Transaction Level Modeling: Flows and Use Models

Adam Donlin
Xilinx Research Labs, 2100 Logic Drive, San Jose, CA 95124.
adam.donlin@xilinx.com

ABSTRACT
Transaction-level models (TLMs) address the problems of designing increasingly complex systems by raising the level of design abstraction above RTL. However, TLM terminology is presently a subject of contentious debate and a coherent set of TLM use-models have not been proposed. In this paper we propose a variety of TLM use-models that reveal paths through the TLM abstraction levels for various types of system. We begin by stating the abstraction levels that comprise 'transaction-level' and identify roles and responsibilities that apply within the use-models. We then take each use-model and discuss the type of system it applies to, the TLM abstraction levels it supports, and the design activites applied at those levels. We also consider the distribution of modeling effort between the various design rôles and apply that to descriptions of various use-model design flows.

Categories and Subject Descriptors
I.6.5 [**Model Development**]: Modeling methodologies

General Terms
Design, Standardization, Verification.

Keywords
TLM, design abstractions, use models, design flows.

1. INTRODUCTION
It is widely believed that RTL-centric design flows simply will not scale adequately to support the design of increasingly complex systems. The design and verification of complex systems is already a particularly perilous undertaking. Those who dare must withstand great pressures from escalating design costs and ever shortening design times. Along the way, they cannot stray far in their efforts without incurring expensive re-spins or missing lucrative market opportunities. Consequently, we are seeing a growth of interest in design at levels of abstraction that are higher than RTL. Transaction-level modeling (TLM) is one such approach. Our contribution in this paper is identifying a series of TLM use-models that describe paths through the TLM abstraction levels for various types of system. The discussion of each use-model addresses the TLM abstraction levels relevant to the use-model and the design activities they support.

2. LEVELS OF ABSTRACTION
Higher level of abstraction is a relative term and before proceeding, it is worthwhile discussing exactly what levels of abstraction constitute the TLM space. Contrary to its name, 'transaction-level' does not denote any single level of detail. Rather, it refers to a continuum of abstraction levels, each varying in the degree of functional or temporal detail they express. Points within this space may be singled out and identified as those abstraction levels that comprise the set of 'transaction level' abstractions. At the time this paper was written, exactly which points in the space should be singled out was a subject of some debate and there is no universally accepted terminology in use. Some alternative terminologies for TLM abstraction levels may be found in [3], [4] and [9]. The terminology used in this paper is largely based on that described in the forthcoming Open SystemC Initiative (OSCI) TLM Standard [8]. For the purposes of this paper, we shall distinguish the following levels of abstraction:

Algorithmic (ALG). This level comprises a behavioral model of the system's functionality. Models expressed at this level are unconcerned with any details of architecture or implementation.

Communicating Processes (CP). In this level, the behavior of the system is partitioned into parallel processes that exchange complex, high-level data structures. Systems modeled at the CP level are still mainly architecture and implementation independent although the act of separating out the functionality into distinct, parallel tasks leans toward architectural concerns. Communication between processes is conducted point-to-point hence there is no arbitration of the data communicated.

Communicating Processes with Time (CP+T). This level is functionally identical to the CP level but the model has been annotated with timing information. The timing models of nodes in a CP+T model may be either quite accurate (in library-oriented use-models, the timing properties of the node IPs can be analyzed in advance) or high-level, multi-cycle estimates. At this abstraction level, the exact communication protocol has not been decided and interconnect timing models are highly abstract.

Programmer's View (PV). The PV level is much more architecture specific than the CP level. This is especially true for communication: bus models or network-on-chip models are instantiated to act as transport mechanisms between the model components and some arbitration of the communication infrastructure is applied. Additionally, the PV level is register accurate to the point that low-level software drivers will see an accurate programmer's representation of the hardware sub-system.

Programmer's View with Time (PV+T). As with the CP+T level, a PV+T model is functionally identical to the PV level model but annotated with timing information. The timing model is of greater accuracy than that available in the CP+T model. Nodes of the CP levels are now fully partitioned into library IP and user defined functionality. Most importantly, the communication structure is now resolved to a given interconnection type. Hence, communication timing models may account the multi-cycle cost of a subset of the transactions possible over the interconnect structure.

CODES+ISSS'04, September 8–10, 2004, Stockholm, Sweden.

Cycle Accurate (CA). CA models capture micro-architectural details and typically have bit-level interfaces. Fully protocol compliant arbitration of the communication infrastructure is modeled. The model is clocked and all timing annotations are accurate to the level of individual clock cycles.

Register Transfer Level (RTL). As used in traditional design, RTL models are implementation and architecture accurate. Intra-clock cycle timing behavior is explicitly simulated.

The ALG and RTL levels are included above only to provide context to the levels between and we do not consider them part of the TLM space. Figure 1 shows the abstraction levels listed above and introduces the potential flows. Within these flows, the five main design activities we wish to support with TLMs are:

- Software Development;
- Executable Specifications;
- Design Space Exploration;
- Performance Analysis;
- and Functional Verification.

3. ROLES AND RESPONSIBILITIES

Before we tackle the use models in detail, it is also helpful to identify the different individuals and roles operating within the use models:

The Application Designer/System Designer has the role of customer. They understand the application domain and, depending on the use model, have varying degrees of appreciation for the low-level details of the implementation platform.

The Implementation/Process Vendor offers a variety of technologies to reduce a system specification to operational hardware. The exact facilities provided by the Implementation/Process vendor will vary with the nature of the products being created. For example, some vendors will offer pre-architected implementation substrates whilst others will offer increasing degrees of ASIC fabrication.

Third Party IP Vendors interact with a subset of the use models and provide pre-verified functionality that the Application/System designer may wish to employ within their products.

4. A PRODUCT DRIVEN VIEW OF TLM USE-MODELS

As we can see in Figure 1, there are a number of possible flows through the TLM abstraction levels. To place some structure on the use model discussion, we will take a "product-driven" approach. We will navigate the space of potential TLM use-models and design flows according to the features and requirements of particular product-types. For example, we want to understand how the TLM use-model supporting a Structured ASIC system would differ from the use-model for a "full custom" system design. In the sections that follow, we discuss the use model for a particular product-type, differentiating it from the preceding use-models, and exploring which aspects of TLM enable high-level design of that particular type of system.

4.1 UM1: ASSP and software dominated design.

System level design for application-specific standard products is the basis of the first TLM use-model. Recent Gartner analysis [1] indicates a strong trend towards the use of Application-Specific Standard Products (ASSPs) as replacements for waning ASIC design starts. As such, UM1 represents an important TLM use model. We consider ASSPs to be primarily microprocessor-bus oriented and, for this class of system, the final hardware architecture is entirely pre-ordained by the ASSP vendor. Texas Instruments' OMAP[10] is a good example of this type of system. The task of the application designer, therefore, is to implement their application on a given ASSP platform, making best use of that platform's available resources. Since the hardware component of the system is fixed, the application designer implements their application in the software destined to execute on the ASSP. This is the primary means of designing customer specific behavior into the product. It is worth noting, however, that a secondary level of design involves the tuning of the set of soft-parameters associated with the ASSP. Examples of such soft parameters include interrupt and arbitration priorities. Further parameters associated with more detailed aspects of the behavior of individual system IPs may also be available.

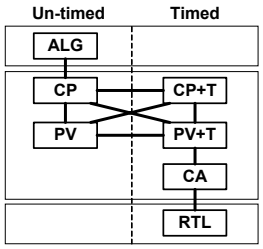

Figure 1: TLM Abstraction Levels and Potential Flows

An important question to consider, and one that we shall address for each of the use models discussed in this paper is: *"who shall write the models?"*. In this use-model, the burden of producing and supporting ASSP TLMs lies squarely with the ASSP vendor. From our description of UM1-type systems, we will now consider the design activities supported by UM1-TLMs. Firstly, the strong bias towards software in this model means any UM1-TLM must support full system simulation at sufficient performance and detail to enable software development. In the same way that the application designer selects a pre-ordained hardware platform to maximize hardware reuse, it is likely that a significant portion of the software subsystem will also come from firmware and 'off-the-shelf' sources. An interesting implication on the appropriate modeling level for software development is this: the firmware libraries will anticipate a register (and memory-map) accurate model with which to interact. This gives us a strong motivation for deploying PV TLMs (which are inherently register accurate) for software development.

Functional verification of the system's software and system performance estimation are the two main design activities in this use-model. Design exploration exists as a minor activity to guide the tuning of the ASSP's parameter set. Functional verification (and software development) are supported by PV level TLMs whereas performance analysis is done with a PV+T ASSP TLM. CA TLMs are of use at the later stages of performance analysis and functional verification of hard realtime software systems. The performance penalties associated with a CA TLM relative to the PV level models confines its use to this final stage of analysis.

An interesting point of conflict in this use model is that it involves a system where a hardware implementation, most likely, already exists. It is conceivable that the application designer choose to work with an actual hardware prototype of the ASSP instead of a model of that hardware. The primary advantage the ASSP TLM has over the ASSP prototype system would be its transparency. Robust TLM models at work in the UM1 flow execute more slowly than the hardware implementation but they can provide

complete observability into the system. This would be an important benefit when analyzing complicated failure scenarios.

We summarize the activities applied at each level for UM1 as:

CP and **CP+T** are not applied in this model but if they had been, their primary use may have been the capture of the system specification. The software dominated nature of the model and acceptance of a pre-ordained hardware subsystem implies a system specification may be captured in existing software flows (the UML[2], for example).

PV is used to enable software development and functional verification of the application design.

PV+T is used to support performance analysis of the application system. This enables identification of communication bottlenecks and some hardware platform parameter tuning. Soft-realtime functional verification is enabled at this level although the level of timing detail and incomplete communications arbitration prevent absolute verification and hard-realtime verification.

CA is applied in the final stages of design flow to enable detailed performance analysis and optimization of the application through detailed platform parameter tuning. Hard-real time software verification is also enabled at this level

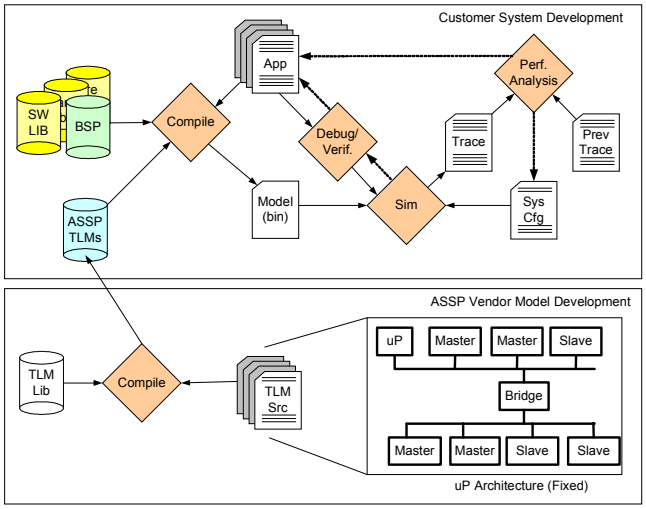

Figure 2: TLM Flow for UM1 - ASSP style, software-dominated systems

4.2 UM2: Structured ASIC - Library-oriented micro-architecture design.

The second use model we shall consider is somewhat characteristic of the domain of Structured ASICs [7]. In this scenario the designer is interested in building a customized micro-architecture to support their particular application. A substantial degree of design reuse is implicit in this use model. In UM1, the entire hardware subsystem was fixed and reused over different designs. In UM2, the designer applies an incremental refinement to an existing reference architecture specification of a hardware subsystem. We constrain the degree of customization applied to the subsystem specification to:

- substituting one IP module for another;
- adding a small number of new pieces of IP;
- or altering the provision of a subset of the existing IPs to support the application requirements (memory, for example).

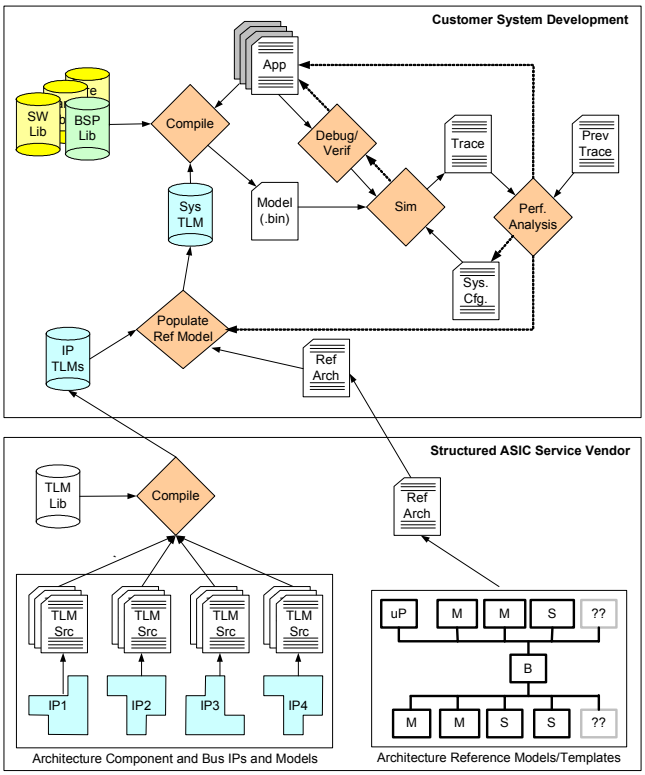

Figure 3: TLM flow for UM2-style Structured ASICs

The IP available to the application designer typically originates at the Structured ASIC vendor. In a slightly diluted form of the use model, however, third party vendors offering pre-designed, pre-verified IP modules that are interoperable with the reference architecture of the Structured ASIC vendor may also be considered. (Such IP would also be supported at the process level by the Structured ASIC vendor). Beyond the hardware subsystem customizations outlined above, the application designer still requires software to express the majority of their application's behavior. As in UM1, higher level system models are key in enabling rapid software development and functional verification.

The application designer is not responsible for creating the component models of the system and is typically not the source of the reference hardware architecture. The Structured ASIC vendor, along with any third-party IP vendors create the component IP models at the each level of abstraction required. The Structured ASIC vendor is also the likely source of the hardware subsystem's reference model. Together, the component models and reference architecture model are injected into the system design flow as indicated in Figure 3. Unlike UM1, the reference system model is not treated as the final application architecture. The designer may choose to begin software development on the reference architecture to provide a first analysis of the bottlenecks in their system. From here the designer is in a stronger position to select replacement IP, introduce new IP or alter the provision existing IP. We anticipate this process will be applied iteratively. When iterating, the hardware or software subsystems may be altered or the IP parameters may be changed. The process stops when the TLM demonstrates, through performance analysis, a solution with appropriate performance. The successful refinement of the reference hardware model to this point closes out the main TLM flow. The TLM itself would be passed to the Structured ASIC

vendor as a clear specification of the hardware architecture to be fabricated for the customer.

With respect to each of the TLM abstraction levels the following activities would be applied:

CP: introduced as a model capturing the application behavior in a functionally accurate but essentially architecture agnostic manner.

CP+T: not used.

PV: enables software development and functional verification of the application design.

PV+T: supports performance analysis of the application system. This enables identification of communication bottlenecks and some hardware platform parameter tuning. As for UM1, PV+T also supports the functional verification of the soft-realtime components of the UM2 system.

CA is again used in the latter stages of design for detailed performance analysis and verification of hard-real time components.

4.3 UM3: Structured ASIC - Semi-custom Micro-architecture Design.

The third use-model we shall discuss fits more closely with the contemporary interpretation of the Structured ASIC domain and introduces yet more options for the system designer. The key difference between UM2 and UM3 is that UM3 introduces the first instance of pre-meditated, custom hardware design. Whereas the UM2 system designer's appetite for customization could be satisfied with IP from Structured ASIC vendor's IP catalogue, in UM3 the system designer requires more extreme measures to meet their design's performance goals, functional goals or both. To meet these needs, the designer again begins with a reference hardware architecture to which they add their own, custom-designed IP to the system's interconnection architecture (typically a bus). As we are still working within the Structured ASIC domain, the dimensions and provisions for such custom hardware will be restricted to a given geometry and limited in magnitude. A definite minority of the Structured ASIC die area will be available for customization. Hybrids of UM2 and UM3 are conceivable where the customizable regions are populated partially with library IPs and partailly with custom logic. The increasing degree of customization available in UM3 reflects a much greater investment in the design of the application's hardware sub-system and, as such, design space exploration plays a much stronger role. The system designer is taking a greater risk by fabricating a line of application-specific devices to support their application and it is important that the tool-flow guides them towards making the most appropriate partitioning of application behavior. UM3 is a particularly significant model for designers wishing to differentiate themselves from existing products or attempting to introduce new functionality that cannot be supported by existing IP catalogues effectively.

Clearly, the introduction of a custom IP path into the system designer's flow will increase their modeling responsibilities. For UM3, we assert the division of responsibility as follows:

The Structured ASIC vendor retains their responsibility for creating models of their reference hardware architecture and any IPs that may be selected to populate the customizable regions of that architecture. Third party vendors have comparable responsibilities to establish their IPs within the design flow.

The system designer is responsible for populating the reference model. Crucially, for UM3 this includes the modeling of custom IP components that are compliant to the TLM-API, the protocols of the interconnection architecture, and any other constraints placed on the custom region by the process vendor. The system designer creates such custom IP models for each relevant level of abstraction.

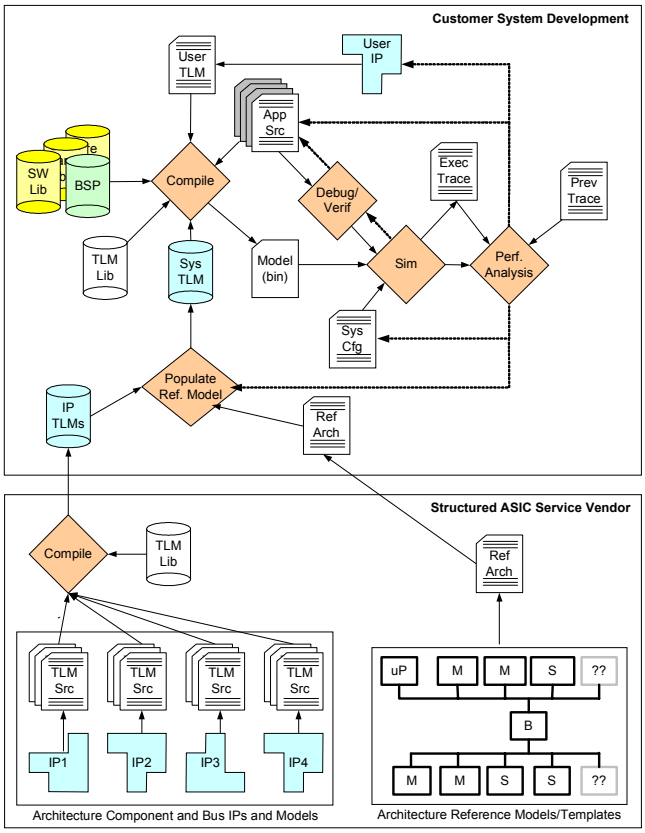

Figure 4: TLM Flow for UM3-style Structured ASICs

Where we stated UM1 and UM2 were "software-dominated", we consider UM3 "software-heavy". Certainly, the system designer now has an opportunity to partition part of their application directly into the hardware domain. However, the constraints inherent in the Structured ASIC domain mean that a substantial component of the system (and its control flow) remain in the software/microprocessor domain. For this reason, the design abstractions supporting software development and functional verification are equally important to UM3 as they were for UM1 and UM2. The essential difference for UM3, however, is that design space exploration and performance analysis are both much more prevalent.

For each abstraction level, the related UM3 design activities are:

CP: An executable specification of the system behavior is captured at this level.

CP+T: High level performance estimation is enabled at this level by timing information annotated into the CP model. The main aim is to support design space exploration and partitioning of application functionality into distinct process nodes.

PV: early development of software and functional verification.

PV+T: for performance analysis of the system, particularly to identify communication bottlenecks in the microprocessor oriented subsystem and tune platform parameters. Functional verification of soft-realtime properties is applied.

CA: for detailed performance analysis and optimization. Holistic verification for both software and hardware subsystems.

4.4 UM4: Hybrid Microprocessor-Communicating Process Design.

The fourth TLM use model moves us out of the domain of the structured ASIC and towards custom designed ASICs. For this scenario, the performance and functionality targets of the application being designed require a more aggressive use of customized hardware. Nonetheless, UM4 has a strong correlation to UM3, differing firstly in the degree of application functionality partitioned into the hardware domain. Relative to UM3, UM4 partitions much more of the application functionality into the hardware domain with the aim of realizing it as *custom* logic. To place a sense of scale of hardware customization we consider the percentage of functionality partitioned into the hardware domain to be on the order of 50% for UM4 versus 10% for UM3. Clearly, such percentages are descriptive and intended to provide more a first-order differentiatiation of the two models than a strict bound. Furthermore, the term custom hardware is used in this context to imply not only a customization of the functionality, but also a more radical customization of system interconnect. Rather than interconnecting optimized blocks over a shared transmission medium (à la UM3), UM4 custom function blocks are interconnected in an ad-hoc manner and often point-to-point. Such interconnect would certainly be more specific to the application and therefore more costly to design and realize. It is an attribute of UM4 that such costs fall within an acceptable tradeoff of design effort and necessary application performance.

An equally significant aspect of UM4 concerns the distribution of control within the system. For UM3, even although some system functionality has been partitioned into hardware, control-flow within the system was still essentially rooted in the microprocessor/software subsystem. In UM4, the partitioning of control flow between the software domain and the hardware domain has increased. To exemplify this, the custom IP created for a UM3 design remains a slave to the control flow of the software system. In UM4, however, the custom logic and the software system share the relationship of equal peers. Each may interact freely and operate independently, only synchronizing when necessary. In terms of impact on the design process, UM4 TLMs must support verification of complex parallelized control between both the hardware and software subsystems.

The high level of application specific customization evident in UM4 pushes a majority of the modeling responsibility onto the shoulders of the application designer. In the same way the systems themselves are increasingly customized, so too are the models. Vendor supplied modeling IP will still have a role to play, particularly as UM4 leverages microprocessor architectures: processor and bus models are unlikely to be hand crafted by the application designer. However, the concept of an initial template or reference architecture as exploited in the previous use models is less relevant. The system designer is now responsible for creating a custom microprocessor-architecture model from the vendor supplied component models or deep customization of any vendor supplied template model.

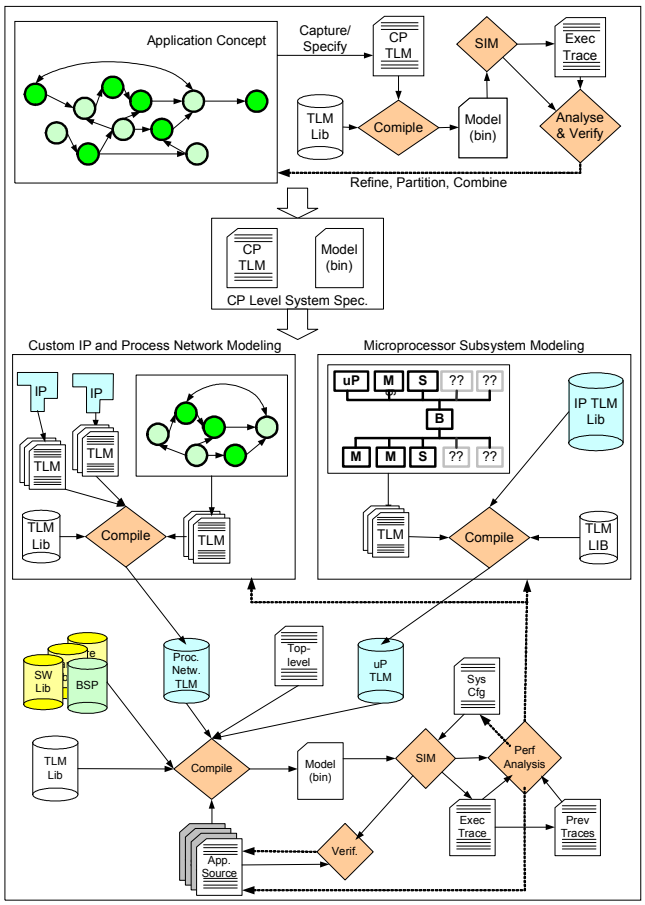

Figure 5: TLM Flow UM4 for Hybrid Microprocessor-Process Network Systems

The set of design activities associated with each abstraction level in UM4 is equivalent to that applied in UM3 but the two models differ in their emphasis on the CP and CP+T levels. UM3 introduced the CP levels as active participants in the design process (above and beyond the use of CP for executable specifications in UM2). UM4 extends this in proportion to the system designer's desire to increase the amount of functionality assigned to the hardware domain. Since part of our system is still expressed in the software domain, PV and PV+T levels retain their relevance for early software development, integration and verification. Similarly, CA offers us the ability to pursue detailed performance analysis and functional verification.

4.5 UM5: Communicating Process Architecture Design.

Hardware-dominated, highly customized designs characterize UM5 systems. In this scenario, all aspects of the system are available for customization as a means of achieving maximum performance and functionality. Systems designed in this model are realized as networks of communicating processes. The implementation of processes is also more logic-centric (as opposed to software-centric). The job of the designer is now to architect a customized network where functionality is partitioned amongst the nodes appropriately. Furthermore, they must provision the point-to-point communication channels between the nodes such that the application's communication requirements will be met. This involves selecting communication channels of

the appropriate width and, within those channels, selecting FIFOs of an appropriate depth. Design space exploration based on performance analysis guides the iterative re-partitioning of functionality allocated to network nodes and provisioning of channel resources. This task-set applies irrespective of the final implementation technology (full-custom, gate array, or FPGA). Functional verification also remains a central activity allowing the system designer to maintain correctness whilst the system is partitioned and moves through abstraction levels.

Control in UM5 is highly distributed: each process node encapsulates its individual control-flow. If necessary, the processes synchronize their control flows through their communications with other nodes. The exact semantics for such synchronization must be selected by the system designer in accordance with their understanding of the target application. That said, the TLM foundation classes used to model these systems would necessarily include a variety of common semantics (eg, Kahn [5], SR [6]).

For the most part, the distribution of modeling responsibility in UM5 lies squarely with the system designer. Whereas UM1-3 were able to offload substantial amounts of the modeling effort to the implementation technology vendor, the availability of large-scale custom hardware design pushes us to the opposite extreme in UM5. Potentially, each process node in the application may be a custom logic design and the application designer is responsible for modeling, potentially, all of the behavior in each of the system's processes. Using pre-architected IP is still conceivable in a diluted form of UM5. In that circumstance, an IP vendor would retain responsibility for crafting models of any such IPs and the application designer concerns themselves only with provisioning and interfacing the correct point-to-point interconnection resources to meet the system's performance goals.

The refinement flow through the TLM abstractions for UM5 is heavily focused on the CP and CP+T levels. For UM5, the CP levels can capture the native architecture of the application. Although it is not strictly part of the TLM space, we introduce the ALG level as an architecture-independent abstraction level for capturing a behavioral specification of the system at a higher level of abstraction. We also differentiate UM4 and UM5, and motivate the migration of emphasis from the PV to CP levels, based on the lesser role of software in UM5. If software subsystems do exist, they are hidden behind the process metaphor and are evident only during a hierarchical refinement of the system.

An interesting issue in this use-model concerns the role of the PV levels in what are, essentially, highly logic-centric systems. Here, there is no software to speak of and there is no strong notion of a 'programmer' to motivate the 'programmer's view'. We treat this as an idiosyncrasy of the 'microprocessor-bus oriented' terminology prevalent in the TLM community. Nonetheless, a refinement of the functionality captured in CP models to a comparable level of *functional detail* expressed in PV models is indeed reasonable. For example, UM5 'PV' models express word-level exchange and manipulation of data between and within processes as opposed to the exchange and manipulation of complex data-structures at the CP level.

5. SUMMARY

This paper has presented a series of use models and flows for the exploitation of transaction level modeling in the development of complex system designs. The use models we described were differentiated according to the particular type of system-product being designed (for example, an ASSP-based system versus structured ASIC versus full custom design). We discussed the use-models in a particular order to accentuate an incremental relaxation of architectural constraints and properties. For example, UM1 represented a heavily constrained system wherein the hardware subsystem is pre-ordained and the application designer's effort is spent, mainly, in the software domain. Progressing through subsequent use models, we saw increasing degrees of optimization and opportunity for customization into the hardware domain: the system designer is given successively larger opportunities to alter the architecture and resource provisioning of their system. This was accompanied by a decreasing emphasis on the software domain as we observe in the transition from a "software-dominated" UM1 towards a "hardware-dominated" UM5. Buses also eventually yield to the ad-hoc interconnection structures of UM4 and UM5's process networks. In addition, we also observe the incremental re-distribution of control within the systems supported by each use model. For example, in UM1 the system design exhibits highly centralized control that is expressed in the software domain. Increasing opportunities to distribute and customize functionality into the hardware domain successively dilute this scenario until we achieve UM5's model of highly distributed control whose influence is localized to a particular node in the system's process network.

6. References

[1] Hsu, *et al*, "Worldwide ASIC/ASSP, FPGA/PLD and SLI/SOC Application Forecast, 3Q03", Gartner, Oct 2003.

[2] Booch, *et al*, "The Unified Modeling Language User Guide", Addison-Wesley, Sep 1998.

[3] Cai *et al*, "Transaction Level Modeling: An Overview", in Proc. 1st IEEE Intl. Conf. on Hardware/Software Codesign & System Synthesis, pp. 19-24, October 2003.

[4] Grotker, *et al*, "System Design with SystemC", Kluwer Academic, 2002.

[5] Kahn, "The semantics of a simple language for parallel programming", Proc. IFIP Congress' 1974. pp. 471-475.

[6] Halbwachs, "Synchronous programming of reactive systems", Kluwer Academic, 1993.

[7] Worchel, "Structured ASICs: The Preferred Design Solution for the Future", InStat Report, Aug 2003.

[8] Burton, Donlin, "Transaction Level Modeling: Above RTL design and methodology". Open SystemC Initiative TLM-Working Group. http://www.systemc.org/. (*in preparation*).

[9] Ghenassia, *et al*, "Using Transactional Level Models in a SoC Design Flow" in "SystemC: Methodologies and Applications", Kluwer Academic, 2003.

[10] P. Cumming "The TI OMAP Approach to SoC", in "Winning the SoC Revolution", Kluwer Academic, 2003.