2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

A Framework for the Execution of Python Tests in SystemC and Specman Testbenches

Christoph Tietz[1], Sebastian Stieber[2], Najdet Charaf[3], Diana Göhringer[3]

1) Bosch Sensortec GmbH 2) StZ System-Level-Modellierung 3) Technische Universität Dresden, Chair of Adaptive Dynamic Systems

BOSCH

Steinbeis-Transferzentrum System-Level-Modellierung und Integration von MEMS Sensorsystemen

TECHNISCHE UNIVERSITÄT DRESDEN

accellera SYSTEMS INITIATIVE

Outline

- Motivation
- Overview of the proposed Framework
- Python Test – SystemC Model API in Detail
- Python Test – Specman API in Detail
- Application results
- Summary

Motivation

- Increasing requirements of ASIC designs
  - Handle complexity and project risk with HW/SW co-design
- Demanding time-to-market goals
  - HW and SW must be developed in parallel
  - Different development teams, platforms and tools
  - Reuse between domains necessary to reach time-to-market goals

Motivation

Software Development

- Testing with a virtual prototype (e.g. SystemC)
- Model with a higher abstraction level for early availability

Hardware Development

- RTL Simulation e.g. in Specman testbenches

Framework targets

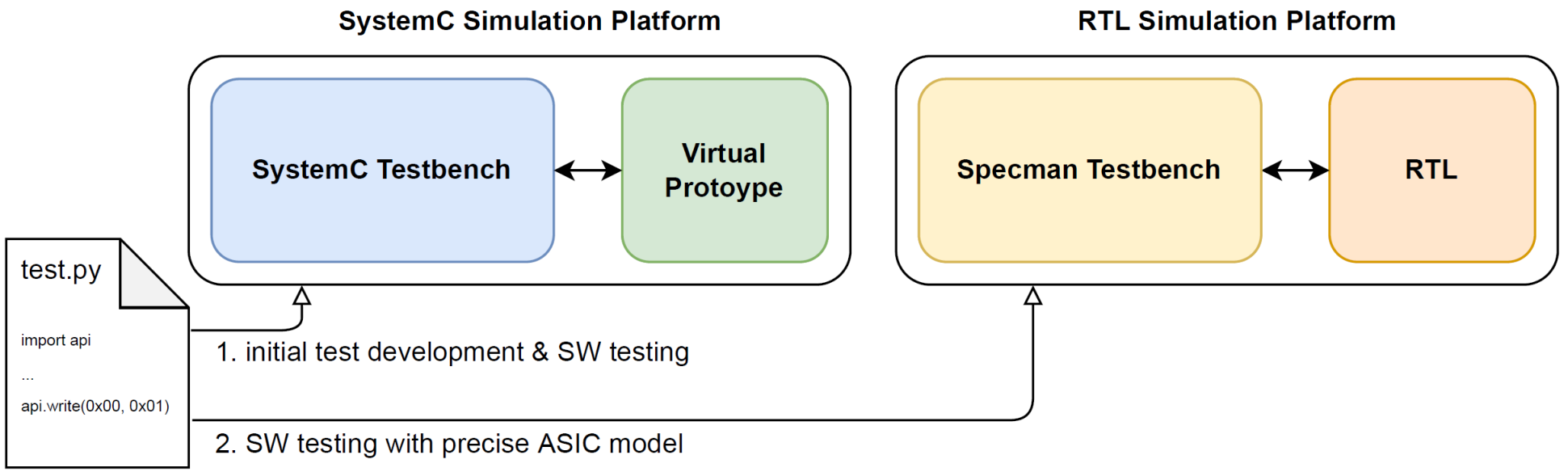
- Easy and flexible access to SystemC and RTL simulation for SW development
- Facilitate reuse of test stimuli between both simulation platforms

Outline

- Motivation
- **Overview of the proposed Framework**
- Python Test – SystemC Model API in Detail
- Python Test – Specman API in Detail
- Application results
- Summary

Overview of the proposed Framework

- Common Python interface to SystemC and RTL top-level simulation
  - Allows scripted host interaction and simulation control



Overview of the proposed Framework

- Common API Methods:
  - read(address, length)
  - read_dbg(address, length)
  - write(address, [list of data])
  - register_cb_int*(callback_func)
  - log(message)
  - wait(time, uint)
  - wait_until(time_s)
  - …

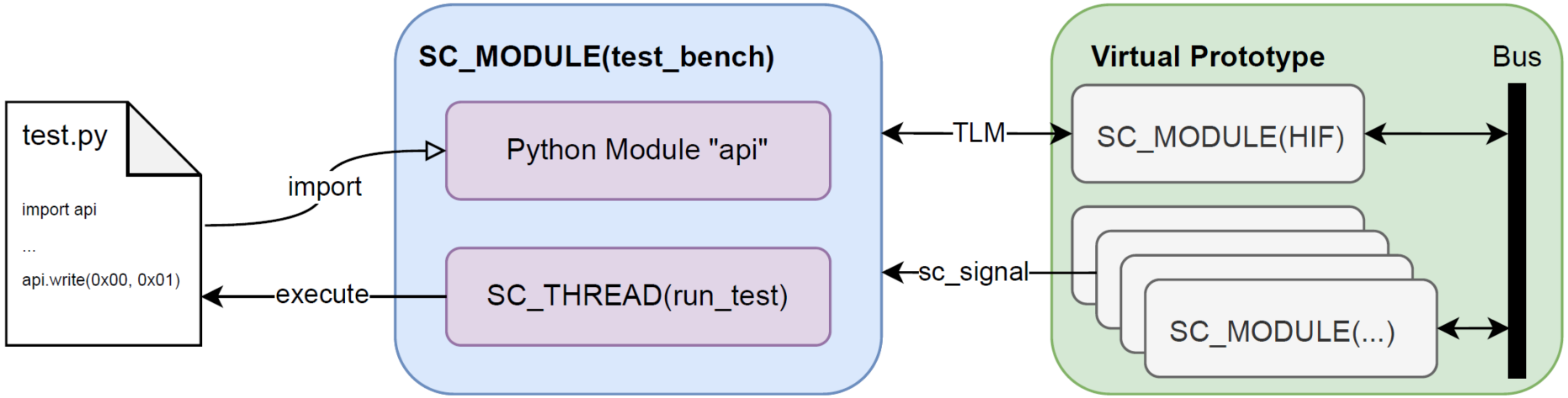
```
import api
from common_functions import *
import json
import regs as DUT

def check_config_register_default():
    check_n_stop("DBG_ADDR_SENSOR_CONFIG_UI_MCU", HOST_BUS.RESET_VAL_EXT_S)
    check_n_stop("DBG_ADDR_GYRO_CONFIG_UI_MCU", HOST_BUS.RESET_VAL_EXT_G)

try:
    api.log("================================================")
    test_count = 0
    api.log("Python test-bench starts here")
    api.wait(5, "ms")
    api.set_spi_mode()
    check_n_stop("SPI_ADDR_P0_MANU_ID_CHIP_ID", 0x1234)
    api.log("################################################")
    api.write(DUT.SPI_ADDR_P0_GP_UI_REG6, [HOST_BUS.FA_6])
    api.write(DUT.SPI_ADDR_P0_GP_UI_REG7, [HOST_BUS.FA_6T])
    api.write(DUT.SPI_ADDR_P0_CHIP_CONFIG, [0x1])
    api.write(DUT.SPI_ADDR_P0_MCU_CONFIG, [0x2])
    api.wait(10, "ms")
# ...
```

Outline

- Motivation
- Overview of the proposed Framework
- **Python Test – SystemC Model API in Detail**
- Python Test – Specman API in Detail
- Application results
- Summary

Python Test – SystemC Model API in Detail

- Testbench embeds a Python interpreter
- Test scripts are passed as an argument to the executable model



Python Test – SystemC Model API in Detail

Benefits of embedding Python in a SystemC testbench:

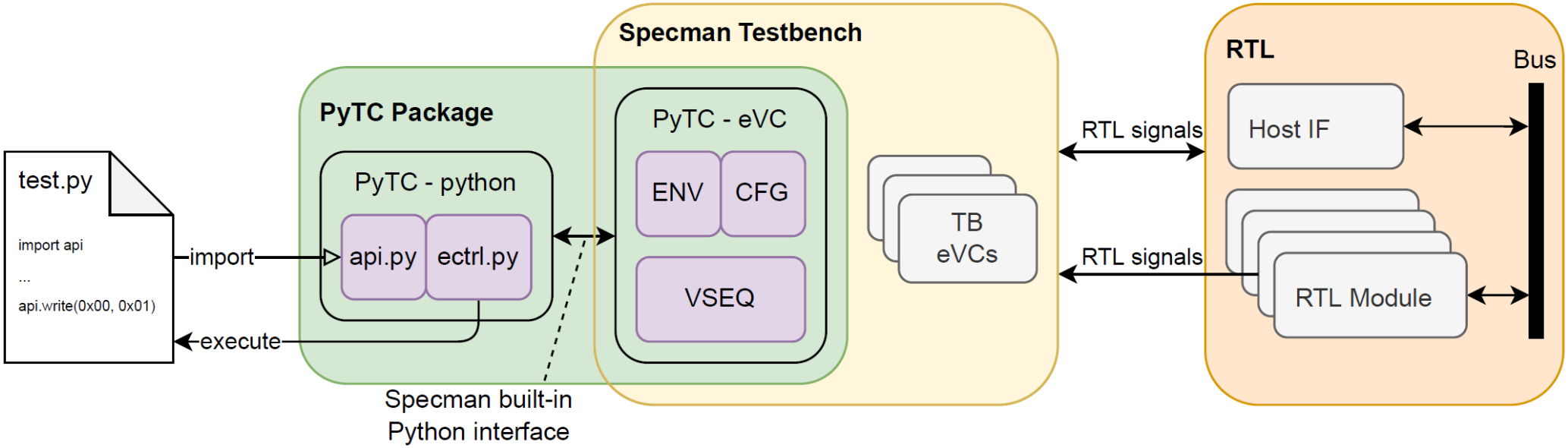
- Change test stimuli without recompiling the model
- Free from license cost
- Host machine independent
- Modelling of the virtual prototype is not limited by certain libraries
  - EDA SystemC simulators require specific SystemC versions to be used
- High simulation performance compared to EDA SystemC simulators
  - Experiments showed that simulation with Xcelium is >5 times slower

Outline

- Motivation
- Overview of the proposed Framework
- Python Test – SystemC Model API in Detail
- **Python Test – Specman API in Detail**
- Application results
- Summary

Python Test – Specman API in Detail

- Implemented as a generic Python Test Case (PyTC) package
- Uses Specman built-in Python interface

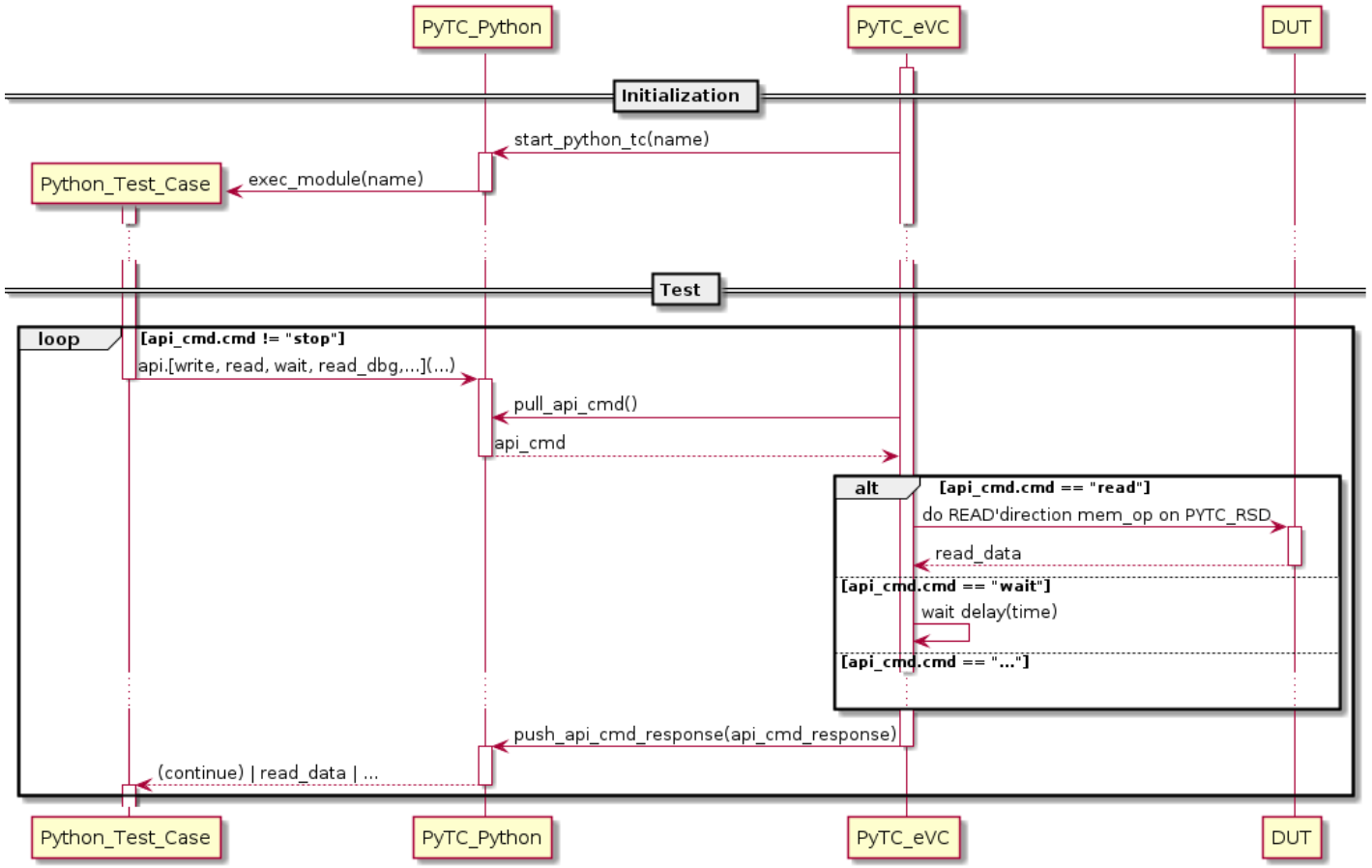


Python Test – Specman API in Detail

- Using queues for interthread communication in Python
- Specman/Python communicate via common data structs/classes

```
struct Api_cmd_s {
  !%cmd : string;
  !%reg_addr : int;
  !%write_data : list of int;
  !%read_length : int;
  !%wait_time : int;
  !%wait_unit : string;
  !%log_string : string;
};
```

```
struct Api_cmd_response_s {
  !%read_data : list of int;
};
```



Python Test – Specman API in Detail

Benefits of using the PyTC package:

- Reuse of test stimuli from SystemC model
- Access to RTL simulation for engineers without Specman knowledge
- Adapt test stimuli without recompilation
- Low integration effort in Specman testbenches

Overhead due to language crossing determined in experiments:

- 100 operations → 0.4% more simulation time
- 1000 operations → 2.0% more simulation time

Outline

- Motivation
- Overview of the proposed Framework
- Python Test – SystemC Model API in Detail
- Python Test – Specman API in Detail
- **Application results**
- Summary

Application results

Python Test - SystemC API

- Used in several sensor projects
- Free of license costs, workstation independent
- Established as a flexible platform for software development and regression

Python Test - Specman API

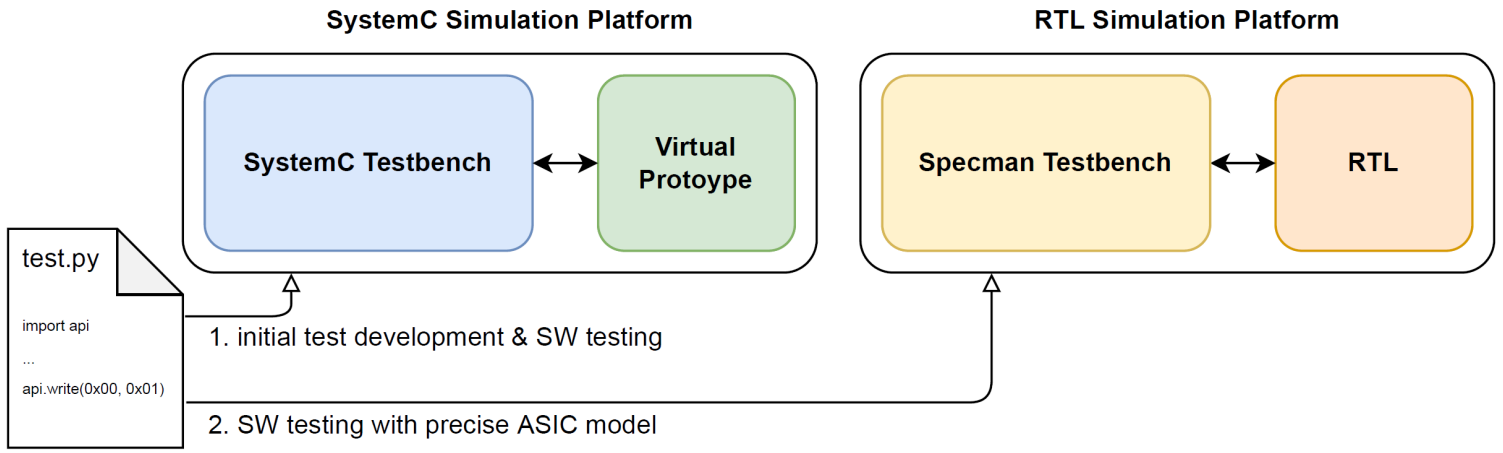
- PyTC integration successfully validated in various development environments
- Used in ongoing ASIC projects → project evaluation pending
  - High reuse expected for tests with equal testbench constraints

Outline

- Motivation
- Overview of the proposed Framework
- Python Test – SystemC Model API in Detail
- Python Test – Specman API in Detail
- Application results
- **Summary**

Summary

- Python Test APIs for scripted host interaction and simulation control
- Usage of both APIs enables efficient software testing methodology



Questions