Subsystem Exchange in a Concurrent Design Process Environment

Marino Strik[1], Alain Gonier[2], Paul Williams[3]
[1]NXP Semiconductors, The Netherlands; [2]Mentor Graphics, France; [3]Mentor Graphics, UK

ABSTRACT

This paper provides insight into the novel solutions used to build SoCs targeting increased productivity in a complex environment. Design of such SoCs relies on multi-team, multi-site cooperation and data exchange. The data exchange, made possible though descriptions based on The SPIRIT Consortium's IP-XACT™ specification and the automation for its processing, forms the basis of the approach. Initially, the specification focused at IP reuse; this has now been extended to SoC subsystem exchange. This paper also describes state-of-the-art subsystem design automation and improvement opportunities, based on a close collaboration between NXP Semiconductors and Mentor Graphics. We do not cover all the aspects of reuse but mainly stress the concurrent engineering process.

I. INTRODUCTION

In a June 2007 EE Times article, Ron Collett, president and CEO of Numetrics Management Systems, Inc. presented his views on IP reuse, which were "based on 1,200 benchmarked IC projects from more than 35 companies: 'There are good and bad news about the reuse situation. Over the past ten years, reuse leverage more than doubled, and more reuse tends to translates into less project effort, shorter cycle times as well as fewer spins and less schedule slip.' Still on the positive side, Collett indicated that the average transistor count per block is growing and the number of blocks per chip is rising. Moving to bad news, Collett noted that the average team size has doubled between the years 2000 and 2006."

This is not cost-effective in today's electronic consumer market where cost pressure is at its peak and margins are tight. Companies often attempt to lower their cost of production by investing in design centers in emerging countries, which scatters the design teams all over the world. In that context, efficiently exchanging and reusing information between the dispersed design teams is even more challenging. Having a common development environment and a consistent way of exchanging subsystems is key to keep up the productivity and time to market brought by IP reuse and flow automation[1].

"Collett also deplored that the semiconductor industry has serious schedule slip problems. About 85 percent of all IC projects miss their original schedule. 'This is chaos. The average schedule slip is 44 percent, and high schedule slip means poor schedule predictability" with a direct impact on cost and margins.

II. THE IP-XACT SPECIFICATION, THE MEDIA FOR EFFICIENT EXCHANGE OF INFORMATION

We stated the need for concurrent design of subsystems and the exchange of design data between sites. Until recently there was no industry-wide answer on how to document IPs and ease their integration into system-on-a-chip (SoC) products usually formed by IPs from multiple sources as illustrated in Figure 1.

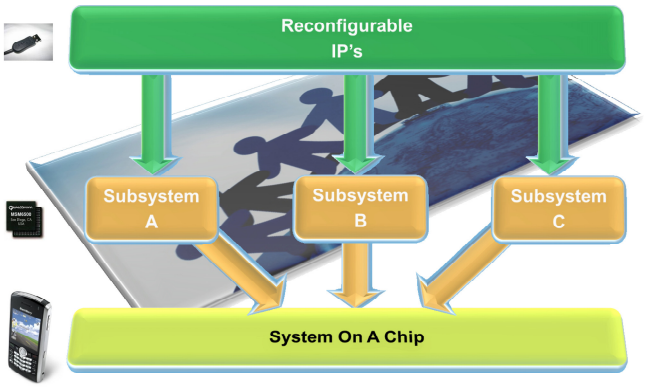

Fig 1. The SoC development eco-system

The SPIRIT Consortium's IP-XACT specification changed the situation. Here EDA vendors, IP providers and SoC integrators cooperate to define the answer to describing IP properties in a common way and ease its integration [2][5]. Tools based on the IP-XACT specification, such as Platform Express™ from Mentor Graphics, are capable of providing an eco-system where one can easily exchange information around an SoC and rapidly produce subsystems and eventually the final SoC. This would have previously required significant integration time and effort in a traditional design flow.

IP and design information such as interface signals and memory map, captured in the IP-XACT format using the XML language, document the IP in a standard and exchangeable way. The IP-XACT specification enables the

978-3-9810801-3-1/DATE08 © 2008 EDAA

XML elements of the specification to be extended to describe information that is not covered through available IP-XACT semantics.

The ability to exchange data is not enough. Productivity increases further if the processing of the exchanged data is automated. Or, to be more precise, a flow and methodology is needed that can process the data coming from multiple sources and which can effectively transform the input data into an SoC. This flow and methodology needs to address assembly, documentation generation, verification, and synthesis [3]. Automating the processing is important in order to be able to quickly handle updated versions of subsystems and generate consistent outputs. The IP-XACT specification, by providing the design context information, enables such an automated flow. The scope of automation is wide; it includes all steps from IP selection and configuration to ready-for-placement netlist with accompanying timing constraints. For instance, SoC assembly, verification, synthesis, and dft insertion all fall within that scope Future work may extend the scope to layout. For now, the ready-for-placement netlist with timing constraints provides a clear and well defined challenging target.

The following sections describe our experience of managing the integration of subsystems provided by multiple, disperse teams. Each subsystem is created according the novel concepts based on the IP-XACT specification. A key criteria for success is the smooth handover of information, allowing verification, documentation and implementation flows to run automatically.

III. The Concurrent Engineering Process Equation

For reuse to be efficient, a repeatable and established methodology is essential. According to Collett, the problem lays in the difficulty to estimate design complexity and especially the impact of reuse. He declared: "The problem is that reuse expectations typically exceed reality, and there is a difference between the assumed effort required and the actual effort required. The effort behind reusing blocks is underestimated, and a leading cause of poor schedule predictability is the inability to assess the impact of IP modifications on project effort."

The two key areas of focus for leading-edge semiconductor firms are schedule predictability and risk management of large platforms or projects, declared Collett.

Although there are good evidence that reuse is working, development productivity is not keeping pace because of team size increase and poor schedule predictability. And, the misunderstanding interplay between reuse and the effort saved is a major cause of poor predictability, Collett summarized.

A. Sharing subsystems across teams

Teams exchange subsystems in many ways. These differ mainly by the amount of flexibility of the subsystem, in other words, how much the end-user is allowed to reconfigure the delivered subsystem.
The three major use models are:

1. Soft core use model
   a soft core use model is applied if the end-user prefers to take ownership of the product. The design sources are transferred with permission to be modified and the warranty from the IP provider is limited. Updating the delivery with incremental releases from the IP provider can get complicated if the end-user changes are not automatically reproducible.

2. Firm core use model
   a firm core use model is applied if the end-user prefers not to take functional ownership of the product. The design sources are transferred, including a synthesized gate level netlist which meets a mutually agreed performance level (area and timing). There is no permission to functionally modify the design as functionality is guaranteed by the IP provider. The end-user needs to integrate the subsystem and create a chip layout, which may require standard cell netlist changes that maintain functional equivalence. Updating the delivery is not as difficult as with soft cores. A new netlist may replace an earlier version especially when pin compatibility is maintained.

3. Hard core use model
   a hard core use model is applied if the end-user prefers not to take ownership of the product. The design sources are transferred, including a synthesized gate-level netlist and a layout view that meets a mutually agreed performance level and footprint. There is no permission to modify the hardened design. Updating the delivery is comparable with the firm core use case. A new layout may replace an earlier version especially when footprint compatibility is maintained.

1) Data Management

Receiving updated data multiple times may require a significant effort to correctly and efficiently apply version management. Two different releases of a subsystem will typically contain many files with the same names, with or without updated content, and only a few newly created or removed files. It is difficult to deal with this difference reliably at the individual file level. Therefore, it is

recommended that the number of elements to be individually updated remains limited. This is handled by treating one or a few directories and their content as the elements to be updated as depicted in Figure 2.

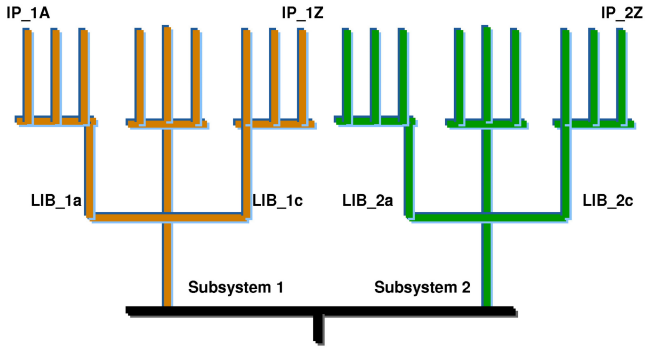

Fig 2, Directory representation of two subsystem structures allowing for easy configuration management

The fact that files are not shared across different subsystems is also important. The integrator will need to agree with its suppliers which (top level) directory names they will use.

2) Naming Conventions

Commonly used languages like verilog define that the last compiled description overrules all earlier descriptions with the same name. However, in order to guarantee that all sub-designs with the same name from different suppliers have identical content, name clashes for all hierarchical design names must be avoided. Using unique names during integration is not an attractive scenario. It requires name clash detection and modification of deliverables which later, for example, could hamper reporting problems to a supplier or violate warranties for firm and hard core use models. Hence it is best to make a serious attempt to avoid name clashes through upfront naming agreements. This requires subsystem providers to fully control the naming of their design elements. If subsystem creators apply reusable IP, then that IP must support configurable names.

3) Use of IP-XACT

Before we continue, it is important to have a common understanding of IP and subsystem terms:

- We define **IP** as being a basic block such as UART, I2C, bus matrix, etc.
- Assembling IPs together is creating a **subsystem**, which can be viewed as a larger IP.

The IP-XACT specification differentiates between component and design XML descriptions:

- Component XML description details the hardware and software interface, focusing on the information required for integration.
- Design XML description details a composition of components, its connectivity, and the component configuration parameters that are specific to each component instance. For integration purposes a component view of the subsystem needs to accompany the design XML.

To distinguish the two different component types, we introduce the term leaf component for component XML that describes a (reusable) IP and which is not a composition of IP described through an IP-XACT design.

Configurable IP is characterized by its IP-XACT Vendor, Library, Name, Version (VLNV) and parameter value pairs. IP parameter values such as IP base address can be manually set by the user or automatically calculated by a generator using context information to compute its value. Configurable IP are explained in greater details in section B.

Except in the case of the softcore IP, parameters are currently not available up in the hierarchy. Hence composed systems are not configurable when being instantiated. For integration of firm or hard subsystems, change of context will not affect parameters value. Hence, all configurable IP parameters must be resolved as should parameters that are derived from the context prior the delivery. There are 2 options to enforce this property:

1. For all IP inside a firm or hard subsystem, the context analysis can be switched off. This will ensure all parameters remain fixed. An integrator can potentially be confronted with issues during functional verification which may be too late and difficult to translate back to the root cause.

2. The context analysis can also be executed as normal, where the process reports an error message as soon as a parameter value is derived, which is different compared to the original value. Preferably an error message is presented only if the context is incompatible with the subsystem. At this moment it is not yet well understood how to automatically distinguish a compatible from an incompatible parameter value. An error message should report to the system integrator which IP and parameter combinations are subject to value change. An integrator will need to decide whether to continue or address the issue.

Having IP-XACT standardized information about parameters that are to be changed supports communication with a supplier of a subsystem. In the scenario where the subsystem is fixed, the IP-XACT parameters help adjust the context of the subsystem.

Today we need to use XML component vendor extensions to qualify a subsystem soft/firm/hard property as the specification does not have dedicated XML elements to hold

this information. Whether there will be a need for having an end-user softening capability where a hard core can be used as firm core or where a firm core is used as soft core is not clear yet, although technically this is certainly possible. The organizational and warranty-related consequences need be further investigated.

B. *Sharing components across subsystems*

Sharing components across teams is fundamental in the concurrent engineering process. One key aspect of IP is how one deals with its configuration. The following sections address what to consider when sharing IPs across subsystems.

1) *Set user configuration boundaries properly*

Having highly reconfigurable IP for extensively reused functional blocks is a must. It ensures that everyone use the same source of IP, and, as a consequence, raise the IP quality and team productivity as everyone contributes to make the IP bug free and repeatedly reused without problems. As stated earlier, we are only covering the concurrent engineering process and not all aspects of IP reuse. If we look at Figure 3 below, we can categorize IPs according to flexibility in parameters and interfacing.

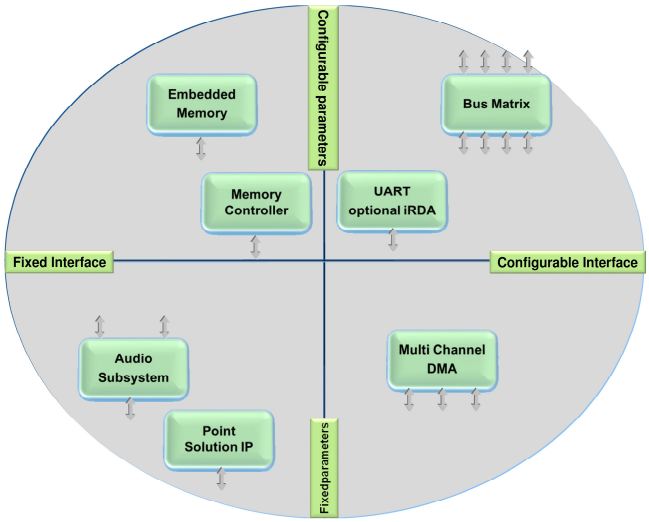

Fig 3. IP Category in terms of configurability

Highly configurable IP will mainly help the subsystem provider to quickly create and configure its subsystems. The subsystem provider, who has a deep knowledge of his/her subsystem, will then be able to quickly derive and generate different implementations and rapidly satisfy end-user requests. In that sense, configurable IP will primarily serve the subsystem provider and have little impact to the end user of the complete subsystem that will not have any or limited access to the IP configuration.

If you take the case of an IP on the right hand-side where its external interface can be modified through configuration, one may consider not giving the end-user access to these parameters as it will modify the internal connections and even potentially modify the subsystem external interface. It is the duty of the subsystem provider to set the right boundaries of the end-user configuration.

2) *Ease configurable IP integration*

Figure 4 illustrates the degree of automation one can expect according to the type of IP used. Typically there are two things that can affect an IP configuration:

1. Context information (data bus width, other IP base addresses)
2. User information (optional IP features)

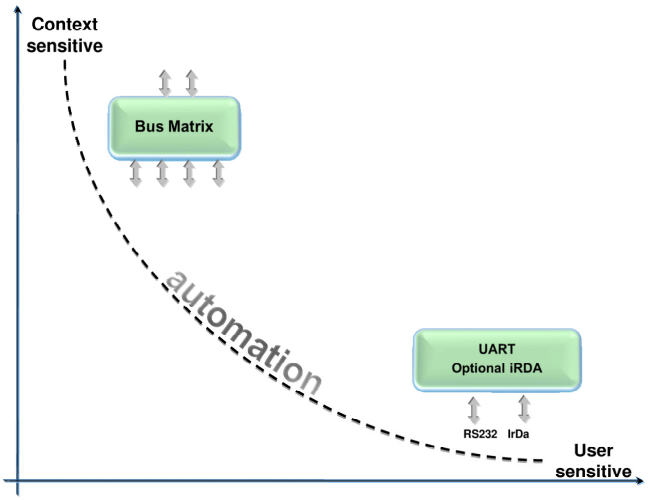

Fig 4. Degree of automation versus IP category

IP such as bus infrastructure matrix can get most of its configuration parameters from the design context and therefore, with a high degree of automation, will require fewer to no inputs from the user. In the latter case, as shown in Figure 4, automation is nearly impossible for parameters which are leaves in a dependency tree and for which a value is an architectural or feature choice.

In a nutshell, the more design-context sensitive one's IP is, the easier it will be to automate its integration into the subsystem. Conversely, the more user-defined one's IP is, the less one will automate and thus will rely on user entry to make it work correctly in its subsystem.

The IP-XACT specification supports both user and context-sensitive configuration so that IP developers can provide a flexible IP whilst preventing non-desired configuration by embedding automation to configure IP parameters depending on the design context. Take the example of an interrupt controller that adapts the size of its decoder according to the number of connected interrupts.

The size of the interrupt bus will vary according to the number of interrupts in the design, and, while the internal behavior will be identical, the interface will differ from one SoC to another; the internal structure of the IP will differ as well, as the decoder logic will adapt to the number of interrupt lines to decode. The integrator's only duty is to connect the different interrupt lines to the interrupt bus interface of the interrupt controller; the corresponding HDL will be generated accordingly with the corresponding interrupt lines and internal decoding logic.

C. Use of Hierarchy Efficiently

The primary purpose of design hierarchy is to provide a mechanism for reducing design complexity by partitioning a design into smaller, more manageable subsystems. These subsystems can then be allocated to separate design teams, each focused on the particular issues associated with that part of the system. Hierarchy can also be used to demark layout requirements, power domain systems, test requirements, etc., but experience shows that visualizing multiple, different hierarchies on a single system adds considerable complexity and should be avoided wherever possible.

1) Approaches to design hierarchy.

Two approaches are common for implementing hierarchy in a complex design. It is worth studying each of these approaches because it helps identify the underlying requirements of subsystem design:–

a) A tagging approach.

Here the whole system is structured as a large flat design, and a combination of tags on components and filtering makes the design appear as hierarchical to tools like viewers and HDL generators. This approach provides considerable flexibility in design manipulation because changes can be achieved by simply moving tags; however, it does not meet the requirements of subsystem design because it does not offer a mechanism for multi-site isolated development and integration.

b) A design cross referencing solution

The alternative approach is that all levels of hierarchy become self-contained designs, and that each design is allowed to cross-reference other designs to build up a hierarchical structure. This is essentially the approach adopted by the IP-XACT specification in that designs contain components and components can have views that are themselves other designs. For sub-system design, this approach has the advantage that it scales; components can be combined to form designs that are wrapped as components, which can then be combined into larger systems.

2) IP-XACT hierarchy

The IP-XACT approach to hierarchy is therefore well suited to partitioning a large design into separate subsystems, which can be designed at different locations and then integrated. A cross-referencing solution allows subsystems to be multi-instantiated in a design, which is powerful but brings its own issues:

a) Consistency Issues

Successful integration of systems requires that each sub-design is created with a level of consistency. The IP-XACT specification greatly helps in this area because it describes interfaces, filesets, and other characteristics of the sub-system in a consistent XML format. However, the specification has very strict rules of consistency that must be followed precisely by all sub-design teams, chief of which is bus definition (busdef) standards; these must be identical between two communicating IPs. Two identical bus definitions from different vendors, or two bus definitions with very slight version number differences, cannot be used for communication between IP blocks.

b) Interface changes, revision control and synchronization issues

A common problem with sub-system integration is the challenge of maintaining a consistent top-level design when sub-systems are being modified and supplied in various stages of development. Changes that modify the interface of a sub-system will impact the integration level of the design; interfaces that are added need to be connected, interfaces that are removed need to be disconnected, and knock-on effects of these changes need to be processed. Keeping the whole design synchronized to changes occurring in a sub-design is a significant integration challenge.

c) Configurable designs issues

Some changes to sub-designs (such as the addition of a new bus interface) have an immediate and obvious impact on integration; other changes are more subtle -- a change in the base address of a component will definitely change the memory map of the whole system and may change the hardware decoding of bridges outside the sub-system. To add to the level of complexity, the subsystem may not be delivered as a fixed configuration; it may be possible to modify certain parameters which impact the content of the sub-system and again synchronization of those changes in higher levels of hierarchy. Worst case scenario, some IPs (such as bridges) modify their configuration not by user input, but by examination of design context (e.g. a memory controller may modify the number of address bits it provides based on the size of memory connected to it). Extreme care must be taken where such design context information is gathered from outside the sub-system.

d) Hierarchical manipulation issues

The IP-XACT specification's representation of hierarchy as components with design views is a rigorous method of documenting existing structures and is well suited to delivery of sub-systems. However, being rigorous, it lacks some of the freedom and ease-of-use that is desirable when creating new designs. Restructuring hierarchy, by moving components from one design to another through existing hierarchical boundaries, will have an impact on the source design, the destination design, and all other designs that are touched by the changes to the external interfaces making these changes. All of these changes need to be synchronized between IP-XACT design files and component files for each hierarchical level. This is a non-trivial refractoring operation that does not lend itself to simple repartitioning of a whole system in the way that could be easily implemented in the labeling approach mention above.

3) Implementation Experience

Nx-Builder [4] and Platform Express implement hierarchy as a collection of designs that can instantiate components and other designs. Being based on an Eclipse framework, related designs are grouped together into a container called a Project. During 2007, NXP and Mentor Graphics worked together to extend this concept to sub-system design. Projects were given the ability to reference designs existing in other projects, thus creating the ability for sub-systems to be developed as separate projects and then integrated in a controlled manner. This approach was adopted in preference to an import mechanism, which would allow designs to be brought together in a single project because it was felt to be preferable to keep each of the subsystems self-contained and simply refer to them from an integration project.

Project cross referencing has allowed NXP to design large systems consisting of multiple sub-systems but has highlighted consistency issues and areas for further development. The consistency issues exist where two projects that are to be combined have non-identical libraries of components available. The areas of further development are in the question of configurability of sub-designs; allowing designers to modify soft IP while restricting the ability to modify hard sub-systems, or at-least identifying that back-end files such as synthesis and layout files, can no longer be reused for sub-systems that have been modified.

IV. SUGGESTED EXTENSIONS

The process of configuring (configurable) IP is addressed differently by all IP providers. This topic is not addressed by IP-XACT or any other standard today and therefore a further opportunity for standardization exists.

For sub-systems there is also the topic of configurability. How should leaf IP parameters be exposed to designers to modify IP in soft subsystems, while restricting the ability to modify hard sub-systems, or at-least identifying that back-end files such as synthesis and layout files can no longer be reused for sub-systems that have been modified?

It will become important to understand the impact of a parameter value change. A parameter classification will help understand whether subsystems may retain configurability options after delivery.

V. CONCLUSION

IP-XACT has initially been used to document IP to introduce automation for integration of subsystems. We illustrate how IP-XACT has become an enabler for creating uniformity in the way of working for cooperating design teams. It facilitates and documents subsystem design data. An important aspect is the ability to integrate subsystems while retaining separation of source data which is needed for efficient intakes of subsequent subsystem releases.

Making best use of the standard, the EDA tool industry contributes tools to increase automation for the design data exchange and integration process. At NXP, teams at different sites develop subsystems with a specialized functional area. Subsystems are created using reusable IP for modem, compression, audio and display functions. Application is found in all TV, car infotainment and mobile communication SoC flagship products.

ACKNOWLEDGMENT

The authors would like to acknowledge the teams in Mentor Graphics and NXP Semiconductors who have contributed to this paper, to Platform Express and to Nx-Builder methodology and development.

REFERENCES

[1] Neal Wingen, "What If You Could Design Tomorrow's System Today?", DATE conference, March 2007: pages 835-840

[2] Christopher K. Lennard, "Industrially proving the SPIRIT consortium specifications for design chain integration", DATE conference, March 2006, Volume: 2, pages: 1- 6

[3] Geoff Mole , "Philips Semiconductors Next Generation Architectural IP ReUse Developments for SoC Integration", IP SoC conference, December 2004

[4] Denis Bussaglia, "Automated Implementation Flows based on IP-level constraints and synthesis intent in XML". IP SoC Conference, December 2005

[5] Wido Kruijtzer, "Industrial IP Integration Flows based on IP-XACT Standards", Proceedings DATE 2008