Introduction to IP-XACT and Kactus2

26 January 2022

SoC HUB

SoC design challenges

HDL level

- RTL code has three aspects mixed
  - Behavioral description
  - Structural description
  - Control for configuration
- Implicit references that get evaluated late in the design flow
- Works fine for small design, but vulnerable to errors in large projects
  - 100k files, multiple vendors, …
  - Wrong path/files, conflicts in naming, custom scripts dependent on file version, …
- Coding style agreements does not seem to help

SoC level

- The scale is so large that nobody can comprehend the whole system in detail
- Abstractions above RTL must be used for design space exploration
- Multitude of tools, languages and specification styles
- Design for deadline often compromises design for reuse
- Integration of IPs from different vendors is difficult without any agreed rules for interoperability

SoC HUB

IP-XACT™

IEEE-1685 Standard

Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

SoC HUB

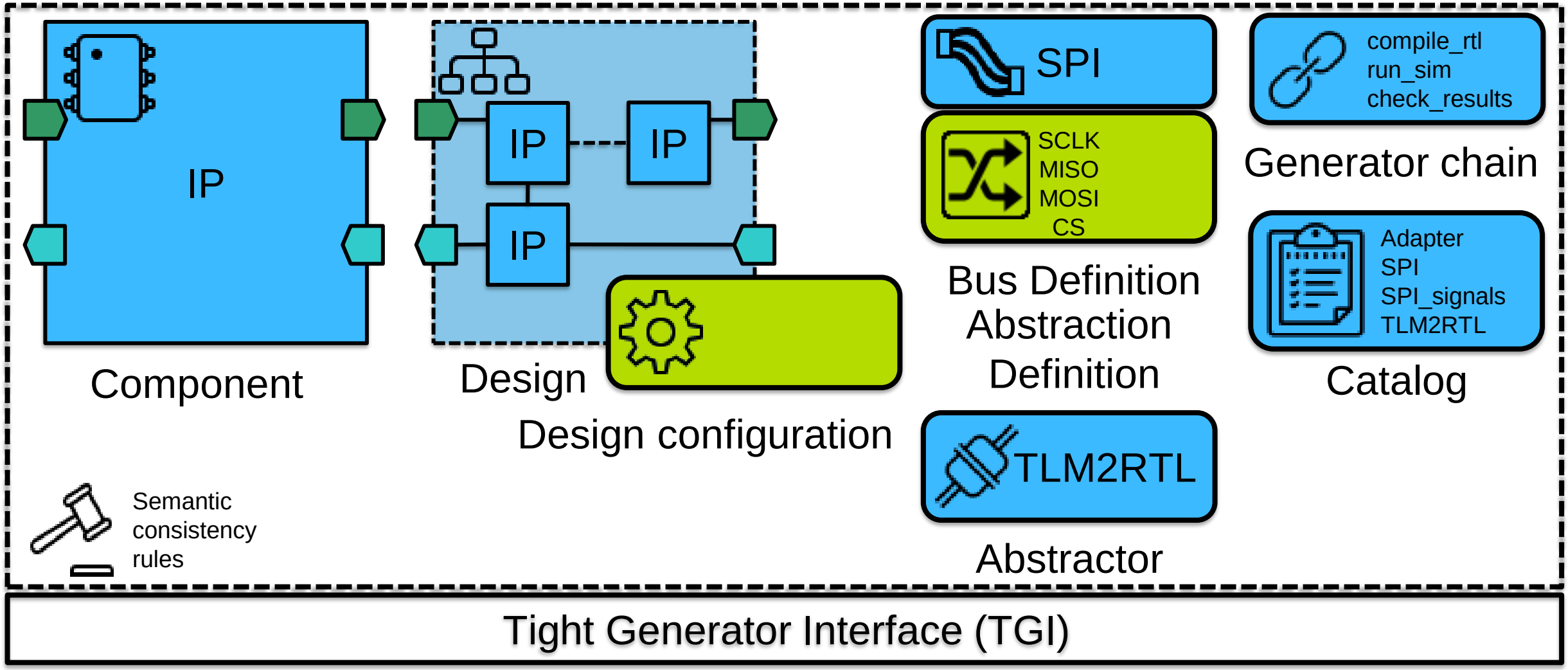
IP-XACT

- Electrical data sheet for Intellectual Property (IP)
- Industry standard for design data exchange
  - XML format
  - 790 unique elements, 241 attributes
  - Vendor extensions
  - Generator interface for tool flow
- First maintained by SPIRIT consortium
- Now IEEE standard 1685, maintained by Accellera Initiative
  - Members: AMD, Arm, Cadence, NXP, Nvidia, …

```
<ipxact:vendor>tuni.fi</ipxact:vendor>
<ipxact:library>cpu.structure</ipxact:library>
<ipxact:name>cpu_example</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:busInterfaces>
    <ipxact:busInterface>
        <ipxact:name>spi_master</ipxact:name>
        <ipxact:busType vendor="tuni.fi" library="interface" name="spi" version="1.0"/>
```

SoC HUB

Overview



SoC HUB

Document identification

- All top-level documents have a unique identifier, VLNV
- Example: tuni.fi:peripheral.components:uart:1.0

  Vendor | Library | Name | Version
- Cross-references by VLNV
  - No broken file paths

Vendor extensions

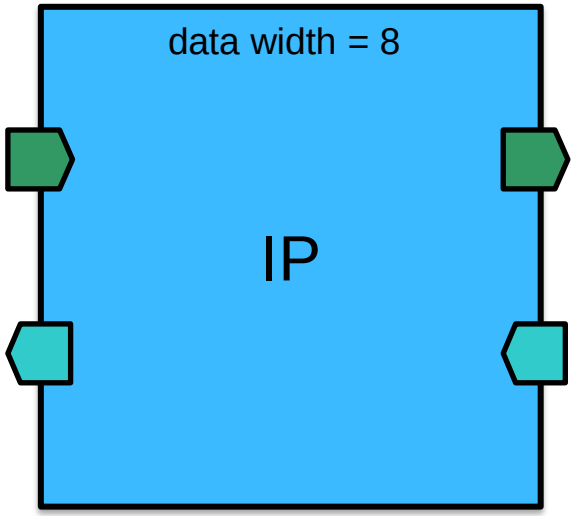
- Standard means for customization
- Any content
- Most, but not all, elements extendable
- Not compilant with other tools 
  - Potential danger for vendor lock-in
  - e.g. Xilinx adds ~200 elements

IP-XACT core elements

SoC HUB

Component

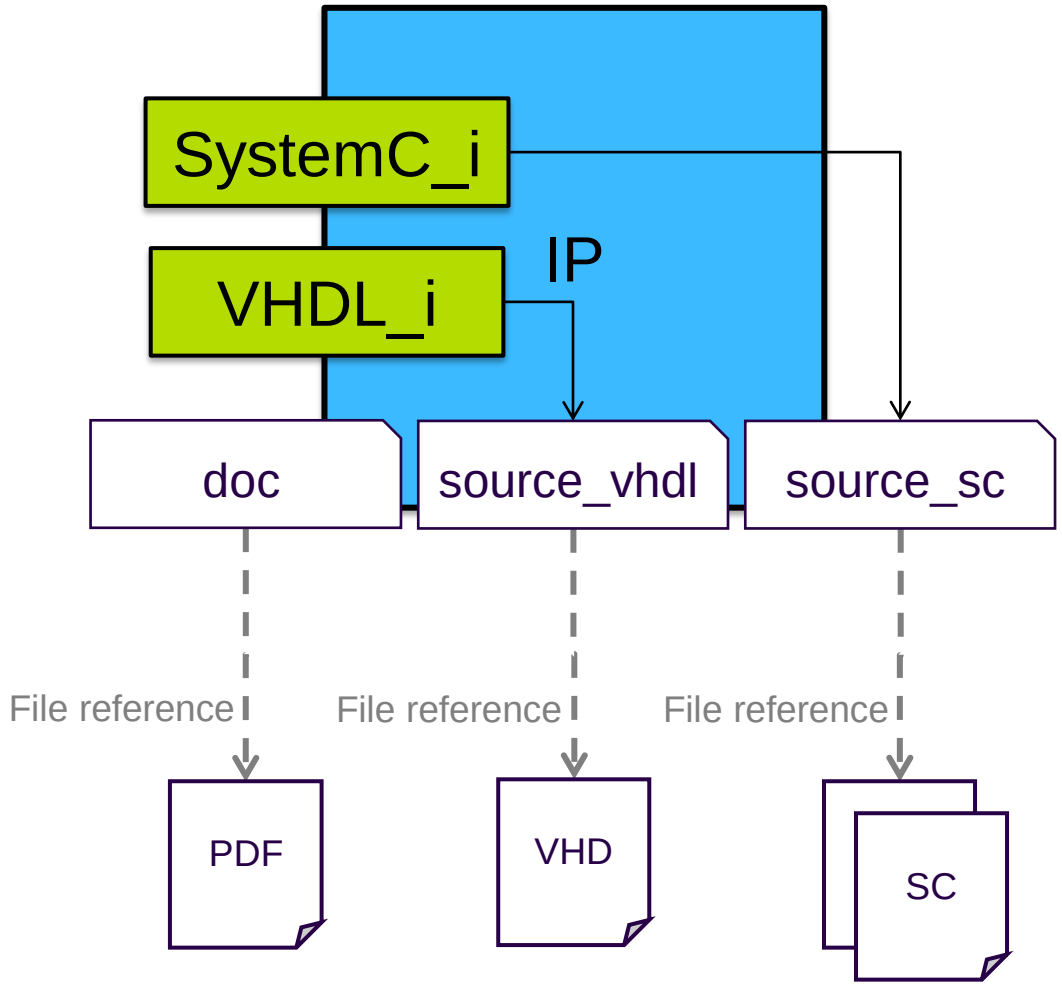
- Static structural model of a single IP
  - processor, uart, crossbar…
- Equivalent to VHDL entity or Verilog module
- Reusable: configurable by **parameters**
- May include multiple implementations
  - Same external interface



SoC HUB

Component implementations

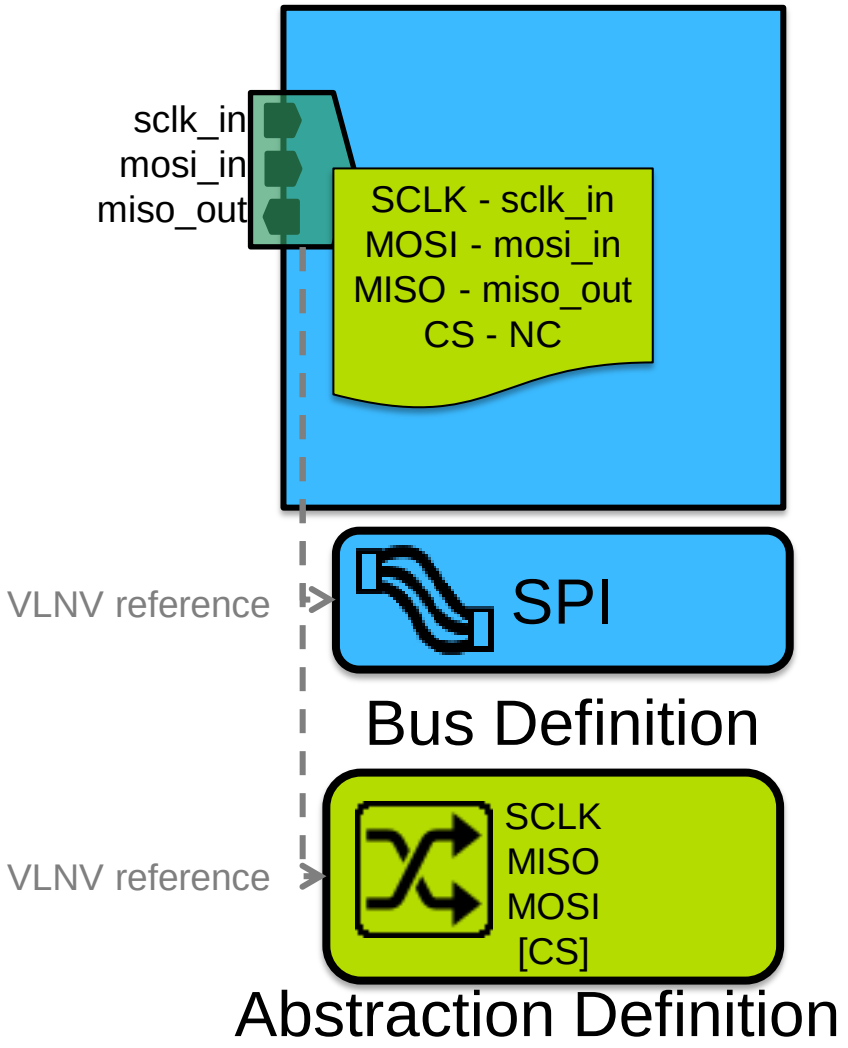
- Relevant **files** are listed in **FileSets**
- **Component instantiations** define implementation specific details
  - Filesets
  - Language
  - Module name
  - Module parameters (language specific)
  - Library/package



SoC HUB

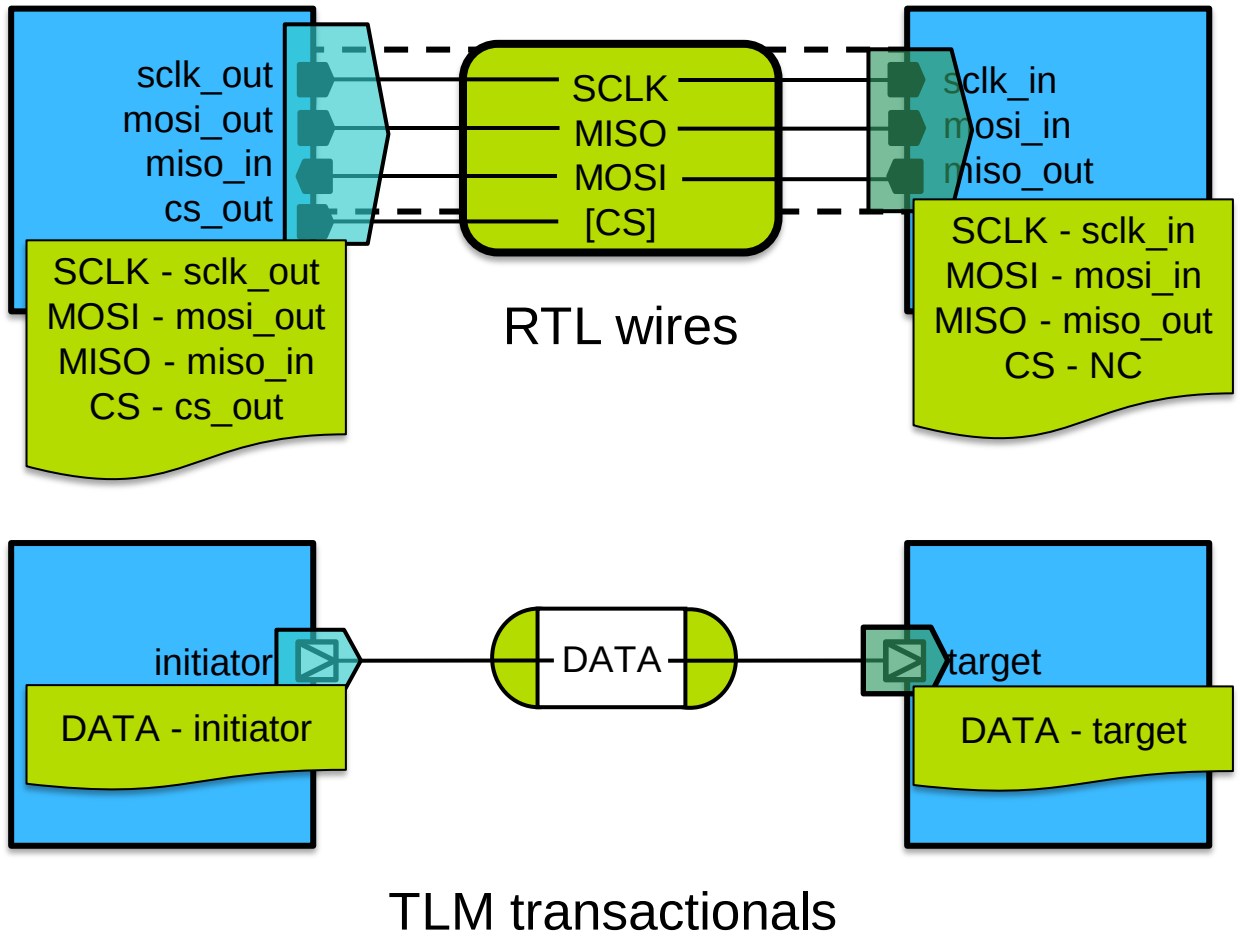
Component interface

- Physical **ports** equivalent to HDL ports
- **Bus interface** groups ports
  - **Bus definition** specifies the protocol
  - **Abstraction definition** defines the logical **signals** on the bus
  - **Port maps** link physical ports to logical signals
  - 7 modes per component role:
    - Master, Slave, System
    - MirroredMaster, MirroredSlave, MirroredSystem
    - Monitor



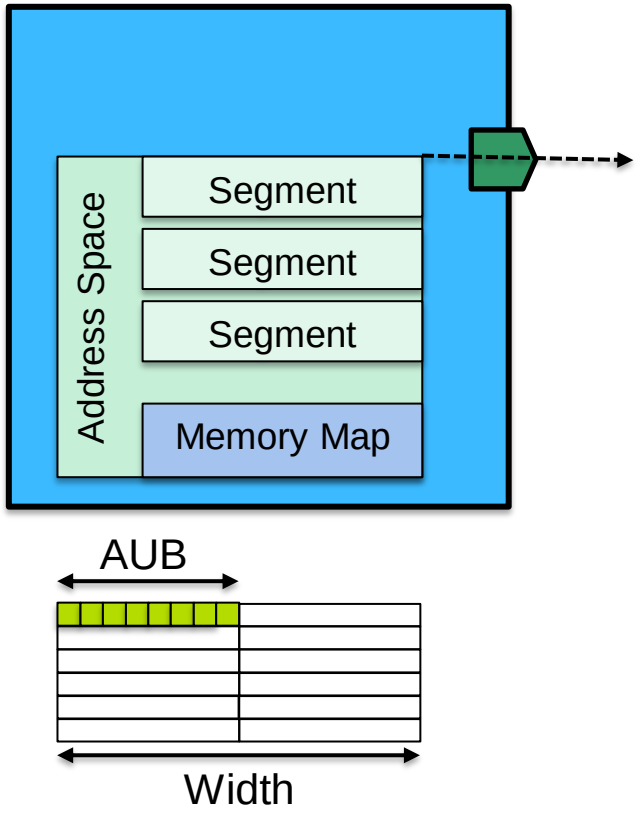
Interconnections

- **Interconnection** are made between component bus interfaces in a **Design**
  - Connects all ports mapped in the end bus interfaces
  - Avoids ad-hoc connections between individual ports
- Similar for both RTL and TLM models
  - RTL signal called **wire**
  - TLM signal called **transactional**
  - RTL-TLM cross-over with **abstractors**



Component address spaces

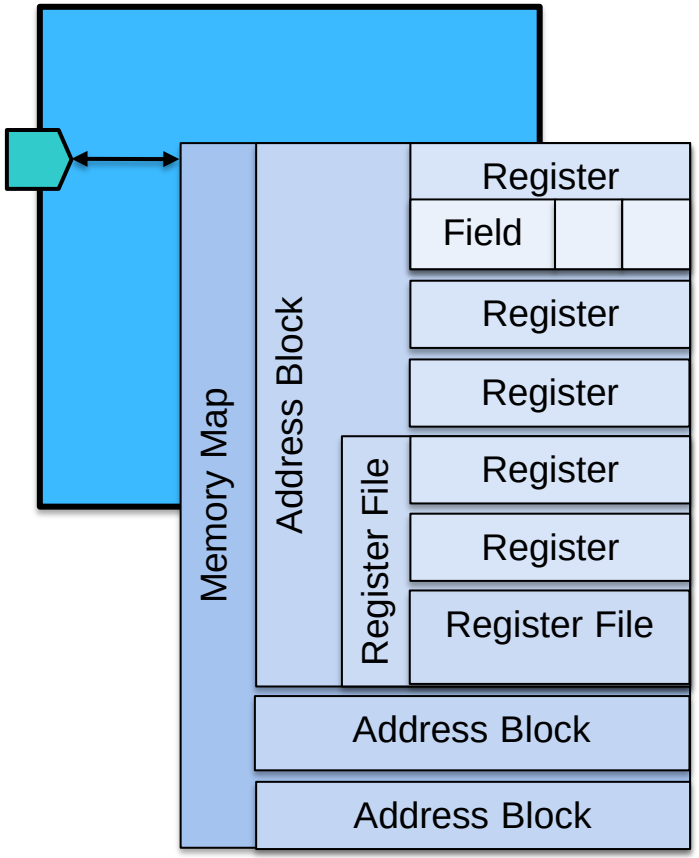
- **Address space** defines the address range that is available out of a master bus interface
  - **Address unit bits (AUB)** defines the number of data bits in each address increment (8 bits by default)
  - **Range** specifies the memory space as the **number of AUBs**
  - **Width** specifies the maximum bit size of a single transfer
- **Segments** describe sections of the address space
- **Local memory map** describes memory visible only in the containing address space



SoC HUB

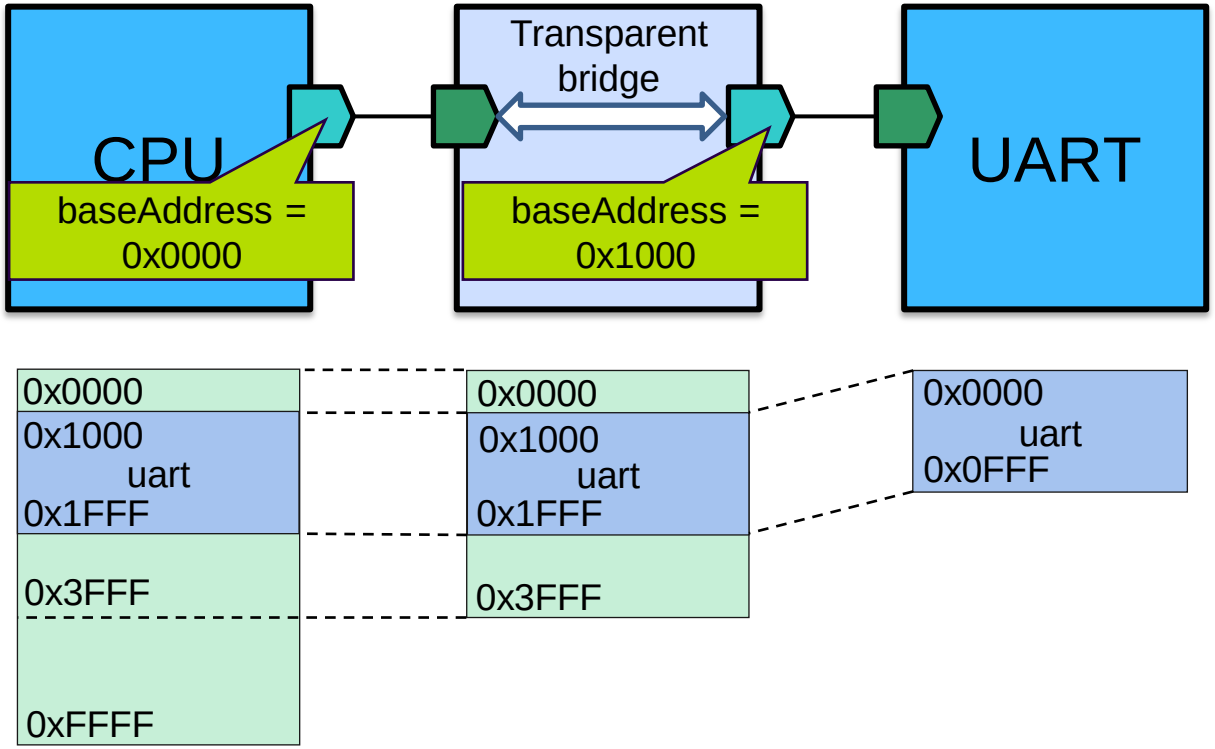
Component registers

- IP-XACT memory model is hierarchical:
  - **Memory map** is accessible through a slave bus interface
  - **Address blocks** are continous blocks of memory, reserved areas or registers
  - **Registers** define the software interface
  - **Fields** describe register bits
  - **Enumerated values** describe field bit pattern intent
- **Register files** group registers and other register files
- Address locations are defined as local offsets



Bridges

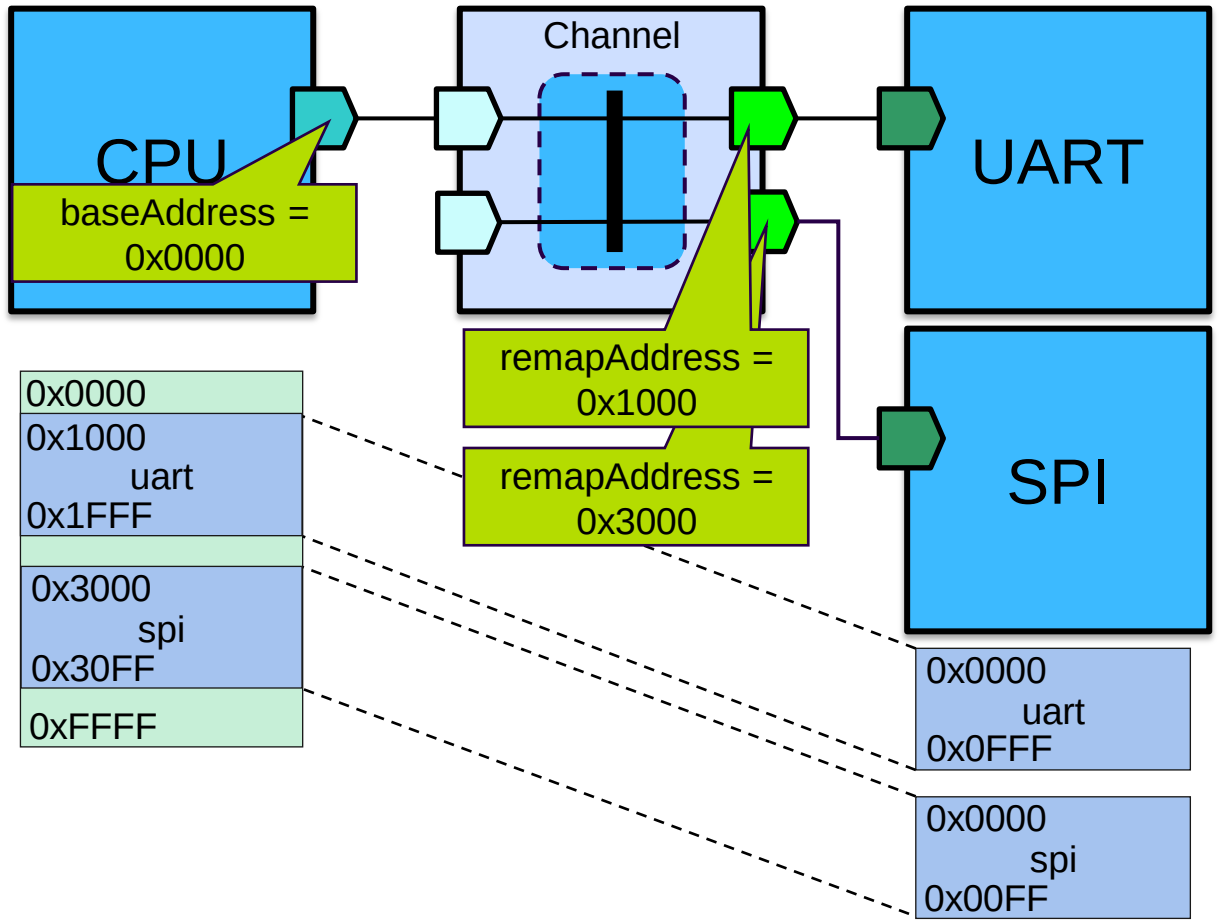
- **Transparent bridge** provides direct access through a component
  - Connects a slave interface to master(s)
  - Component may introduce protocol changes
  - Addressing remains unchanged
- **Opaque bridge** hides the connected IP memory map



SoC HUB

Channels

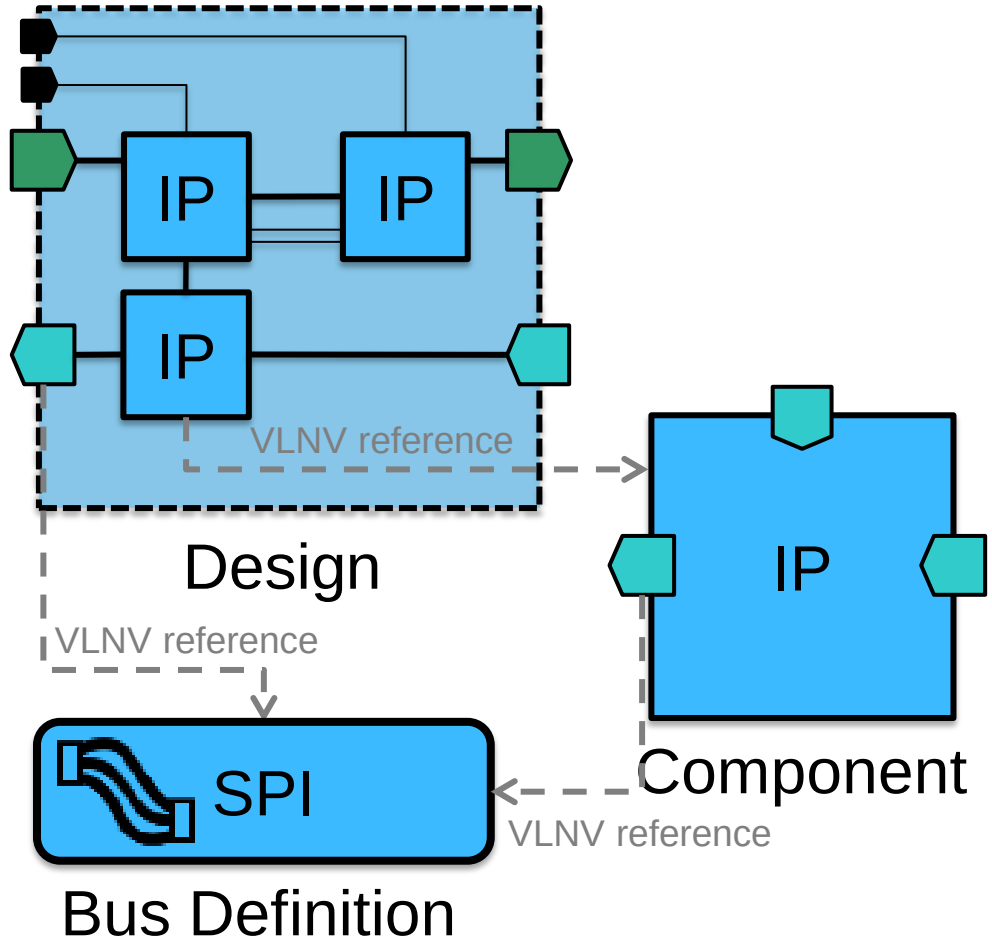
- **Channel** connects multiple mirrored bus interfaces into a single bus
  - Slave addresses are consistent for all masters
  - Only one master may initiate a transfer simultaneously



SoC HUB

Design

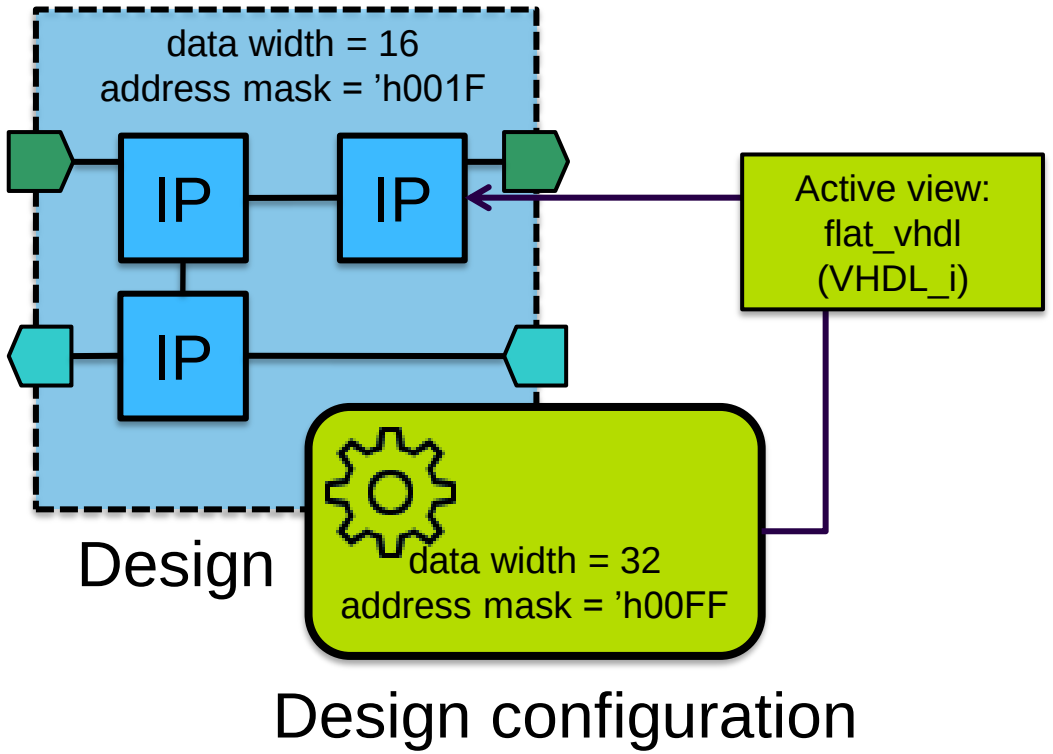
- A model of the system structure
- **Component instances** define sub-components
- **Interconnections** for communication
  - Formalized by Bus definitions
- **Ad-hoc connections** between ports



SoC HUB

Design configurations

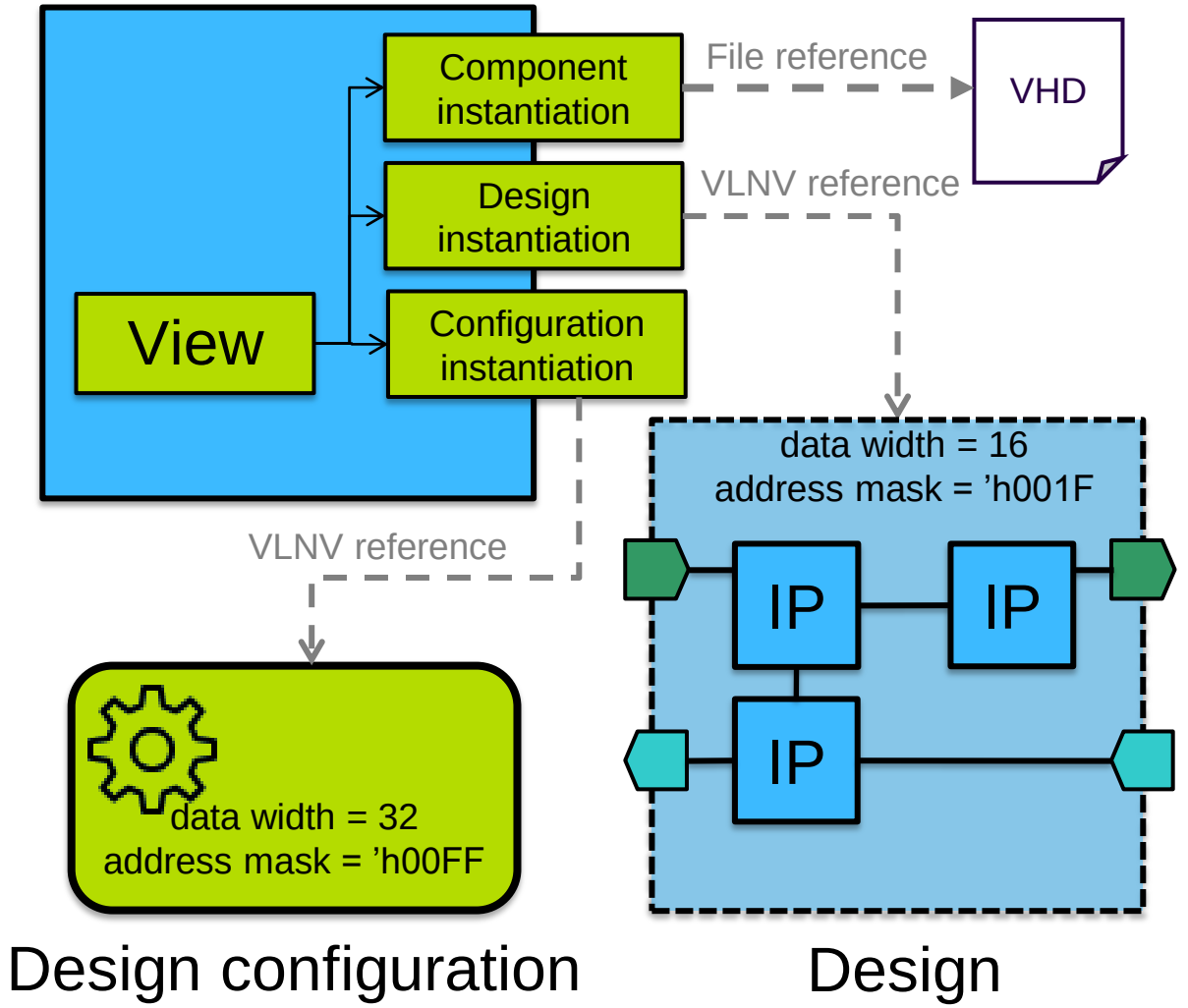
- A design is configured by
  - Parameter values
  - Component instance **active views**
- Active view selects the instance implementation
- Cannot add/remove instances or connections



SoC HUB

Component views

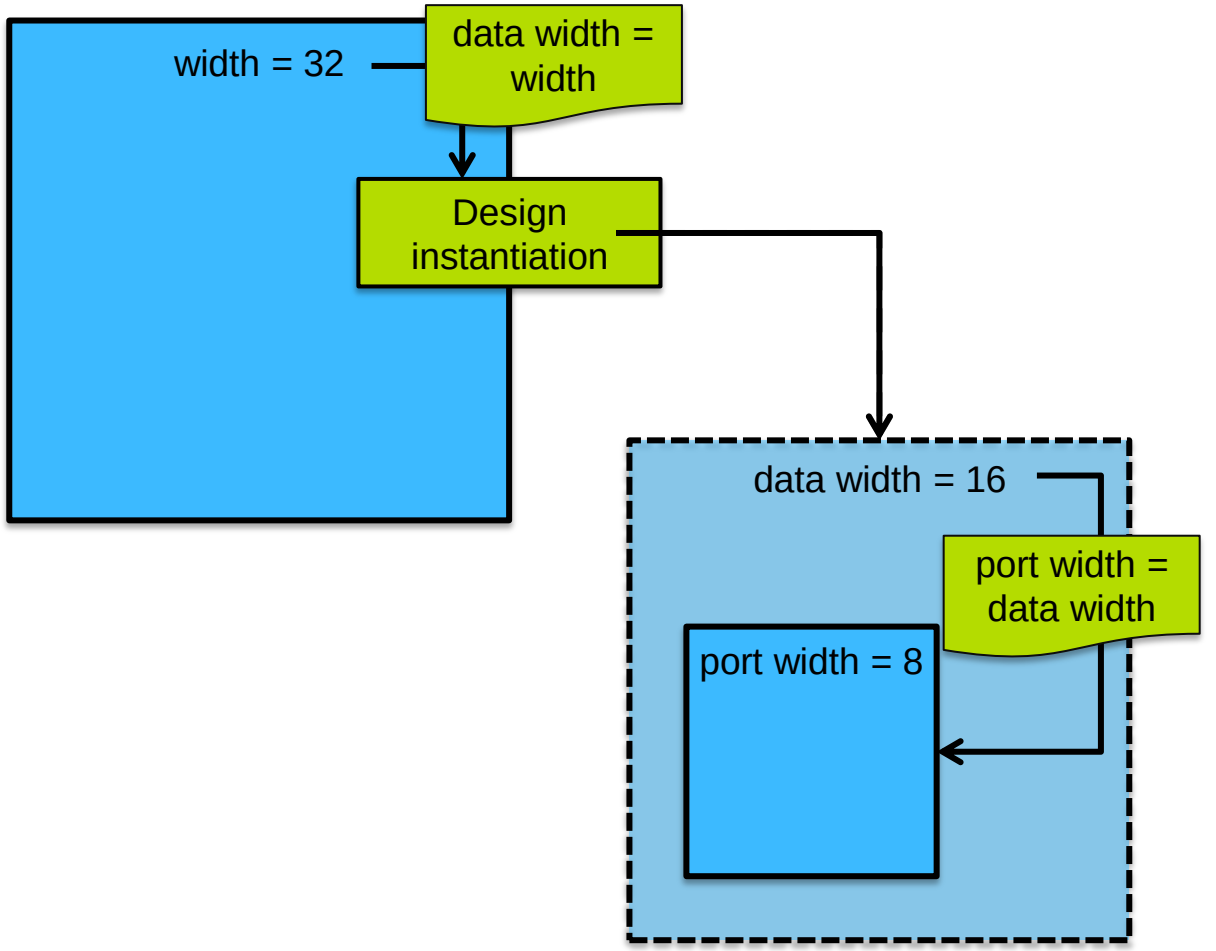
- A **view** defines a representation of the component
  - RTL, TLM, simulation, synthesis…
- Refer any combination of instantiations:
  - **Component instantiation** for selecting the implementation
  - **Design instantiation** for identifying the hierarchical design
  - **Design configuration instantiation** for selecting the configuration for the design



SoC HUB

Parameter propagation

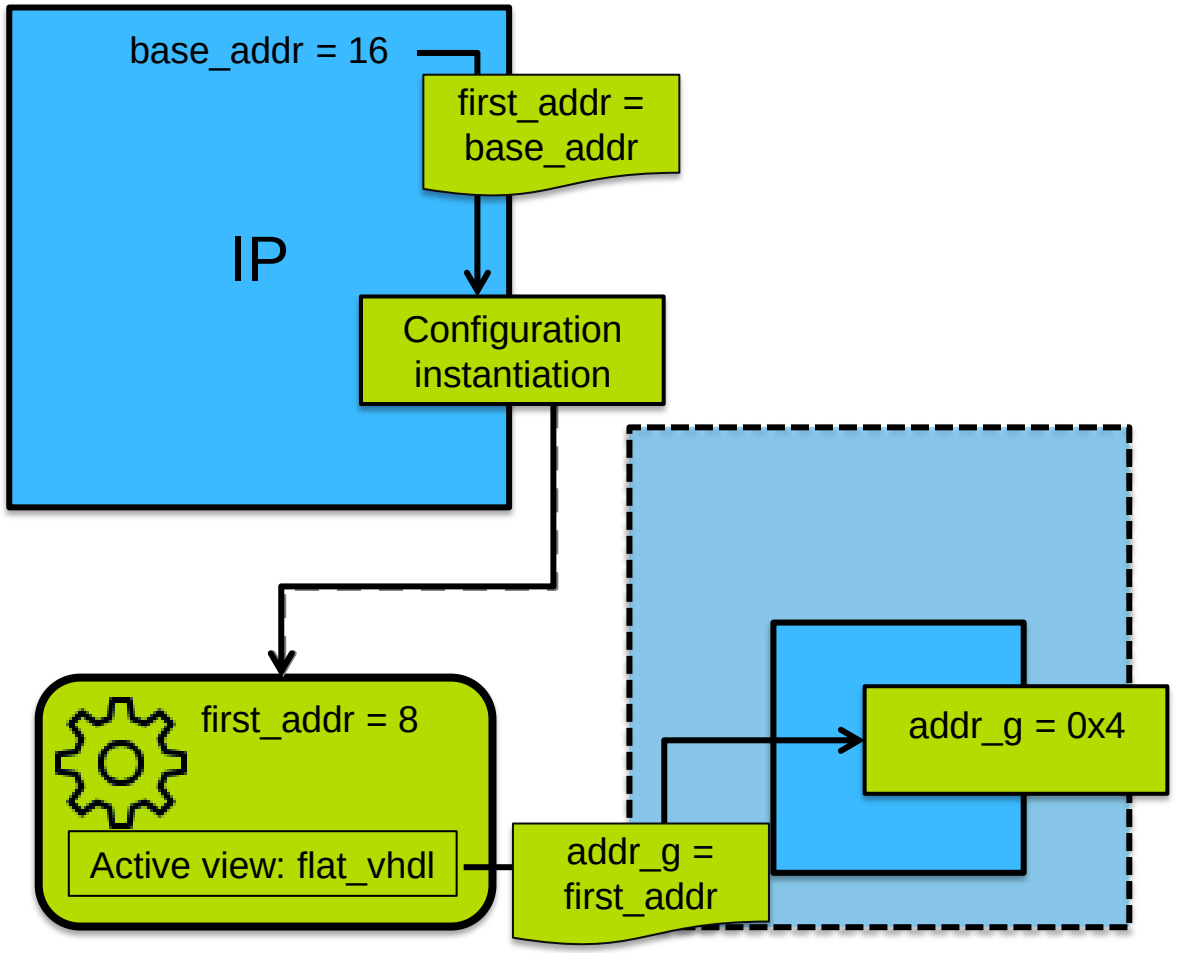
- Design may override parameter values in component instances
  - **Configurable element values**
  - Unless set to resolve **immediate** value in the component (default)
- Design instantiation may override design parameters
  - May reference containing component parameters



SoC HUB

Module parameter propagation

- Design configuration may override **module parameters** in component instantiation
- Design configuration instantiation may overried design configuration parameters
  - May reference component parameters
- Propagation rules:
  - Value propagates down one level in hierarchy
  - May only reference parameters in the same document



SoC HUB

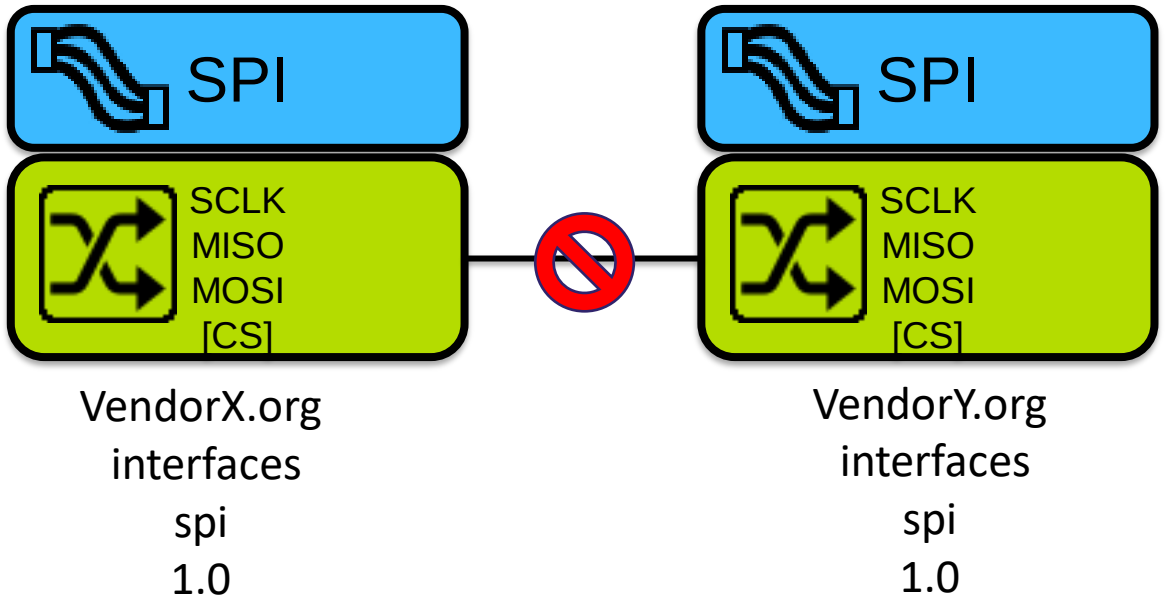
IP-XACT features

- Companies seem to be most interested in:
  - File management
    - Help organize IP related files
  - Connectivity
    - Help plug-and-play IP-blocks
    - Benefits the HW team
  - Address definitions
    - Help manage tables of registers and memory maps
    - Benefits the SW team
- The standard is versatile in some respects, limited in others
  - Complementing standards: SystemRDL and Unified Power Format (UPF)

SoC HUB

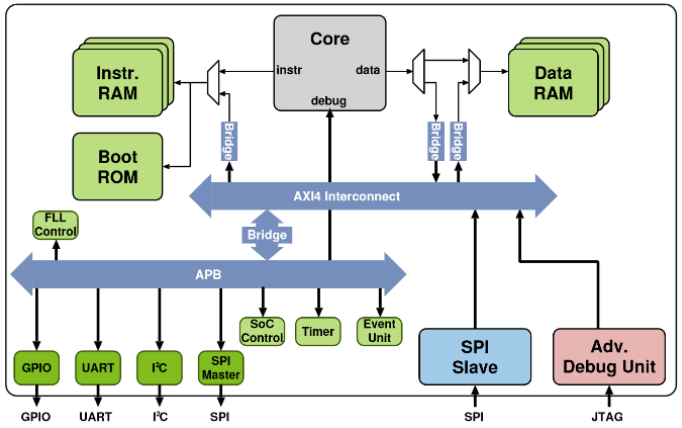
Practical challenges

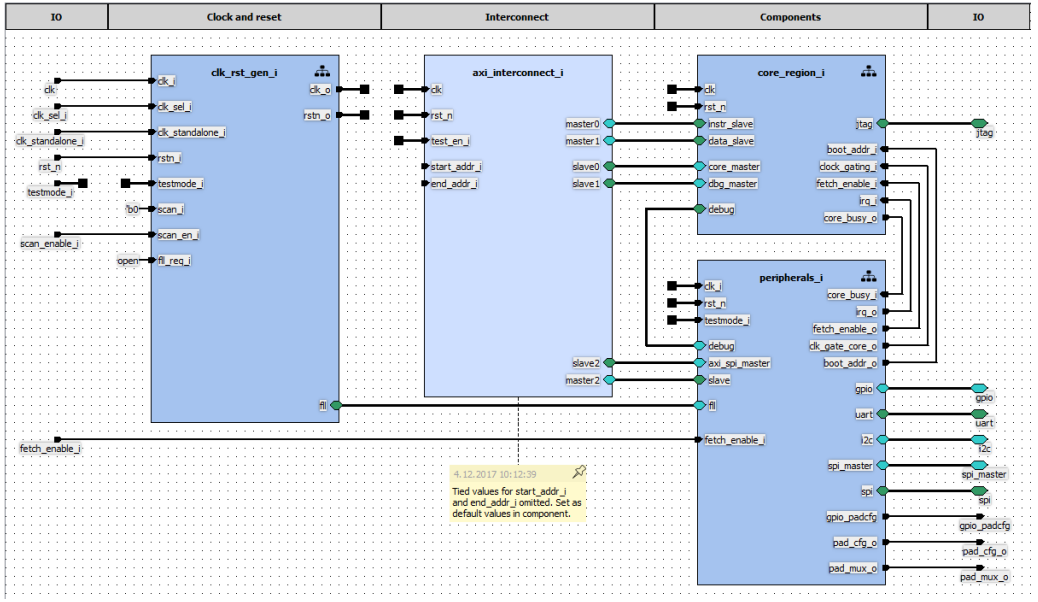
- No compatibility of bus/abstraction definitions from different vendors
  - Very few are publicly available
- Trying to force IP-XACT to conform to
  - Overly generic HDL
  - HDL with structure, behavior and configuration mixed
  - (File)name dependencies, virtual libraries
- Vendor extensions complicate data exchange



Lessons learned from packaging RISC-V

- Open-source RISC-V microprocessor project, PULPino [1], approximately 250 SystemVerilog files across 21 repositories
- The project was packaged with Kactus2 resulting in 95k lines in 169 files [2]
- Enforce good HDL practices early
  - Decouple structure and behavior
  - Design module interfaces carefully
- Reference bus definitions are mandatory for true compatibility of IPs





[1] A. Traber and M. Gautschi, "Pulpino: Datasheet," ETH Zurich, Tech. Rep., 2016. [Online]. Available: https://github.com/pulp-platform/pulpino/blob/master/doc/datasheet/datasheet.pdf

[2] E. Pekkarinen and T. D. Hämäläinen, "Modeling RISC-V Processor in IP-XACT," 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 140-147.

SoC HUB

HDL challenge: structure and behavior

- Glue logic for simple data manipulation added as part of structural description
  - Typically multiplexing
- All behavior must be contained within components
  - Move glue logic to new components
  - Impractical for basic ANDs etc.

```
logic is_boot, is_boot_q;
...
boot_rom_wrap
  #(
    .DATA_WIDTH ( DATA_WIDTH )
  )
  boot_rom_wrap_i
  (
    .clk     ( clk                          ),
    .rst_n   ( rst_n                        ),
    .en_i    ( en_i & is_boot               ),
    .addr_i  ( addr_i[`ROM_ADDR_WIDTH-1:0] ),
    .rdata_o ( rdata_boot                   )
  );

assign rdata_o = (is_boot_q == 1'b1) ? rdata_boot : rdata_ram;

always_ff @(posedge clk, negedge rst_n)
  begin
    if (rst_n == 1'b0)
      is_boot_q <= 1'b0;
    else
      is_boot_q <= is_boot;
  end
```

Snippet from instr_ram_wrap.sv

SoC HUB

HDL challenge: conditional structure

- Configuration value enables/disables part of structure
  - Module instantiation
  - Wires
  - Ports
- Very flexible in (System)Verilog
- Inherently IP-XACT structure is static
  - **isPresent** attribute is sufficient where applicable
  - With instances consider creating an alternate design

```
`ifndef VERILATOR
  apb_uart apb_uart_i (
    ...
  );
`else
  apb_uart_sv
    #(
       .APB_ADDR_WIDTH( 3 )
    )
    apb_uart_i
    (
      ...
    );
`endif
```

Snippet from peripherals.sv.

SoC HUB

HDL challenge: generate loops

- Create regular structure
  - Module instantiations
  - Wire connections
- Replace with static instances

```
generate
   genvar i;
     for (i = 0; i < APB_NUM_SLAVES; i = i + 1) begin
      cluster_clock_gating core_clock_gate
      (
        .clk_o     ( clk_int[i]                    ),
        .en_i      ( peripheral_clock_gate_ctrl[i] ),
        .test_en_i ( testmode_i                    ),
        .clk_i     ( clk_i                         )
      );
    end
endgenerate
```

Snippet from peripherals.sv.

HDL challenge: configurable interfaces

- Reusable modules with configurable interface
  - N master and M slave interfaces
- Bus interfaces are statically defined. Options:
  - Define maximum set, leave any unused unconnected or use isPresent
  - Automatically generate a component with correct number of bus interfaces and ports from a template

```
`include "axi_bus.sv"

module axi_node_intf_wrap
  #(
    parameter NB_MASTER      = 4,
    parameter NB_SLAVE       = 4,
    parameter AXI_ADDR_WIDTH = 32,
    ...
    )
  (
    // Clock and Reset
    input logic clk,
    input logic rst_n,
    input logic test_en_i,

    AXI_BUS.Slave slave[NB_SLAVE-1:0],

    AXI_BUS.Master master[NB_MASTER-1:0],

    // Memory map
    input  logic [NB_MASTER-1:0][AXI_ADDR_WIDTH-1:0]  start_addr_i,
    input  logic [NB_MASTER-1:0][AXI_ADDR_WIDTH-1:0]  end_addr_i
  );
```

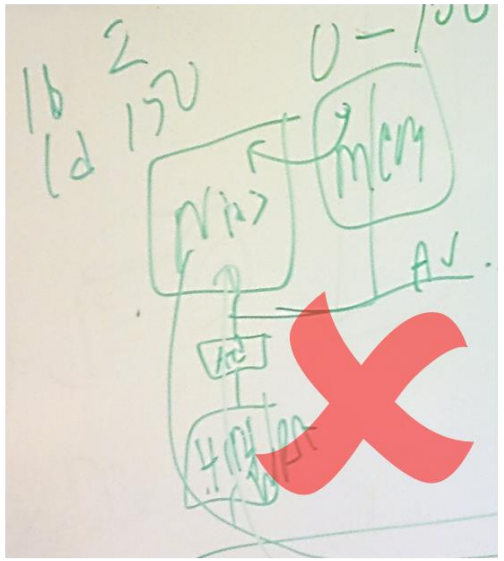
Snippet from axi_node_intf_wrap.sv

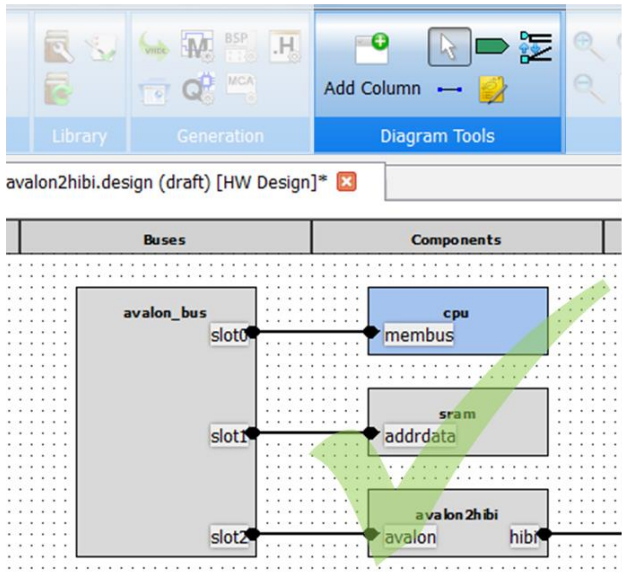
SoC HUB

IP-XACT Design Environment

Project motivation

- IP-XACT the most promising standard for common data exchange format
  - IP integrator companies
  - IP providers, SMEs, subcontractors
  - University teaching and research
- Disjoint effort in tool development
- Commercial IP-XACT tools are:
  - Expensive
  - Difficult to use
  - Disregarding the standard





SoC HUB

Kactus2 project

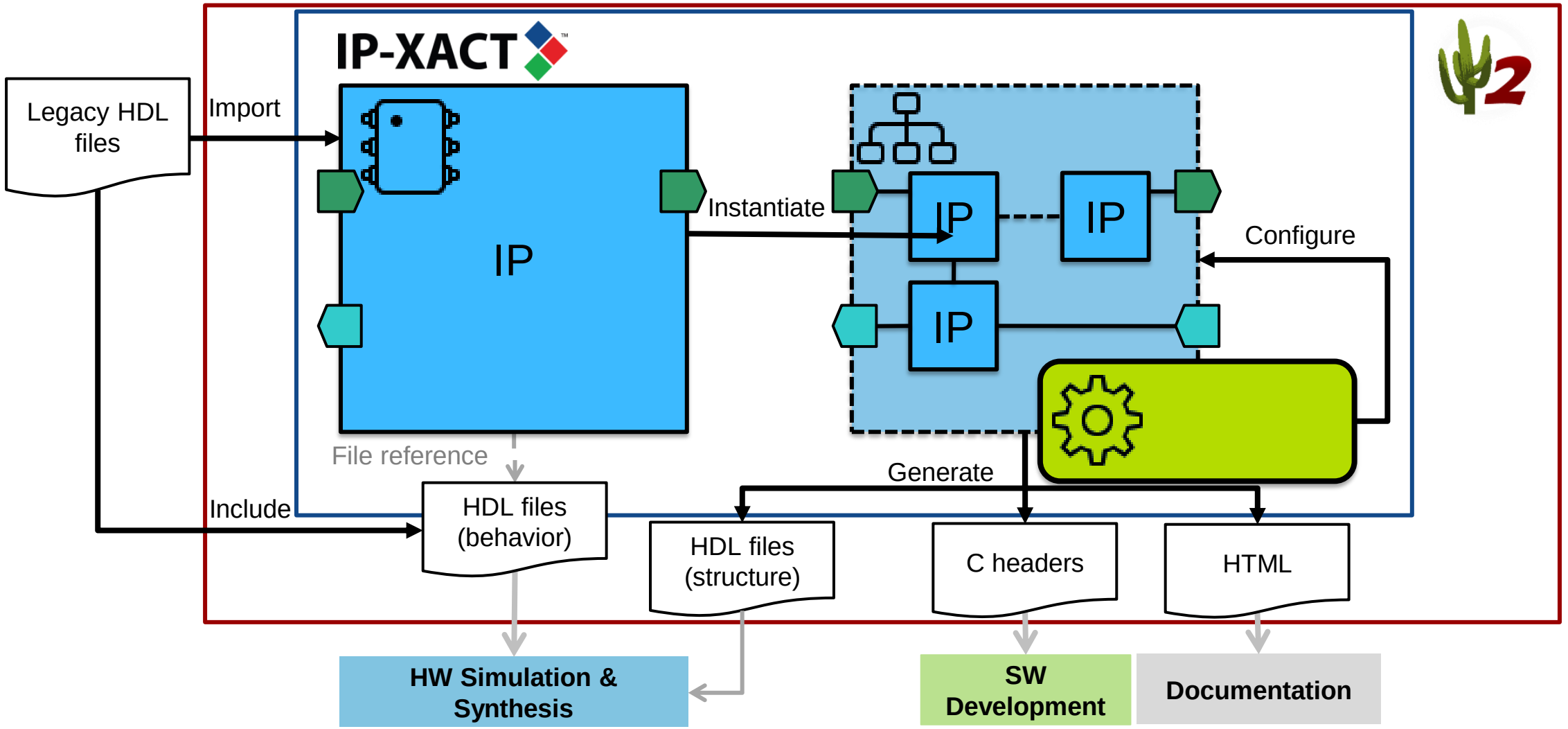
- Project started in TUT in 2009
  - Collaboration with 9 embedded system companies
  - Since then collaboration/support requested by 20+ companies
- Kactus2 released open-source in 2011
  - Initially a graphical editor for IP-XACT XML
  - Now over 500k lines of C++/Qt in total
- 19,896 downloads since release

GitHub

https://github.com/kactus2/kactus2dev

SoC HUB

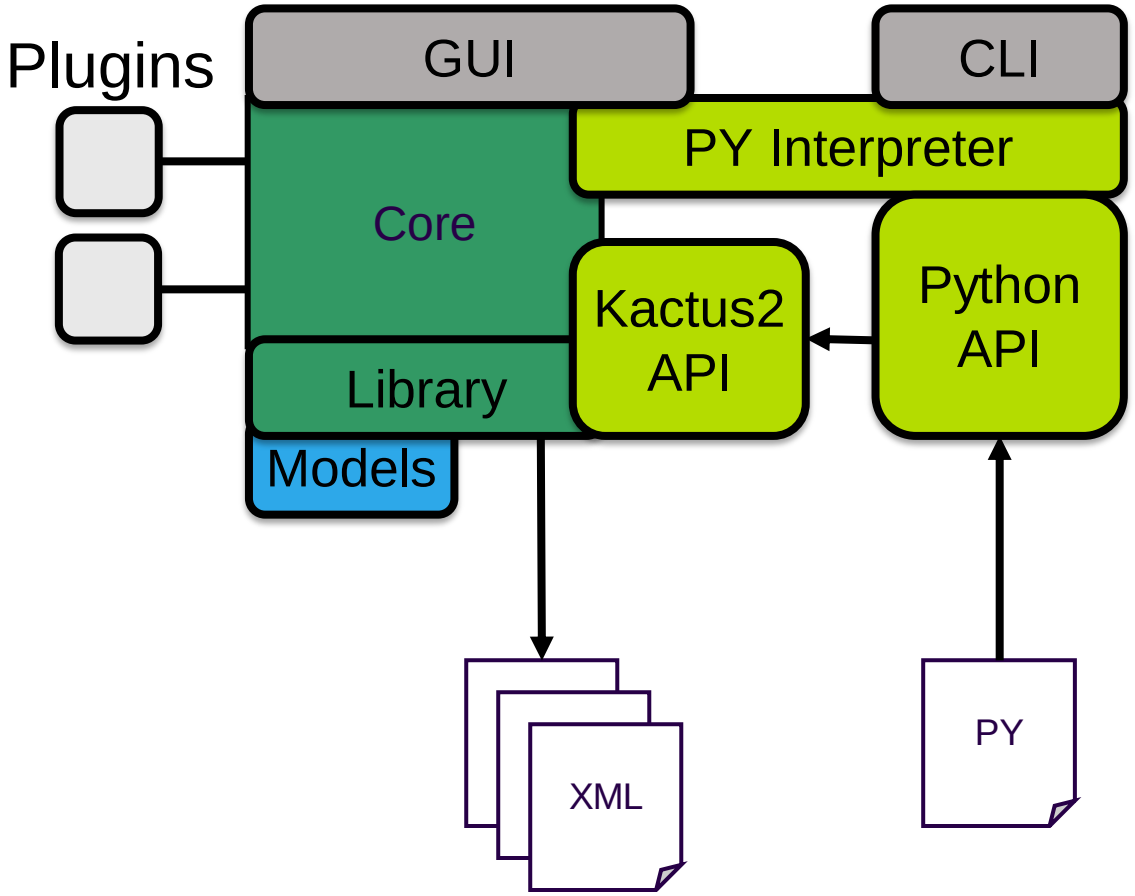
IP-XACT-based design flow



SoC HUB

Kactus2 structure

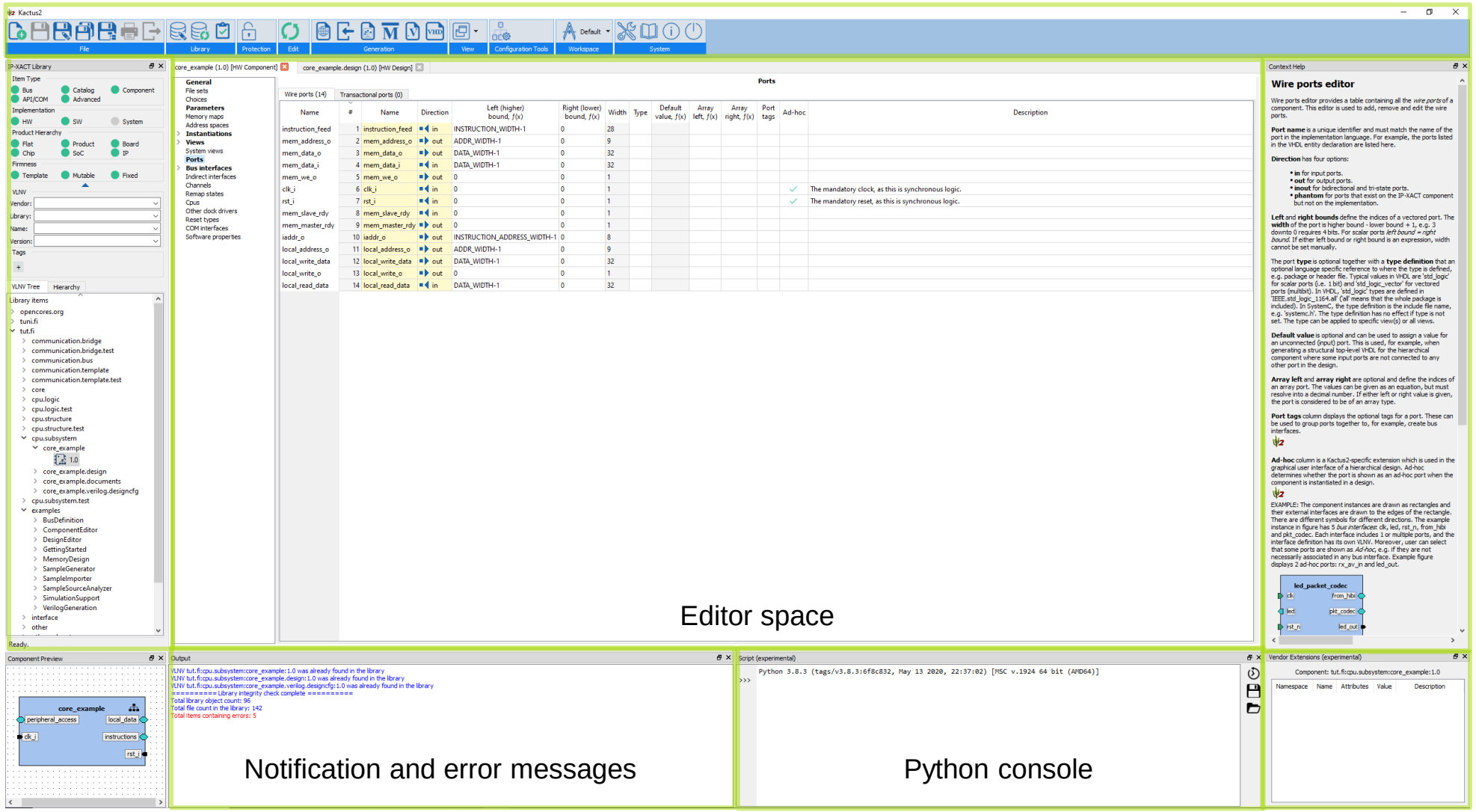
- All access to IP-XACT files is controlled by the library interface
  - Data is parsed into model objects
- User input with graphical user interface (GUI) or command-line interface (CLI)
  - Both have access to Python interpreter since version 3.9.0
- A set of available functions are defined in PythonAPI
  - Scripting for batch jobs
  - Access to core functions and library through Kactus2 API
- Extendable with plugins
  - Importers for fast RTL to IP-XACT conversion
  - Generators for code/documentation
  - Source analyzers for file dependency information



Kactus2 graphical user interface

Toolbar

IP-XACT document library

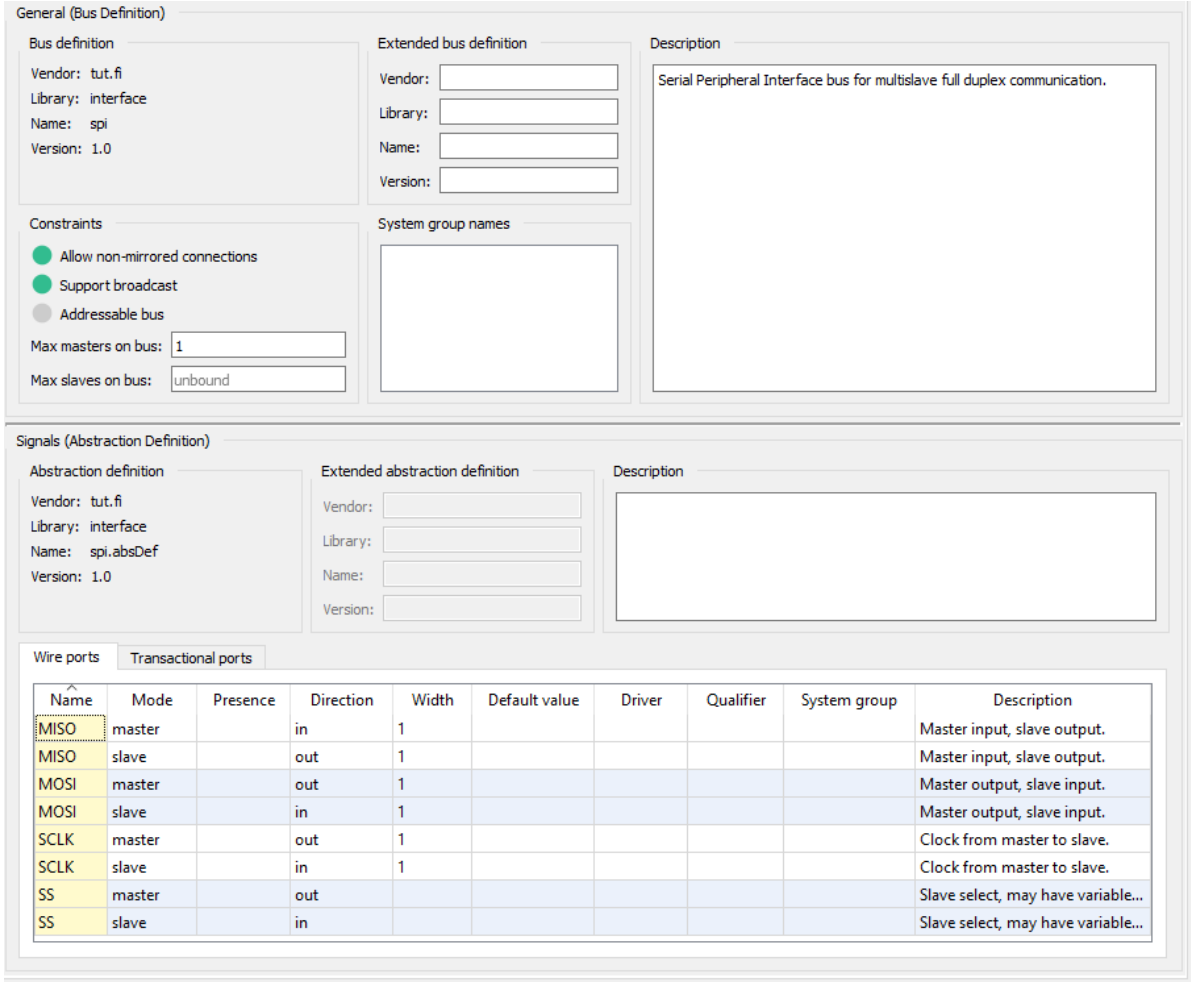


Context-sensitive help

Selection-sensitive editors

SoC HUB Bus definitions editor

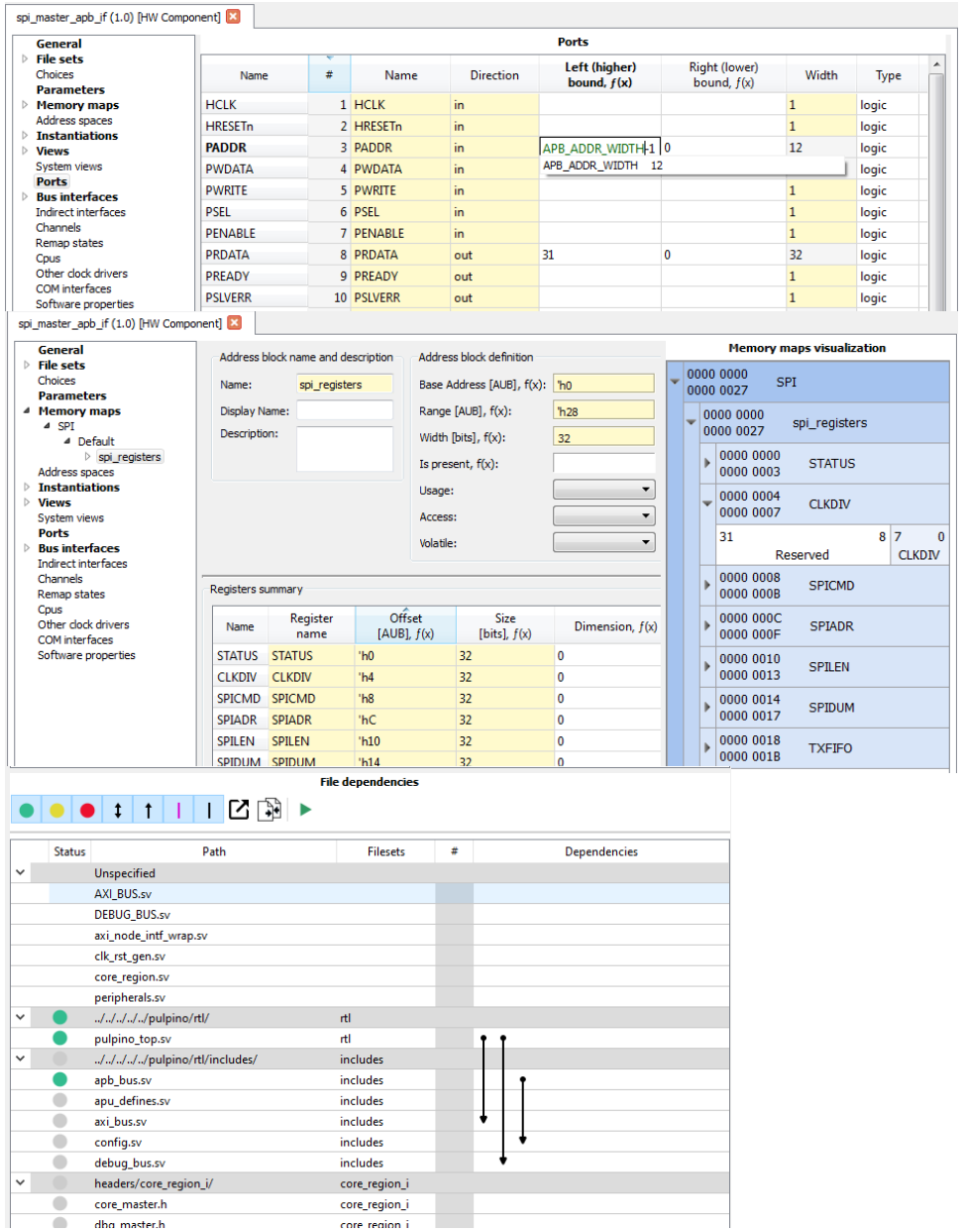
- Kactus2 groups together bus and abstraction definition
- Abstraction definition includes
  - Qualifiers (address, data, clock, reset, any)
  - Presence (required, optional, illegal)
- Signal conditions can be created
  - e.g. some signal is not allowed in master interface



SoC HUB

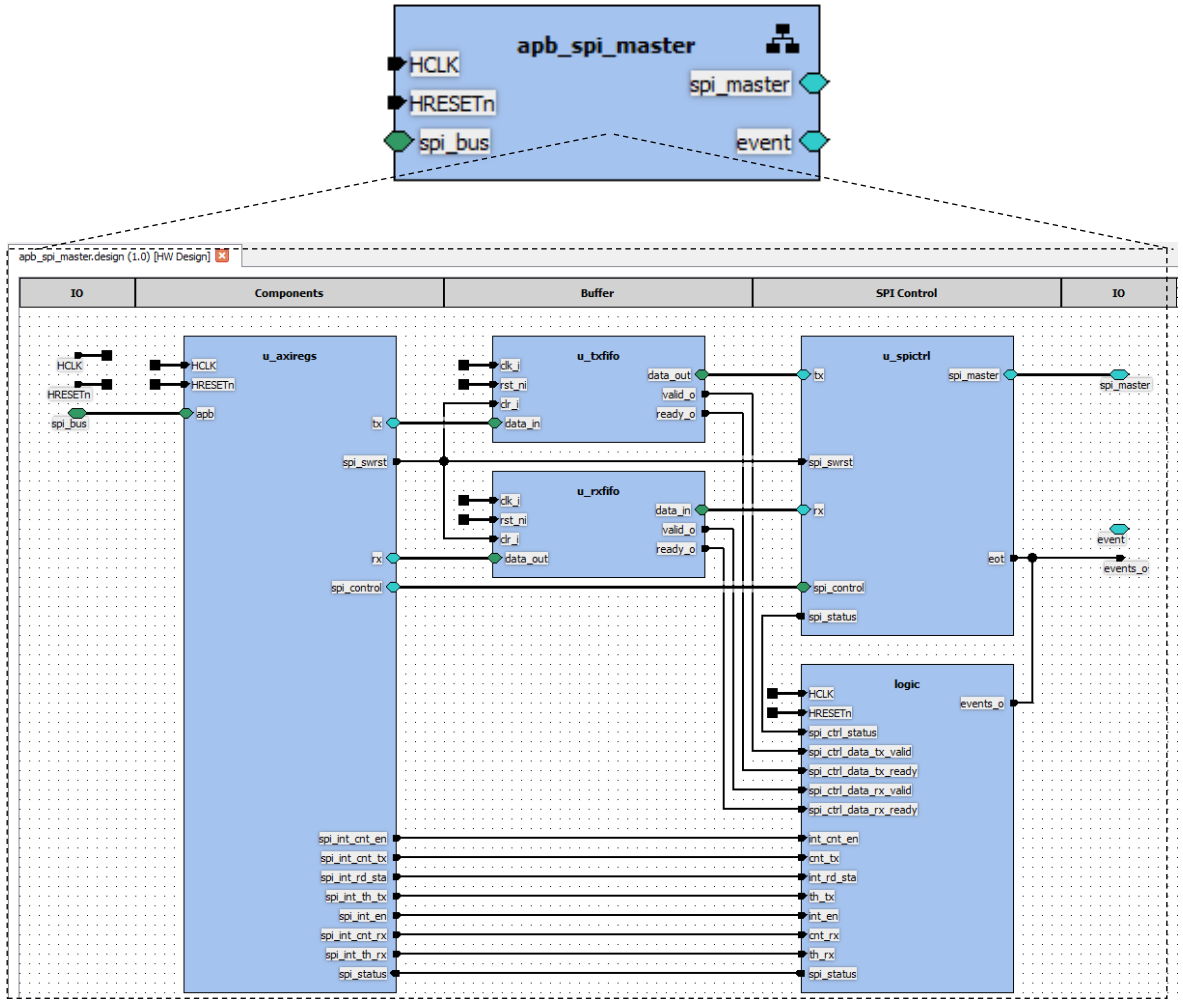
Component editor

- Define IP details e.g.
  - Ports
  - Parameters
  - Registers
  - Related files, file sets
- File set dependency analysis
- Memory map visualization
- Automatic validity checks



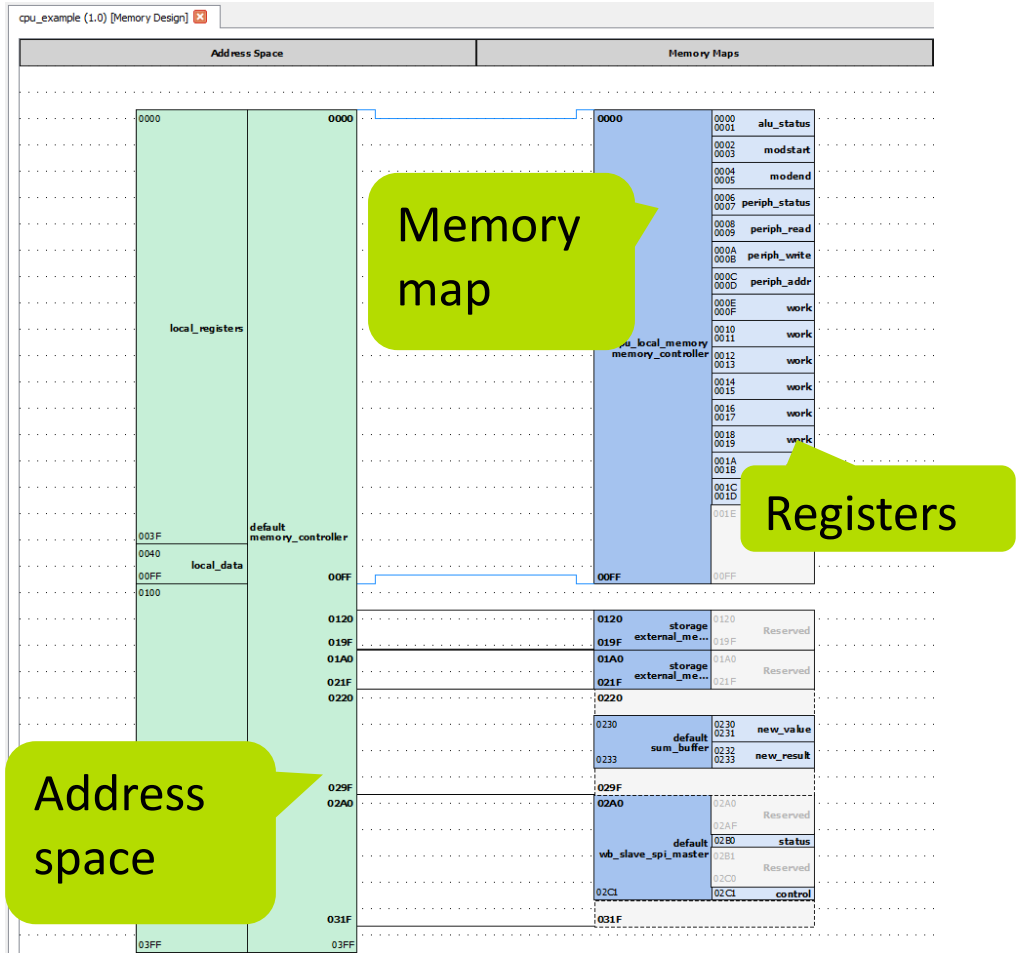
SoC HUB Design editor

- Create hierarchical structure using existing components
- Configure instances
- Speculate with draft instances
- Automatic checks for valid connections



SoC HUB Memory designer

- Align address spaces with memory maps
  - Multiple components
  - Multiple hierarchy levels
- Resolve register addressing based on connectivity
- HW designer can check address definitions in all components



[1] M. Teuho, E. Pekkarinen and T. Hämäläinen, "Visualization of Memory Map Information in Embedded System Design," 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 163-166 .

Summary

SoC HUB

Summary

- IP-XACT targets to ease reuse and data exchanged between IP vendors
- Lack of commonly shared bus definitions and examples slow down adoption
- Components and designs capture the SoC structure
- Kactus2 is the open-source IP-XACT tool by Tampere University

SoC HUB

Resources

- SoC Hub: www.sochub.fi
- Kactus2 home: https://research.tuni.fi/system-on-chip/tools/
- Kactus2 source code and issue tracking: https://github.com/kactus2/kactus2dev
- Kactus2 installer download: https://sourceforge.net/projects/kactus2/
- Kactus2 video tutorials: https://www.youtube.com/user/Kactus2Tutorial/