# A Platform-Based Taxonomy for ESL Design

**Douglas Densmore**
University of California, Berkeley

**Alberto Sangiovanni-Vincentelli**
University of California, Berkeley

**Roberto Passerone**
University of Trento

*Editor's note:*
This article presents a taxonomy for ESL tools and methodologies that combines UC Berkeley's platform-based design terminologies with Dan Gajski's Y-chart work. This is timely and necessary because in the ESL world we seem to be building tools without first establishing an appropriate design flow or methodology, thereby creating a lot of confusion. This taxonomy can help stem the tide of confusion.

*—Gary Smith, Gartner Dataquest*

**THE GROWTH OF THE EDA INDUSTRY** has been less than satisfactory in the past few years. For example, in 2005 growth was only 0.6%,[1] and in 2006 it is predicted to be less than 3%.[2] The reasons are varied and are beyond the scope of this article. However, one of the main issues is the failure of EDA to address new customers. New customers imply a revenue potential that is not consuming present business, thus allowing real industry growth. Traditionally, EDA has served the IC industry, where the demand for tools has been rampant since the early 1980s. An obvious adjacent market for EDA growth is electronic system-level (ESL) design. (See the "Trends affecting the ESL design market" sidebar for a brief history and explanation of how various market factors have contributed to developments in ESL design.)

The 2004 *International Technology Roadmap for Semiconductors* (*ITRS*) placed ESL "a level above RTL," including both hardware and software design. The *ITRS* defined ESL to "consist of a behavioral (before HW/SW partitioning) and architectural level (after)" and claimed it would increase productivity by 200,000 gates per designer-year. The *ITRS* states that ESL will improve productivity by 60% over an "Intelligent Testbench" approach—the previously proposed ESL design improvement.[3] Although these claims cannot yet be verified and seem quite aggressive, most agree that ESL's overarching benefits include

- raising the abstraction level at which designers express systems,
- enabling new levels of design reuse, and
- providing for design chain integration across tool flows and abstraction levels.

The purpose of this article is to paint the ESL design landscape by providing a unified framework for placing and analyzing existing and future tools in the context of an extensible design flow. This approach should help designers use tools more efficiently, clarify their flow's entry and exit points, and highlight areas in the design process that could benefit from additional tools and support packages. This framework is based on *platform-based design* concepts.[4,5] Using this framework, we've classified more than 90 different academic and industrial ESL offerings and partitioned the tool space into metaclasses that span an ideal design flow. (Although we try to cover as much of the ESL tool space as possible, we make no claim of completeness. We apologize in advance to the authors of tools we have inadvertently ignored. Also, we don't analyze the extensive literature that describes these tools; rather, we identify Web sites that contain relevant information.)

We used this framework to explore three design scenarios to demonstrate how those involved in ESL design at various levels and roles can effectively select tools to accomplish their tasks more efficiently than in a traditional IC design flow. The ability to study design scenarios goes beyond mere classification, because our framework exposes the relationships and constraints

### Trends affecting the ESL design market

The number of electronic system-level (ESL) designers is reportedly several orders of magnitude larger than the number of IC designers. However, until the late 1990s, the system-level design market had been highly fragmented. Consumers were unwilling to pay a high price for tools, so EDA companies produced relatively simple tools. For most of the products in this market, the end product's complexity was not a limiting factor.

In the late 1990s, the situation began to change dramatically as system complexity reached an inflection point with the appearance of increasingly powerful electronic devices. Demand increased for demonstrably safe, efficient, and fault-tolerant operation of transportation systems such as automobiles and airplanes. Demand also increased for greater functionality in IT and communication devices, such as computing equipment and cell phones. During the past 10 years, several recalls (consider those from BMW and Daimler-Chrysler alone in the past two years, for example) and delays in the launch of previously announced products in the consumer electronics sectors demonstrated that new design methods, tools, and flows were sorely needed to prevent expensive fixes in the field and to bring new products to the market more quickly and reliably.

This situation created the conditions for the birth of new tool companies and new offerings in established EDA companies to address the needs of a changing market. However, because the system industry landscape is very diverse—with companies varying as widely as Nokia and General Motors, Boeing and Otis Elevators, and Hewlett-Packard and ABB—a design approach that could satisfy all these diverse needs would have required a large investment, with a high risk of failure. Hence, the bulk of the ESL design effort (with a few notable exceptions) has come from academia and some small start-up companies trying to address a subset of the many problems and geared toward a limited number of potential customers.

For years, Gartner Dataquest has predicted dramatic growth in ESL tool revenues, which unfortunately has failed to materialize. One of the reasons for unrealized growth is the lack of a vision in EDA of what system-level design ought to be and of how various tools fit in an overall methodology that the system industry at large could satisfactorily adopt. Consequently, there is confusion about the very definition of ESL and about what role it could play in the overall design of electronic products. Some companies have adopted ESL methodologies and tools, developed either internally or in academic circles, integrating some commercial tools as well. However, we are certainly at a relatively early stage of adoption.

among different classes to the designer, who may wish to implement a specific integration flow. (The "Related work" sidebar discusses other efforts to categorize ESL design approaches.)

## The ESL classification framework

The design framework shown in Figure 1 is based on the platform-based design (PBD) paradigm presented by Sangiovanni-Vincentelli and Martin.[5] This framework treats the design process as a sequence of steps that repeat themselves as the design moves from higher abstraction levels to implementation. The primary structure is a Y shape; thus, it is similar to the famous Y-chart introduced by Gajski. The left branch expresses the functionality (*what*) that the designer wishes to implement; the right branch expresses the elements the designer can use to realize this functionality (*how*); and the lower branch identifies the elements the designer will use to implement the functionality (the *mapping*).[6] In this context, the right branch is the *platform*, and it includes

- a library of elements, including IP blocks and communication structures, and composition rules that express which elements can be combined and how; and
- a method to assess the quantities associated with each element—for example, power consumed or time needed to carry out a computation.

Each legal composition of elements from the platform is a *platform instance*. Mapping involves selecting the design components (choosing the platform instance) and assigning functionality parts to each element, thus realizing the complete functionality, possibly with overlaps. Designers optimize this process according to a set of metrics and constraints defined from the cost figures provided, or quantities mentioned. The designers then use these metrics to evaluate the design's feasibility and quality.

This view of the design process is basically an abstraction of a process that designers have used implicitly for years at particular abstraction levels. For exam-

## Related work

We are not the first to realize the importance of categorizing ESL design approaches. Smith and Nadamuni used two axes for this purpose.[1] The first axis contains three methodology components: an algorithmic methodology, a processor and memory methodology, and a control-logic methodology. Each refers to the way in which a designer thinks about the design or its components. The second axis includes the abstraction levels to express the designs: behavioral, architectural, and platform based. Smith and Nadamuni examined approximately 50 approaches in this framework.

Maniwa presented a similar approach, also based on two axes, to categorize industrial tools.[2] The first axis is the design style: embedded software, SoC (hardware), behavioral, or component. The second axis is the language (for example, C, C++, or Verilog) to describe the design. Maniwa examined approximately 41 approaches.

Gries also used two axes to classify ESL tools developed in academia and industry.[3] The axes in this case related to abstraction levels (for example, system level and microarchitectural level) and design stages (such as application, architecture, and exploration). Gries examined approximately 19 approaches.

Finally, Bailey, Martin, and Anderson provided a comprehensive set of taxonomies: a model taxonomy, a functional-verification taxonomy, a platform-based design taxonomy, and a hardware-dependent software taxonomy.[4] To the best of our knowledge, their book provides the best classification of high-level design tools, and we follow its definitions when appropriate. Compared to their approach, our paradigm places tools in a more general design context and gives guidelines on how to connect the available tools, and IP blocks and their models, in a design flow.

### References

1. G. Smith and D. Nadamuni, “ESL Landscape 2005,” Gartner Dataquest, 2005.
2. T. Maniwa, “Focus Report: Electronic System-Level (ESL) Tools,” *Chip Design*, Apr./May 2004, http://www.chipdesignmag.com/display.php?articleId=23&issueId=4.
3. M. Gries, “Methods for Evaluating and Covering the Design Space during Early Design Development,” *Integration: The VLSI J.*, vol. 38, no. 2, Dec. 2004, pp. 131-138.
4. B. Bailey, G. Martin, and T. Anderson, *Taxonomies for the Development and Verification of Digital Systems*, Springer, 2005.

ple, interpreting the logic synthesis process in this framework, we find the following:

- RTL code or Boolean functions represent the design’s functionality.
- The platform includes a library of gates, or higher-complexity logic blocks.
- Mapping is the actual logic synthesis step that implements the functionality as an interconnection of gates (platform instance) optimizing a set of metrics involving area, power, and timing; the synthesis tool then exports the mapped design (gate-level netlist) to the layout phase, and the physical design tool maps this representation to a physical platform.

The PBD paradigm applies equally well to the application and algorithmic levels, where functionality can be a mathematical description—for example, a Moving Picture Experts Group (MPEG) encoding algorithm. Also, the platform can be a set of subalgorithms for implementing each functional block of the encoding method. The result of the mapping process then goes to a lower level, where the left branch is a mapped platform instance, and the right

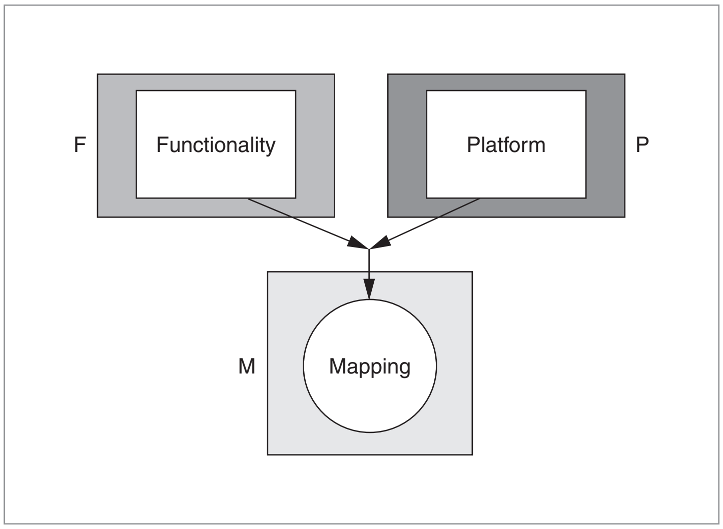


**Figure 1. Platform-based design classification framework elements. *Functionality* indicates functional representations of a design completely independent of implementation architectures. *Platform* concerns the modules used to implement the functional description—for example, processors, memories, and custom hardware. *Mapping* refers to instances of the design in which the functionality has been assigned to a set of correctly interconnected modules.**

branch is a new set of elements for implementing the mapped platform instance. This process repeats until the result of the mapping process is a fully implemented solution. Thus, the design process is partitioned into levels, where each level represents a particular abstraction. The corresponding platform and mapping process optimizes specific aspects of the design.

This framework prescribes a unified design methodology and hence is useful for identifying where existing tools and flows fit and how to integrate them in the overall system design process.

## Classifying ESL tools

We use the PBD paradigm to classify several ESL-related tools. Doing so casts present system-level design efforts in a global framework that serves as a unifying element. Of course, existing approaches may fall into more than one classification category because they cover more than one step of PBD. We could consider this a fault of the classification method, because a classification is effective only if it can cleanly partition the various objects being classified. However, partitioning the design steps rather than the tool coverage is more powerful because it identifies the tools' roles in the overall design paradigm. Indeed, the classification criteria can provide hints on how to connect different tools to yield an encompassing design flow. We've developed an environment for design space exploration called Metropolis, which completely reflects the design paradigm followed here. Metropolis can serve as the unifying framework for system design, where tool developers can embed tools, libraries, and approaches if the appropriate interfaces are built.

The classification classes reflect the Y-shaped diagram, with an additional classification criterion related to the abstraction level at which the tools work (see Figure 1):

*Bin F* consists of functional representations of a design independent of implementation architectures and with no associated physical quantity, such as time or power. For example, a Simulink diagram expressing an algorithm for automotive engine control and a Ptolemy II description of an MPEG-decoding algorithm both belong to this bin. These diagrams could be refinements of more abstract representations such as metamodels, as in Metropolis. To this bin, we assign tools that manipulate, simulate, and formally or informally analyze functional descriptions.

*Bin P* represents the library of modules for implementing the functional description. The modules are architectural elements such as processors, memories, coprocessors, FPGAs, custom hardware blocks, and interconnections (buses, networks, and so on). The elements also include middleware, such as operating systems for processors and arbitration protocols for buses, because these software components present the architectural services that the hardware offers to the application software. To this bin, we assign tools for connecting or manipulating the modules, as well as tools for analyzing the property of the complete or partial platform instances obtained.

*Bin M* represents mapped instances of the design in which the designer or an automatic mapping tool has assigned functionality to a set of correctly interconnected modules. The connection between bins F, P, and M represents the mapping process. In this bin, we classify any tool that assigns architectural elements to functionality or generates the design's mapped view. For example, bin M would include a high-level synthesis tool because the designer has assigned, perhaps manually, part of the functionality to a virtual hardware component in the platform and is asking the tool to generate the lower-level view, in this case an RTL description of the design. By the same token, we can classify a code generation tool in bin M because the designer has assigned (perhaps manually) part of the functionality to a software-programmable element of the library and is asking the tool to generate the lower-level view. In this case, the view is a software program—whether assembly language, C, or a higher-level language—which is then compiled to move toward implementation. In this article, we consider the compilation phase and the synthesis from RTL to gates to be part of a traditional design flow and thus not part of our ESL tool classification.

Some tools can handle two or even all three aspects of the PBD paradigm. To classify these tools, we introduce metaclasses (or metabins), indicated by combinations of F, P, and M. For example, in metabin FM, we assign a synthesis tool that handles functional components along with their mappings to platform components. Tools classified in metaclasses cover several parts of the PBD design flow. Designers using these tools can benefit from the design view we propose by clearly decoupling function from architecture and mapping. Doing so can enhance reusability and help the designer reach a correct implementation efficiently.

To make the partitioning of the tools finer, we introduced another, orthogonal criterion for classification: the abstraction level at which the tools operate. Whereas PBD doesn't limit the abstraction levels that designers use per se, most of the tools we reviewed work at three levels, listed here from highest to lowest:

**Table 1. Tools in bin F: Industrial. (C: component level; I: implementation level; S: system level)**

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| MathWorks | Matlab | High-level technical computing language and interactive environment for algorithm development, data visualization, analysis, and numeric computation. | S: Matlab language, vector, and matrix operations | http://www.mathworks.com/products/matlab |
| Scilab | Scicos | Graphically model, compile, and simulate dynamic systems | S: Hybrid systems | http://www.scilab.org |
| Novas Software | Verdi | Debugging for SystemVerilog | I: Discrete event | http://www.novas.com |
| Mentor Graphics | SystemVision | Mixed-signal and high-level simulation | S: VHDL-AMS, Spice, C | http://www.mentor.com/products/sm/systemvision |
| EDAptive Computing | EDAStar | Military and aerospace system-level design | S: Performance models | http://www.edaptive.com |
| Time Rover | DBRover, TemporalRover, StateRover | Temporal rules checking, pattern recognition, and knowledge reasoning | C: Statecharts assertions | http://www.time-rover.com |
| Maplesoft | Maple | Mathematical problem development and solving | S: Mathematical equations | http://www.maplesoft.com |
| Wolfram Research | Mathematica | Graphical mathematical development and problem solving with support for Java, C, and .Net | S: Mathematical equations | http://www.wolfram.com |
| Mesquite Software | CSIM 19 | Process-oriented, general-purpose simulation toolkit for C and C++ | S: C, C++ | http://www.mesquite.com |
| Agilent Technologies | Agilent Ptolemy | Functional verification | C: Timed synchronous dataflow | http://www.agilent.com |
| National Instruments | LabView | Test, measurement, and control application development | S: LabView programming language | http://www.ni.com/labview |

- *System level S* corresponds to heterogeneous designs that use different models of computation (MoCs) to represent function, platforms, and mappings.
- *Component level C* involves subsystems containing homogeneous components.
- *Implementation level I* comprises the final design step, when the design team considers the job complete.

We now present our classification, beginning with tools that fall into individual bins—those meant to be part of a larger tool flow or that work in a very specific application domain. We then address tools that cover larger portions of the design flow space.

## Bin F

Tools in this bin often serve to capture designs and their specifications quickly without making any assumptions about the underlying implementation details (see Tables 1-3). At this level, the descriptions might include behavioral issues such as concurrency, or communication concepts such as communication protocols. Some tools handle only one MoC—for example, finite-state machines (FSMs). Others are more general, handling a set of MoCs or having no restrictions. For example, the Simulink representation language handles discrete dataflow and continuous time. Hence, it is a limited heterogeneous modeling-and-analysis tool. Ptolemy II, with its actor-oriented abstract semantics,

Table 2. Tools in bin F: Academic.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Univ. of California, Berkeley | Ptolemy II | Modeling, simulation, and design of concurrent, real-time, embedded systems | S: All MoCs | http://ptolemy.eecs.berkeley.edu |
| Royal Inst. of Technology, Sweden | ForSyDe | System design starts with a synchronous computational model, which captures system functionality | C: Synchronous MoC | http://www.imit.kth.se |
| Mozart Board | Mozart | Advanced development platform for intelligent, distributed applications | S: Object-oriented GUI using Oz | http://www.mozart-oz.org |

Table 3. Tools in bin F: Languages.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Celoxica | Handel-C | Compiling programs into hardware images of FPGAs or ASICs | C: Communicating sequential processes | NA |
| Univ. of California, Irvine | SpecC | ANSI-C with explicit support for behavioral and structural hierarchy, concurrency, state transitions, timing, and exception handling | C: C language based | http://www.ics.uci.edu/~specc |
| Inria | Esterel | Synchronous-reactive programming language | C: Synchronous reactive | http://www-sop.inria.fr/meije/esterel/esterel-eng.html |
| Univ. of Kansas | Rosetta | Compose heterogeneous specifications in a single declarative semantic environment | S: All MoCs | http://www.sldl.org |
| Mozart Board | Oz | Advanced, concurrent, networked, soft real-time, and reactive applications | C: Dataflow synchronization | http://www.mozart-oz.org |
| Various | ROOM | Real-time object-oriented modeling | S: Object oriented | NA |

can handle all MoCs. Depending on the MoC supported, design entry for each tool could start at a higher or a lower abstraction level.

### Bin P

This category includes providers of platforms or platform components, as well as tools and languages that describe, manipulate, or analyze unmapped platforms (see Tables 4 and 5). Similar to tools in bin F, those in bin P can span several abstraction layers and support different kinds of architectural components. For example, Xilinx and Altera mainly concern programmable hardware devices, whereas Tensilica focuses on configurable processors. Others, such as Sonics and Beach Solutions, focus on integration and communication components. This category's main characteristic is configurability, which ensures the applicability of a platform or components to a wide variety of applications and design styles.

### Bin M

This bin contains tools dedicated to refining a functional description into a mapped platform instance, including its performance evaluation and possibly the synthesis steps required to proceed to a more detailed abstraction level (see Tables 6-8). The tools in bin M vary widely in particular design style, MoC, and supported application area. To provide the necessary quality of results, the tools are typically very specific.

Table 4. Tools in bin P: Industrial.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Prosilog | Nepsys | Standards-based IP libraries and support tools (SystemC) | C: RTL and transaction-level SystemC; VHDL for SoCs | http://www.prosilog.com |
| Beach Solutions | EASI-Studio | Solutions to package and deploy IP in a repeatable, reliable manner | C: Interconnection | http://www.beachsolutions.com |
| Altera | Quartus II | FPGAs, CPLDs, and structured ASICs | I: IP blocks, C, and RTL; FPGAs | http://www.altera.com |
| Xilinx | Platform Studio | IP integration framework | C: IP blocks, FPGAs | http://www.xilinx.com |
| Mentor Graphics | Nucleus | Family of real-time operating systems and development tools | S: Software | http://www.mentor.com/products/embedded_software/nucleus_rtos |
| Sonics | Sonics Studio | On-chip interconnection infrastructure | I: Bus-functional models | http://www.sonicsinc.com |
| Xilinx | ISE, EDK, XtremeDSP | FPGAs, CPLDs, and structured ASICs | I: IP blocks, C, and RTL; FPGAs | http://www.xilinx.com |
| Design and Reuse | Hosted Extranet Services | IP delivery systems | S: All types of IP | http://www.design-reuse.com |
| Stretch | Software Configurable Processor compiler | Compile a subset of C into hardware for instruction extensions | C: Software-configurable processors | http://www.stretchinc.com |
| ProDesign | CHIPit | Transaction-based verification platform | C: FPGA-based rapid prototyping | http://www.prodesign-usa.com |

Table 5. Tools in bin P: Languages.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Spirit Consortium | Spirit | IP exchange and integration standard written in XML | S: Various IP levels | http://www.spiritconsortium.com |

## Metabin FP

This category consists of languages that can express both functionality and architecture (see Tables 9 and 10 on p. 368). Typically, they express algorithms and different styles of communication and structure for different MoCs. Assertions, or constraints, complement the platform description. In the case of Unified Modeling Language (UML), the semantics are often left unspecified.

## Metabin FM

This metabin reflects tools that provide some combination of functional description and analysis capabilities plus mapping and synthesis capabilities (see Table 11 on p. 368). In this case, the platform architecture is typically fixed. This lack of flexibility is offset by the often superior quality of achievable implementation results.

## Metabin PM

This metabin includes tools that combine architectural services and mapping (see Tables 12-14 on pp. 369-370). These tools have a tight coupling between the services they provide and how functionality can map to these services. They require the use of other tools for some aspect of system design (often in the way the design functionality is specified).

## Metabin FPM

Entries in this category are the frameworks that support the PBD paradigm (see Tables 15 and 16 on p. 371).

Table 6. Tools in bin M: Industrial, set I.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| MathWorks | Real-Time Workshop | Code generation and embedded-software design | S: Simulink-level models | http://www.mathworks.com |
| dSpace | TargetLink | Optimized code generation and software development | S: Simulink models | http://www.dspace.com |
| ETAS | Ascet | Modeling, algorithm design, code generation, and software development, with emphasis on the automotive market | S: Ascet models | http://en.etasgroup.com/products/ascet/index.shtml |
| Y Explorations | eXCite | Take virtually unrestricted ISO or ANSI-C with channel I/O behavior and generate Verilog or VHDL RTL output for logic synthesis | S: C language input | http://www.yxi.com |
| AccelChip | AccelChip and AccelWare | DSP synthesis; Matlab to RTL | C: Matlab | http://www.accelchip.com |
| Forte Design Systems | Cynthesizer | Behavioral synthesis | C: SystemC to RTL | http://www.forteds.com |
| Future Design Automation | System Center Co-development Suite | ASCI-C to RTL synthesis toolset | C: C to RTL | http://www.future-da.com |
| Catalytic | DeltaFX, RMS | Synthesis of DSP algorithms on processors or ASICs | I: Matlab algorithms | http://www.catalytic-inc.com |
| ACE Associate Compiler Experts | CoSy | Automatic generation of compilers for DSPs | I: DSP-C and embedded-C language extensions | http://www.ace.nl |
| Tenison | VTOC | RTL to C++ or SystemC | I: RTL, transactional | http://www.tenison.com |

In particular, Metropolis fully embodies this paradigm, covering all bins and all abstraction layers. In this category, we include design space exploration tools and languages that can separately describe the functionality on the one hand, and the possible architectures for an implementation on the other. These tools can also map the functionality onto the platform instances to obtain metrics for the implementation's performance.

## Design scenarios

Here, we use the PBD framework of Figure 1 to map three design flow scenarios on the tool landscape. Figure 2 (see p. 372) shows the metabins and the hierarchical levels where activities take place.

### Scenario 1: New application design from specification

The requirements of this scenario include the need to start from a high-level specification; the desire to capture and modify the initial specification quickly; the ability to express concurrency, constraints, and other behavior-specific characteristics efficiently; and the ability to capture useful abstract services for implementing high-level specifications into a more detailed functional view. The flow thus starts at the higher abstraction levels in bin F of our classification. We can expand these levels into a Y diagram of the same structure as the one described in Figure 1. This structure offers

- flexible specification capture—no ties to a particular implementation style or platform;
- services that help move the abstract design toward a more constrained version (for example, algorithms that can implement functionality); and
- independent mapping of functionality onto algorithmic structures that enable reuse of the functional specification.

Table 7. Tools in bin M: Industrial, set II.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Sequence Design | ESL Power Technology, Power Theater, CoolTime, CoolPower | Power analysis and optimization | I: SystemC level | http://www.sequencedesign.com |
| PowerEscape (with CoWare) | PowerEscape Architect, PowerEscape Synergy, PowerEscape Insight | Memory hierarchy design, code performance analysis, complete profiling | C: C code | http://www.coware.com/products/powerescape.php |
| CriticalBlue | Cascade | Design flow for application-specific hardware acceleration coprocessors for ARM processors | I: C code to Verilog or VHDL | http://www.criticalblue.com |
| Synfora | PICO Express | C to RTL, or C to System C (transaction-level models) | I: Pipeline processor arrays | http://www.synfora.com |
| Actis | AccurateC | Static code analysis for SystemC | C: C syntax and semantic checking | http://www.actisdesign.com |
| Impulse Accelerated Technologies | CoDeveloper | C to FPGA | C: C code | http://www.impulsec.com |
| Poseidon Design Systems | Triton Tuner, Triton Builder | Design flow for application-specific hardware acceleration coprocessors | C: C and SystemC | http://www.poseidon-systems.com |
| SynaptiCAD | SynaptiCAD line | Testbench generators and simulators | C: RTL and SystemC | http://www.syncad.com |
| Avery Design Systems | TestWizard | Verilog HDL, VHDL, and C-based testbench automation | I: RTL and C | http://www.avery-design.info |
| Emulation and Verification Engine | ZeBu | Functional verification | I: Hardware emulation | http://www.eve-team.com |

Table 8. Tools in bin M: Academic.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Univ. of Illinois at Urbana-Champaign | Impact Compiler | Compilation development for instruction-level parallelism | S: C code for high-performance processors | http://www.crhc.uiuc.edu/Impact |

Let's examine an example in the multimedia domain: the implementation of a JPEG encoder on a heterogeneous multiprocessor architecture such as the Intel MXP5800. This architecture has eight image signal processors ($ISP_1$ to $ISP_8$) connected with programmable quad ports (eight per processor).[7] The encoder compresses raw image data and emits a compressed bitstream. The first step in the scenario is to choose a

**Table 9. Tools in metabin FP: Industrial.**

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| MathWorks | Simulink, State Flow | Modeling, algorithm design, and software development | S: Timed dataflow, FSMs | http://www.mathworks.com |

**Table 10. Tools in metabin FP: Languages.**

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Open SystemC Initiative | SystemC | Provide hardware-oriented constructs within the context of C++ | S: Transaction level to RTL | http://www.systemc.org |
| Object Management Group | Unified Modeling Language | Specify, visualize, and document software system models | S: Object-oriented, diagrams | http://www.uml.org |
| Accellera | SystemVerilog | Hardware description and verification language extension of Verilog | S: Transaction level, RTL, assertions | http://www.systemverilog.org |

**Table 11. Tools in metabin FM: Industrial.**

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Celoxica | DK Design Suite, Agility Compiler, Nexus-PDK | Algorithmic design entry, behavioral design, simulation, and synthesis | C: Handel-C based | http://www.celoxica.com |
| BlueSpec | BlueSpec Compiler, BlueSpec Simulator | BlueSpec SystemVerilog rules and libraries | S: SystemVerilog and term-rewriting synthesis | http://www.bluespec.com |
| I-Logix | Rhapsody and Statemate | Real-time UML-embedded applications | S: UML based | http://www.ilogix.com |
| Mentor Graphics | Catapult C | C++ to RTL synthesis | C: Untimed C++ | http://www.mentor.com |
| Esterel Technologies | SCADE, Esterel, Studio | Code generation for safety-critical applications such as avionics and automotive | I: Synchronous | http://www.esterel-technologies.com |
| Calypto | SLEC System | Functional verification between system level and RTL | C: SystemC, RTL | http://www.calypto.com |

particular MoC to describe the design's functionality. To be more efficient in applying our proposed design paradigm, the designer should use a MoC that is also suitable for describing the architecture's capabilities. Hence, the designer eases the mapping task and the analysis of the mapped design's properties. In addition, a synthesis step could execute the mapping process automatically.

Because this is a data-streaming application that maps onto a highly concurrent architecture, it is natural to use a Kahn process networks (KPN) representation. In KPN, a set of processes communicate through one-way FIFO channels. Reads from channels are blocked when no tokens are present; processes cannot query the channel status. However, this model is Turing complete, so scheduling and buffer size are undecidable. The KPN model of the JPEG encoder algorithm is completely independent of the target architecture sat-

Table 12. Tools in metabin PM: Industrial, set I.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| ARM | RealView MaxSim | Embedded microprocessors and development tools; system-level development tools | C: C++ ARM processor development | http://www.arm.com |
| Tensilica | Xtensa, XPRES | Programmable solutions with specialized Xtensa processor description from native C and C++ code | C: Custom ISA processor, C and C++ code | http://www.tensilica.com |
| Summit | System Architect, Visual Elite | Efficiently design and analyze the architecture and implementation of multicore SoCs and large-scale systems | C: SystemC | http://www.sd.com |
| VaST Systems Technology | Comet, Meteor | Very high-performance processor and architecture models | S: Virtual processor, bus, and peripheral devices | http://www.vastsystems.com |
| Virtio | Virtio Virtual Platform | High-performance software model of a complete system | I: Virtual platform models at SystemC level | http://www.virtio.com |
| Cadence | Incisive | Integrated tool platform for verification, including simulation, formal methods, and emulation | S: RTL and SystemC assertions | http://www.cadence.com |
| Mentor | Platform Express | XML-based integration environment | C: XML-based structure | http://www.mentor.com |
| SpiraTech | Cohesive | Protocol abstraction transformers | C: Transaction level, IP blocks | http://www.spiratech.com |
| ARC International | ARC | Embedded microprocessors and development tools | I: ISA extensions, microarchitectural level | http://www.arc.com |
| Arithmatica | CellMath Tool Suite | Proprietary improvements for implementing silicon computational units | I: Microarchitectural datapath computation elements and design | http://www.arithmatica.com |

isfying the requirements for this scenario. We could use Ptolemy II to capture this model and simulate the selected algorithm's behavior.

To allow a better analysis and to refine the model toward implementation, we can map this model into another dataflow model, similar to cyclostatic dataflow,[8] which permits only one writer per channel but allows multiple reader processes. For all channels, each reader process can read each data token exactly once. Also, this dataflow model allows limited forms of data-dependent communication. To enable the execution of multiple processes on a single processing element, this MoC supports multitasking. In particular, the system may suspend a process only between firings. Because of the limitations just discussed, this MoC lets designers decide scheduling, buffer sizing, and mapping. It is easy to express the model in Ptolemy II and to describe it in Simulink or the Signal Processing Worksystem (SPW). This first step—mapping a more flexible model for the functionality into a more restricted one that is easier to implement and analyze—is critical in any system-level design.

Subsequently, the mapped specification becomes the functional representation for the diagram in Figure 1. So, the flow can continue at lower abstraction levels with tools in metabin FM for an integrated solution, or in bin F followed by M for a multitool solution. Because most of the architecture is fixed, an efficient, specialized approach is more appropriate. Figure 2a shows a

**Table 13. Tools in metabin PM: Industrial, set II.**

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Target Compiler Technologies | Chess (compiler), Checkers (ISS) | Retargetable tool suite for developing, programming, and verifying embedded IP cores | I: Mapping of C code to processors written in nML | http://www.retarget.com |
| Arteris | Danube, NoCexplorer | Synthesis of NoC | C: NoC dataflow | http://www.arteris.net |
| ChipVision Design Systems | Orinoco | Pre-RTL power prediction for behavioral synthesis | C: SystemC algorithm input | http://www.chipvision.com |
| Wind River Systems | Various platform solutions | Provide various platforms for different design segments (auto, consumer) | I: Software API | http://www.windriver.com |
| CoWare | ConvergenSC | Capture, design, and verification for SystemC | S: SystemC functionality input; SystemC, HDL services | http://www.coware.com |
| Carbon Design Systems | VSP | Presilicon validation flow | C: Verilog and VHDL, bus protocols | http://www.carbondesignsystems.com |
| GigaScale IC | InCyte | Chip estimation and architecture analysis | S: High-level chip information (gate count, I/O, IP blocks) | http://www.chipestimate.com |
| Virtutech | Virtutech Simics | Build, modify, and program new virtual systems | I: C language and ISAs | http://www.virtutech.com |
| National Instruments | LabView 8 FPGA | Create custom I/O and control hardware for FPGAs | C: LabView graphical programming | http://www.ni.com/fpga |
| CoWare | LisaTek | Embedded-processor design tool suite | C: Lisa architecture description language | http://www.coware.com |

**Table 14. Tools in metabin PM: Academic.**

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Carnegie Mellon Univ. | MESH | Enable heterogeneous microdesign through new simulation, modeling, and design strategies | C: C input; programmable, heterogeneous multiprocessors | http://www.ece.cmu.edu/~mesh |
| Univ. of California, Los Angeles | xPilot | Automatically synthesize high-level behavioral descriptions for silicon platforms | C: C, SystemC | http://cadlab.cs.ucla.edu/soc |

potential traversal of the framework. For our JPEG case, we can map the functionality onto the MXP5800 using the Metropolis environment to analyze potential problems with the architecture or to optimize the application's coding for the chosen platform instance.

## Scenario 2: New integration platform development

This scenario describes the development of a new integration platform: a hardware architecture, embedded-software architecture, design methodologies

Table 15. Tools in metabin FPM: Industrial.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| CoFluent Design | CoFluent Studio | Design space exploration through Y-chart modeling of functional and architectural models | S: Transaction-level SystemC | http://www.cofluentdesign.com |
| MLDesign Technologies | MLDesigner | Integrated platform for modeling and analyzing the architecture, function, and performance of high-level system designs | S: Discrete event, dynamic dataflow, and synchronous dataflow | http://www.mldesigner.com |
| Mirabilis Design | VisualSim product family | Multidomain simulation kernel and extensive modeling library | S: Discrete event, synchronous dataflow, continuous time, and FSM | http://www.mirabilisdesign.com |
| Synopsys | System Studio | Algorithm and architecture capture, performance evaluation | S: SystemC | http://www.synopsys.com |

Table 16. Tools in metabin FPM: Academic.

| Provider | Tools | Focus | Abstraction | Web site |
|---|---|---|---|---|
| Univ. of California, Berkeley | Metropolis | Operational and denotational functionality and architecture capture, mapping, refinement, and verification | S: All MoCs | http://www.gigascale.org/metropolis |
| Seoul National Univ. | Peace | Codesign environment for rapid development of heterogeneous digital systems | S: Objected-oriented C++ kernel (Ptolemy based) | http://peace.snu.ac.kr |
| Vanderbilt Univ. | GME, Great, Desert | Metaprogrammable tool for navigating and pruning large design spaces | S: Graph transformation, UML and XML based, and external component support | http://repo.isis.vanderbilt.edu |
| Delft Univ. of Technology | Artemis, Compaan and Laura, Sesame, Spade | Workbench enabling methods and tools to model applications and SoC-based architectures | C: Kahn process networks | http://ce.et.tudelft.nl/artemis |
| Univ. of California, Berkeley | Mescal | Programming of application-specific programmable platforms | S: Extended Ptolemy II, network processors | http://www.gigascale.org/mescal |

(authoring and integration), design guidelines and modeling standards, virtual-components characterization and support, and design verification (hardware-software, hardware prototype), focusing on a particular target application.[9] Unlike the first scenario, this one is not concerned with the design of a particular application but rather with the development of a substrate to realize several applications. Characteristic of this scenario is the service- and mapping-centric requirements that concern tools in metabin PM for development and analysis at the desired abstraction level. The platform developer builds the substrate, or platform, and uses the tools in metabin PM. The platform user proceeds in metabin FM to map the desired

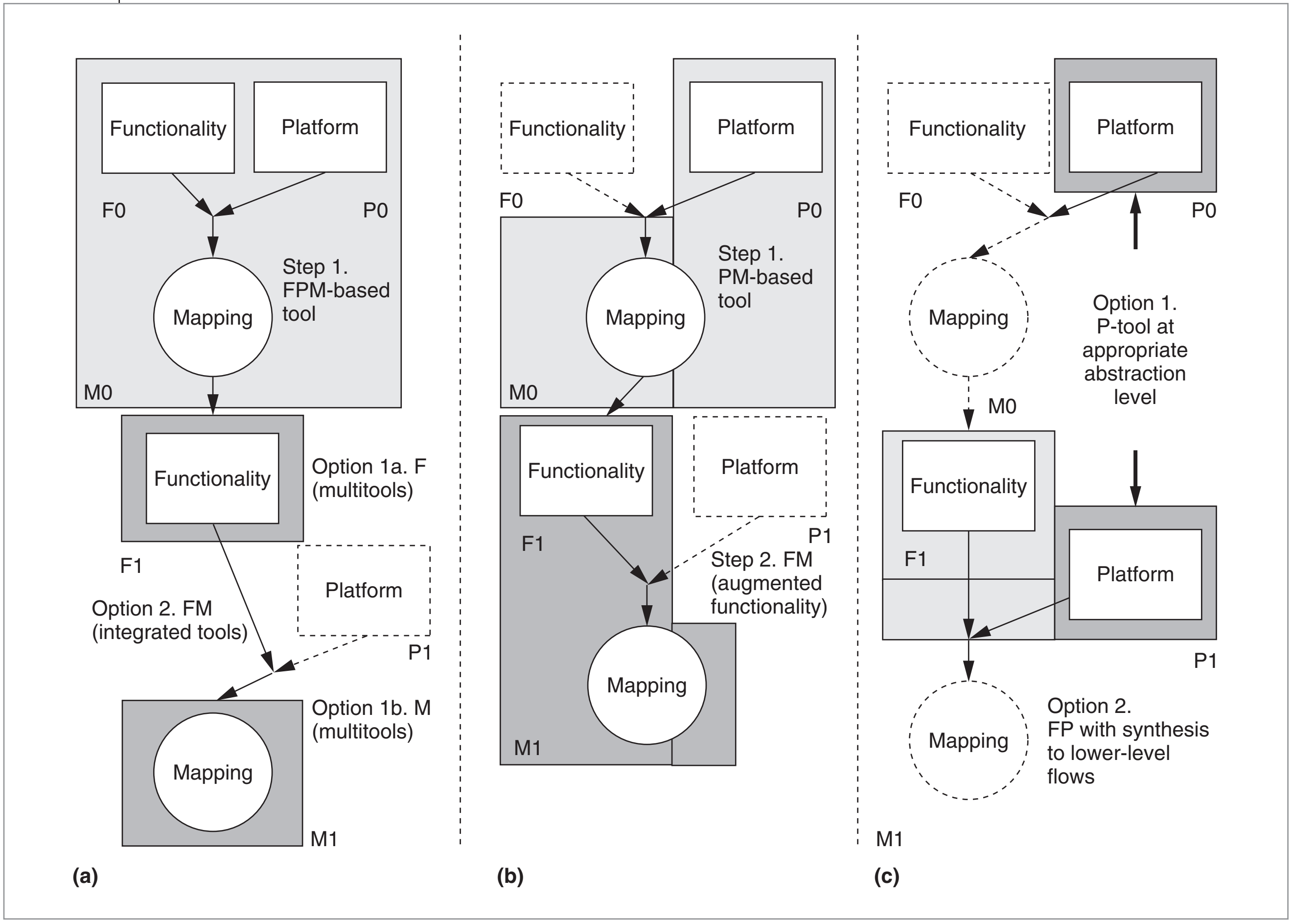


**Figure 2. Metabins and hierarchical levels for three design scenarios: new application design from specification (a), new integration platform development (b), and legacy design integration (c).**

functionality to the selected platform instance. Figure 2b illustrates the metabin flows that support these development requirements.

Consider as a test case the development of a new electronic control unit (ECU) platform for an automotive engine controller. The application designers have already developed the application code for the platform, but a Tier 1 supplier wants to improve the cost and performance of its part of the platform to avoid losing an important original equipment manufacturer (OEM) customer. If the designers employ the paradigm described in this article, the application becomes as independent on the ECU platform as possible. Next, in collaboration with a Tier 2 supplier (a chip maker), the Tier 1 supplier determines qualitatively that a dual-core architecture would offer better performance at a lower manufacturing cost. A platform designer then uses a tool for platform development, such as LisaTek, to capture the dual-core architecture. If the dual core is based on ARM processing elements, the designers and the Tier 1 supplier can also use ARM models and tool chains. An appropriate new real-time operating system could exploit the implementation's multicore nature. At this point, the designers map the application onto one of the possible dual-core architectures, considering the number of bits supported by the CPU, the set of peripherals to integrate, and the interconnect structure. For each choice, the designers simulate the mapped design with the engine control software or a subset of it to stress the architecture. These simulations can employ the ARM tools or VaST offerings to rapidly obtain important statistics such as interconnect latency and bandwidth, overall system performance, and power consumption. At the end of this exercise, the Tier 2 supplier is fairly confident that its architecture is capable of supporting a full-fledged engine control algorithm. Any other Tier 1 supplier can use this product now for its engine control offering.

### Scenario 3: Legacy design integration

The final scenario represents a common situation for many companies wishing to integrate their existing designs into new ESL flows. In this case, it's difficult to separate functionality and architecture, because in most embedded systems the documentation refers to the final implementation, not to its original specifications and the relative implementation choices. If modifying the design is necessary to implement additional features, it's very difficult to determine how the new functionality will affect the existing design. This situation calls for reverse engineering to extract functionality from the final implementation. The most effective way to do this might be to start the description of the functionality from scratch, using tools in bin F. An alternative might be an effective encapsulation of the legacy part of the design so that the new part interacts cleanly with the legacy part. We could then consider existing components as architectural elements that we must describe using tools in bin P. This, in turn, is possible at different abstraction levels. Because legacy components typically support a specific application, mapping is often unnecessary, and functional or architectural cosimulation can validate a new design. Metabin FP at the system level is therefore the appropriate flow model in this case. Figure 2c illustrates this scenario.

**ESL WILL EVENTUALLY BE** in the limelight of the design arena. But structural conditions in the EDA and electronics industry must change to offer a sufficiently receptive environment that will allow the birth of new companies and the evolution of present ones into this exciting area. An important technical prerequisite is industry and academia agreement on a holistic view of the design process in which to cast existing and future tools and flows. Our unified design framework can act as a unifying element in the ESL domain. However, standardization of system-level design will take years and require significant effort to fully materialize. ■

## Acknowledgments

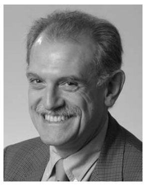
We thank the following for their support in reviewing this article and in helping to classify the various ESL approaches. Without them, this article would not have been possible: Abhijit Davare, Alessandro Pinto, Alvise Bonivento, Cong Liu, Gerald Wang, Haibo Zeng, Jike Chong, Kaushik Ravindran, Kelvin Lwin, Mark McKelvin, N.R. Satish, Qi Zhu, Simone Gambini, Wei Zheng, Will Plishker, Yang Yang, and Yanmei Li. A special thanks goes to Guang Yang and Trevor Meyerowitz for their valuable feedback. This work was done under partial support from the Center for Hybrid Embedded Software Systems and the Gigascale Systems Research Center.

## ■ References

1. G. Smith et al., *Report on Worldwide EDA Market Trends*, Gartner Dataquest, Dec. 2005.
2. J. Vleeschouwer and W. Ho, "The State of EDA: Just Slightly up for the Year to Date Technical and Design Software," *The State of the Industry*, Merrill Lynch report, Dec. 2005.
3. *International Technology Roadmap for Semiconductors 2004 Update: Design*, 2004, http://www.itrs.net/Links/2004Update/2004_01_Design.pdf.
4. A. Sangiovanni-Vincentelli, "Defining Platform-Based Design," *EE Times*, Feb. 2002, http://www.eetimes.com/news/design/showArticle.jhtml?articleID=16504380.
5. A. Sangiovanni-Vincentelli and G. Martin, "Platform-Based Design and Software Design Methodology for Embedded Systems," *IEEE Design & Test*, vol. 18, no. 6, Nov.-Dec. 2001, pp. 23-33.
6. D.D. Gajski and R.H. Kuhn, "Guest Editors' Introduction: New VLSI Tools," *Computer*, vol. 16, no. 12, Dec. 1983, pp. 11-14.
7. A. Davare et al., "JPEG Encoding on the Intel MXP5800: A Platform-Based Design Case Study," *Proc. 3rd Workshop Embedded Systems for Real-Time Multimedia* (ESTIMedia 05), IEEE CS Press, 2005, pp. 89-94.
8. G. Bilsen et al., "Cyclo-Static Dataflow," *IEEE Trans. Signal Processing*, vol. 44, no. 2, Feb. 1996, pp. 397-408.
9. H. Chang et al., *Surviving the SOC Revolution: A Guide to Platform-Based Design*, Kluwer Academic Publishers, 1999.

**Douglas Densmore** is a PhD candidate in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. His research interests focus on system-level architecture modeling, with emphasis on architecture refinement techniques for system-level design. Densmore has a BS in computer engineering from the University of Michigan, Ann Arbor, and an MS in electrical engineering from the University of California, Berkeley. He is a member of the IEEE.

**Roberto Passerone** is an assistant professor in the Department of Information and Communication Technology at the University of Trento, Italy. His research interests include system-level design, communication design, and hybrid systems. Passerone has a Laurea degree in electrical engineering from Politecnico di Torino, Italy, and an MS and a PhD in electrical engineering and computer sciences from the University of California, Berkeley. He is a member of the IEEE.



**Alberto Sangiovanni-Vincentelli** holds the Buttner Endowed Chair of the Electrical Engineering and Computer Sciences Department at the University of California, Berkeley. His research interests include design tools and methodologies, large-scale systems, embedded controllers, and hybrid systems. Sangiovanni-Vincentelli has a PhD in engineering from Politecnico di Milano. He is cofounder of Cadence and Synopsys, an IEEE Fellow, a member of the General Motors Scientific and Technology Advisory Board, and a member of the National Academy of Engineering.

■ Direct questions or comments about this article to Douglas Densmore, Dept. of Electrical Engineering and Computer Sciences, Univ. of California, Berkeley, 545Q Cory Hall (DOP Center), Berkeley, CA 94720; densmore@eecs.berkeley.edu.

**For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.**