Python and SystemC

March 17, 2021

Rocco Jonack, Arteris IP

Eyck Jentzsch, MINRES Technologies GmbH

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Arteris IP** and **MINRES Technologies GmbH** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on the standard.

Motivation

- MINRES focuses on VPs for eSW development and architectural exploration
- Development often in parallel with HW development
  - what if scenarios are important for architecture decisions
  - Platform definition is not fixed
- VP based embedded software development for large systems requires the use of partial and subsystems to get reasonable simulation speed and runtime
- Flexibility in Reconfiguration is key for efficient model development

Agenda

- PySysC
- SCC/VP-VIBES productivity library
- Python based performance analysis

Addressing Flexibility

- Complex configurations system
  - Reading and interpreting a configuration file
  - Done in several tools by parsing XML or JSON files
- Code generation
  - Based on some configuration input generated glue logic
- Scripting languages as frontend
  - There are tools which provide such solutions
  - Allows integration of different functionality
  - Limited by scripting API

Scripting Solutions for SystemC

- There are several existing integration into scripting languages
- As part of commercial tools based on TCL/TK, Python
- Open-source solutions
  - SoCRockets Universal Scripting Interface (USI)
  - GreenSoCs GreenScript
  - SystemPy
  - Kosim

SystemC and Python

- We opted for an interpretation "frontend" based on Python
- Python is well-known and existing libraries can be reused
- Beyond support for structural construction, simulation control and dynamic model parametrization should be supported
- Existing Python integrations require preparation work
  - Definitions of API into libraries which have been compiled
  - Quite often modification of the libraries to fulfill requirements implied by the interpreter
- Hence there are no integrations for SCV or CCI available

PySysC

- CERN developed several tools for the analysis of LHC generated data
  - CINT: home-grown Python bindings piggy-backed on C++ reflection for serialization and interactivity
  - CLING: C++ interpreter (https://root.cern/cling)
  - CPPYY: Cling-based Python-C++ bindings
- Cppyy can be leveraged for any library
- This is the basis for the PySysC module

PySysC Advantages

- No preparation of libraries to be integrated
- No need to have the sources of the code, even 3rd party binary only libraries can be used
- Allows introspection of the interfaces and thus dynamic generation
- If Python is not sufficient JIT allows to compile on-the-fly generated C++ code

PySysC Example

- Instantiation of a module
- Instantiation of a templated module
- Named signal connection
- TLM2.0 socket connection
- Simulation run

```
from cppyy import gbl as cpp
from cppyy.gbl import sc_core
from pysysc.structural import Connection, Signal, Module, Simulation
# loading required libraries
...
# instantiating modules
clk_gen = Module(cpp.ClkGen).create("clk_gen")                    ## (1)
initiator = Module(cpp.Initiator).create("initiator")
memories = [Module(cpp.Memory).create(name)
                  for name in ["mem0", "mem1", "mem2", "mem3"]]
router = Module(cpp.Router[4]).create("router")                   ## (2)
# creating connections
clk = Signal("clk")
        .src(clk_gen.clk_o)
        .sink(initiator.clk_i)
        .sink(router.clk_i)                                       ## (3)
[clk.sink(m.clk_i) for m in memories]
Connection()
        .src(initiator.socket)
        .sink(router.target_socket)                               ## (4)
[Connection()
        .src(router.initiator_socket.at(idx))
        .sink(m.socket)
        for idx,m in enumerate(memories)]
# run simulation
sc_core.sc_start()                                                ## (5)
```

accellera SYSTEMS INITIATIVE™

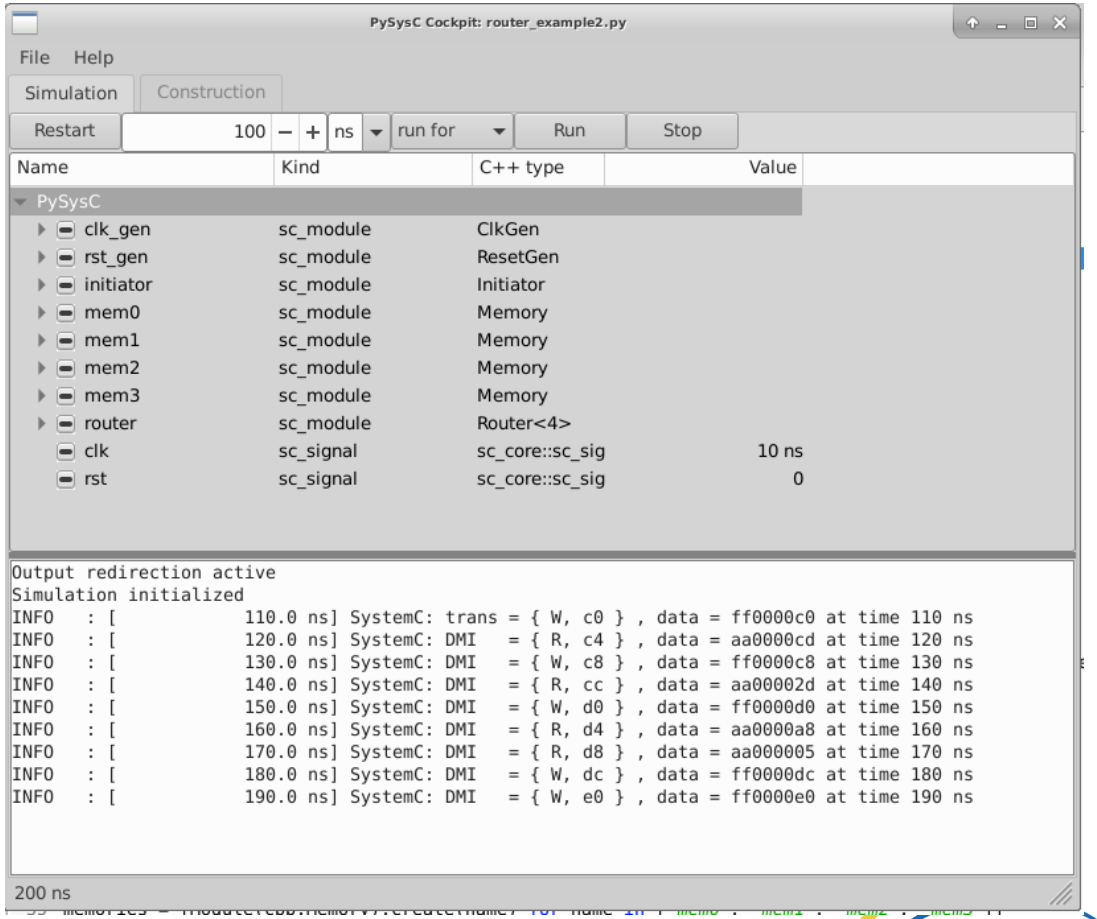
SYSTEMC™ EVOLUTION FIKA

Status

- PySysC is available as module via git
  - The Python module: https://github.com/accellera-official/PySysC
  - The examples: https://git.minres.com/SystemC/PySysC-SC
- Development is work in progress
- Will be used as a basic building block of the BMBF funded project ”RAVEN: Acceleration of Virtual Hardware/Software Development Platforms by Reconfigurable Logic”

Advantages of Python Usage

- Due to broad availability of Python integrations plenty of libraries can be used and combined
  - Computational models using numpy/scipy etc.
  - UIs and cockpits using GTK, wxWidgets or Qt

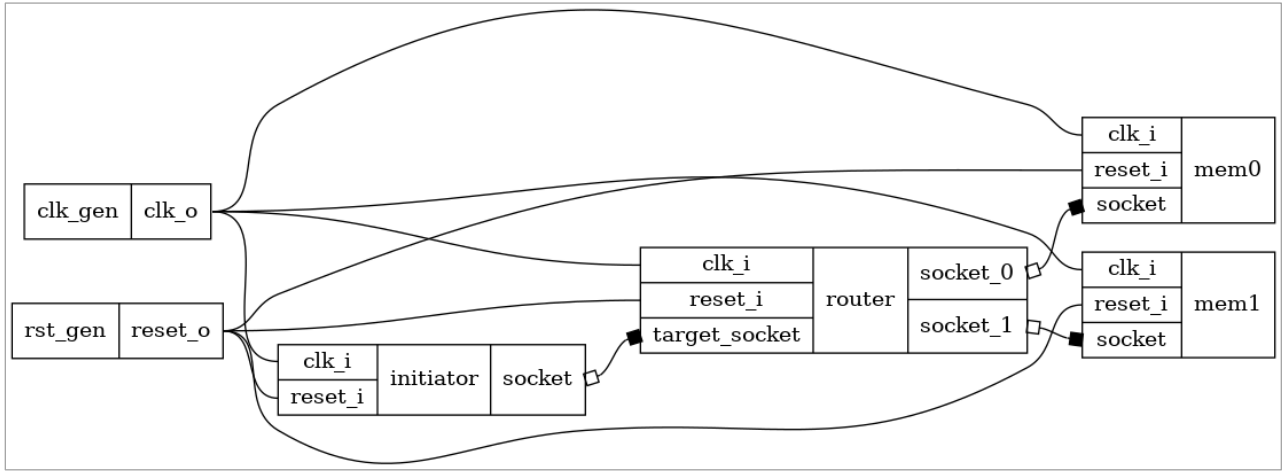


accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

Advantages of Python Usage

- Pythonization layer allows for advanced features and functionalities:
  - insert signal type converters
  - Insert transaction trace recorder e.g. from SCC
  - Extract connectivity information



accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

Demo

PySysC Module: definition

```
# defining a toplevel class
class TopModule(cpp.scc.PyScModule):
    def __init__(self, name):
        super().__init__(self, name)
        ##########################################################################
        # instantiate
        ##########################################################################
        self.clk_gen = Module(cpp.ClkGen).create("clk_gen")
        ...
        self.memories = [Module(cpp.Memory).create("mem%d"%idx) for idx in range(0,num_of_mem)]
        self.router = Module(cpp.Router[num_of_mem]).create("router")
        ##########################################################################
        # connect them
        ##########################################################################
        self.clk = Signal("clk").src(self.clk_gen.clk_o).sink(self.initiator.clk_i).sink(self.router.clk_i)
        ...
        [Connection().src(self.router.initiator_socket.at(idx)).sink(m.socket) for idx,m in enumerate(self.memories)]

    def EndOfElaboration(self):
        print("Elaboration finished")

    def StartOfSimulation(self):
        print("Simulation started")

    def EndOfSimulation(self):
        print("Simulation finished")
```

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

PySysC Module: use

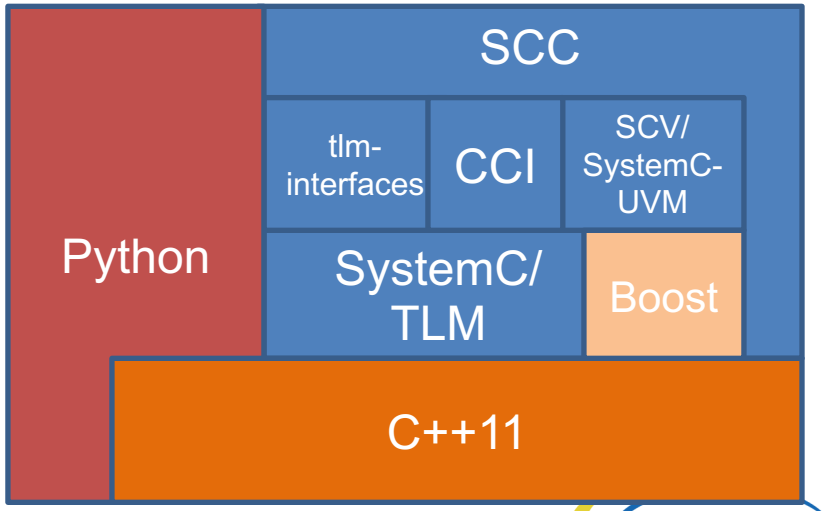
```
from cppyy import gbl as cpp
from cppyy.gbl import sc_core
from pysysc.structural import Connection, Signal, Module, Simulation
################################################################################
# setup  and load
################################################################################
logging.basicConfig(level=logging.DEBUG)
################################################################################
...
################################################################################
# configure
################################################################################
Simulation.setup(logging.root.level)
################################################################################
# instantiate
################################################################################
from modules import TopModule
dut = Module(TopModule).create("dut")
################################################################################
# run if it is standalone
################################################################################
if __name__ == "__main__":
    Simulation.configure(enable_vcd=False)
    Simulation.run()
    logging.debug("Done")
```

Agenda

- PySysC
- SCC/VP-VIBES productivity library
- Python based performance analysis

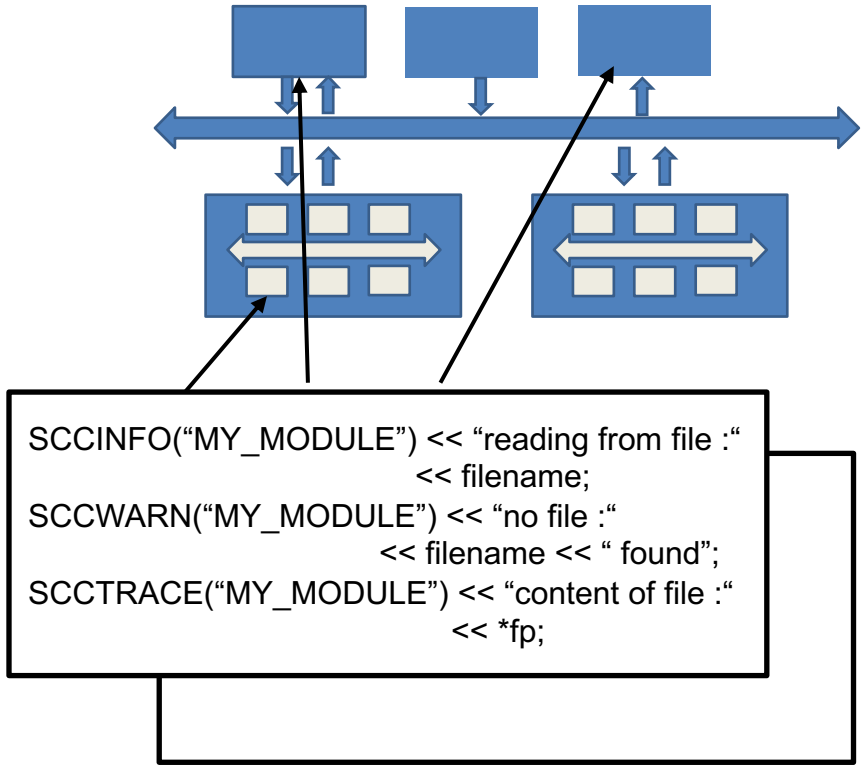
C++ class libraries for modeling

- SystemC class library on top of C++
  - Structural description elements
  - Data types
  - Event driven simulation kernel
- TLM2 based modeling 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, logging, infrastructure
  - Examples of libraries
    - Proprietary: Synopsys SCML, Intel ISCTLM, ASTC Genesis
    - Open Source: Greensocs Greenlib, MINRES SC Components



SCC Productivity Library

- Common components use common elements
  - Registers, Memories, etc.
  - Efficient for stubs or preliminary implementations
- Common infrastructure tasks to be modeled on top of common classes
  - Logging
  - Tracing
  - Parametrization
  - Domain handling (Clock, reset, power)

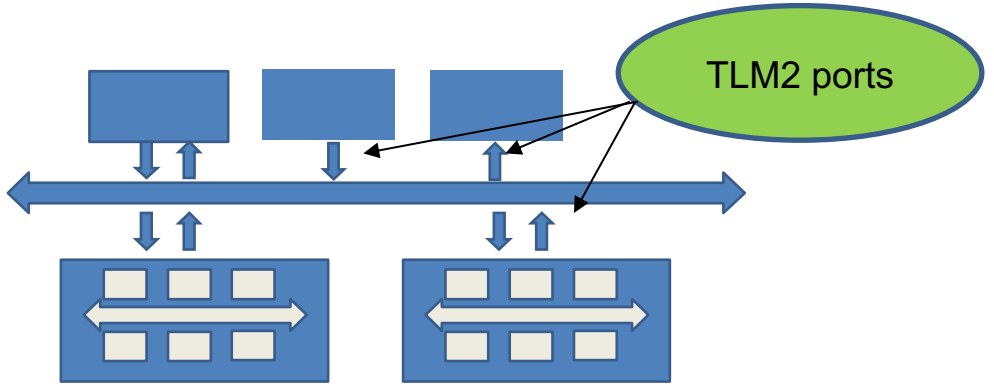


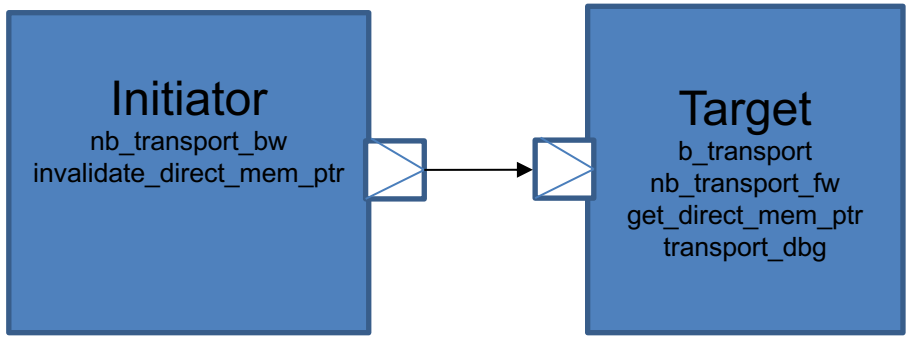
accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

SCC: TLM2 based models

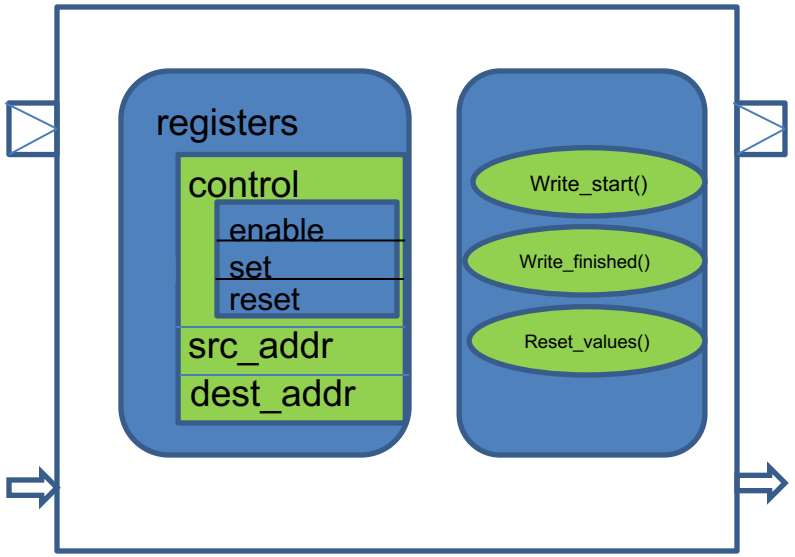
- TLM2 ports as standardized interfaces between modules
  - Allows for reuse of components
- Modeling detail depends on requirements
  - Generic TLM2
  - AMBA extensions
- Enable abstraction level flexibility
- Provides infrastructure for modeling
- LT support (DMI, temporal decoupling, debug support)
- Flexible logging
- Tracing including transaction level tracing





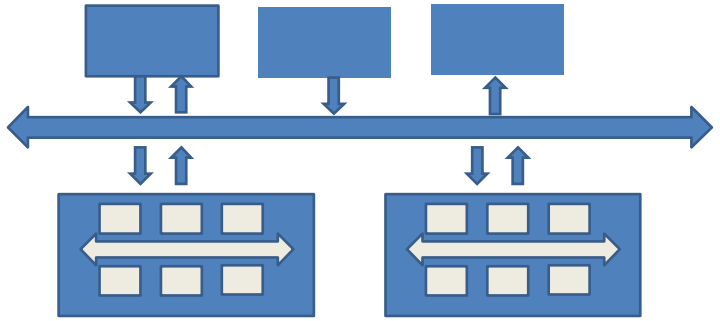
SCC: Register modeling

- 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

- Using meta data is often useful
  - Data visualization
  - Data exchange
  - Python allow flexible meta data reading, generation and writing
- IPXACT as XML based schema for IP descriptions
- Other examples for meta data representations are RDL, RAL, sysML
- IP specific configuration data



```
<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>
```

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

Improvement of standards

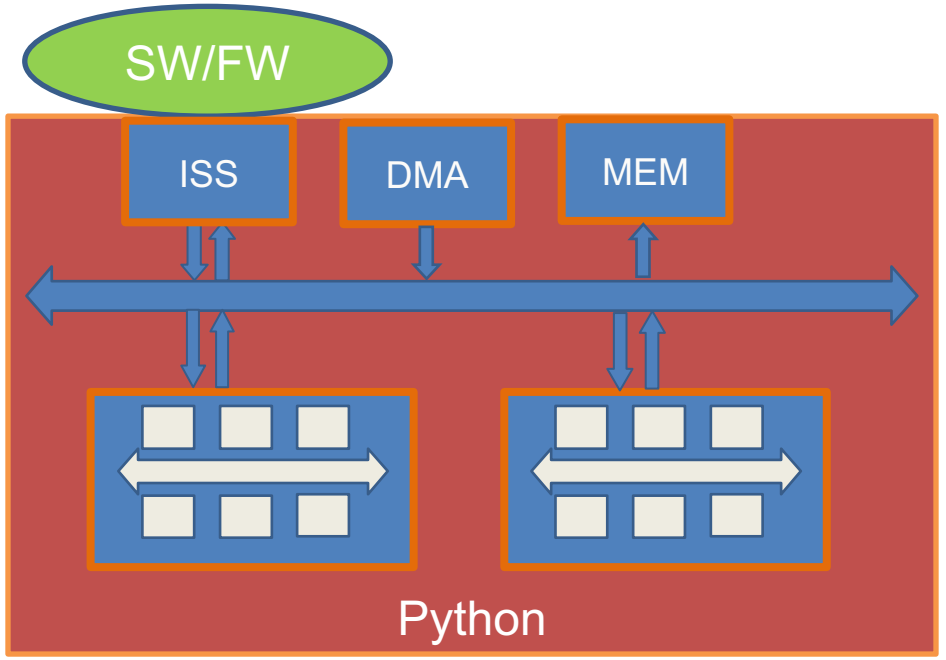
- TLM2 provides a generic transport mechanism and extension mechanisms
  - Specific protocols are not described
    - AMBA, PCI, MIPI are left to implementer
- Productivity libraries are not standardized
- More support for common infrastructure tasks
  - Tracing
  - Profiling
  - Parallelization

VP-VIBES Project

- Create joint VP Building block system (aka VP LEGO)
- RISC-V Virtual Platform Builder for Embedded Systems (RISC-V VIBES)
  - Contributions from industry and academia
    - DBT-RISE RISC-V: Industry-ready ISS for SW Development (MINRES)
    - ETISS: Concept Level ISS with Plugin-mechanism for early Performance, Power, and Soft Error Resilience analysis (TU Munich)
  - SCC: Productivity Library for fast SoC prototyping
  - VP-Per: Component Library of ready-to-use RISC-V Peripherals
  - Demonstrator VPs of RISC-V SoCs (SiFives HiFive1 and more in the making)
- Fully Open Source@ https://github.com/VP-Vibes

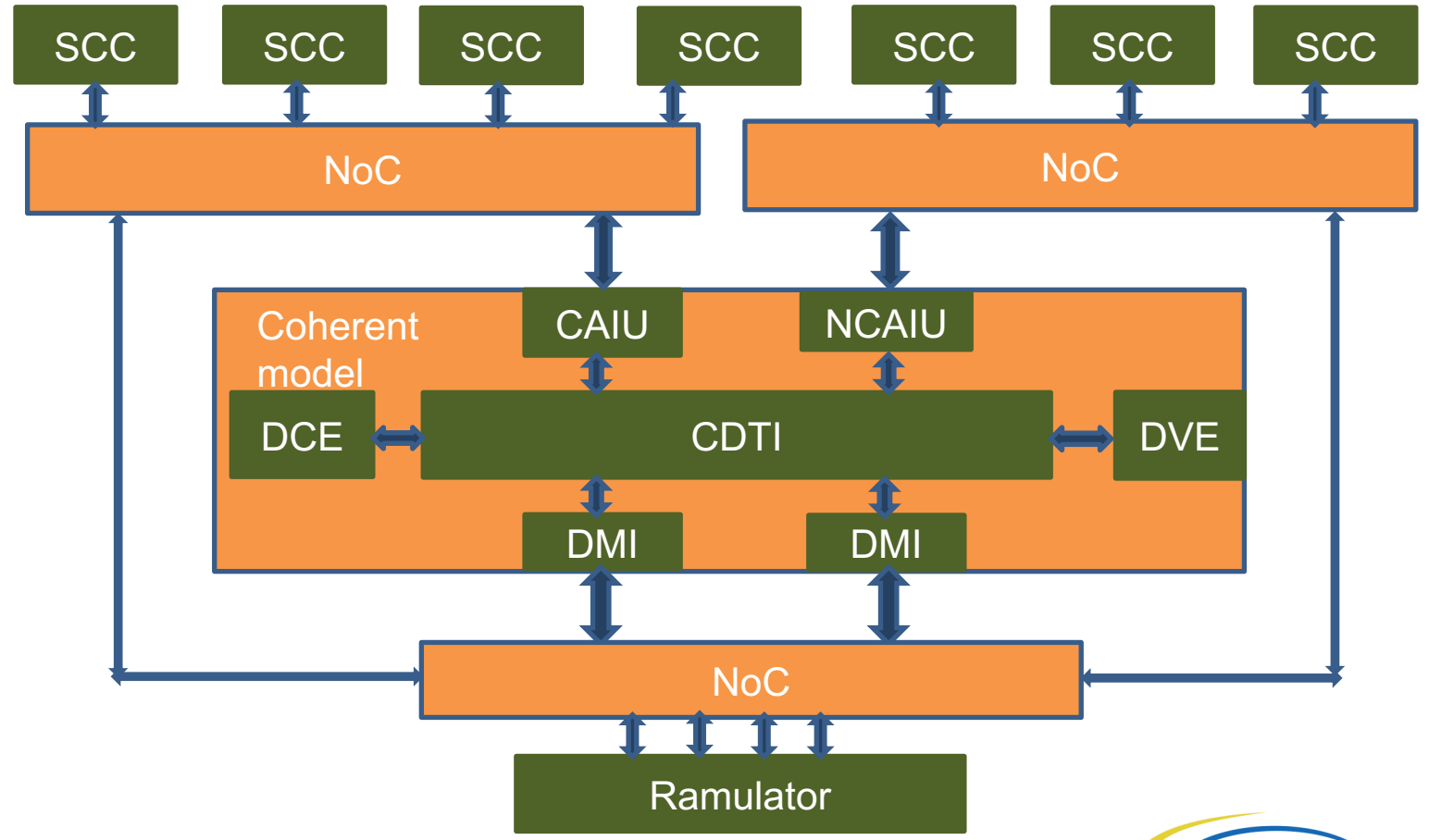
Modeling systems

- Core functionality provided by libraries
  - Infrastructure
  - ISS
  - Peripherals
  - Interconnects
- Python acts as glue between building blocks
- Setting up a VP becomes like LEGO®
  - Relying on Accellera standards
  - Using best practices and libraries



Modeling interconnect

- Interconnects are performance critical
  - Span distance, clock and power domains
  - Perform necessary conversions
  - Enable caching
  - QoS, redundancy, security…
- Using models from IP provider minimizes risk and increases performance
  - Arteris provides a variety of interconnect components



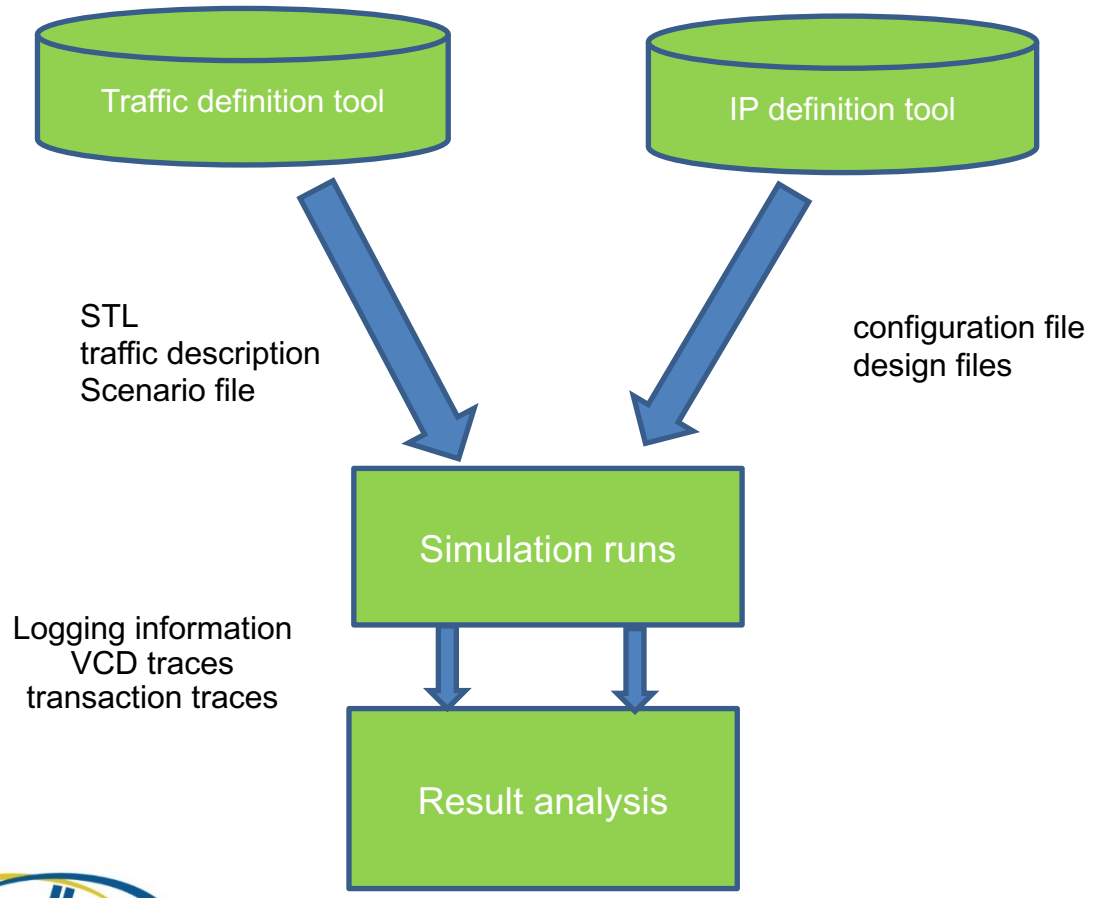
accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

Agenda

- PySysC
- SCC/VP-VIBES productivity library
- Python based performance analysis

Typical performance analysis flow



- Goal is the result analysis
- Uses :
  - Output from simulations
    - VCD, SCV traces
  - Information about IPs
    - XML, JSON
  - Information about traffic description
    - ATP, STL

accellera SYSTEMS INITIATIVE™

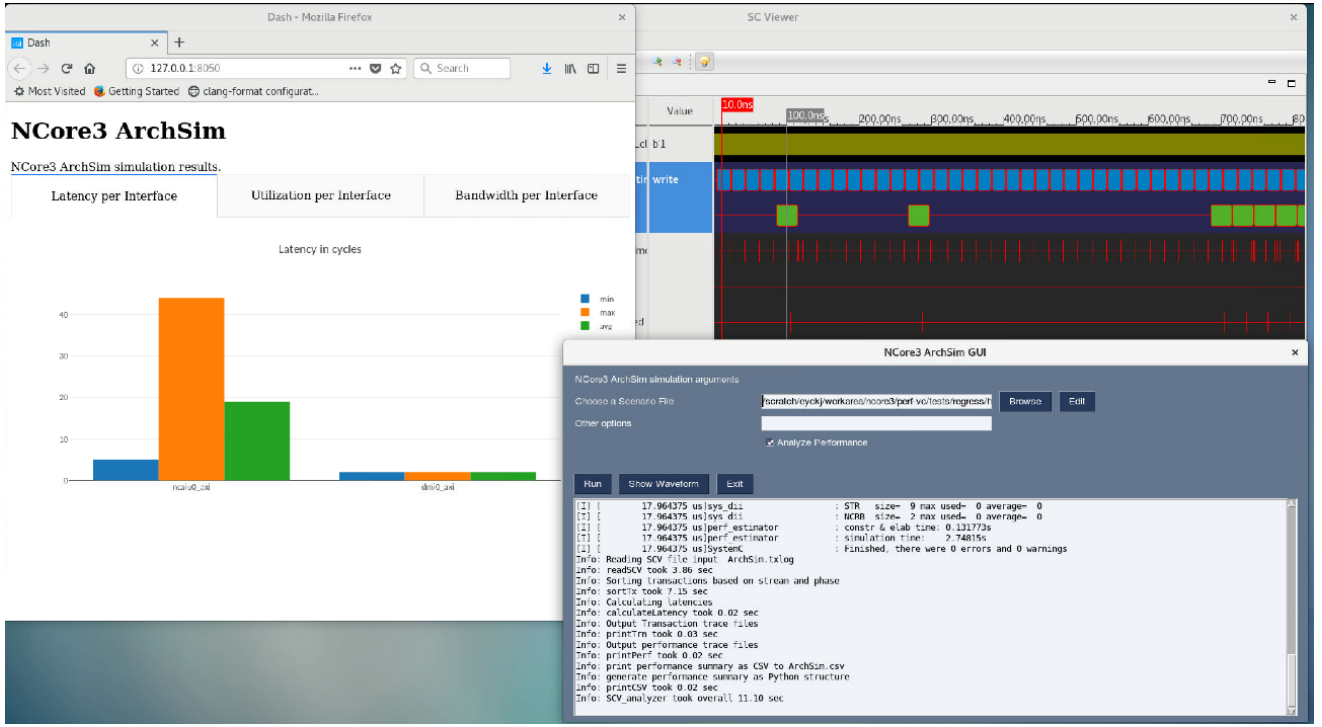
SYSTEMC™ EVOLUTION FIKA

Performance Analysis Requirements

- Analyze key performance indicators and investigating bottlenecks
  - Definition of KPI is design specific
  - Common indicators exist
- Reading efficiently well formatted output from simulations and design information
  - Building up on industry standard formats
  - JSON, XML, VCD, SCV
- Graphical components to allow visualization
  - Visualization of values and transactions over time
  - Flexible viewing techniques like charts as bar, pie, line, heatmaps
  - Searchable table views
  - Use of well know visualization frameworks
- Textual components to allow export for analysis in regression type result data bases
  - Simple text files for postprocessing
- Open for user extensions
- Translation utilities to tie into commercial tools

Analysis example

- Python libraries allow
  - Result visualization using dash
  - Simple dedicated simulation control
- Waveform tracing tool (SCViewer)
  - Visualizes SCV and VCD
- High performance database handler



accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA

Questions

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION FIKA