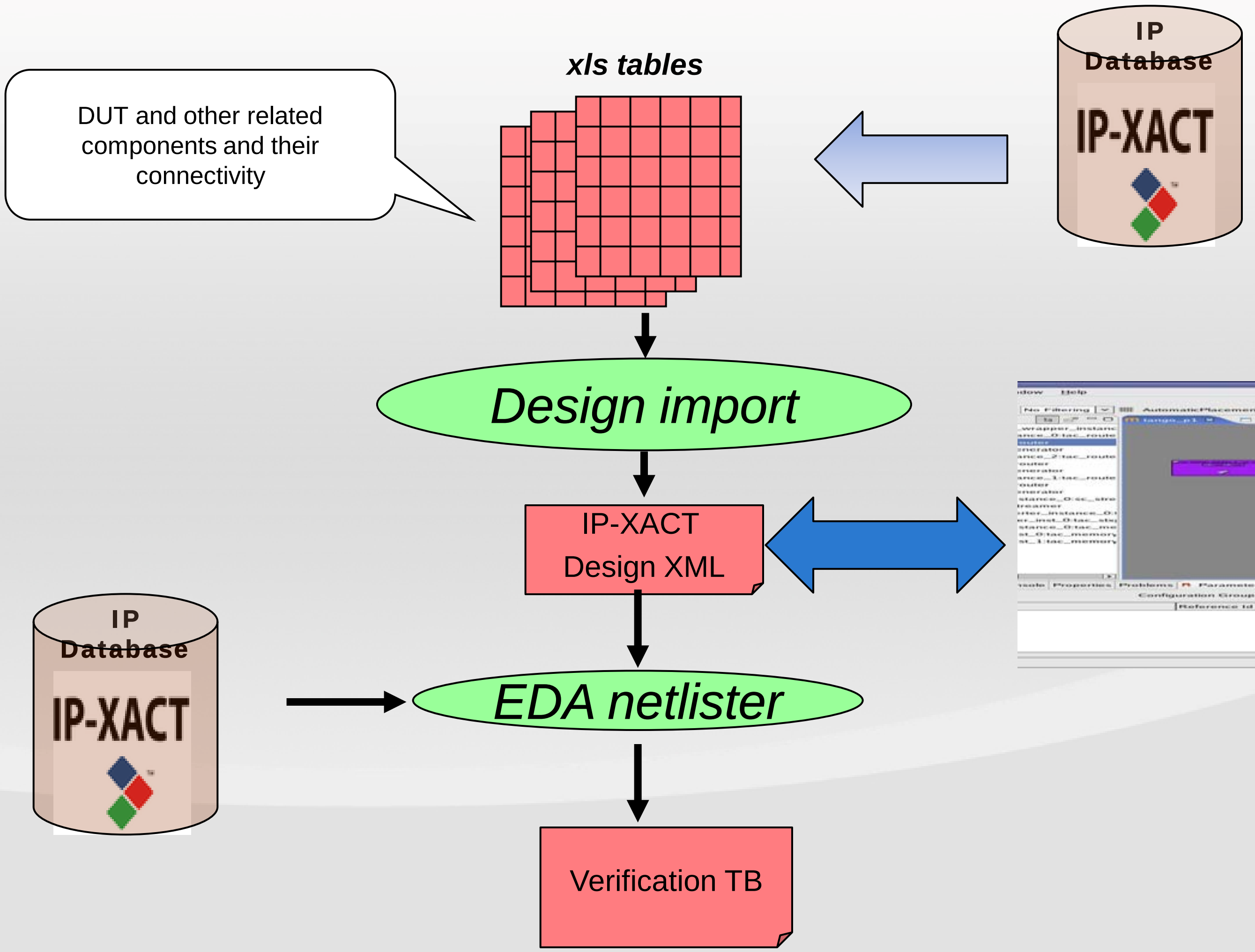
- **Based on IP-XACT (1.4/IEEE 1685-2009)**
- **Automatic IP Packaging in IP-XACT via**
  - Functional specification (Framemaker, Word)
  - HDL (Verilog, VHDL)
  - Legacy format (PMAP), custom Excel descriptions
- **Automatic generation of verification testbenches**
  - From specs, Excel, EDA GUI, etc.
  - In TLM, RTL, mixed TLM-RTL abstraction
- **Quickly adaptable to any change in DUT, design, etc.**
- **Reusable across different design teams and different projects**

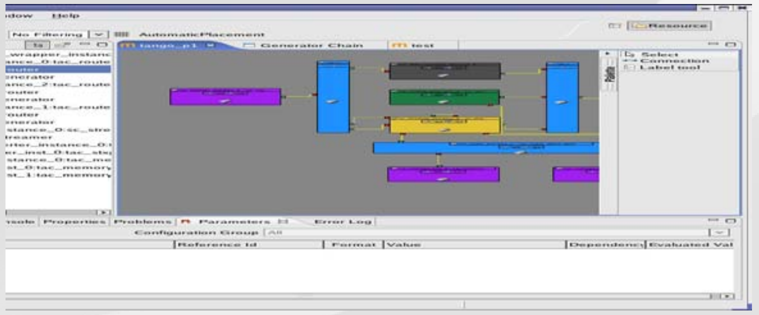
Applying IP-XACT for Verification Automation



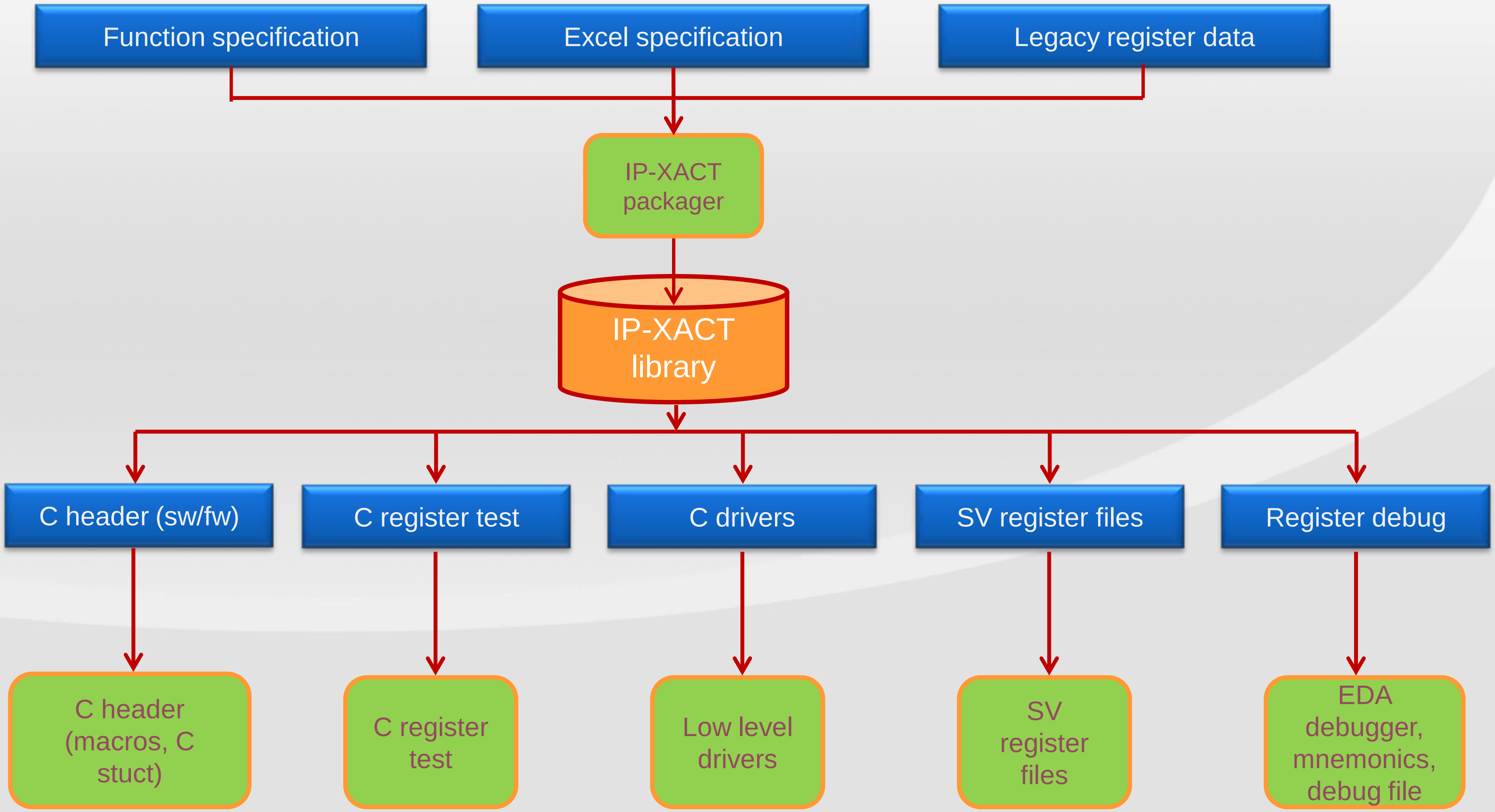
IP-XACT

Applying IP-XACT for Verification Automation





Applying IP-XACT for Verification Automation



- Quickly adaptable to specification changes
- Single source ensures coherency

Applying IP-XACT for Verification Automation

register Input format.

- *Function spec*
- *Excel register description*
- *Legacy format for registers(PMAP)*
- *Register gui*
- *Others*

Single Source (IP-XACT XML)

```
<spirit:name>MCR</spirit:name>
<spirit:addressOffset>0x0000</spirit:addressOffset>
<spirit:size>32</spirit:size>
<spirit:access>read-write</spirit:access>
<spirit:reset>
 <spirit:value>0x00004001</spirit:value>
</spirit:reset>
<spirit:field>
<spirit:name>MSTR</spirit:name>
<spirit:bitOffset>31</spirit:bitOffset>
<spirit:bitWidth>1</spirit:bitWidth>
<spirit:access>read-write</spirit:access>
…
</spirit:field>
```

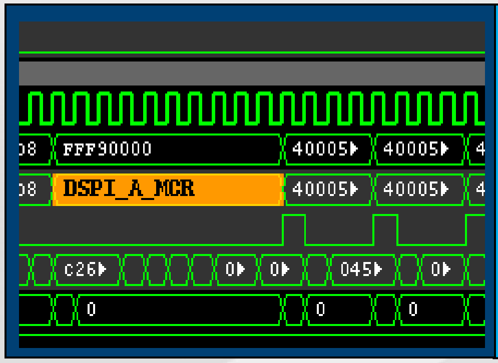
C header

```
#define MCR_SIZE     (32)
#define MCR_OFFSET (0x4)
#define MCR_RESET_VALUE (0x6)
#define MCR_BITFIELD_MASK (0xFFFFF
…
..
#define DATAREADY_OFFSET  (0x0)
#define DATAREADY_WIDTH     (1)
#define DATAREADY_MASK    (0x1)
```

C register test

```
errorNbr += VALregister_test_32(address, data)
pattern = 0x55555555;
errorNbr += RWregister_L_test_32(address,
                      pattern,uart_data_RWMASK);
pattern = 0xAAAAAAAA;
errorNbr += RWregister_L_test_32(address,
                     pattern,uart_data_RWMASK);
```

Mnemonic Map



ASM

```
# MCR - Module Configuration Register
.equ DSPI_A_MCR, (DSPI_A_REGS_BASE+0x0)
```

SV

```
/* MCR - Module Configuration Register */
`define DSPI_A_MCR  (`REG_BASE + 32'h0000_0000)
```

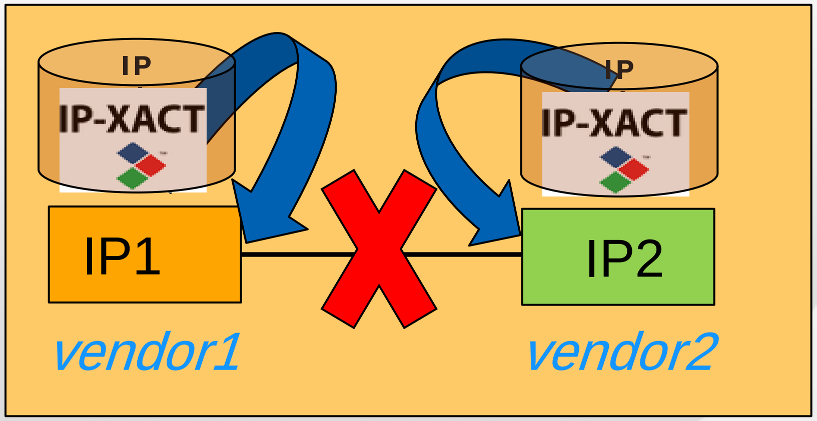
C

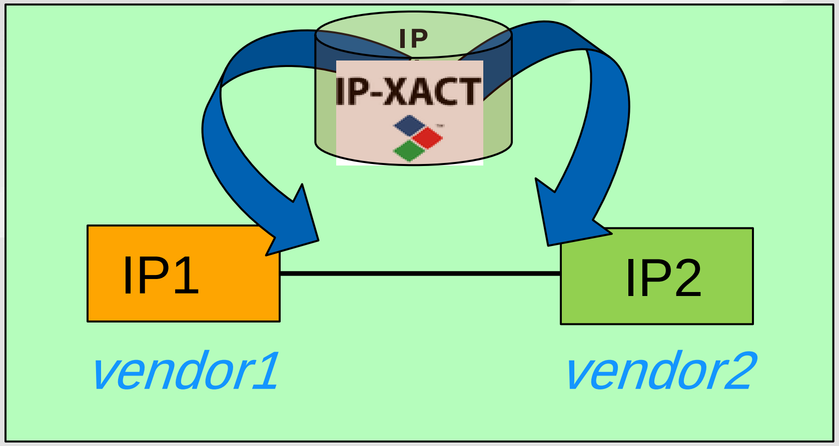
```
/* MCR - Module Configuration Register */
#define DSPI_A_MCR   (*(vuint32_t *) (DSPI_A_BASEADDRESS+0x0))
```

Overcoming Flow Challenges

- **Bus-definition misalignment**
  - Integration issues due to misalignment in bus definitions
  - Use of own copy of bus definition/ abstraction definition of same protocol
  - Inconsistency and misalignment between different teams

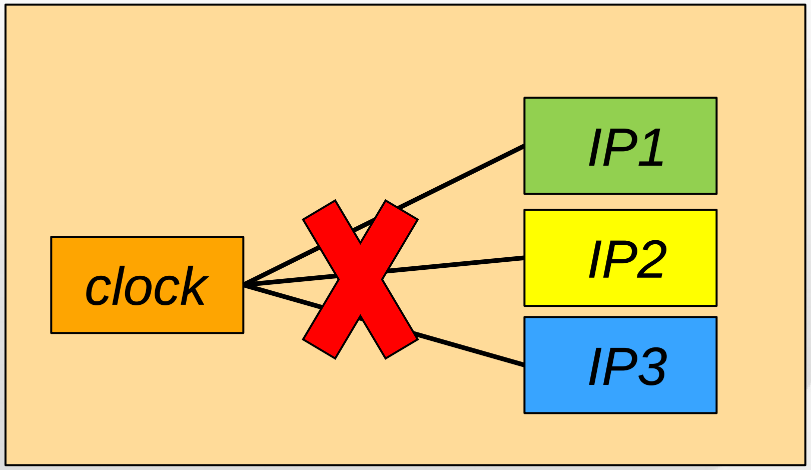
- **Solutions**
  - Standardize generic bus definitions
  - Centralized bus definitions used by different teams

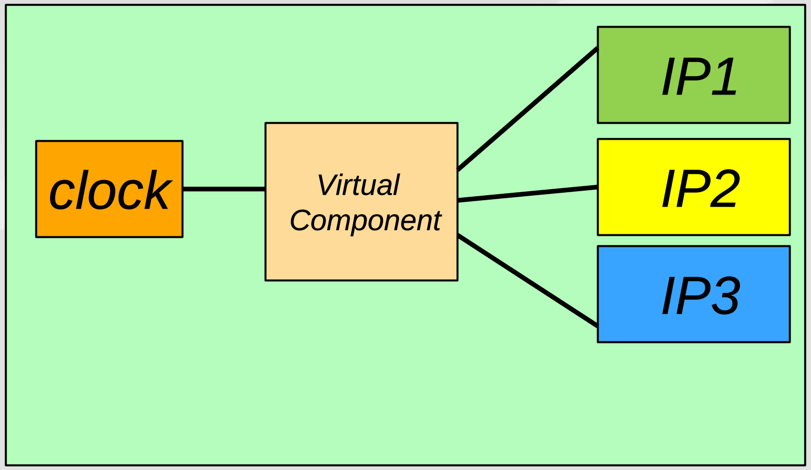




Overcoming Flow Challenges

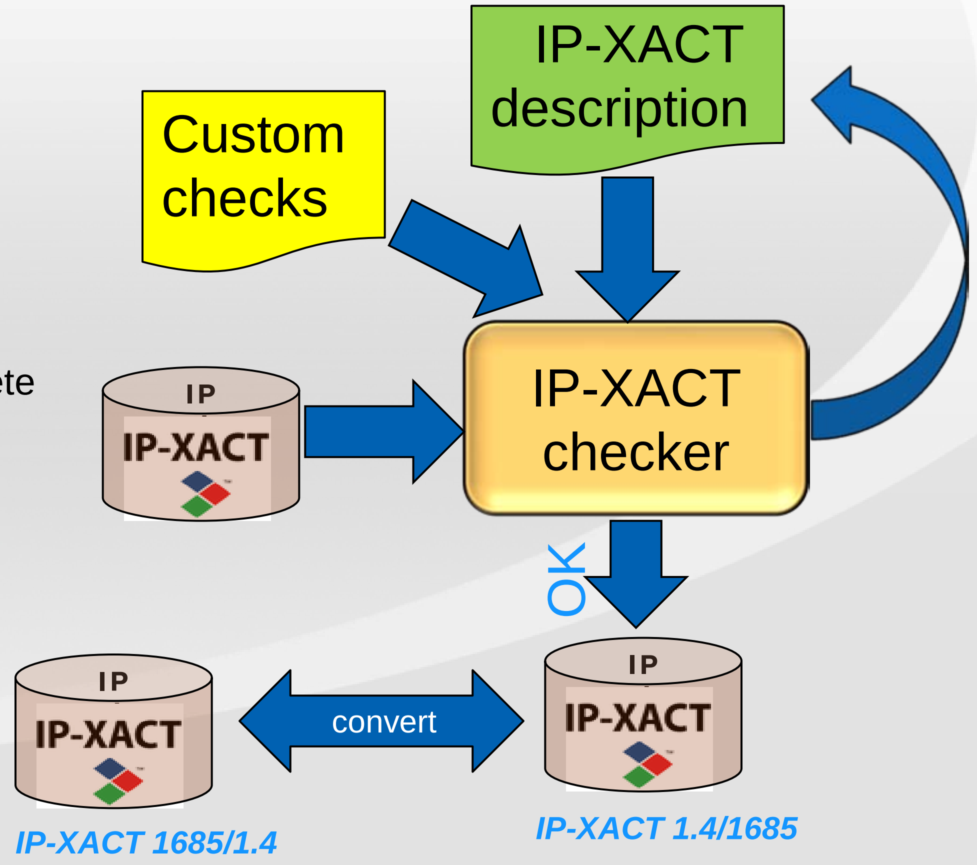
- **One-to-many connections for bus interfaces are not allowed in IP-XACT**
  - « IP-XACT SCR 2.3 : A particular component/bus interface combination shall appear in only one interconnection element in a design »
- **Solutions**
  - EDA tools allow one to many connections (with a warning)
    - violates IP-XACT compliance
  - Auto-insertion of a virtual component to manage the one-to-many connection
  -





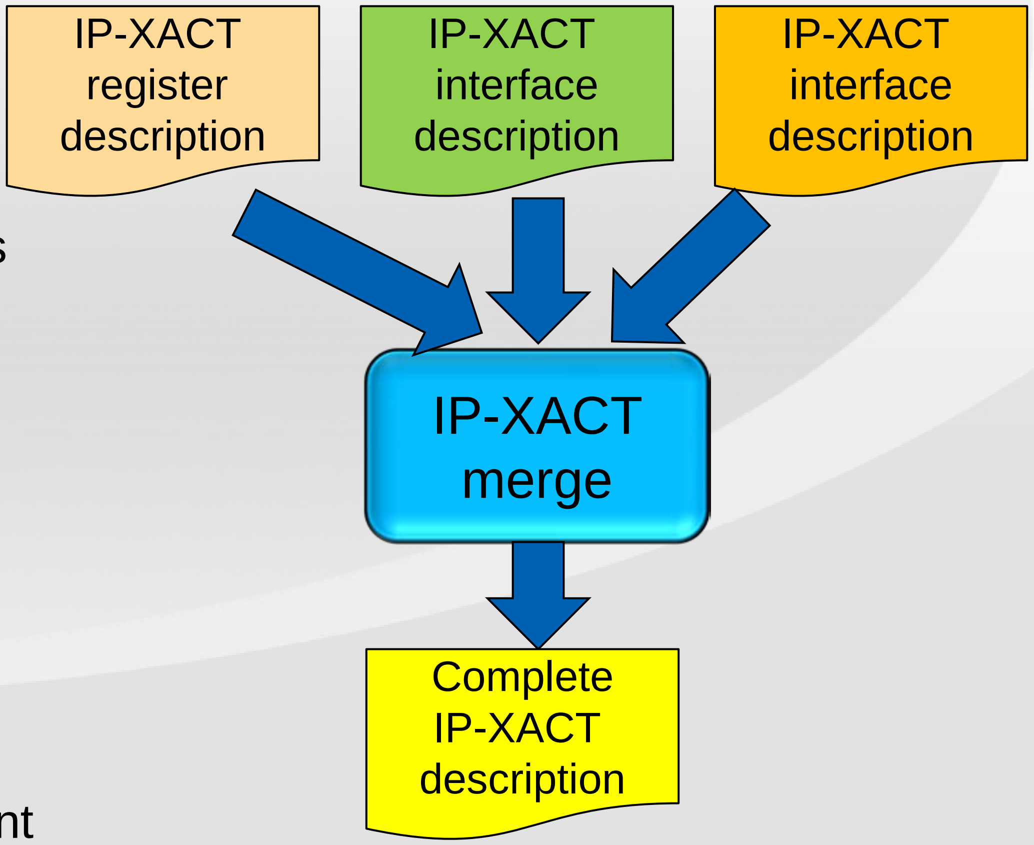
Overcoming Flow Challenges

- **Integration issues due to incomplete/incorrect IP-XACT descriptions**
  - **Solution**
    - Built a set of utilities to create and complete the IP-XACT descriptions
    - Built/use checker utility to ensure
      - IP-XACT description compliancy w.r.t. IP-XACT schema, semantic rules
      - IP-XACT description compliancy w.r.t. to custom requirements for specific flows
- **Integration issue due to different schema versions**
  - **Solution**
    - Built convertors to align on schema version (1.4 => 1685-2009)



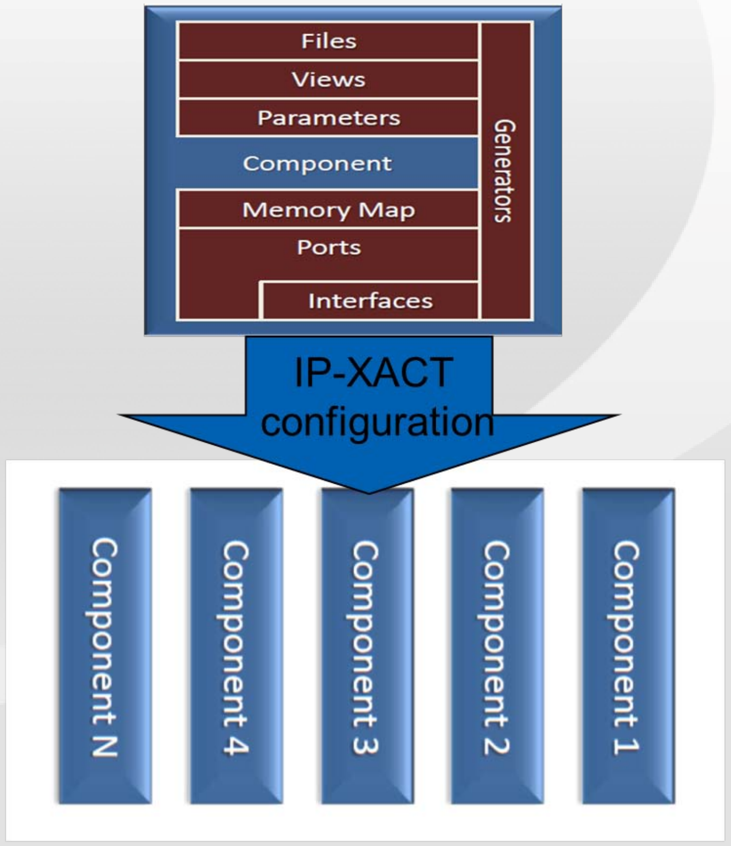
Overcoming Flow Challenges

- **Multiple IP-XACT view addressing different needs**
  - Different teams responsible for assembly, verification, etc. results multiple IP-XACT files
  - Registers from spec/Excel
  - Interfaces from HDL
  - Additional information (fileSets, etc.)
- **Solution**
  - Built a utility to merge the several IP-XACT descriptions with different information


IP-XACT register description
IP-XACT interface description
IP-XACT interface description
IP-XACT merge
Complete IP-XACT description

Overcoming Flow Challenges

- **Configurable IP-XACT descriptions and TGI limitations**
  - TGI APIs are not capable enough to handle generic IP-XACT component descriptions
    - Accellera Systems Initiative IP-XACT TC requirement 42, SWG
  - Not easy to handle configurable IPs
- **Solution**
  - Defined a set of vendor extensions to specify configurability (plan to be standardize later in Accellera/IEEE)
    - No of ports, registers
    - Presence, absence of interfaces/registers/ports
    - Many more
  - Built a generic generator based on the predefined vendor extensions to generate configured IP-XACT description

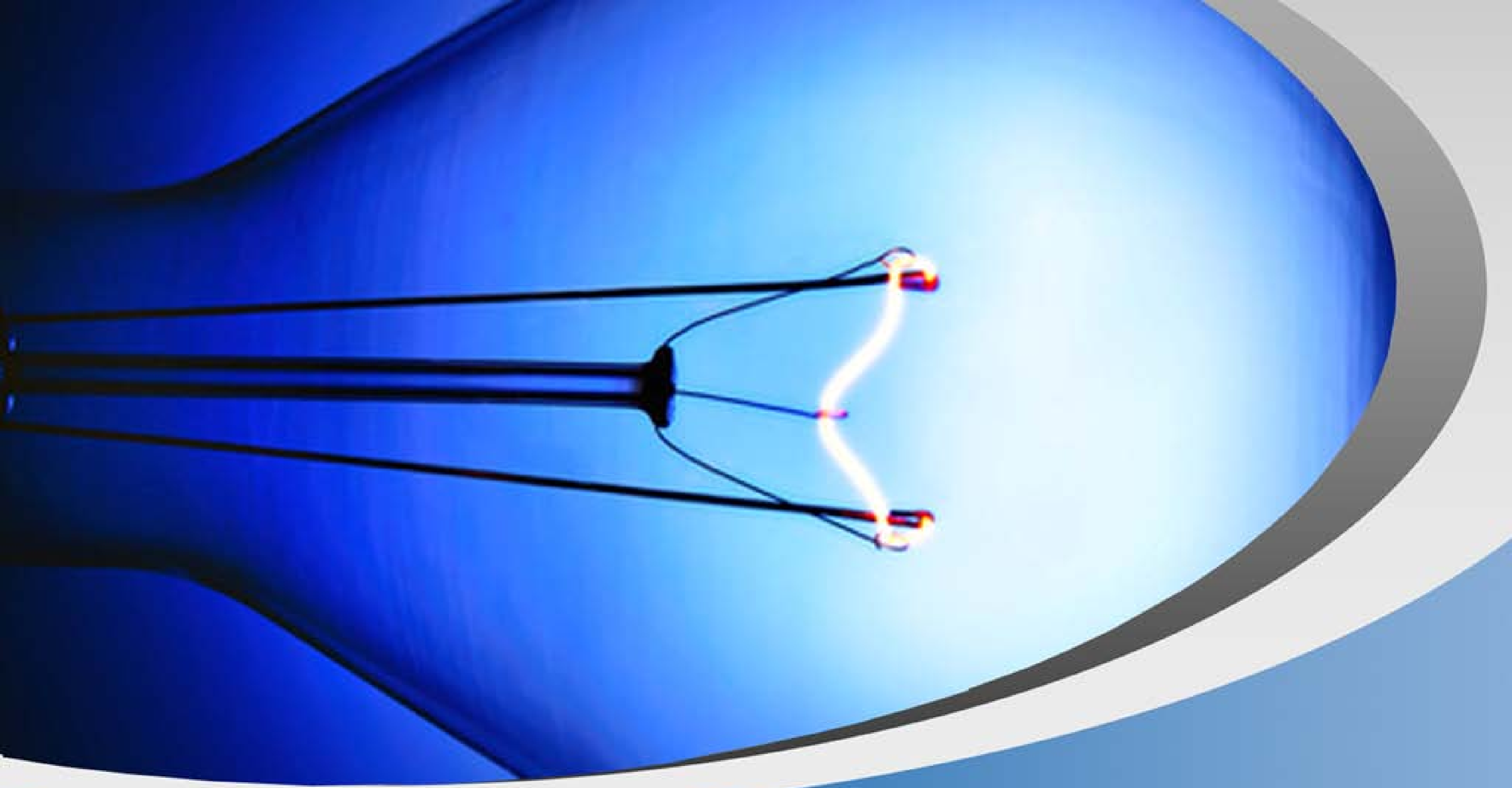


Overcoming Flow Challenges

- **Register configurability**
  - Added specific vendor extensions to specify the configurability and standalone generator based on these vendor extensions to create configured IP-XACT
- **Register side effects**
  - Added specific vendor extensions to handle register side effects
- **Iterated register descriptions**
  - Compact notation to describe iterated registers in spec
  - Added specific vendor extensions to describe iterated registers
- **Special registers behaviors**
  - Added specific vendor extensions to describe special registers and their behaviors
- **Custom flow to address specific needs and legacy**
  - Through specific vendor extensions
  - Through command line, GUI options
- **UVM specific needs**
  - Some specific vendor extensions has been added to address specific needs w.r.t. UVM (to be standardized in Accellera Systems Initiative)

Conclusion

- IP-XACT simplified integration, verification
- Automatic flow to avoid manual repetitive jobs
- Maximum reuse, no duplication
- Quickly adaptable to any changes
- Ensure coherency with other design teams
- Standard allows multi-vendor IPs/EDA tools use



Verification and Automation Improvement Using IP-XACT

Members of IP-XACT Technical Committee

IP-XACT™

DVCon 2012 Design & Verification Conference & Exhibition

accellera SYSTEMS INITIATIVE

IP-XACT and UVM

David Murray

Duolog Technologies

Agenda

- Introduce IP-XACT/UVM standards
- Look at benefits of using these standards together
- Use Case: HW/SW interface verification automation
- Conclusions

What is IP-XACT (IEEE-1685)

- **IP-XACT is an XML format that defines and describes electronic components and their designs.**
- **The goals of the standard are**
  - to ensure delivery of compatible component descriptions from multiple component vendors,
  - to enable exchanging complex component libraries between electronic design automation (EDA) tools for SoC design (design environments),
  - to describe configurable components using metadata, and
  - to enable the provision of EDA vendor-neutral scripts for component creation and configuration (generators, configurators).

UVM – Universal Verification Methodology

- **Goal**
  - Provide a single, open standard to deliver verification productivity within design teams and across multi-company design and verification collaborative efforts
- **Advanced verification methodology**
  - Coverage-Driven Verification
  - Randomization, phasing, coverage, scoreboard
- **HW/SW applications**
  - Contains a ‘Register Package’
  - Predefined test cases
- **Benefits**
  - Open standard: Accellera Systems Initiative
  - Advanced Verification Capability
  - Interoperability & Reuse
  - HW/SW interface verification productivity

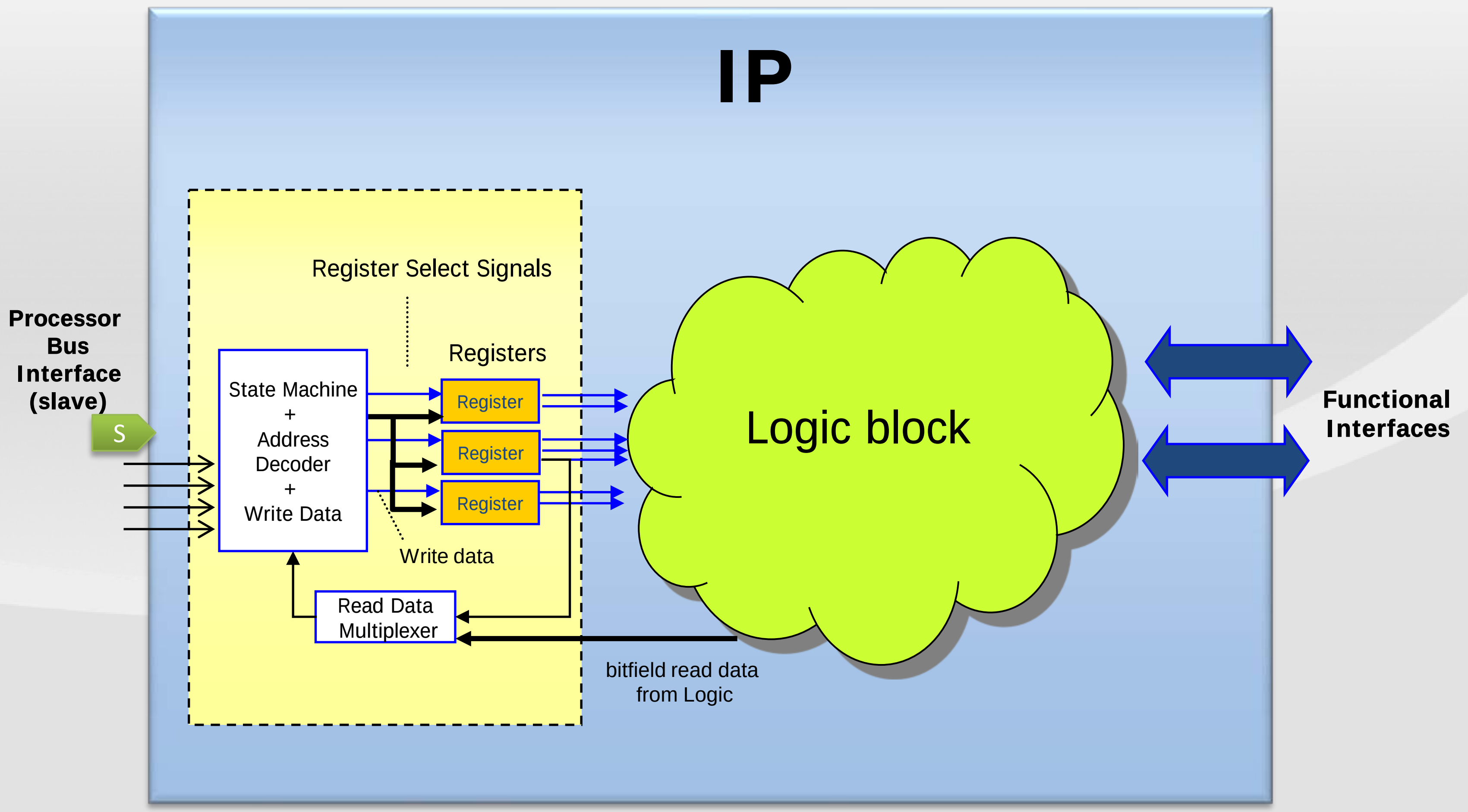
Benefits of using these standards together

- **IP-XACT can be a single source specification for IP metadata**
  - Specification is standardized and leads to :
    - Less ambiguity
    - Higher quality because of SCR checks
    - Higher levels of automation through generators
    - High levels of interoperability
- **UVM provides advanced verification capabilities**
  - High level of HW/SW verification capability using the built-in UVM test sequences
  - Randomization, phasing, coverage, scoreboard
- **If we can leverage the two standards we can get significant levels of verification automation and productivity**
  - Does IP-XACT link well with UVM?
  - **Focus**: Let’s investigate HW/SW interface verification

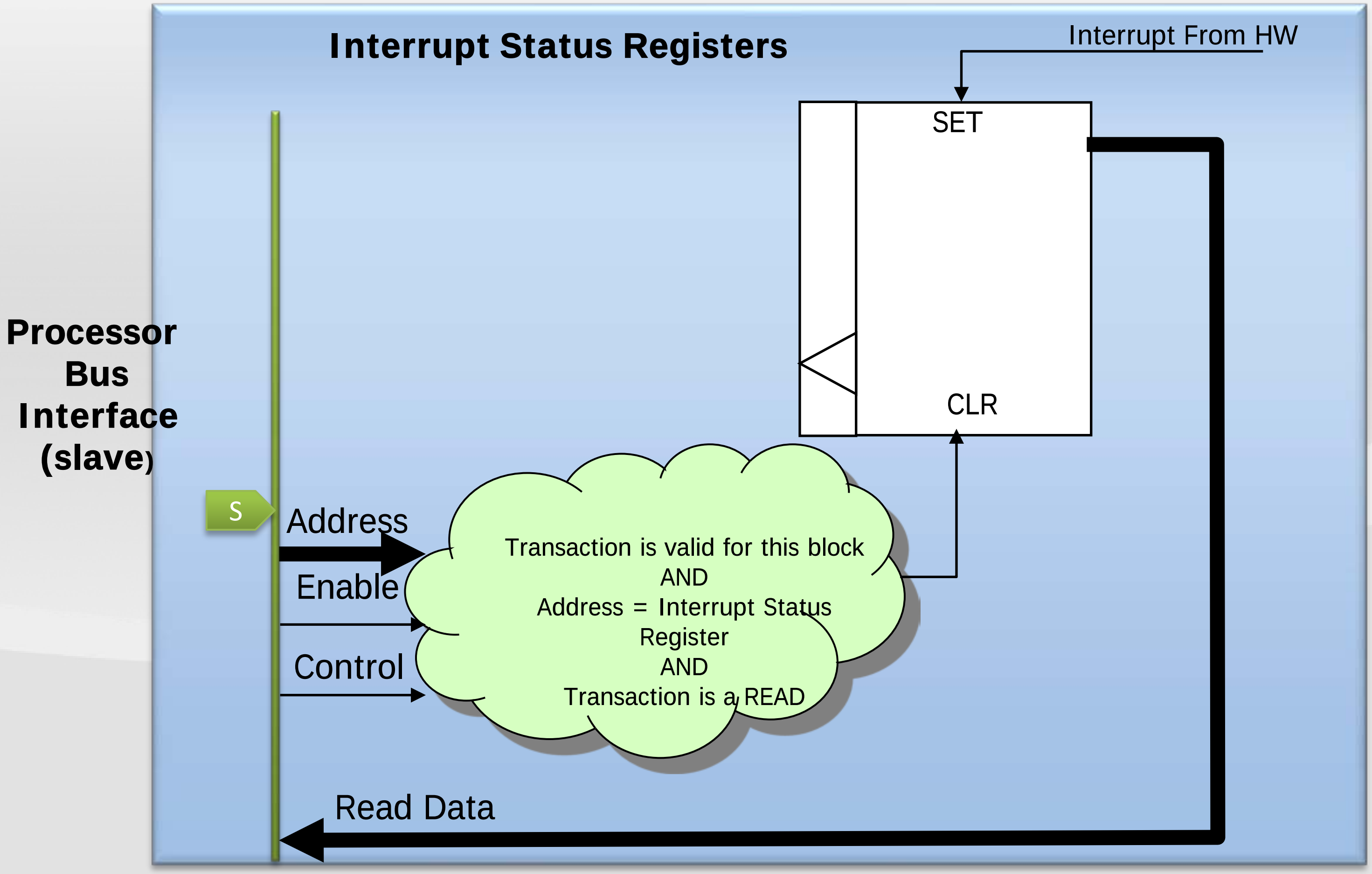
Use-Case: Automating HW/SW Interface Verification

- Overview of HW/SW interface
- UVM Register Model
- IP-XACT Register model
- IP-XACT⬄ UVM mapping

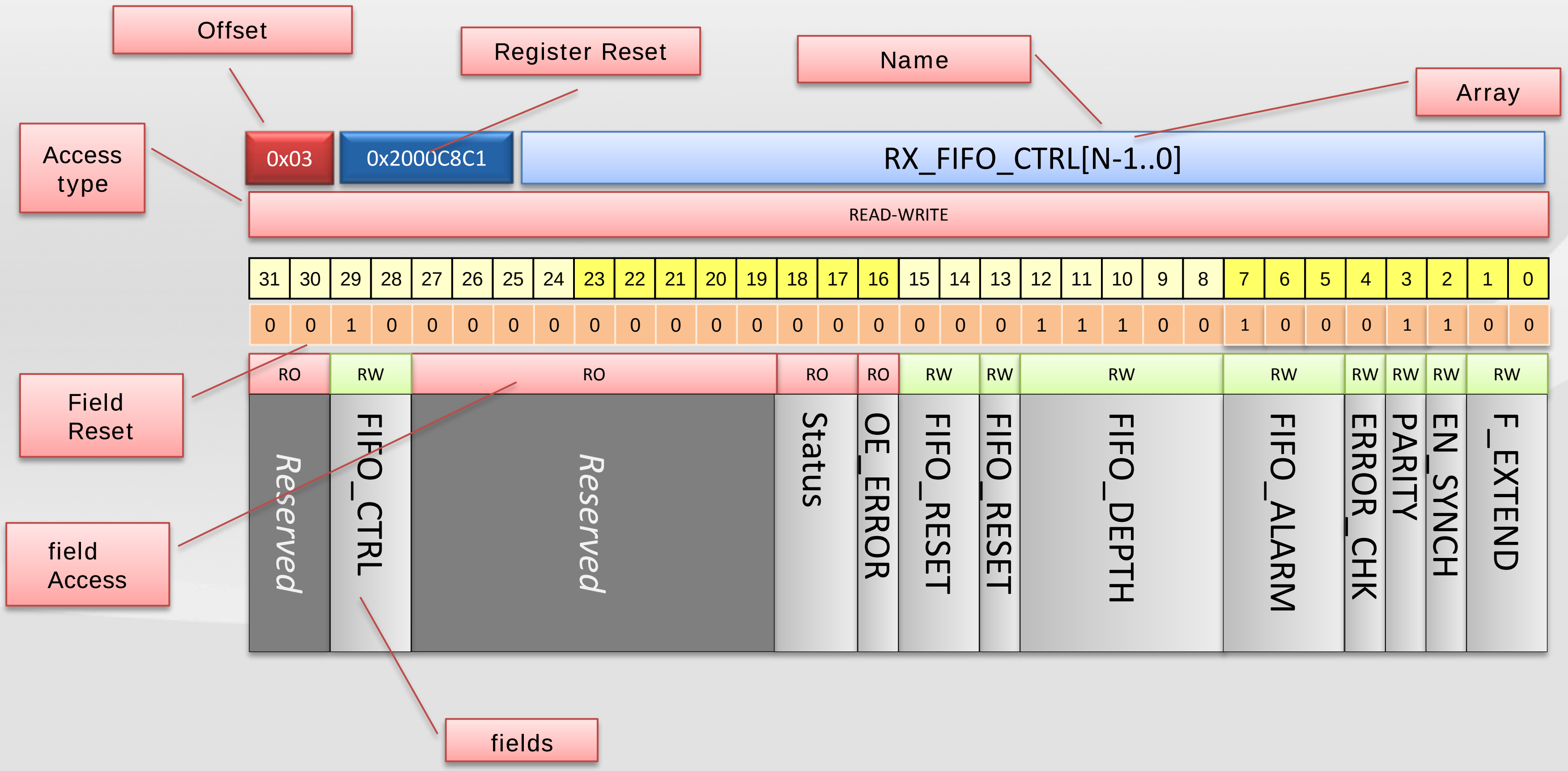
HW/SW Interface: IP/Component



HW/SW Interface: Interrupt Register

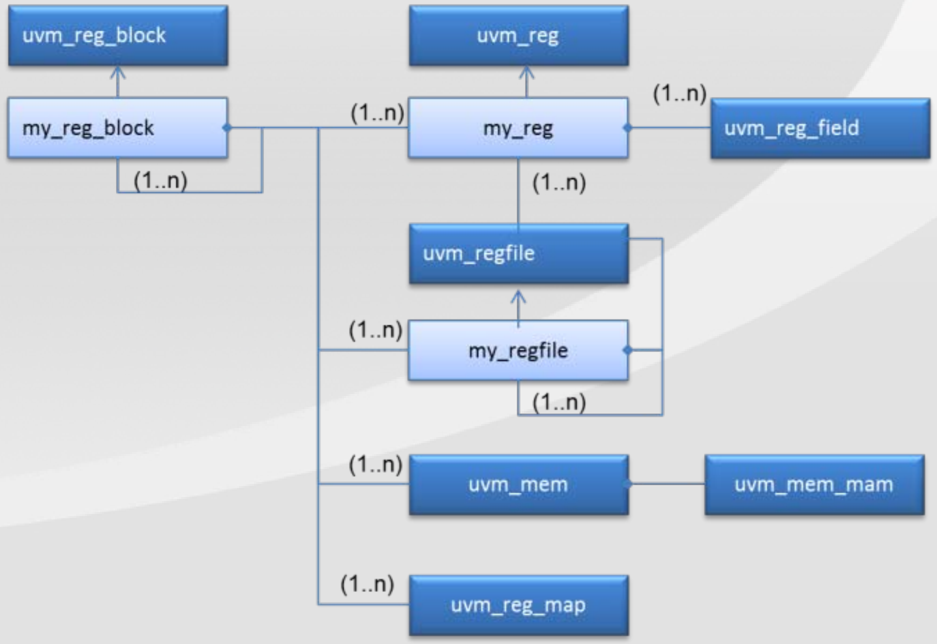

Interrupt Status Registers
Interrupt From HW
SET
CLR
Processor Bus Interface (slave)
S
Address
Enable
Control
Transaction is valid for this block AND Address = Interrupt Status Register AND Transaction is a READ
Read Data

HW/SW Interface: Registers

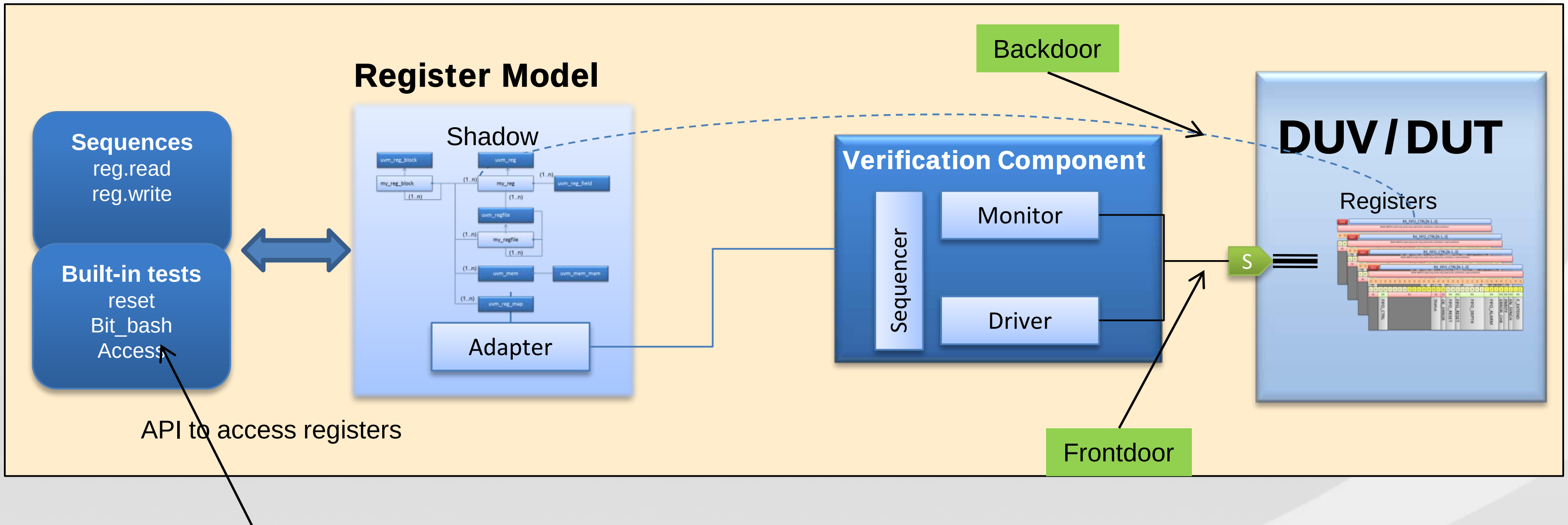


UVM Register Modeling

- **UVM contains a set of** *register layer classes* **which are used to..**
  - Create a high-level, object-oriented model that define the structure and behavior of memory-mapped registers/memories
  - Abstract the read/write operations to registers and memories in a DUT/DUV
  - Provide a test sequence library with predefined test cases which can be used to verify the correct operation of registers



UVM Environment



Predefined Test Sequence	Description
uvm_reg_hw_reset_seq	Reads all the register in a block and check their value is the specified reset value.
uvm_reg_single_bit_bash_seq	Sequentially writes 1's and 0's in each bit of the register, checking it is appropriately set or cleared, based on the field access policy specified for the field containing the target bit.
uvm_reg_bit_bash_seq	Executes the uvm_reg_single_bit_bash_seq sequence for all registers in a block and sub-blocks.
uvm_reg_single_access_seq	For each address map in which the register is accessible, writes the register then confirms the value was written using the back-door. Subsequently writes a value via the backdoor and checks the corresponding value can be read through the address map.
uvm_reg_shared_access_seq	Requires the register be mapped in multiple address maps. For each address map in which the register is accessible, writes the register via one map then confirms the value was written by reading it from all other address maps.

Creating a UVM Register Model (Manual)

A Manual process is very error prone.
Is the RTL Aligned to the spec?

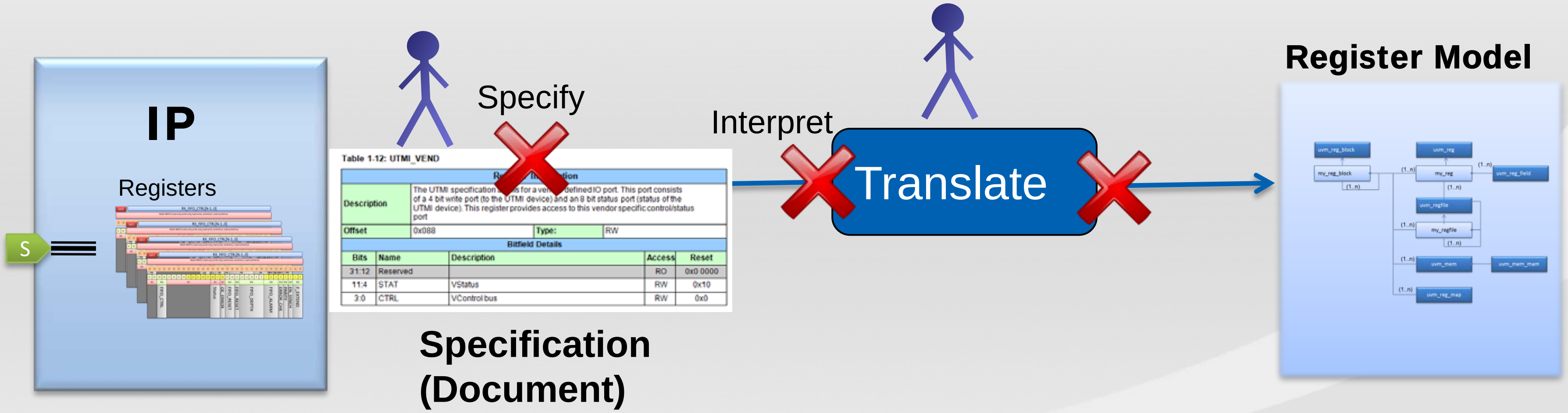


Table 1-12: UTMI_VEND

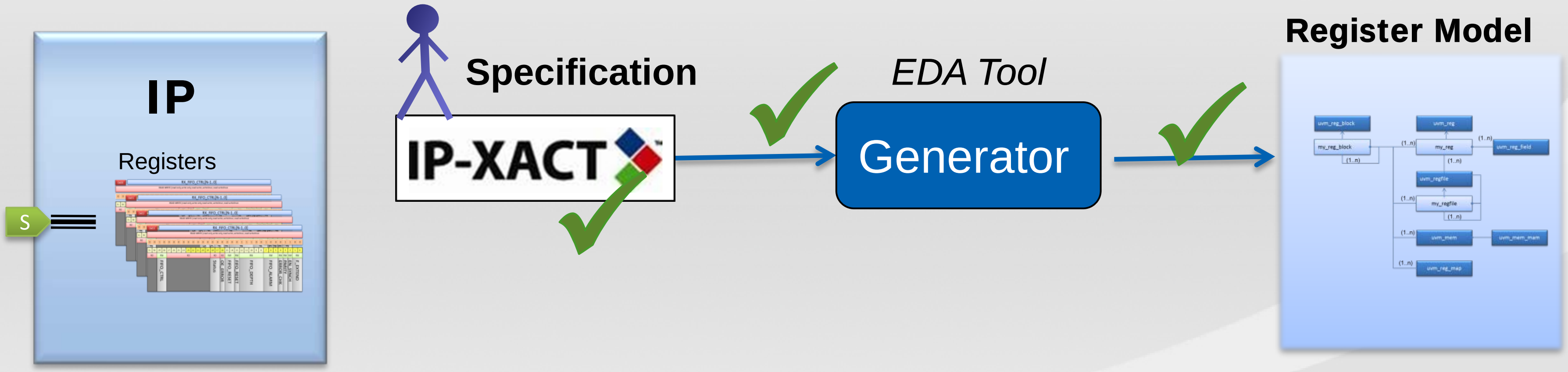
R[illegible] In[illegible]tion			
Description	The UTMI specification [illegible]s for a ven[illegible] defined IO port. This port consists of a 4 bit write port (to the UTMI device) and an 8 bit status port (status of the UTMI device). This register provides access to this vendor specific control/status port		
Offset	0x088	Type:	RW

Bitfield Details				
Bits	Name	Description	Access	Reset
31:12	Reserved		RO	0x0 0000
11:4	STAT	VStatus	RW	0x10
3:0	CTRL	VControl bus	RW	0x0

- ***UVM User Guide:*** *"Due to the large number of registers in a design and the numerous small details involved in properly configuring the UVM register layer classes, this specialization is normally done by a model generator. Model generators work from a specification of the registers and memories in a design and thus are able to provide an up-to-date, correct-by-construction register model.*

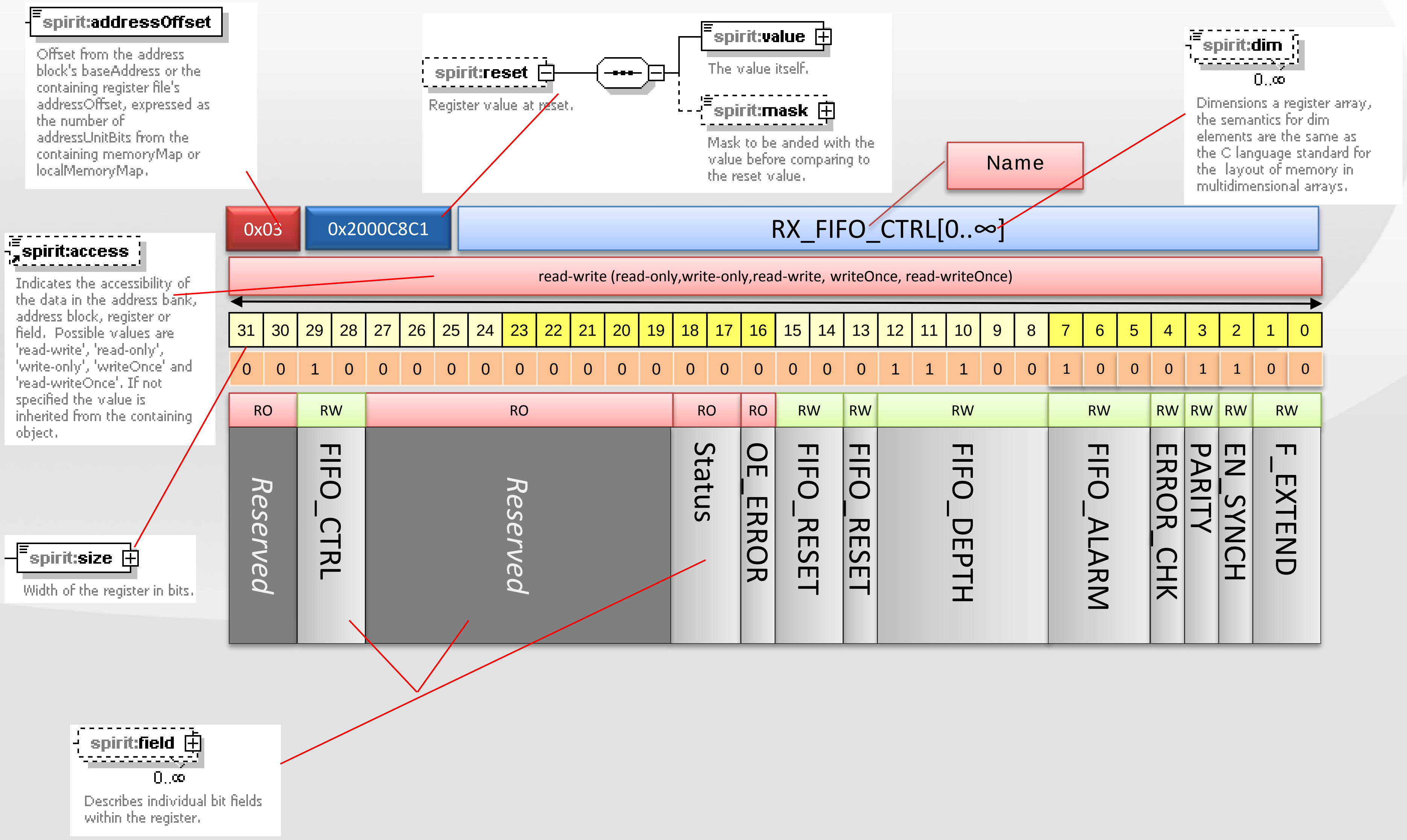
Creating a UVM Register Model (Automatic)

IP-XACT can be used as a source specification for registers



HOWEVER -> How well do they match?

IP-XACT Register Description



spirit:addressOffset

Offset from the address block's baseAddress or the containing register file's addressOffset, expressed as the number of addressUnitBits from the containing memoryMap or localMemoryMap.

spirit:reset

Register value at reset.

spirit:value

The value itself.

spirit:mask

Mask to be anded with the value before comparing to the reset value.

spirit:dim

0..∞

Dimensions a register array, the semantics for dim elements are the same as the C language standard for the layout of memory in multidimensional arrays.

spirit:access

Indicates the accessibility of the data in the address bank, address block, register or field. Possible values are 'read-write', 'read-only', 'write-only', 'writeOnce' and 'read-writeOnce'. If not specified the value is inherited from the containing object.

spirit:size

Width of the register in bits.

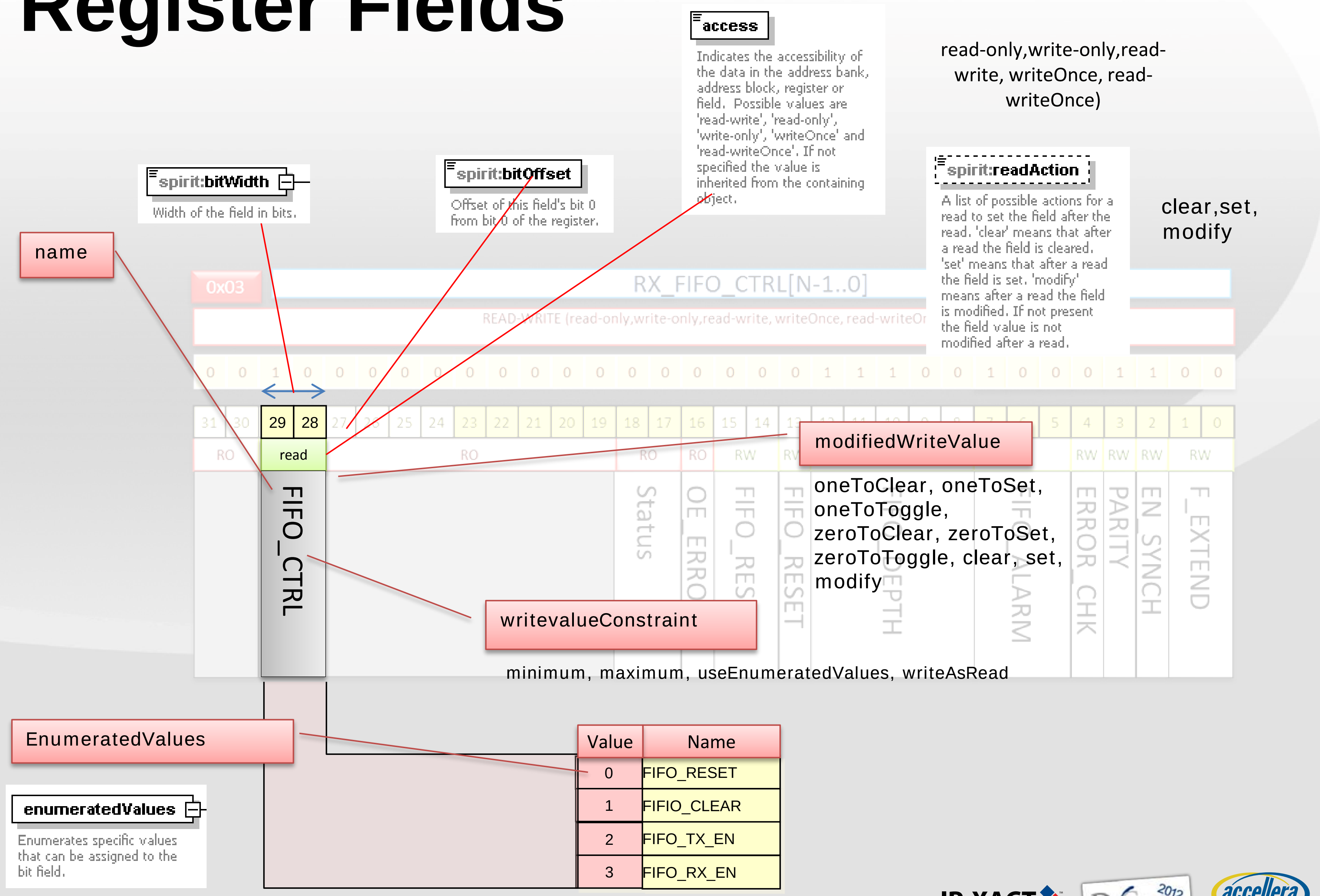
spirit:field

0..∞

Describes individual bit fields within the register.

IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

Register Fields



access

Indicates the accessibility of the data in the address bank, address block, register or field. Possible values are 'read-write', 'read-only', 'write-only', 'writeOnce' and 'read-writeOnce'. If not specified the value is inherited from the containing object.

read-only,write-only,read-write, writeOnce, read-writeOnce)

spirit:bitWidth

Width of the field in bits.

spirit:bitOffset

Offset of this field's bit 0 from bit 0 of the register.

spirit:readAction

A list of possible actions for a read to set the field after the read. 'clear' means that after a read the field is cleared. 'set' means that after a read the field is set. 'modify' means after a read the field is modified. If not present the field value is not modified after a read.

clear,set, modify

oneToClear, oneToSet, oneToToggle, zeroToClear, zeroToSet, zeroToToggle, clear, set, modify

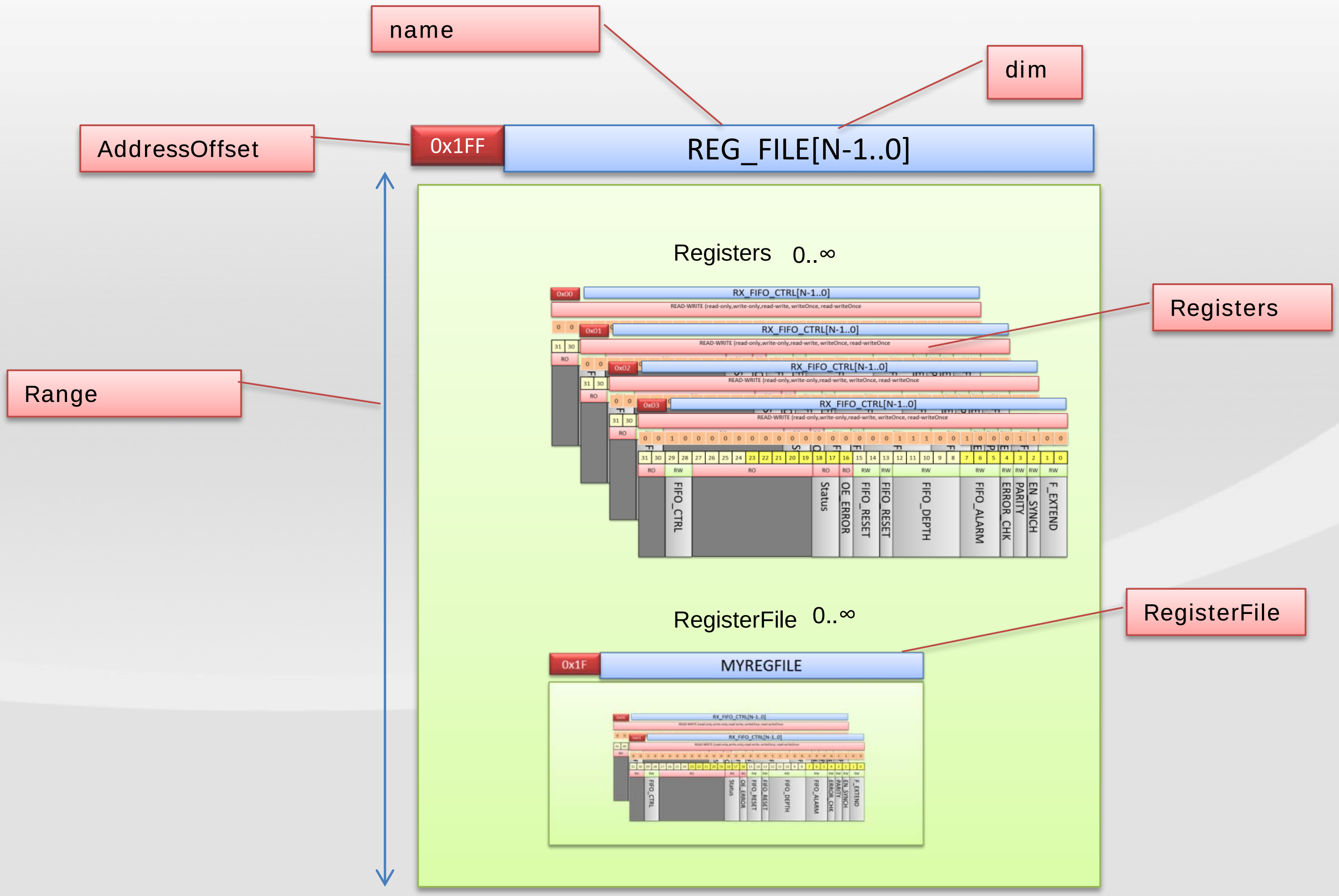
minimum, maximum, useEnumeratedValues, writeAsRead

Value	Name
0	FIFO_RESET
1	FIFIO_CLEAR
2	FIFO_TX_EN
3	FIFO_RX_EN

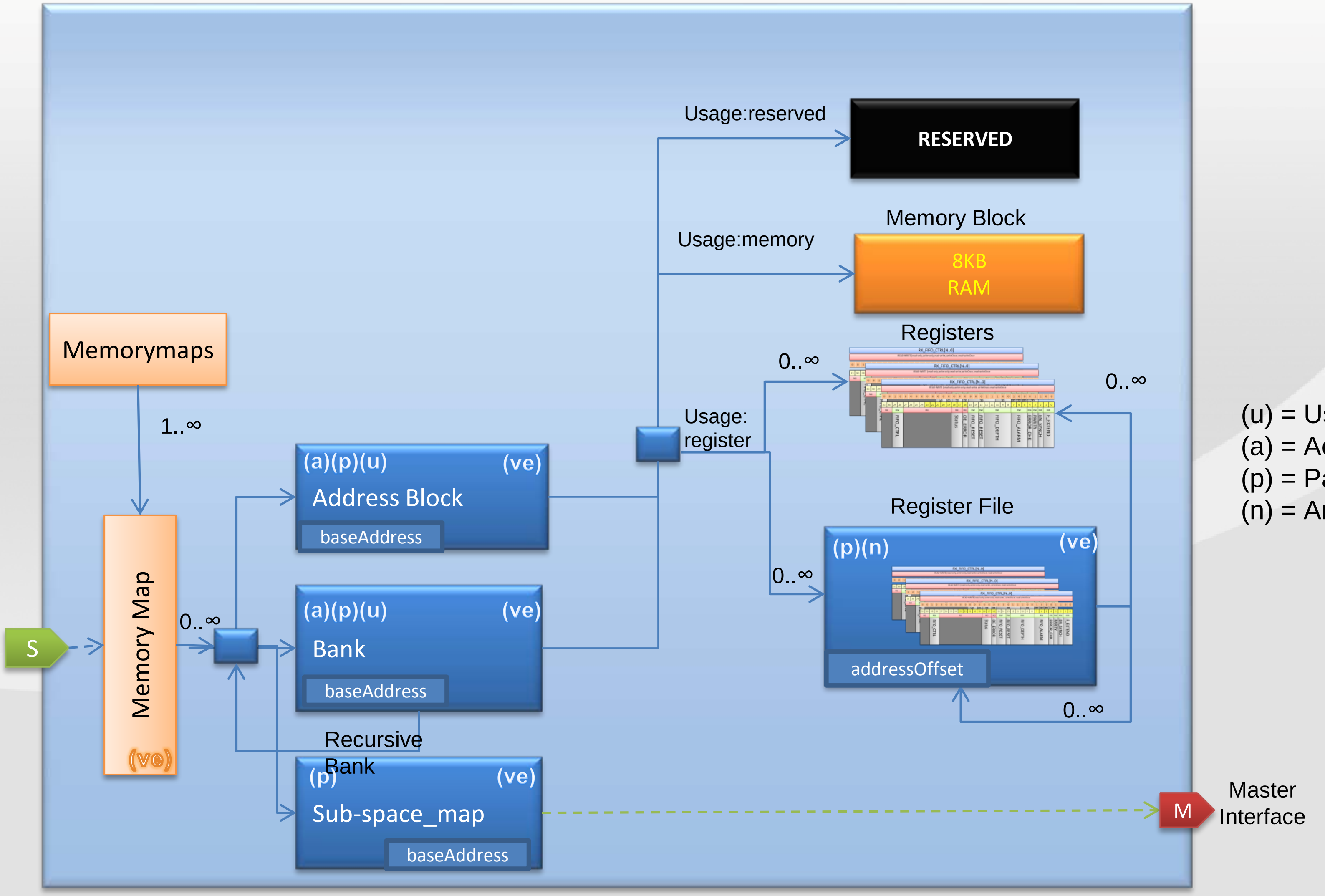
enumeratedValues

Enumerates specific values that can be assigned to the bit field.

Register Files/Blocks

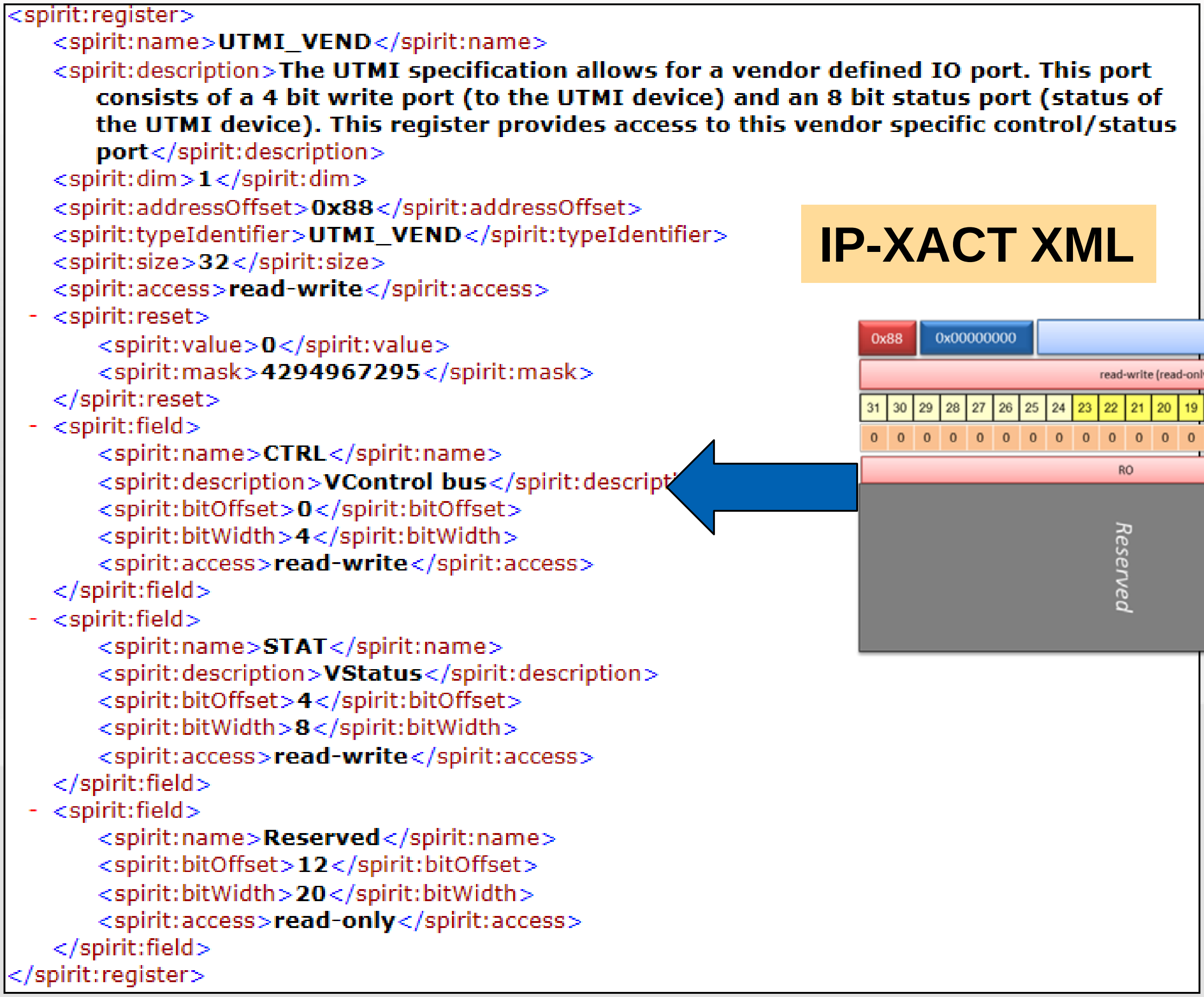


Memory Maps



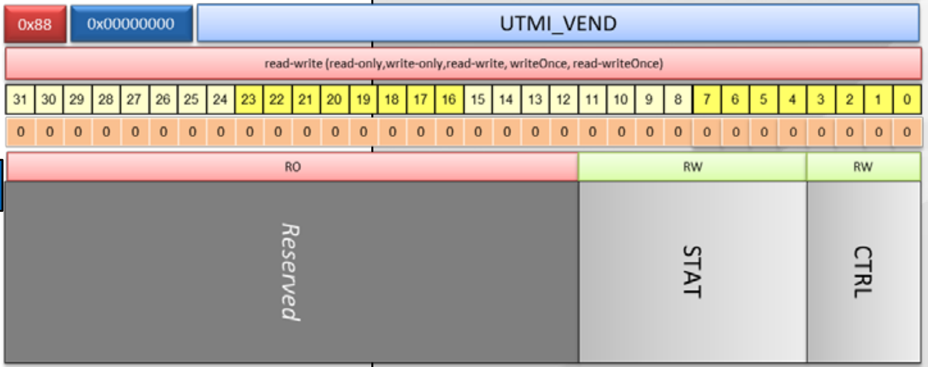
(u) = Usage
(a) = Access
(p) = Parameters
(n) = Array

IP-XACT

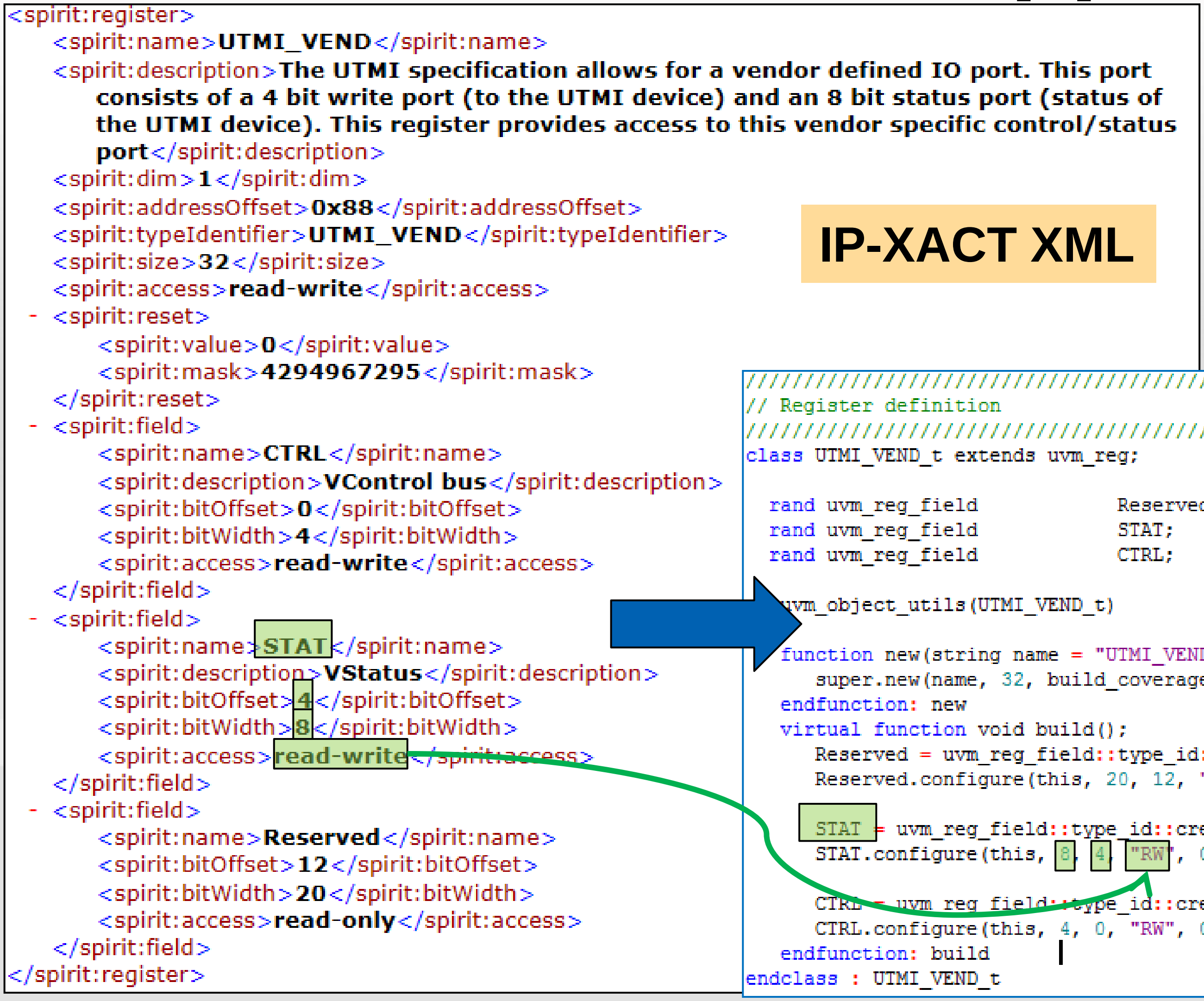


```
<spirit:register>
    <spirit:name>UTMI_VEND</spirit:name>
    <spirit:description>The UTMI specification allows for a vendor defined IO port. This port
        consists of a 4 bit write port (to the UTMI device) and an 8 bit status port (status of
        the UTMI device). This register provides access to this vendor specific control/status
        port</spirit:description>
    <spirit:dim>1</spirit:dim>
    <spirit:addressOffset>0x88</spirit:addressOffset>
    <spirit:typeIdentifier>UTMI_VEND</spirit:typeIdentifier>
    <spirit:size>32</spirit:size>
    <spirit:access>read-write</spirit:access>
  - <spirit:reset>
        <spirit:value>0</spirit:value>
        <spirit:mask>4294967295</spirit:mask>
    </spirit:reset>
  - <spirit:field>
        <spirit:name>CTRL</spirit:name>
        <spirit:description>VControl bus</spirit:descript
        <spirit:bitOffset>0</spirit:bitOffset>
        <spirit:bitWidth>4</spirit:bitWidth>
        <spirit:access>read-write</spirit:access>
    </spirit:field>
  - <spirit:field>
        <spirit:name>STAT</spirit:name>
        <spirit:description>VStatus</spirit:description>
        <spirit:bitOffset>4</spirit:bitOffset>
        <spirit:bitWidth>8</spirit:bitWidth>
        <spirit:access>read-write</spirit:access>
    </spirit:field>
  - <spirit:field>
        <spirit:name>Reserved</spirit:name>
        <spirit:bitOffset>12</spirit:bitOffset>
        <spirit:bitWidth>20</spirit:bitWidth>
        <spirit:access>read-only</spirit:access>
    </spirit:field>
</spirit:register>
```

IP-XACT XML

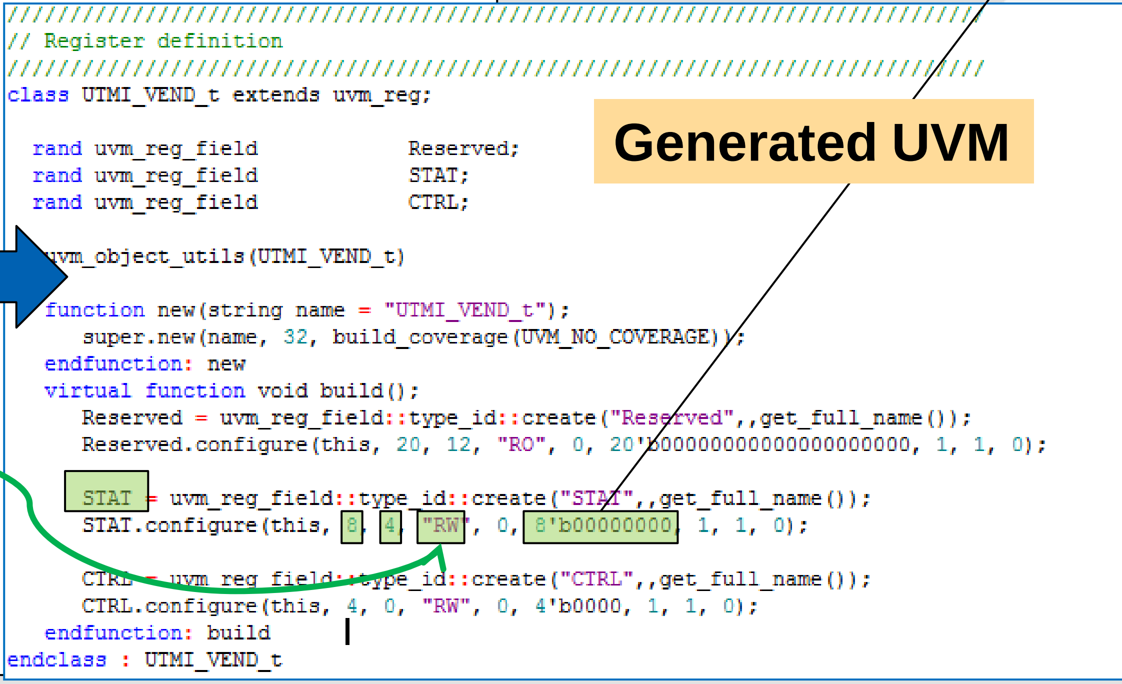


IP-XACT => UVM Mapping



IP-XACT XML

```
<spirit:register>
    <spirit:name>UTMI_VEND</spirit:name>
    <spirit:description>The UTMI specification allows for a vendor defined IO port. This port
        consists of a 4 bit write port (to the UTMI device) and an 8 bit status port (status of
        the UTMI device). This register provides access to this vendor specific control/status
        port</spirit:description>
    <spirit:dim>1</spirit:dim>
    <spirit:addressOffset>0x88</spirit:addressOffset>
    <spirit:typeIdentifier>UTMI_VEND</spirit:typeIdentifier>
    <spirit:size>32</spirit:size>
    <spirit:access>read-write</spirit:access>
  - <spirit:reset>
        <spirit:value>0</spirit:value>
        <spirit:mask>4294967295</spirit:mask>
    </spirit:reset>
  - <spirit:field>
        <spirit:name>CTRL</spirit:name>
        <spirit:description>VControl bus</spirit:description>
        <spirit:bitOffset>0</spirit:bitOffset>
        <spirit:bitWidth>4</spirit:bitWidth>
        <spirit:access>read-write</spirit:access>
    </spirit:field>
  - <spirit:field>
        <spirit:name>STAT</spirit:name>
        <spirit:description>VStatus</spirit:description>
        <spirit:bitOffset>4</spirit:bitOffset>
        <spirit:bitWidth>8</spirit:bitWidth>
        <spirit:access>read-write</spirit:access>
    </spirit:field>
  - <spirit:field>
        <spirit:name>Reserved</spirit:name>
        <spirit:bitOffset>12</spirit:bitOffset>
        <spirit:bitWidth>20</spirit:bitWidth>
        <spirit:access>read-only</spirit:access>
    </spirit:field>
</spirit:register>
```

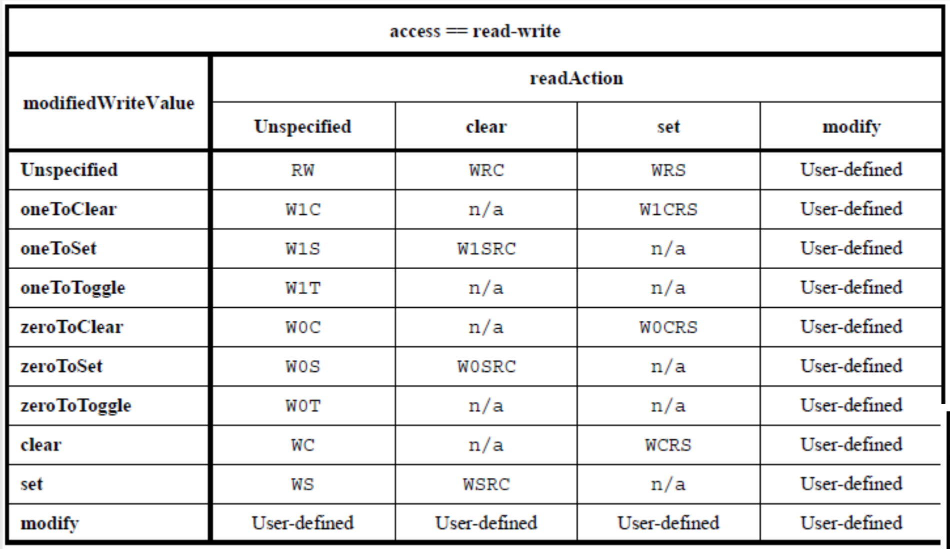


Generated UVM

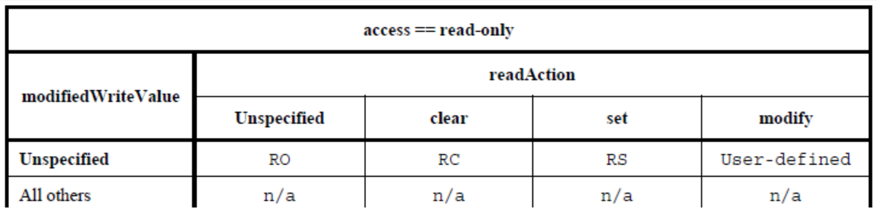
Reset

```
//////////////////////////////////////////////////////////////////////////////////
// Register definition
//////////////////////////////////////////////////////////////////////////////////
class UTMI_VEND_t extends uvm_reg;

  rand uvm_reg_field            Reserved;
  rand uvm_reg_field            STAT;
  rand uvm_reg_field            CTRL;

  `uvm_object_utils(UTMI_VEND_t)

   function new(string name = "UTMI_VEND_t");
      super.new(name, 32, build_coverage(UVM_NO_COVERAGE));
   endfunction: new
   virtual function void build();
      Reserved = uvm_reg_field::type_id::create("Reserved",,get_full_name());
      Reserved.configure(this, 20, 12, "RO", 0, 20'b00000000000000000000, 1, 1, 0);

      STAT = uvm_reg_field::type_id::create("STAT",,get_full_name());
      STAT.configure(this, 8, 4, "RW", 0, 8'b00000000, 1, 1, 0);

      CTRL = uvm_reg_field::type_id::create("CTRL",,get_full_name());
      CTRL.configure(this, 4, 0, "RW", 0, 4'b0000, 1, 1, 0);
   endfunction: build
endclass : UTMI_VEND_t
```

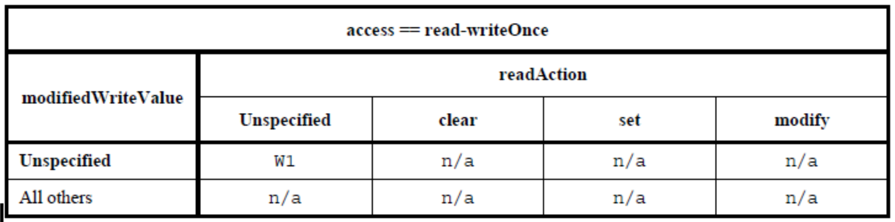
IP-XACT=> UVM Access Mapping



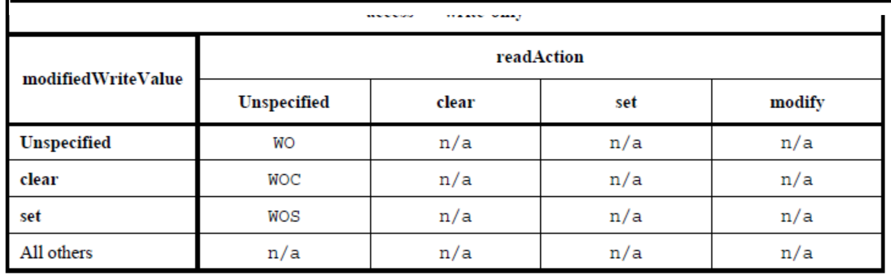
access == read-write				
modifiedWriteValue	**readAction**			
	Unspecified	**clear**	**set**	**modify**
Unspecified	RW	WRC	WRS	User-defined
oneToClear	W1C	n/a	W1CRS	User-defined
oneToSet	W1S	W1SRC	n/a	User-defined
oneToToggle	W1T	n/a	n/a	User-defined
zeroToClear	W0C	n/a	W0CRS	User-defined
zeroToSet	W0S	W0SRC	n/a	User-defined
zeroToToggle	W0T	n/a	n/a	User-defined
clear	WC	n/a	WCRS	User-defined
set	WS	WSRC	n/a	User-defined
modify	User-defined	User-defined	User-defined	User-defined



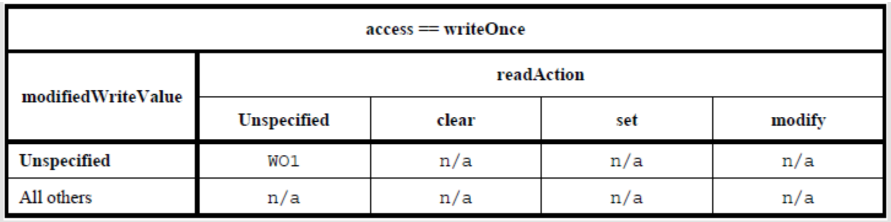
access == read-only				
modifiedWriteValue	**readAction**			
	Unspecified	**clear**	**set**	**modify**
Unspecified	RO	RC	RS	User-defined
All others	n/a	n/a	n/a	n/a



access == read-writeOnce				
modifiedWriteValue	**readAction**			
	Unspecified	**clear**	**set**	**modify**
Unspecified	W1	n/a	n/a	n/a
All others	n/a	n/a	n/a	n/a



access == write-only				
modifiedWriteValue	**readAction**			
	Unspecified	**clear**	**set**	**modify**
Unspecified	WO	n/a	n/a	n/a
clear	WOC	n/a	n/a	n/a
set	WOS	n/a	n/a	n/a
All others	n/a	n/a	n/a	n/a



access == writeOnce				
modifiedWriteValue	**readAction**			
	Unspecified	**clear**	**set**	**modify**
Unspecified	WO1	n/a	n/a	n/a
All others	n/a	n/a	n/a	n/a

Mapping MemoryMaps

IP-XACT XML

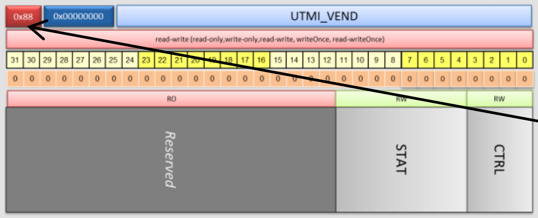
```
- <spirit:memoryMaps>
  - <spirit:memoryMap>
      <spirit:name>USB_MMAP</spirit:name>
    - <spirit:addressBlock>
        <spirit:name>MEM1</spirit:name>
        <spirit:baseAddress spirit:resolve="user">0</spirit:baseAddress>
        <spirit:range spirit:resolve="user">20</spirit:range>
        <spirit:width spirit:resolve="user">32</spirit:width>
        <spirit:usage>register</spirit:usage>
```

In UVM, uvm_reg_blocks are used to describe components, memory maps, banks, adddressBlocks, etc

In UVM Registers are mapped using a map within the block and not on the register itself

Generated UVM

```
///////////////////////////////////////////////////////////////////////////////
// uvm_reg_block definition
///////////////////////////////////////////////////////////////////////////////


class USB2_0_t extends uvm_reg_block;
  rand UTMI_VEND_t UTMI_VEND;

 `uvm_object_utils(USB2_0_t)

  function new(string name="USB2_0_t");
    super.new(name, build_coverage(UVM_NONE));
  endfunction: new

  virtual function void build();
    default_map = create_map("", 0, 4, UVM_BIG_ENDIAN);

    UTMI_VEND = UTMI_VEND_t::type_id::create("UTMI_VEND",, get_full_name());
    UTMI_VEND.configure(this);
    UTMI_VEND.build();
    default_map.add_reg(UTMI_VEND, 'h88,  "RW");

   endfunction : build

endclass : USB2_0_t
```



IP-XACT => UVM Mapping Issues

- **Mapping Issues**
  - Some UVM constructs are not available in IP-XACT, e.g.:
    - HDL Path to Registers
    - Multiple Resets
    - Coverage Control
    - Verification attributes
    - Additional register types (indirect accessed, aliased/mirrored, FIFO)
- **These can still be handled by IP-XACT**
  - Extend IP-XACT using VendorExtensions
  - IP-XACT TC is aiming to standardize these extensions under '**StandardExtensions**'
- **Standardization Route**
  - These requirements included into the next standard: UVM requirements

New IP-XACT Requirements

Requirement Number	Description	Group	PSS	VE(SE) proposal to EWG
REQ26	UVM : Registers/register files may share an address	RWG	Multiple Registers	Yes
REQ43	UVM : Allow multiple reset values for registers	RWG	Multiple Regsiter Resets	Yes
REQ44	UVM : Allow aliased/mirrored registers/register files/memories. Aliased/mirored objects can be accessed on multiple offsets in the memory map with a possible access restrictions.	RWG	Multiple Registers	Yes
REQ45	UVM : Support for indirect accessed registers	RWG	Multiple Registers	Yes
REQ46	UVM : Support for indirectly accessed memories	RWG	Multiple Registers	
REQ47	UVM : Support for FIFO registers	RWG	Multiple Registers	Yes
REQ52	UVM : Verification extensions: backdoor access specification	SWG	UVM Related Attributes	Yes
REQ51	UVM : Support for verification extensions/verification views	SWG	UVM Related Attributes	Yes
REQ53	UVM : Verification extensions: randomization constraints and coverage specifications	SWG	UVM Related Attributes	Yes

- Additional functional requirements not covered in IP-XACT
- Additional register verification attributes
  - HDL Path, Testability, Constraints, Coverage

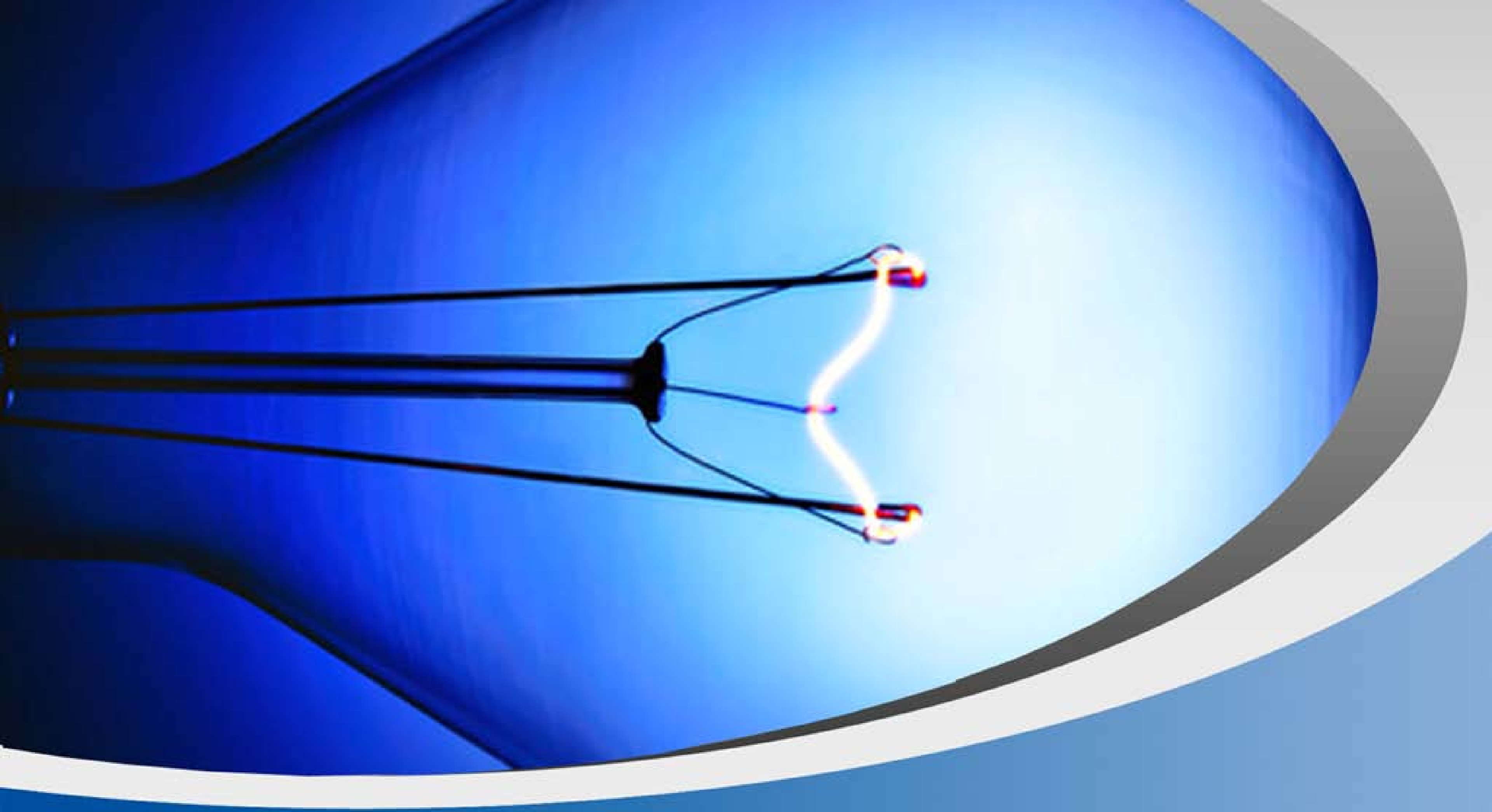
Benefits

- **IP-XACT, as a single source specification for IP, can be used to automatically create UVM Environments**
  - Fewer Specification bugs
  - Fewer interpretation bugs
  - Fewer translation bugs
  - Quicker turn-around time
- **Automation Scopes**
  - HW/SW interface automation
  - Testbench Automation
- **Advanced verification capabilities**
  - Built-in register test sequences give high levels of verification productivity

Faster Time-to-HW/SW Interface Qualification

Conclusion

- UVM delivers advanced verification capabilities
- IP-XACT provides a standardized way to define registers
- UVM can be generated from IP-XACT
- Missing UVM constructs in IP-XACT can be modeled by extensions
- Significant boost HW/SW Interface Verification productivity



Verification and Automation Improvement Using IP-XACT

Members of IP-XACT Technical Committee

IP-XACT™

DVCon 2012 Design & Verification Conference & Exhibition

accellera SYSTEMS INITIATIVE

IP-XACT Extensions

Sylvain Duvillard

Magillem
Design Services

Erwin de Kock

NXP
Semiconductors

Outline

- Introduction
- IP-XACT vendor extensions
- Accellera standard extensions
- Summary
- Acknowledgement

IP-XACT Vendor Extensions

The IP-XACT standard (IEEE 1685-2009) defines

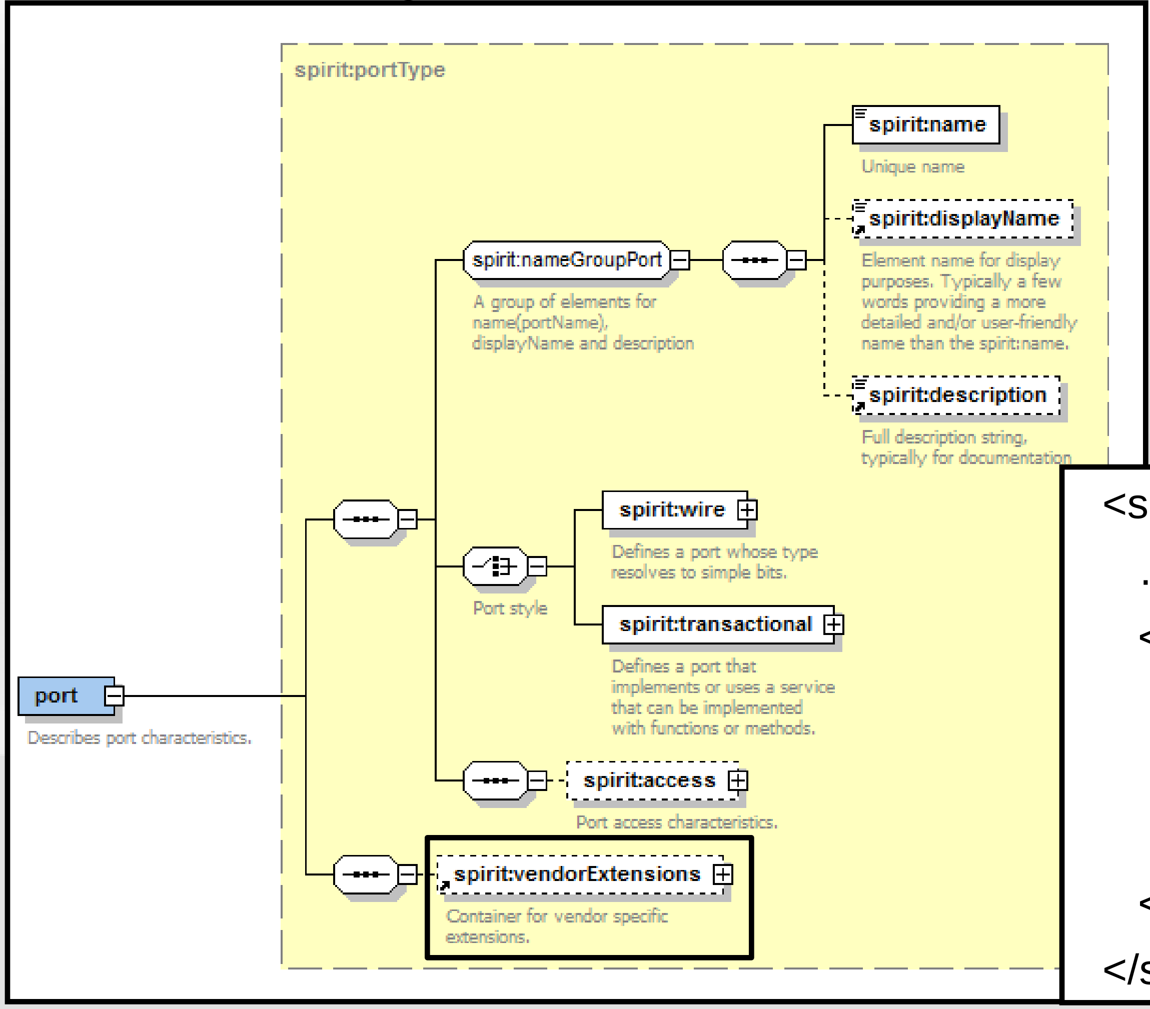
- XML schemas for describing meta-data of IPs, designs, and flows
- Tight Generator Interface for tool access to meta-data

The IP-XACT standard allows extensions

- XML schemas contains extension points
- Tight Generator Interface provides tool access to extension points

Example IP-XACT Vendor Extension

XML schema fragment


spirit:portType
spirit:name
Unique name
spirit:displayName
Element name for display purposes. Typically a few words providing a more detailed and/or user-friendly name than the spirit:name.
spirit:nameGroupPort
A group of elements for name(portName), displayName and description
spirit:description
Full description string, typically for documentation
spirit:wire
Defines a port whose type resolves to simple bits.
Port style
spirit:transactional
Defines a port that implements or uses a service that can be implemented with functions or methods.
port
Describes port characteristics.
spirit:access
Port access characteristics.
spirit:vendorExtensions
Container for vendor specific extensions.

XML document fragment

```
<spirit:port>
  …
  <spirit:vendorExtensions>
    <myNameSpace:myMetaData>
      …
    </myNameSpace:myMetaData>
  </spirit:vendorExtensions>
</spirit:port>
```

Usage of IP-XACT Vendor Extensions

Companies have been using vendor extensions

- To store company specific IP metadata, e.g., verification data
- To implement specific tool features, e.g., GUI related data

European Projects have been using vendor extensions

- To work together on new areas using IP-XACT
- To propose extensions for the IP-XACT standard

Accellera Standard Extensions

Standard Extensions are vendor extensions defined by Accellera

- **Accellera IP-XACT Extensions Working Group**

Goals of Standard Extensions

- **To support IP-XACT usage in areas not covered by the standard yet**
- **To foster cross-company IP-XACT usage in these areas**
- **To prepare and validate potential extensions of the standard**

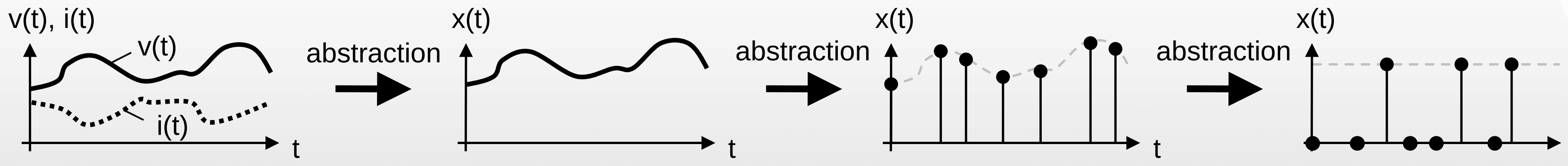
Please join the Accellera IP-XACT EWG if you want to participate

Upcoming Standard Extensions

Currently, the Accellera IP-XACT Extensions Working Group is defining vendor extensions in the following areas

- **Analog-Mixed Signal**
  - Contribution has been received from European Project Beyond Dreams
- **Physical Design Planning**
  - Contribution has been received from ST Microelectronics
- **Power**
  - Contribution has been received from Magillem Design Services
- **Hardware Dependent Software**
  - Contributions have been received from European Project SoftSoc and Vayavya Labs
- **Universal Verification Methodology**
  - Contribution has been received from Accellera VIP-TC represented via Duolog

Analog-Mixed Signal Extensions



Electrical Networks

- Conservative description represented by two dependent quantities, e.g. the voltage *v*(*t*) <u>and</u> the current *i*(*t*)
- Continuous in time and value
- Analog solver will resolve the *Kirchhoff's Laws*
- Called ***'electrical'*** in Verilog-AMS

Covered in IP-XACT extensions

Signal Flow

- Non-conservative description represented by single quantity *x*(*t*), to <u>represent</u> e.g. the voltage <u>or</u> current (not both)
- Continuous in time and value
- Often called ***'voltage'*** or ***'current'*** in Verilog-AMS

Covered in IP-XACT extensions

Discrete-time

- Non-conservative description represented by single quantity *x*(t), to <u>represent</u> e.g. the voltage <u>or</u> current (not both)
- Discrete-time samples only, can hold any arbitrary data type
- Called "real-value-modeling" (RVM), like ***'wreal '*** in Verilog-AMS

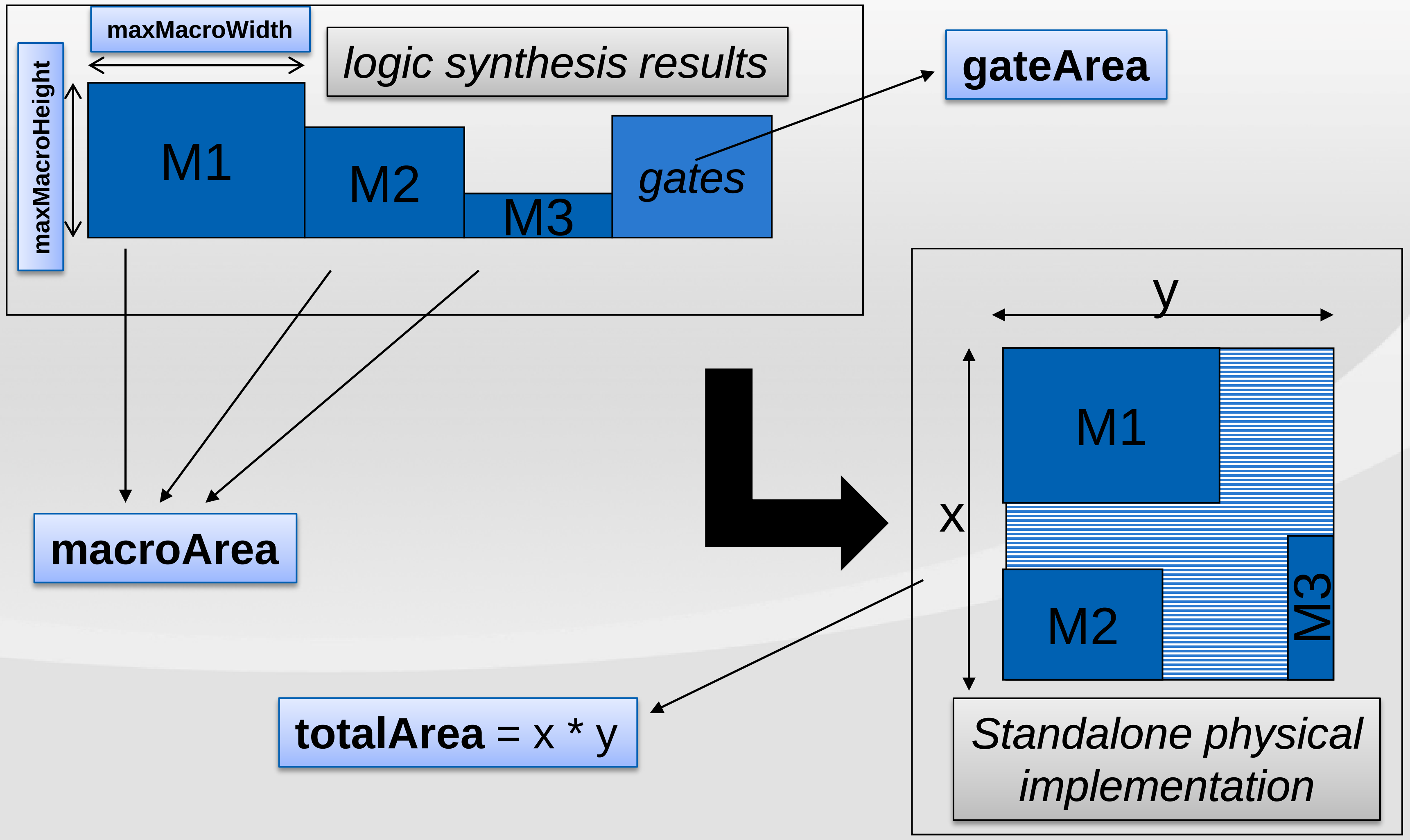
Covered in IP-XACT extensions

Discrete-time & value

- Non-conservative description represented by single quantity *x*(t), to <u>represent</u> whether there is e.g. a voltage or current
- Discrete-time and discrete-value defined as logical "0" , "1", "Z" or "X"
- Called ***'logic'*** in Verilog-AMS

Covered in IP-XACT standard

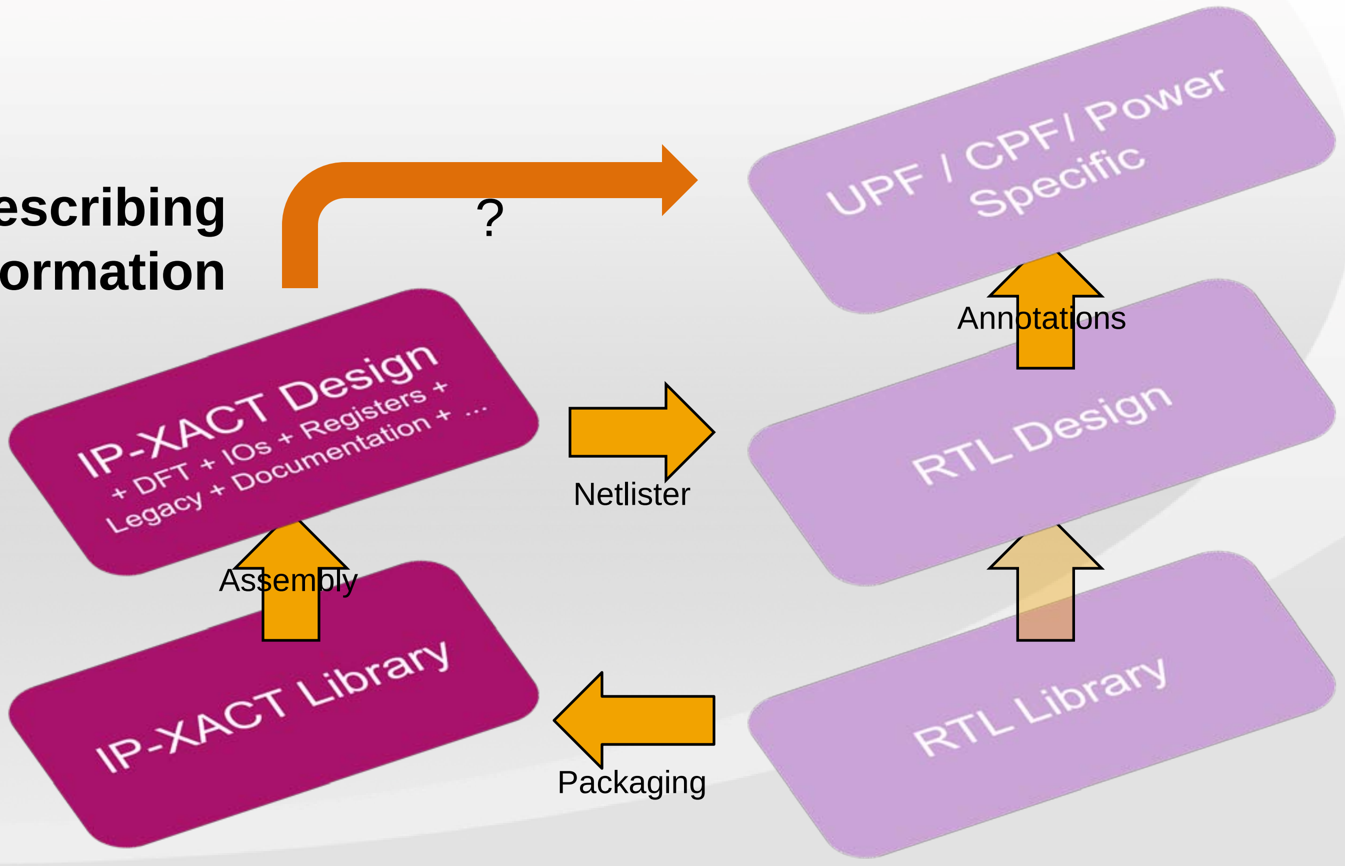
Physical Design Planning Extensions



IP-XACT DVCon 2012 accellera SYSTEMS INITIATIVE

Power Extensions

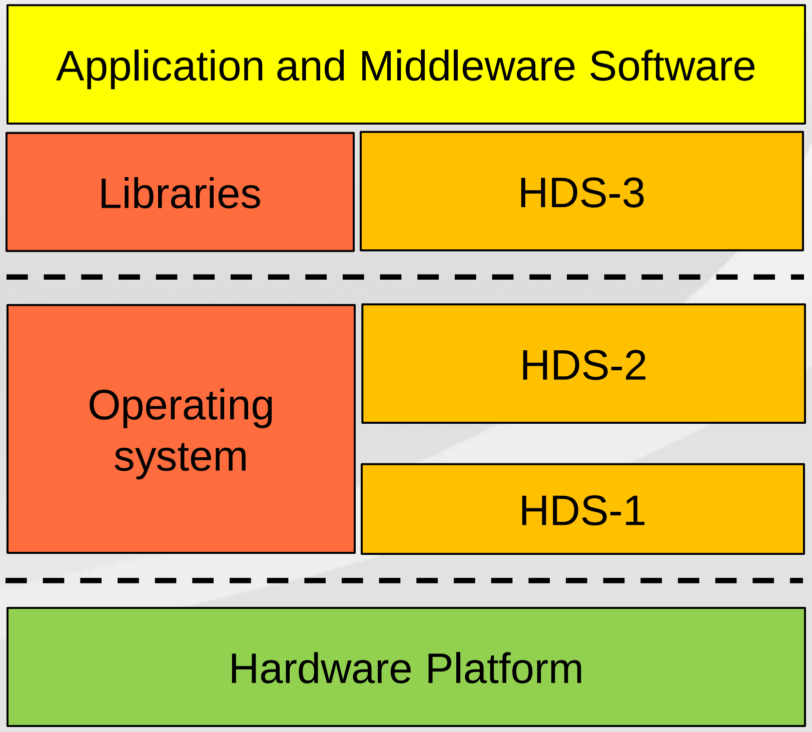
- Extensions for describing power related information



- Power data may depend on DfT, IO, and other elements described within IP-XACT flows

Hardware Dependent Software Ext.

- **Extensions for HDS integration**
  - HDS-1 Hardware Access Layer
    - Provides access methods to HW IPs
    - Abstracts from CPU I/O interface
  - HDS-2 Driver Layer
    - Provides 'classical' device driver
    - Implements control and SW functions
    - Integrates with OS
  - HDS-3 Feature Abstraction Layer
    - Provides abstract features
    - Can aggregate multiple HDS-2
- **Extensions for driver generation**

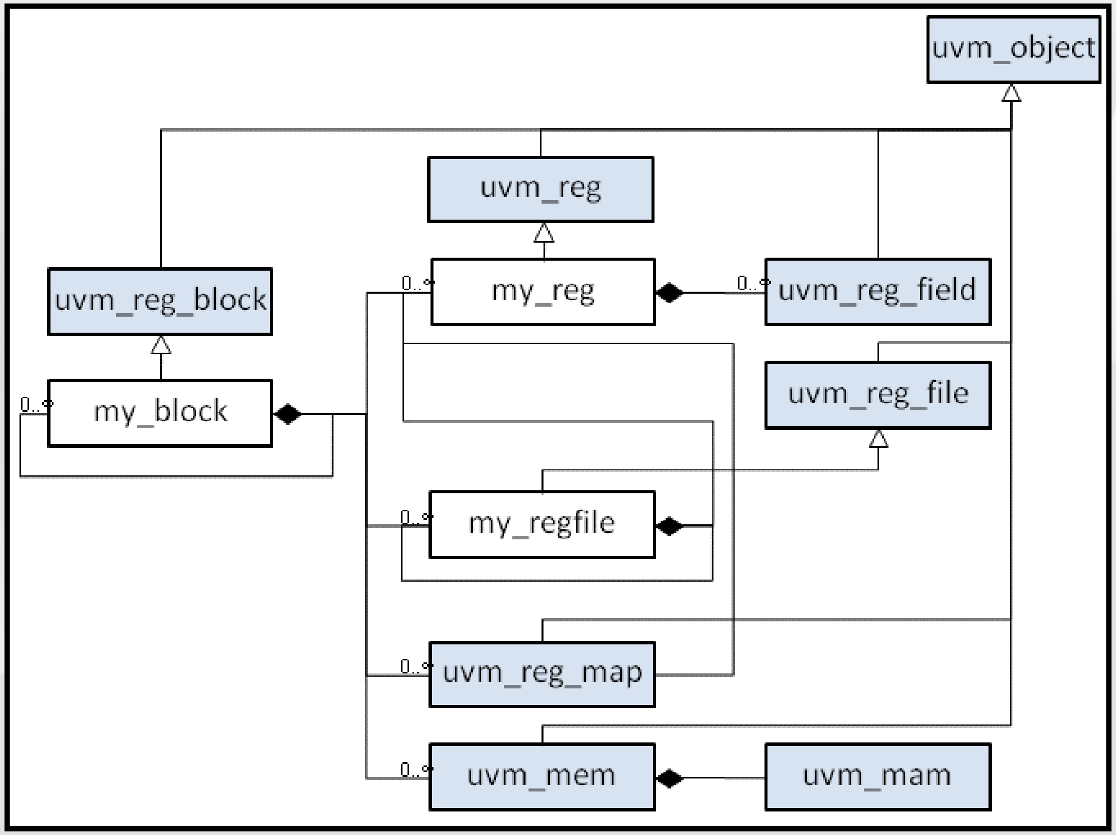


Universal Verification Methodology Ext.

Extensions for generating UVM register models from IP-XACT register descriptions, including

- HDL paths to support backdoor access
- Randomization constraints
- Coverage specifications

UVM register model



Summary

- **The IP-XACT standard supports user-defined vendor extensions**
  - Typically used by companies to implement specific tool or flow features
- **Accellera targets “standard extensions” to enable cross-company IP-XACT usage in new areas such as**
  - Analog-Mixed Signal
  - Physical Design Planning
  - Power
  - Hardware Dependent Software
  - Universal Verification Methodology
- **Early release will be available at DAC 2012**

Acknowledgement

Thanks to all people who contributed to the material presented here

- Grégoire Avot
- Martin Barnasconi
- Mukesh Chopra
- Ruud Derwig
- Olivier Florent
- Karthick Gururaj
- Zoltan Sugar
- Emmanuel Vaumorin

And thanks to all people, not mentioned here, who are contributing to the discussions in the Accellera IP-XACT EWG.