



# **Co-Simulation Workflow, Integration of Third Parties Components, Automated Bug Detection**

***Tutorial @ FPL Conference 2017 – Ghent – Belgium***

**Pietro Fezzardi**

Politecnico di Milano  
Dipartimento di Elettronica, Informazione e Bioingegneria  
[pietro.fezzardi@polimi.it](mailto:pietro.fezzardi@polimi.it)

## ❑ Co-Simulation Workflow

- ▶ How to use co-simulation
- ▶ How to provide user-defined inputs to co-simulation

## ❑ Integrations of Third Parties Component

- ▶ How to integrate third parties components
- ▶ How to co-simulate in presence of external component

## ❑ Automated Bug Detection with Discrepancy Analysis

# Co-Simulation Workflow

- ❑ A semi-automated design flow for testing your design
- ❑ It performs three main operations
  - ▶ it executes the high-level program with user-provided input
  - ▶ it simulates the HDL generated with HLS, with the same inputs
  - ▶ it compares the end results of the two executions
- ❑ Only the final results on the output pins are compared
- ❑ Not all the intermediate results are checked

- ❑ You provide input values for the tests
- ❑ The C code is executed with those inputs
  - ▶ If the top-function is not `main()` it also wraps it for testing
  - ▶ The return values are considered the golden reference for HW
- ❑ A testbench wrapper in HDL is generated to test the design
  - ▶ It communicates with the top-level to start the computation
  - ▶ It collects the computed results
- ❑ If the result do not match it emits an error message

- ❑ `--simulate` – activates co-simulation, does **NOT** select a simulator
- ❑ `--simulate=SIMULATOR_NAME` – selects the simulator

Valid values for `SIMULATOR_NAME` are:

- `VERILATOR` – Verilator, an open source cycle-based Verilog simulator
- `ICARUS` – Icarus Verilog, an open source event-based Verilog simulator
- `MODELSIM` – ModelSim from Mentor (Verilog, VHDL, Mixed)
- `XSIM` – The Vivado Simulator from Xilinx (Verilog, VHDL, Mixed)
- `ISIM` – The ISim ISE Simulator from Xilinx (Verilog, VHDL, Mixed)

- ❑ `--generate-tb=FILE_NAME.xml`

uses the values contained in the file `FILE_NAME.xml` as input

- ❑ Basic co-simulation
- ❑ Co-simulation with user-provided testbench
- ❑ Details of the XML file for testbench specification

## ☐ Pointer arguments

- ▶ comma-separated list of values used to initialize the memory
- ▶ co-simulation workflow and the testbench generation handles the rest
- ▶ both for C and for HDL

## ☐ They must respect any intrinsic assumption the code makes

## ☐ The provided inputs are *untyped*

## ☐ The parameter matching is *name based*



# Integration of Third Parties Components

- ❑ Integrating third-parties IPs into HLS designs
- ❑ Integrating hand-written HDL components in HLS designs
- ❑ Examples:
  - ▶ Using HLS for fast prototyping and then plug in optimized IPs
  - ▶ Using HLS to compose C, hand-written HDL and third party IPs

- ❑ Map an external HDL module onto a function call in C
- ❑ No need to provide the C code for the hardware module
  - ▶ Only the function declaration must be visible in C
  - ▶ For return value and parameters
- ❑ Use an XML file to inform BAMBU about the mapping

External IPs need to respect BAMBU calling conventions

	Fixed Latency Components	Variable Latency Components
bounded (bool)	1	0
cycles (integer)	N	Ignored
execution_time	float, in ns	Ignored
start and done ports	Ignored	used
Input	Function parameters	Function parameters
Output	Return value	Return value
Clock, reset	Necessary	Necessary
Pipelined	Supported	Not supported

## Integration of an External Component

- ❑ You need to provide a golden reference in C for the IPs
  - ▶ It is not used to generate HDL, just for co-simulation
  - ▶ It is common to have it when using HLS for prototyping
  - ▶ `--C-no-parse=file.c,names.c` flag to pass the reference C files
- ❑ Without the C version the co-simulation does not work
- ❑ It is useful to test external components compared to reference

## Co-Simulation with External Components

**See Directory 4.1**



# **Automated Bug Detection**

## **With**

# **Discrepancy Analysis**

- ❑ `--discrepancy`  
activates the Discrepancy Analysis
- ❑ `--discrepancy-force-uninitialized`  
be more strict on discrepancies involving uninitialized values
- ❑ `--discrepancy-no-load-pointers`  
assumes pointers are never stored in memories
- ❑ `--discrepancy-permissive-ptrs`  
do not trigger hard errors on pointers and addresses
- ❑ `--discrepancy-only=comma,separated,list,of,functions`  
activates Discrepancy Analysis only inside the given functions

## Co-Simulation with External Components

**See Directory 4.2**

---

# Questions?