

Design and Verification of FPGA Applications

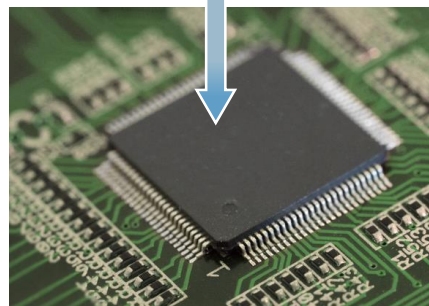
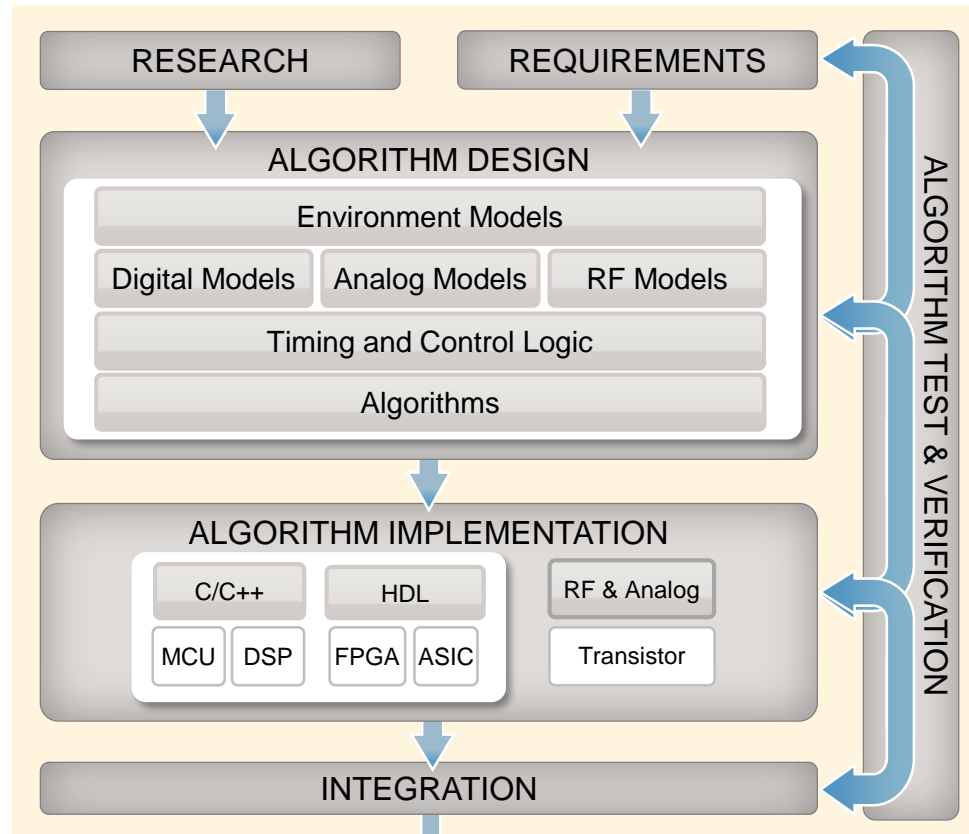
Giuseppe Ridinò giuseppe.ridino@mathworks.it
Paola Vallauri paola.vallauri@mathworks.it
MathWorks

Torino, 19 Maggio 2016, INAF

Agenda

- Model-Based Design for FPGA
- Generating HDL Code from MATLAB and Simulink
 - For prototyping and production
 - Optimizing code for efficiency
- Verifying HDL Designs with MATLAB and Simulink
 - Co-simulation with HDL simulators
 - FPGA-in-the-Loop verification
- Verifying HDL Designs outside MATLAB and Simulink
 - Generating code for integration with SystemC/TLM and SystemVerilog/DPI-C

Model-Based Design for FPGA



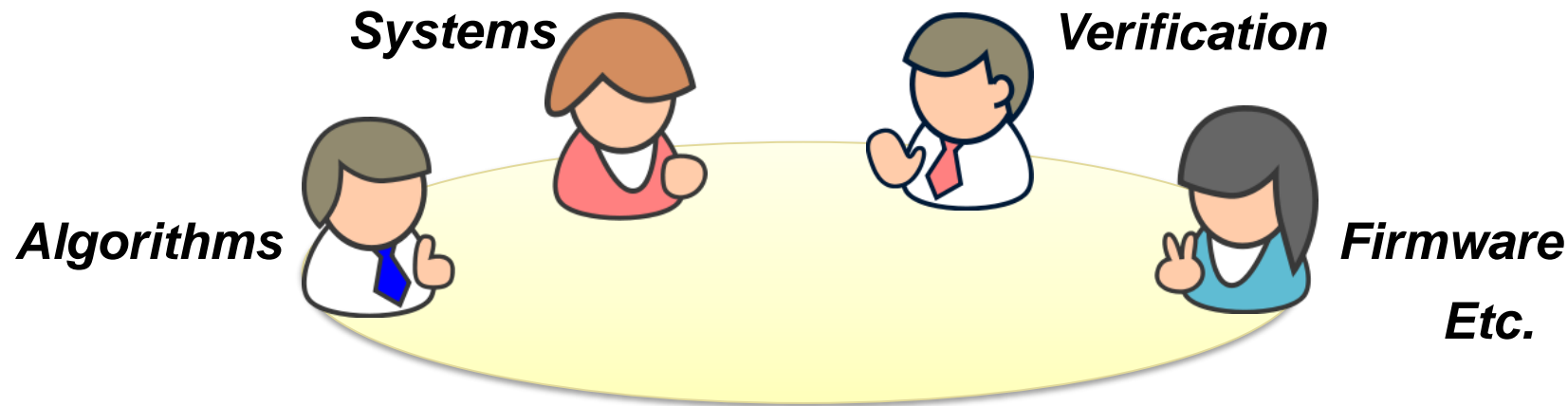
- Model multi-domain systems
- Explore and optimize system behavior
- Collaborate across multi-disciplinary teams

- Generate bit-accurate models
- Explore and optimize implementation tradeoffs
- Generate efficient code

- Verify designs to detect errors earlier in development
- Reuse testbenches
- Automate regression testing

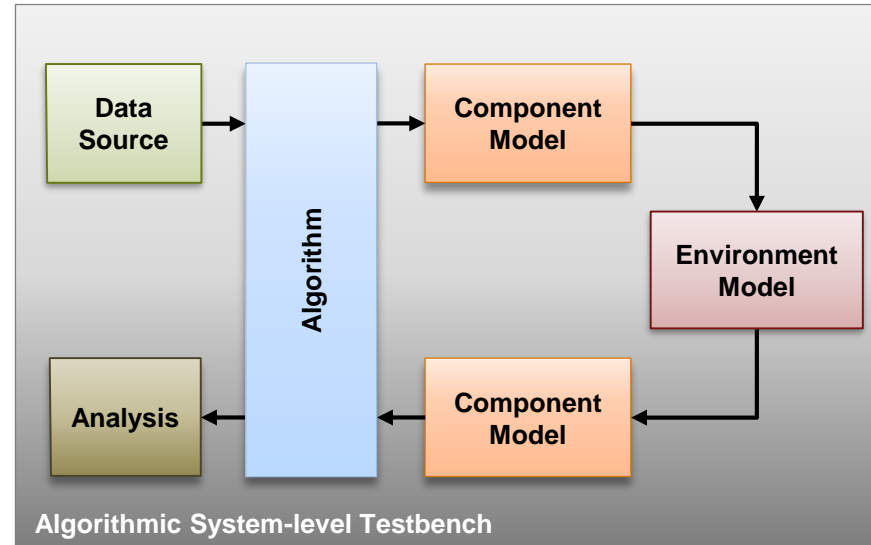
It's about Collaboration

- Usually, many engineers get involved in different parts of the design flow:



- Each brings valuable expertise from their discipline
- Model-Based Design aids collaboration across the project
 - integrating the workflow
 - providing the backbone of a common modelling environment

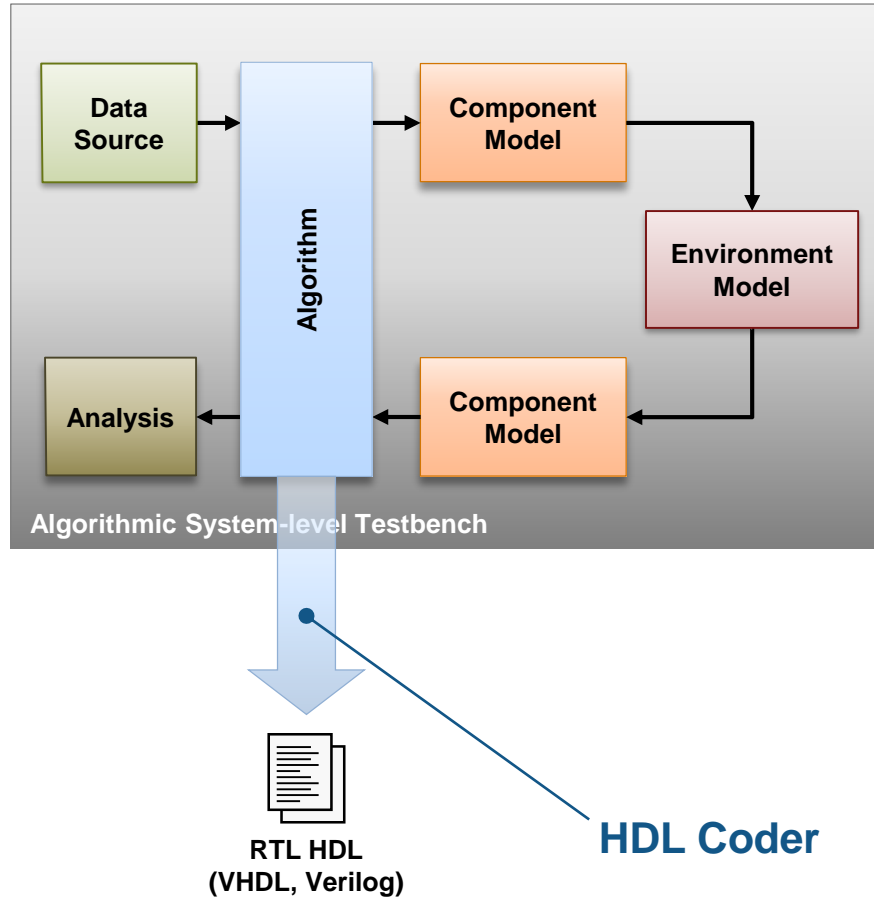
A Typical Model Structure



- **Algorithm** interacts with outside **environment** through other **components**.
- **Algorithm** is stimulated with **data**
- **Algorithm** performance is **analysed**.

Algorithm Development

Generation of HDL Source Code

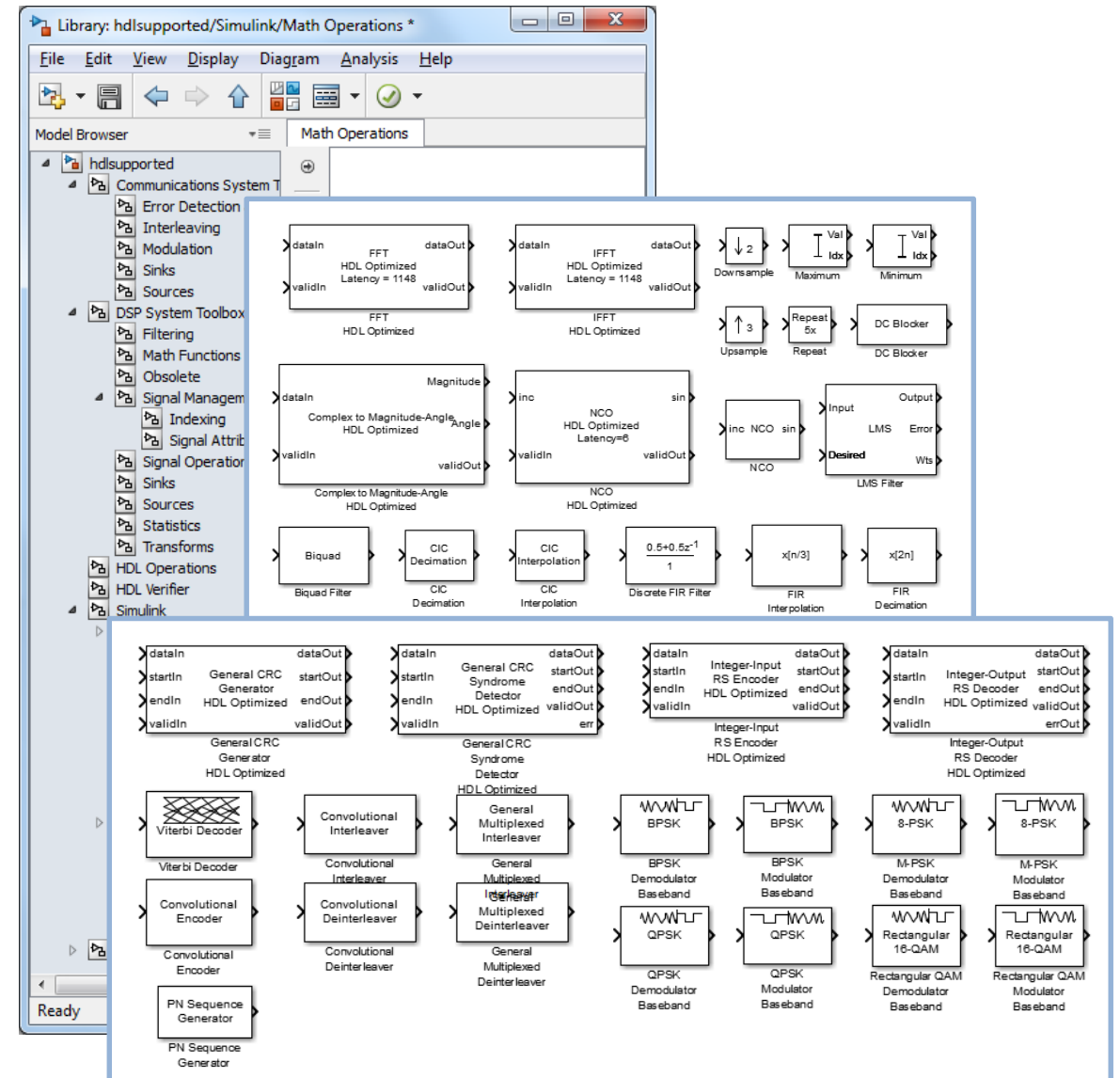


- HDL Coder
 - Generation of synthesible RTL HDL (VHDL or Verilog)
- Support for
 - MATLAB
 - Simulink
 - Stateflow
- Workflow Advisor
 - Guides through process
 - Preparing model for generation of HDL
 - Configuring HDL Generation options
 - Integrated with FPGA synthesis tools for timing annotation on model
 - Configurations for turnkey FPGA targets and IP Core generation

Simulink Library Support for HDL Generation

HDL Supported Blocks

- ~180 blocks supported
- Core Simulink
 - Basic and Array Arithmetic, Look-Up Tables, Signal Routing (Mux / Demux, Delays, Selectors), Logic & Bit Operations, Dual and single port RAMs, FIFOs, CORDICs, Busses
- Digital Signal Processing
 - NCOs, FFTs, Digital Filters (FIR, IIR, Multi-rate, Adaptive, Multi-channel), Rate Changes (Up & Down Sample), Statistics (Min / Max)
- Communications
 - Pseudo-random Sequence Generators, Modulators / Demodulators, Interleavers / Deinterleavers, Viterbi Decoders, Reed Solomon Encoders / Decoders, CRC Generator / Detector



MATLAB & Stateflow for HDL Generation

HDL Supported Blocks

- **MATLAB**

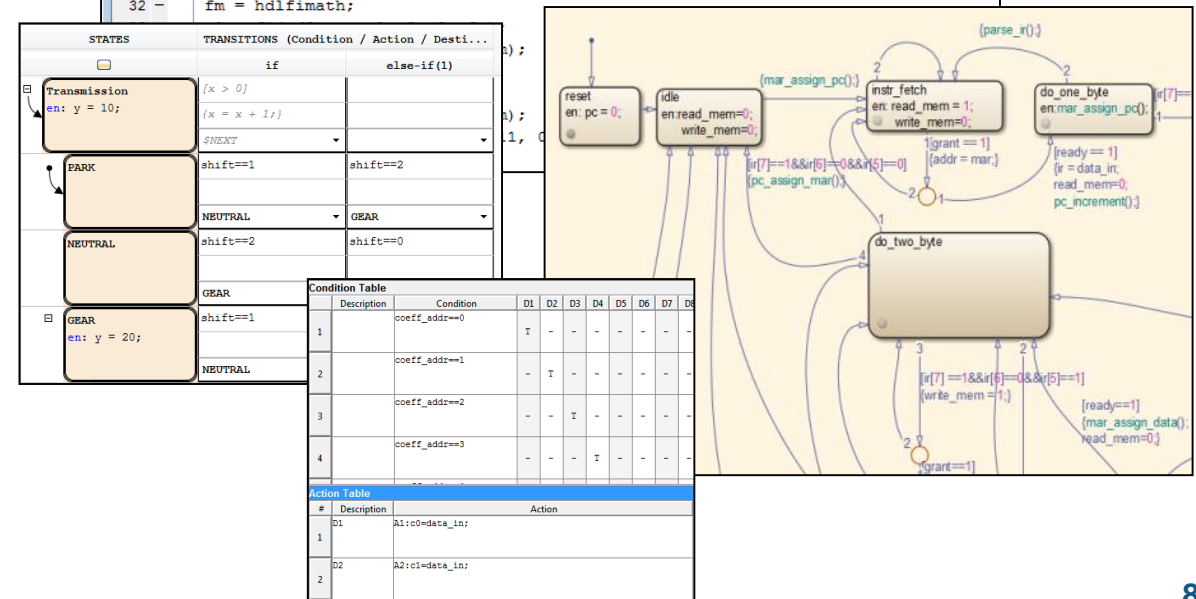
- Relevant subset of the MATLAB language for modeling and generating HDL implementations
- Useful MATLAB Function Block Design Patterns for HDL

- **Stateflow**

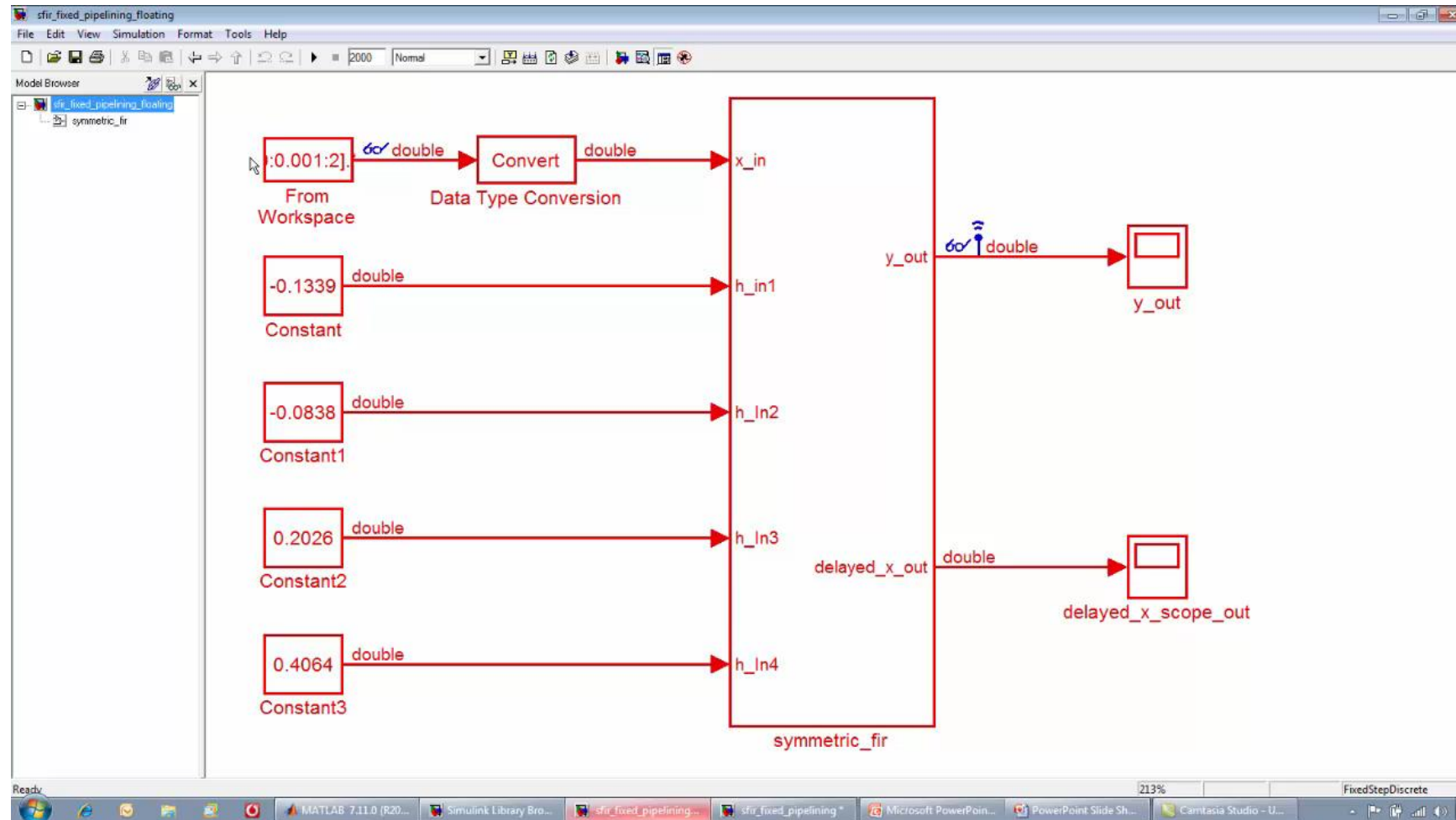
- Modeling FSMs (Mealy, Moore)
- Different modeling paradigms (Graphical Methods, State Transition Tables, Truth Tables)
- Integrate MATLAB code

```

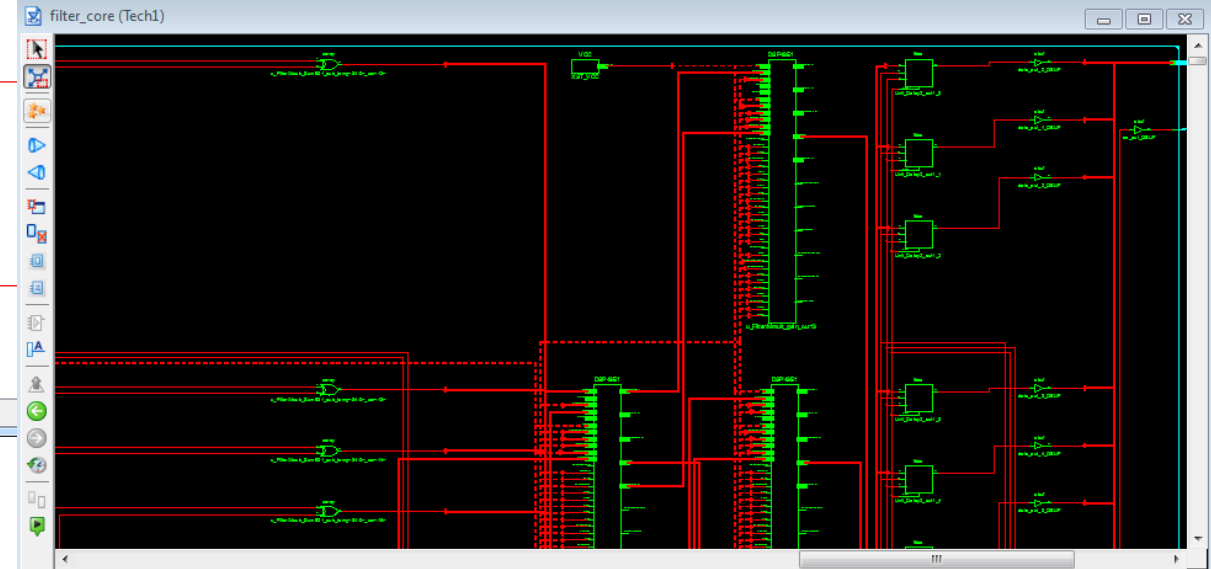
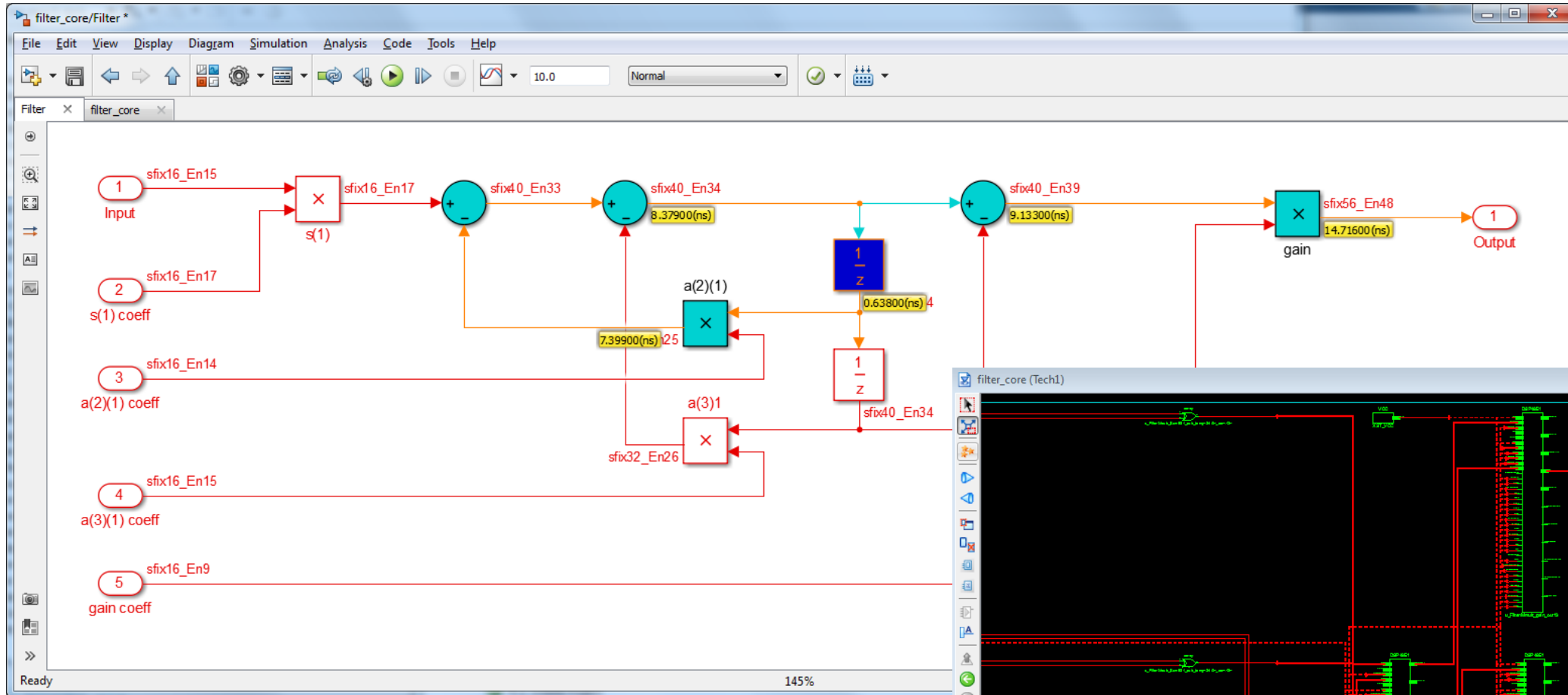
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 function xf_out = f12_xf(u, xd1, xd2, lb2, zd1, zd2)
18
19 fm = hdlfimath;
20 c2 = fi(2, 0, 2, 0, fm);
21 t1 = fi(xd1*c2, 0, 9, 0, fm);
22 a1 = fi(u + t1 + xd2, 0, 10, 0, fm);
23 t1 = fi(zd1*c2, 0, 9, 0, fm);
24 a2 = fi(lb2 + t1 + zd2, 0, 10, 0, fm);
25 xf_out = fi(a1 - a2, 1, 11, 0, fm);
26 end
27 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28 % Compute y gradient
29 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
30 function yf_out = f14_yf(xd2, u, yd2, lb1, zd2, lb2)
31
32 fm = hdlfimath;
  
```



HDL code generation



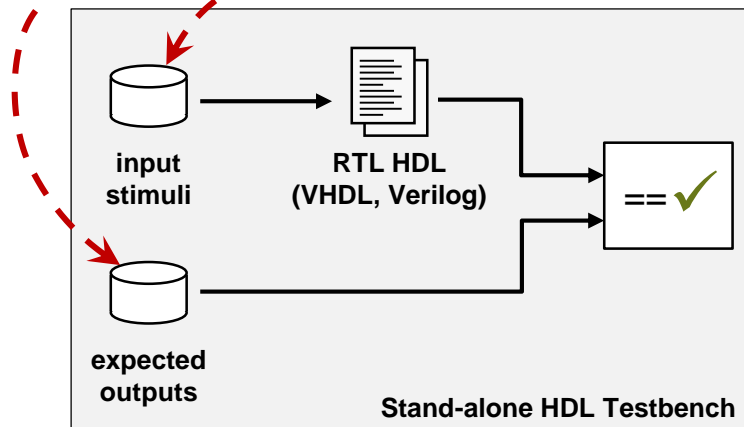
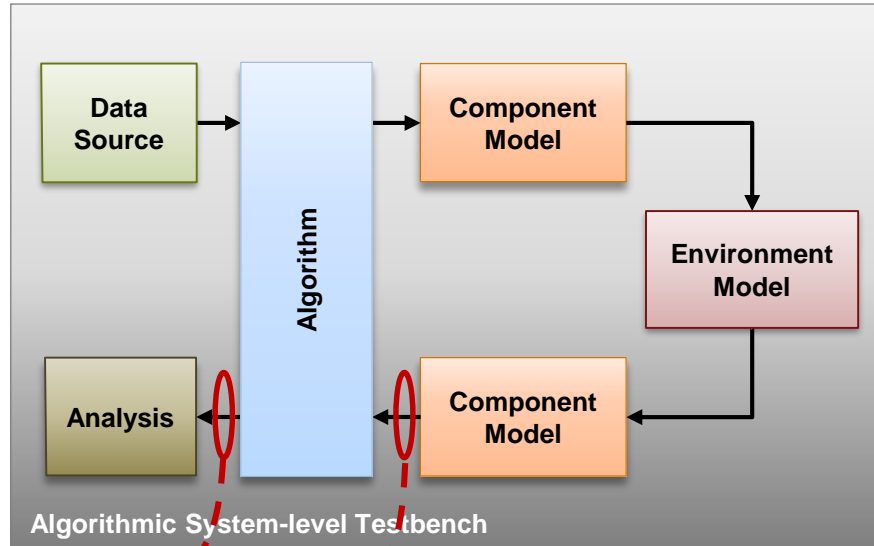
Critical Path Highlighting and Design Review



- Feedback in Simulink
- Review results in synthesis tools

Algorithm Verification

Data-driven Verification of HDL Source Code

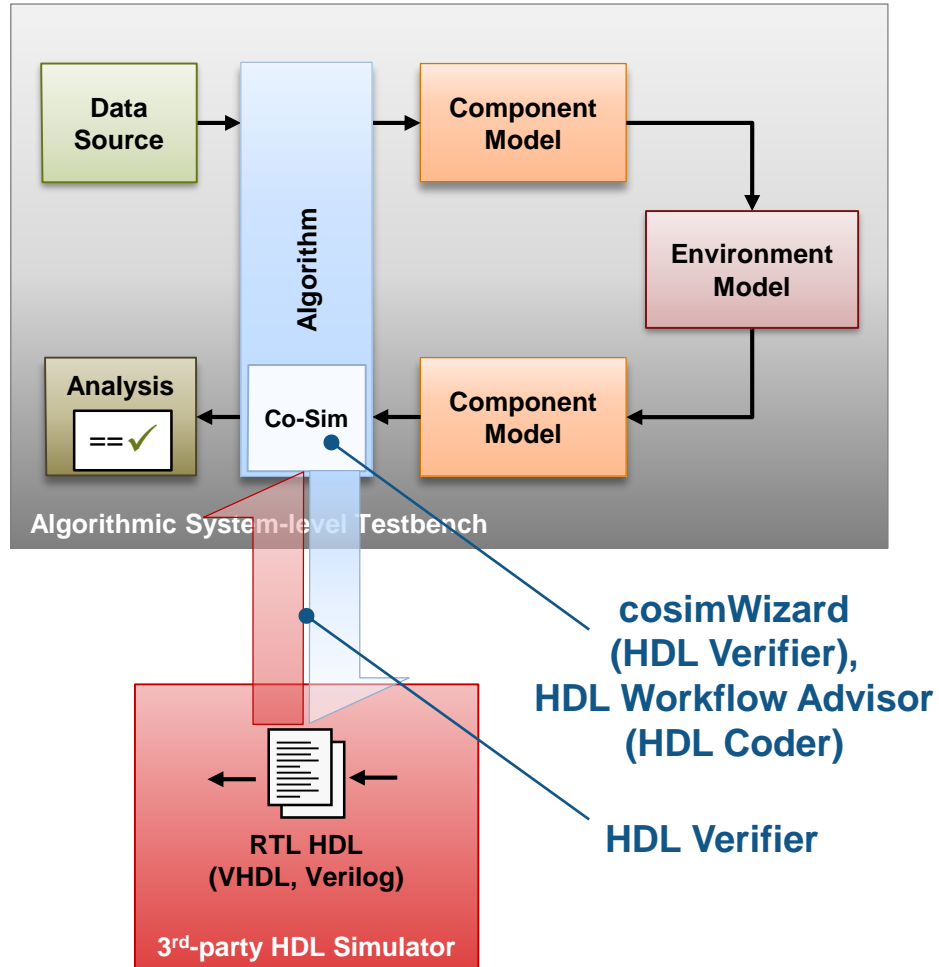


- Stand-alone HDL testbench
 - Stand-alone
 - Executable in any 3rd-party HDL simulator
 - Self-contained
 - Instantiated algorithmic RTL HDL (DUT)
 - Input stimuli stream at DUT top-level interface
 - Expected output stream at DUT top-level interface
 - Self-testing
 - Checks on bit and cycle accuracy

- Handwritten or generated code
 - With HDL Coder, RTL HDL and standalone testbenches are created automatically

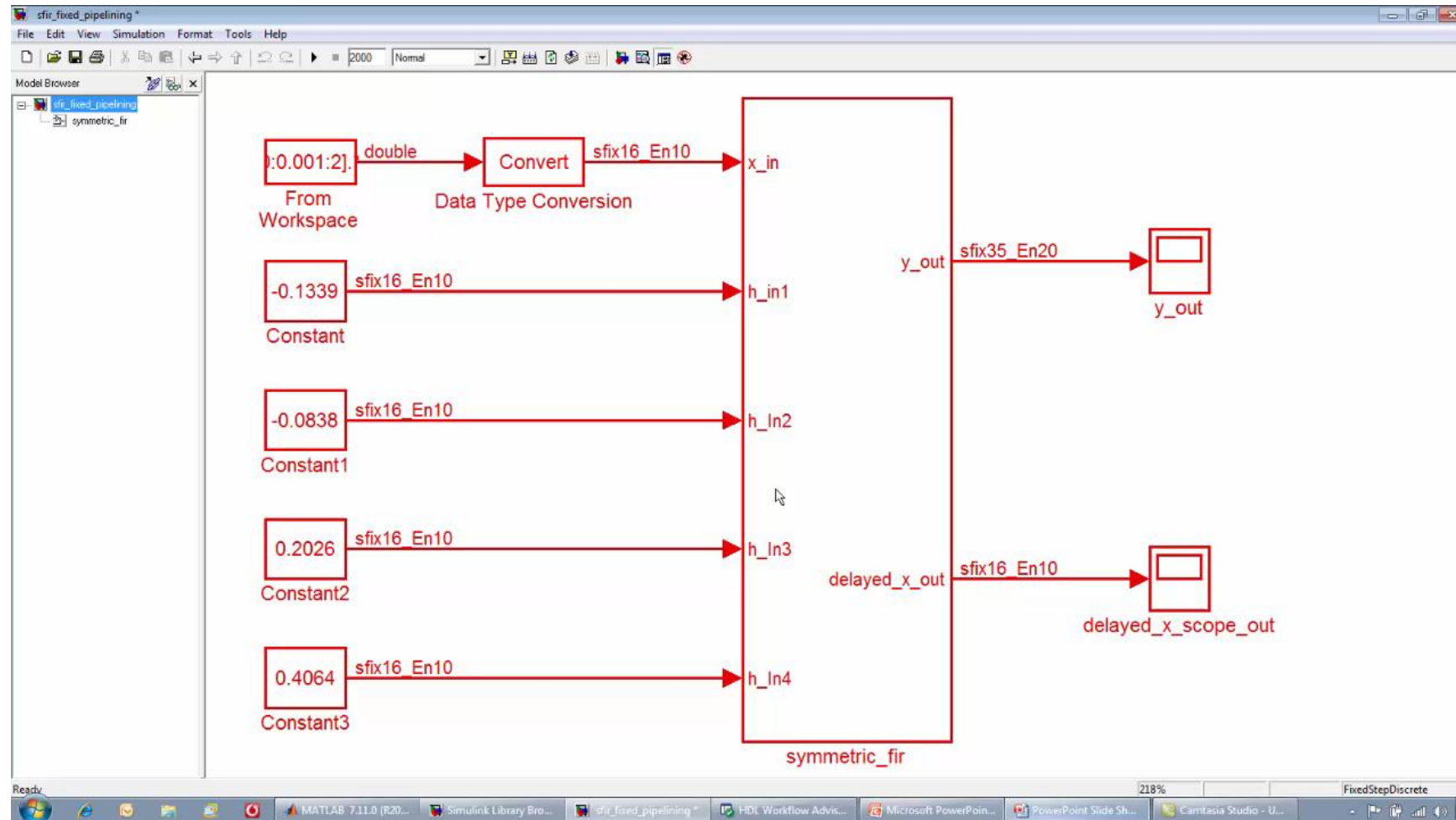
Algorithm Verification

Co-simulation for Verification of HDL Source Code



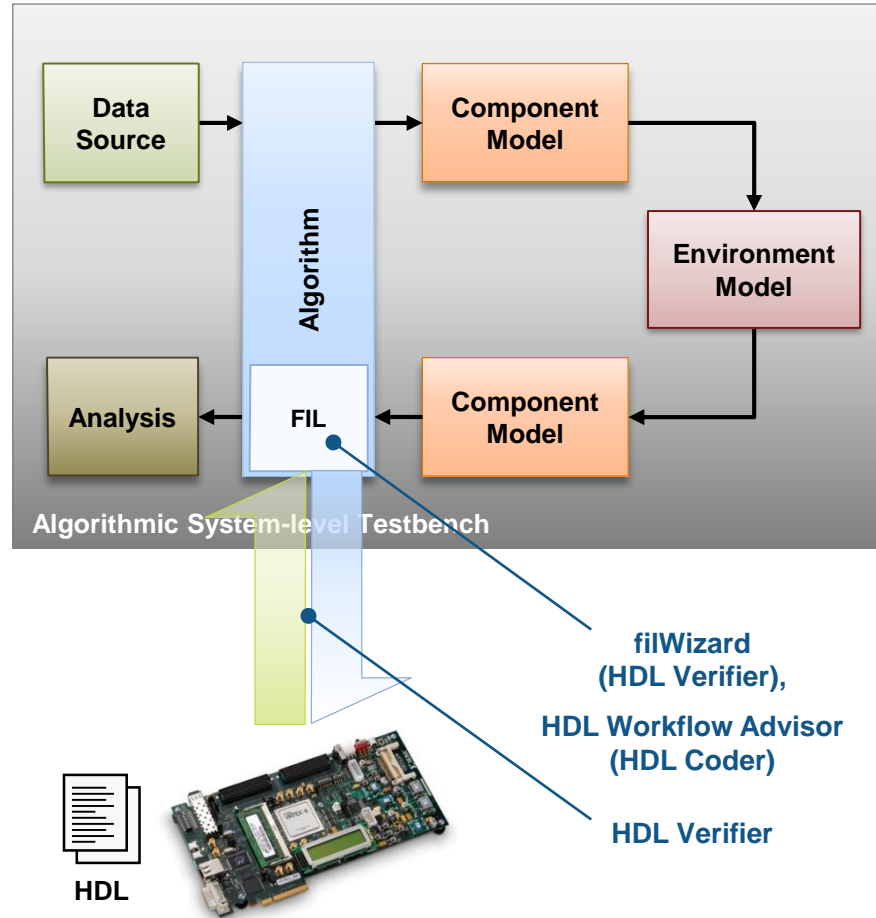
- Co-simulation with 3rd-party HDL simulator
 - Reuse of existing testbench in MATLAB/Simulink
 - HDL code execution in 3rd-party HDL simulator
 - Flexible HDL sources
 - Handwritten or generated code
 - Automated generation of co-simulation infrastructure
 - Automatic handshaking
 - Combined analysis and debugging in both simulators

Co-simulation



Algorithm Verification

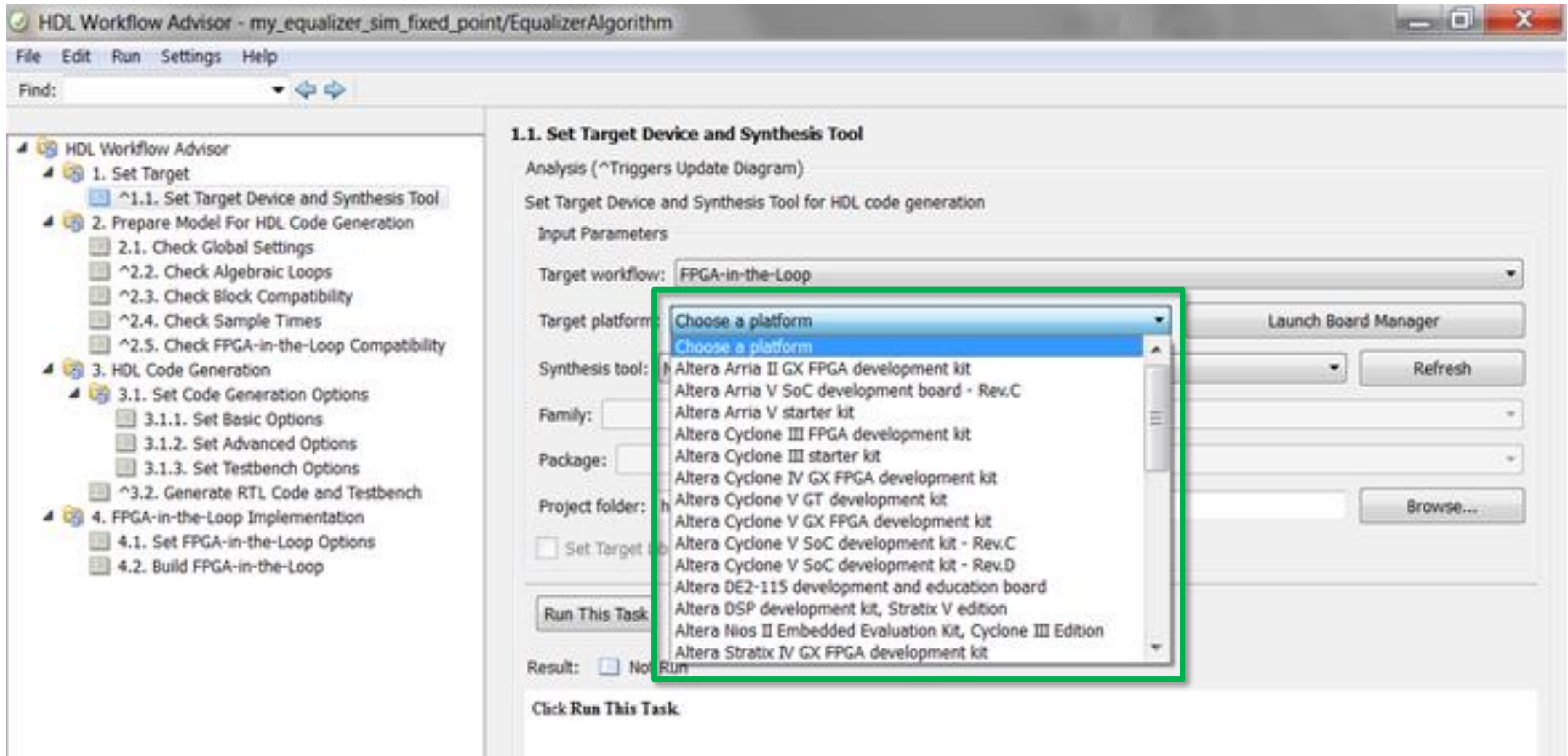
FPGA-in-the-Loop Verification of HDL Source Code



Prototype your algorithm in hardware connected to the system-level test environment

- FIL simulation with FPGA development board
 - Reuse existing testbench
 - HDL code execution on FPGA
 - Handwritten or generated HDL code
 - Automated generation of co-simulation infrastructure
 - Encapsulation of algorithm within GBit Ethernet MAC, or JTAG
 - Automatic handshaking

FPGA-in-the-Loop Target Device



The screenshot shows the HDL Workflow Advisor interface for the task "1.1. Set Target Device and Synthesis Tool". The "Target workflow" is set to "FPGA-in-the-Loop". The "Target platform" dropdown menu is open, showing a list of available FPGA development kits and boards. The list includes:

- Altera Arria II GX FPGA development kit
- Altera Arria V SoC development board - Rev.C
- Altera Arria V starter kit
- Altera Cyclone III FPGA development kit
- Altera Cyclone III starter kit
- Altera Cyclone IV GX FPGA development kit
- Altera Cyclone V GT development kit
- Altera Cyclone V GX FPGA development kit
- Altera Cyclone V SoC development kit - Rev.C
- Altera Cyclone V SoC development kit - Rev.D
- Altera DE2-115 development and education board
- Altera DSP development kit, Stratix V edition
- Altera Nios II Embedded Evaluation Kit, Cyclone III Edition
- Altera Stratix IV GX FPGA development kit

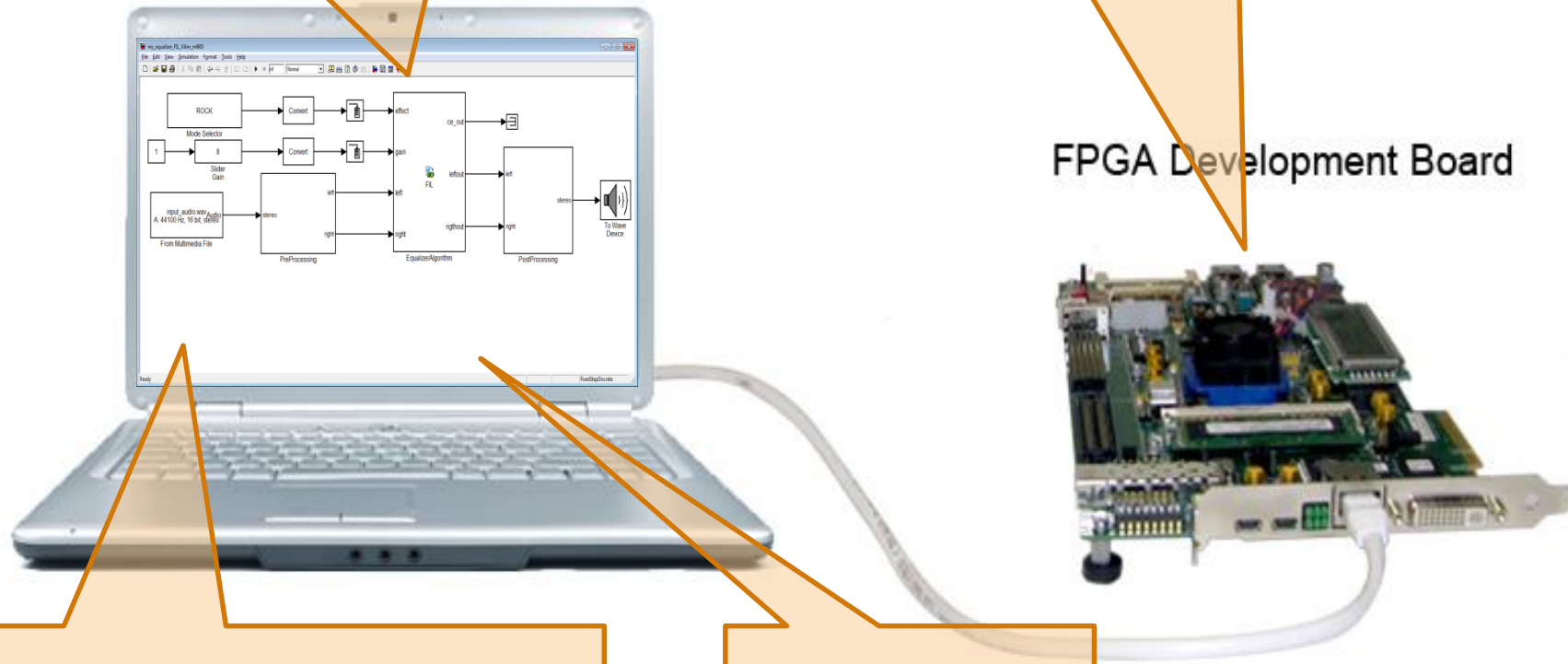
The "Run This Task" button is visible, and the "Result" is currently "No Run".

FPGA-in-the-Loop

Enable regression testing with FPGA-in-the-loop simulation

Re-use test benches for regression testing

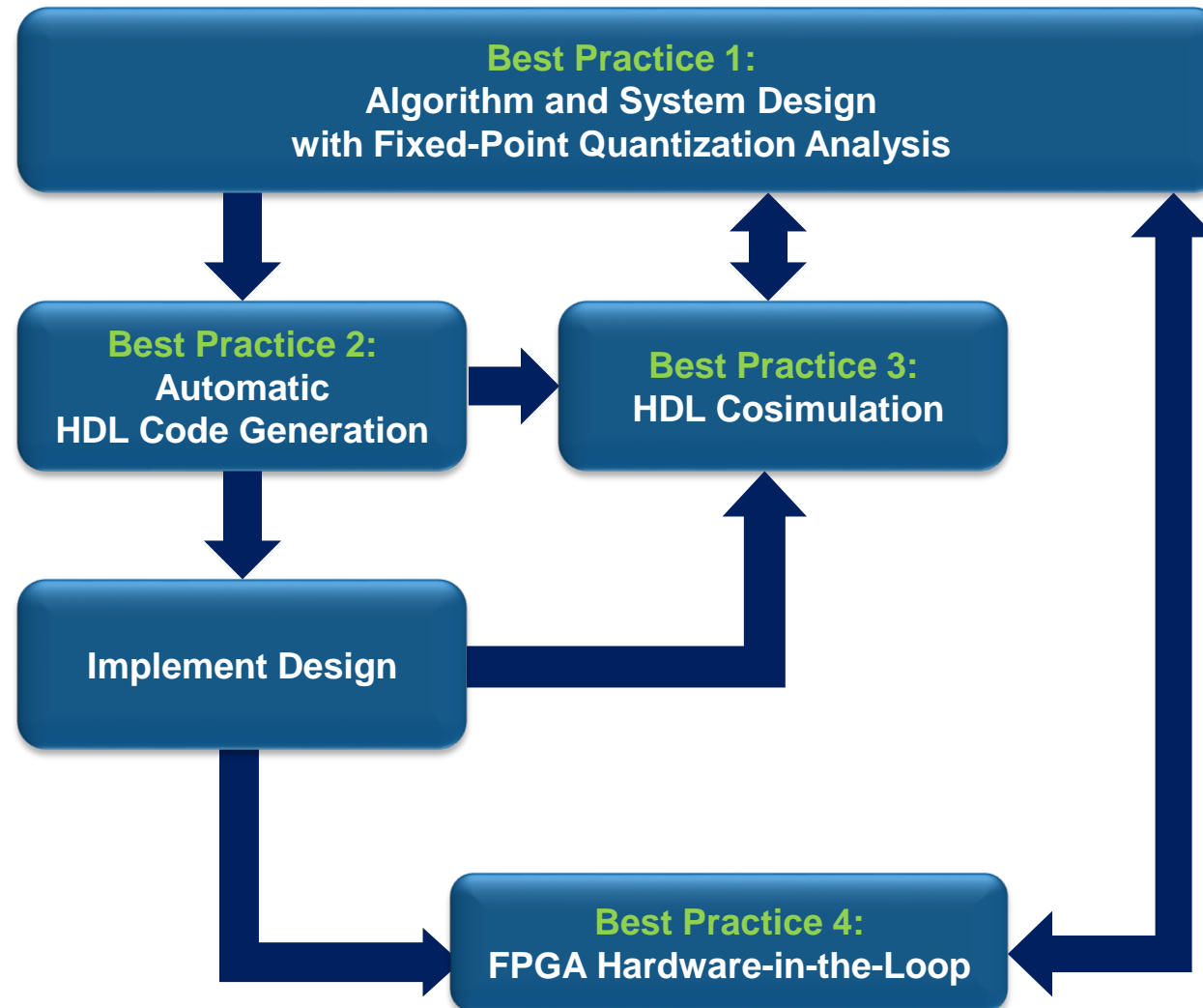
Integrate with Altera / Xilinx FPGA Development Boards



Flexible test bench creation: closed loop, multi domain

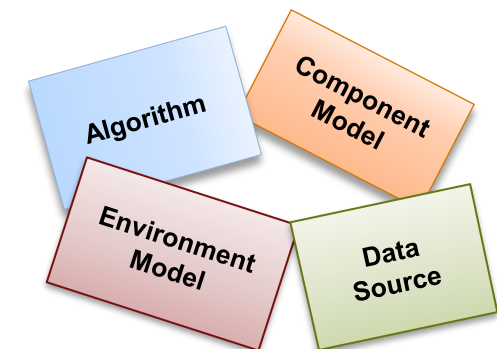
Also works with handwritten code

Key Takeaway



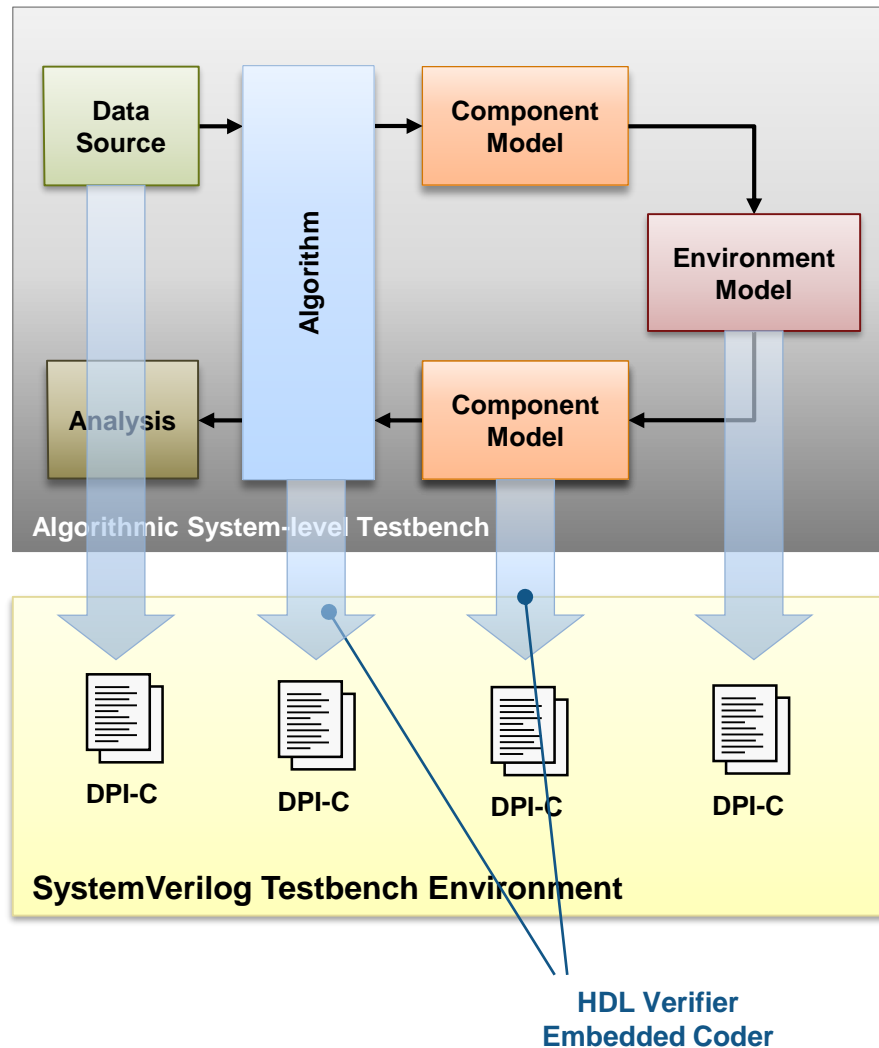
Integrating with other Verification Activities

- Verification is the single biggest cost in hardware design
 - Investment in developing simulations for verification
 - SystemVerilog and UVM test frameworks
 - SystemC/TLM virtual platforms
 - Shift towards ‘model-based’ verification
 - Enabling techniques like Constrained Random testing
- Rather than recreate a behavioural model, we can reuse the assets developed in the system models in MATLAB & Simulink
 - Maintains connection with earlier part of the flow
 - Removes risk of manual error in test framework
 - Avoids duplicating effort



System Verification

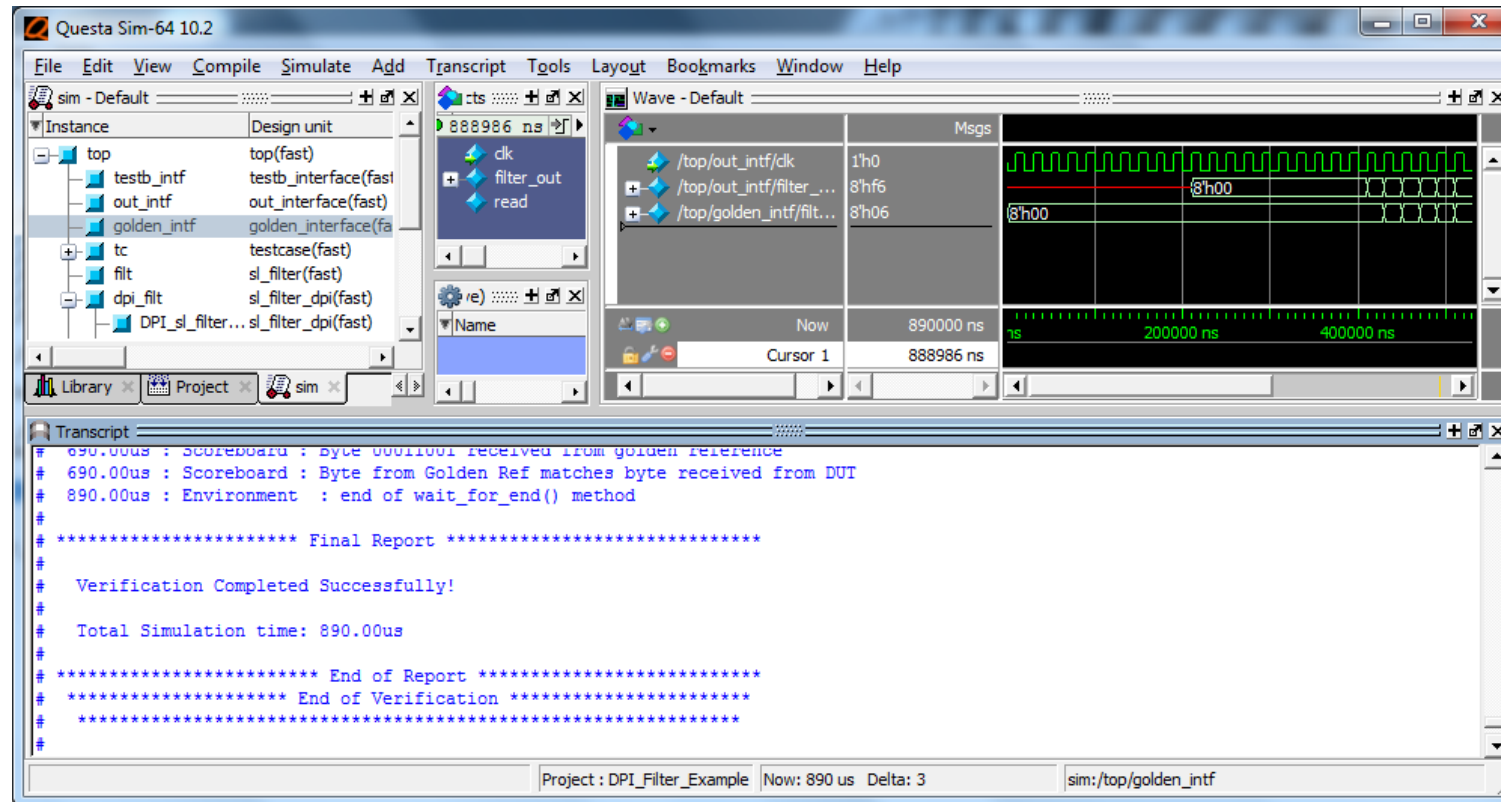
Reuse of models in SystemVerilog Testbench



- Code generation translates models to other languages (e.g. C, HDL)
 - Implementation code
 - Verification models
- For verification, C code generation is convenient
 - analog and digital models
 - Wider block and language support for C generation
- HDL Verifier extends code generation tools to provide wrappers for
 - SystemVerilog DPI-C
 - SystemC TLM

Integrating DPI-C/SV into Existing Testbench

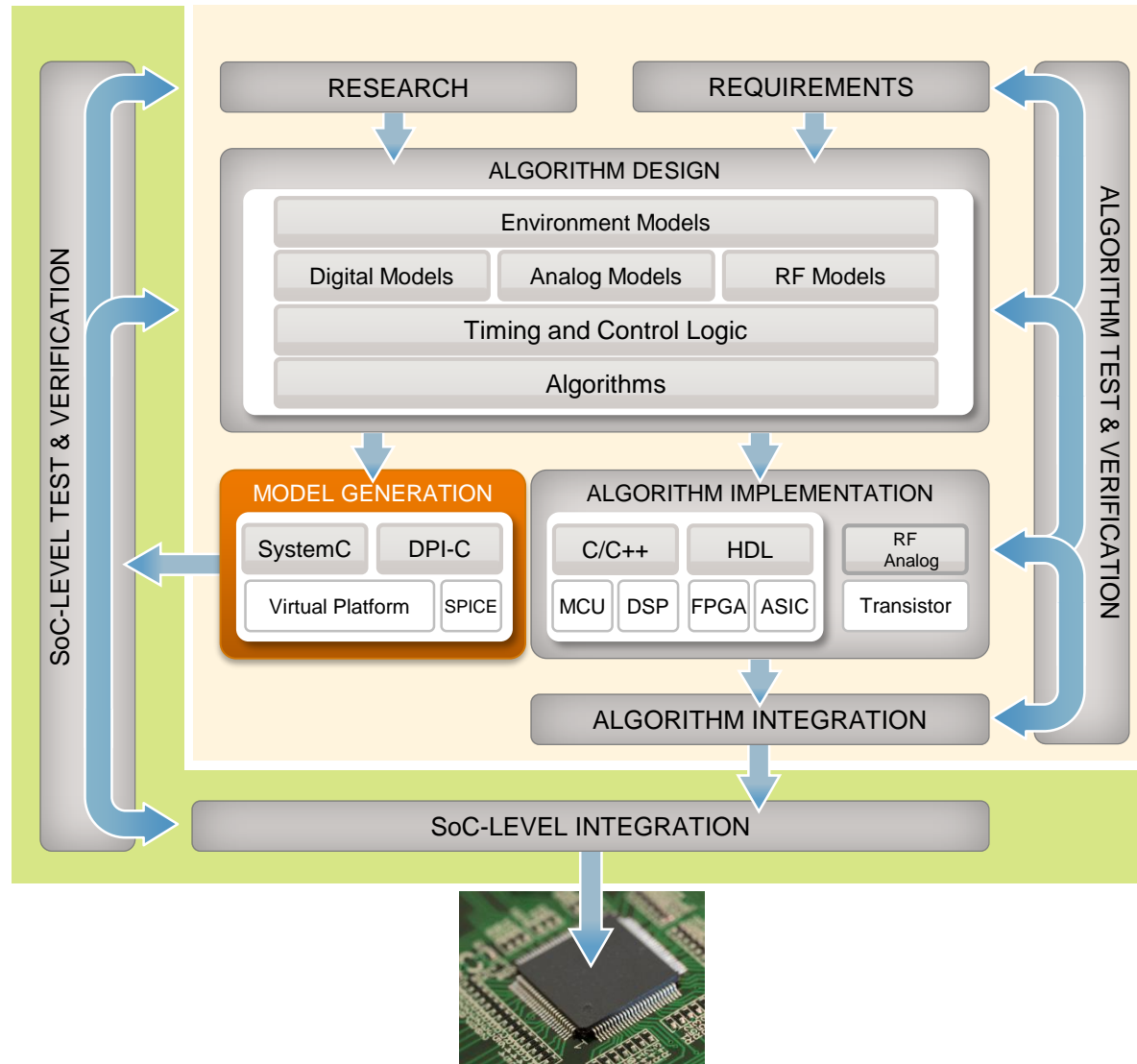
- Using a public SystemVerilog Testbench example*, adapted to execute the DPI-C as a golden reference:



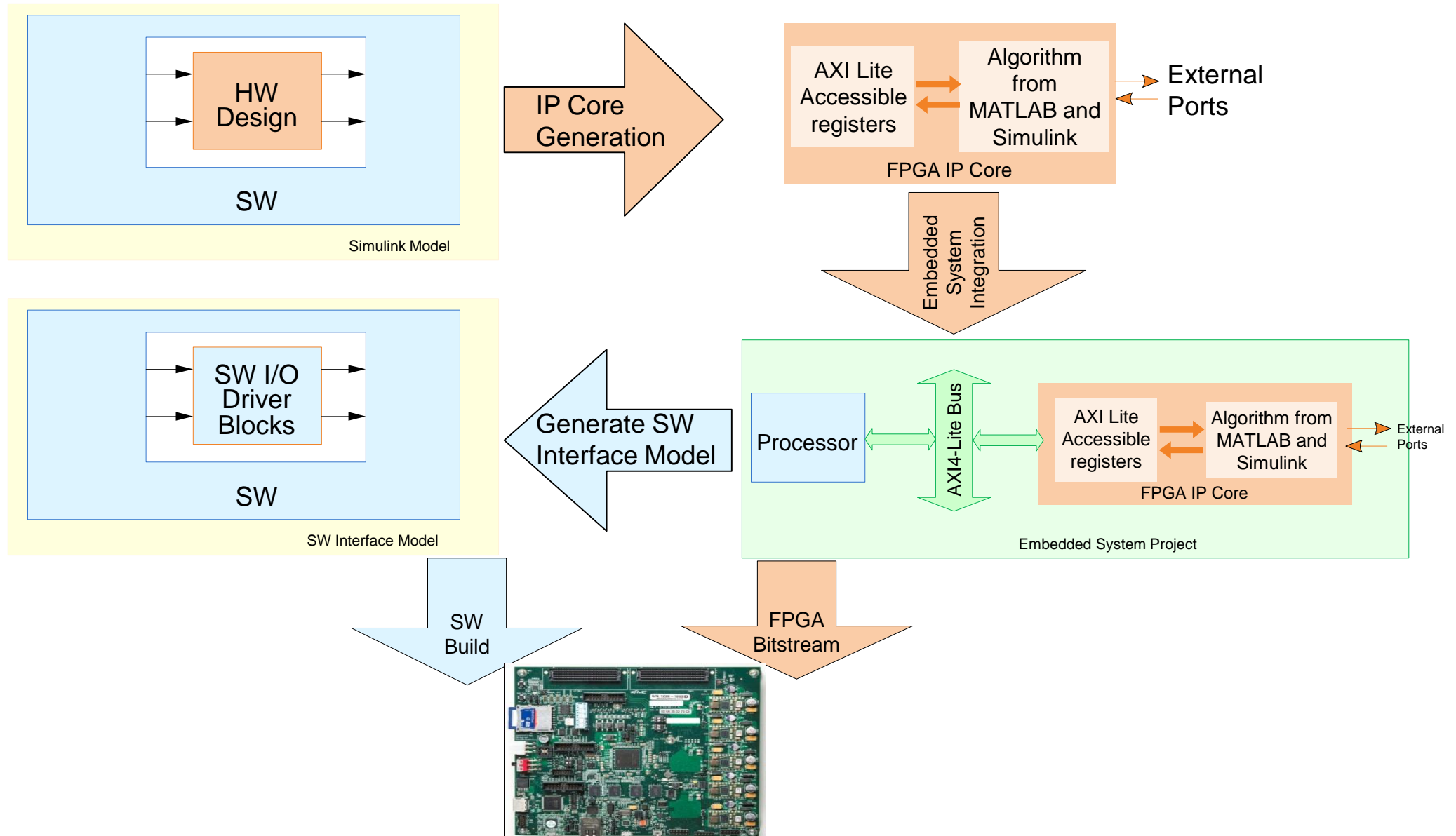
* Example from MicroElectronics Student Group at the University of Porto:
<http://wiki.usgroup.eu/wiki/public/tutorials/svverification>.

Integrated Verification

Model-Based Design and SystemVerilog/SystemC



Zynq HW/SW Co-design Workflow Summary



Summary

- Model-Based Design for FPGA
- Generating HDL Code from MATLAB and Simulink
 - For prototyping and production
 - Optimizing code for efficiency
- Verifying HDL Designs with MATLAB and Simulink
 - Co-simulation with HDL simulators
 - FPGA-in-the-Loop verification
- Verifying HDL Designs outside MATLAB and Simulink
 - Generating code for integration with SystemC/TLM and SystemVerilog/DPI-C