Efficient use of Virtual Prototypes in HW/SW Development and Verification

Rocco Jonack, MINRES® Technologies GmbH
Eyck Jentzsch, MINRES® Technologies GmbH

Agenda

- Virtual prototype by example
- VP use in HW Development
- VP use in SW Development

Eyck Jentzsch

VIRTUAL PROTOTYPE BY EXAMPLE

Motivation

- Usage of VP is beneficial as part of cyber-physical systems (CPS)
- CPS– mixture HW, SW running on that HW and the surrounding system consisting of sensors and actors interacting with environment
- Useful for automotive, medical, (I)IoT but also others
- VPs allow flexible and early prototyping
- This tutorial aims to illustrate these statements

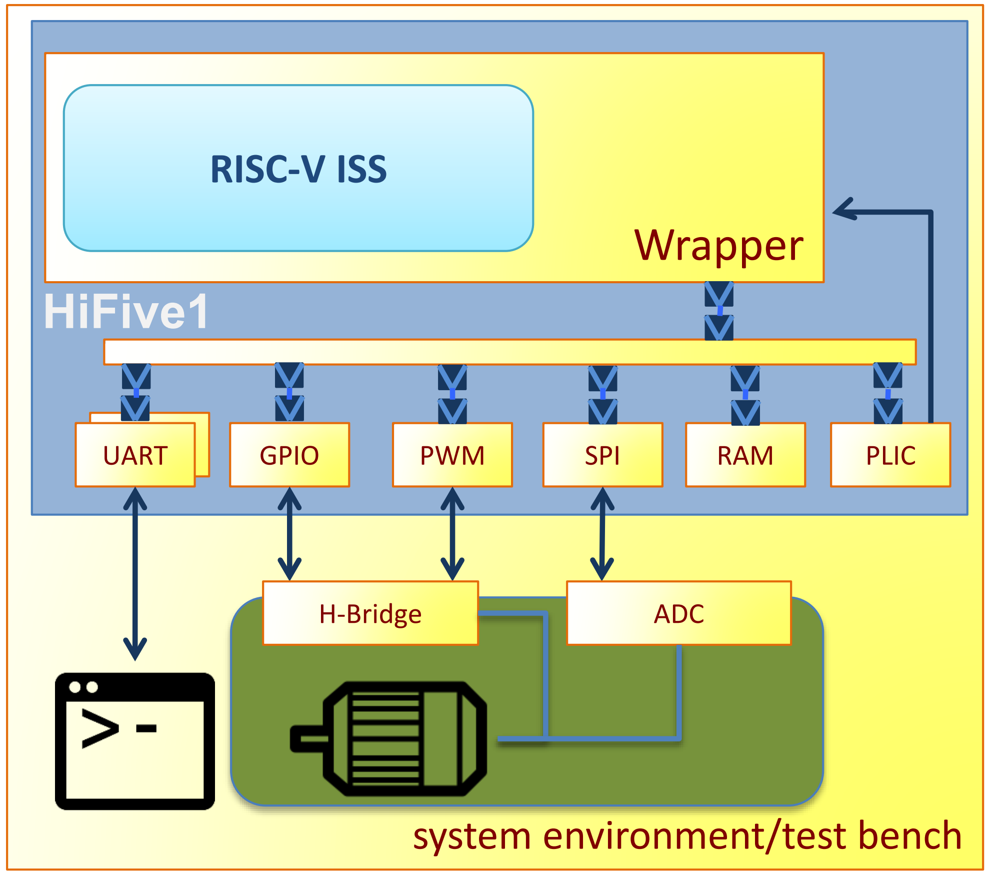
Overview

- The Virtual Prototype shown implements a Brushless-DC (BLDC) motor control
- As such it contains analog and digital components interacting with each other
- There are 2 flavors of the VP:
  - a pure digital model solely using SystemC
  - a analog/mixed signal (AMS) model using SystemC/AMS

Structure of the Platform

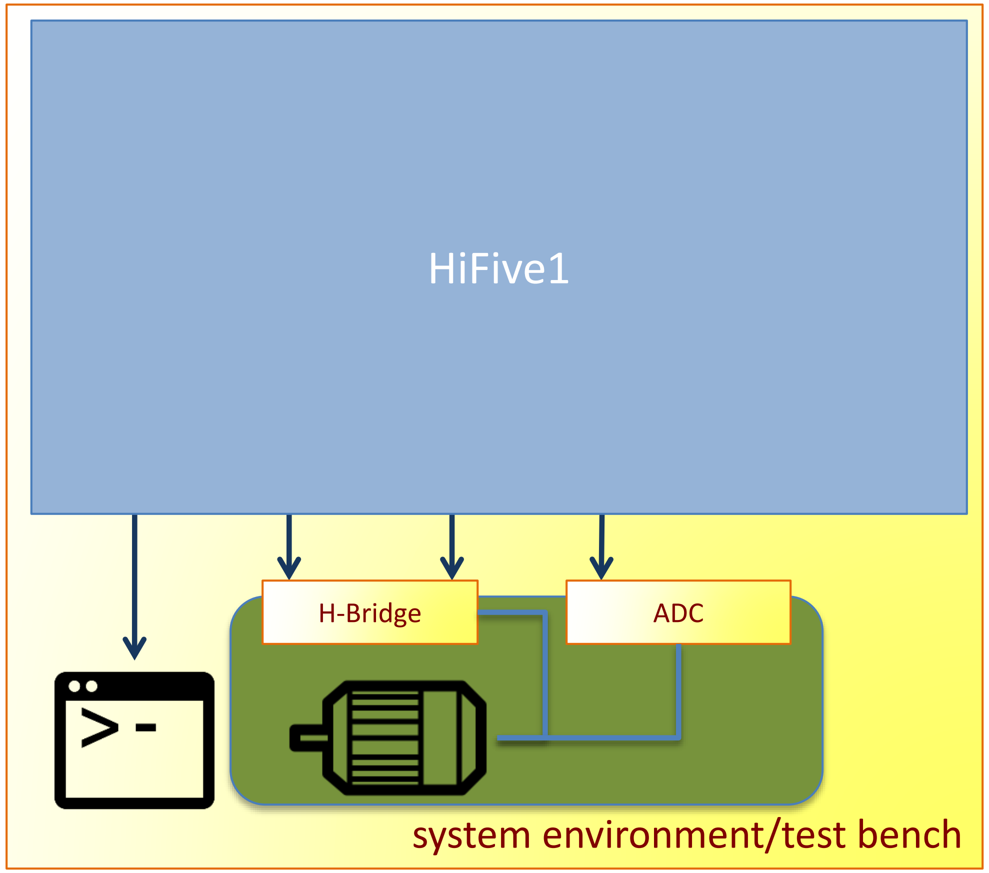
The model consists of

- Motor & load model
- ADC, H-Bridge
- Microcontroller
  - Peripherals
  - Interconnect
  - RISC-V ISS



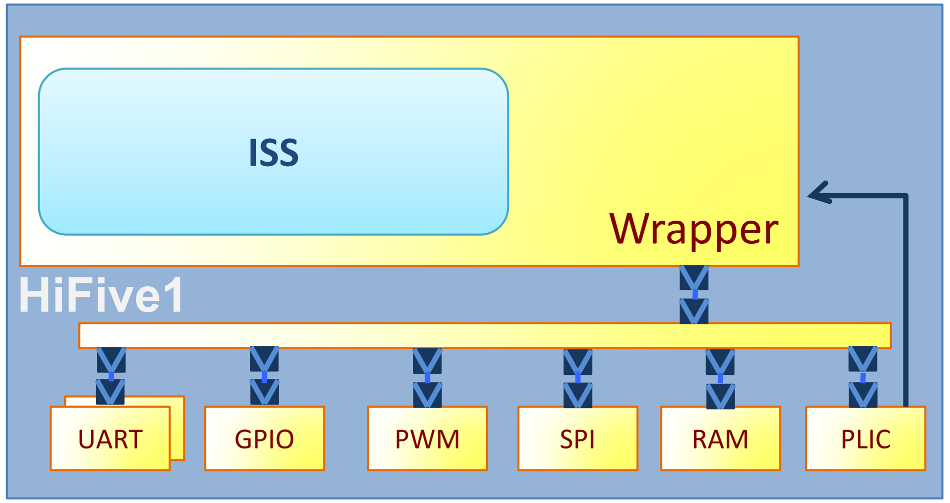
System components

- BLDC Motor model
  - Ordinary Differential Equation (ODE) model solved using Runge-Kutta method with fixed step width
- H-Bridge
  - simplified Switch Model
- Analog Digital Converter (ADC)
  - 8-channel 10bit ADC with SPI Interface
  - equivalent to MCP3008
- HiFive1 microcontroller platform



HiFive1 Platform components

- Peripherals
  - register accurate
  - functionality and timing implemented as needed
- Interconnect
  - loosely-timed router
- Wrapper
  - SystemC Wrapper containing pure C++ based ISS of RISC-V



RISC-V – Some Background

- HiFive1 is the first incarnation of the RISC-V Instruction Set Architecture
  - developed by SiFive, a company established by the creators of the RISC-V ISA
  - implements the RV32 base instruction set (I) together with the M (Integer multiplication) and C (compressed instructions) extensions
  - There are ports of the design to Xilinx Artix chips
- RISC-V ISA is open source (governed by the RISC-V foundation) and there are plenty of open- and closed-source IPs and tool chains
- Further information can be found at https://riscv.org/

RISC-V – Some Background

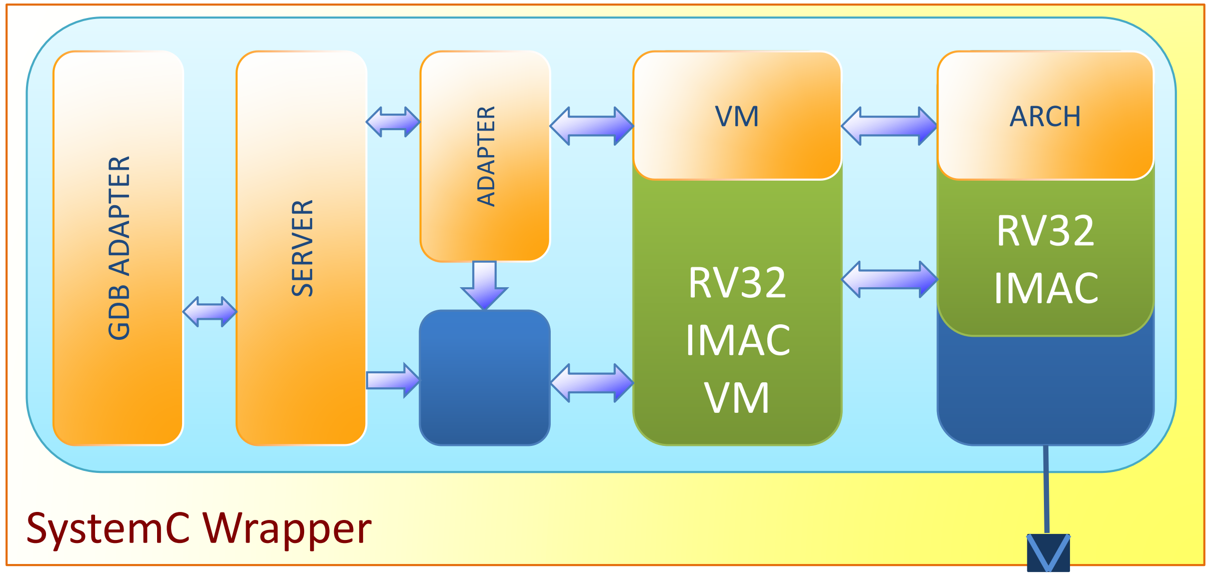
- There are several ISS available. e.g
  - Spike – the reference simulator but too slow for SW development
  - QEmu – very fast DBT based ISS but GNU license restrictions limit commercial use
  - RV8 – DBT based full-system ISS, difficult to tailor to ISA modifications, limited debugging capabilities

DBT-RISE based RISC-V ISS

- Alternative: DBT-RISE-RISCV
  - BSD licensed open-source
  - fast ISS (suitable for SW development)
  - easy to extend and tailor to ISA combinations and extensions
  - easy to integrate into SystemC simulations
- builds on DBT-RISE, an infrastructure and library for rapid implementation of DBT based instruction set simulators

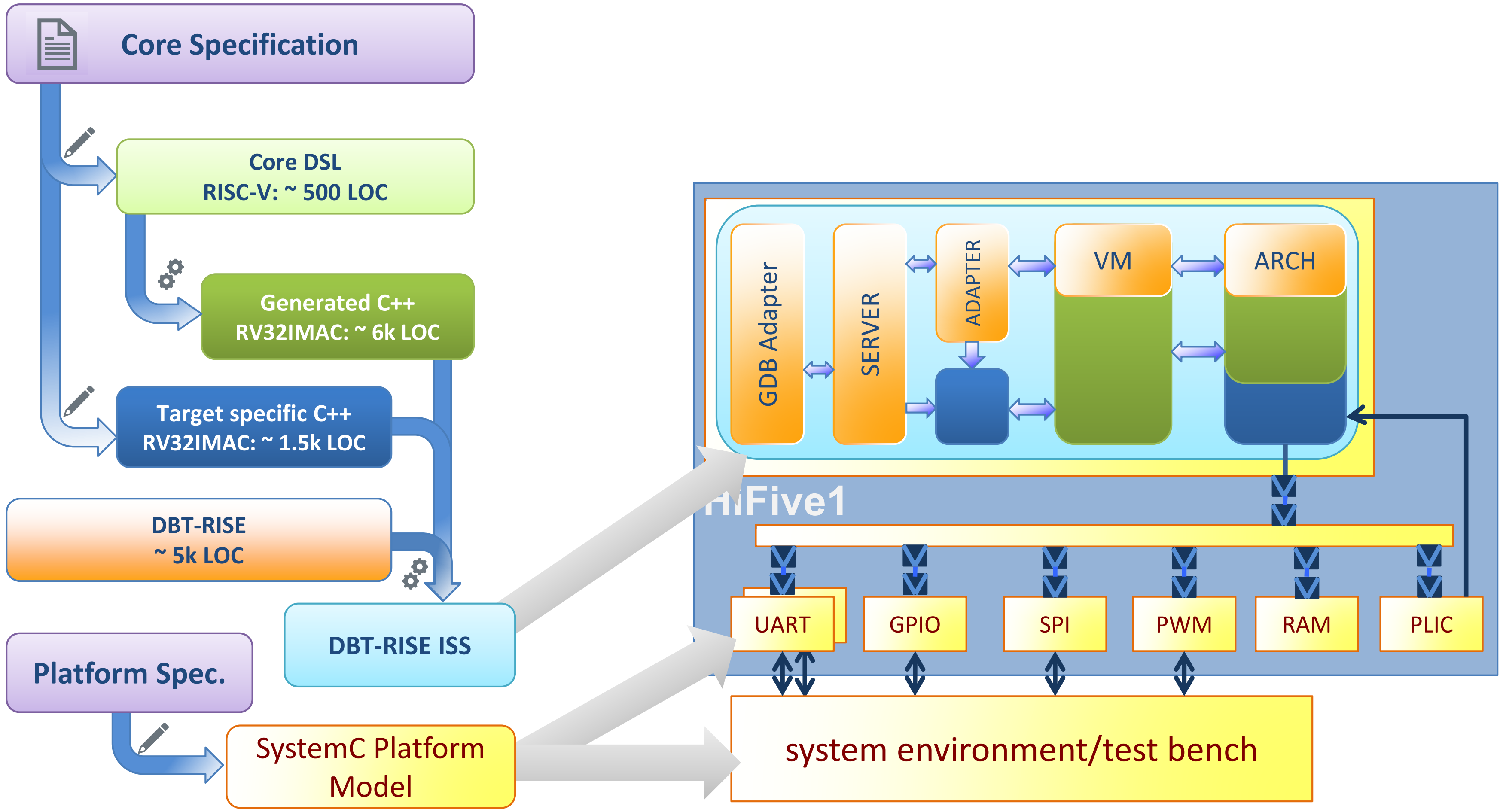
DBT-RISE-RISCV ISS

- ‘full-system’ simulator based on DBT-RISE
- uses dynamic binary translation to achieve high simulation speed
- instruction accurate with cycle estimation



- generated out of a DSL describing the machine instructions
  - fast turn-around times when changing/extending the ISA
  - well suited for HW/SW co-design & exploration

DBT-RISE based Implementation Flow



Implementation details

- SystemC Configuration and Control Interface used to configure the entire design
- Transaction tracing using SystemC Verification Library (SCV)
- Modeling blocks taken from SystemC Components library (SCC) available at https://git.minres.com/SystemC/SystemC-Components
- Register implementation generated from SystemRDL description
- Source available at https://git.minres.com/DVCon2018

Demo

Rocco Jonack

VP USE IN HARDWARE DEVELOPMENT

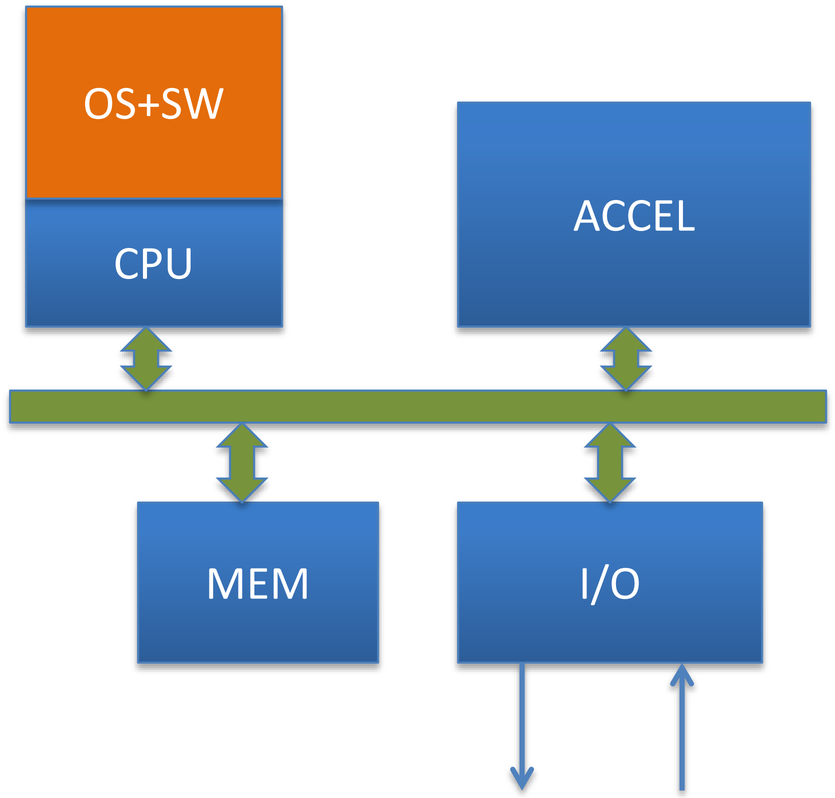
Motivation

- VP model usage for hardware design
  - Software driven testing
  - Augmentation of HW tests with realistic stimulus
  - Bridging gaps in model availability
- Different abstraction level imply different requirements
  - Mixing abstraction levels is powerful
  - Planning of reasonable use models is required

Modeling abstractions

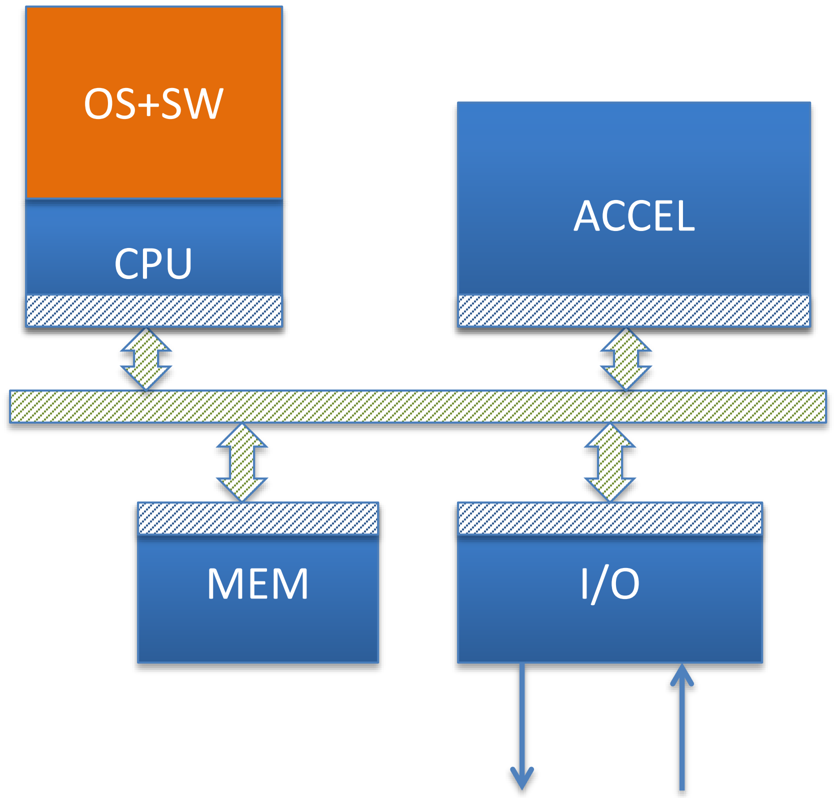
- Behavioral/untimed
  - Used for algorithm modeling
- Functional/loosely timed (LT)
  - Used for SW interaction, also called programmers view (PV)
  - virtual prototyping (VP)
- Cycle-accurate/approximately timed (AT)
  - Used for architectural modeling, also called architecture view (AV)
- Register-transfer-level
  - Link into verification domain

Functional/loosely timed (LT)



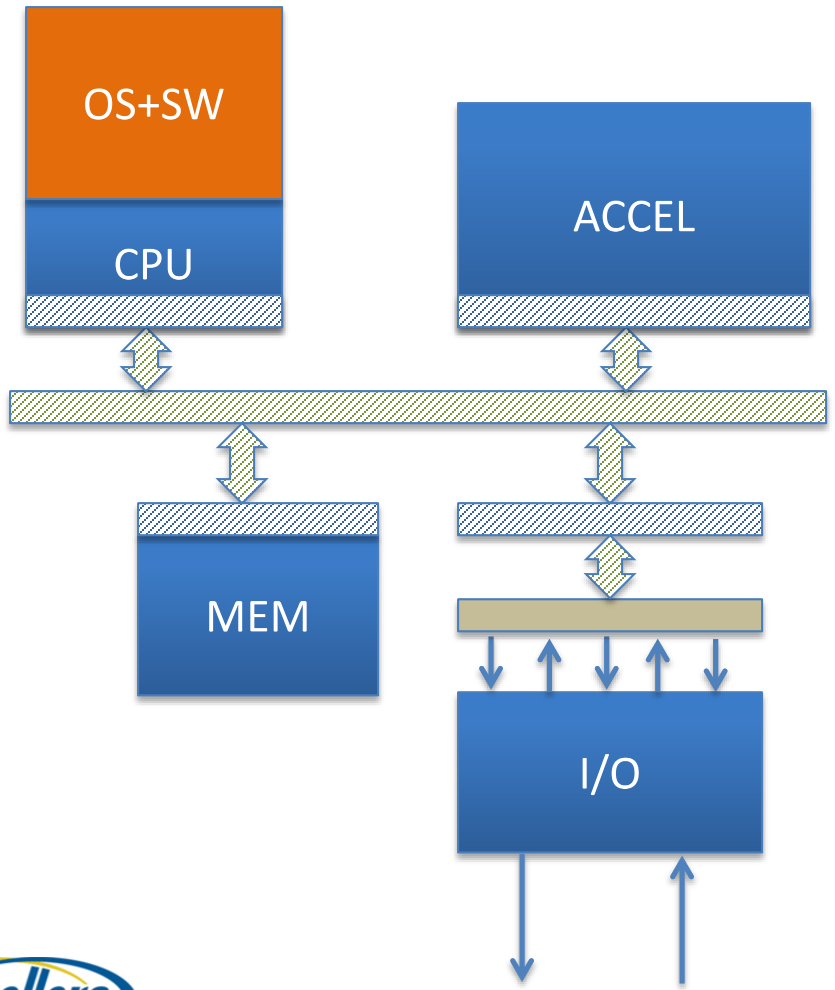
- The loosely timed model is a structural and behavioral refinement of the functional model.
- Mapping of functional blocks to HW and SW components and communication interfaces in-between based on a chosen architecture
- Subsystems can execute ‘ahead-of-time’

Cycle-accurate/approximately timed (AT)



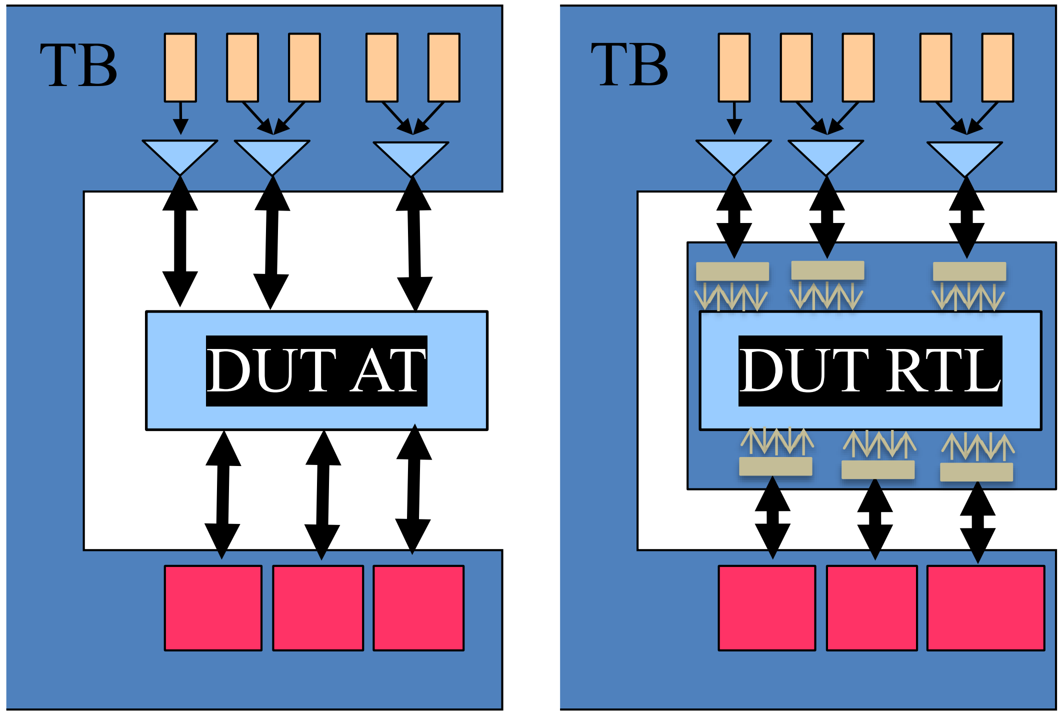
- Approximated timing on bus communication and on hardware resource access
  - Interface communication time
  - Average processing time in hardware IP
- Transactions are broken down into a number of phases corresponding much more closely to the phasing of particular hardware protocols

Pin accurate models - RTL



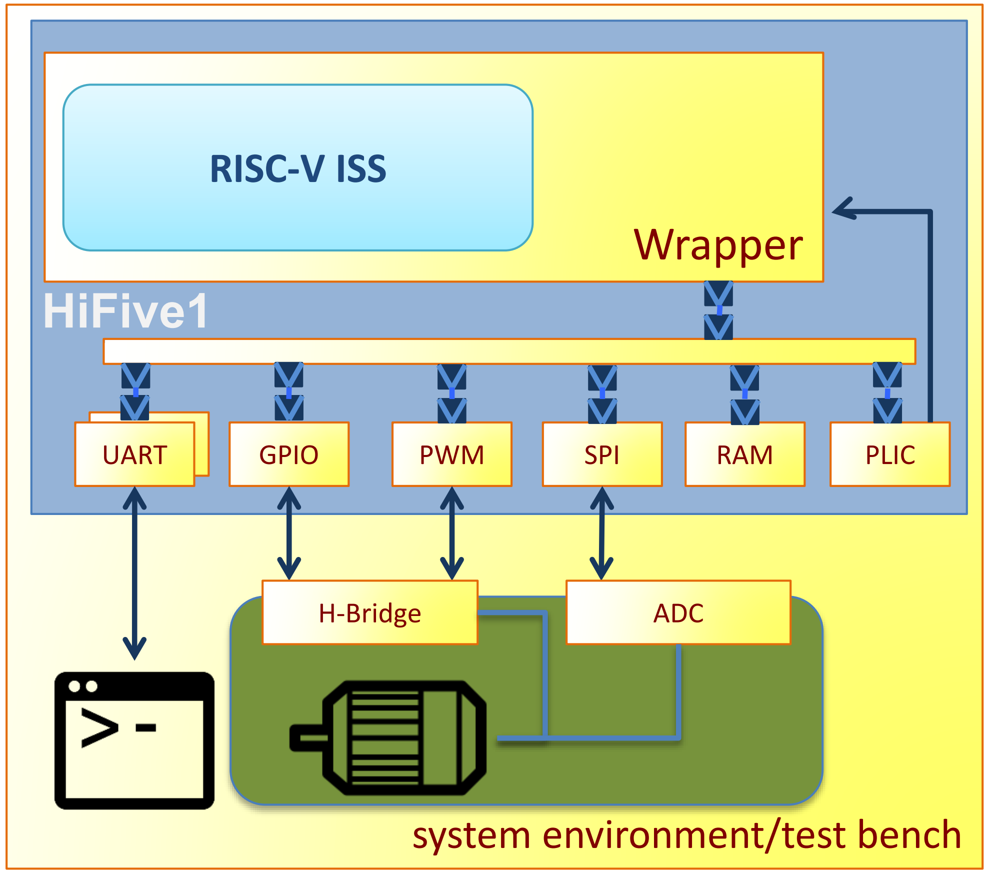
- Pin accurate in SystemC
  - Adapter between transactions and pin level
  - RTL simulators support pin level interfaces for standard types
  - Requires typically usage of simulator provided library versions and compilers
- Verification effort
  - RTL models are typically more detailed and require more attention to details
- Bridging multiple abstractions levels introduces need for interpretation

Integration environments for testing



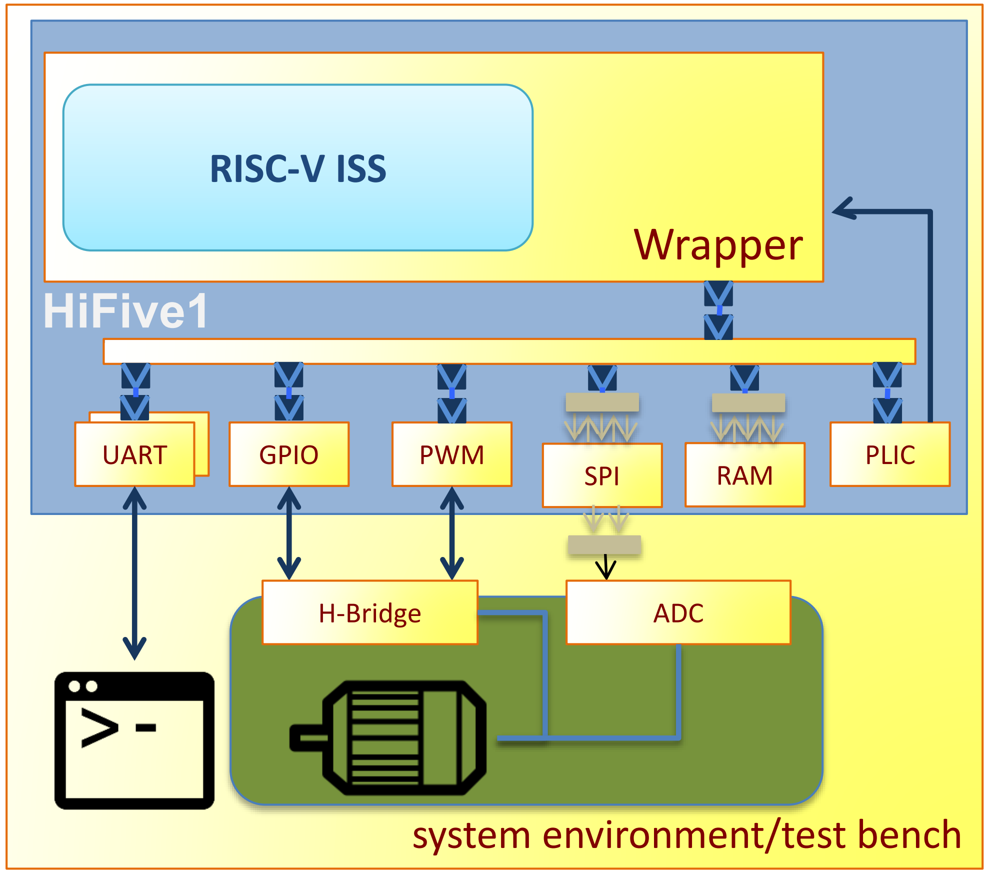
- Standalone testing environments
  - Easier analysis and debugging
    - Lighter/faster environments
- Reuse of common transactors
  - Using common components like TLM2 BFMs and memories
    - Productivity libraries
- Links to tools and open source projects
  - C based environment allows easy integration

RTL components integrated into example system I



- Replacing some components with RTL through TLM2pin adapters
  - Mix allows stepwise refinement of prototype

RTL components integrated into example system II



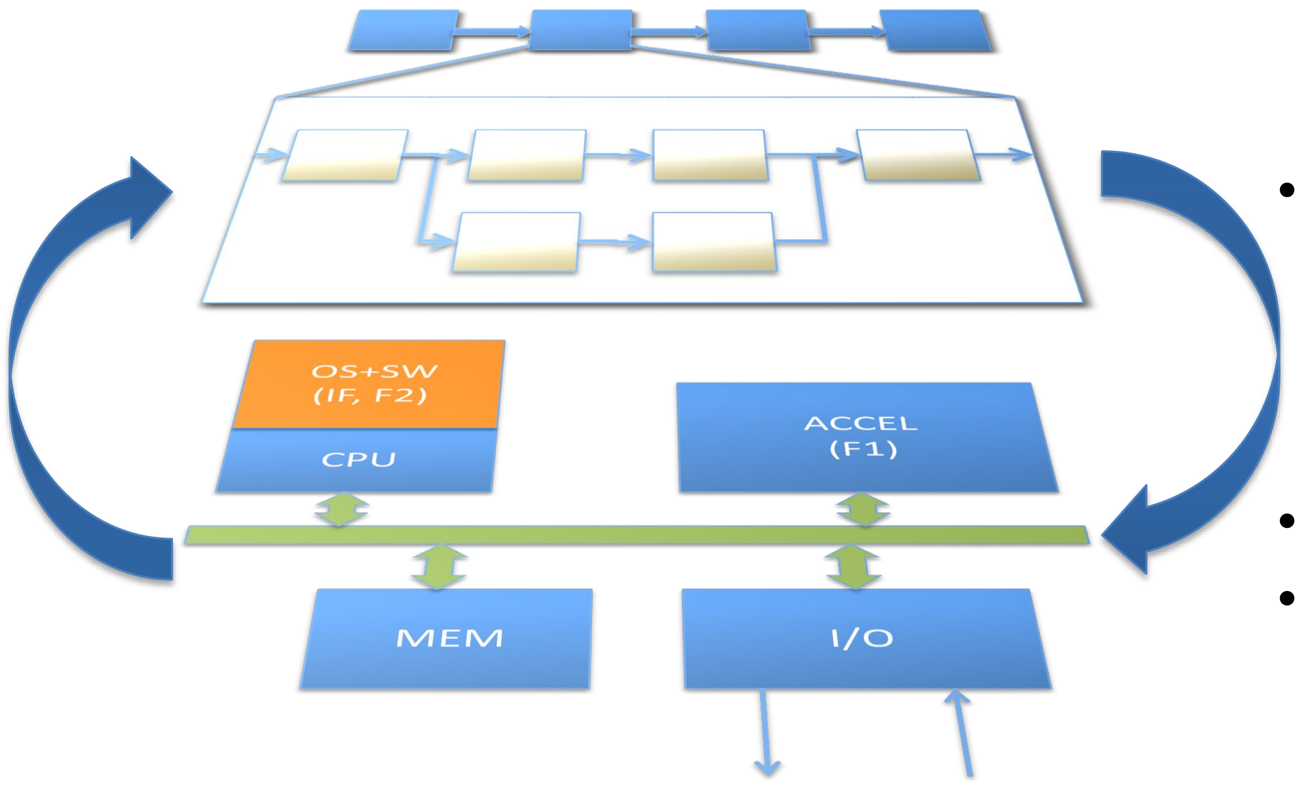
- Replacing some components with RTL through TLM2pin adapters
  - Mix allows stepwise refinement of prototype

Demo

Recommendations

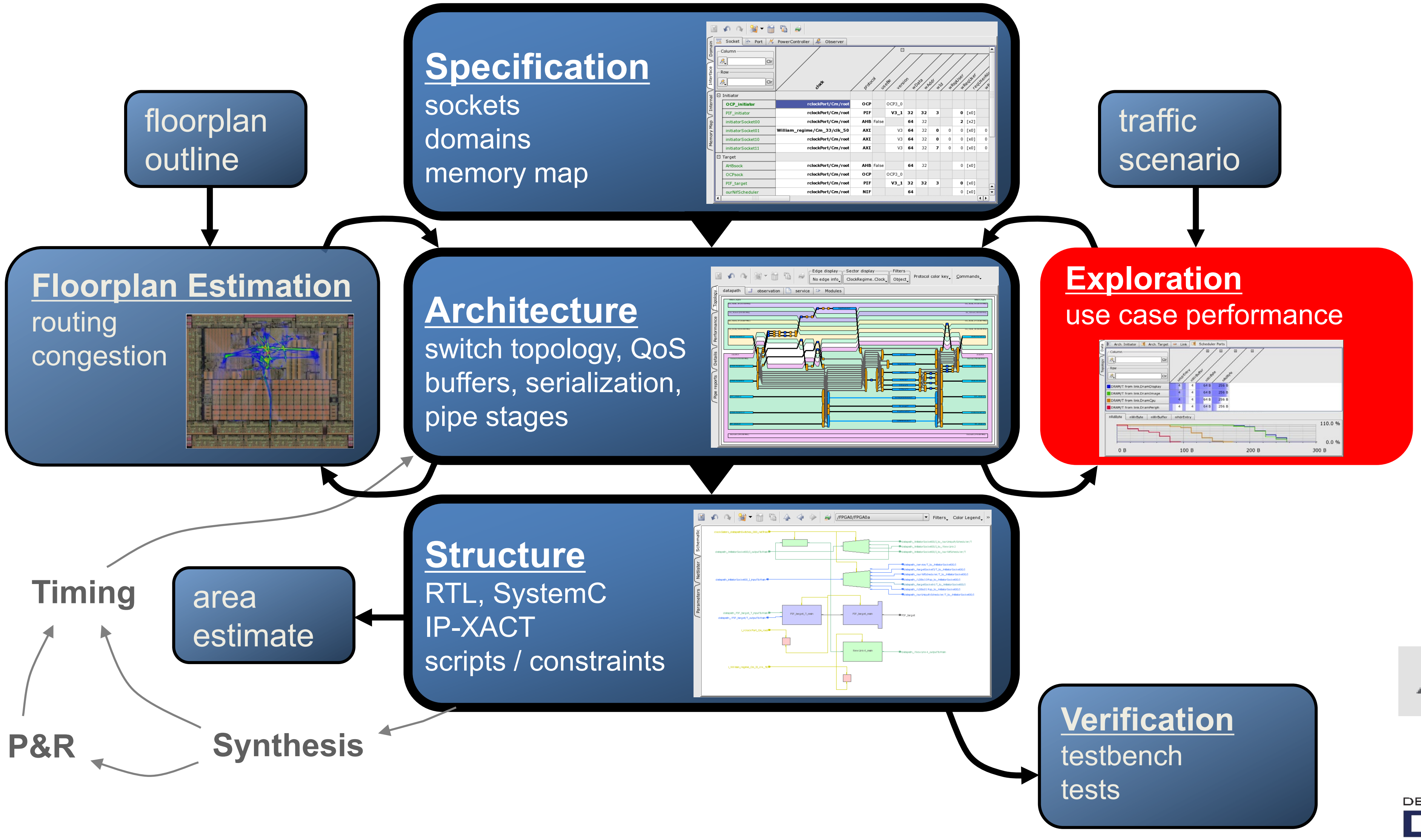
- Use common interfaces for bridging between abstraction levels
  - Preferably well known and standardized interface like AMBA, OCP
- Use libraries of common components
  - Productivity libraries decrease turnaround time
  - Tools often contain useful components but also imply specific infrastructure
- Planning and testing of components
  - Functionality requirements
  - Speed requirements
  - Debugging and analysis
- Dynamic switching between abstraction levels is very powerful

Architectural exploration & performance analysis

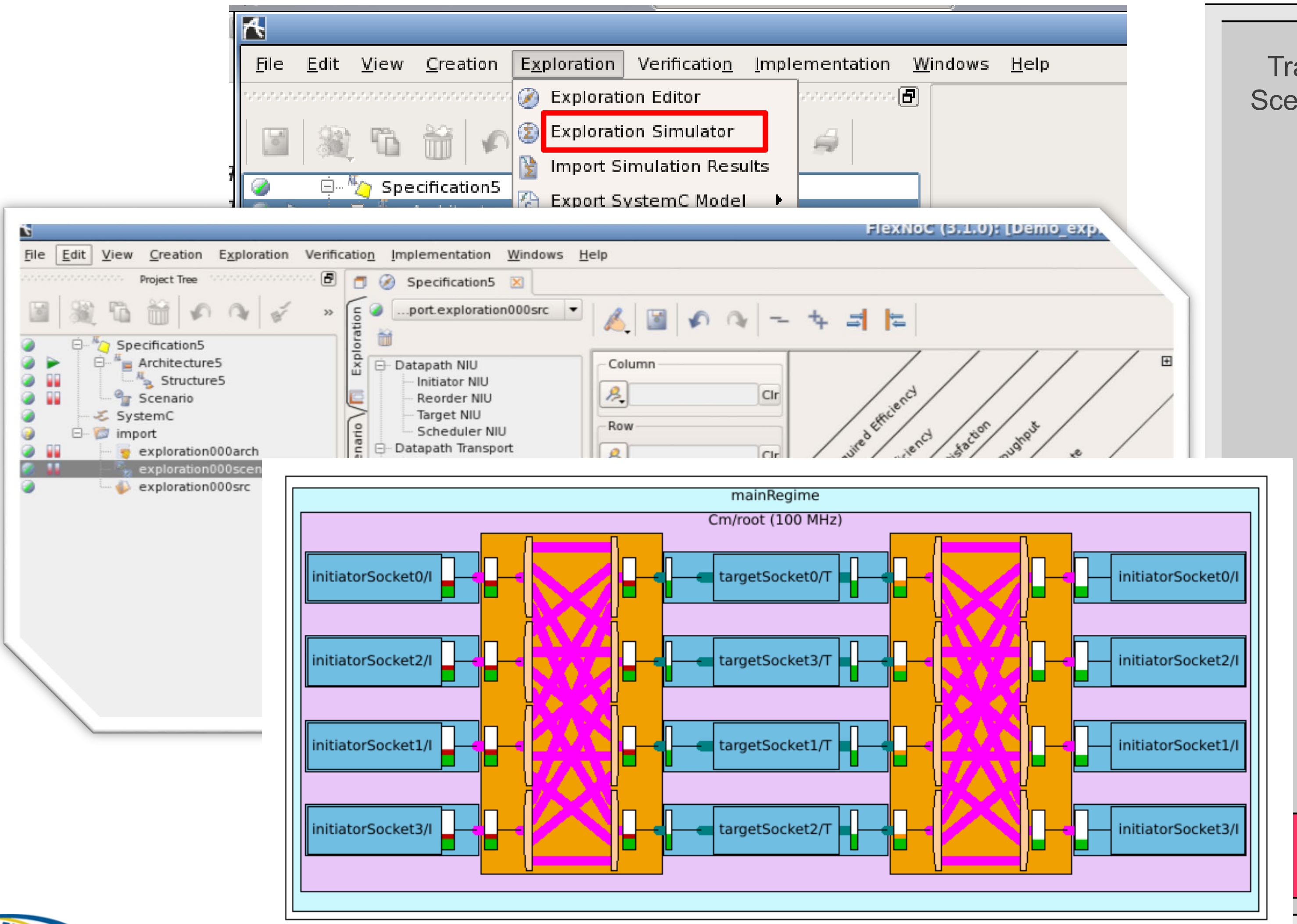


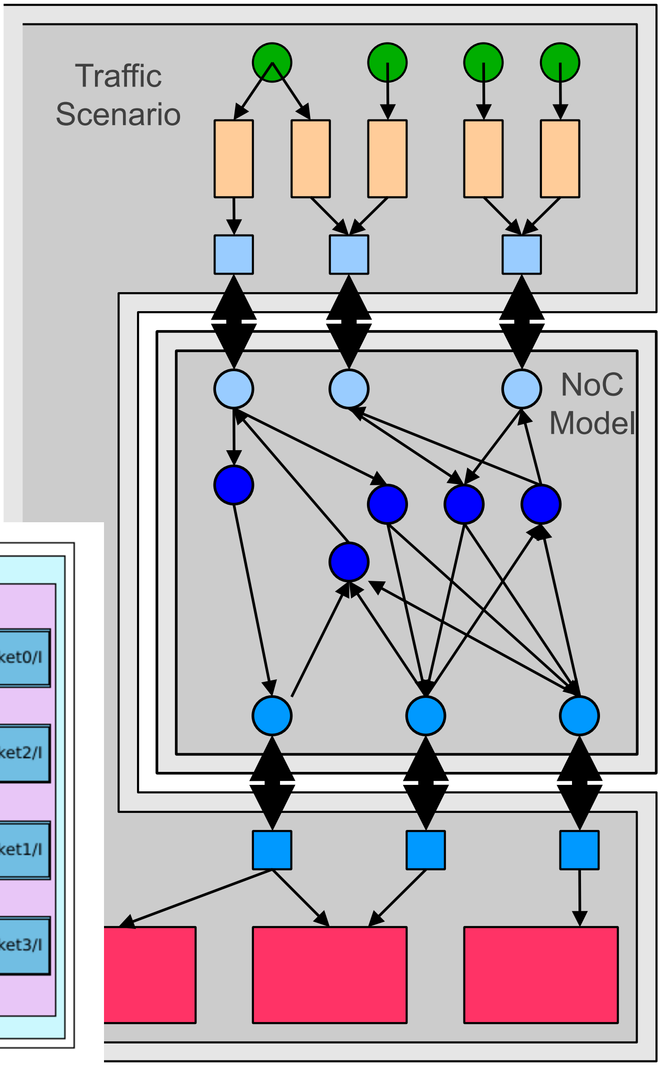
- Mapping of a behavioral model to one or more points in the architectural space consisting of different HW implementations
- Evaluation based on performance characteristics for different system architectures, such as a HW/SW split, communication system, or system components
- Typical use case for platform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

Tool Flow

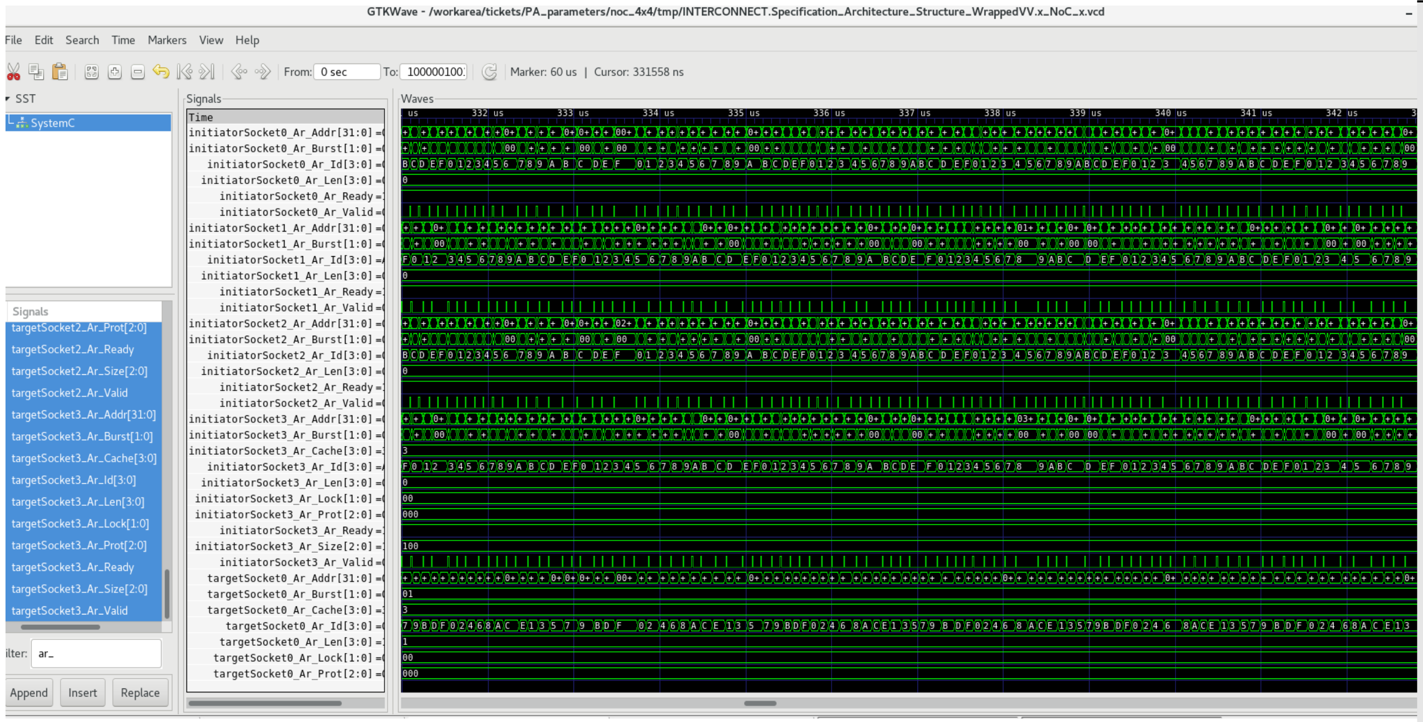

Specification
sockets
domains
memory map
floorplan outline
traffic scenario
Floorplan Estimation
routing congestion
Architecture
switch topology, QoS
buffers, serialization,
pipe stages
Exploration
use case performance
Structure
RTL, SystemC
IP-XACT
scripts / constraints
area estimate
Timing
P&R
Synthesis
Verification
testbench
tests

AT Simulation





RTL Simulation



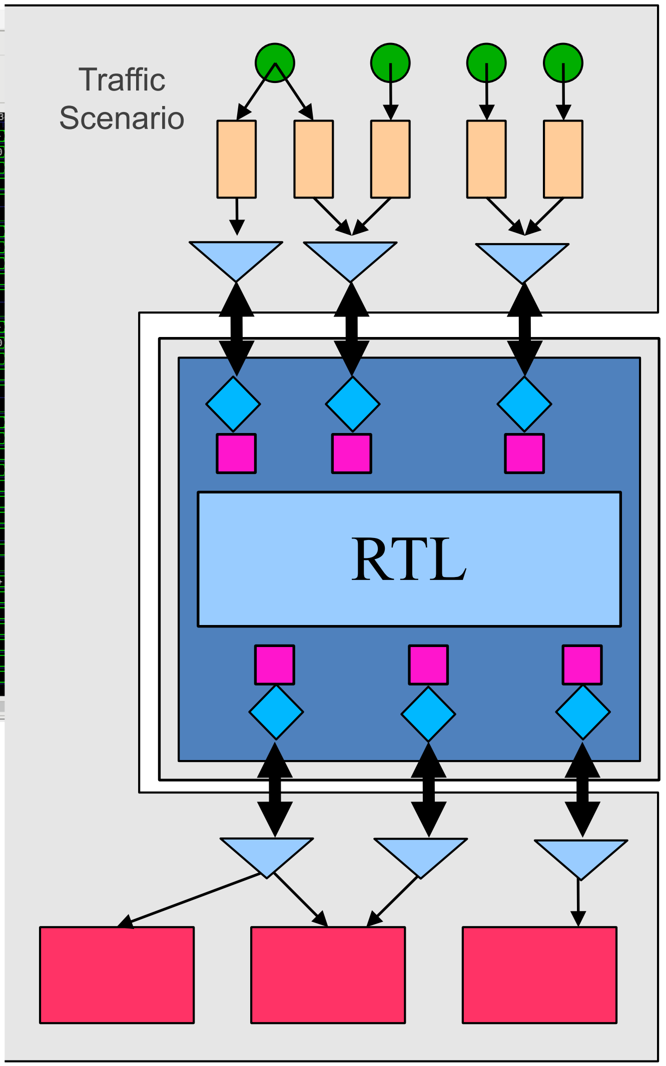
GTKWave - /workarea/tickets/PA_parameters/noc_4x4/tmp/INTERCONNECT.Specification_Architecture_Structure_WrappedVV.x_NoC_x.vcd

From: 0 sec To: 100000100 Marker: 60 us | Cursor: 331558 ns

```
Time
initiatorSocket0_Ar_Addr[31:0]
initiatorSocket0_Ar_Burst[1:0]
initiatorSocket0_Ar_Id[3:0]
initiatorSocket0_Ar_Len[3:0]
initiatorSocket0_Ar_Ready
initiatorSocket0_Ar_Valid
initiatorSocket1_Ar_Addr[31:0]
initiatorSocket1_Ar_Burst[1:0]
initiatorSocket1_Ar_Id[3:0]
initiatorSocket1_Ar_Len[3:0]
initiatorSocket1_Ar_Ready
initiatorSocket1_Ar_Valid
initiatorSocket2_Ar_Addr[31:0]
initiatorSocket2_Ar_Burst[1:0]
initiatorSocket2_Ar_Id[3:0]
initiatorSocket2_Ar_Len[3:0]
initiatorSocket2_Ar_Ready
initiatorSocket2_Ar_Valid
initiatorSocket3_Ar_Addr[31:0]
initiatorSocket3_Ar_Burst[1:0]
initiatorSocket3_Ar_Cache[3:0]
initiatorSocket3_Ar_Id[3:0]
initiatorSocket3_Ar_Len[3:0]
initiatorSocket3_Ar_Lock[1:0]
initiatorSocket3_Ar_Prot[2:0]
initiatorSocket3_Ar_Ready
initiatorSocket3_Ar_Size[2:0]
initiatorSocket3_Ar_Valid
targetSocket0_Ar_Addr[31:0]
targetSocket0_Ar_Burst[1:0]
targetSocket0_Ar_Cache[3:0]
targetSocket0_Ar_Id[3:0]
targetSocket0_Ar_Len[3:0]
targetSocket0_Ar_Lock[1:0]
targetSocket0_Ar_Prot[2:0]
```

```
targetSocket2_Ar_Prot[2:0]
targetSocket2_Ar_Ready
targetSocket2_Ar_Size[2:0]
targetSocket2_Ar_Valid
targetSocket3_Ar_Addr[31:0]
targetSocket3_Ar_Burst[1:0]
targetSocket3_Ar_Cache[3:0]
targetSocket3_Ar_Id[3:0]
targetSocket3_Ar_Len[3:0]
targetSocket3_Ar_Lock[1:0]
targetSocket3_Ar_Prot[2:0]
targetSocket3_Ar_Ready
targetSocket3_Ar_Size[2:0]
targetSocket3_Ar_Valid
```

Filter: ar_



- Running RTL model provides full signal access
- Reuse of traffic scenarios

Recommendations

- (re-)using a VP for different purposes needs careful planning
- critical components need to provide the important information and accuracy
- component in different representations need to be compatible wrt. to communication, build settings etc.

Eyck Jentzsch

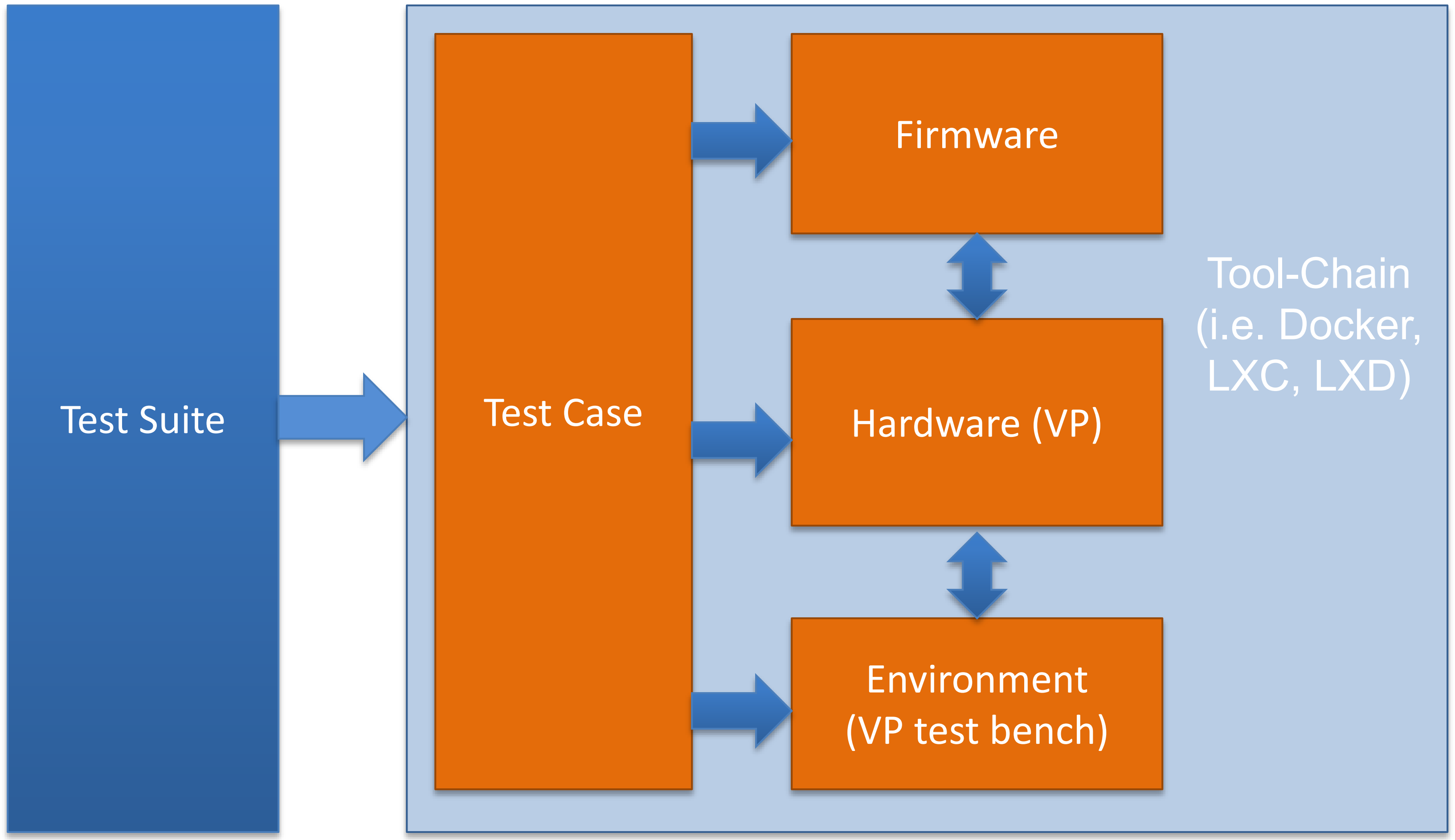
VP USE IN SOFTWARE DEVELOPMENT

Test-Lab vs. VP

Test-Lab vs. VP

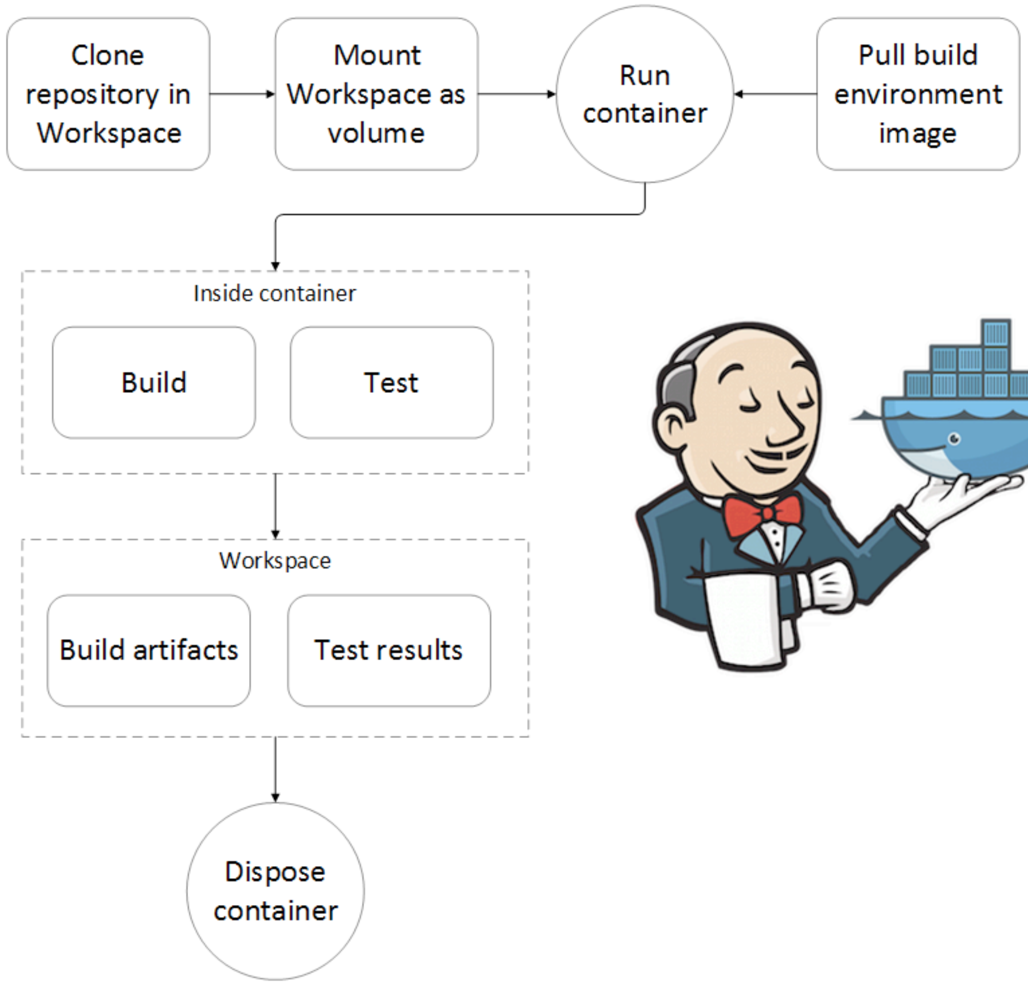
- A classical Test-Lab uses Hardware-in-the-loop (HiL)
  - HiL requires expensive Test-Equipment and space
  - Bring-up of the lab setup is usually time consuming
- Virtual Prototype is a simulation providing FW a runtime environment
  - Test-Suites don’t need much space, just a work station or compute server
  - Test-Suites can be scaled easily by adding compute power
  - Simulations are reproducible
  - internal states can easily be traced
  - Environment can be stressed easily
  - Simulation can be executed virtually “anywhere”

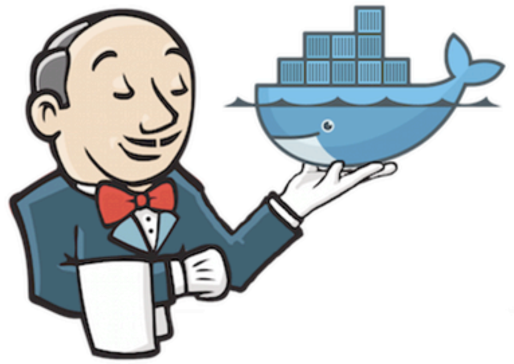
Anatomy of a VP Test Suite


Test Suite
Test Case
Firmware
Hardware (VP)
Environment (VP test bench)
Tool-Chain (i.e. Docker, LXC, LXD)

Anatomy of a VP Test Suite

- A Test-Suite works as follows:
  - Fetch the source code repositories
  - Start Container containing stable and host-independent tool-chain
  - Build Firmware
  - Build VP
  - Build Environment/test-bench
  - Run the test case(s)
  - Evaluate test case execution and collect results





Source: https://stefanprodan.com/2016/continuous-integration-with-jenkins-and-disposable-containers/

Integration-Aspects

- Tool-Chain (i.e. Third-Party libs, Compiler version) can be switched back and forth (i.e. by using different container images)
- Run different FW-Versions on same HW (VP)
- Run same FW-Version on different HW (VPs)
- Failing tests are always reproducible and (i.e. in a Debug-Session on developers computer)
- Internal states can be traced (VCD, Trace- and Log-Files)

Short Demo

Stressing the Environment

- The VP-Environment is a just a model as well
- A virtual environment can stress your system beyond reality
- Loads can be applied arbitrarily
- Stress the environment (VP-)Model to push FW into corner cases
- Whenever a random scenario causes a failure, the scenario can be reproduced

Code-Coverage

- An embedded system doesn’t give users much insight into the system (e.g. CPU states, program counters)
- therefore coverage collection is difficult
- VP-Simulations support Code- and (in some cases) Branch-Coverage collection
- Contributors (Test-Cases) to overall Code-Coverage can be identified to streamline testing

Continuous Integration

- A VP-Test-Suite can easily be combined with Git/Gerrit
  - Run selected VP-Tests upon every FW-Code-Commit e.g. highest contributors to code coverage
  - Run full VP-Test-Suite over night
  - Generate FW-Releases e.g. ‘nightlies’
- Ensures to have stable working mainline
- Combined with a FW test plan it allows to monitor implementation progress

Benefits of using VP based SW development

- Each SW change is tested before propagating to the main line of development
- Allows close monitoring of the eSW development progress
- effort of VP development pays off esp. when consistently used in SW development even after the availability of the HW
- Maximum benefits in situations where one software system addresses multiple hardware variants in different system contexts

Questions