TheSyDeKick

System development and verification framework in Python

Presenter: Marko Kosunen
Authors: TheSyDeKick contributors

October 27, 2022

Outline

- TheSyDeKick - What it is
- Simulator interfaces
- Principle of simulation
- Next step: Connecting Entities
- Example 1: Inverter tests
- Example 3: ADC model
- Example 4: Multi-user MIMO receiver
- Example 5: Outphasing transmitter
- Example 6: A-Core RISC-V processor chip
- Example 7: EM field simulations
- Conclusion and acknowledgements

TheSyDeKick - What it is

TheSyDeKick project strusture

TheSyDeKick projectTheSyDeKick project

TheSyDeKick project

helpers
Entities

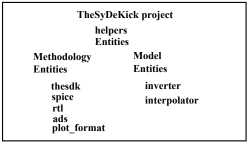
Methodology
Entities

- **thesdk**
- **spice**
- **rtl**
- **ads**
- **plot_format**

Model
Entities

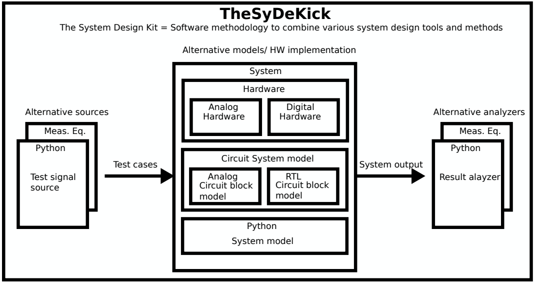
- **inverter**
- **interpolator**

What is TheSyDeKick



- TheSyDeKick is a well structured container for:
  - Python packages (Entities), that provide *formalism and methods to run simulations in python, and with external simulators*
  - Python packages (Entities), that provide formalism to create and use *IO compatible models*
- TheSydeKick is a cumulative collection of Python classes and methods that help the designer to carry out the most common tasks encountered in microelectronics design. Most likely the *simulation task you have in mind has some supporting methodology available in TheSyDeKick*
- TheSyDeKick is *nothing more*

What is it?



- TheSyDeKick formalism helps the designer to:
  - Write and simulate modular, hierarchical, IO compatible hardware models with various levels of abstraction (python, spice/rtl or even EM field simulatotor)
  - Control the abstraction level with one parameter on chosen level of hierarchy.

https://github.com/TheSystemDevelopmentKit/thesdk_template/wiki/The-System-Development-Kit

What is it?

- Minimum set of structural constraints,
  - IO definitions
  - Blocks described as connected *Entities*.
- Aims to automation of *repetitive, well structured tasks*
  - Model and modeling environment structure with init scripts
  - Running simulations
  - Defining simulator calls
  - Documentation with Docstrings see for example.

    `https://thesystemdevelopmentkit.github.io/docs/index.html`
- Currently supports Python, Verilog, VHDL and three Spice variants.
- Open source simulators: Icarus for Verilog, NgSpice for analog circuits.
- Under construction: Measurement equipment.

`https://github.com/TheSystemDevelopmentKit`

TheSyDeKick project file structure

TheSyDeKick project structure

```
thesydekick_project/
└── Entities/
    ├── thesdk/
    ├── rtl/
    ├── spice/
    ├── ads/
    └── amplifier/
        ├── amplifier/
        │   └── __init__.py
        ├── spice/
        ├── simulations/
        └── doc/
```

https://github.com/TheSystemDevelopmentKit/thesdk_template

Simulator interfaces

Analog simulator interface

- Calling analog simulators is handled through a common interface, the **spice** module
- Ultimate goal of **spice**: support for most industry standard simulators
  - General (w.r.t to simulator), reusable simulation testbenches
  - Centralized post processing based on open source tools and libraries.
- Save time in re-writing testbenches for different simulations and centralize effort to all-in-one testbench
  - Currently support for DC, AC, and Transient analysis. Easy to add other analysis types as IO formats are the same.

`https://github.com/TheSystemDevelopmentKit/spice`

Digital simulator interface

- Calling digital simulators is handled the **rtl** module
  - Most commonly used simulation testbenches automatically generated for both Verilog and VHDL design under tests.
  - Automated file IO generation for strings (bit vectors), integers, and complex numbers.
  - Support for any signal type through customizable format parameter.

`https://github.com/TheSystemDevelopmentKit/rtl`

EM field simulator interface

- Interface enables EM simulator Momentum from python desing environment.
- Used so far for example:
  - Together with Berkeley Analog Generator in characterization of parametrized inductors transformers.
  - In extraction of S-parameters from 3D layout structures.

`https://github.com/TheSystemDevelopmentKit/ads`

TheSyDeKick simulation procedure

1. Write the testbench
   1. Specify analysis type, inputs, outputs, supplies etc. based on TB properties
   2. Configure simulator options, corners, etc. based on testbench properties
   3. Specify desired outputs, e.g. FFT of waveform, transient waveform
   4. Specify run cases, e.g. transient analysis for 2 different netlists, etc.
2. Run the simulation
3. Analyze results
4. (Optional) Repeat simulation for different sets of parameters

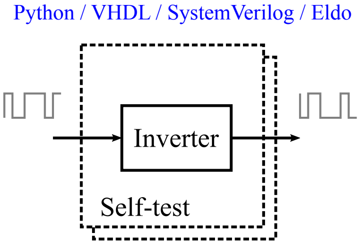
- Once you have one testbench, reuse for similar applications makes things faster
- Results saved (if user chooses so), no need to repeat lengthy simulations to fix errors in post processing

Principle of simulation

First step: Self test

- Example: inverter
- Functional model: Inverts input signal
- IOS is a Bundle-type container for storing IO data
- Can have as many members as needed

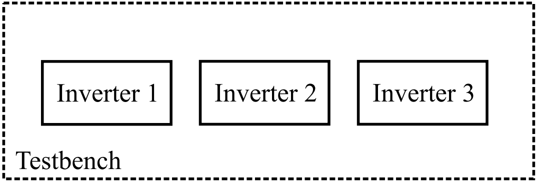
First step: Self test



- Multiple models for the same entity: Python,spice,rtl
- Self-test is a way to verify the functionality of entity “in vacuum”
- Self-test generates and feeds the identical input data to all simulators

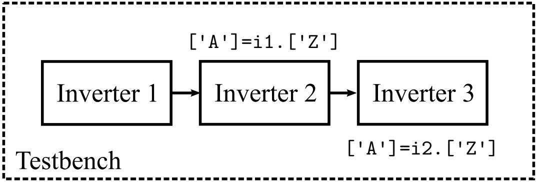
Next step: Connecting Entities

Example 2: Inverter Chain



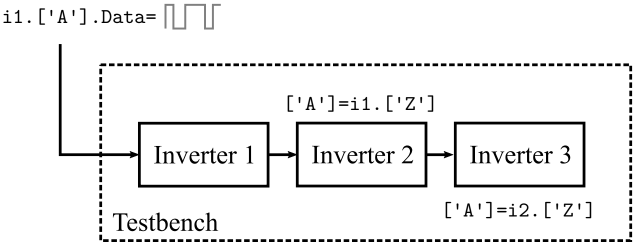
- Entities care connected inside another entities, (or in a testbench).
- Chain of inverter instances can created with a for loop.

Example 2: Inverter Chain



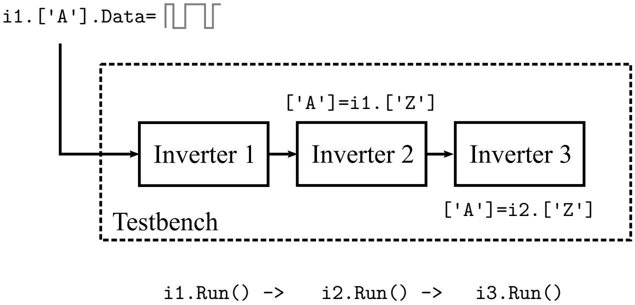
- Entities care connected inside another entities, (or in a testbench).
- Chain of inverter instances can created with a for loop.
- Data flow defined by IOS connections

Example 2: Inverter Chain



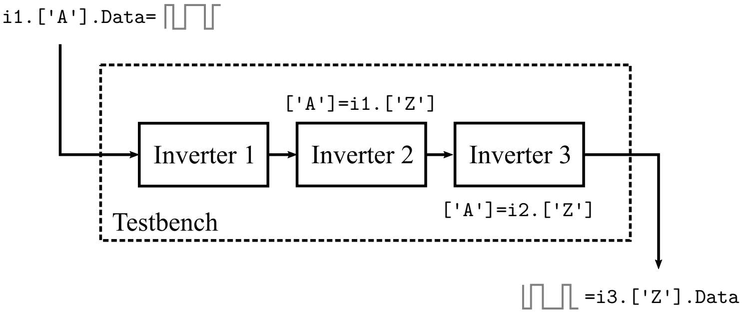
- Entities care connected inside another entities, (or in a testbench).
- Chain of inverter instances can created with a for loop.
- Data flow defined by IOS connections
- Signal is assigned as input of 1st element

Example 2: Inverter Chain



- Entities care connected inside another entities, (or in a testbench).
- Chain of inverter instances can created with a for loop.
- Data flow defined by IOS connections
- Signal is assigned as input of 1st element
- Signal is propagated through the chain by running the entities in user defined sequence

Example 2: Inverter Chain



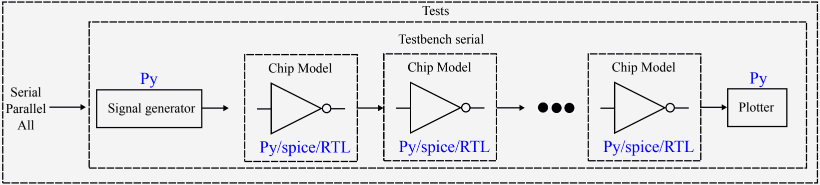
- In the end, the output of the last element contains processed data

Example 1: Inverter tests

Verifying inverter system

- Example of test hierarchy from bottom up: inverter, inverter testbench and inverter tests.

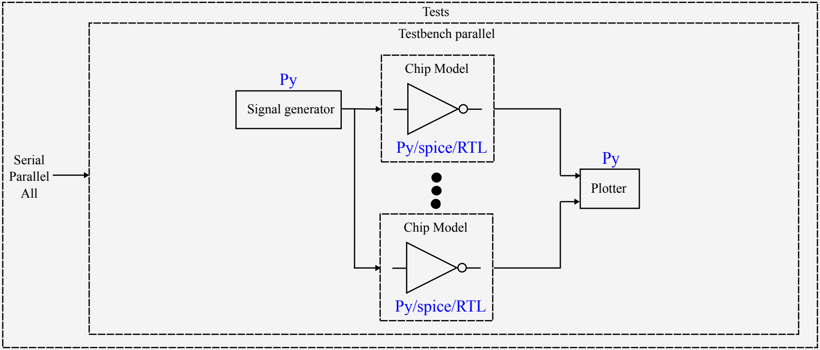
Inverter system serial test configuration



Multiple testbench configurations

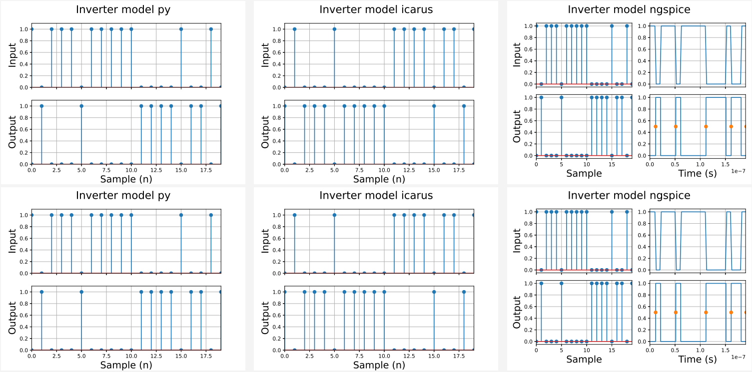
- Construction methods of testbench class enable various test configurations.
- All parallel executions can be executed in parallel threads resulting in remarkable speedup.

Inverter system parallel test configuration



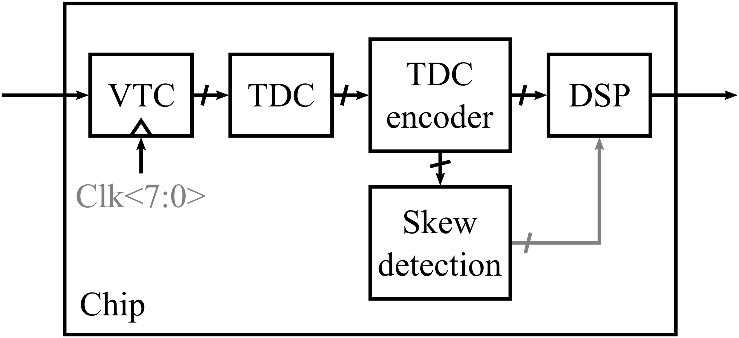
Inverter system verification results

Inverter system results

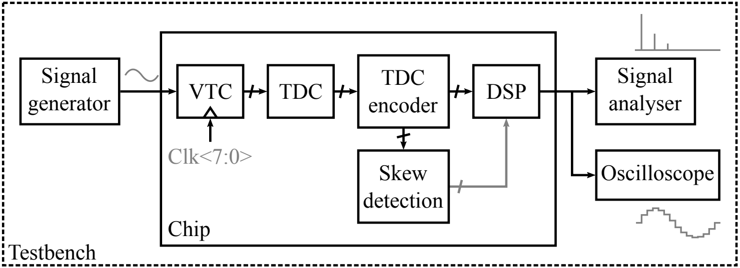


Example 3: ADC model

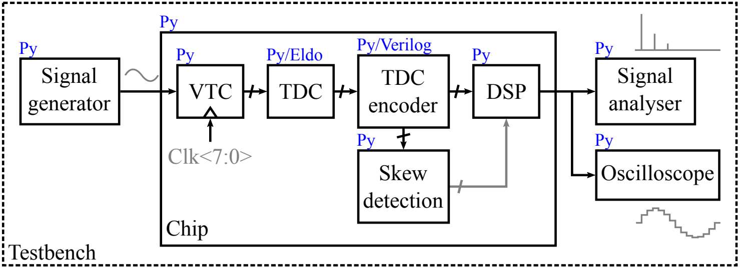
Example 3: ADC Model



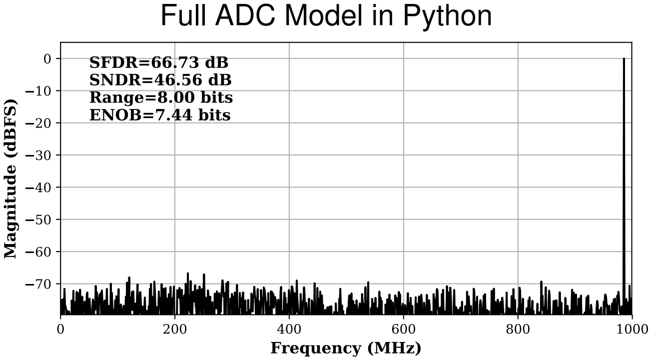
Example 3: ADC Model



Example 3: ADC Model



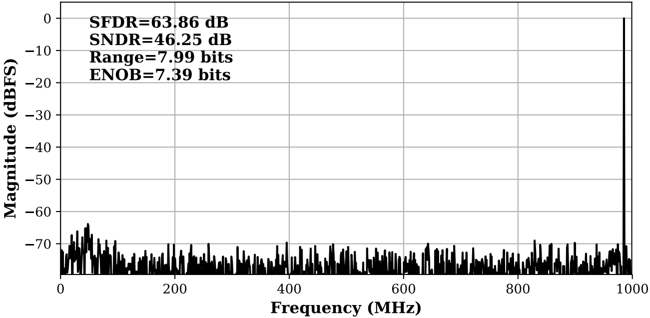
Example 3: ADC Model



- Spectrum plotted by signal_analyser -module, which also calculates SNDR etc.

Example 3: ADC Model

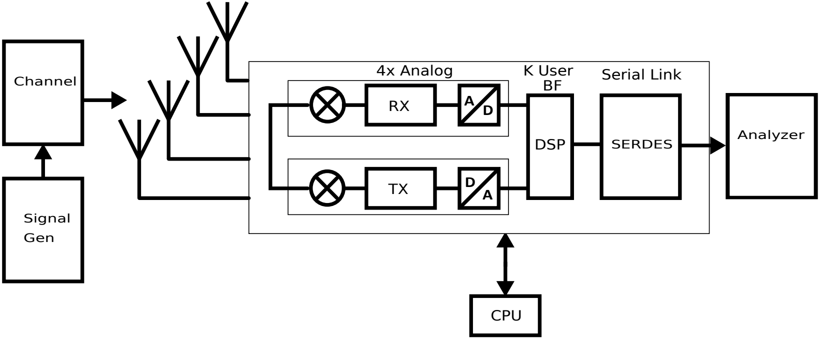
Full ADC Model in Python with Transistor-level TDC Model



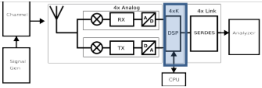
- Same exact simulation as before, except the TDC block is simulated as a spice-netlist in Eldo

Example 4: Multi-user MIMO receiver

Example 4: Multi-user MIMO receiver

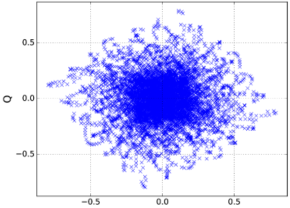


- Goals:
  - First, model the receiver of a single chip with python
  - Python models swappable to RTL and analog models
  - Generated sub-blocks verified at the system level
  - Control the HW generators of the block (emerging feature)

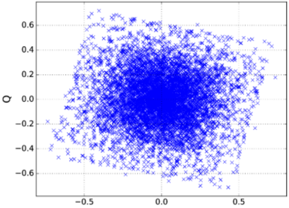


Channel 802.11n C

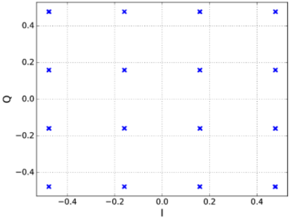
DSP, Ant=0, Usr=0, py, EVM=0.19 dB, BER= 0.364



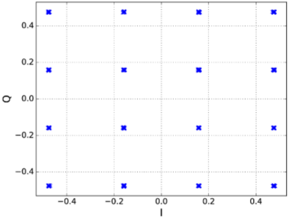
DSP, Ant=0, Usr=1, py, EVM=2.09 dB, BER= 0.453

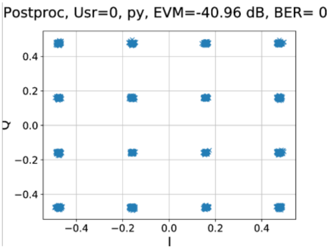


Postproc, Usr=0, py, EVM=-55.09 dB, BER= 0

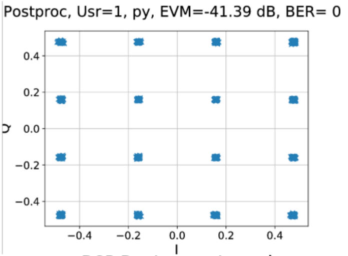


Postproc, Usr=1, py, EVM=-52.50 dB, BER= 0

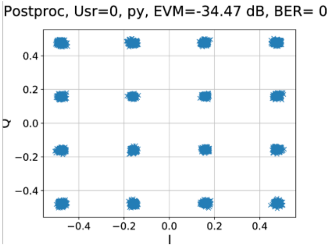




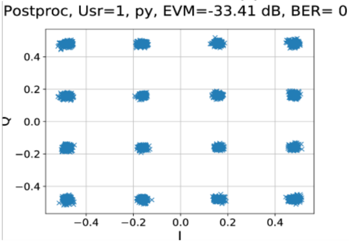
DSP Decimator in python



DSP Decimator in python



DSP Decimator in verilog

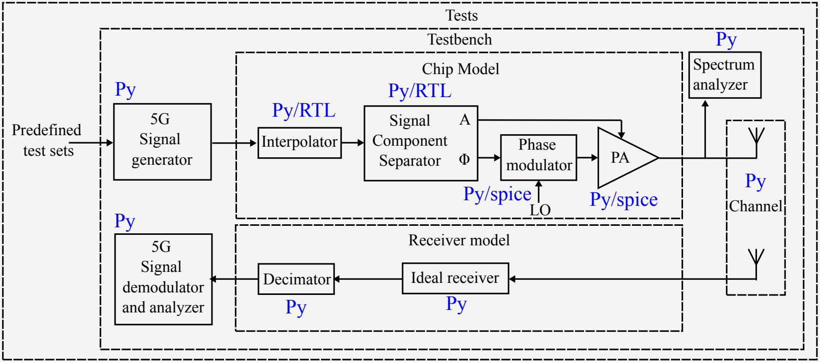


DSP Decimator in verilog

Example 5: Outphasing transmitter

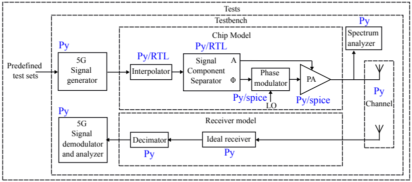
Outphasing transmitter system

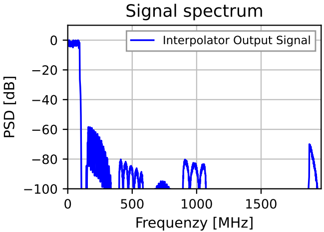
Outphasing system simulation



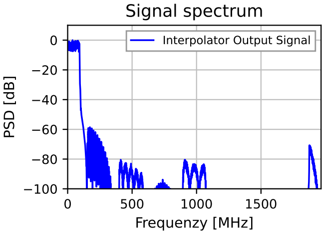
- System development
  - Model the transmitter chip with python
  - Model the ideal receiver in python
  - Simulate the functionality

Interpolator frequency response



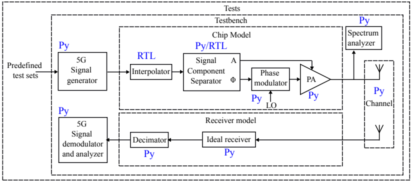


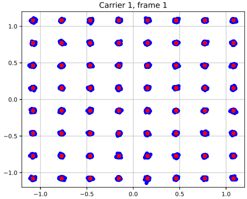
(a) Interpolator SW



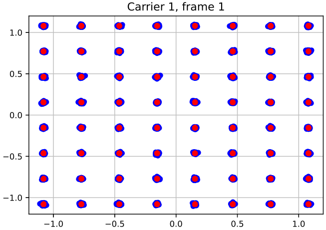
(b) Interpolator HW

Interpolator EVM simulation





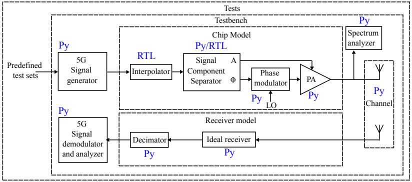
(c) Interpolator HW (EVM, 16-bit)

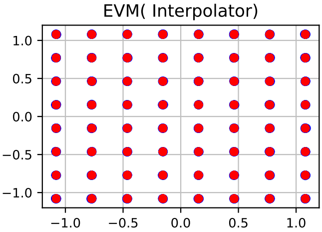


(d) Interpolator HW (EVM, 20-bit)

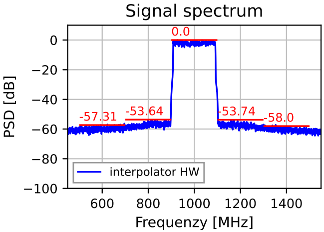
- EVM(16-bit): 1.6 - 1.52%
- EVM(20-bit): 1.3 - 1.08%

Interpolator EVM and ACLR





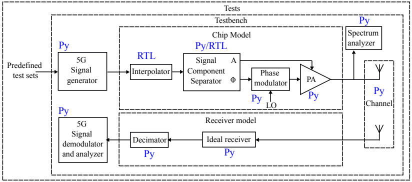
(e) Interpolator (HW, EVM)

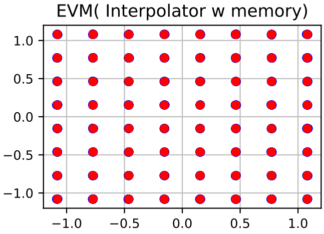


(f) Interpolator (HW, ACLR)

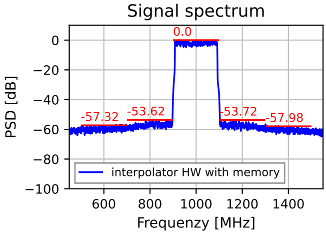
- EVM : 0.338%
- ACLR: $-53.64, -53.74$ dB

Interpolator hardware EVM and ACLR





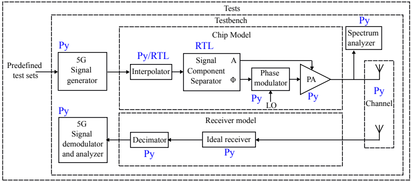
(g) Interpolator (HW, EVM)

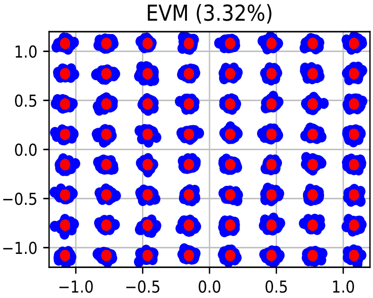


(h) Interpolator (HW, ACLR)

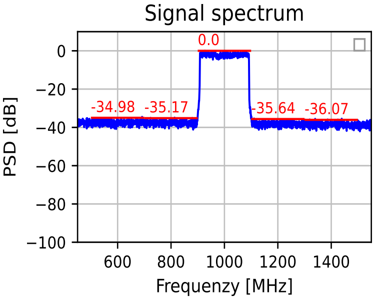
- EVM : 0.339%
- ACLR: -53.62, -53.71 dB

SCS 6-bit resolution EVM



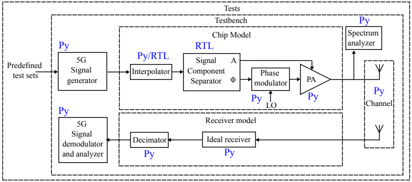


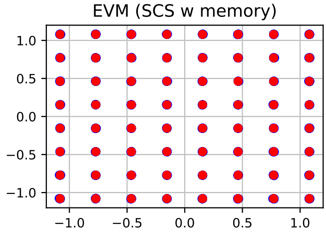
(i) EVM, SCS RTL simulation)



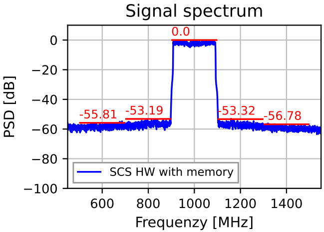
(j) ACLR, SCS RTL simulation)

SCS 10-bit resolution hardware EVM and ACLR





(k) EVM SCS RTL simulation



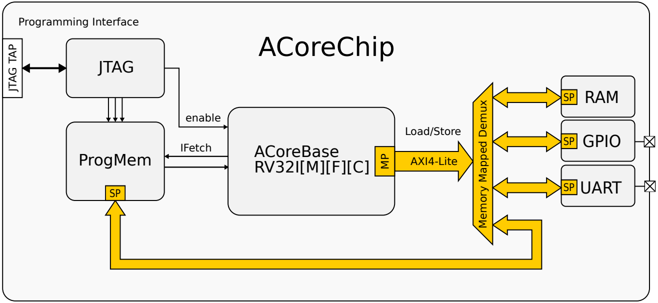
(l) ACLR SCS RTL simulation

- EVM: 0.336%
- ACLR: -53.18, -53.31 dB

Example 6: A-Core RISC-V processor chip

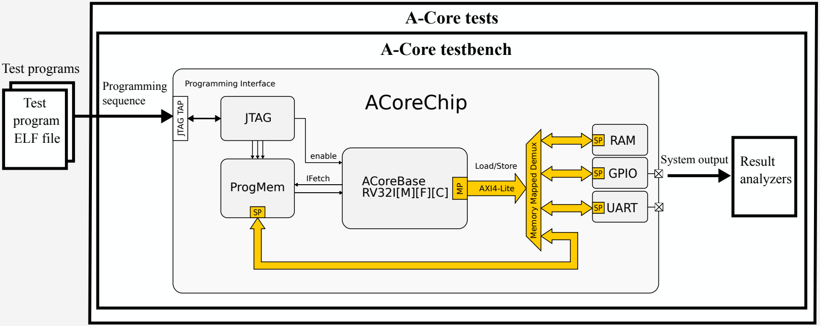
A-Core RISC-V processor chip

- SoC built with ACoreBase
- AXI4-Lite on-chip interconnect
- Memory mapped peripherals
  - Random Access Memory (RAM)
  - Read-only access to instruction memory (ProgMem)
  - General Purpose Input/Output (GPIO)
  - Custom accelerator IP



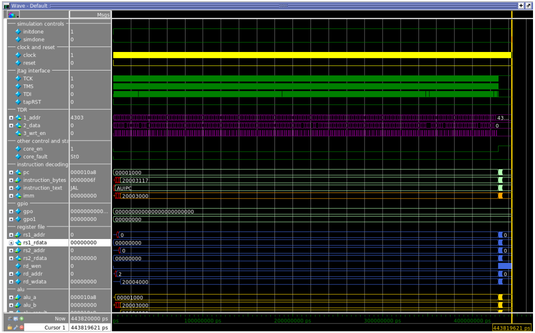
A-Core chip verification

A-Core test setup



TheSyDeKick simulations

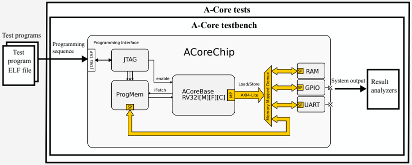
- For system-level simulations
- TheSyDeKick automates the following verification tasks
  - Generate a simulation test vector
    - JTAG programming waveform from ELF file
    - Chip control signals (e.g. core enable)
  - Generate a verilog testbench
  - Invoke a digital simulator (ModelSim)



ACoreChip Entity

- Represents a simulatable model of ACoreChip
- Submodules
  - ACoreChip Chisel generator **`chisel/`**

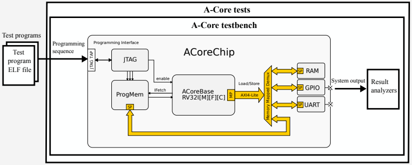
A-Core test setup



ACoreTestbenches

- `ACoreTestbenches` instance represents a test bench configuration.
- Testbench configuration is selected with appropriate `define*` method
  - For example, **`define_acorechip_sim_testbench(**kwargs)`** configures test-bench as a acorechip simulation.
  - Software, jtag config., etc. are provided as keyword arguments to **`define*`**

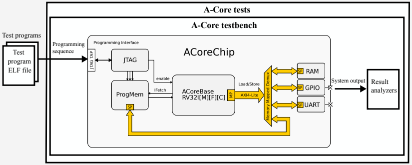
A-Core test setup



ACoreTestbenches

- A testbench configuration is a general combination of entities and related configuration for representing a unique test setup.
- The simulation testbench has two options: The SyDeKick and cocotb
  - The SyDeKick uses its own structures to generate the Verilog testbench, IO's and run with modelsim
  - Cocotb runs a cocotb test - can be used to create interactive debug environments in the future

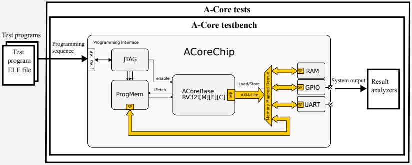
A-Core test setup



ACoreTests

- `ACoreTests` contains different concrete parameterizations of `ACoreTestbenches` and their dependencies:
  - Test software live under **`sw/`**
  - Simulation specific do-files live under **`interactive_control_files/modelsim`**
- Each test is defined as a method under `ACoreTests` class.
- For example, `sim_f_tests` configures a testbench with
  - F-extension enabled
  - test program for f-extension

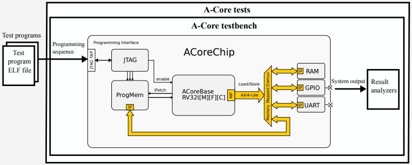
A-Core test setup



ACoreTests

- The test to be executed is selected with a command line argument. For example,
  `$ ./configure && make sim test_target=sim_f_tests` runs the simulation defined as `sim_f_tests`
- The simulator option can be selected with `simulator` argument for make, for example
  `$ make sim test_target=sim_f_tests simulator=cocotb`

A-Core test setup

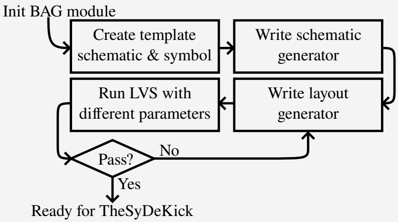


Example 7: EM field simulations

Layout preparation

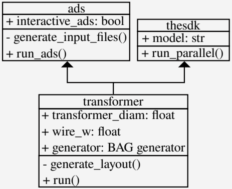
- Preparations: Constructing the layout from TheSyDeKick with Berkeley Analog Generator.
- Active development of BAG2 currently ongoing at *https://gitlab.com/mosaic_group/mosaic_BAG*

BAG workflow



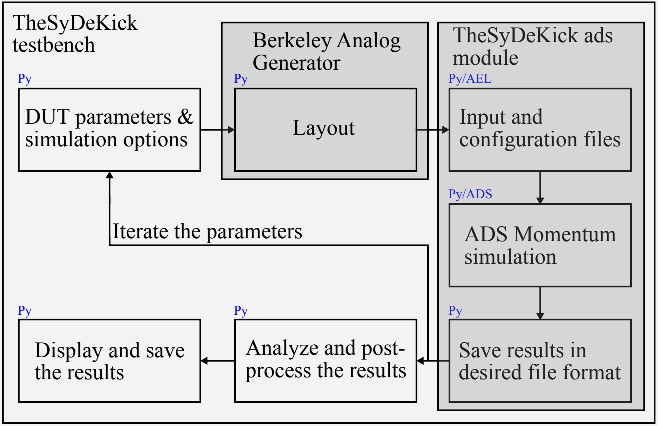
TheSyDeKick project structure

```
TheSyDeKick-project
|--- helpers
|--- Entities
    |--- thesdk
    |--- ads
    |--- spice
    |--- transformer
```



Transformer design flow diagram

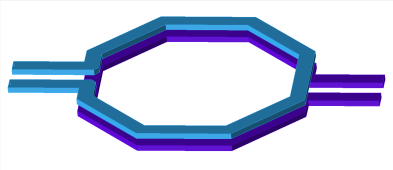
Design flow



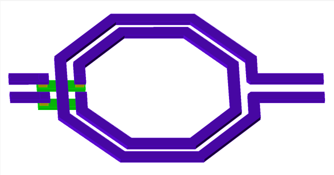
Transformer structures

- Parameters: Wire width, transformer diameter.

A stacked transformer

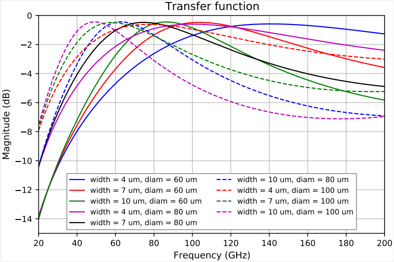


An Interwound transformer

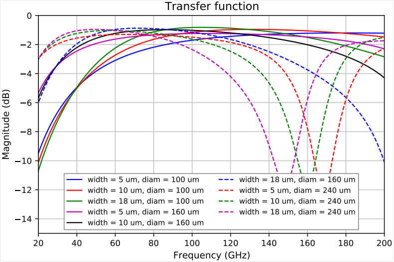


Transfer functions

Transfer function of a stacked transformer

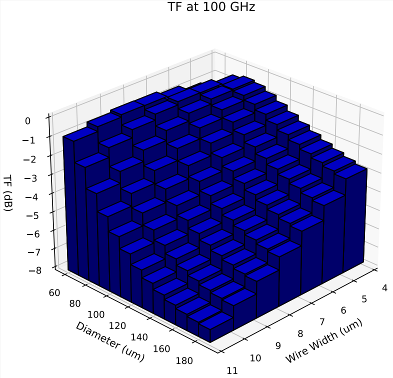


Transfer function of an interwound transformer

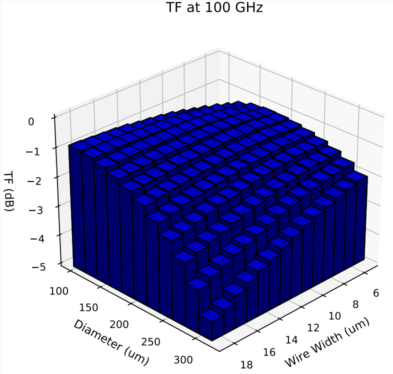


Insertion loss

Insertion loss of a stacked transformer



Insertion loss of an interwound transformer



Conclusion and acknowledgements

Conclusion

- Modular design environment through well defined IO boundaries and *Entity* definitions.
- Open Source
- Automates repetitive verification tasks and provides means for programmatic verification with various simulators
- Support for measurement equipment under development.
- *Enables* co-development by multiple designers using various verification tools.
- *Enabled by* utilization of programming methodology and version control in hardware design context.

Acknowledgement

- TheSyDeKick framework has been initiated during 2017-2019 under Marie Sklodowska-Curie project *ADVANTAG5*, in collaboration with Aalto University and University of California, Berkeley. (See licences, *https://github.com/TheSystemDevelopmentKit/thesdk*)
- Since the end of *ADVANTAG5* 2019, Lot of valuable development work has been carried out by several students of Aalto University, Finland.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 704947. Results reflects only the author's view and the European Comissions's Research Executive Agency Agency is not responsible for any use that may be made of the information it contains.