Platform-Based Co-Design and Co-development: Experience, Methodology and Trends

Grant Martin
Cadence Design Systems
2001 Addison Street, Third Floor
Berkeley, California 94704 U.S.A.
+1-510-647-2804
gmartin@cadence.com

Jean-Yves Brunel
Cadence Design Systems
18 rue Grange Dame Rose
78148 Velizy, France
+33-1-34-88-5386
brunel@cadence.com

ABSTRACT

In this paper we focus on platform-based design of complex SoC products and the methodologies involved in both creating platforms and using them to design derivatives. In particular, we concentrate on the co-development of software and hardware: including co-design and co-verification stages. After defining the methodologies and describing their key steps, we summarise specific industrial experience, and draw conclusions about the future evolution of these methodologies. This includes some key ideas about the future of co-design as seen by embedded SW developers.

1. INTRODUCTION

The co-development of complex embedded HW-SW products involves the relatively new steps of co-design and co-verification. These methods have been talked about for a number of years but have not been central to the design of many systems up until now. Indeed, system-level design and associated methodologies have hardly taken off. However, the growing importance of platform-based design places new emphasis on the key technologies of HW-SW co-development. We start first by coming up with a simple definition of platform-based design.

2. PLATFORM-BASED DESIGN

Platform-based design of complex products including embedded system-on-chip (SoCs) has been discussed frequently in recent years [1,2]. It is defined in various ways, but one source of good definitions (as of February 2002) comes from the VSIA Platform-Based Design Study Group. It defines a platform as "a library of virtual components and an architectural framework consisting of a set of integrated and pre-qualified software and hardware IP blocks, models, EDA and software tools, libraries and methodology to support rapid product development through architectural exploration, integration and verification." Platform-based design is then defined as: "An integration-oriented design approach emphasizing systematic reuse, for developing complex products based upon platforms, intended to reduce development risks, costs and time to market. Within the VSIA the scope of the platform is bounded to the SoC." [3]

This design approach has implications for the design methodology and type of system-level design tools used to support it. In particular, we note that since we are talking about complex embedded systems, with substantial software content, the issues of co-design and co-verification, in which both hardware and software considerations are important, must be a major part of the methodology.

3. METHODOLOGIES

In platform-based design, we need to distinguish clearly between the process of platform development, and the process of specific product development using the platform. The latter is commonly referred to as 'derivative design'. The key tasks involved in creating the platform, and in using it, may be superficially similar, but differ substantially in details. The implications for co-design and co-verification are important. *Figure 1* illustrates the key co-development methods overall, that would be used in designing a product in one pass from start to finish.

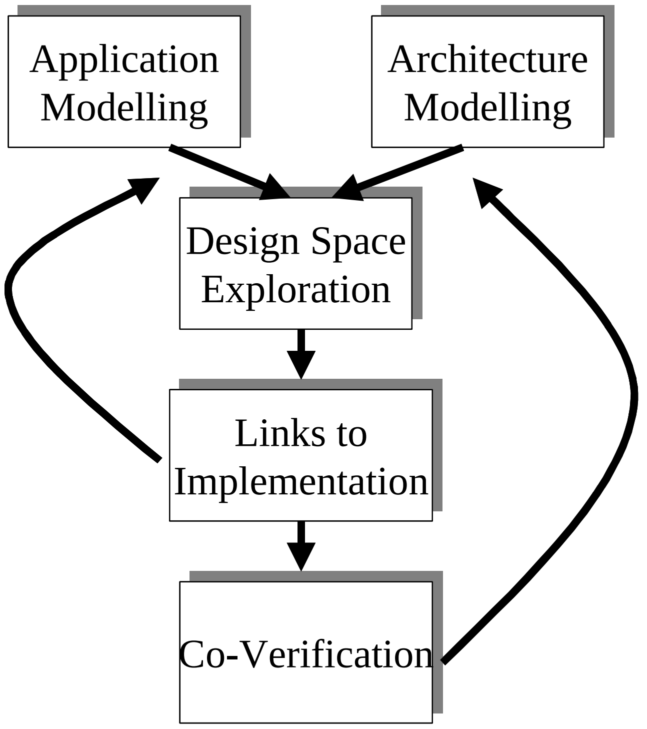


Figure 1: Key Co-Development Methods

The general tasks include modelling applications and platform architectures, design space exploration, links to implementation, and co-verification. These are described in more detail as part of platform development and derivative design in the next sections.

The general approach used is that of 'function-architecture co-design'.

3.1 Platform Development Methodology

In platform development, co-design is most significant. That is, the issue of partitioning and mapping system functionality into portions that will be implemented as software running on embedded processors vs. direct hardware implementations assumes fundamental importance. Note that in addition, the communications between architectural components implied by these partitioning and mapping decisions is also of key importance. The idea of 'software-software' co-design, where most functions are implemented in software on a platform with multiple embedded processors, makes the issue of which task maps to which processor, and the inter-processor communications requirements entailed by this choice, of growing importance. To support co-design and this partitioning, design space exploration is the key task.

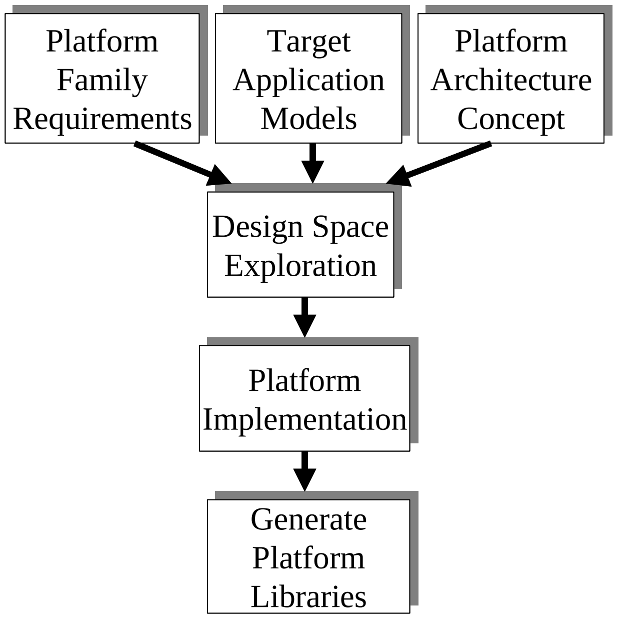

Platform Family Requirements
Target Application Models
Platform Architecture Concept
Design Space Exploration
Platform Implementation
Generate Platform Libraries

Figure 2: Platform Development Methodology

In *Figure 2*, we show the key tasks involved in platform development. These include:

- Gathering platform family requirements: determining the common requirements for the intended application domain and family of applications that will form the underpinning of platform decision-making. This includes identification of current and possible future standards, which the platform will support.
- Developing or re-using target application models: these may come from standards bodies, and a variety of internal and external sources.
- Developing a platform architecture concept: this includes selection of candidate processing and on-chip communications IP blocks, hardware blocks, RTOSs as appropriate, and other hardware-dependent software, as well as middleware and target-domain application software. The concept may include multiple architectures and candidate IP blocks.
- Design Space Exploration (DSE): mapping the application models to the candidate architectures and IP blocks to determine the fit of the applications to the architecture, the remaining design margin, the sensitivity of the architecture-application combinations and to answer key design questions. This involves a variety of simulation-based and analytical approaches for performance, power, and cost.
- Platform Implementation: once a candidate architecture and IP libraries are chosen, implement the platform to the design hand-off level chosen by the design team and company business model, which will be the starting point of derivative design. This could be at RTL, layout, or indeed full-chip level for a reconfigurable platform.
- Generation of Platform Libraries: generate the design artefacts, which will be the starting point for derivative design and rigorously store them into well-organised libraries.

The key step here is that of *Design Space Exploration*. This will be discussed in more detail later, with a multimedia example.

3.2 Derivative Design Methodology

In derivative design, co-verification is the key design step. Derivative design emphasises the rapid assembly of a product-specific design using the platform via detailed configuration, and verification that the derivative product will fit on the platform and that the components and the platform architecture chosen will meet design requirements. Co-verification is a key task for ensuring this.

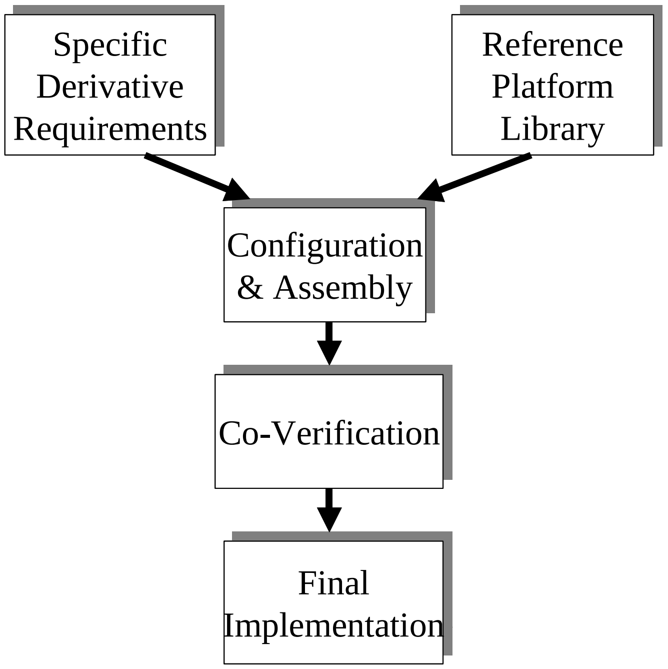


Figure 3: Derivative Design Methodology

In *Figure 3*, we show the key steps involved in derivative design. These include:

- Gather the requirements for the specific derivative product and ensure that the chosen platform 'field of use' is appropriate.
- Draw on the reference platform library for basic architecture and choice of pre-validated and characterised virtual components (HW and SW), which are required for the specific product derivative.
- Configure the platform and assemble the selected virtual components into it.
- Verify the resulting derivative product design. Since any significant platform involves embedded processor(s), and SW components – at least Hardware-dependent Software (HdS) such as the real-time operating system (RTOS), and device drivers – the derivative design will involve significant software at both HdS and application levels as well as appropriate middleware. Thus some amount of HW-SW co-verification will be required. For platforms that are mainly software-programmable, this may be an almost pure validation of the software's 'fit' onto the platform, both for computing and communications resources. Such co-verification could include both SW-based simulation and a variety of emulation and rapid prototyping methods.
- After co-verification has proven the derivative design will work, there is a final implementation stage. This may be a significant IC design experience (for platforms defined at the mostly RTL level, for example) or a small one (for 'hard' layout platforms with little hardware variability), to almost none at all (for a reconfigurable logic-based platform with embedded processors – a "highly programmable platform" or what has been called a "platform FPGA"). Thus the time and effort required for final implementation, fabrication and prototype testing will vary widely, depending on the implementation style.

Co-verification will be discussed in more detail later with an automotive example.

4. INDUSTRIAL EXPERIENCE

We first discuss the state of design tools for various steps of the co-development process, and then discuss some specific industrial experience in automotive and multimedia application areas.

4.1 The State of the Design Tools

Co-design and co-development tools have emerged in the last several years, although none of the current crop of tools has taken off into widespread commercial use. This is part and parcel of the problems with the adoption of system-level design tools within the industry. Notwithstanding the relative lack of adoption, it is indeed possible to construct a credible platform-based co-development flow using commercial tools, albeit with a significant amount of flow engineering and model development. Many design groups continue to rely on internal tool developments and ad-hoc methods for system design.

In co-design, notable commercial tools include VCC from Cadence, Archimate and Cosimate from Arexys (now TNI-Valiosys), Foresight from Foresight Systems (was Nuthena) and, to a more limited extent, N2C from CoWare. In co-verification, key commercial tools include Seamless from Mentor (having the lions share of the commercial market), N2C from CoWare, and, to a more limited extent, Eaglei from Synopsys (widely rumoured to be no longer under development) and V-CPU from Innoveda. SW-based co-simulation suffers from the significant effort required to acquire or develop Instruction Set Simulation models that integrate effectively with HDL simulation. This has led to two other approaches for co-verification: first, the creation of 'Virtual Prototyping' environments at the system-level using high level models of both processors and HW peripherals; and second, the use of HW-based rapid prototyping or emulation systems. Prominent examples for the first approach include Virtio, Axys, and VAST. The second approach has been advocated by traditional emulation approaches such as Quickturn, IKOS, and Mentor Celaro; and other architectures such as Aptix, Axis, Tharas, and Simpod.

Before system-level co-development tools become more popular commercially, certain requirements for commonly-adopted and widely-accepted modelling infrastructure must be met. Among these are be the widespread use of a common system level design language such as SystemC so that models have a much higher likelihood of true interoperability across various proprietary and internal tools.

4.2 Automotive Industry

Cadence has carried out significant work in adapting co-design technology (VCC) to the automotive industry, notably in partnership with BMW [4,5]. In this approach, the underlying system platform architecture is that of a network of electronic control units (ECUs) interconnected by a system bus implementing one of a variety of automotive communications protocols such as CAN, TTP, ByteFlyte, or FlexRay. Each ECU consists of an embedded microcontroller and associated memory; these are usually discrete units, not integrated into an SoC, and the system bus is discrete wiring. However, the principles of co-design and co-verification are similar no matter whether the integration is into one SoC or a more distributed embedded system.

The primary co-design task in this design is actually what we call 'software-software co-design'. This is targeted at designing specific automotive derivative products, as the basic platform is well known. The distributed network of ECUs is the delivery vehicle for a number of different functions implemented as software tasks running on the ECUs. These software tasks, in the VCC flow for BMW, are created using the ETAS ASCET tool, which models software as finite state machines. The ASCET flow generates executable code for the set of tasks that are to be mapped to the ECUs. The key co-design questions, which need to be answered, are:

- How many ECUs are required to adequately execute the tasks and yet leave sufficient margin for future software running on the network and for modifications required to the existing functions?
- How are the ECUs to be configured? Memory requirements are especially important as the automotive domain exhibits severe cost-sensitivity.
- Is the communications network adequate for the inter-task and inter-ECU communications (message and data passing) implied by the application tasks and their mapping and distribution on the ECUs? This is in

terms of bandwidth, latency, redundancy, and safety considerations. Tasks running on the same ECU may communicate via shared-memory message passing; tasks on different ECUs need to use the system bus.

To provide communications model flexibility for existing and future standards, without imposing an extraordinary modelling effort, Cadence and BMW have worked to create a Universal Communications Model (UCM), which can be used to model several different protocols. This will also allow virtual prototyping of future communications protocols without requiring their detailed implementation to carry out co-design studies and tradeoffs.

Although SW estimation methods can be used in this application, the desired user accuracy for SW task resource consumption requires using a mix of accurately pre-characterised SW task models for known and unchanging functions, with new functions modelled using general and specifically tuned SW estimators.

This approach can best be described as 'co-design of derivatives' based on a co-verification approach using virtual prototypes.

4.3 Digital Multimedia Industry

VCC has been used in the design methodology for platform-based digital video products [6,7] and indeed extensive modelling work has been done to extract high-level statistical design data from design space exploration of a digital multimedia platform. This work on design space exploration has been done in conjunction with Philips. The kind of design used in this study is illustrated in *Figure 4*, which shows application function, platform architecture and a candidate mapping.

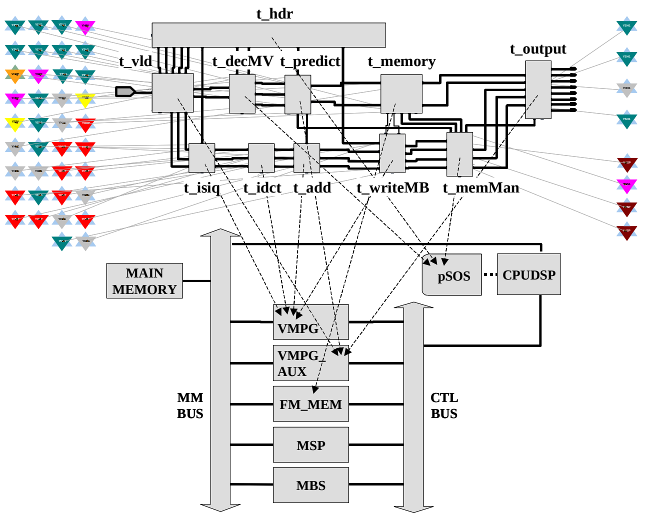


Figure 4: Multimedia application, architecture and mapping

This study has defined the role of the Design Space Architect, who plays a central role in fostering interactions in the product design chain. The design space architect provides quantitative comparisons of various hypothetical products that can be implemented on the platform. This can be used for negotiation between designers at different levels of abstraction: the application, SoC integration and hardware or software module implementation. The guiding principle of the design space architect is to document the entire set-up context and the results of the comparison studies so that they can be reproduced. This ensures that various design teams are working with the same assumptions, and also allows adaptability of the design decisions to market or technology changes that occur during the product development. The design space architect uses a methodology that allows storage, analysis and communication of all relevant information about a virtual product between design teams at their own levels of abstraction. Execution throughput and memory bandwidths specified at the system-level can be unambiguously communicated to implementation teams, and negotiation can take place around a central set of models, analyses and assumptions. The reproducibility of performance measurements allows incremental refinement of design decisions during implementation, as the availability of measurements from designed modules allows back-annotation of more accurate data to the system models.

The design space exploration (DSE) approach adds the following key methods on top of the function-architecture co-design method supported by the basic tool:

- Building a database of 'mappings' of function to architecture. Since design space exploration involves exploring a number of candidate mappings of function onto HW and SW components, and analysis of the communications implications of the same, it is important to build up a comprehensive database of mapping experiments. To assess and fully characterise the platform over its desired field of use, the mappings should include both 'good' and 'bad' experiments (i.e. those which do not meet overall platform objectives). A relational database is used to store this information.

- Specialised statistical probes to extract higher-level data from the various DSE experiments without needing to keep huge files of simulation results, and to allow synchronisation of system state for particular DSE experiments. DSE needs a mapping-invariant execution context to be set up to compare alternative implementations. This approach uses functional or high-level events to synchronise system state, and also gathers data based on these functional events rather than low-level data and control activity. Such probes include observers of CPU, memory and bus activity.

- Gathering statistical data through VCC-based simulation runs of the platform and through pruning the vast amount of output data with the statistical probes. *Figure 5* illustrates output from statistical probes – an example showing write transaction delays per picture frame on the multimedia platform.

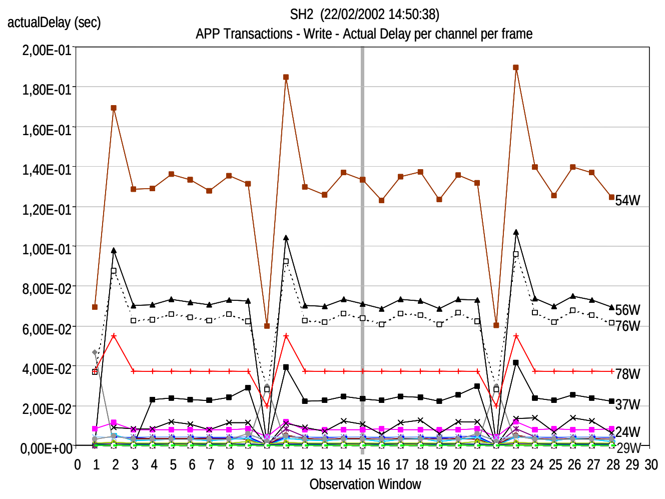
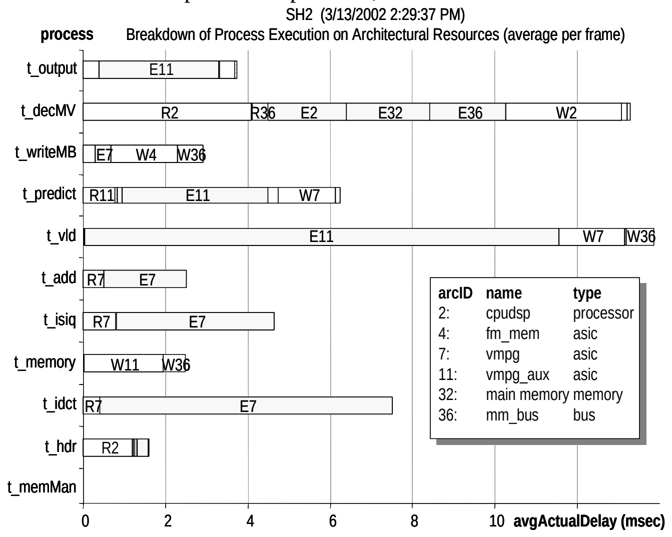


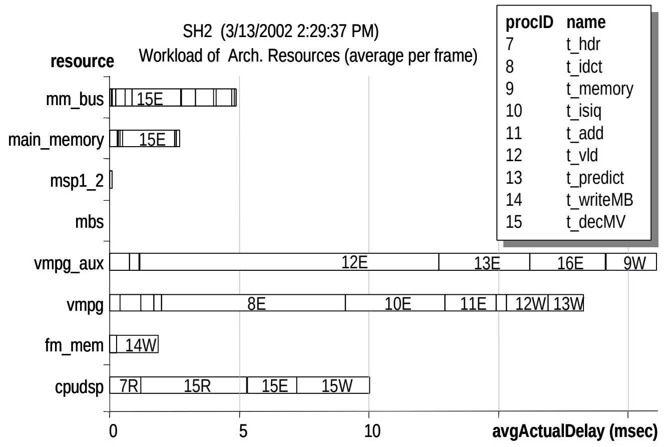
Figure 5: Output of Statistical Probes

- Support for intelligent DSE queries which allow users to enquire for statistics relevant to particular design mapping experiments
- Analysis 'workbooks' that support application analysis (how well are certain multimedia applications supported on the particular platform, in terms of their overall timing and resource consumption and constraints), process analysis (detailed analysis at the task level rather than the overall application), and communications analysis (bandwidth achieved in terms of high level communication transactions). *Figures 6, 7 and 8* illustrate analysis outputs for the resource utilisation, process, and communication bandwidth views.



arcID	name	type
2:	cpudsp	processor
4:	fm_mem	asic
7:	vmpg	asic
11:	vmpg_aux	asic
32:	main memory	memory
36:	mm_bus	bus

Figure 7: Process View



procID	name
7	t_hdr
8	t_idct
9	t_memory
10	t_isiq
11	t_add
12	t_vld
13	t_predict
14	t_writeMB
15	t_decMV

Figure 6: Resource Utilisation View

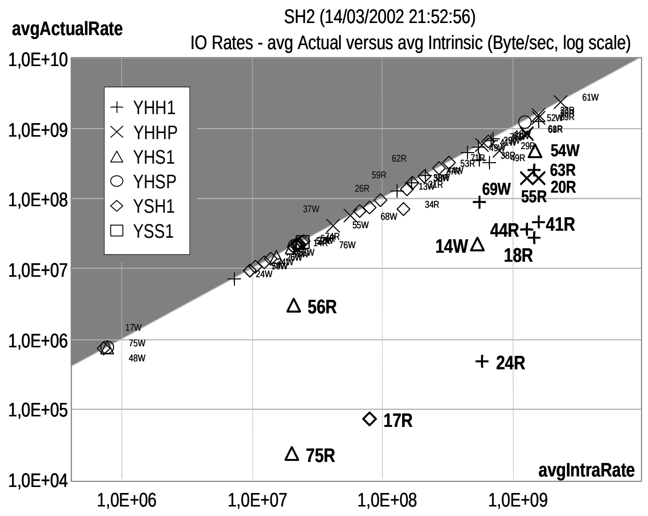


Figure 8: Communications View - Actual vs. intrinsic IO rates

- Advanced statistical analysis techniques that support more sophisticated analysis; for example, Principal Component Analysis in which high-level characteristics can be correlated with various performance measures to understand the key characteristics contribution to overall performance and provide a statistical basis for system level design rules of thumb. Figure 9 illustrates Principal Component Analysis results applied to various communication channels and correlations with design attributes. The relative correlations of one architecture vs. another allow design decisions to be taken at a more abstract level.

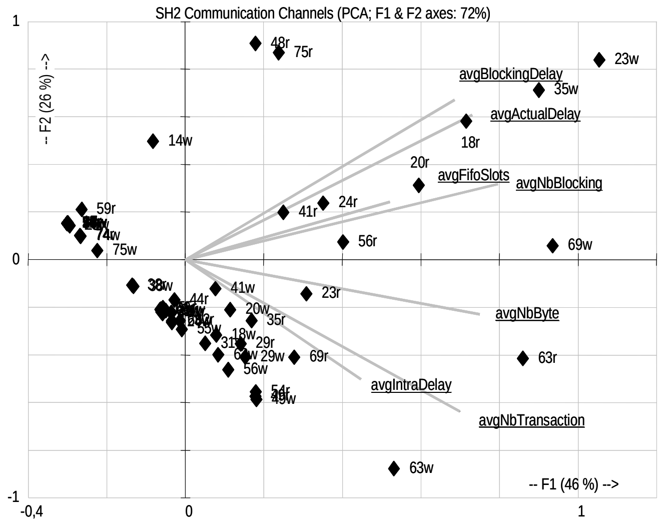


Figure 9: Principal Component Analysis of Communications Channels

With this advanced DSE 'toolbox' it then becomes possible to analyse new derivative applications and standards for digital multimedia, to determine without vast amounts of simulation their fit to the platform and required changes. Essentially DSE moves the extensive analysis required for a specific derivative forward in time so that less simulation is required for a specific product, and the risk of implementing it is reduced.

One area of future extension of this methodology is to look at more sophisticated bundling of multiple SW tasks, so that instead of each functional process mapping to a separately schedulable RTOS task, multiple related processes can be combined into one internally statically scheduled RTOS task. This will allow a more efficient use of system processor resources and the implications of this on overall system efficiency can be studied. Methods such as quasi-static scheduling could be used.

5. SOFTWARE

Our experiments with co-design methods and tools indicate that the methodology is sound and that sophisticated DSE is possible. However, for many applications, which are mostly implemented in software on a well-understood platform, this approach may both be too hardware-centric, and require too much additional effort from system and software architects to be the best methodology.

Alternative approaches that might be considered are the use of rapid platform prototyping, to provide a 'real' or realistic HW base on which to run application software, or higher-level software abstractions, including virtual prototyping. VCC is a kind of virtual prototype; other alternatives are the linking of high-level processor and hardware models using C/C++ and variations such as SystemC. We have already mentioned approaches of the second kind such as Virtio, VAST, and Axys. Rapid prototyping suffers from relatively poor observability and controllability, although able to run huge amounts of system data and very large testbenches. In addition, rapid prototyping demands that the software tasks be relatively close to final implementation in order to run with the RTOS, middleware and Hardware-dependent Software on the platform rapid prototype. While credible late in a design project, it is often not suitable early in design (even for derivative design where most of the SW is based on reused components). The virtual prototyping approaches still are more suited to HW-centric designers.

An alternative approach is to link the HW and SW worlds in a way that stays within the SW modelling and development environments used by SW architects. This includes simulators available within Integrated Development Environments offered by RTOS vendors (for example, VxSim within VxWorks offered by Wind River), and looking for projections of underlying platform characteristics within SW modelling and code generation approaches. The latter includes both SDL (the ITU-T language often used for communications protocols) and UML (the Unified Modelling Language supported by the Object Management Group (OMG)).

Although the OMG is evolving UML to support real-time extensions and related areas of need in its 2.0+ UML standards efforts, (including adding action semantics, additional software component support, and modelling of performance, schedulability and time), there are still some key areas lacking [8]. The two areas which need additional work are platform service descriptions, where platforms consist of 'stacks' of offered services layered on underlying HdS and hardware; and better support for mapping as a refinement approach moving from high-level software models to detailed implementations, via manual or automated code generation.

Further work in this direction is likely over the next several years as the traditional HW-based system design and SW-based high-level SW modelling worlds collide.

6. CONCLUSIONS

In this paper we have talked about a number of platform-based co-development approaches, spanning a variety of co-design and co-verification methods and tools.

What is the likely future of these methods? It seems likely that as additional platforms are defined in a variety of product domains (wireless and wired communications being a strong opportunity, especially at the consumer end), extensive methods for Design Space Exploration during platform creation will be required. By characterising platforms abstractly and statistically, derivative design will be less risky and require less extensive system level simulation or virtual prototyping. Given a derivative design process that concentrates mainly on the fit of a design to the platform computing and communications resources, hardware-based rapid prototyping will substitute for SW-based simulation in the derivative verification process. Thus co-verification will become increasingly hardware-based.

Finally, projections of platform characteristics into SW modelling and automated code generation approaches will allow more optimal SW development processes to be used by more and more SW designers, leading to increases in productivity and true optimising SW synthesis that bring some of the HW flow productivity to the SW world.

It is also possible that in the future, breakthroughs in system design methodology may not come only from tools and models, but from the nature of the architecture itself. We see that in modern processors, it is often vital to co-design the CPUs and the compilers, in order that the concurrent hardware resources offered by multiple processors can be effectively exploited by the software whose access to those resources is mediated by the compiler. In a similar fashion, new architectural concepts, to be effectively exploited by systems designers, may need to be co-developed with the system design tools and methods so that advanced architectural features have practical methods to use them. This may require, in the design space exploration methods discussed herein, development of new measurement methods and their direct support by hardware and software monitors built into the platform itself.

7. REFERENCES

[1] Alberto Sangiovanni-Vincentelli and G. Martin, "A Vision for Embedded Systems: Platform-Based Design and Software Methodology", *IEEE Design and Test of Computers*, Volume 18, Number 6, November-December, 2001, pp. 23-33.

[2] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly and L. Todd, *Surviving the SOC Revolution: A Guide to Platform-Based Design*, Kluwer Academic Publishers: November, 1999.

[3] Jim Tobias (editor), VSI Alliance, Platform-Based Design DWG, *List of Terms and Definitions*, Revision 4, 4 March 2002.

[4] Barry O'Rourke, Paolo Giusto, Thilo Demmeler and Steve Wisniewski, "Rapid prototyping of automotive communication protocols", *12th. International Workshop on Rapid System Prototyping*, 2001, pp. 64 –69.

[5] Thilo Demmeler and Paolo Giusto, "A universal communication model for an automotive system integration platform", *Design, Automation and Test in Europe (DATE) 2001*, pp. 47 –54.

[6] E.A. de Kock, G. Essink, W.J.M. Smits, R. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A.Vissers, "YAPI: application modeling for signal processing systems", *Design Automation Conference 2000*, pp. 402 –405.

[7] J.-Y. Brunel, W.M. Kruijtzer, H.J.H.N. Kenter, F. Petrot, L. Pasquier, E.A. de Kock, and W.J.M. Smits, "COSY communication IP's", *Design Automation Conference 2000*, pp. 406 –409.

[8] G. Martin, L. Lavagno, J. Louis-Guerin, "Embedded UML: a merger of real-time UML and co-design", *CODES 2001*, Copenhagen, pp 23-28.