

Intel® CoFluent™ Methodology for SysML*



UML* SysML* MARTE* Flow for Intel® CoFluent™ Studio

An Intel® CoFluent™ Design White Paper By Thomas Robert and Vincent Perrier
www.cofluent.intel.com

Acronyms and abbreviations

BDD	Block Definition Diagram
CPU	Central Processing Unit
DSL	Domain-Specific Language
HW	Hardware
IBD	Internal Block Diagram
IC	Integrated Circuit
IP	Intellectual Property
MARTE*	Modeling and Analysis for Real-Time and Embedded systems
SW	Software
SysML*	Systems Modeling Language
TLM	Transaction-Level Modeling
UML*	Unified Modeling Language

Abstract

SysML is a UML profile that allows the creation of standard descriptions of a system. However, this profile is too generic to address embedded and real-time system design. The MARTE UML profile attempts to fill this gap by providing elements from both embedded software and hardware engineering. Unfortunately, it remains mainly descriptive in nature, since no commercial tools are available to simulate the models and extract performance data.

Intel® CoFluent™ technology offers methodology for SysML to provide comprehensive framework and guidelines for joint use of SysML and MARTE. The methodology offers simulation of multicore/multiprocessor hardware/software embedded system and chip models, enabling designers to observe the system behavior and analyze performance properties. The methodology delivers modeling rules and method with tool support.

Intel® CoFluent™ technology's tool support includes:

- Intel® CoFluent™ technology UML profile extending UML 2.4, SysML 1.3 and MARTE 1.0 profiles
- Integration with leading UML modeling environments

- Link to Intel® CoFluent™ technology's Intel® CoFluent™ Studio SystemC 2.3-based simulation environment for model execution and extraction of performance figures

The link to Intel® CoFluent™ Studio is achieved by model transformation from SysML to Ecore-based internal model description. The methodology for SysML complies with Intel® CoFluent™ Studio's embedded system architecting flow and the MARTE profile's intent that separates the application or functional view from the execution platform view. The execution platform view is often called the hardware resource view.

Hardware/software partitioning is described in a mapping or allocation view, and the resulting allocated view represents the actual embedded software threads executing on the various cores and operating systems that constitute the hardware and firmware. The resulting allocated view is a "virtual system", since it encompasses the full hardware/software system.

Existing virtual platform and virtual prototype environments require the assembly of detailed intellectual property block models. Intel® CoFluent™ technology's virtual system modeling and simulation technology overcomes many of the limitations of virtual platforms since it can be executed before detailed hardware intellectual property block models and embedded software are available. Thus, it removes the inherent limitations due to the availability of the models or their important development time and associated cost. The Intel® CoFluent™ technology also goes beyond traditional UML simulation that does not take into consideration architectural and non-functional performance dimensions such as thread priorities and scheduling, time constraints, bus transactions, memory accesses, power consumption, memory footprint, cost, etc. Virtual systems provide fast and accurate evaluation of various use cases and design scenarios by executing SysML specifications and predicting the behavior, performance and power consumption. Accurate prediction is critical for multicore and low-power designs.

Intel® CoFluent™ Studio with SysML support enables system designers to store and exchange design information internally and between third parties in a standard format. It allows the delivery of executable specifications and SystemC test cases for further system validation with SystemC-based virtual platform environments.

» Prerequisite:

The reader is assumed to have a basic knowledge of UML, SysML and MARTE technologies in order to fully understand this white paper.

» Acknowledgement:

SysML models shown in screenshots were captured with No Magic's MagicDraw* tool.

Table of Contents

1	Introduction.....	3
2	Model Packaging	5
3	Application Modeling.....	7
3.1	Structure.....	7
3.2	Behavior.....	11
3.3	Intel® CoFluent™ Application Profile	12
3.4	Tools: UML Modelers and Intel® CoFluent™ Studio Integration	12
3.4.1	MagicDraw*	12
3.4.2	Papyrus*	13
4	Platform Modeling.....	16
5	Allocation.....	18
5.1	Allocated Model Execution in Intel® CoFluent™ Studio	19
6	Conclusion	21

Table of Figures

Figure 1: Package diagram of the model	6
Figure 2: BDD of the application Leaf functions and relations appear as properties in the lowest-level containers.....	8
Figure 3: Leaf functions defined in Functions package.....	8
Figure 4: IBD of the application	9
Figure 5: Port type definition.....	10
Figure 6: SysML activity diagram.....	11
Figure 7: Intel® CoFluent™ Studio with integrated MagicDraw	13
Figure 8: Intel® CoFluent™ Studio behavioral model simulation results	14
Figure 9: Blocks defined in Components package.....	16
Figure 10: Composite diagram of the platform.....	17
Figure 11: Assignments for application to platform mapping	18
Figure 12: Allocated model simulation timeline	19
Figure 13: Performance profiling windows in Intel® CoFluent™ Studio	20

1 Introduction

This paper presents the Intel® CoFluent™ methodology for SysML and associated tool support. The methodology is based on UML with SysML and MARTE profiles. It allows automatic model transformation from this format to the Intel® CoFluent™ technology DSL.

The methodology can be seen as a set of modeling rules and restrictions applied to SysML/MARTE. These rules enable the simulation of multicore/multiprocessor hardware/software system models for behavioral and performance data prediction, without restricting the expressiveness of SysML and MARTE. The Intel® CoFluent™ methodology for SysML does not add new elements to standard profiles. It does, however, offer an optional profile to represent several features related to Intel® CoFluent™ DSL and necessary for efficient embedded system modeling and simulation.

An “Audio Video Player” example model is used to illustrate the design flow. This model is a high-level description of an audio-video decoder running on a generic platform. The generic platform consists of a processor and hardware accelerator.

» Throughout this paper, the following typographic rules are applied:

- Specific UML / SysML / MARTE words are displayed in a different font. Example: `block`.
- Names of model elements (`property names`, `block names`, `package names...`) and `profile` elements are displayed in italic. Example: *Functions*.

2 Model Packaging

The model packaging reflects the separation between application and platform models. The application or functional model includes elements of the environment generating stimuli for system simulation under specific use scenarios or use cases. The application model is mapped onto the platform model to add generic physical constraints. The result is a refined virtual system model.

- Functional architecture describes applicative structures with timed behaviors. Functions execute in full parallelism. They synchronize through events and exchange data.
- Physical architecture describes a generic execution platform, including processing units such as a CPU or IC, physical links such as bus or routing network, storage units including memories, and firmware such as schedulers. In this view, the platform is “empty” in the sense that CPUs execute no instructions and ICs are not yet wired with logic.
- Virtual system architecture is the result of the mapping step, which consists in allocating functions to processing units, data to storage units, and routing inter-processor data on physical links. The mapping or allocation can be seen as a model itself. The obtained virtual system model is the functional architecture with execution constraints such as scheduling, bus arbitration and memory size.

Packaging is also important to maximize both reuse and model readability. Figure 1 shows the proposed package hierarchy.

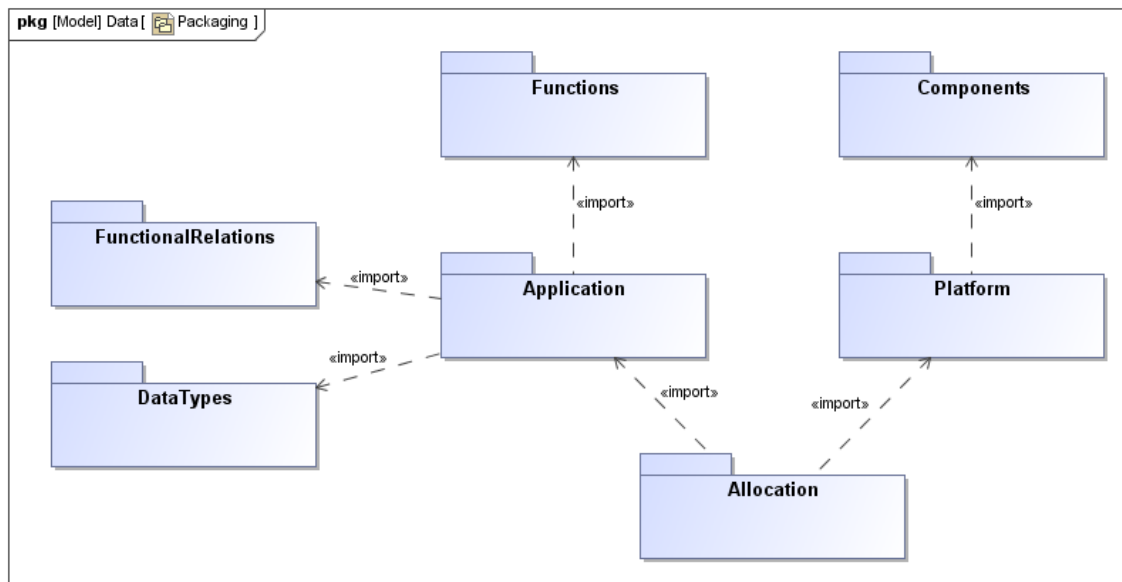


Figure 1: Package diagram of the model

The package named *Functions* contains functions of the application model. These may be reused in other models. *ComponentsLib* contains components of the platform model, which also may be reused in other models. *Application* and *Platform* packages contain models themselves. The *Allocation* package contains the allocation model, using elements from application and platform models.

Two other `packages` can be defined containing the data types and the functional relations (data communication and synchronization elements between functions). It is merely illustrative packaging, since only *Application*, *Platform* and *Allocation* `packages` are mandatory.

3 Application Modeling

The application model is a set of functions exchanging data and/or synchronizing through events, shared variables and message queues. Functions can be internal to the system under design or pertain to the environment and thus, contribute to exercising the chosen use case scenario by interacting with the system under a stimuli/responses scheme. There are two different aspects in an application model:

- The structural aspect encapsulates the different functions and the relations, or data paths, between them.
- The behavioral aspect is the control flow used to process data and synchronize functions.

3.1 Structure

Functions encapsulating other functions are called “container functions”. Functions defined at the lowest hierarchical levels are called “leaf functions”.

A SysML internal block diagram (IBD) is used to represent the applicative structure of the system. A top-level SysML block is used as a root container.

Functions of the lower hierarchical level are all properties of the top-level blocks. Blocks appearing in the *Application* package, or in an imported package, define the types of these properties. Sub-functions of these blocks are also represented as properties, and so on.

The block definition diagram (BDD) in Figure 2. shows the functional hierarchy. This representation is not required in order to correctly represent the Intel® CoFluent™ technology model: an IBD is sufficient. However, a BDD aids in understanding functional hierarchy in the application, particularly the distinction between leaf and container functions. Moreover, a BDD is compliant to existing SysML modeling methodologies and most of the designers feel more comfortable having both a BDD and an IBD.

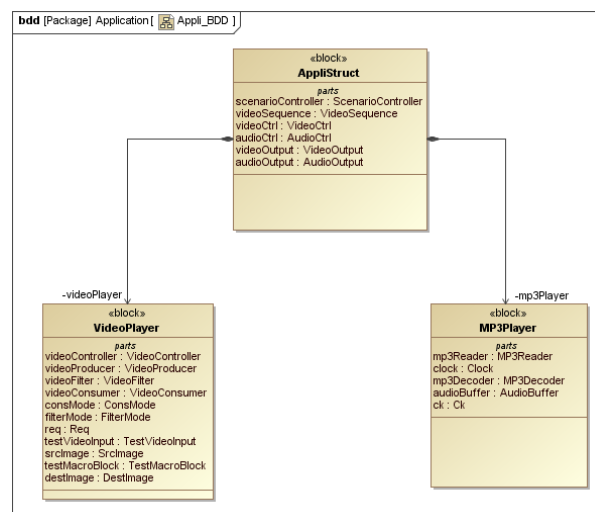


Figure 2: BDD of the application
Leaf functions and relations appear as properties in the lowest-level containers

Figure 3 shows the leaf functions, which are the lowest hierarchical level functions defined in *Functions* package of the SysML model.

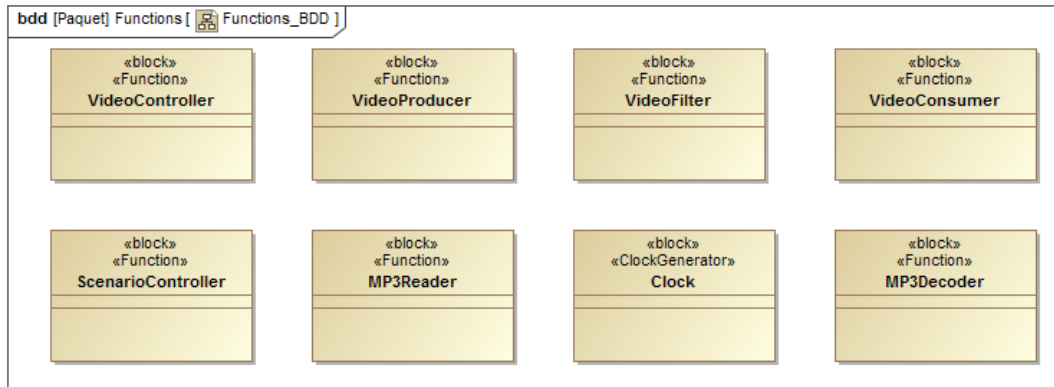


Figure 3: Leaf functions defined in Functions package

Using these blocks as types for the container functions `properties` enables reuse, as other `properties` could be typed with these blocks. The blocks can be reused in the same model or in other application models.

The connections between application elements are represented in the IBD of the top-level block, named *AppliStruct* for the example as shown in Figure 4.

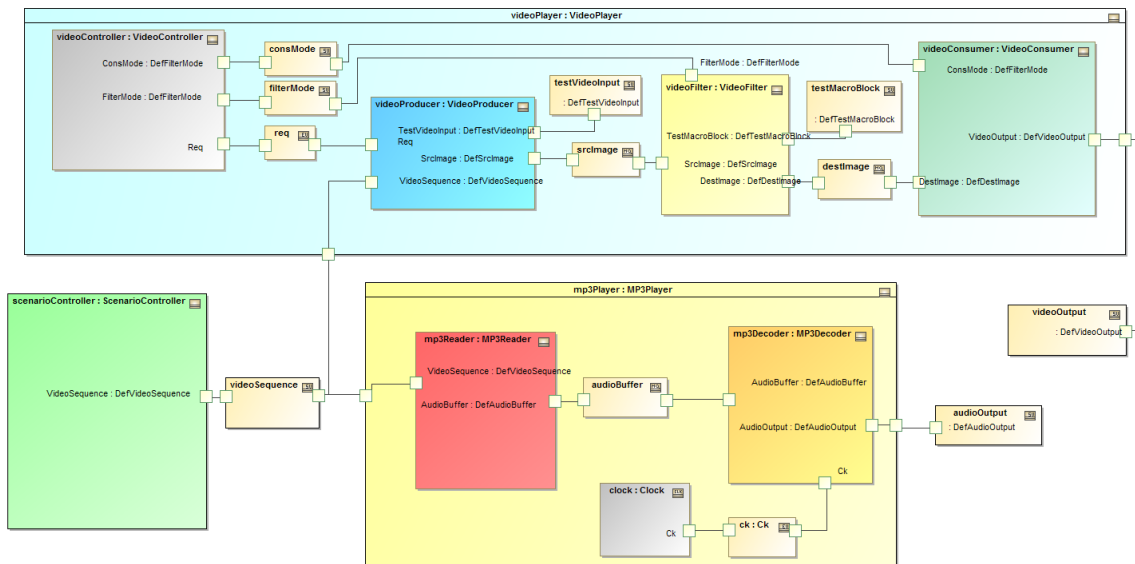


Figure 4: IBD of the application

Relational elements are represented by properties with two input and output ports. The types of these properties are SysML blocks defined in the *FunctionalRelations* package with the following stereotypes from MARTE profile:

- NotificationResource: Event enabling synchronization between functions. Equivalent to Intel® CoFluent™ technology “event”.

- `SwMutualExclusionResource` with `SharedDataComResource`: Shared variable enabling asynchronous data read/write. Equivalent to Intel® CoFluent™ technology “shared variable”.
- `MessageComResource`: Enables synchronous buffered data exchanges. Equivalent to Intel® CoFluent™ technology “message queue”.

There are assembly connectors between ports of different properties. Delegation connectors are used for ports forwarding between external and internal functions.

Data types are defined by blocks in the *Application* package or imported from other packages. The data types are carried by the ports. Ports are MARTE* FlowPorts, with directions in, out, inout.

Figure 5 shows the type of the selected flow port. Here, the data type *DefAudioBuffer* is a block defined in the *DataTypes* package.

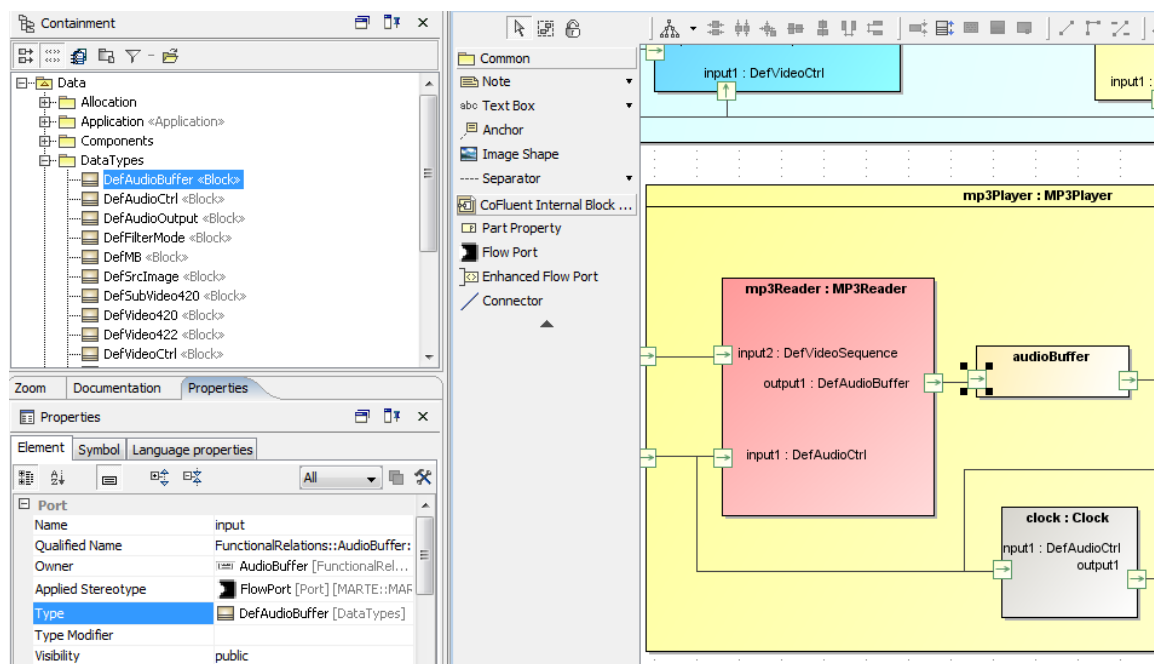


Figure 5: Port type definition

As shown on the IBD, application is composed of three main parts:

- *scenarioController*, shown in the bottom left corner, provides the *videoSequence* and controls execution of video and MP3 players.
- *videoPlayer*, shown at the top, decodes *videoSequence*.
- *mp3Player*, shown at the bottom, decodes the MP3 encoded sound contained in *videoSequence*.

» Important note about stereotypes application:

Models described in this document make an extensive use of MARTE stereotypes. It is also possible to apply optional Intel® CoFluent™ technology stereotypes.

Stereotypes applied to blocks may be also applied to properties typed by these blocks. Stereotypes can even be applied only to properties. For instance, it is possible to apply a stereotype to a property of a container function in the application model, independently from the stereotype applied to the block defined in the function library as type of the property.

If both block and property are stereotyped with the same stereotype, tagged values from the property will be used when transforming to the Intel® CoFluent™ model.

3.2 Behavior

Leaf function types are blocks that can contain a behavior. An activity diagram can describe this behavior.

Placing the activity under the block in the hierarchy of the SysML model is sufficient to express that the activity is representing the behavior of the block. However, this link can be explicitly described by setting the ClassifierBehavior property of the block.

Figure 6 depicts an example of an activity, *MP3DecoderAct*.

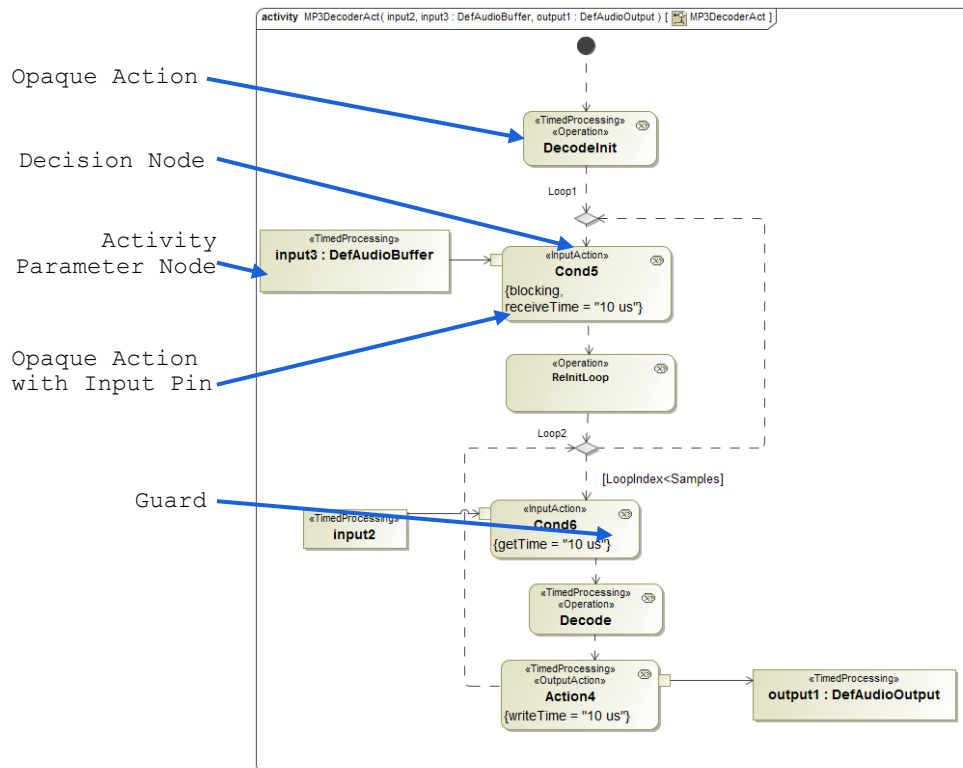


Figure 6: SysML activity diagram

Inputs and outputs of the activity are activity parameter nodes, allocated to

corresponding ports of the application structure via the MARTE `Allocate` stereotype.

Input and output pins allow read/write on ports in behaviors. Opaque actions model operations to be performed, for example data read/write and processing. Decision node/fork and join/fork and merge can be used in the control flow description. In Figure 6, the first decision node represents an infinite loop, whereas the second is a loop with condition, where $LoopIndex < NbSamples$.

For opaque actions, the MARTE stereotype `TimedProcessing` can be used in order to set operation duration. C/C++ is used as action language. Code can also be added in opaque action body for SystemC-based simulation within Intel® CoFluent™ Studio after model transformation.

3.3 Intel® CoFluent™ Application Profile

Use of the Intel® CoFluent™ profile is optional. The *Application* package of the profile allows accessing the following specific features.

- Specific model element attributes including read and write times for relations and specific Intel® CoFluent™ functions such as sampled function, clock generator, divider, message routing function.
- Specific C/C++ code: global declarations, pre-processing and post-processing code, local declarations for each function, and type definitions.
- Design parameters: they are used to change values in the model at simulation time.

There are four stereotypes for design parameters: *EnumerationDP* for enumerations, *SetDP* for sets, *RangeDP* for ranges and *VariableDP* for variables.

In order to add a design parameter to the SysML model, the parameter must first be defined with a block to which the right stereotype is applied. Then, attribute values of the parameter can be set. Lastly, a property must be added in the block to which the parameter is applied. The type of this property is the generic parameter block previously defined. If the generic parameter is global, a property typed by the generic parameter block must be added to the root block of the application.

3.4 Tools: UML Modelers and Intel® CoFluent™ Studio Integration

3.4.1 MagicDraw*

The Intel® CoFluent™ *Application* profile and associated Intel® CoFluent™ technology plugin are currently available for No Magic's MagicDraw UML modeling tool.

The SysML/MARTE application model can be captured in MagicDraw and then imported from Intel® CoFluent™ Studio. MagicDraw can also be directly integrated into Intel® CoFluent™ Studio as an Eclipse* plugin, with one-click integration for the user, in order to work within a single modeling environment.

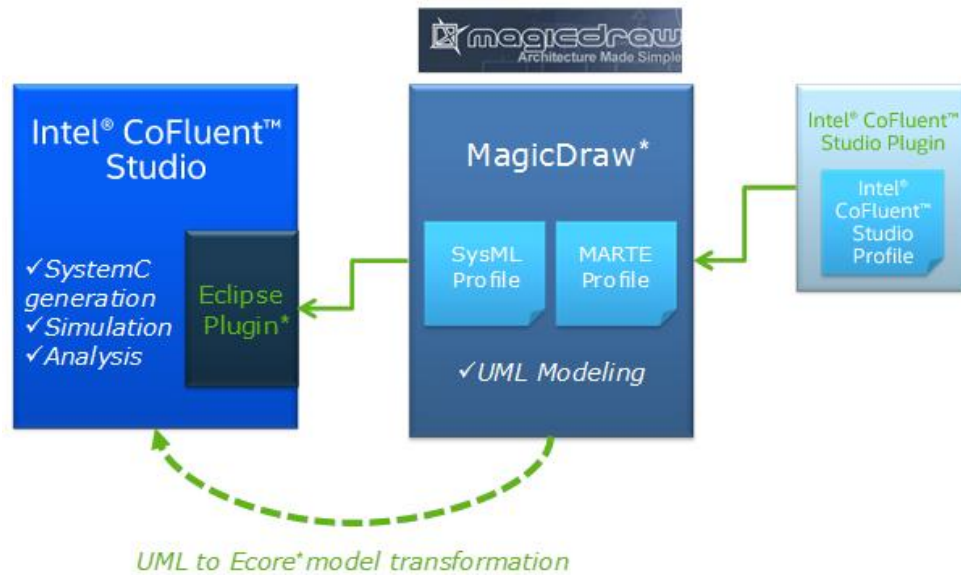


Figure 7: Intel® CoFluent™ Studio with integrated MagicDraw

MagicDraw customization has been developed to ease the modeling guidelines application and the Intel® CoFluent™ profile usage. A MagicDraw validation module for Intel® CoFluent™ technology is available to ensure that the SysML/MARTE model is compliant to the methodology and can be transformed into an Intel® CoFluent™ model. All these elements are part of the Intel® CoFluent™ plugin for MagicDraw.

3.4.2 Papyrus*

The Intel® CoFluent™ profile, which includes *Application*, *Platform* and *Allocation* packages, is also available for the Papyrus UML modeling environment. However, a Papyrus to Intel® CoFluent™ Studio model transformation is not commercially available.

3.5 Application Model Execution in Intel® CoFluent™ Studio

The SysML/MARTE application model can be automatically transformed into an Intel® CoFluent™ model. The transformation is implemented with Java and the Eclipse Modeling Framework (EMF). The output is a tool-compatible model compliant with Intel® CoFluent™ meta-model. The meta-model in turn conforms to Ecore, Eclipse's Meta-Object Facility (MOF).

In Intel® CoFluent™ Studio, the obtained Ecore* model is used to automatically generate a SystemC TLM model that includes the user-defined C/C++ code. The generated code uses Intel® CoFluent™ SystemC 2.3 and TLM 2.0-based simulation library. SystemC models can be generated for various environments such as OSCI SystemC library, CoWare, Synopsys Innovator*, Synopsys System Studio*, and Mentor Graphics Questa*. This feature enables reuse of Intel® CoFluent™-generated SystemC models as test cases for further verification and validation within a wide range of virtual platform environments.

Once the SystemC model is generated and compiled, the model can be run to extract behavioral and timing information. Deadlocks or conflicts in data accesses can be identified. Latencies can be measured. It is also possible to change generic parameter values and see how the changes impact model execution.

Figure 8 shows the model simulation results in Intel® CoFluent™ Studio.

» Current tool support:

Intel® CoFluent™ Studio v5.1 supports application modeling and simulation with SysML. Platform model and mapping are captured using the Intel® CoFluent™ DSL.

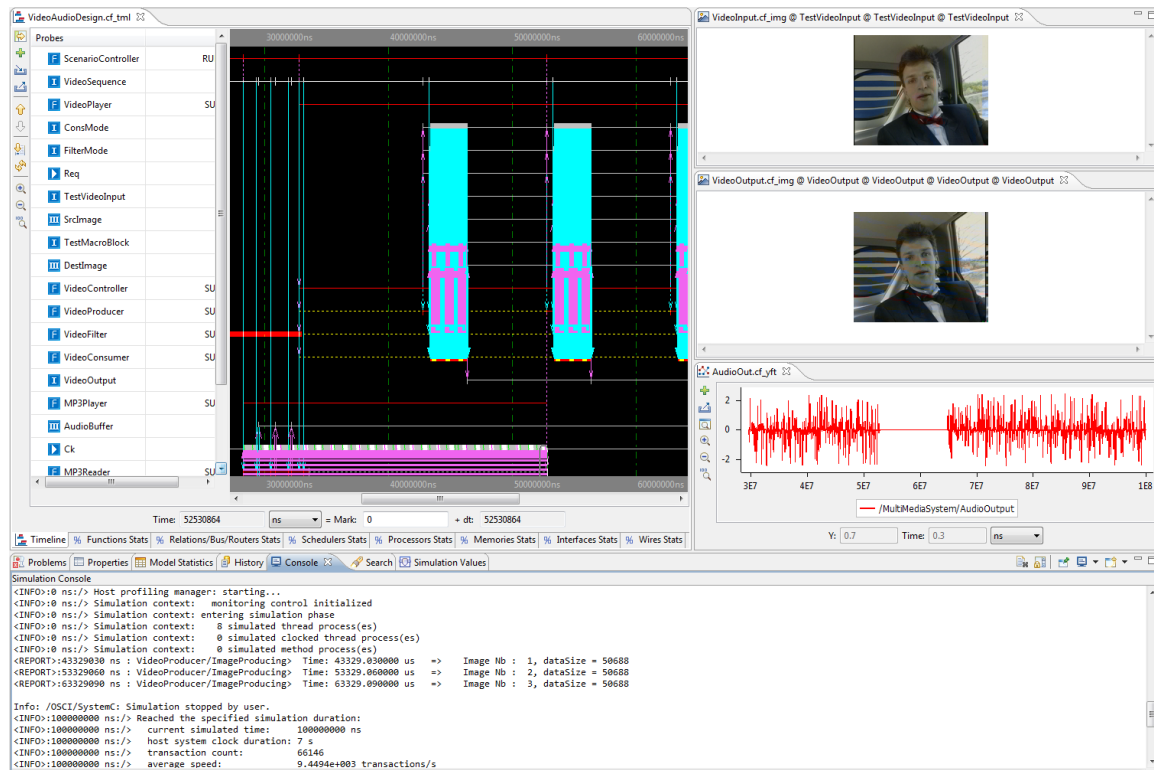


Figure 8: Intel® CoFluent™ Studio behavioral model simulation results

The user monitors the activity of the different functions, as well as the data exchanges between the functions, in the timeline window. With this feature, the user can clearly recognize the messages that are sent to the two players. The different functions and message queues related to the MP3 player are represented in the lower part of the window. The video player is located in the upper part.

The Intel® CoFluent™ Studio timeline is similar to a UML sequence diagram displayed horizontally (time runs from left to right). It offers additional capability as well. The state of each function, running, blocked, and idle, can be observed and analyzed in detail. Computation and data read/write transaction durations can be precisely observed since the model execution is fully timed.

The Console view displayed below the timeline in Figure 8 is used to print text information during the simulation. Just as software programmers that can add print statements in their code for printing information during the program execution, users can also monitor and print out information during the simulation execution in this window.

Image processing on the right side of the screen in Figure 8 shows the different steps of image processing through the video channel. The first picture represents the input picture that is processed and the second one the *videoOutput*. It is also possible to plot variables against the execution time to monitor their values, such as audio output level of the *mp3Player* in the window shown in the lower right section of the screen.

4 Platform Modeling

The platform is a set of computation and storage resources connected by physical links. It is represented using `Hardware Resource Model` from MARTE profile.

The platform structure is described by an `IBD`. A top-level SysML block is used as a root container.

Components of lower hierarchical levels are properties of the top-level blocks. Types of these properties are defined by MARTE stereotyped blocks, appearing in the *Platform* package or in an imported package. Sub-functions of these blocks are also represented as properties, and so on.

Schedulers can be allocated to MARTE `HwProcessor` using the MARTE `allocate` stereotype. The schedulers are defined as blocks stereotyped with MARTE `GaExecHost`.

Figure 9 depicts blocks defined in the imported *Components* package. The MARTE stereotype used for each one is visible. The package contains computation resources (*UserComputer*, *CoProcessor*), communication resources (*Bus*, *USB*, *SPI*), memories (*DisplayEnv*, *Source*, *Mem*) and a scheduler (*Scheduler*).

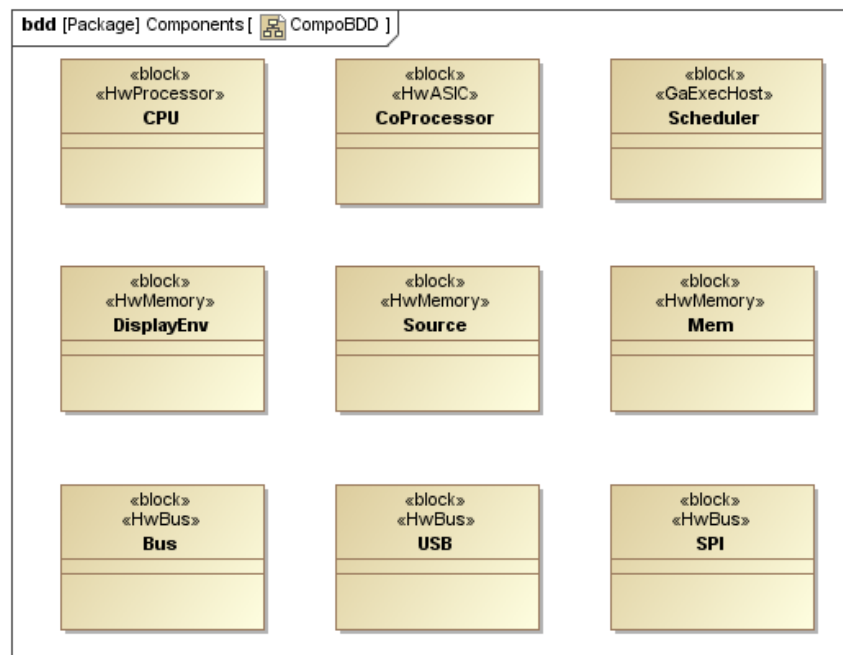


Figure 9: Blocks defined in Components package

Figure 10 depicts the IBD of the platform. Ports are MARTE hwEndPoints. Assembly connectors are used to connect end points to buses. Delegation connectors are used for ports forwarding between different hierarchical levels. Schedulers are allocated to hardware processors using Assign stereotype from MARTE.

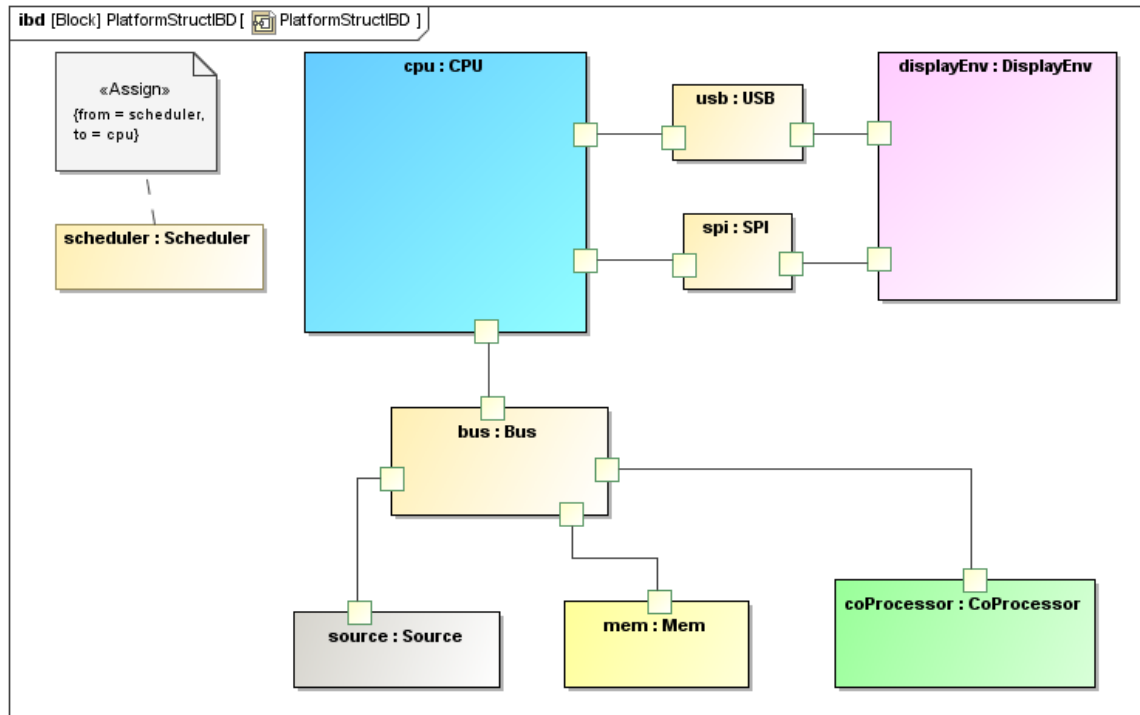


Figure 10: Composite diagram of the platform

5 Allocation

To obtain an allocated virtual system model, functions of the application are mapped onto computation resources of the platform, while shared variables and exchanged data are mapped onto memories and physical links. *Assign* from MARTE profile is used to model this mapping or allocation.

Software mapping is achieved by assigning functions from the application to tasks (*blocks* stereotyped with MARTE *schedulable resource*). The schedulers of the platform schedule tasks.

Hardware mapping is done by directly assigning functions to ICs of the platform.

Link mapping is the allocation of functional relations to physical links or memories of the platform.

Figure 11 shows the different assignments for the example model.

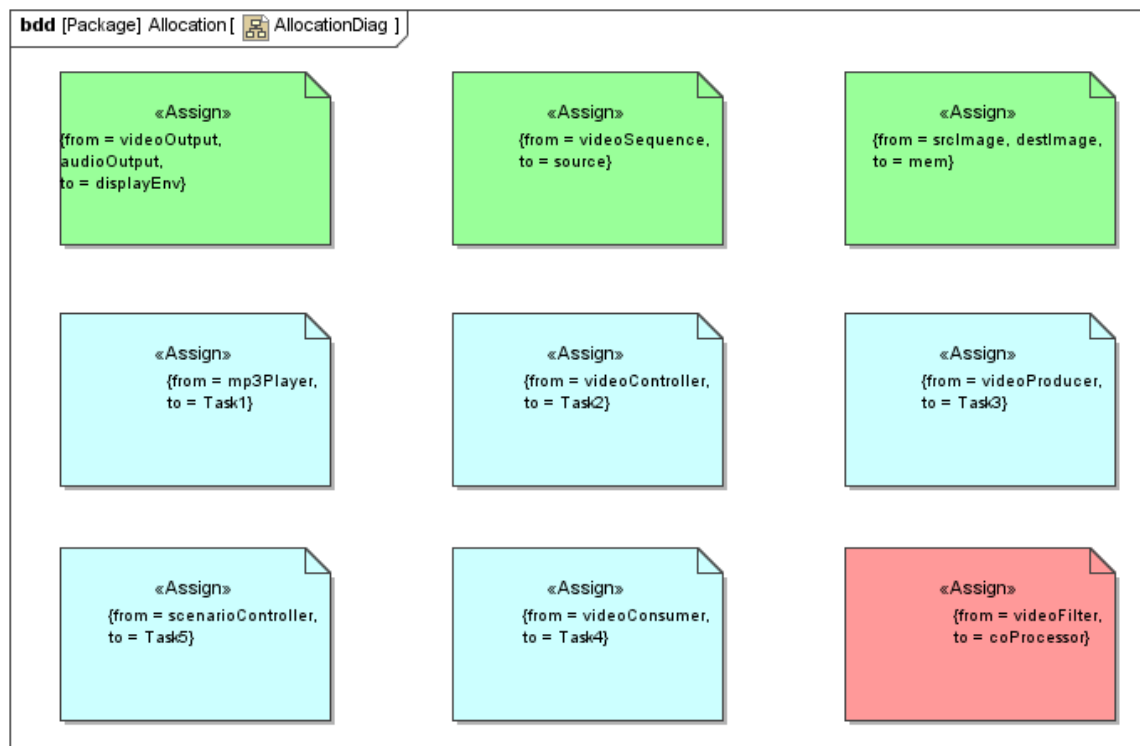


Figure 11: Assignments for application to platform mapping

5.1 Allocated Model Execution in Intel® CoFluent™ Studio

Figure 12 illustrates the obtained timeline for the allocated, or architecture, model simulation in Intel® CoFluent™ Studio. The descriptions were created with Intel® CoFluent™ DSL.

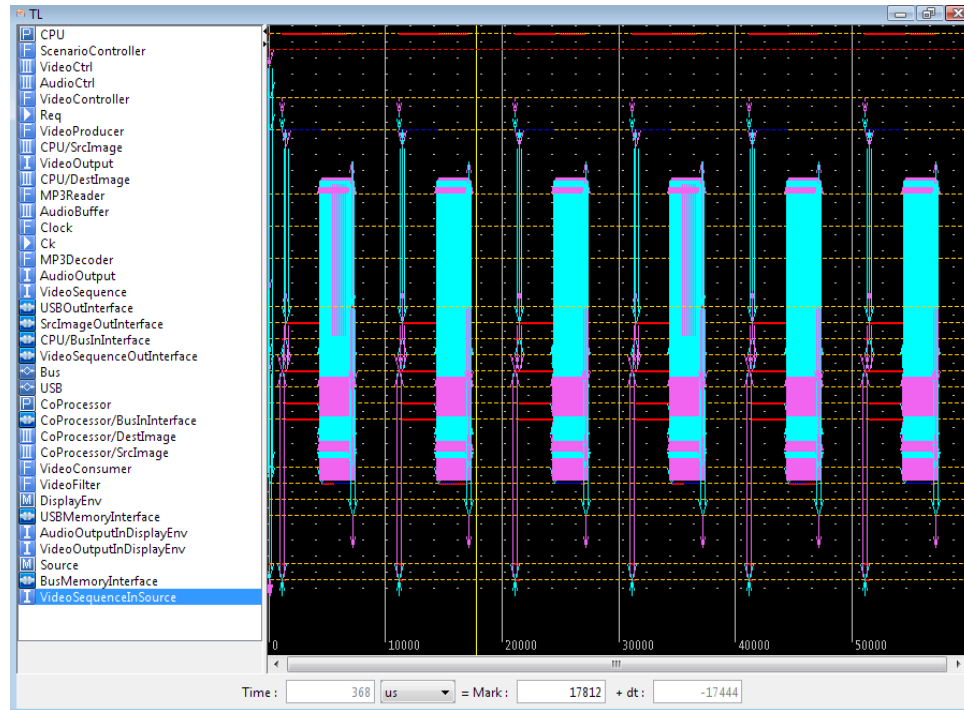


Figure 12: Allocated model simulation timeline

Unlike results available for the behavioral simulation, the model execution results include the activity of the processors, busses and interfaces. This feature is useful to evaluate if resources of the system under study are overloaded. It also will help determine if the software partitioning is adequately done. Additionally, the ability of a software-processing unit to perform several tasks and the impact of a bus to transfer can be monitored.

Figure 13 shows the performance profiling windows obtained at the end of the simulation. The first window displays a global results table, and the second window displays the dynamic load of selected resources.

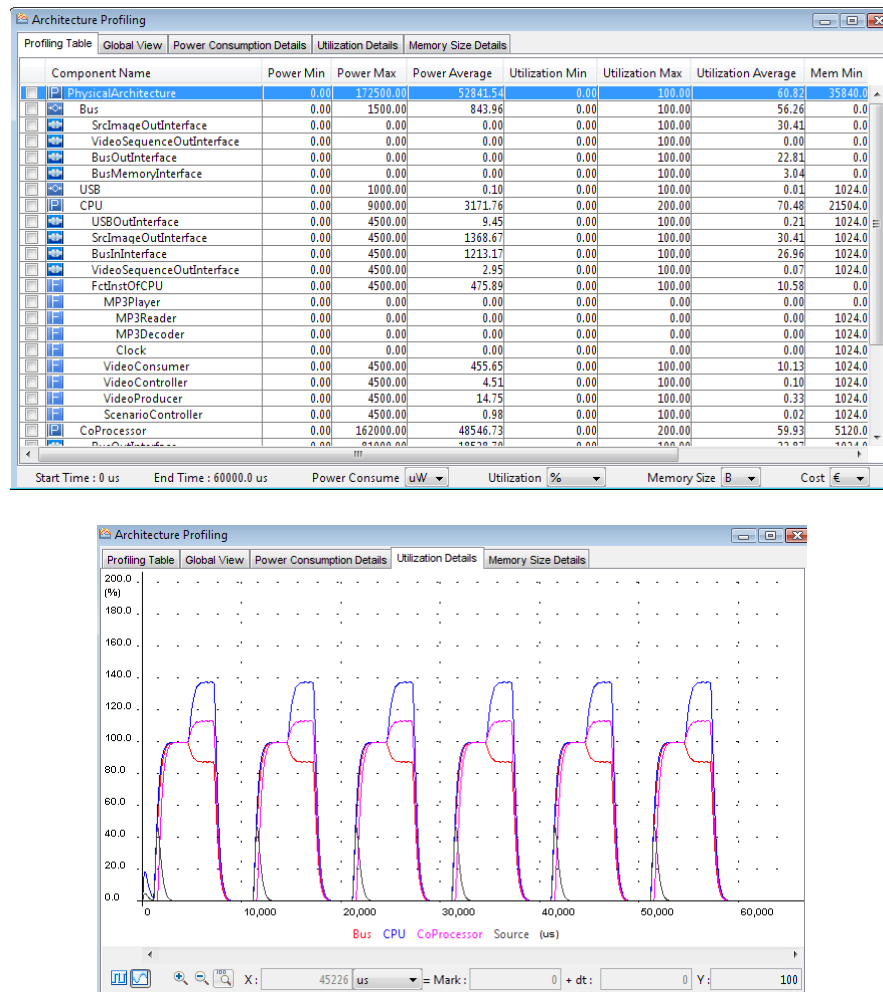


Figure 13: Performance profiling windows in Intel® CoFluent™ Studio

Performance profiling results include power, resource utilization (loads in percentage of Mcycles/s), and memory. For instance, the simulation indicates a *CPU* use of 70.84% and a *Bus* use of 56.26%.

If the *CoProcessor* was not used and all the functions were mapped onto the *CPU*, the *CPU* utilization would increase and the bus utilization would decrease. This kind of mapping change is easily done in Intel® CoFluent™ Studio, enabling fast “what if” analysis.

6 Conclusion

This paper describes an embedded system architecting flow that utilizes virtual system models obtained from SysML/MARTE specifications. The specifications are translated into executable SystemC transaction-level models. Virtual system technology overcomes many of the limitations of virtual platforms, since it is accessible when specialized hardware and software IP models or code are not yet available. It provides the simulation capability for performance analysis, behavioral and architectural verification, and use cases and tradeoff analysis.

The use of standard notations of SysML/MARTE profiles to describe virtual systems enables system architects to store and exchange design information internally and between third parties in a standard format. Furthermore, virtual systems allow fast and accurate evaluation of different use cases and design scenarios by allowing the execution of SysML specifications. Automatic SystemC TLM code generation allows reusing models in other SystemC-based virtual platform and verification environments.

Intel Corporation

Robert Noyce Building
2200 Mission College Blvd.
Santa Clara, CA 95052-8119
USA

Intel Corporation SAS

Les Montalets
2 rue de Paris
92196 Meudon cedex
France
Phone: +33 158 877 115
Fax: +33 158 877 000

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, Intel CoFluent Studio, and Intel CoFluent Reader 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 0612/VP/OCG/XX/PDF Please Recycle 327566-001US