Virtual Prototypes and Platforms
A Primer

Eyck Jentzsch, MINRES Technologies GmbH
Rocco Jonack, MINRES Technologies GmbH
Josef Eckmüller, Intel Deutschland GmbH

MINRES®
TECHNOLOGIES

accellera
SYSTEMS INITIATIVE

2017
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

FOUNDATIONS OF VP

What are not virtual prototypes & platforms

- These are not virtualization solutions or virtual machines like VirtualBox or VMWare
  - A virtual machine (VM) is an emulation of a computer system (...) and provides functionality of a physical computer (Wikipedia)
  - A VM or virtual machine was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real computer machine."
- Although some concepts and mechanisms are also used in VPs

What is a virtual prototype or platform

- Virtual platform is a hardware simulator executing embedded software (eSW)
- Usually built on top of an instruction set simulator (ISS) of the processor(s) connected via buses to memory and peripherals such as timers, general I/O, communication interface, etc.
- While the VP is mostly implemented in SystemC the ISS may be implemented in some other general purpose language.

Differences between virtual prototype and platform

- Virtual prototype
  - Is in development and does not have final structure and partitioning.
  - Possibility to quickly change this
  - Comprehensive analysis means
  - Used to optimize SoC design wrt. power, area, speed, throughput, and/or latency
- Virtual platform
  - Represents a SoC in its final shape,
  - Aims at simulation speed as well as functional accuracy and completeness,
  - Used for (e)SW development

Alternatives to VPs

- Emulation
  - high accuracy and speed
  - needs Register-Transfer-Level (RTL) description, expensive to scale
- FPGA
  - high accuracy and speed
  - needs RTL implementation to do synthesis, place and route ➔ long implement-compile-debug loops
- Hardware in the loop (HIL)
  - very high accuracy and speed (real-time)
  - comes late in the design process, scalability challenge

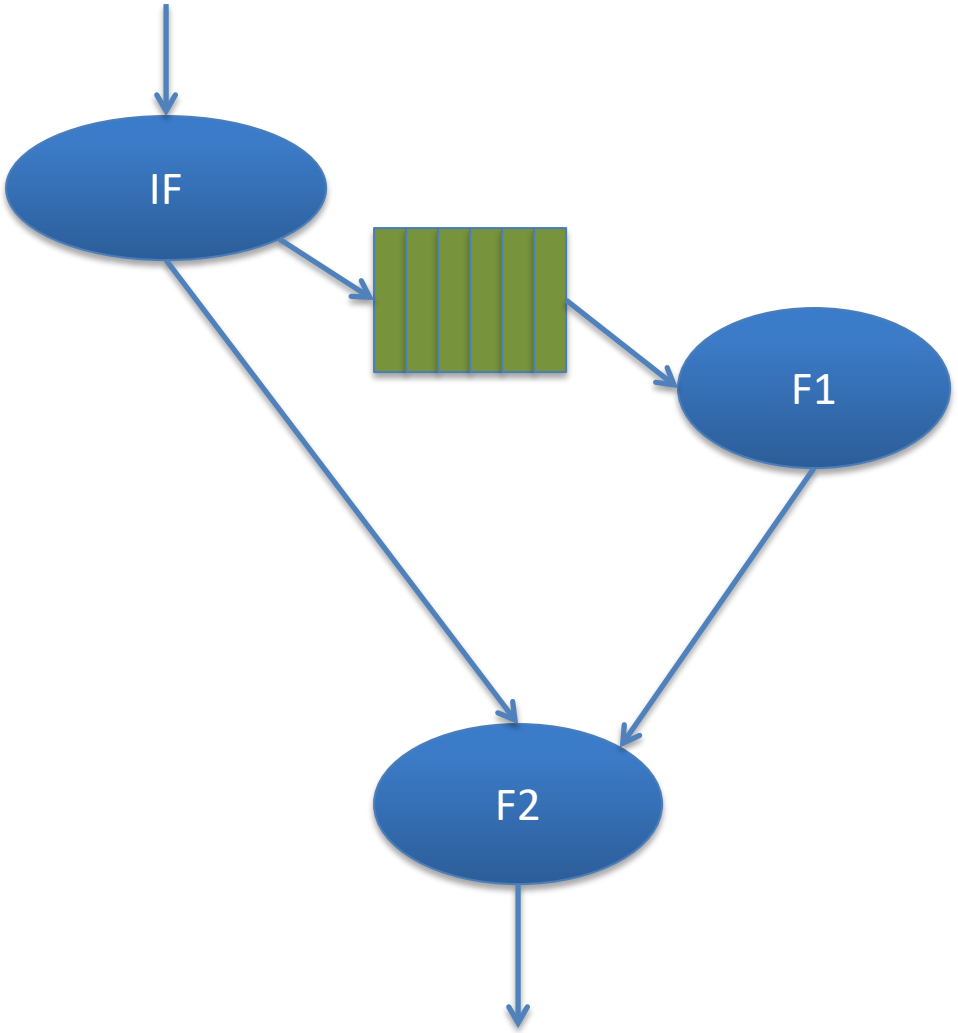
VP standards

- Focus on standards for VP development
  - SW and FW running on VP may have different requirements/standards
- Usage of C++
  - Performance requirements
  - Compliance with existing models
  - Most tools provide APIs
  - Linking of scripting environments for dynamic control e.g. Python, TCL, Lua
- Standards are important for IP exchange
  - Internal implementation might be different
  - Standards often imply standard interfaces/APIs
  - Standards typically allow easier tool deployment
- Tools can blur the line between standard and proprietary implementation
  - Additional functionality provided that is vendor specific
  - Model libraries may incur tool specific functionality

Modeling techniques

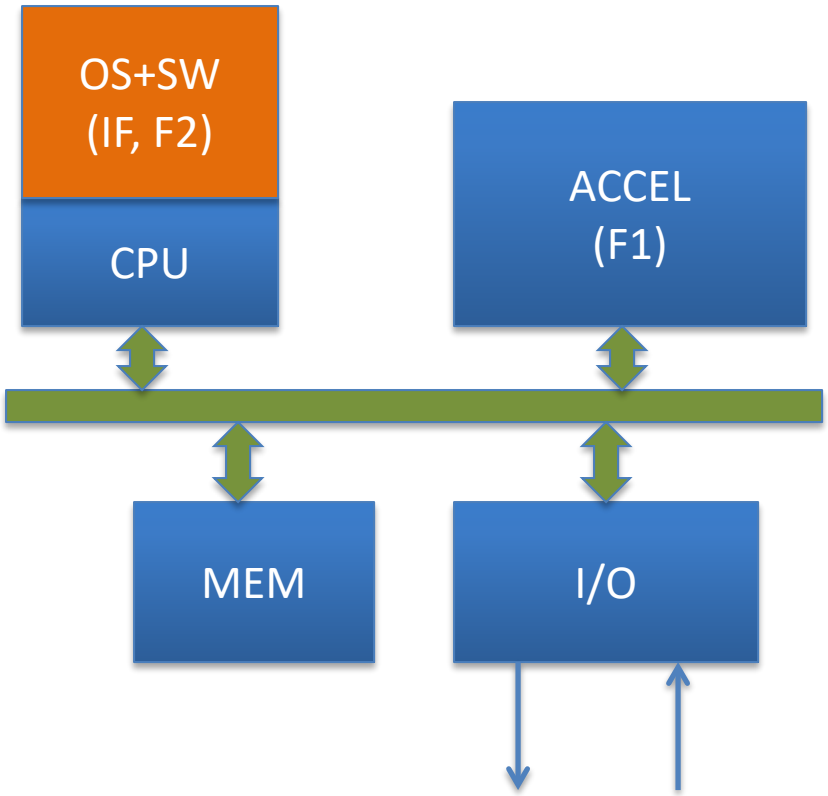
- Several modeling approaches are used to implement a VP and its components
  - Behavioral/untimed
  - Functional/loosely timed
  - Cycle-accurate/approximately timed
  - Register-transfer-level
- A particular VP usually is a mix-and-match of these techniques

Behavioral/untimed



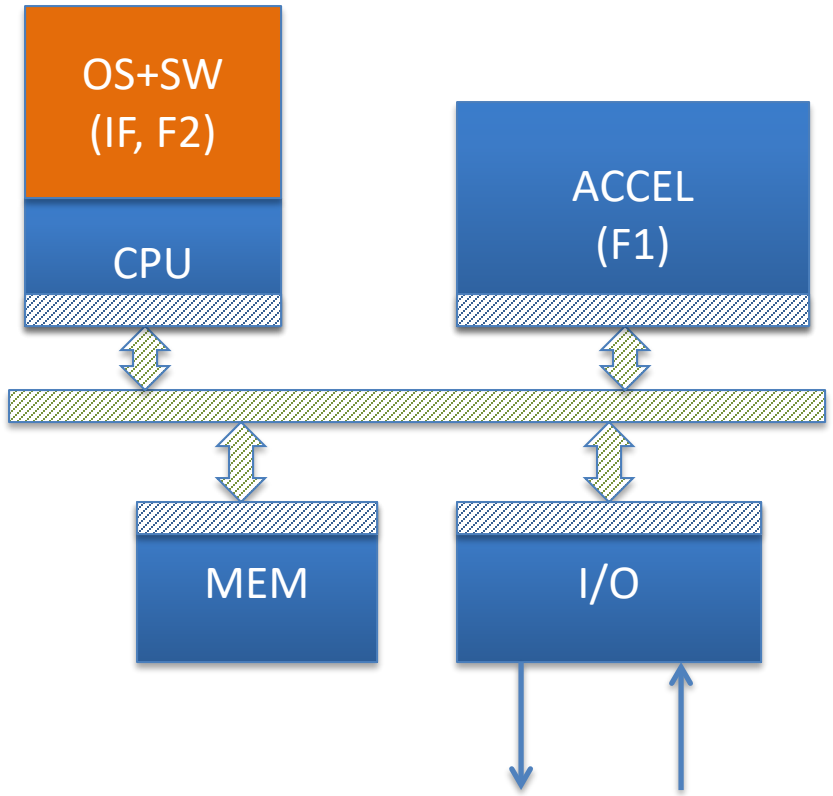
- The untimed model is a functional description having no notion of time
- A set of algorithmic blocks communicating using function calls or message pipes
- Keeps the causality and order of invocation

Functional/loosely timed (LT)



- The loosely timed model is a structural and behavioral refinement of the functional model.
- Mapping of functional blocks to HW and SW components and communication interfaces in-between based on a chosen architecture
- Subsystems can execute ‘ahead-of-time’
- Transactions at communication interfaces correspond to a complete read or write across a bus or network in physical hardware and provide timing at the level of the individual transaction

Cycle-accurate/approximately timed (AT)



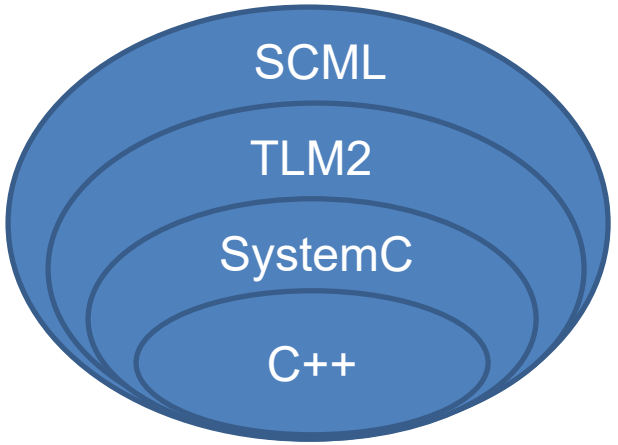
- Approximated timing on bus communication and on hardware resource access
  - Interface communication time
  - Average processing time in hardware IP
- Transactions are broken down into a number of phases corresponding much more closely to the phasing of particular hardware protocols

Register-transfer level (RTL)

- Models a block or SoC in terms of the flow of data between hardware registers, and the logical operations performed on this data
- Starting point for physical implementation by synthesis, place, and route

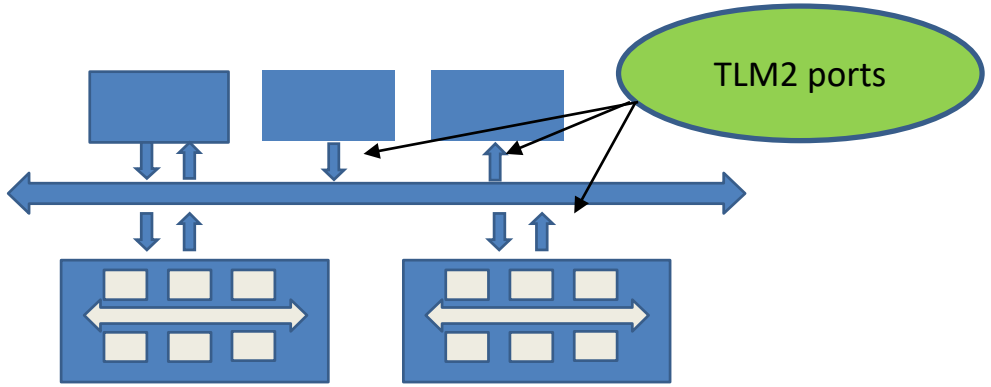
C++ class libraries for modeling

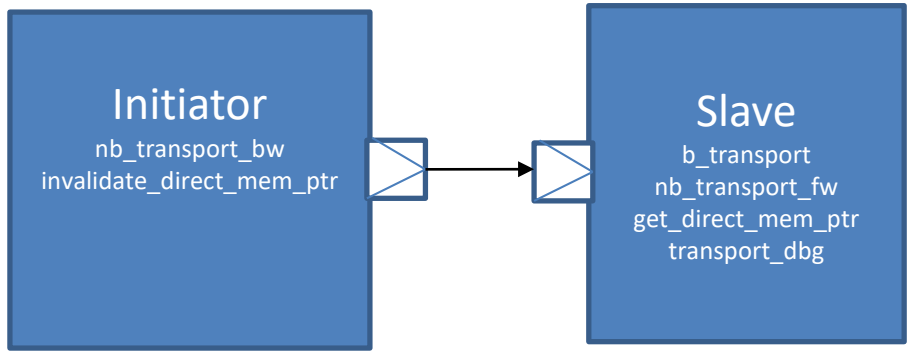
- SystemC is a class library on top of C++
  - Structural description elements
  - Data types
  - Event driven simulation kernel
- TLM2 modeling allows for interoperability
  - Reuse of existing components whether 3rd party or inhouse
  - Tool environments support TLM2
  - Interaction with more cycle-accurate models is well defined
  - Even bridging into RTL simulators is well defined and supported by (commercial) tools
- Productivity libraries enable faster modeling of common components
  - Registers, memories, infrastructure tasks
  - Examples of libraries
    - Propriatary: Synopsys SCML, Intel ISCTLM, ASTC Genesis
    - Open Source: Greensocs Greenlib, MINRES SC Components



TLM2 based modeling

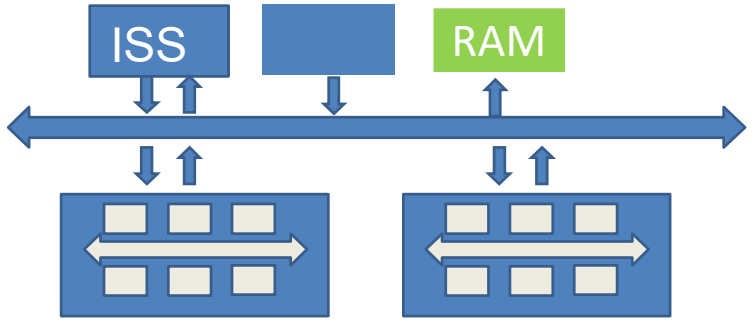
- TLM2 ports are the interfaces between modules
  - Retain connection to architectural view of SoC
  - Allows for reuse of components
  - Focus on interfaces, not on functionality
- Provides infrastructure for modeling efficiency
  - LT modeling allows optimization of events
  - Direct-memory-interface (DMI) calls can provide very high memory-access efficiency
  - Temporal decoupling allows models to delay synchronization with other components

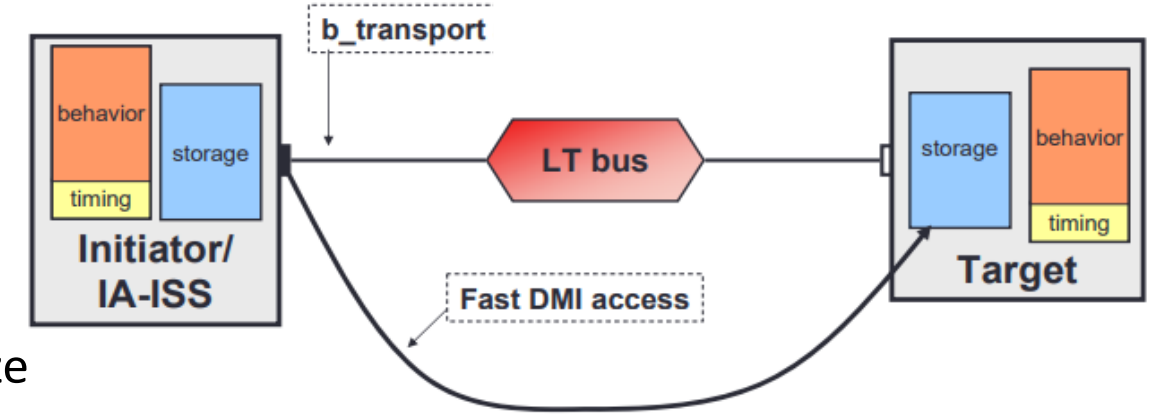




Modeling efficiency with TLM2 LT

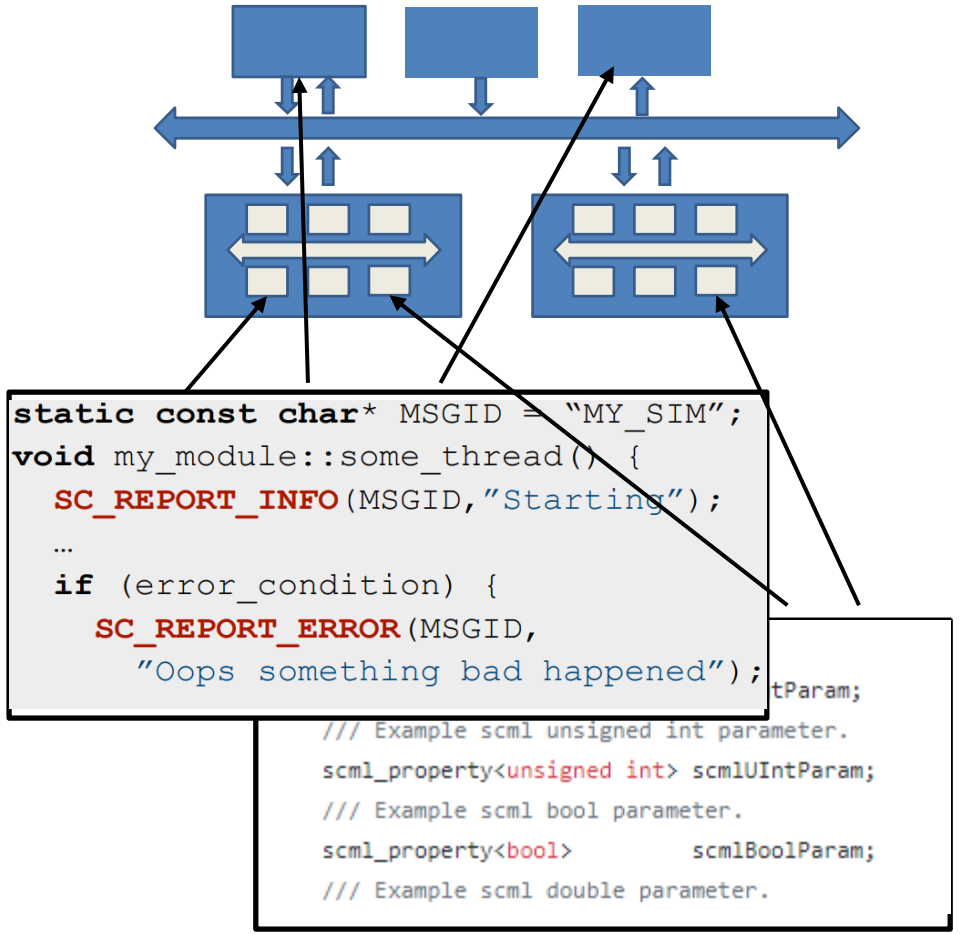
- LT (loosely timed) specifies different transport mechanisms
  - b_transport() as blocking access per transaction
  - DMI calls to request memory access by pointer
- Design goals
  - Minimizing synchronization with environment
  - Localizing workload
  - Allows for registration of callback functions
- Works well in conjunction with temporal decoupling
  - Avoid synchronization in blocking calls if possible
  - Processor models can decide when to synchronize





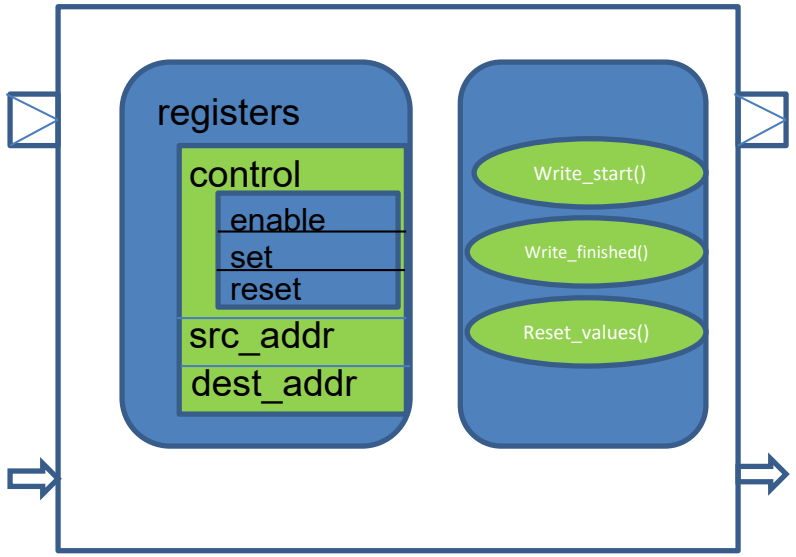
Productivity Libraries

- Common elements should be modeled on top of common classes
  - Registers, Memories, etc.
  - Efficient for stubs or preliminary implementations
- Common infrastructure tasks should be modeled on top of common classes
  - Logging
  - Parametrization
  - Domain handling (Clock, reset, power)
- SCML, ISCTLM, and Genesis are examples of (proprietary) productivity libraries
  - Concepts and implementations might serve as starting point for standardization



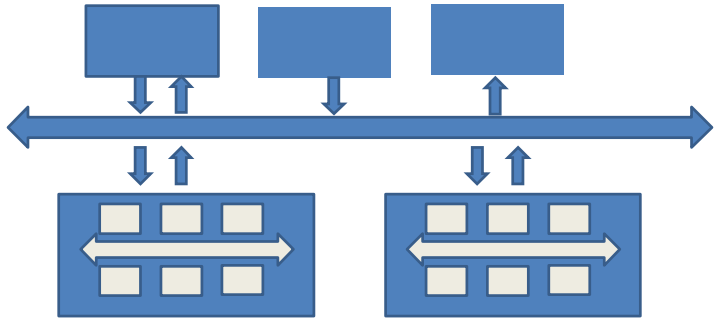
Register modeling

- Registers are one of the main interface between HW and SW
- Registers mean different things to HW and SW groups
  - HW contains many registers, but only few are exposed to SW
  - SW visible registers should be implemented, documented and tested
- Consider automated code generation
  - Large amount of registers
  - changing requirements
  - Meta data formats like IPXACT, RDL, RAL
- Having a model that allows consistent HW and SW access modeling is important
  - Access modes, reset and retention values
  - Efficient access through callback functions
  - Productivity layer provides register classes



Model meta data

- Stitching models together is a recurring and error prone task
- Using meta data is often useful
  - Data visualization
  - Data exchange
- IPXACT is a XML based schema for interface descriptions
  - Some tools can provide or read IPXACT
- XML parsing for simple tasks can be done using Open source libraries
- Other examples for meta data representations are RDL, RAL, sysML



```
        <spirit:name>fl-can_2.0</spirit:name>
        <spirit:version>1.0</spirit:version>
        <spirit:directConnection>false</spirit:directConnection>
        <spirit:maxSlaves>0</spirit:maxSlaves>
        <spirit:signals>
                <spirit:signal>
                        <spirit:logicalName>CANH</spirit:logicalName>
                        <spirit:onMaster>
                                <spirit:bitWidth>1</spirit:bitWidth>
                                <spirit:direction>inout</spirit:direction>
                        </spirit:onMaster>
                </spirit:signal>
                <spirit:signal>
                        <spirit:logicalName>CANL</spirit:logicalName>
                        <spirit:onMaster>
                                <spirit:bitWidth>1</spirit:bitWidth>
                                <spirit:direction>inout</spirit:direction>
                        </spirit:onMaster>
                </spirit:signal>
        </spirit:signals>
</spirit:busDefinition>
```

Improvement of standards

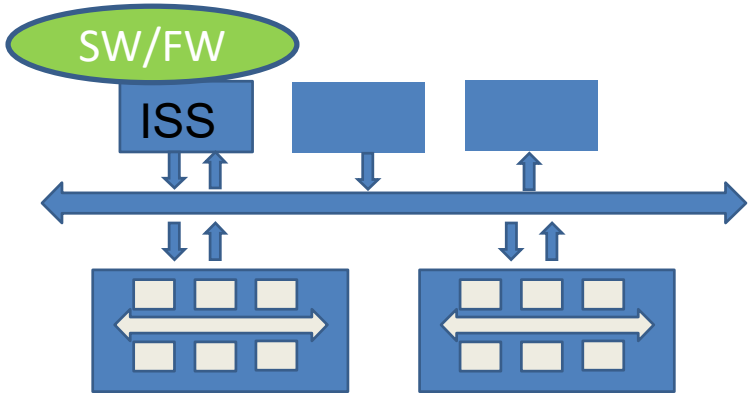
- TLM2 provides a generic transport mechanism and extension mechanisms
  - Specific protocols are not described
    - AMBA, PCI, MIPI are left to implementer
- Productivity libraries are not standardized
- More support for common infrastructure tasks
  - Tracing
  - Profiling
  - Parallelization

Using IP for Virtual Platforms

- VP represents hardware
  - SoC design typically contains a significant number of 3rd IP components
  - Ideally all IP should come from same source (RTL, architecture, verification, VP models)
- Availability of IP for VP is improving, but still limited
  - Availability of models often defines the usage
  - Consider Co-simulation
- Processor-based models that execute SW
  - User of VP is exposed to those models
- Peripheral models
  - Execute functionality autonomously, but under control of SW
- Interconnect models
  - Connecting different elements based on addresses

Modeling processors

- Core functionality of VP
- Need support for high speed, good debug interface, profiling, etc.
- Most embedded processor providers also provide ISS models
  - Speed versus timing accuracy
    - Dynamically configurable to run in performance mode or accuracy mode
  - Links to debug environment and other SW tools should be consistent with user expectation

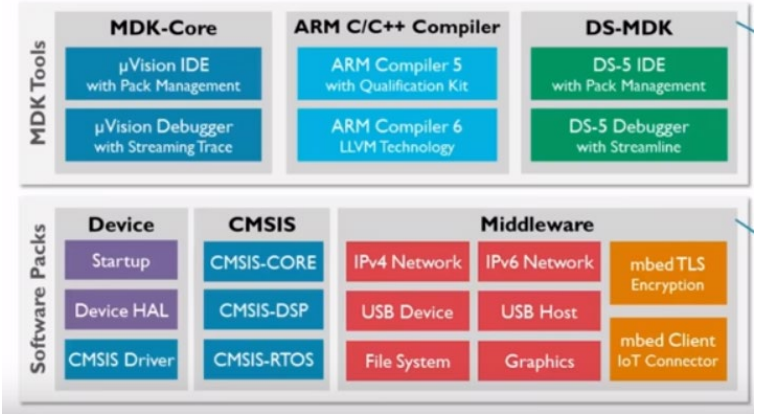


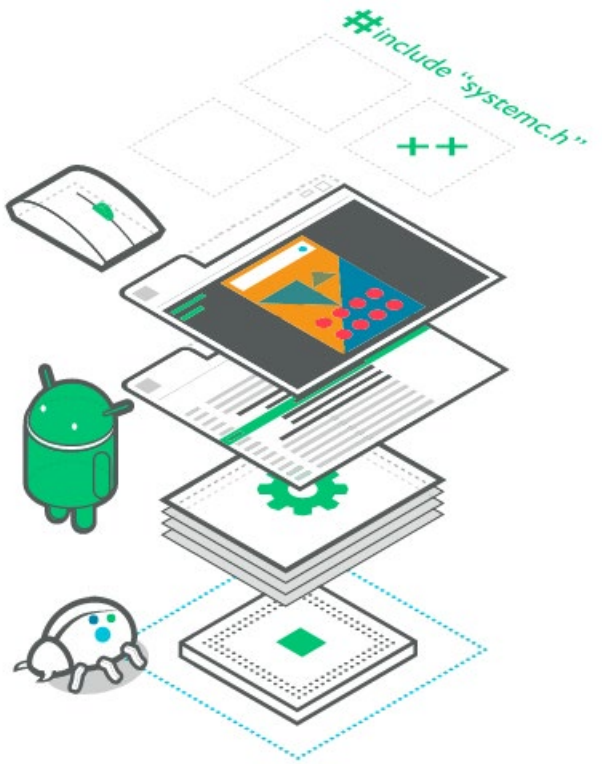
Processor modeling techniques

- Interpreting execution
  - Slow but easy to implement and accurate
- Just-in-time (JIT) compiling/dynamic binary translation (DBT)
  - Virtualization library converts target instructions to host instructions e.g. using LLVM or QEMU right before executing them
- Host based/host compiled simulations
  - Instead of running the target SW/FW binary on a target processor (model), run a host-compiled representation interacting with the model of the platform
  - Virtualization environments allow running the SW/FW binary

ARM processor models

- ARM fast models
  - LT based models for speed with debug interfaces
- Cycle accurate model based on Carbon tool translation
  - Cycle accuracy guaranteed

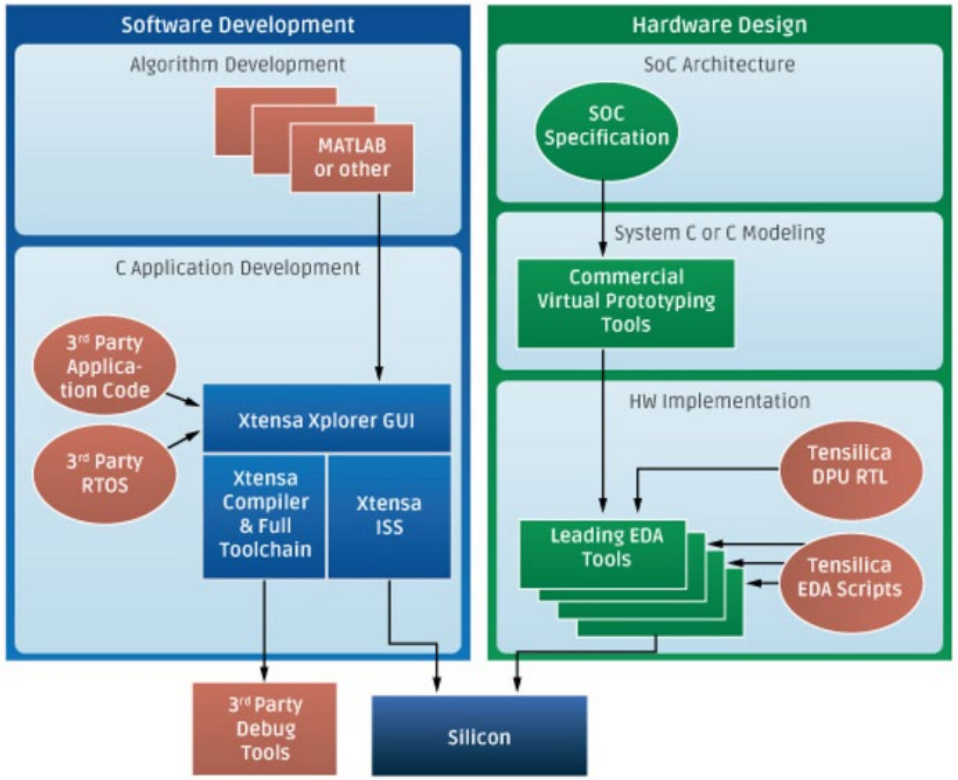




Source: ARM support website

Tensilica processor models

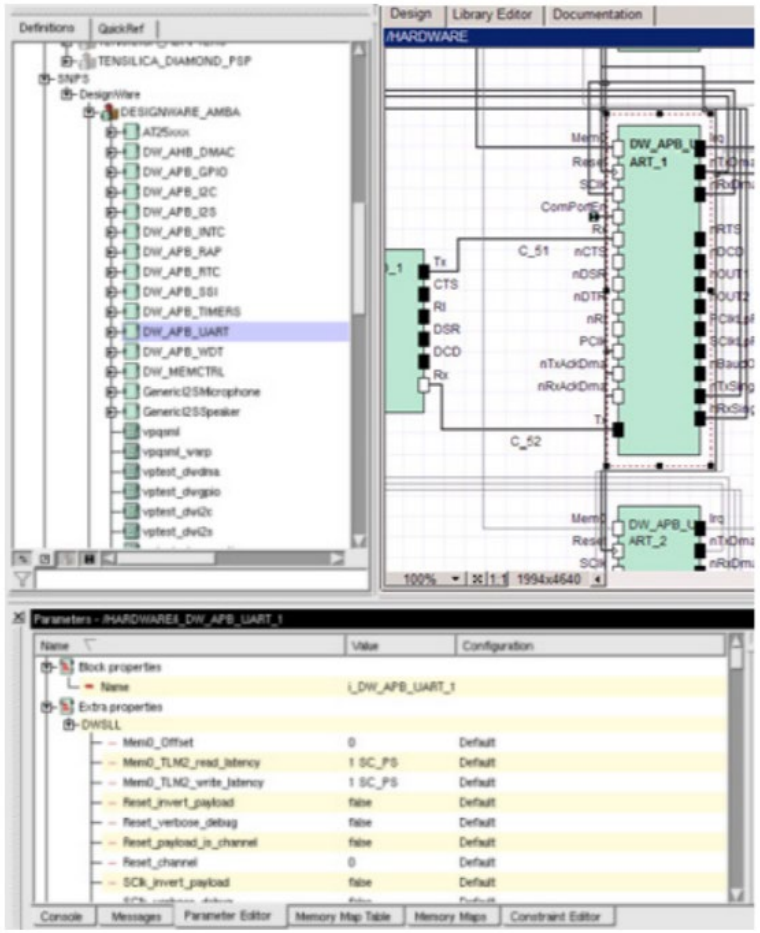
- ISS from Tensilica is highly configurable
  - Turbo mode (JIT) is using various mechanisms to speed up simulation
  - Non-turbo mode (interpreting) is instruction accurate
- Models provide TLM2 based interfaces
- Tensilica provides extension to gdb for debugging
- Integration into Tensilica tool environment by attaching simulation through TCP port



Source: Cadence support website

Modeling peripherals

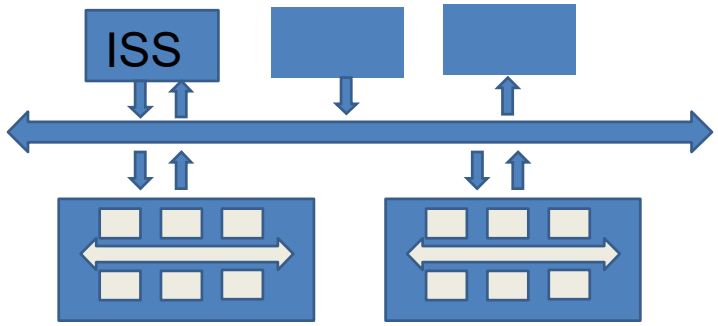
- Typical peripherals are UART, PCI, USB, timers
  - May connect to the ‘real’ world by using the host workstation resources
  - Not main functionality, but important to get FW contents right
  - Synopsys DW library provides TLM2 based versions of various IP
  - Some vendors react on demand
- Various versions of peripherals are common
  - Upgrading even to a new version may imply changes in the interface
- Peripherals can have significant contribution to system performance
  - DMAs, peripherals sending video data, real-time constraints

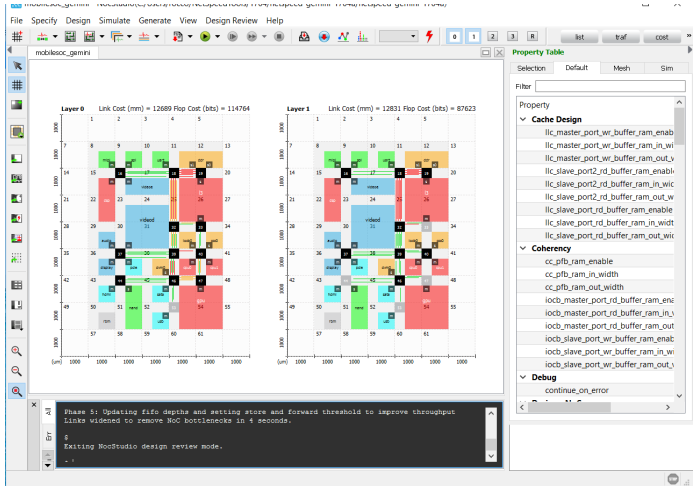


Source: screen shot from PlatformCreator

Modeling interconnect

- The model that connects everything
  - Mostly address map in VP context
  - Consistent view of registers
  - May contain some SW visible functionality like firewalls, address shifting, domain crossings
- Some libraries and tools contain generic models
  - Fast initial implementation
  - High performance
- Some vendors provide TLM2 LT interconnect model, e.g. NetSpeed
  - Ensures consistency within design flow





Source: screenshot from NetSpeed NocStudio

Custom build models

- Custom models are necessary, for instance
  - In house IP
  - Important IP without any available 3rd party model
  - Extensions to existing models
- Consider cost of maintenance
  - Build based on productivity libraries
  - Reuse existing models for instance untimed models
- Consider RTL co-simulation or translation like Carbon tools

Commercial tools available for VP development (exemplary list)

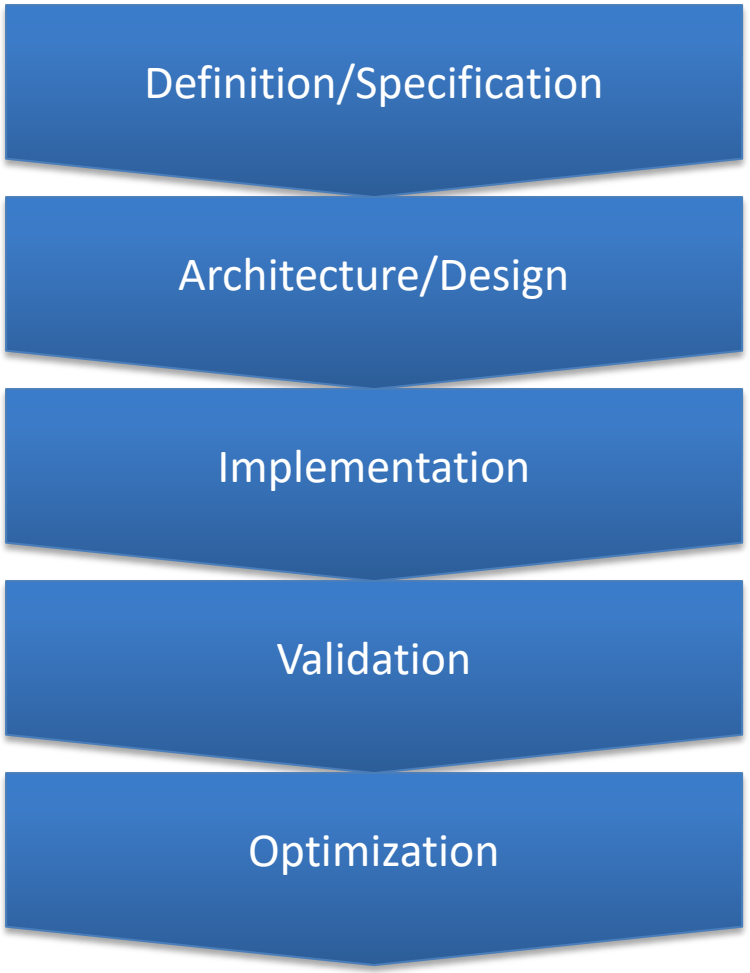
- Synopsys Processor Designer, Virtualizer
- Cadence Virtual System Platform
- Siemens Mentor Vista
- MathWorks Matlab, Simulink
- ASTC VLAB, Processor Modeling Studio
- Wind River Simics
- Xilinx Vivado
- ARM Carbon Model Studio, SoC Designer Plus

SystemRDL, IPXACT & SystemC

DEMO

DEVELOPMENT AND APPLICATIONS OF VP

SoC/system & VP development phases

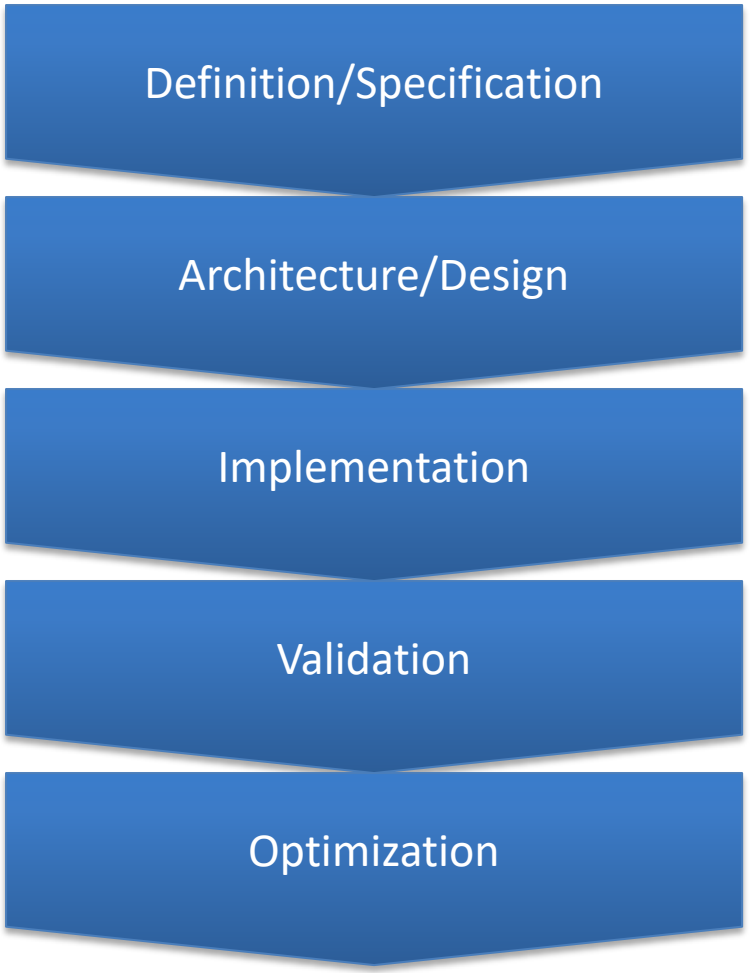


- Behavioral description of the SoC or system
- Partitioning and Mapping to Architecture (HW) and SW
- HW & FW development
- Functional, timing, power, ... validation
- SW development and further optimization

accellera SYSTEMS INITIATIVE

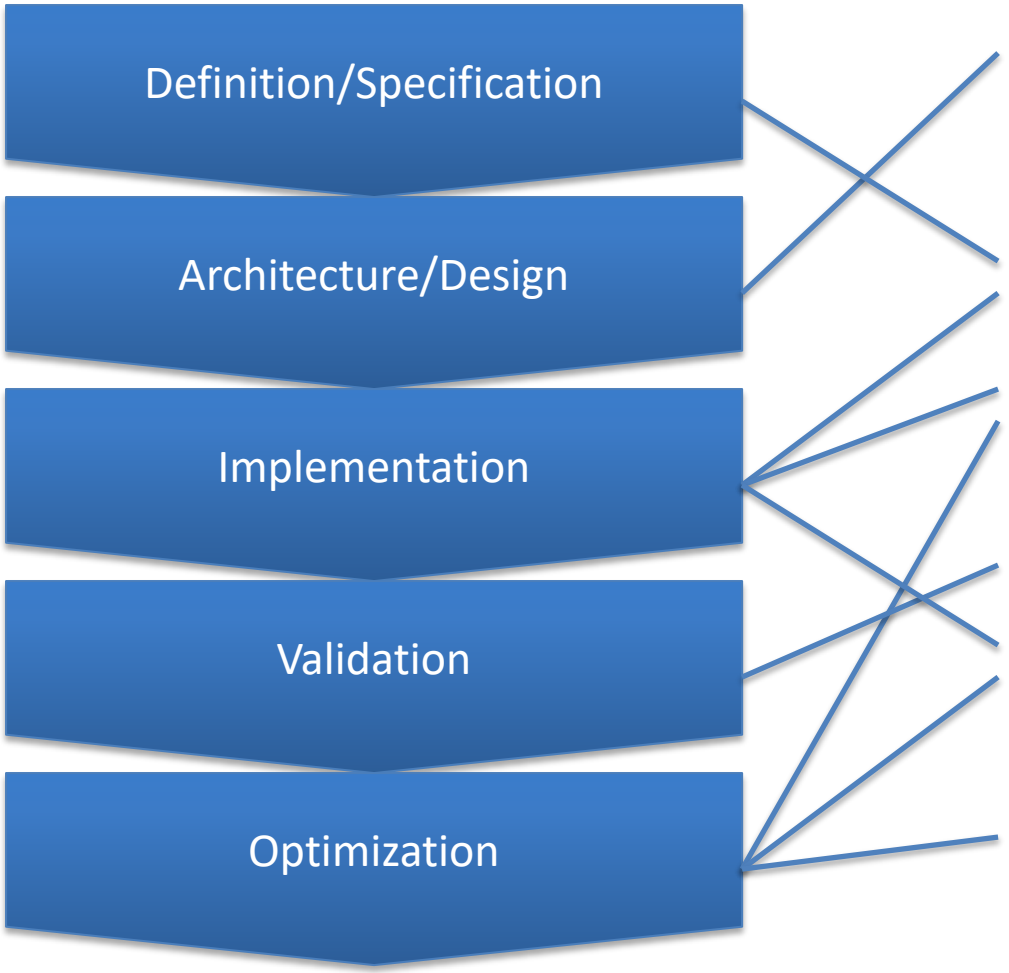
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

User roles in the VP dev phases



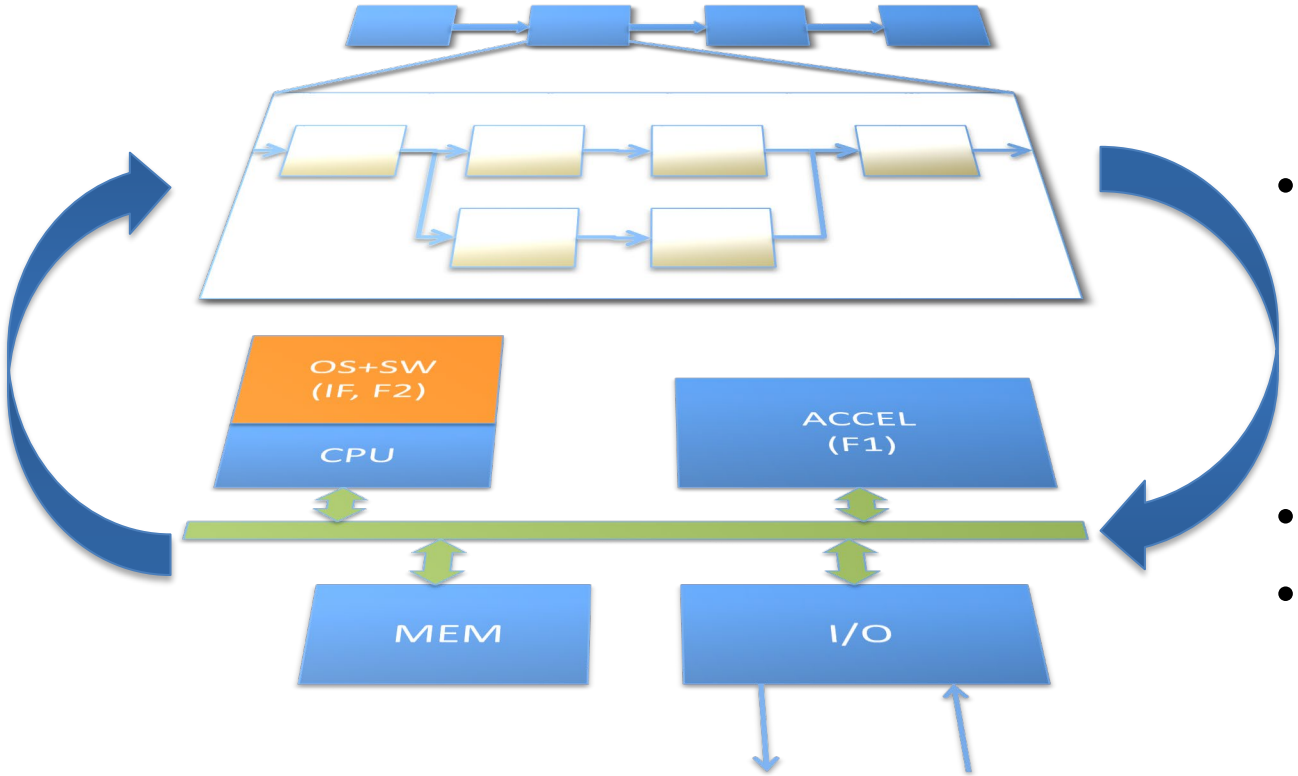
- Platform authors, IP authors
- Platform authors
- Platform authors, IP authors
- Platform authors, platform users (SoC & system integrators)
- Platform users

VP uses cases



- Architectural exploration and performance analysis
- Functional verification
- FW development (driver, HAL, OS)
- System performance validation
- Debug & analysis in FW/SW development
- Continuous integration & test-driven SW development

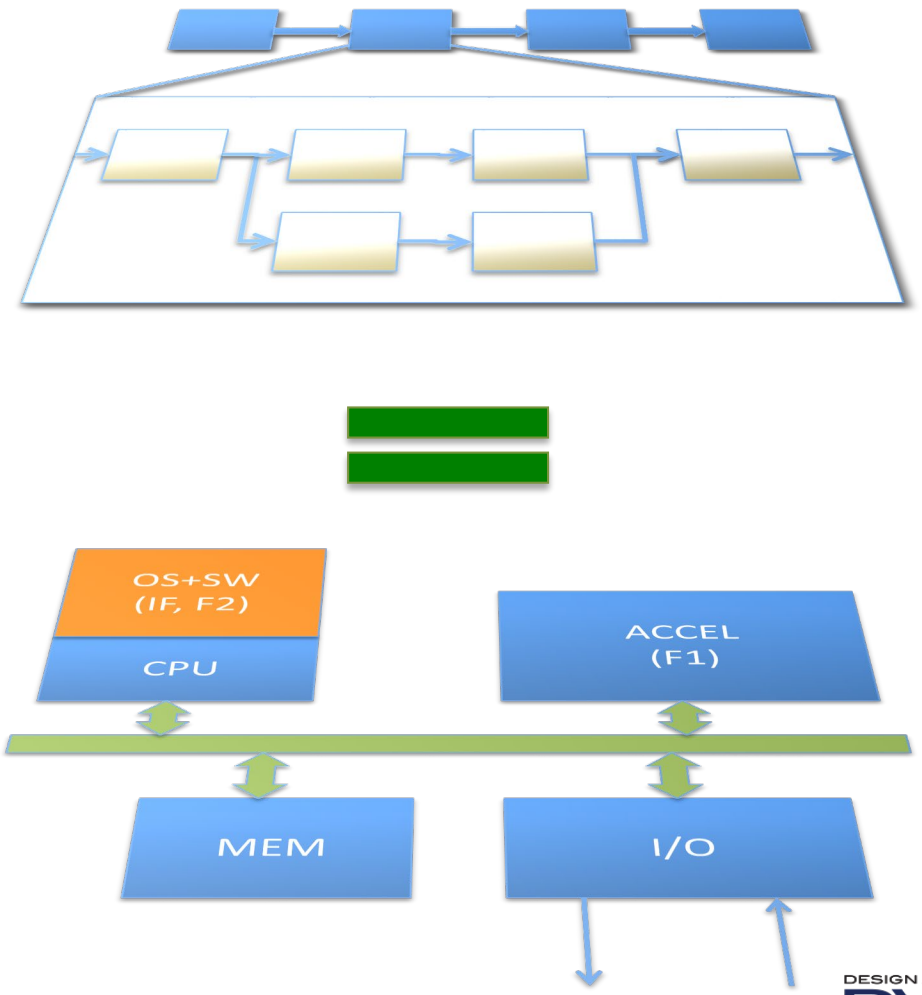
Architectural exploration & performance analysis



- Mapping of a behavioral model to one or more points in the architectural space consisting of different HW implementations
- Evaluation based on performance characteristics for different system architectures, such as a HW/SW split, communication system, or system components
- Typical use case for platform authors
- Important properties: accuracy wrt. to performance metrics i.e. timing, latency, throughput, power

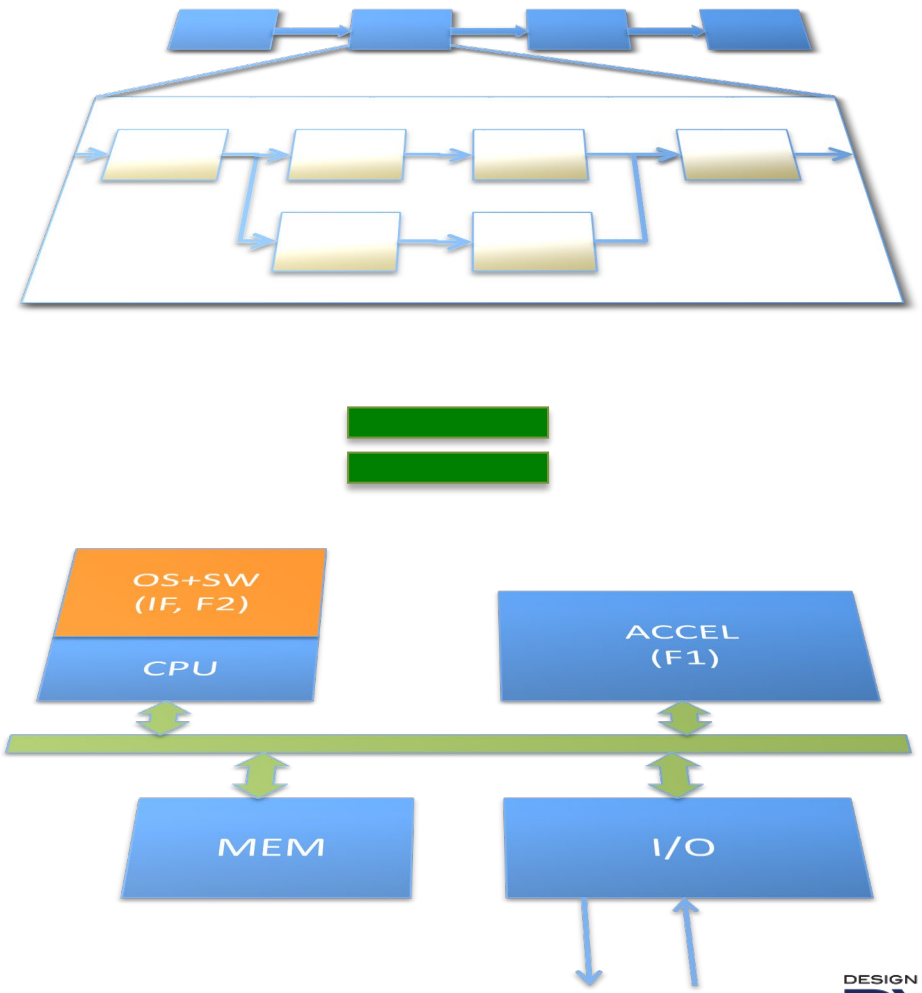
Functional verification

- Behavioral model allows early development of system and integration test thus validating the specification
- Refined VP with final architecture and HW/SW partitioning serves as executable specification for hardware implementation

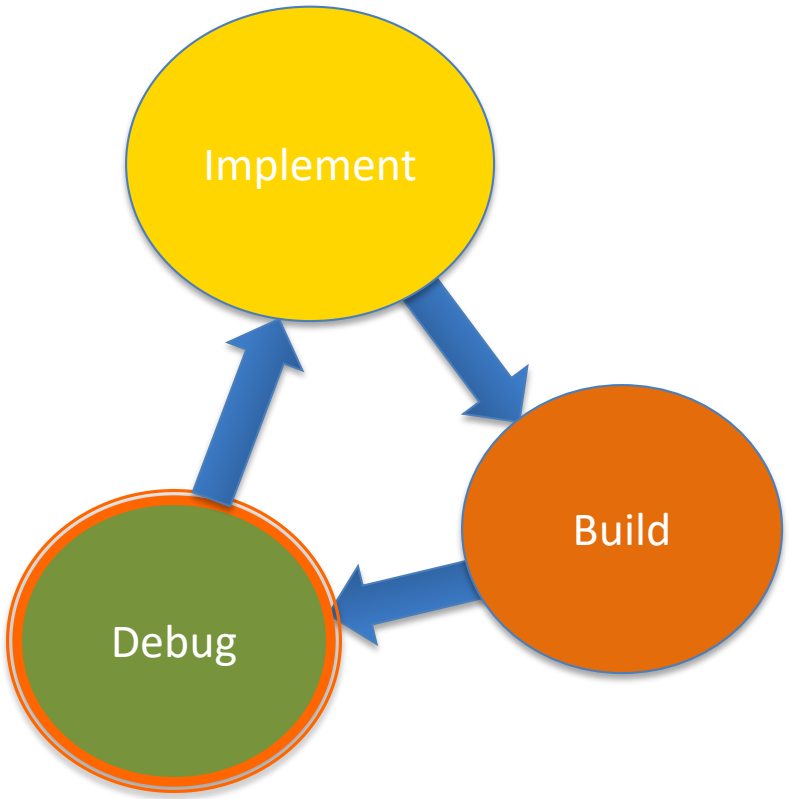


Functional verification

- Can be used as a reference model for the SoC implementation as well as for component development (RTL block verification)
  - Replace (IP-)blocks by their RTL counterpart and validate in system context
  - VP as part of the block test bench
- Reuse of stimuli from VP to RTL, component to system
  - E.g. using portable stimuli (Accellera standard)



FW development (driver, HAL & OS)



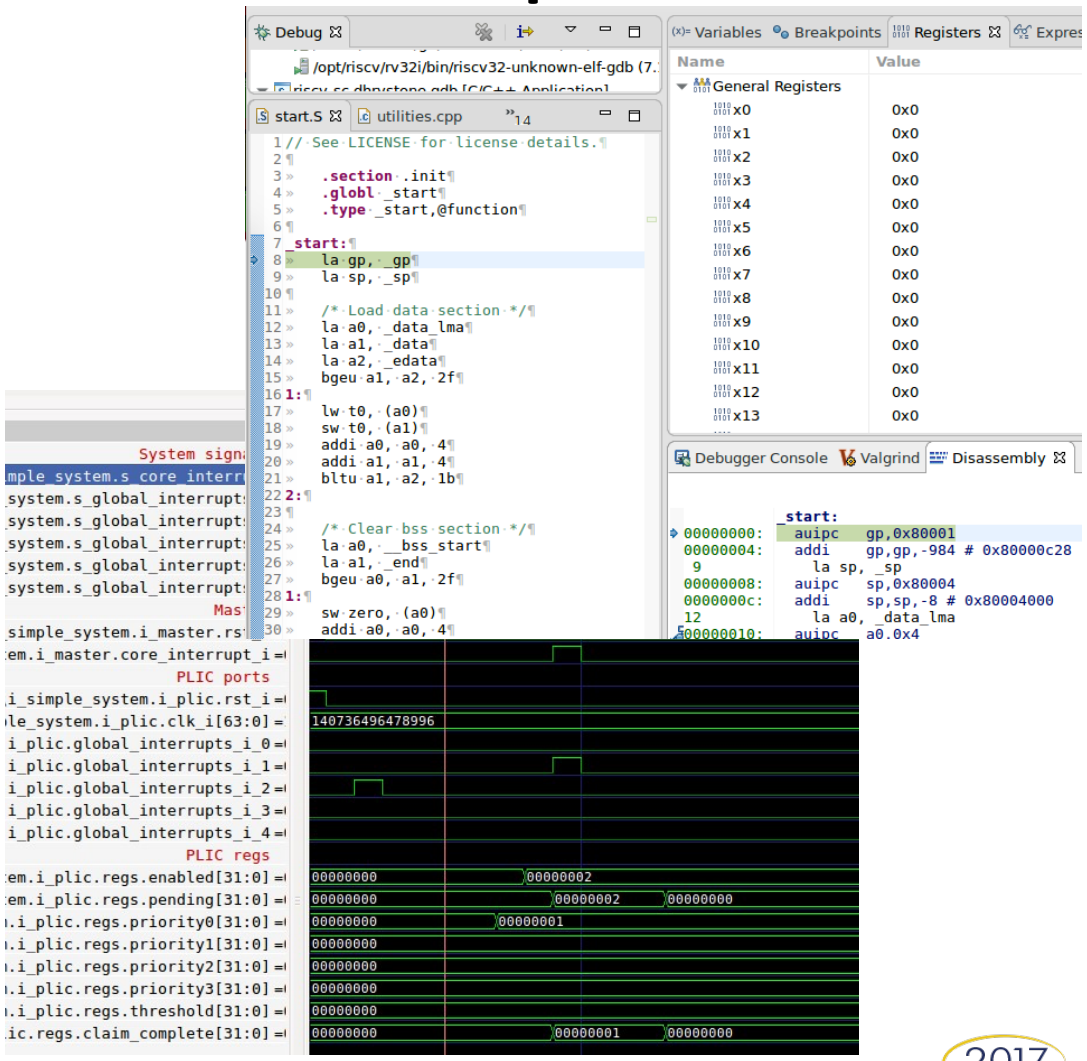
- Use the VP as a vehicle to execute target SW by simulating HW behavior
- Allows shift-left by starting SW development early in the design process
- Used by platform authors and system integrators
- Important properties: balance between speed vs. accuracy
- Comprehensive analysis, visibility and debug means needed
  - Logging of HW and SW events (e.g. via host I/O)
  - Signal & transaction tracing
  - Non-intrusive debugger integration

System performance validation

- Validate performance of platform and SW in application scenarios within target environments
  - Reception of data at wireless modems
  - Event response time in realtime critical systems
- Used by system integrators

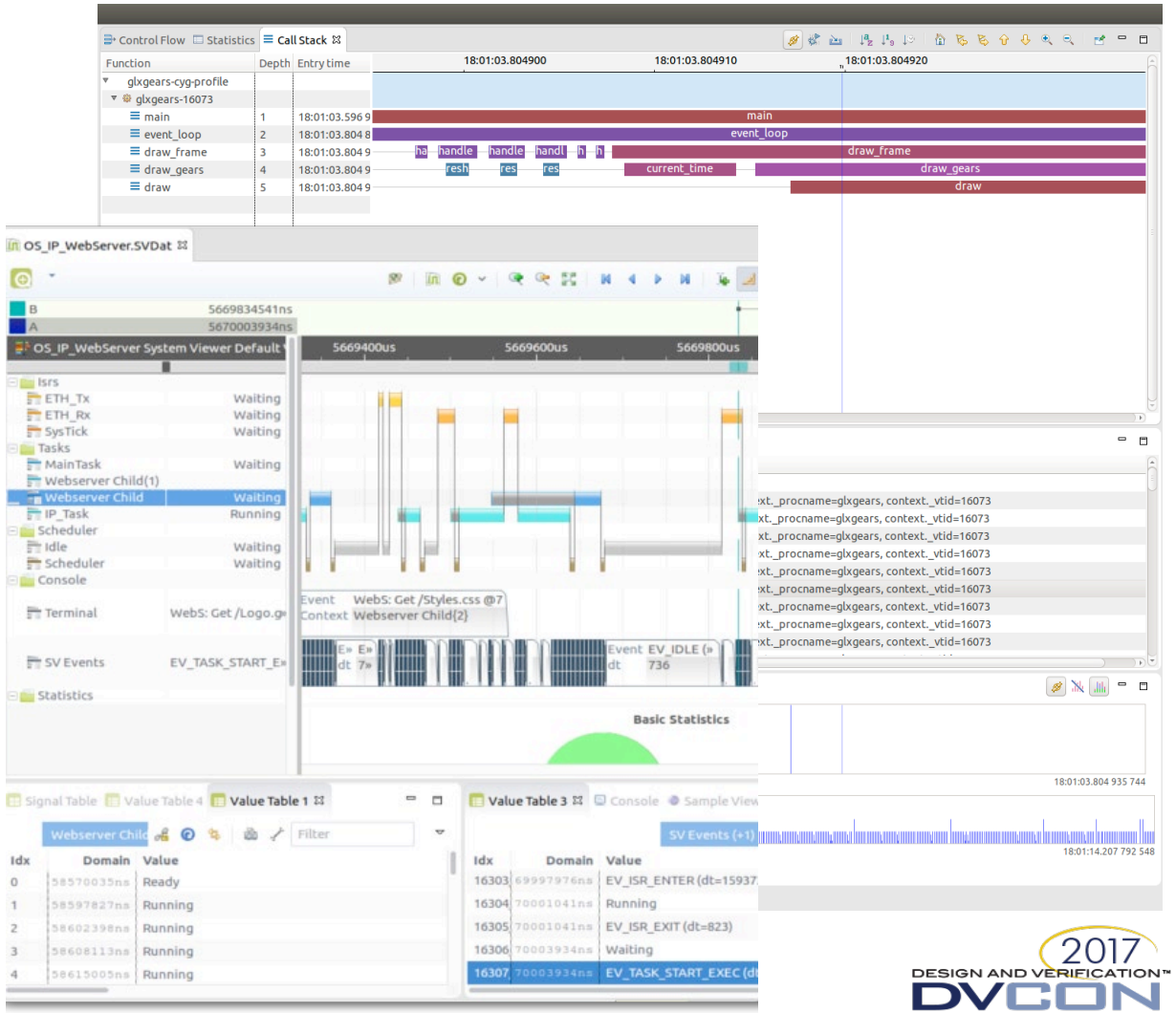
Debug & analysis in FW/SW development

- Similar to FW development
- Used by system integrators
- Focus even more on speed as not only the platform is being simulated rather also the target environment e.g. cellular base stations, entire motor controller including the control loop containing actuators and sensors



Debug & analysis in FW/SW development

- Comprehensive analysis, visibility, tracing, and debug enables non-intrusive observation and debug of behavior thus easing the debugging
- Tools like Eclipse Trace Compass or impulse allow to fuse HW and FW/SW events to ease debugging system behavior



Continuous integration & test-driven SW development

- The all-in-software approach allows to deploy modern SW development techniques like continuous integration (CI) and test-driven design (TDD)
- Each feature is being (unit-) tested so that functionality regressions can be detected early
- Each SW change is tested before propagating to the main line of development
- Allows close monitoring of the eSW development progress
- Maximum benefits in situations where one software system addresses multiple hardware variants in different system contexts
- Used by system integrators

FW debugging in Eclipse using VP as target.

DEMO

ADVANTAGES, LIMITATIONS, AND CHALLENGES

Advantages

- Early availability & easy reconfiguration
- Observability
- System fault injection & fault simulation
- Simulation speed and accuracy
- Scalability
- Enabler of agile eSW development methodology

Limitations

- Simulation speed vs. accuracy
- Limited multi-threading capabilities in SystemC
  - advantageous for sub-system integration and multi-core simulations
- Missing standards for runtime configuration, inspection, logging and tracing
  - integration of IPs from different vendors still very tedious

Challenges

- Simulation speed vs. accuracy
- Use case broadening (FW/SW performance validation, dynamic power estimation)
- Integration of sub-system and systems
  - happens at at different levels (sub-component, component, sub-system, system)
  - re-use of test infrastructure between levels
  - multi-core debugging, SMP, AMP
  - efficient sub-system integration process – dynamic routing and linking
  - availability of lower FW/SW layer required to integrate higher layer SW
- Conflict of interest between platform authors, tool and IP vendors
  - platform author need maximal flexibility in terms of IP selection and tool support
  - IP and tool vendor aims to lock-in platform author and user
- Exchange of VP platforms between platform authors and platform users
  - legal and IP protection challenges

Simulation speed vs. accuracy

- Simulation speed is crucial for the adoption of VPs
- Modern ISS reach several tens or hundreds of MIPS depending on the complexity of the processor
- Modeling the entire platform slows down esp. If components or communication interfaces need to be modeled in more detail and/or with higher accuracy
- Single-threaded SystemC kernel

Simulation speed and accuracy

- Ways to address this:
  - Mix-n-match approach (coarse and fine grain modeled components)
  - use case adapted VPs (variants)
  - Multi-threading and simulation partitioning
  - Simulation check-pointing
  - Faster host hardware 😉

Early availability & easy reconfiguration

- Earlier system availability by agile development
  - Implementing only functionality that is needed for the use case (software development, performance analysis)
  - Incremental extension until completion with intermediate deliveries
- As communication is abstracted, a VP can easily be re-partitioned
  - Graphical tools are available to ease this
- By using software build mechanisms or runtime configuration mechanisms it's easy to create and maintain variants or derivatives to suit different requirements

Observability

- Simulation can be stopped upon occurrence of important or interesting events
- State of the platform (both the hardware and the software state) is frozen and can be examined e.g. using debuggers
- Non-intrusive debugging: other than ‘real’ hardware the system cannot determine that it has been stopped
- Same or better debugger access than hardware
- Extensive logging and tracing e.g. VCD or transaction recording allows comprehensive post-simulation analysis

Observability

- Combining hardware traces and logs with SW events and traces gives even more insight
- Detailed analysis of system w/o
  - Expensive on-chip measurement
  - Slow HDL simulation

Scalability

- The all-in-software approach executes target code on standard hardware
- No dependencies on special hardware
  - Other than FPGA, Emulation or HIL
- Each developer has its own target system to debug the code
- Easy scalability by adding off-the-shelf hardware like workstations or servers

Enabling agile eSW development methodology

- The all-in-software approach allows to deploy modern SW development techniques like continuous integration (CI) and test-driven design (TDD)
- It can be easily scaled to meet developers demand and computing requirements for CI
- Allows close monitoring of the eSW development progress and provides benefits:
  - Better code structure and quality by frequent checking and extracting software metrics
  - Easier debug and less impact (less code to roll-back)
  - Fewer major integration bugs (detected early and easy to track)
  - Constant availability of a "current" build for testing, demo, or release purposes

References

- IP-XACT, RDL, SystemC/TLM 2.0: http://www.accellera.org
- ASTC VLAB: http://vlabworks.com/
- ARM support website: http://infocenter.arm.com
- Tensilica support website: https://ip.cadence.com/support
- GreenSocs Greenlib: https://www.greensocs.com/docs
- MINRES SC-Components: https://www.minres.com/#opensource
- Eclipse Trace Compass: http://tracecompass.org/
- impulse: http://toem.de/index.php/projects/impulse

Questions