System Level Design for System on a Chip

W. Shields Neely
Systems Methodology Manager
Architecture Laboratory
National Semiconductor
Santa Clara, CA 95052
w.shields.neely@nsc.com

Vincent John Mooney III*,[1]
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250
mooney@ece.gatech.edu
* National Semiconductor Fellowship recipient.

Abstract

We present the systems requirements generation and executable specification capture of a single chip LAN Adapter for communicating using the IEEE 1394 Serial Bus Protocol. The requirements generation starts with high level performance simulation and then passes to an executable specification suitable for implementation using a hardware/software co-design tool. The reuse of pre-existing components is supported, as well as synthesis of the system interface, but only after much work is done to program the hardware/software co-design tool. The actual design flow described allows feedback among all design levels, e.g. from implementation up to requirements, throughout the process.

1 Introduction

Hardware/software co-design of embedded systems [1, 2, 3, 4] can help address some of the most important aspects of System on a Chip (SoC) design. For example, the Ptolemy[5] tool and the CoWare[6, 7] tool have addressed critical issues. Ptolemy is an environment and simulator for highly heterogeneous systems and CoWare is a design system for embedded telecommunication applications. CoWare, for example, supports inputs in C, VHDL, and DFL[9] (a data-flow language), with VHDL and DFL mapped to hardware while the C code is compiled onto a DSP processor[6, 7]. This paper presents a case study of one attempt to realize a design using hardware/software co-design tools and methodology beyond the level of simulation.

The rest of the paper is organized as follows. Section 2 describes briefly our motivation. In Section 3 we describe the *workflow based design flow* used in industry. Section 4 describes the assumptions and limitations in a real design case of a LAN adapter. Section 5 explains the design methodology and tool flow used for the LAN Adapter. Finally, Section 6 gives some conclusions based on the real design and suggests future directions.

2 Motivation

The main message of this paper can be summarized in the following four succinct points.

(1) The world is messy. In real design, parts often don't fit together, and idealized CAD flows often don't work.

(2) Models are not available. Often models are available at a low level with much detail (e.g. transistor layout) or at a high level with little detail (e.g. C or behavioral Verilog). Rarely are they available at both.

(3) Global partitioning is not the main issue. Much of the global partitioning is already done for you – most real designs start with a lot of pre-existing components that for various reasons (backward compatibility, strategic partnerships, etc.) must be used. Partitioning can be carried out only on small portions of the design. Hardware often comes in the form of *Intellectual Property* (IP) which is thus already designed and readily available. Much more flexibility exists in software.

[1] The second author was a National Semiconductor Fellowship recipient at Stanford Univeristy when this paper was written.

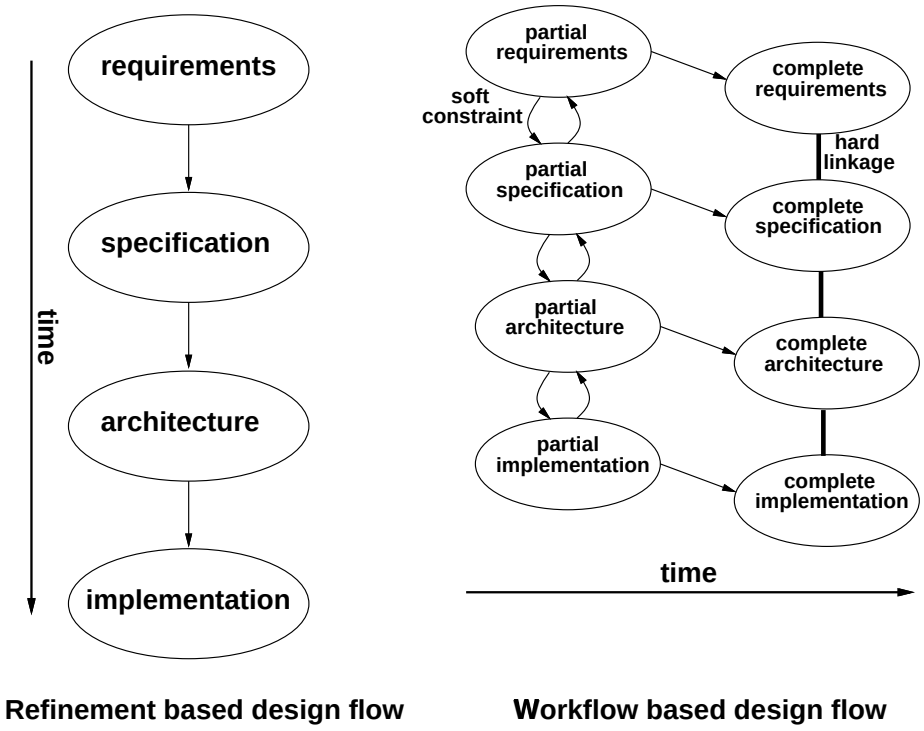

Figure 1: Comparison of Design Flows

(4) Leverage points are key. In real designs, system architects need to find leverage points where local partitioning can be applied to maximal advantage. For example, scheduling of hardware and software components is often a critical part of the design with a large impact on performance. Another clear example is in bandwidth management. Do we need one bus or multiple busses? Automated synthesis tools applied to interface design can speed up the design process significantly and allow tradeoffs to be made at higher levels of abstraction.

2.1 System on a Chip Design

Design methodology for SoC cannot be a linear extension of board design or ASIC design. To begin with, architectural issues in SoC are vastly different. We often want to reuse previous hardware and software but use the hardware or software in a very different environment – namely, on the same chip, which has greater interconnect available but is at the same time much less testable. In SoC we often want to explore different architectural options by moving functionality around among hardware and software. All of this leads to a greater amount of and much more complicated interactions. Finally, we find great heterogeneity in the abstract models themselves, since we have to support multiple domains interacting together, such as the following:

- synchronous dataflow
- control dominated reactive systems
- analog

Not only architecture but also validation of SoC is not the same as board design or ASIC design for several reasons. First of all, one cannot use prototyping as the major validation method because (i) the cost is too high if there is a design error and (ii) complex interactions between components (hardware or software) may not be visible just from the pins of the chip. Secondly, validation as is done in the case of ASICs will not work because of the following:

- sheer size
- CPU control ASIC as a slave too limited
- large number of, and greatly complicated, interactions
- **not** verifying each component in isolation, but the system's emergent behavior
- interaction of two large state spaces

Thirdly, we need to back away from detail and push up the level of abstraction. A big problem with higher levels of abstraction is the design time needed to write high level models of components for which you already have working detailed models.

The design methodology for SoC is not a linear extension of board design or ASIC design. We hope to drive these points home as we describe our design example. What is needed is a new design methodology for SoC.

3 Workflow Based Design Flow

Figure 1 shows the difference between the refinement based design flow and the workflow based design flow. The former is a CAD-centric, idealized view of the design activity, whereas the latter is the design flow actually used in industry.

A detailed process description of work is to be done and how that work is to be accomplished is referred to as a workflow [10]. Each individual step in a workflow is an activity. An activity may be accomplished by either a human or a program, or a human using an interactive program. The workflow paradigm is particularly useful for SoC design work, since early phases of a SoC design involve primarily human work (e.g. English language software and hardware specifications) while latter stages involve primarily programs (e.g. logic synthesis, program compilation, and layout). The move towards providing executable workflows for well defined tasks is underway [11].

In this paper, we describe a manual workflow in which a system level simulation written in BONeS can provide iterative refinement between the system level requirements behavioral specifications in CoWare.

With the workflow based design flow a system design team can handle the situation where various parts of the design process are done out of order. For example, consider design reuse. We may want to reuse blocks for which we have only a gate-level description. To simulate the system with such reuse we have to generate a high level description in order to achieve reasonable simulation speeds. This introduces an additional validation problem, that of validating the new high level description against the reused gate-level description.

The workflow based design flow also supports the same steps as the refinement based design flow, such as the generation of a high level model from the requirements or specification.

4 Design Case – LAN Adapter

For our design case we consider a Local Area Network (LAN) Adapter, based on the IEEE-1394 Serial Bus Protocol, to support high speed node applications. Figures 2 shows a sample context for the LAN Adapter. In the figure, the adapter has three connections on the 1394: a PC, which acts as the root for assigning address space on the 1394 connection, and two peripherals. The goal of the design is to develop a common system level approach to building nodes using National Semiconductor's well proven micro-controller technologies.

The objectives of the design are fourfold:

- reusable system level model
- reusable device driver approach
- reusable core definition for target modules
- hardware/software co-design of nodes

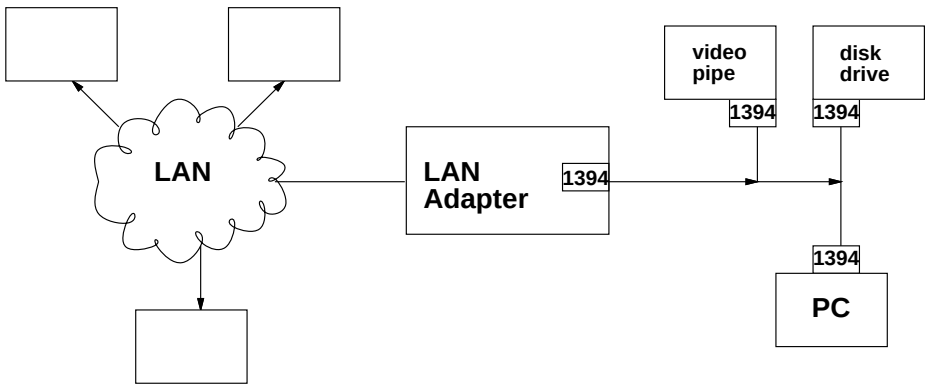

Figure 2: LAN Adapter example

To support the rapid design of nodes, common reusable elements are needed. A systems-based methodology which stresses reuse of PC side driver software, 1394 and Universal Synchronous Bus (USB) node cores, and efficient design of the hardware and embedded software in the node can provide a significant performance improvement and provide a broad range of applications that use the LAN Adapter in node designs. This pilot design was developed to help provide the basis of a reusable system-level nodes methodology. It integrates the use of abstract system level models, device driver approaches, hardware requirements for a node core, and hardware software co-design of the node core.

Before describing the node architecture, we describe briefly the network terminology.

Transport Layer Protocol	**3**
Link Layer Protocol	**2**
Media Access Layer	**1**
Physical Interface Layer	**0**

Figure 3: Network Layers

4.1 Network

Figure 3 shows the four network layers used. The bottom layer is the physical interface that specifies the voltage used, the clock frequency of the link, etc. The next layer is the Media ACcess (MAC) layer. It specifies, for example, two- or four-phase communication, split transaction bus, collision detection, and retransmission of packets. The next highest layer specifies the link layer protocol used, e.g. Ethernet. Finally, the top layer specifies the transport layer, e.g. TCP/IP.

4.2 Node Architecture

Figure 4 shows a more detailed description of the original design. In the bottom right we see the LAN physical interface layer implemented in the form of a hard macro (transistor netlist and layout). Just above the LAN physical interface is the LAN MAC layer specified in a Register Transfer Level (RTL) description using VHDL. The CPU connected to the LAN MAC layer implements the link layer protocol and the transport layer protocol (Ethernet and TCP/IP, respectively, in this case). The CPU communicates with the rest of the system via a circular buffer connected to the 1394 MAC, which at the beginning of the project had no specific design

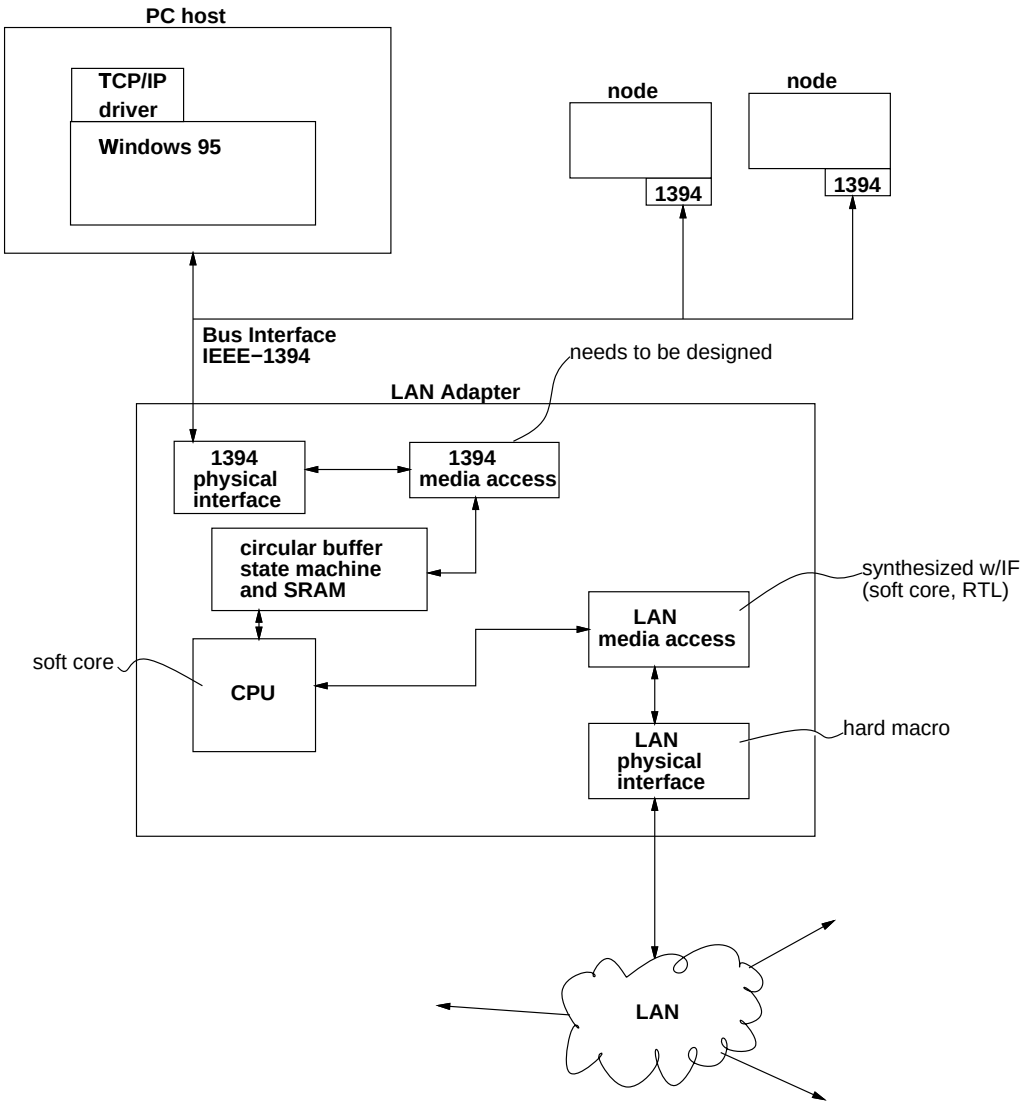

Figure 4: LAN Intelligent Network Adapter Card

implementation whatsoever associated with it (thus, it needed to be designed from scratch based in the 1394 specification). The 1394 MAC creates packets, generates the appropriate ACK packets, generates and checks the Cyclic Redundancy Check (CRC) information, and arbitrates use of the bus. Finally, in the top left we have a hard macro of the IEEE-1394 firewire physical interface layer. The main source of flexibility in the design lay in the 1394 MAC layer and the LAN MAC layer.

The following features are provided on the 1394 Serial Bus:

- supports two transmit contexts (for requests and responses)
- supports two receive contexts (for requests and responses)
- supports dedicated or shared access to an SRAM
- supports operation up to 800 Mb/s
- optimized for high-speed node operation

- connects to a variety of devices for plug-n-play

All packets are transmitted and received to a RAM that is accessible with guaranteed latency and bandwidth. Up to 64 KB of memory can be addressed.

4.2.1 Asynchronous Transmission

There are two categories of asynchronous packets transmitted on the 1394 – asynchronous requests and responses. To keep each use of a transmit packet separate, several transmit contexts are supported. Retransmission of requests must never block transmission of response packets.

Each transmit context supports transmission from the local memory upon command. The LAN adapter only keeps track of one packet per context at any given time.

4.3 Software Architecture

The software architecture is based on the Serial Bus Protocol 2 (SBP-2)[8]. SBP-2 is a transport protocol defined for the 1394-1995 Serial Bus. It defines facilities for requests originated by Initiator (which are the request originators) to be communicated to other Serial Bus devices (called Targets). It also defines a facility to transfer data and status information between these devices. The Target actions are specified by means of Requests that are created by the Initiator. A Request is contained within a data structure called an Operation Request Block (ORB). Upon completion of a request, the Target writes back the completion status to the Initiator at an address that was provided earlier by the Initiator. The types of request blocks required for the 1394 LAN profile are as follows:

- Login Request: used by the Initiator to obtain access to the Target resources
- Command Request: used by Initiator to transport commands to the Target
- Management Request: used to set or release target resources

An ORB can describe a data buffer directly (for a contiguous data buffer) or via a page table (for disjoined data buffers). A typical ORB has the structure shown in Figure 5.

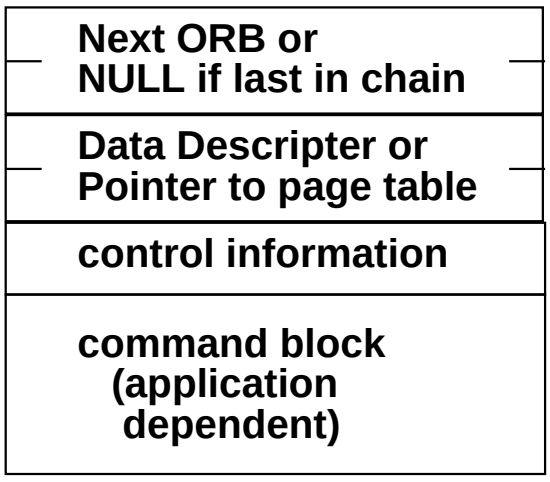

Figure 5: Structure of Operation Request Block (ORB)

In case of Transmit operations, the PC host will issue commands (via SBP-2 ORB command requests) to the LAN Adapter to transmit Ethernet packets. Each ORB describes one Ethernet packet. The ORBs can be chained together to describe multiple packets. The various configurations possible for Transmit are as follows:

- Fetch a single ORB with its corresponding packet. Transmit the packet and write the status back to the PC host.
- Pre-fetch all the ORBs in the chain. Fetch the data buffer when the ORB is being processed.
- Pre-fetch all the ORBs in the chain and also corresponding data packets. Process each ORB one at a time and write the status back to the PC host each time.

Hence, SBP-2 can be used flexibly to implement a data transfer scheme that best suits the price-performance requirements of the end product.

Based on the hardware resources available on the LAN Adapter, it can pre-fetch the ORB chain in order to speed up the data transfer. Also, the PC host can dynamically append a new ORB to the end of the current ORB chain, as defined by the SBP-2 specification.

4.3.1 Ethernet networking model in Windows

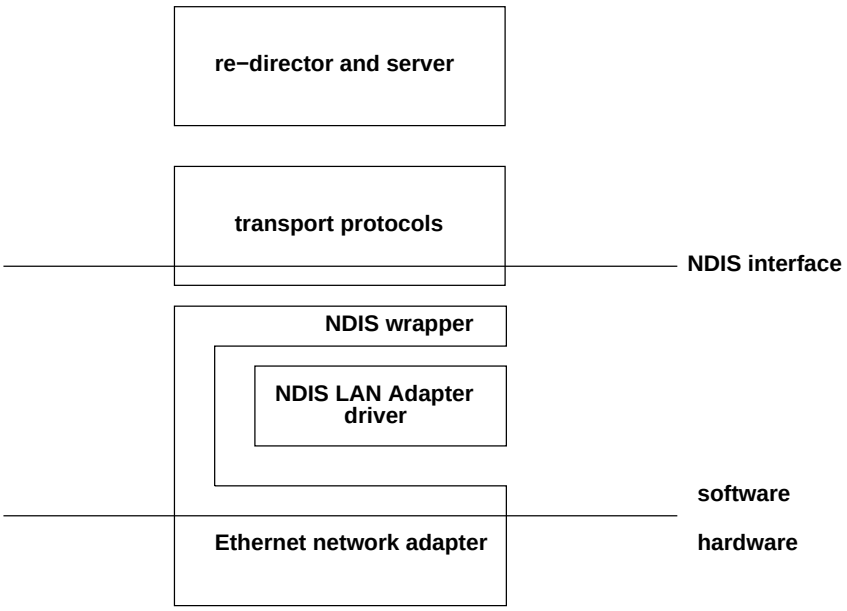

Figure 6: Software architecture of Windows Ethernet drivers

For the packets received by the LAN Adapter from the Ethernet end, the LAN Adapter software can upload the received packets in the PC host memory. The upload address is described by an ORB chain that was fetched by the LAN Adapter at login time. Based on the resources available on the PC, the PC host can decide to dedicate as many receive buffers (described by an ORB chain) as it can afford.

The Network Device Interface Specification (NDIS) also supports what is called an **Intermediate Driver** architecture. This allows the driver to manage an adapter with a new media type (e.g. 1394) without the need to change the upper layer architecture. The software on the PC host will then have an Intermediate Driver which interfaces with the NDIS layers and the 1394 software stacks.

5 Design Methodology and Tool Flow

The design methodology used for the LAN Adapter consisted of the following steps:

(1) High level performance estimation

(2) Detailed functional level performance estimation

(3) Paper specification of hardware and software design

(4) Normal RTL design methodology

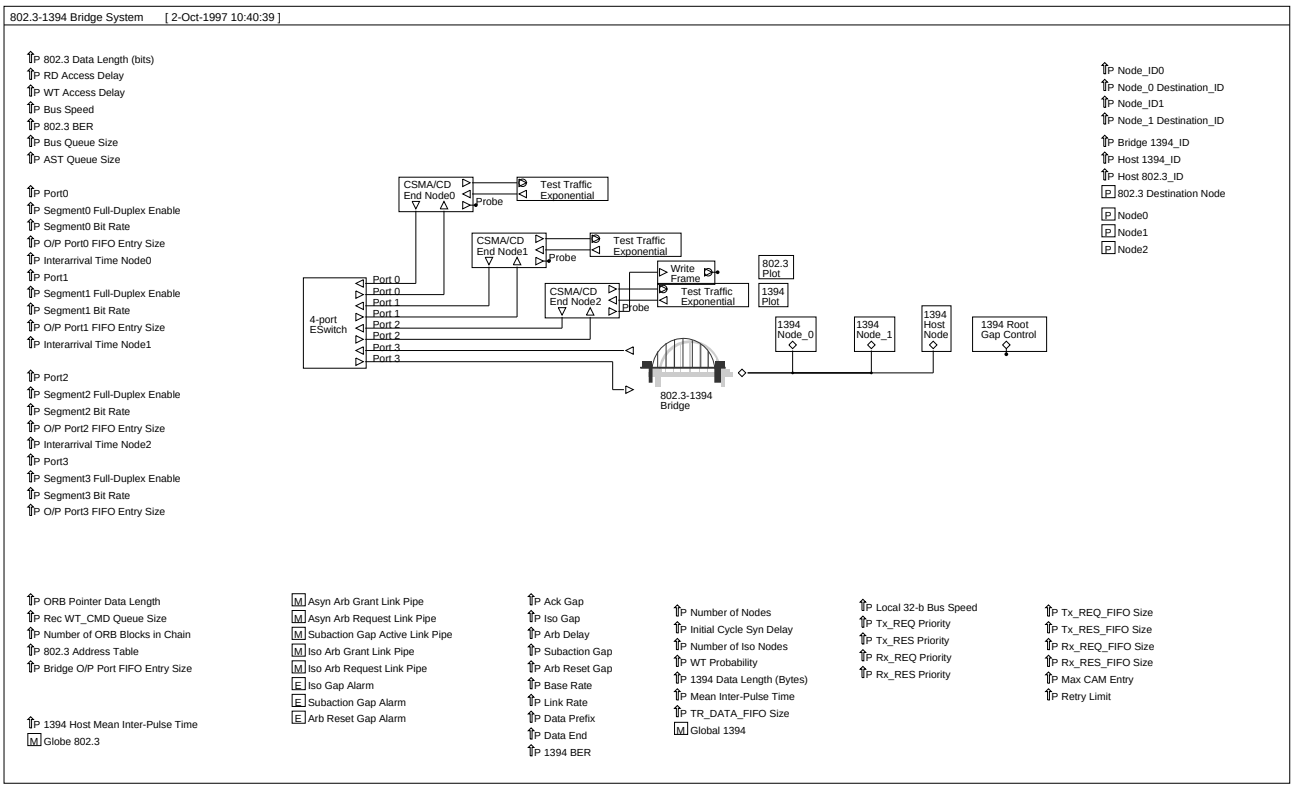

Figure 7: BONeS System Level Model

5.1 System Simulation and Architectural Exploration

Steps **(1)** and **(2)** were performed with BONeS Designer^{TM} from the Alta Business Unit of Cadence. BONeS models and simulates event-driven data transfer systems at the protocol/messaging layers, allowing system performance modeling and analysis of throughput, latency and resource sizing.

BONeS models were written for each of the items in Figure 4 – including the LAN traffic, PC, and peripherals. The LAN traffic is estimated for a four port Ethernet switch. The resulting data traffic on the 1394 bus is used to estimate the traffic load for a typical application.

The approach to generating a system can be characterized as a library based approach. However, in order to perform high level performance estimation, it was often necessary to generate and validate lightweight models for pre-existing low level models. Furthermore, we were unable to enforce a total top-down methodology (refinement based design flow), but instead had to use a workflow based design flow.

The steps used provide a flow from performance analysis to implementation.

5.1.1 The system level model

The BONeS system level model, shown in Figure 7, assumes a 1394 host node, two other nodes on the bus, and the LAN Adapter communicating to a four port Ethernet switch. The data traffic on the 1394 bus is

used to estimate the traffic load for a typical application. The critical system parameters are passed to the top level of the system simulation. By selecting the parameters, such as the bus speed to be supported (400 Mbit/sec, 800 Mbit/sec, and so forth) and running the simulation, internal measurements of the system can be used to determine the buffer size of the buffer SRAM and the operation speed of the buffer state machine. By running simulations of the configuration process (address assignment), the time required for various processors to perform configuration can be determined. The processor overhead to perform SBP-2 encapsulation can be estimated by counting the encapsulated packet rate.

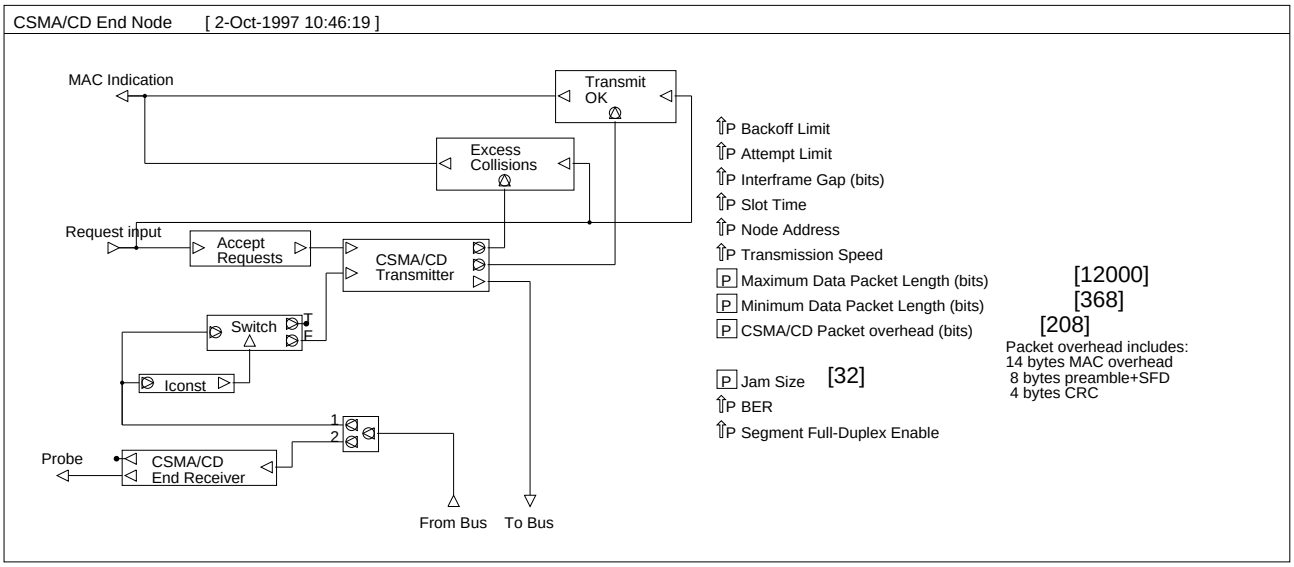

Figure 8: LAN node

5.1.2 LAN node

A generic LAN node is used to estimate the effects of the LAN components of the complete system. Each LAN node has parameters to describe packet length and number of overhead bits.

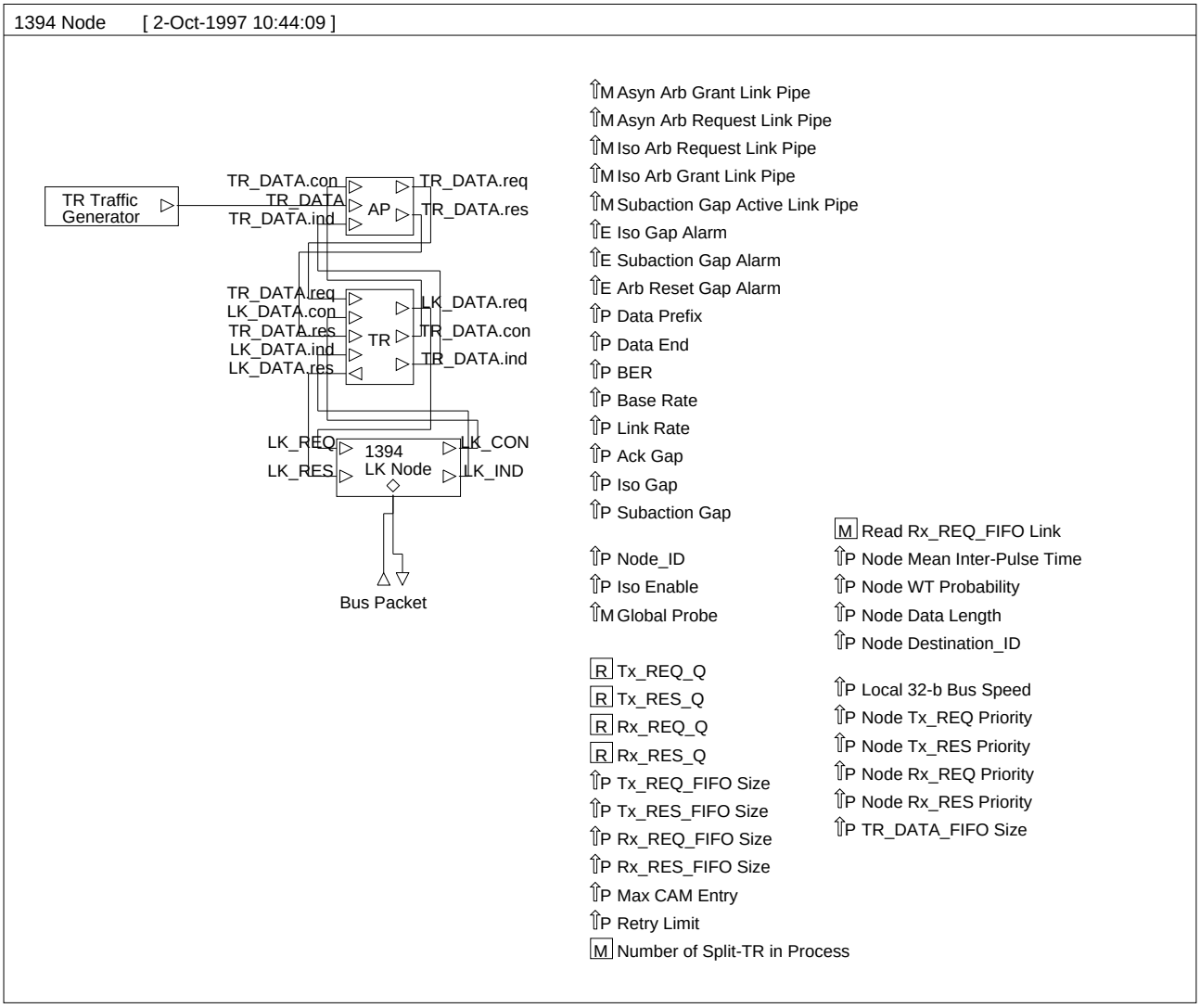

Figure 9: 1394 Node

5.1.3 1394 Node

Figure 9 shows the 1394 traffic generator node used to accept packets from the 1394 bus and to generate, according to a desired traffic profile, a typical level of bus congestion. For each 1394 Node, there are Link (LK), Transaction (TR), and Application (AP) Modules, with interfaces corresponding to the four "data primitives" – Request, Indication, Response and Confirm.

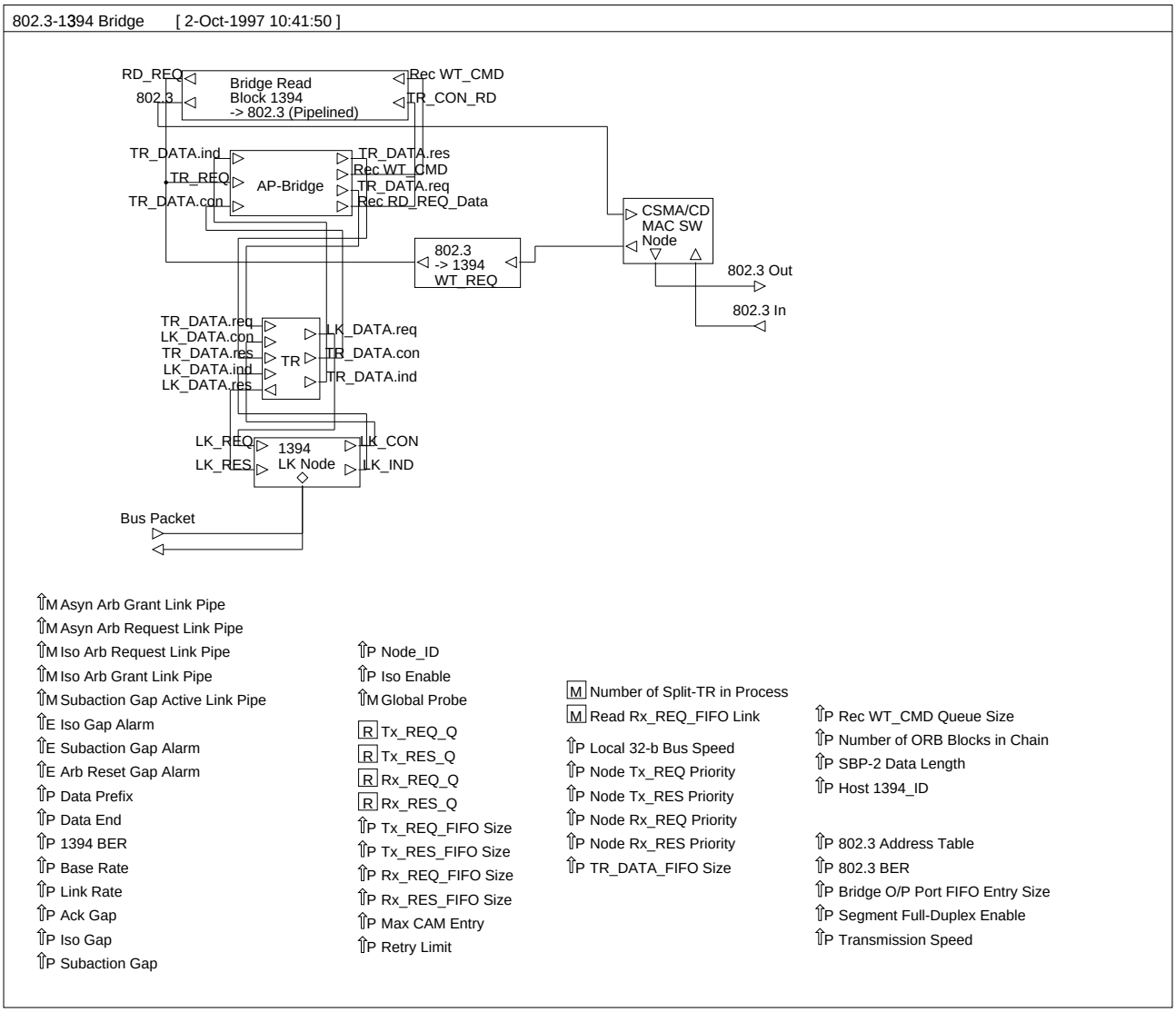

Figure 10: LAN 1394 bridge

5.1.4 The LAN 1394 bridge

Figure 10 shows the LAN 1394 bridge, which describes the functional operation of the LAN Adapter hardware and software, The structure of the LAN-Bridge model is determined only by the desired behavior of the LAN Adapter and is not limited or constrained by the adapter architecture. Different forms, such as CPU based or hardware-based architectures, will need to perform the same functions. Parameters can be set that will allow assessment of the effect of using different hardware architectures.

The BONeS simulation must be finished before beginning the description of the LAN Adapter in C using CoWare.

5.2 Hardware-Software Co-Design with CoWare

The node product planners will need to analyze which part of the overall system is best executed on the software programmable CPU core and which functions are best implemented in dedicated on-chip hardware. These architectural decisions will have a dramatic impact on the amount of on-chip memory required, the overall die size, system performance, and power consumption. Delaying the system partitioning to late in the design process allows the architectural decisions to be made with the greatest level of information possible.

The CoWare hardware/software co-design environment allows the cospecification of hardware and software components using existing languages such as VHDL, DFL[9], Silage and C [6, 7]. CoWare provides unambiguous specification of interfaces between hardware and software, and correct synthesis of these interfaces in hardware and software by generating both hardware interfaces and device drivers.

CoWare is based on a data model of communicating processes and supports the gradual refinement of a high level description into an interconnection of programmable processors and dedicated, synthesizable hardware. The model supports the re-use and encapsulation of hardware and software by a clear separation between the functional behavior and the communication behavior of a system component.

The basis for this specification method is a data model that is based on communicating processes. The model supports a strict separation between functional and communication behavior. Designs are made reusable by describing their functional behavior while maintaining an abstract model of their communication behavior. When a design is actually (re-) used in a system, the specification method allows one to refine the abstract communication model into a detailed behavior that is more appropriate in the system context. The same specification method is used to model off-the-shelf programmable processors and these models are used in a processor independent hardware/software co-design methodology.

Synthesis tools and compilers are able to implement all processor, accelerator, and memory components once the global system architecture has been defined. The CoWare design environment provides for integration of existing design technology by automatically generating the interfaces that link these design environments and by

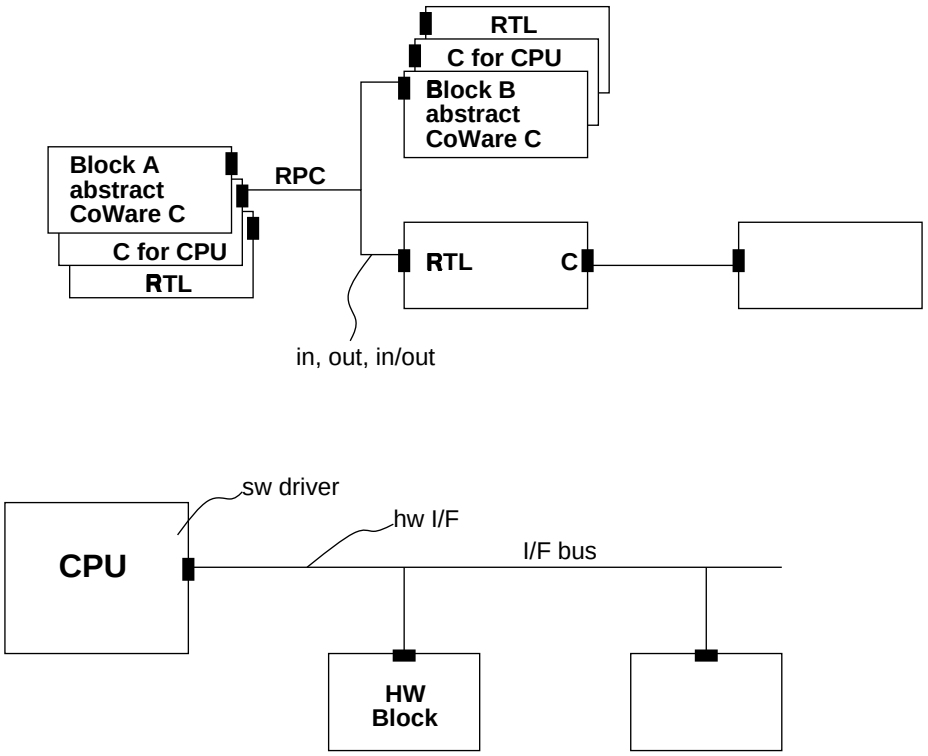

Figure 11: CoWare simulation paradigm and sample implementation

interfacing the generated and off-the-shelf processors in a way that is consistent with the system specification.

Designing a system with the CoWare environment involves four steps: *functional specification, architecture definition, communication selection,* and *component implementation*[6].

Functional specification: A system is specified by means of communicating processes that exchange data via channels. The behavior of a process can be entered using a host language such as C, DFL or VHDL.

Architecture definition: Optimally allocate processors, accelerators and memories, binding them to the functional specification. This interactive allocation and binding step includes the hardware/software partitioning.

Communication selection: Automatically generate the necessary software and hardware to make processors, accelerators and the different environments communicate. This step is performed via the SYMPHONY interface synthesis toolbox. Communication blocks (CB) provide pipelining and synchronization between accelerators. The communication between the hardware and the software for the ARM processor is more complex. The ARM interface includes address decoders, DMA channels, interrupts and I/O ports. Within the ARM, software drivers must be synthesized and linked to the processes running on the ARM.

Component implementation: All components in the system such as accelerator processors, interface hardware and software, memories, software running on a processor core, and debugging blocks are implemented using existing design environments. CoWare embeds different component compilers into the design environment, such as the ARM C-compiler and commercial VHDL/DSP synthesis environments.

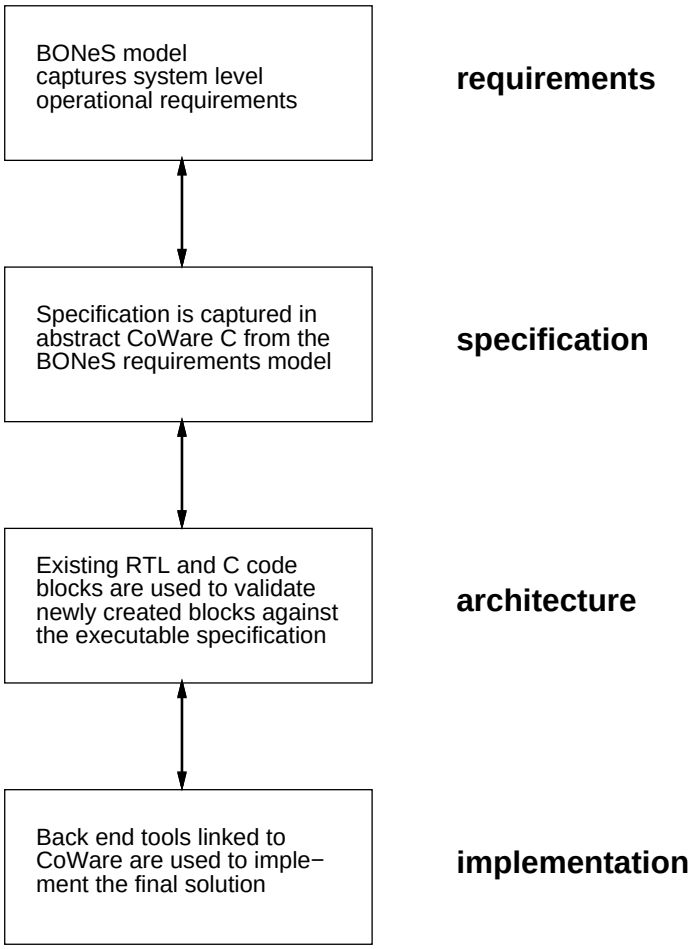

Figure 12: System Level Design Flow

5.2.1 CoWare simulation

The basic model of communication in CoWare is the *remote procedure call* (RPC). An example can be seen in Figure 11. The RPC connections can be seen between blocks. The cascaded blocks show different abstraction levels of the same functionality. The "abstract CoWare C" is C code written for CoWare and not targeted to any particular processor. "C for CPU" is C code targeted to a particular processor, e.g. an ARM. Finally, RTL is a Register-Transfer Level description in some HDL, typically Verilog or VHDL. Notice that any RPC connection can communicate with an RPC connection at any other level of abstraction – abstract CoWare C, C for CPU, or RTL.

The bottom half of Figure 11 shows a hardware implementation of the RPC communication paradigm. CoWare synthesizes the software device driver as well as the logic in hardware to read data from the interface (I/F) bus.

Thus, a mixed system level specification in which part of the system is already implemented while another part is still specified at the behavioral level can be co-simulated. For this purpose, existing simulators can be integrated into the environment. At the time of use, a commercial VHDL simulator and the ARM instruction set simulators (both instruction accurate and cycle accurate) had been embedded.

CoWare operates very much like a linker, providing an executable that can be linked to instruction set simulators as well as other modules.

5.2.2 CoWare limitations

The CoWare methodology imposes increased demands on the generation of library elements. Abstract and detailed models of IP blocks usually do not both exist. Existing IP blocks, for which Verilog code currently exists, will require additional work to generate validated abstract CoWare C models.

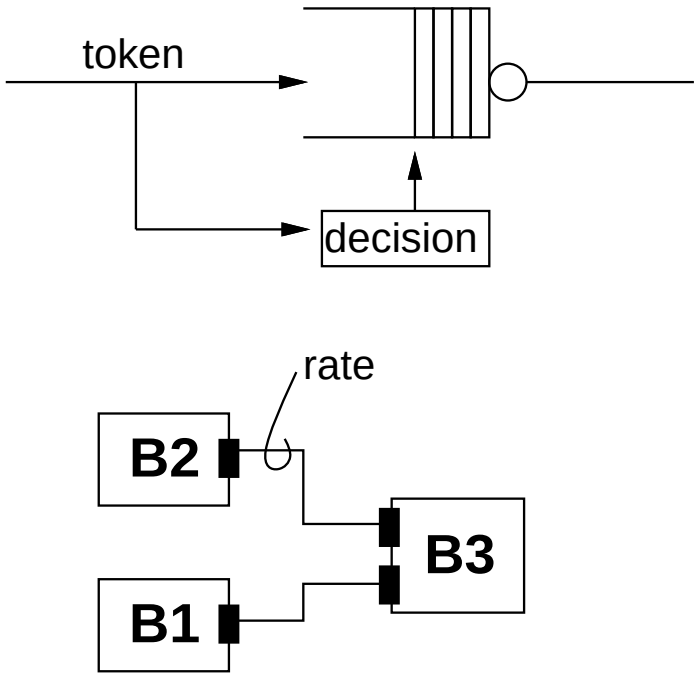

Figure 13: BONeS Queuing Model and Three Implementation Blocks: A simple BONeS queuing simulation is used to model the performance of RPC's in the more detailed CoWare system models.

5.3 System Level Design Flow

The leading edge system level design flow used in this case study at National Semiconductor can be seen in Figure 12. This is the specific workflow based design flow adopted for the LAN Adapter design project.

As an example pass through some of the workflow stages of Figure 12, consider the BONeS queueing model shown at the top of Figure 13 and the logical implementation giving rise to the queue at the bottom of the figure. In the BONeS model, tokens move around the model and proceed through the queue based on a "decision." The decision is based on one of the following:

- a predetermined rate
- a logical decision based on arrivals of tokens, e.g. an *and* of two queues (both queues must have at least one token before passing the tokens through)
- random

Thus, in the BONeS model, there is no concept of the implementation. But the logical blocks `B1`, `B2` and `B3` shown at the bottom of Figure 13 **do** affect other blocks in the system – it's just that the details are abstracted away in the BONeS model.

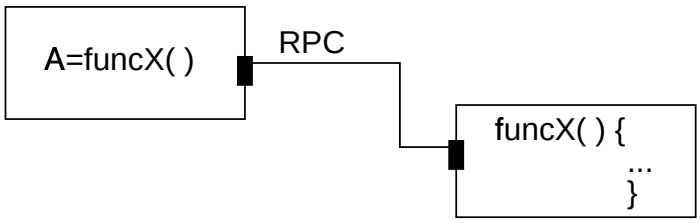

Figure 14: CoWare model of two implementation blocks

Consider Figure 14 which shows how two of the implementation blocks from the bottom of Figure 13 are modeled in CoWare. The designer accomplishes refinement by replacing abstract CoWare C models for the blocks with C code targeted to the ARM or by RTL. Everything in CoWare looks like a procedure call, which is coded up in detail in the C/RTL models. For example, in the LAN design, one could examine some of the CoWare blocks corresponding to blocks in the BONeS model of Figure 8 to see the resolution of ethernet packets.

Now, execution of the model in Figure 14 results in extremely long simulation time. Fortunately, however, we have a corresponding model in the top of Figure 13 which simulates much faster, e.g. by replacing a bit-level simulation producing packets with a process emitting tokens (packets) according to a probability distribution. Thus, when trying to accomplish a complete system simulation modeling the network node, CPU, operating system, and network peripherals, the BONeS simulation is preferred. However, to keep the BONeS simulation as accurate as possible, it will be necessary to model the actual blocks in CoWare and run the CoWare simulation for a short time to obtain more accurate "decision" parameters for BONeS. This allows the system designer to learn many things, e.g. how often a particular behavior is executed. For example, we could find out that `B2` is executed at a high rate, making a hardware implementation for `B2`'s functionality more attractive. Choosing such an implementation alters the BONeS parameters. Thus, we move back and forth in a workflow based design flow. For the LAN design, about 20 iterations back and forth between BONeS and CoWare were needed. Note that BONeS is **not** a testbench as we were not attaching BONeS models to an external event driver. Clearly, the value of using BONeS depends on the accuracy of the assumptions and parameters used.

The design flow of Figure 12 for node applications attempts to take maximum advantage of reuse of hardware and software elements by leveraging system level modeling. The flow applies to many different possible end applications. Significant effort in library creation is required before the benefits are seen.

5.4 Time line information for BONeS/CoWare work

The following summarizes the time spent by three engineers full time on the LAN adapter design:

(A) LAN (e.g. 100BaseT Ethernet) recommendations and framework
Completed June 13, 1997

(B) LAN (e.g. 100BaseT Ethernet) evaluation
Completed June 20, 1997

(C) IEEE 1394 recommendations and framework
Completed July 11, 1997

(D) IEEE 1394 evaluation
Completed July 25, 1997

(E) 1394/LAN bridge recommendations and framework
Completed August 8, 1997

(F) 1394/LAN bridge evaluation
Completed August 15, 1997

The "recommendations and framework" represents a complete pass from requirements to implementation, as shown in Figure 1. An "evaluation" included working the process from implementation back through to updated requirements and reconciling each element. Most of the time was spent **not** on simulation (either BONeS or CoWare) but instead was spent manually coding up and verifying the different models – abstract CoWare C, C for CPU, and RTL. For example, a common occurence was to run CoWare for a minute, get a parameter to apply to BONeS and then run BONeS for two minutes. Then, the next half hour was spent coding a new model.

Notice that the workflow here is sort of like a spiral, with parts of the design being modeled and tested before other parts are defined. i.e., the "versions" of the spiral represent growth of complete functionality in an evolutionary manner.

The interface between the "partial architecture" (in BONeS) and the "partial implementation" (CoWare abstract C) was tested only to the extent that we knew that we could move forward to an implementation path.

5.5 Project status

The following items were completed for the LAN adapter project:

- developed complete abstract model(s)
- defined chip and embedded software
- defined windows driver architecture
- tested BONeS – CoWare refinement

On the other hand, the following two items were not accomplished:

- enter complete, full design in CoWare
- complete SBP-2 based LAN drivers

The project was conceived in order to explore a new hardware/software co-design methodology that could be used at National Semiconductor. Thus, we completed the developing of a system level simulation in BONeS, working out the driver architecture, and providing an implementation design flow.

6 Conclusion

The next major step in the industrial adoption of system level design tools is improved model generation and validation of IP block models across mixed abstraction levels. System level design makes extensive use of valuable existing IP blocks. Since the existing blocks are described in the most expressive language format available for the particular block, many formats and coding styles are used. This heterogeneous implementation style makes the process of generation of libraries of abstract components difficult.

Hand coding a system level description of an existing IP block is an expensive and error prone task. What is needed are improved methods of validating, automatically generating, and encapsulating existing blocks into system level frameworks. Better tools and methodologies for validating newly hand generated abstract descriptions against existing implementation level models are needed to speed the library creation processes in cases where the system level simulation is of crucial importance. In the future, the hand generation process needs to be supplemented by automatic tools which can generate a system level model from observations of the behavior of implementation simulations. In the meantime, more effective methods of encapsulating implementation models in a wrapper which can perform abstraction level matching at run time could fill the gap between hand generated systems models and automatically generated systems models, as well as speed the adoption rate of systems based design.

Acknowledgments

We wish to thank National Semiconductor for supporting the publication of this work and for supporting research at Stanford by joining the Center for Integrated Systems as well as by awarding a National Semiconductor Fellowship to the second author of this paper. The second author is also partially supported by ARPA, under grant No. DABT63-95-C-0049.

References

[1] G. De Micheli and M. Sami, editors, *Hardware/Software Co-Design,* Kluwer Academic Publishers, Norwell, MA, 1996.

[2] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, Ellen Sentovich, Kei Suzuki and B. Tabbara, *Hardware-Software Co-Design of Embedded Systems The Polis Approach,* Kluwer Academic Publishers, Norwell, MA, 1997.

[3] J. Henkel, Th. Benner, R. Ernst, W. Ye, N. Serafimov and G. Glawe, "COSYMA: A Software-Oriented Approach to Hardware/Software Co-design," *The Journal of Computer and Software Engineering,* Vol. 2, No. 3, pp. 293-314, 1994.

[4] R. K. Gupta, *Co-Synthesis of Hardware and Software for Digital Embedded Systems*, Kluwer Academic Publishers, Boston, MA, 1995.

[5] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal on Computer Simulation,* Vol. 4, pp. 155-182, April, 1994.

[6] D. Verkest, K. Van Rompaey, I. Bolsens & H. De Man, "CoWare–A Design Environment for Heterogeneous Hardware/Software Systems," *Design Automation for Embedded Systems,* Vol. 1, No. 4, pp. 357-386, October 1996.

[7] H. De Man, I. Bolsens, B. Lin, K. Van Rompaey, S. Vercauteren, and D. Verkest, "Co-design of DSP Systems," in [1], pp. 75-104.

[8] Serial Bus Protocol 2, ftp://ftp.symbios.com/pub/standards/io/x3t10/drafts/sbp2 .

[9] P. Willekens, et. al., "Algorithm Specification in DSP Station using Data Flow Language," *DSP Applications,* pp. 8-16, January 1994.

[10] M. Kamath and K. Ramamritham, "Modeling, Correctness & Systems Issues in Supporting Advanced Database Applications Using Workflow Management Systems," UMass CS Technical Report 96-07, Jan 1996. Available from http://www.cs.umass.edu/csinfo/techrep.html .

[11] H. Lavana, A. Khetawat, F. Brglez, and K. Kozminski, "Executable Workflows: A Paradigm for Collaborative Design on the Internet," *Proceedings of the 34th Design Automation Conference,* pp. 553-558, June 1997.