Microarchitecture Development via Metropolis Successive Platform Refinement

Douglas Densmore
University of California, Berkeley
545Q Cory Hall
Berkeley, CA 94720
densmore@eecs.berkeley.edu

Sanjay Rekhi
Cypress Semiconductor
3901 North First Street
San Jose, CA 95134-1599
syr@cypress.com

Alberto Sangiovanni-Vincentelli
University of California, Berkeley
515 Cory Hall
Berkeley, CA 94720
alberto@eecs.berkeley.edu

Abstract

Productivity data for IC designs indicates an exponential increase in design time and cost with the number of elements that are to be included in a device. Present applications require the development of complex systems to support novel functionality. To cope with these difficulties, we need to change radically the present design methodology to allow for extensive re-use, early verification in the design cycle, pervasive use of software, and architecture-level optimization. Platform-based design as defined in [1], has these characteristics. We present the application of this methodology to a complex industrial application provided by Cypress Semiconductor. In this case study, we focus on a particular aspect of this methodology that eases considerably the verification process: successive refinement. We compare this approach versus a parallel team of designers who developed the IC using standard design approaches.

1. Introduction

The design of complex ICs such as a microprocessor often starts with a high-level description of the intended operation described in C, C++, or, more recently with SystemC [7]. Verification is performed using ad hoc simulators that typically run orders of magnitude faster than RTL ones. Once the designers are satisfied, the description of the design is translated manually in an RTL description. This RTL description becomes the "golden model" against which all implementations are compared. There is no guarantee that the high-level description of the design is functionally equivalent to the RTL description and there is no method for back annotation of the original model for modifications. As complexity increases, this traditional method shows signs of severe stress since verification times are reaching levels that are no longer affordable. Verification could be much faster and accurate were higher levels of abstraction introduced. Rigorous "refinement" into the lower levels would allow verification performed at higher levels of abstraction to not be repeated. Hence, design modifications would be reflected through all layers of abstraction. Figure 1 demonstrates this approach (b) vs. a traditional flow (a). To implement (b), we need an *intellectual framework, a design methodology, and the supporting tools*.

These three elements can be found in the Platform-based design methodology as described in [1] and in the Metropolis design environment [3]. The *intellectual framework* is provided by the semantics of the language used to represent the design formally so that its properties can be assessed. The semantics have been designed to support a variety of heterogeneous models of computation allowing the designer maximum flexibility. The semantics have been designed to support both high-level abstractions and implementations thus allowing heterogeneity in the design both "horizontally" (it supports multiple models of computation), and "vertically" (it supports different abstraction layers).

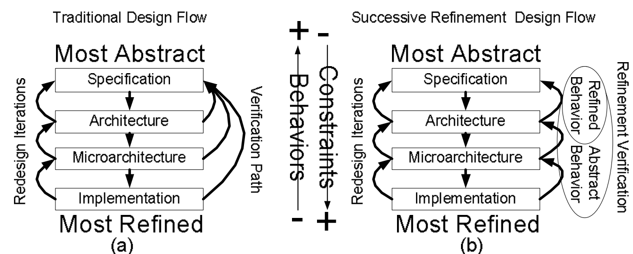

Figure 1: Abstraction and Verification in Design Flows

An essential element of the methodology and of the formal semantics is the notion of *separation of concerns* [5]. This is the idea that computation, communication, and coordination should be considered as *separate elements* of the design. By keeping functionality and architecture separate, we can explore the design space in the architectural and micro-architectural domain while keeping the functionality intact. By separating communication and computation, we favor design re-use and allow composability, so that the properties of the design could be assessed by looking at the properties of the elements and of their communication mechanism in isolation. Constraints are also important in design capture and should also be kept separate from functionality. Constraints are expressions involving design variables that cannot be evaluated early since they often relate to physical properties of the implementation.

Platform-based design has a "top-down" component and a "bottom-up" one. The bottom-up is related to the re-use of existing modules and components. Of more interest to this paper, is the top-down part, *where we "refine" the original specification in successive levels of abstraction (platforms).* Refinement can be defined precisely as commonly done in the formal verification community [8]. The fundamental aspect of this approach is that the behavior of the refined design is "contained" in the behavior of the level of abstraction above. If this refinement relation holds, properties of the design assessed at the higher level are valid at the lower level and there is no need to repeat verification of these properties at the lower level thus saving on verification costs.

The goal of this paper is to validate, at least in part, the methodology and the modeling strategy incorporated in Metropolis on a complex industrial case study provided by

1530-1591/04 $20.00 © 2004 **IEEE**
IEEE Computer Society

Cypress Semiconductor. We present the process by which Metropolis was used to go from a behavioral specification toward a specific microarchitecture via a series of abstractions and refinement verifications in the *Successive Platform Refinement Methodology*. Finally, we compare some key data points with the same design done in parallel in SystemC by another group as described in [6].

This paper is organized as follows: Section 2 describes the specifics of the case study problem. Section 3 describes design methodology and background terminology. Section 4 describes the development of the design and its platform abstractions. Section 5 examines the implemented Metropolis approach. Section 6 concludes with comments on the methodology.

2. Case Study

The goal of this exercise was to analyze the architecture of an interface unit for a very high bandwidth Optical Internetworking Forum (OIF) standard, e.g., System Packet Interface Level-4 (SPI-4), Level-5 (SPI-5) [4] with the following requirements:

- Interface must provide maximum bandwidth as required by the specification.
- No loss of data with minimum backpressure; backpressure reduces upstream traffic flow. Generate backpressure only if downstream system requires it.
- Optimally sized standard embedded memory elements. Optimal is a lower bound size with no packet loss.
- The interface supports multiple input channels.
- The insertion of idles (no activity) when packets are of different size must be minimized.

For this project we defined a simple SPI-5 data generator model that generates data every clock cycle for given number of channels. Two types of parameters are considered: *architecture* and *application*. Architecture parameters help to determine the microarchitecture parameters for various application parameters. *We should choose a set of architecture parameters that match all application parameters for a given specification.*

2.1 Application Parameters

Number of Channels (N_P) – Number of PHY units. A PHY unit is a physical layer device that converts the serial optical signal to an electrical signal.

Data Rate/Channel (B_P) – What configuration of PHY units can be used. The electrical signals from the PHY units are typically in a byte or multiples of bytes format.

The application parameters define what different types of PHY units our design could interface with. This is a tradeoff between flexibility and clock frequency. A smaller *Bp* will deliver data at a higher clock frequency.

2.2 Architecture Parameters

Our objective was to devise a robust architecture that will allow our design to interface with different types of systems. To evaluate the various architectures we defined the following two parameters:

Number of channels/bus (N_B) - Number of channels that can simultaneously deliver data at the same time.

Bytes of data/bus (B_B) - Number of bytes from each channel.

In the simplest case $N_P = N_B$ and $B_P = B_B$, i.e., the system is configured to accept and deliver the data when all channels are equivalent. However, each channel can deliver data at a different rate. The only characteristic known is that the aggregate data from all the channels will be no more than 40 Gb/sec.

We support up to 16 channels. For 16 channels, each channel must be 2.5 Gb/sec, to get an aggregate of 40 Gb/sec rate (2.5*16). Alternatively, 4 channels can each be 10 Gb/sec.

The various parameters also control the internal bus width and internal clock frequency:

Bus Width (B_W) $= N_B * B_B$

$N_P*B_P*C_{SYS}/B_W$ $\rightarrow$ (Ideally small as possible); where C_{SYS}, is the system clock frequency.

An interface unit that can interact with the PHY units and deliver data to the downstream modules can now be designed. However, the effect of our decisions at this level will impact the operation and storage requirements of the design.

Example: Consider $N_B = 8$ and $B_B = 4$, then $B_W = 32$ (Bytes). Then when $B_P = 4$ and $N_P = 16$, then data sequence is produced as shown in Table 1. For the first clock cycle, the 1st byte from the selected channels appears on the bus. In the second clock, the 1st byte from the remaining channels is delivered.

Table 1: Example of SPI-5 Data Generation

Data Transfer Byte (Bp = 4)							
1st SOP Byte		2nd		3rd		4th EOP Byte	
Channels using Bus (N_B = 8, Np = 16)							
0-7	8-F	0-7	8-F	0-7	8-F	0-7	8-F
Clock Cycle							
C=0	C=1	C=2	C=3	C=4	C=5	C=6	C=7

For this system configuration, it will take 8 clock cycles to send 256 bytes of data over an 8 channel wide bus (16 total channels) with packets of 4 data units each. This is a simplified scenario. A more constrained implementation has been described in [6].

The purpose of this study was to use Metropolis to quickly evaluate the impact of various parameters on the entire design while minimizing the verification effort.

3. Design Methodology

3.1 Refinement Verification

The notion of refinement verification for this paper stems from model checking work such as [8]. We define the problem as follows:

A model is generically defined as an object that can generate a set of finite sequences of behaviors, ***B***. One of these possible finite sequences, ***B***, is considered a trace, **a**. A trace, **a,** is considered a sequenced set of observable values for a finite

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)
1530-1591/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

execution of the module. An observable value is produced by an [Obs]ervable variable (an API of the model). A projection of a trace, **a**[ObsY], is the trace produced on module *Y* for the execution which created **a** over the observable variables of *Y*. Given a model *X* and a model *Y*, *X* refines the model *Y*, denoted $X < Y$ if given a trace **a** of *X* then the projection **a**[ObsY] is a trace of *Y*. The two modules *X* and *Y* are trace equivalent, $X \approxeq Y$, if $X < Y$ and $Y < X$. *The answer to the refinement problem (X, Y) is YES if X refines Y and otherwise NO.*

This notation refers to the refinement verification procedure.

3.2 Design Flow

We wanted to (1) observe if Metropolis could effectively aid in the process of microarchitecture design and verification as compared to the other model and (2) derive the *architecture and application parameters* described in Section 2. Our design flow should simplify the microarchitecture development and help to determine which portions of the design need to be further refined with formal analysis methods.

The notion of *successive platform refinement* was essential in our flow. Each Metropolis model represented a specific *platform* instance. Each subsequent platform_{i+1}, kept a reusable abstract specification with correct behavior and equally importantly, each successive platform held the refinement relationship *required* with its parent platform. Theoretically any microarchitecture is a candidate for refinement. In our case, we require the presence of observable communication involving computation elements.

Platform abstraction was driven by the separation of concerns as mentioned. Beginning with the initial specification each subsequent platform would address previous platform constraints and application and architecture parameters. At each step, we performed refinement verification. If the refinement relationship held, we would collect a set of data points concerning various metrics relevant to the design. Figure 2 illustrates our design flow.

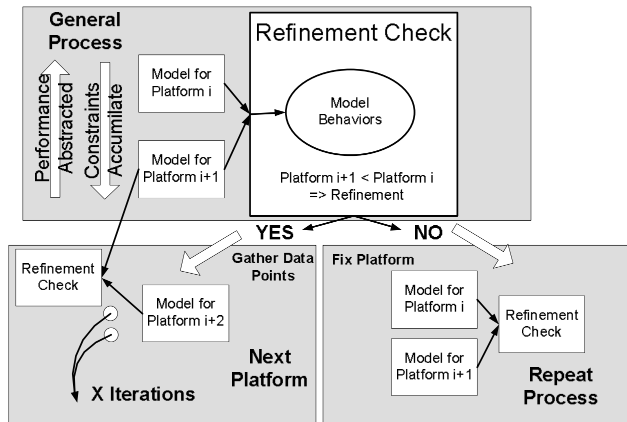

Figure 2: Successive Platform Refinement Methodology

This methodology produced several different platforms, which exposed different aspects of the application. We referred to these platforms sequentially; they drove the micro-architecture design by revealing designs that did not meet the constraints implied by the *application parameters*. Simulation performance analysis drove refinement to the next platform.

4. Platform Development

The goal of platform development is to address and transform some of constraints of the previous platform and develop the architecture and application parameters outlined previously. This creates a hierarchy of platforms with their corresponding successors and parents. Platforms naturally address changes to computation, communication, or coordination structure. This was natural for this application but can be more ambiguous for other applications. Metropolis semantics make this relatively easy.

4.1 Platform 0

Platform 0 represents the minimally constrained functionality of the initial specification. This provides the initial platform in Figure 3. This is a buffered producer/consumer where there is a data source (producer), some internal storage (buffer) and a packet processor (consumer). There is communication (A, B) but no notion of what architectural form they take (i.e. bus, shared memory, etc). There is only notion of direction (read or write) and that A and B can only be accessed by one element per unit time. The initial system has only "constraint 0":

Constraint 0 - *Only complete packets can be delivered to the packet processors. Partial packets have to remain in the internal storage or dropped based on other system requirements.*

Inherent constraints are reflected by the application topology:

$$\mathit{MaxRateProduction(DS)} \leq \mathit{MinRateConsumption(IS)} \quad (1)$$

$$\mathit{MaxCapacity(IS)} \geq \mathit{MaxProduction(DS)} - \mathit{MinConsumption(PP)} \text{ at any instant t} \quad (2)$$

$$\mathit{DataFormat(DS)} = \mathit{DataFormat(IS)} = \mathit{DataFormat(PP)} \quad (3)$$

DS = Data Source; IS = Internal Storage; PP = Packet Processor

Equations (1) and (2) ensure that this is a loss less communication mechanism while (3) captures the fact that these are primitive communication mechanisms in which data is merely transferred not transformed. The next platform should look to transform some of these constraints.

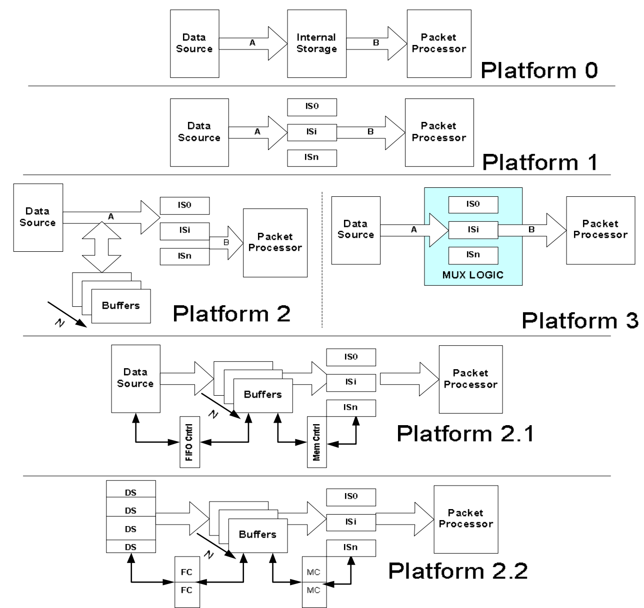

Figure 3: Platform Development

1530-1591/04 $20.00 © 2004 IEEE

IEEE COMPUTER SOCIETY

4.2 Platform 1

The internal storage for each channel depends on the data rate of the channel. A simple implementation due to this constraint can be stated as a set of refined constraints on the internal storage.

Constraint 1 - *B_P is an application parameter; hence the internal memory must allow storage space for each channel to be dynamically adjusted. Aggregate data rate of 40Gb/sec must be preserved. The number of divisions (N_M) must equal the number of PHY units, i.e., $N_M = N_P$*

With the aggregate data rate and different data rate per PHY units, application parameters were combined as in Table 2.

Table 2: Application Parameters and Platform 1

Data Rate/Phy	N_P
40Gb/sec	1
10Gb/sec	4
2.5Gb/sec	16
1.25Gb/sec	64
625Mb/sec	256

Simulations indicated that for large number of channels the current bus architecture would not be sufficient.

We discussed with other groups and decided to restrict N_P to 1, 4 and 16.

As with the previous platform there are still constraints but now they generate a relationship between platforms. These can be derived from topology (5) as before or Metropolis semantics (4).

Coordination (Platform1) > Coordination (Platform 0) **(4)**

Processes (Platform 1) = Processes (Platform 0) **(5)**

This relation indicates that platform 1 will require more explicit coordination with equal processes. This will restrict behaviors, which hold a refinement relationship.

4.3 Platform 2 and Platform 3

Analysis with the above set of constraints imposes strict timing based on the clock frequency. For a large memory this will be a difficult constraint to meet. The constraint of platform 1 needs to be further refined or implemented differently. As the constraint refinement proceeds, implementation related considerations dominate. The refined constraint can now be stated as:

Constraint 2 *-The data rate and number of channel based internal storage should have pipelined writes.*

The implementation with this constraint leads to:

- Using a mux-based logic organization as shown in platform 3. We chose not to implement this scheme due to lack of formal refinement relationship.

- Using an external buffer to intermediately store incoming packets (read transaction) and then pass them to the internal storage (write transaction), as shown in platform 2.

The coordination introduced in platform 1 manifests itself as control logic as shown platform 3. This makes the coordination explicit but does not ensure refinement. We need communication refinement and to revert to a previous communication refinement of the *IS* as in Platform 2.

As Figure 3 shows, if communication (A) is actually refined into buffers as in platform 2 then there is no need for platform 3. As hoped, this will prevent the continued growth of the coordination overhead introduced in platform 1 and the refinement of the *IS* into internal memory does not change the platform properties in platform 0. The design will now proceed from platform 2.

4.4 Platform 2.1

Our analysis indicated that during peak times the read transactions dominated the system. Therefore:

Constraint 3 - *The pipelined write transaction should be independent to the read transaction.*

Platform 2.1 recognizes that coordination must be added in order to manage buffers and for constraint 3. This will require two units of control introducing added coordination. This coordination will further constrain the behavior into the refinement relationship. Figure 3 shows this refinement where the two additional process objects added in order to provide buffer management.

At this point, few architecture parameters are changing, but the refinement is proceeding more closely to a final implementation.

4.5 Platform 2.2

The "final" constraint on the system was added to have independently operating PHY units. This is important because we wanted to ensure that there were no assumptions built into our data generation and internal bus organization. The final constraint can be stated as:

Constraint 4 - *Packet generation from various channels should be independent activities.*

This refinement is performed on the data source and implements the application parameters, that is:

Number of DS = N_P **(6)**

Size of DS = B_P **(7)**

This platform shows a final refinement of the microarchitecture. This computation refinement requires a coordination refinement in order to process this data properly.

Notice that the *DS* block now is made up of multiple blocks. This requires a similar transformation for the FIFO Control (FC) and the memory control (MC). This final refinement will be **by design** a refinement of all previous platforms before it.

5. Metropolis Investigation

5.1 Metropolis Overview

Metropolis is a design environment based on the principles of platform based design [1]. Its core is the Metropolis Meta Model (MMM). This meta-model provides, via its components, processes, media, and schedulers (quantity managers), strict separation of computation, communication, and coordination concerns respectively. While enforcing this separation, it does not enforce any one particular model of computation [2]. Since it adheres to a precise semantic, the MMM supports a number of *backends* that provide services such as simulation, verification, and synthesis. For more information we refer the reader to [3].

5.2 Metropolis Models

Taking a functional specification and expressing it in Metropolis requires a decomposition into processes, media, and schedulers. Metropolis models were derived to represent platforms 2, 2.1

and 2.2. Figure 4 shows a diagram of the "final" model, platform 2.2. Metropolis processes correspond to computation (DS, FC, MC) in platform 2.2 while Metropolis media reflect memory elements (buffers). Implicit in 2.2 is the FIFO Scheduler process.

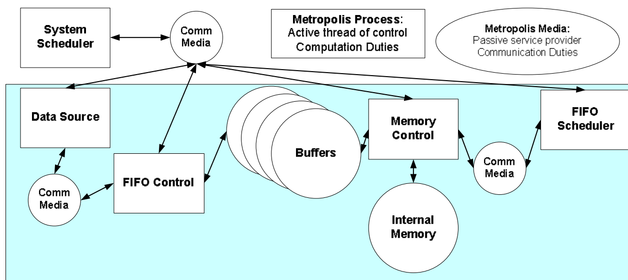

Figure 4: Metropolis Model of Platform 2.2

Processes (squares) and media (circles) define observability. The FIFO Scheduler is examined in section 5.4.

5.3 Refinement Concept in Metropolis

Due to the semantics of Metropolis, processes must communicate via media. This leads to a natural definition of observable behavior of processes in terms of their function calls to media. Recall that observable traces were a key part of our refinement definition (section 3.1).

Definition: *Metropolis Observable Behavior* – the ordered set of function calls to media that a process may execute.

This *Observable Behavior* can be captured for the processes in the model via the creation of a control flow graph (CFG). This is a tuple of <Q, q0, X, op, →>. Q is a finite set of control locations, q0 is an initial location, X is a finite set of variables, and Op are operations which denote (1) function calls to media (2) Boolean predicates. An edge (q, Op, q') is a member of a finite set of edges and the transition relationship→, is defined as (Q x Op x Q). An edge makes a transition based on the Op present, q→Op q'.

The creation of a CFG for *single threaded processes* will be used in our discussion to check for refinement relationships between the various abstraction levels.

5.4 Metro Platform Refinement Verification

After the creation a subsequent $platform_{i+1}$, the second step to continue our development process was refinement verification. Our procedure, in keeping with our *successive platform refinement* methodology, was concurrent with each subsequent platform development. **We would consider this Platform $_{i+1}$ only if the answer to the question, (Platform $_{i+1}$, Platform $_{i}$) was YES.**

Refinement verification required the creation of a control flow graph (CFG) for both the abstract and the refined model to capture the behaviors, *B*, of each model. The CFG creation can be done via a backend service (Section 5.1) in Metropolis that extracts this information automatically.

A trace, **a**, is determined by the traversal the CFG. This represents execution of the model. Once the set of traces, *B*, for each model is determined, the refinement verification stage is simply ensuring that the behavior of the refined model is a subset of the abstract behavior.

Refinement Verification via CFG Creation - For each process, *P*, in the Model, *M* (1) Create a CFG with the Metropolis Backend. (2) Identify a cycle in the CFG, this is a trace **a**. (2) Add, **a,** to the set of behaviors, **B**. (3) Continue until all cycles are identified. (4) Compare the behaviors B_{ref} to the abstract behavior B_{ab}. (5) If $B_{ref} \subseteq B_{ab}$ return YES; Else return NO.

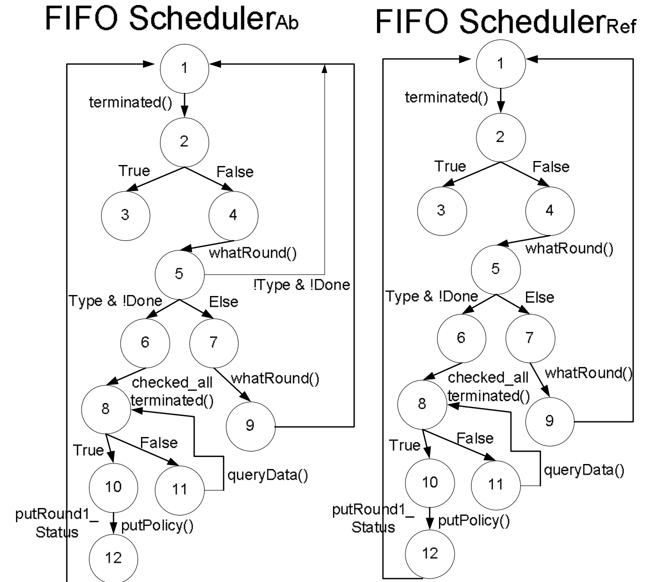

Figure 5: CFGs for Metropolis Processes

Figure 5 shows the control flow graphs for two particular processes in platform 2.1 (ab) and 2.2 (ref). This is just one example of the 5 processes shown in figure 4. The circles are the control locations, Q. Control location 1 is the initial location, q0. The operations, Op, on each transition,→, are specific function calls used in the model (denoted by "()") or Boolean predicates. The cycles in these represent possible execution traces of the model and are show in Table 3.

Table 3: Traces for FIFO Scheduler Process, B_{ref} and B_{ab}

Trace	FIFO Scheduler Process Traces (*function calls abbreviated)			
T1	Terminated()			
T2	Terminated()	wRnd()*		
T3	Terminated()	wRnd()*	wRnd()*	
T4	Terminated()	wRnd()*	Tnated()*	qData ()$^{*\omega}$
T4 cont	putPolicy()	PR1S()*		
Bref = {T1, T3, T4} ⊆ Bab = {T1, T2, T3, T4} ⇒ Refinement!				

Naturally since these are cyclic graphs there must be some notion that each cycle may be subsequently followed by any other cycle in the set infinitely often. We will use ω for this.

Therefore, the abstract FIFO scheduler is **$\{T1, T2, T3, T4\}^{\omega}$** and the refinement is **$\{T1, T3, T4\}^{\omega}$**. Notice that the FIFO scheduler trace has a function call, qData(), which also is denoted with a ω. This is due to the loop shown in the graph containing finitely many calls to this function. This shows the nested use of ω.

The creation of the CFG is automatic and the evaluation of the traces via graph traversal is manual but will be automated in the

IEEE COMPUTER SOCIETY

future. This demonstrates refinement verification in the design flow prior to creating another platform and gathering data.

5.5 Design Data Points

There were two primary concerns of this investigation. (1) How did this method compare to the concurrent investigation in [6] (see Table 4). (2) What information could we learn regarding the application and architecture parameters?

Table 4: Design Effort Comparison

Development	SystemC [6]	Metropolis
Man Weeks	15	4
Code Size (~lines)	5200	1700
Design Aids	Algorithm failure and system throughput	Evaluate alternate architectures quickly
Primary Effort	Test environment development	Modeling and Model refinement
Strengths	Mature, commercial environment	Platform Based Design Models of Computation Flexibility

The SystemC modeling and testing environment provides a good mechanism to measure architecture metrics. However, it still requires a considerable effort, because the testing and modeling environment have to be built together. Evaluation of alternate architectures is also a considerable effort with the SystemC-based method. However, once an architecture platform has been obtained, reuse of the test environment to measure throughput and find algorithmic bugs is simplified.

The Metropolis environment not only provides the same capability as [6]'s design specific modeling and test environment, but it also provides a means to evaluate alternate architectures quickly. We estimate that, assuming knowledge of Metropolis design, this procedure is roughly a *5% increase in effort* (design time and tool overhead) compared to "normal" Metropolis use.

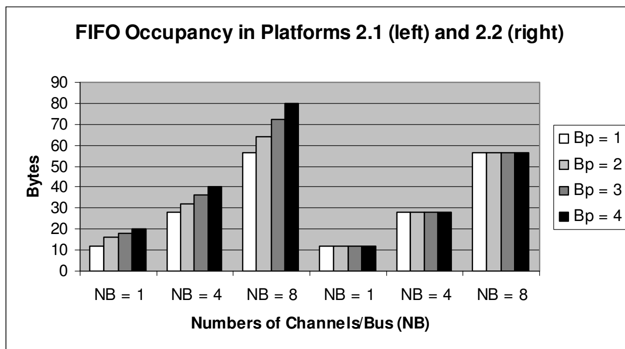

Figure 6: FIFO Occupancy Data

Finally, Figure 6, a sample of the data analysis possible in our design, shows FIFO occupancy between subsequent platforms in combination with changes in both architecture (N_B) and application (B_P) parameters. Notice that the FIFO occupancy in the refined model is bounded by the worst case in the abstract model. This type of data will drive the platform development and demonstrates design exploration.

6. Conclusions

We feel that the Metropolis environment was used successfully to demonstrate microarchitecture development via a successive platform refinement methodology. It provided not only a reusable model at many different abstractions but also supplied an environment to continue the refinement process. Each platform gave a unique perspective of the application and architecture parameters and produced a new design space given a different focus on a refinement property.

Platform 2.2 could be further refined towards implementation or we could explore the possibility of defining a higher platform than platform 0 in a quest for an abstraction that will ease the process of defining design variants.

The design was functionally verified at each refinement step, thus providing increased confidence in its correctness.

Metropolis is an integrated framework where the entire design could be carried out. In addition, given its architecture and the structure of the meta-model, the design can be exported to other design environments, especially for implementation, providing flexibility that is of high value for industrial designers.

Acknowledgements

Thanks to entire Metropolis design team in particular Yoshi Watanabe, Guang Yang, Felice Balarin, Alessando Pinto, John Moondanos, Harry Hsieh, and Cypress' Sri Purisai.

References

[1] A.Sangiovanni-Vincentelli, "Defining Platform Based Design", EE Design, March 5th, 2002.

[2] E.Lee, A.Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17(12):1217-29, Dec 1998.

[3] F.Balarin, Y.Watanabe, H.Hsieh, L.Lavagno, C.Passerone, A.Sangiovanni-Vincentelli, "Metropolis: An Integrated Electronic System Design Environment", IEEE Computer, April 2003, p 45-52.

[4] K. Gass, and R. Tuck, "System Packet Interface Level 5", Optical Internetworking Forum Contribution – OIF 2001.134, November 2001.

[5] K.Keutzer, S.Malik, A.R. Newton, J.Rabaey, A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform Based Design", IEEE Transactions on Computer Aided Design, December 2000.

[6] S.Rekhi, R.Purasai, "The Next Level of Abstraction: Evolution in the Life of an ASIC Design Engineer", Synopsys Users Group (SNUG), San Jose, 2003.

[7] T. Grotker, S. Liao, G. Martin, S. Swan, "System Design with SystemC", Kluwer Academic Publishers, May 2002.

[8] T. Henzinger, S. Qadeer, S.K. Rajamani, "You Assume, We Guarantee: Methodology and Case Studies", 10th International Conference on Computer Aided Verification (CAV), Lecture Notes in Computer Science 1427, Springer-Verlag, 1998, p.440-451.

IEEE COMPUTER SOCIETY