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 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 interfaces.

Figure 1. Device Modeling with Intel® CoFluent™ Studio
**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)

---

**Figure 2. Validating the Functional Model**
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)

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 Loosely Timed (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.

Generated code for Simics includes a C++ functional core, based on the objects and classes of the SIML, as well as C/DML wrappers to enable interfacing the device model’s functional core to a specific DML interface (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. Template models with predefined integration code are available for most common Simics interfaces.

Figure 3. Integrating the IP to a SystemC* TLM-2 Virtual Platform
**Table 1. Supported platforms**

<table>
<thead>
<tr>
<th>Integrated Development Environment (IDE)</th>
<th>Windows*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silicon Compiler and third-party SystemC* 2.2</td>
<td>Windows, Linux*</td>
</tr>
<tr>
<td>Wind River Simics* 4.6</td>
<td>Windows, Linux</td>
</tr>
</tbody>
</table>

Figure 4. Generated code structure for Wind River Simics*

To learn more about CoFluent, visit cofluent.intel.com