Overview of the platform based design (PBD) concept related to system on chip (SoC) development

Alexander Larsson  
Mälardalens högskola  
Email: aln04008@student.mdh.se

Adam Edvardsson  
Mälardalens högskola  
Email: aen04011@student.mdh.se

Index Terms—SoC, Co-synthesis

Abstract—This paper is meant to be a brief overview of the platform based design (PBD) concept related to system on chip (SoC) development. A paper "Platform Based Design" by Alberto Sangiovanni Vincentelli [1] lies as a foundation to this work. Alberto’s paper tries to forge a generalized formalized approach for platform based design. In our definition, a platform is simply a transparent abstraction layer which hides implementation refinements details for overlaying layers but allows constrains propagation to underlying layers. When investigating some recently performed projects involving PBD we found out that many parallels can be drawn to Vincentelli framework. Also a brief presentation of tools and framework is carried out because of its importance in the PDB.

I. INTRODUCTION

The design process in embedded systems has been going from a hardware oriented approach to a more software oriented approach in the latest decade. The increased complexity and tighter time to market (TTM) demands is the main factor for this. Today when designing embedded systems, a mixed set of people from both hardware and software engineering are involved. In the earliest approach of mixed SW/HW design the hardware engineers first build the platform which then the software engineers build their application on. Thus the software engineers have to wait for the first "silicone" before they can start debugging; this is off course very time inefficient.

The idea behind Hardware/software co-design is to develop software and hardware in parallel of each other. By providing methods at different abstractions level, methods like modeling, analyzing and synthesis the TTM can be cut. Modeling methods provides means for fast prototyping and simulation. Analytical methods increase the predictability of the system by informing the designer if a system meets it performance, power and size goals. Synthesis methods let the designers rapidity test many design methods. The research field co-design arose in the beginning of the 1990s when the developers of embedded systems started to use microprocessors more frequently in embedded systems instead of just ASIC (Application-Specific Integrated Circuit) [17]. The increased complexity forced designers to analyze the performance in three dimensions; hardware, software and overall system at the same time.

Since that time lots of tools and methods have been developed in the co-design area. The Co-design tasks can be divided into three major areas: requirements modeling & specification analysis, de-sign/synthesis, and validation. In the modeling & specification analysis area high level system description language like SystemC [15], specC [12] and specChart [14] have an important role as they allows the developer to describe the system at a high abstraction level and perform early simulation. Some of those languages have been developed further [5] so they are capable of generate readable and synthesizable hardware description language (HDL) code. For design / synthesis for real-time and reactive systems description the synchronous programming language like Lustre, Signal and ESTREL [8] are widely used in the co-design tool chain.

At different steps in the design cycle, different standalone tools are used. Therefore special unified frameworks like POLIS [16], have been developed to combine different tools and methods to keep a unified framework. From a specification written in a high level language POLIS translate the specification model with (ESTREL) to Co-design finite state machines CFSM [13], then VIS is used for a Formal validation. PTOLEMY [11] are then used for simulation. Other features of POLIS are Design Partitioning, Hardware Synthesis and Software Synthesis.

The design process in embedded systems has been going from a hardware oriented approach to a more software oriented approach in the latest decade. The increased complexity and tighter time to market (TTM) demands is the main factor for this. Today when designing embedded systems, a mixed set of people from both hardware and software engineering are involved. In the earliest approach of mixed SW/HW design the hardware engineers first build the platform which then the software engineers build their application on. Thus the software engineers have to wait for the first "silicone" before they can start debugging; this is off course very time inefficient.

The idea behind Hardware/software co-design is to develop software and hardware in parallel of each other. By providing methods at different abstractions level, methods like modeling, analyzing and synthesis the TTM can be cut. Modeling methods provides means for fast prototyping and simulation. Analytical methods increase the predictability of the system by informing the designer if a system meets it performance, power and size goals. Synthesis methods let the designers rapidity test many design methods. The research field co-design arose in the beginning of the 1990s when the developers of embedded systems started to use microprocessors more frequently in embedded systems instead of just ASIC (Application-Specific Integrated Circuit) [17]. The increased complexity forced designers to analyze the performance in three dimensions; hardware, software and overall system at the same time.

Since that time lots of tools and methods have been developed in the co-design area. The Co-design tasks can be divided into three major areas: requirements modeling & specification analysis, de-sign/synthesis, and validation.

In the modeling & specification analysis area high level system description language like SystemC [15], specC [12] and specChart [14] have an important role as they allows the developer to describe the system at a high abstraction level and perform early simulation. Some of those languages have been developed further [5] so they are capable of generate readable and synthesizable hardware description language (HDL) code.

For design / synthesis for real-time and reactive systems description the synchronous programming language like Lustre, Signal and ESTREL [8] are widely used in the co-design tool chain.

At different steps in the design cycle, different standalone tools are used. Therefore special unified frameworks like POLIS [16], have been developed to combine different tools and methods to keep a unified framework. From a specification written in a high level language POLIS translate the specification model with (ESTREL) to Co-design finite state machines CFSM [13], then VIS is used for a Formal validation. PTOLEMY [11] are then used for simulation. Other features of POLIS are Design Partitioning, Hardware Synthesis and Software Synthesis.

Today's system on chip (SOC) embedded system become more and more popular. SoCs are highly configurable systems where it is possible to implement new CPU's, RAM, ROM and custom hardware as components on a single chip. The increased possibilities also increase the complexity and thus the pressure of a good design method. The line between software and hardware becomes smaller. Different co-design methods for SoC have been used during the latest decennium; among those are time driven and block-based design. Next step in the evolution of design methods are the platform based design method (PBD).

In this paper we will look closer on the PBD area, the pros and cons of this method, what problem they solve and what problems exists with them. Looking how it co-operates with the goals of co-design of embedded system, short TTM to a low cost with reusable components.

The Outline for this paper are as following, first some background of the most common design methods so far, to see why PBD is needed. Then we are presenting an overview of the PBD area, starting with describing the baseline for PBD, continuing with some investigation of project using PBF, and
finally we are looking at some tools and frameworks for PBD.

II. BACKGROUND

A. Time-Driven design (TDD) [7]

TDD was invented in the middle of 1990, with the basic idea to consider the timing aspects at all levels of the design process. Prior to this the system designer first built the system to be functionally correct, then tested for correct timing behavior. Incorrect timing behavior forced the engineer to go over the implementation again and fine tune sections. This method created area effective solutions, but took too much time to use and was therefore not suitable for bigger systems. With the increased complexity better methods were needed, this is where the TDD came in the picture.

B. Block based design (BBD)[9]

Some of the biggest drawbacks with the TDD method is its low level of abstraction, the lack of design reuse and it has difficulties with high complex problems with many logical gates. Another problem with TDD is that it is too hardware aimed, making it hard for designers without hardware oriented background. Block based design methods builds on the reuse of intellectual property (IP), going towards the same code reuse philosophy as in software engineering. IP block reuse dramatically shortens the development time and the components are well tested and documented. BBD is a model at behavior level where simulators and emulators are used for co-verification of software and hardware.

C. Platform based design (PBD)

BBD is not suitable for bigger projects (around 1.5M gates) [8] or where the design is performed by multiple groups. PBD combines the best part of TDD and BDD, it has extended support for design reusability. The main idea is a meet in the middle approach bottom-up top-down approach.

The evolvement from Time-Driven design to Platform-Based design is illustrated in figure 1.

Platform-based design has emerged over the years as a way of coping with the Problems listed above. The term ”platform” has been used in several domains: From service providers to system companies, from tier 1 supplier to IC companies. Platform based design has been around for several decades, approached in different ways in different fields. This lead to a heterogeneous view of the platform and the PBD concept. In the same paper ”Platform-based Design” by Vincentelli a generalized formalized approach for platform based design for electronic systems composed by software and hardware are presented. By setting up a homogeneous definition of the PBD concept he urges to forge a ground for continued design-technology, research and practices.

PBD is depending on the definition of platform, Vincentelli denote a platform as ”abstraction layers that hide the unnecessary details of lower level of abstractions”. Parallels are drawn to the PC platform, where you can change component independently of each other (As long the interface are the same). This is the main concepts in the PBD, to divide the design process in to different abstraction layers called platforms. Each layer (platform) support a design stage which provide a transparent abstraction to lower level for possibility to accurate performance estimations. A platform consist of a set of library components together with their rules. Both functional and communicational components are provided, those can be combined in different ways for solving complex problems. For every platform level there are methods to map to lower level of abstractions and to carry out performance estimations. One pair of platforms, the tools and methods that are used to map the upper layer of abstraction into the lower level creates a platform stack (fig 2).

PBD builds on a meet in the middle approach, a combination of the top-down and bottom-up approach. Top-down maps from higher to lower platform instances and the bottom-up approach build a platform by defining its components and their
The degree of freedom to systems which have the same instruction set architecture. An effective platform has to offer a design environment to cope with development costs. The Architecture Platforms is in need of a software development environment and a set of tools that insulate the details of the architecture from application software. This is leading us to the next platform, the API platform.

B. API platform

The goal here is to hide as much of the architecture implementation details as possible for the software developer, a high level interface Application Programmer Interface (API) is available to the software developer for communication with the hardware. The architecture platform is hidden with a software layer. This allow the programs to be used with different architecture platform, just the software wrapper has to be changed. However this is still a complex and time-consuming task but it’s doable.

C. System Platform-Stack

The idea behind the system platform stack is shown in figure 1 to combine application platform with the API platform. The designer describes its application on an abstract level on the API platform. From here an execution model that estimates the performance of the lower level architecture platform, is exported. The execution model may include size, power consumption and timing estimations; variables that are associated to the lower level abstraction. This can’t be calculated at API level, but we can pass information to lower abstraction level, (recall fig 2), in return we get performance estimation which can be used to refine the process.

The real mapping to an actual architecture can be carried out partly automatically with tools, one tool for each part of the software layer (software synthesis, RTOS synthesis, device-driver synthesis). The tools have to combine the feature of the architecture and the API to be able to translate the application to the required architecture.

The architecture search space depends on how abstract the programmer’s model is. A highly abstract model is fast to create but give a wide set of plausible platforms, thus make it difficult to find the optimal implementation.

In summary, the system platform-stack is a comprehensive model that includes the view of platforms from both the application and the implementation point of views. Note that the system platform effectively decouples the application development process (the upper triangle) from the architecture implementation process (the lower triangle).

D. Silicon Implementation Platform Stack

"The Silicon Implementation Platform (SIP) Stack of the design flow takes all components of the architecture platform and maps them to a physical implementation”. SIP platform builds on the same principles as the system platform stack above. Here with architecture as the top level and silicon implementation as the lower level platform. Also here information is sent from upper layer to lower, to allow the SIP platform to find a particular layout of the architecture platform instance that can be used. It can also send information upward.
IV. PBD IN REALITY

In a real design environment it’s not common to see a pure PBD divided like this, but many of the basic elements from this earlier work are present. For this reason we studied some small newly performed projects where PBD was used. We wanted to see what kind of parallels and similarity can be drawn to A. Vincentelli framework for PBD.

A. SoC Platform Based Design of MPEG-2/4 AAC Audio Decoder

The design process was divided into 5 different steps; the first step was a high level modeling in the C-language. The designer chooses a suitable platform that he is suitable for the application, they choose the ARM (Advanced RISC Machine) platform. Noticeable here are the similarity with the architectural platform. Next step in their process was the Porting and Profiling, translating the model to be runnable on the ARM and make modification to the model to suite the architecture. In other words moving down in abstraction and on the same time proving the higher level with additional information. The third step contained the division hardware and software; here 4 different mixed software & hardware approaches are tested for their performance. The hardware/software partition design is a tradeoff between pure software and pure hardware method. The next step is the implementation of the two partitions to respective SW and HW. In the last step the entire system is integrated on the ARM platform. A connection between the software on the CPU and the custom hardware FPGA has to be implemented. Step 5 here is what A. Vincentelli in his model called the “Implementation Platform”.

To assist in the decision between which implementation to utilize SoC Platform Based Design of MPEG-2/4 AAC Audio Decoder[4] defined the gain as the ratio between the increase in speed from implementing a module in hardware and the increase in complexity[4] by making the implementation in hardware. The gain for various configurations was calculated and the solution offering the highest gain was implemented.

In their work SoC Platform Based Design of MPEG-2/4 AAC Audio Decoder[4] notes that hardware implementations has a speed advantage but software has a flexibility advantage. As such dedicated hardware should be utilized for heavy computational tasks and software implementations should be used for complex but computationally cheap tasks.

B. Platform Based Design for Automotive Sensor Conditioning[21]

A mixed signal platform based design methodology for a automotive sensor interface are being presented. An interface for sensors is designed, where the modeling of the sensors is performed at high abstraction level in MATLAB. Then it is co-simulated in a process that changes its behavior by rewarding or punishing a subject each time an action is performed until the subject associates the action with pleasure or distress helping the designer to find the most appropriate elaboration chain for the given sensor. By simulation design iteration the entire project space are explored. Then the model can be portioned down to software and hardware. Further optimization is performed by parallel information exchange between the different platforms. The above mentioned design flow ends up with the prototyping phase, through which we can quickly test the whole system (or parts of it) in real operating conditions. Also this project describes a typical Architecture-SIP stacks design.

V. THE IMPORTANCE OF TOOLS IN PBD

PBD require lots of different tools for a good result, a common problem here is that the tools in the tool chain are poorly integrated with each other. To get ride of this problem, some work has been done to create frameworks. The DISYDENT[5], HSCDE[11] and Metropolis[6] are some of them.

A. Frameworks

1) Metropolis: Metropolis is a design environment that provides a uniformed integrated support for analysis, verification, and synthesis tools. This framework is an expansion of the earlier mentioned POLIS, therefore builds on the same ideas. Metropolis “seek to avoid imposing a specific language or flow model on designers, instead making their preferred approach more robust and rigorous”

Unfortunately it is hard to put a crisp number on complexity.
2) DISYDENT: DISYDENT, a framework dedicated to system-on-a-chip (SoC) platform-based design for shared memory multiple instructions multiple data (MIMD) architectures. It requires strict definition of the system HW, components, communication, HW behavior has to be in RTL (Register transfer level) level, this framework is a complicated setup but seems to provide much support for the design of highly complex parallel applications.

3) SystemC: PBD maps a previously specified behavior (e.g. with SystemC) to a specific architecture selected by simulation tools, as shown in the system platform stack.

The role of the system level description language is important in platform based design because the design-flows are changing toward an integrated framework. Co-design, simulations, synthesizing, operates together. Therefore SystemC will play an important role in the future as it is an integrated framework that handles the performance analysis and verification from a system level specification.

SystemC is an extension to the C++ programming language that enables modeling of hardware properties on a high level of abstraction.

Carsten Schmitt claims that as SystemC is an executable specification so it is simple to test a specification written in this language by simulating it. Furthermore the programmer will also have the benefits of built in C++ optimization done by the compiler. Another advantage brought forward is that SystemC uses a Single Source Approach, meaning the same specification is utilized for all steps.

The first step in synthesizing products in SystemC is to make a pure specification (specification level) disregarding which processes should be implemented in hardware and which should be implemented in software. The result of this will be a executable file that a standard personal computer can execute. Next step would be to decide which processes should execute in hardware and which processes should execute in software, and to finally make the implementation. After partition the code is at algorithmic level, at this level some of the SystemC constructions needs to modified. The modification is handled on the library level which makes the change hidden from the designer who needs not to concern himself/herself with this task. As the change is done on library level the original source code remains unchanged. When generating code for the final product (generation phase) processes marked as implemented in hardware are discarded from the code generation. When generating code for the generation phase SystemC is modified to act as a wrapper for the underlying RTOS (realtime operating system), that is SystemC calls the functions of the RTOS in such a way that the end result corresponds to the SystemC specification.

At algorithmic level it is possible to co simulate the system utilizing an RTOS to SystemC kernel interface. The interface models the relationship between the hardware and the software running in the RTOS.

Most embedded software is manually written from the system specification. The resulting code will include several RTOS calls for synchronization and concurrency. When compared to the equivalent SystemC description a very high correlation is found, the RTOS functions that supports concurrency and synchronization are very similar to the SystemC calls. When working at the specification level, SystemC works like a operating system handling concurrency, synchronization and miscellaneous other tasks generally handled by the operating system.

Disadvantages with SystemC include it is unable to accurately model a RTOS on a GPOS (general purpose operating system). SystemC also incurs a constant 50Kb overhead. Dynamic memory overhead is mostly composed of communication channel overhead which increases nonlinearly. The other component depends on which functionality is included in the project, a test showing an increase of 1Kb overhead for every 50-70 lines of code. However overall overhead is negligible because the overhead is not the overhead of SystemC, rather it is the overhead of the implementation of SystemC on the RTOS which is unavoidable.

4) HSCDE: HSCDE divide the SoC co-design process into three levels, system-modeling (level 1), virtual components (level 2) and real components (level 3). HSCDE support mapping between those levels by Design Planning and Virtual-Real Synthesis. At the first level, the system modeling level a Constrained Taskflow Graph is used. The second level acts as an abstraction layer for the real components and help reduce complexity. The final level is the actual design and hardware utilized. Premade components are stored in a component library to facilitate reuse.

Xiong considers system modeling to be the first problem that should be resolved. The model should describe system functionality clearly, but also performance constraints such as power and speed as described in Xiong "A platform-based SoC hardware/software co-design environment"

Traditional system models (such as discrete event, Finite State Machine, Data flow/Control flow, Petri net) supports behavior description, they can’t be used to describe performance constraints.

HSCDE provides automatic partitioning into hardware and software. When moving from virtual components level to real components level the virtual components used in the model are mapped to real IP cores. Hardware components are mapped to hardware implementations that meet the functional and performance requirements while software components are mapped to software processes running in a RTOS. Next HSCDE generates the communication interfaces utilizing interface synthesis technology. In the third step the software source code is compiled to executable binary code and optimized. Finally the performance constraints given to the virtual components are transformed to the real components.

VI. DISCUSSION

PBD might not be the final solution to the development of SoC, but it provides allots of valuable methods and ideas. One thing it has been criticised for is that it favours software implementations, this is a drawback for SoC developer. It generally require a lot of different tools, different models...
and the division to different abstraction layers. So for smaller project PBD has not so many advantages over BBD and TDD methods. The strength comes in when the design can be composed by many specialised sources, when dealing with complex problems the high abstraction is appealing. Also when building system platforms that is going to be used for futures product like a cell phone backbone, PBD is strong due to the future opportunities of component reuse.

The main challenges in adopting this methodology are all related to the lack of precise definitions and characterization of platforms and of the associated design flow in the industry today.

VII. FUTURE RESEARCH

The entire path from system to hardware is pretty well documented so far. The automatic generation of communication with hardware is also well documented. However documentation on the hardware side is lacking. All silicon design breakthroughs happen at the architectural level, and these changes can be rapid in SoC environ and the remapping to other architectures is not possible if the system was built without the API platform. Even if it was its not sure if it can be mapped automatically as it require special tools. In general we feel that the entire field dealing with hardware implementations in respect to SoC is lacking. [4] brings up the complexity of a hardware implementation, how exactly do you measure complexity in hardware and software?

Even if current frameworks and design processes gives the impression of being mature, we believe that they may be improved on even further especially in the hardware oriented direction.

VIII. CONCLUSION

In this paper we have briefly summarized time based design and block based design. We did also summarize the platform based design concept and highlighted some tools, frameworks and methods used in synthesis. Historically what exactly constitutes platform based design is in the eye of the beholder, but with A. Vincentelli work a conceptual view was forged. The key for platform based design is the platforms division, the meet in the middle design flow and the successive refinements of specifications. His viewpoint and notations can be distinguished in project presented nowadays, as we saw when we studied the different projects. Numerous similarities were found, however just simple and smaller project was studied, which only explored a few abstraction layers (platforms). A study of more complex problems should give us more information how other platforms layer can be used.

Some tools like HSCDE will automatically try to divide your specification in hardware implementations and software implementations. In this paper we have briefly summarized time based design and block based design.

REFERENCES


[2] Carsten Schmitt, Embedded Software Generation from SystemC for Platform Based Design


