**Models for Virtual Platforms**

Virtual prototyping environments provide an efficient solution for creating software virtual hardware platforms for hardware architecture design and/or embedded software/firmware development. Creation of a virtual platform relies on the assembly of models of hardware devices – a.k.a. intellectual property (IP) blocks – available in the library from tool, semiconductor or IP vendor. Proprietary or third-party IP models that can’t be found in the library provided by the vendor must be created by the user or a subcontractor.

**Intel® CoFluent™ Studio** offers an alternative to the manual modeling of IP and IP-level or platform-level test cases for virtual platforms by providing an efficient graphical modeling entry and automatic device/IP model code generation. It offers significant productivity gains compared to manual programming and can be used by non-experts in modeling languages such as **SystemC** or the **Device Modeling Language (DML)**. It accelerates the availability of a complete virtual platform environment and facilitates the creation of application-realistic workloads/use cases when software is not available yet.

Graphical models are captured at functional level, and model behavior and functional timings are validated at the same level within **Intel® CoFluent™ Studio for Timed-Behavioral Modeling (TBM)**.

Accellera **SystemC Transaction-Level Modeling (TLM)** or Wind River Simics** Device Modeling Language (DML)** code can be automatically generated from the same model description. Generated SystemC code can be integrated to SystemC-based virtual prototyping environments through standard TLM-2.0 interfaces. Generated DML code can be integrated to the Simics environment through DML registers and interfaces.

![Figure 1. Device Modeling with Intel® CoFluent™ Studio](image-url)
**Graphical Functional Modeling**

Intel CoFluent Studio for TBM allows the design intent and system use cases to be captured in simple and intuitive graphical models. The system and use case behavior is represented as a network of concurrent processes (called “functions”) that act and interact to accomplish some logical end. Data and control flows between and for each function are described using a graphical language: Intel CoFluent domain-specific language (DSL). ANSI C/C++ is used as an action language for describing data types, algorithms and control flow conditional statements.

**Graphical Functional Validation**

Models captured by users are translated by default into SystemC code that is instrumented and built. Their execution generates traces that can be controlled and monitored from the Intel CoFluent Studio graphical user interface. A rich set of monitoring tools allows analyzing, understanding, debugging and validating that the system model behaves as expected and meets functional timing requirements. No particular SystemC knowledge is required at this point as models are monitored and analyzed at the Intel CoFluent DSL level.

**Modeling Libraries**

Functional models created with Intel CoFluent Studio are based on the Intel CoFluent DSL semantics that are typically higher level than software programming (C, C++) or hardware description languages (VHDL, Verilog). In order for CoFluent models to be executed in SystemC or DML, a modeling library providing the objects and classes that implement the Intel CoFluent DSL semantics is required for each target simulation environment.

Available modeling libraries are:

- Intel® CoFluent™ C++ Modeling Library for SystemC/TLM2 (SCL)
- Intel® CoFluent™ C++ Modeling Library for Simics (SIML)

**Code Generators**

Intel CoFluent Studio provides code generators for translating models captured by the user into executable models targeting each simulation environment. The generated code uses objects and classes of the associated modeling library for the target simulation environment.

Available generators are:

- Intel® CoFluent™ C++ Generator for SystemC/TLM2 (SCG)
- Intel® CoFluent™ C++ Generator for Simics (SIMG)

---

**Figure 2.** Validating the functional model of a simple Direct Memory Access (DMA) component.
TLM2 Device Model Integration

Intel CoFluent models captured by users are purely functional and platform-independent. Users can add specific TLM2 request and response communication channels so the functional model can communicate using standardized TLM2 protocols. TLM2 target and initiator wrappers are automatically generated by Intel CoFluent Studio to convert TLM2 transactions into Intel CoFluent DSL messages. Supported protocols are TLM-2.0 LooselyTimed (LT) and Approximately Timed (AT).

As Intel CoFluent Studio’s native simulation environment is based on SystemC, external SystemC IPs can be directly integrated into the graphical model for simulating the device model as part of an external SystemC-based virtual platform.

DML Device Model Integration

Once a device model has been created and validated by simulation within Intel CoFluent Studio, it can be taken to the Simics simulation environment and integrated to a virtual platform model.

A device model for Simics requires a new modeling element called Device Programmer’s View (DPV) that defines the software programming interface of the hardware device. DPV allows driving the simulation of a device model like a device driver would do: Whereas a “hardware testbench” directly accesses the inputs/outputs (regular CoFluent DSL communication channels) of the device’s hardware behavior model (called “functional core”), a “software testbench” interacts with the device’s behavior model through its DPV elements using MethodCall Interfaces (MCI). DPV elements include registers, connected/implemented interfaces. Each element has read/write callbacks that can be triggered by the MCI from the testbench. The DPV and device’s functional core model are connected by regular CoFluent DSL relations or MCI.

Generated code for Simics includes the C++ functional core, based on the objects and classes of the SIML, a C/DML functional interface, and DML/C wrappers translated from DPV that enable interfacing to specific DML interfaces (e.g., memory-mapped registers, USB, Ethernet, SATA, etc.). Those wrappers provide methods that can be called both ways to pass data between the DML interface and the generated C++ functional core. DML interfacing and integration of the device model to the virtual platform has to be done manually by the user from within Simics.

Table 1. Supported platforms

<table>
<thead>
<tr>
<th>Intel® CoFluent™ Studio</th>
<th>Windows*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated Development Environment (IDE)</td>
<td></td>
</tr>
<tr>
<td>Simulation</td>
<td>Accellera SystemC* 2.3 Windows, Linux*</td>
</tr>
<tr>
<td>Wind River Simics* 4.8</td>
<td>Windows, Linux</td>
</tr>
</tbody>
</table>

Figure 3. Integrating the IP to a SystemC* TLM-2 virtual platform
To learn more about CoFluent, visit cofluent.intel.com

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit www.intel.com/performance/resources/limits.htm or call (U.S.) 1-800-628-8686 or 1-916-356-3104.

All dates and products specified are for planning purposes only and are subject to change without notice.

Relative performance for each benchmark is calculated by taking the actual benchmark result for the first platform tested and assigning it a value of 1.0 as a baseline. Relative performance for the remaining platforms tested was calculated by dividing the actual benchmark result for the baseline platform into each of the specific benchmark results of each of the other platforms and assigning them a relative performance number that correlates with the performance improvements reported.

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel’s Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Copyright © 2014 Intel Corporation. All rights reserved. Intel, the Intel logo, and Intel CoFluent are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others. Printed in USA 0914/VP/OCG/XX/PDF Please Recycle 328107-002US.

---

**Figure 4.** Device model with functional core and Device Programmer’s View (DPV) connected to “software” testbench

**Figure 5.** Generated code structure for Wind River Simics®