Chapter 6

ARCHITECTURE ANALYSIS AND SYSTEM DEBUGGING

A Transactional Debugging Environment

Antoine Perrin and Gregory Poivre
STMicroelectronics France

Abstract: Given the complexity of SoC development in the nanotechnology, it has become critical to fully validate the system performance at the early stage of the SoC design flow. This chapter describes the tools and methods for evaluating the overall SoC interconnect performance, for which the commercial solutions are not yet available. The proposed methodology is based on SystemC simulation using a generic IP Traffic Generator (IPTG) and a powerful monitoring mechanism called SysProbe, which are applicable all through the SoC analysis flow ranging from the transactional to register transfer level (RTL) simulations. Such Traffic Generators model the system IPs and the system traffic dependency with a refinement flow, while real slaves or targets are used to generate the correct latency. The SoC architecture is modeled either at the transactional or RTL level according to the requirements of development costs, simulation speed and precision. SysProbe provides the results of the architectural analysis to SoC architects.

Key words: transaction; architecture analysis; architecture platform; transactional debugging; monitoring; transactional viewer; IP traffic generator; SysProbe; traffic characterization; configuration file; initiator; target; interconnect; communication model; memory structure model; cycle accurate model.

1. DEFINING SYSTEM-ON-CHIP ARCHITECTURE

1.1 Architecture Definition

Defining a SoC architecture and micro-architecture that will sustain the real-time constraints of the targeted application is a great challenge. It is yet

F. Ghenassia (ed.), Transaction Level Modeling with SystemC, 207-240.

again another challenge to verify whether such an architecture or micro-architecture fulfils the target real-time constraints.

Assume that every IP of a SoC is sustaining its real-time constraints, the architecture/micro-architecture definition and verification with respect to the SoC performance must then focus on the following critical components:

- communication structures[1];
- shared memory controllers.

To help define these communication and memory structures, an environment comprises the appropriate tools, models, and the associated method must be made available. This environment addresses not only the SoC architects working on the communication and memory structures, but also the verification engineers verifying the compliance of the SoC implementation with the application constraints.

Two main input categories are distinguished for this environment:

1. *IP Traffic Characterisation*.
   Every SoC IP that influences the architecture definition must be modeled in terms of the traffic it generates.
2. *Application Real-time Constraints*.
   A given SoC targets a specific application or application domain. The real-time constraints associated with this application must be made accessible so that the estimated or measured SoC performance can be compared to the performance results analyzed using the application constraints.

The greatest challenge to implementing the methodology based on the above environment is getting SoC architects, who currently use spreadsheets to define the SoC architecture, to adopt this new approach. The appropriate solution must therefore propose a very simple iteration cycle loop without obligating the users to learn a new debugging language. The number of components should also be reduced to the minimum by eliminating those components that have no direct impact on the performance analysis and system debugging.

The simplification of the SoC platform assembly can be attained by adopting the SPIRIT automation strategy and the SPIRIT compliant tools (see Chapter 7). The SPIRIT automation flow is a compulsory pathway to implementing the new methodology described in this chapter, i.e. transactional architecture analysis and system debugging. This automation flow combines SoC components of various abstraction levels with high

[1] Including FIFOs (first-in-first-out) used to access to the communication backbone.

efficiency. The next section gives the list of the components constructing this environment.

1.2 Components of Architectural Platform

The transactional debugging and architecture analysis environment is composed of the following components:

- *Analysis Tool (AT)*
  AT monitors a simulated system in order to provide the results that are directly related to the target application constraints.
- *Intellectual Property Traffic Generator (IPTG)*
  The IPTG reads a configuration file describing an IP in terms of its traffic in order to re-generate the corresponding traffic on the communication backbone. Advantages of using generic traffic generators include avoiding time delay due to unavailable models, easier maintenance than C models, and direct accesses to the traffic scenario for validation.
- *Communication Model (COM)*
  The COM models the communication backbone of a SoC platform. It serves during the analysis phase to help define the communication micro-architecture features such as topology, arbitration, and FIFO size.
- *Instruction Set Simulator (ISS)*
  The ISS is used for three cases. First, communication structures and memory controllers are often programmed by a processor. During the architecture analysis, the ISS is used to perform this programming task. Second, the ISS is frequently used for the analysis of interrupts. Third, the traffic generated by the processor must be taken into account as well. The ISS can handle this task adequately. While the usage of ISS is obligatory for the second purpose, the other two purposes can be served just as well by using a generic traffic generator.
- *Bus Functional Model (BFM) or Transactor*
  The BFM or transactor establishes and assures the correct integration and communication between components of different abstraction levels. This component allows a progressively refined model to be easily integrated into a SoC platform throughout the design cycle, for instance, starting from TLM IP, to BCA IP, and finally RTL IP.
- *Memory Structure Model (MEM)*
  The MEM models the memory controller and the memory module with enough details to accurately represent the access latency.

The components introduced above define the overall architecture of a SoC platform as depicted in Figure 6-1. These components can be applied in a modular manner to adapt for the specific context of a SoC design team.

A generic approach to the SoC performance analysis and verification is described hereafter as the starting point for defining specific approaches.

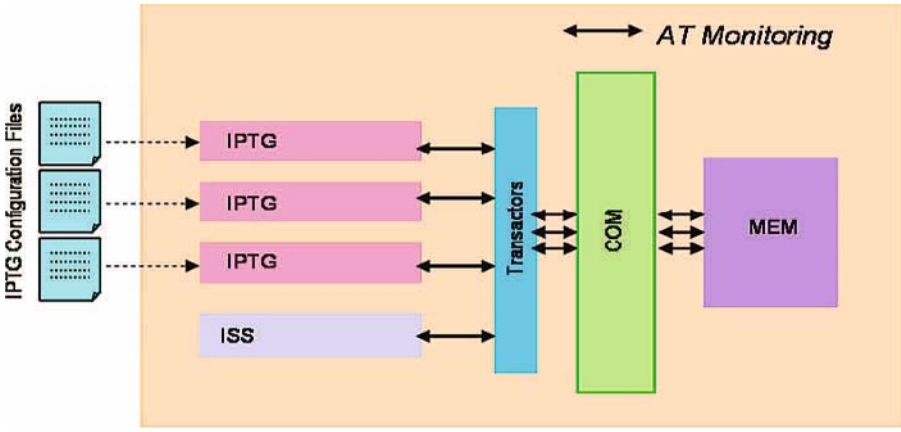


Figure 6-1. The Architecture of a SoC Platform

Three phases are undertaken as generic approaches, starting from the early definition of the main SoC architectural components down to the verification of the real chip performance.

1. *Early Micro-Architecture Definition.*
   This definition is based upon IPTG, COM and MEM models. The RTL model of the memory controller and the behavioral hardware description level (HDL) memory model could probably be used if the abstract models are not available. The analysis environment is provided by the AT. This phase aims at defining the major micro-architectural SoC features such as topology, FIFO size, arbitration, and IP clustering.
2. *RTL Performance Verification.*
   This verification is based on the IPTG and RTL implementations of the components under study. The AT computes the performance figures and compares them to the same features estimated during the early micro-architecture definition. The IPTG configuration files applied in the first phase are reused here to generate the identical traffic. Indeed, this phase verifies if the communication and memory models are in compliance with the equivalent RTL implementation.
3. *On-Chip Performance Verification.*
   The third phase is based on the real chip. During the chip verification, an on-chip performance monitor extracts traces from the chip activity.

These traces are given as input to the AT that will subsequently compute the performance figures and compare them with the measured performance results on the RTL implementation. Some discrepancies will be noticed because the traffic is generated by the real IPs in this phase while it is generated by IPTG in the RTL model. This comparison is very useful to understand how accurate the IPTG configuration files are with respect to the real IP traffic. The third phase verifies the accuracy of the IPTG versus the real IPs in the real context.

2. TRANSACTIONAL DEBUGGING

2.1 The Need for Transactional Debugging

The current SoC generations are based on the multiple initiators/masters and the multiple targets/slaves. A powerful routing system is required to interconnect all of these IP blocks, for instance, OCP [1], STBus [2], and AMBA3.0 [3]. The efficiency of a routing system in conducting the performance analysis depends strongly on the functionality and the programming of the system.

Today, the routing system has two drawbacks. First, the complexity of the routing system continues to grow exponentially. Such growth makes it impossible to carry out the conventional manual traffic analysis and architecture study on paper. Although this manual analysis continues to be helpful in defining the basic system architecture, a simulation tool must be used to perform a complete traffic analysis and architecture study. Second, the routing system may result in a system with mixed frequencies and a huge number of IP instantiations of mixed protocols during the simulation of the system integration.

If a problem occurs during the SoC integration, engineers will need to check all components of the SoC platform simultaneously. This could be a tedious and lengthy job. Consider that a routing connection includes 20 signals in average. If the integration problem occurs, engineers might have to check up to thousands of signals! Each of these signals represents one line in a traditional waveform viewer. Bear in mind that a real signification of these signals can only be interpreted by combining several signals.

All the problems described above have raised the need for an efficient solution. This chapter describes our methodological approach, *transactional debugging*. The principle of the transactional debugging lies in the transformation of such signal combinations into a unified transaction, with the intention to collect all the necessary information in the same location.

By adopting the transactional debugging methodology, the debugging effort is significantly reduced. In the example above, a direct advantage is the reduction of 20-line simultaneous cross-check in a waveform viewer per signal trouble-shooting.

Moreover, the transactional debugging helps to avoid the typical lengthy and tedious study of the different bus protocols for understanding the bus communication in a system. Not only are time and efforts saved, but the analysis results of the transactional debugging are much more user-understandable and user-interpretable than those of signal analysis.

Another interesting advantage of the transactional debugging is that all types of protocols and abstraction levels could have the same representations and attributes. In addition, there is at least a common set of parameters made available for all kinds of point-to-point connections. The rest of the parameters and transactional structures are defined by the communication structure.

2.2 Definition of Transactional Debugging

Before getting into further details of the transactional debugging, certain conceptual definitions are briefly described in this section.

A transaction is defined as a unified element representing a set of data being exchanged. It includes a list of parameters with each characterized by its name and value. These parameters can be called later as attributes of the transaction. The transfer of a transaction is denoted by a starting and ending date.

A transaction stream is a set of transactions occurring under a particular context. For instance, transactions between two routing interconnections are grouped as a specific transaction stream. According to the interconnection properties, transactions can be overlapped. An overlap of transactions occurs when a transaction starts its transfer on a stream before other transactions previously stored on the same stream end their transfers. To indicate the hierarchy between different transactions, logical relations can be defined to represent their inter-relations; for instance, predecessor-successor or parent-child relations.

The transactional information is fully compatible with the corresponding signal information. Both of them can thus exist in the same environment.

2.3 Transactional Debugging Environment

The transactional debugging is essential for the current SoC generations. It raises the observation level from signals to transactions, and thus reduces the complexity of the interconnection or communication representation.

To apply the transactional debugging environment in the SoC analysis, some fundamental building blocks are required. First, monitors for all interconnections of different abstraction levels must be made available. Second, an environment supporting the transactional debugging needs to be set up. Therefore, the AT must be equipped with a set of monitors and an analysis environment for transactional debugging. Such AT environment is called *SysProbe*, standing for System Probe.

The AT monitor is a Finite State Machine (FSM) that recognizes the protocol of the communication structure in a SoC platform for extracting information such as addresses and data transferred. The AT analysis environment should support the recording, visualization, and analysis of transactions. It should nevertheless be able to mange traditional signals too.

2.4 Monitoring Principles

In the transactional debugging, monitors are made available on a given SoC platform for:

- different abstraction levels of the same communication structure;
- different communication structures.

Such monitors are built in different manner according to the associated abstraction levels. Natively available in TLM, the monitor is only surveying the actual TLM interface or communication function. Quite opposed to the idea of cycle accurate monitoring as depicted in Figure 6-2, the transactional monitor is composed of the following components:

1. Data Acquisition Components
   Either group of modules listed below is in charge of data collecting:
   a) Simulator Link Layer. Its role is to assure the connection between the monitor and the simulator for a dynamic transaction recording. The key advantage of such layer is to obtain transactions during the simulation runtime session. The only disadvantage is that the monitor must be manually instantiated before launching the simulation. Through the SPIRIT design automation, this part will be fully automated and transparent for end users.
   b) Value Change Dump (VCD) File Parse. This module is another option for data acquisition that is used in the post-processing

mode. This method is not interactive because the results can only be studied at the end of the simulation.

2. Finite State Machine
   This module is responsible for extracting the signal information collected by data acquisition components, and processing them into the transactional information according to the associated bus protocol before sending them to output modules.

3. Output Modules
   There are several output modules for handling the simulation output:
   a) Transaction Dumper. This module obtains the transactional information from the finite state machine, prepares them into the final database formats, and dumps them into the database.
   b) Protocol Checker. This module has two missions. Its first mission is to assure the transaction integrity by detecting protocol violations that may affect the attribute integrity of the information. To perform its first mission correctly, the protocol checker must be able to verify a minimum set of the protocol rules. Thus, the module is actually performing its secondary mission to verify partially the protocol compliance.
   c) Performance Analyzer. The analyzer records the native information of the transaction such as latency, frequency, occupancy, etc.
   d) Transaction Linker. Based on a specific algorithm, the linker is in charge of detecting the relationship between all transactions to deduce a "system-level link" between all transactions.
   e) Traffic Generator. This module creates a configuration file that could be reused by IPTG.

The following are the main performance figures logged for performance evaluation of a given SoC platform:

1. latency statistics;
2. pipeline statistics;
3. opcode distribution;
4. occupancy;
5. throughput;
6. bandwidth;
7. bandwidth occupation.

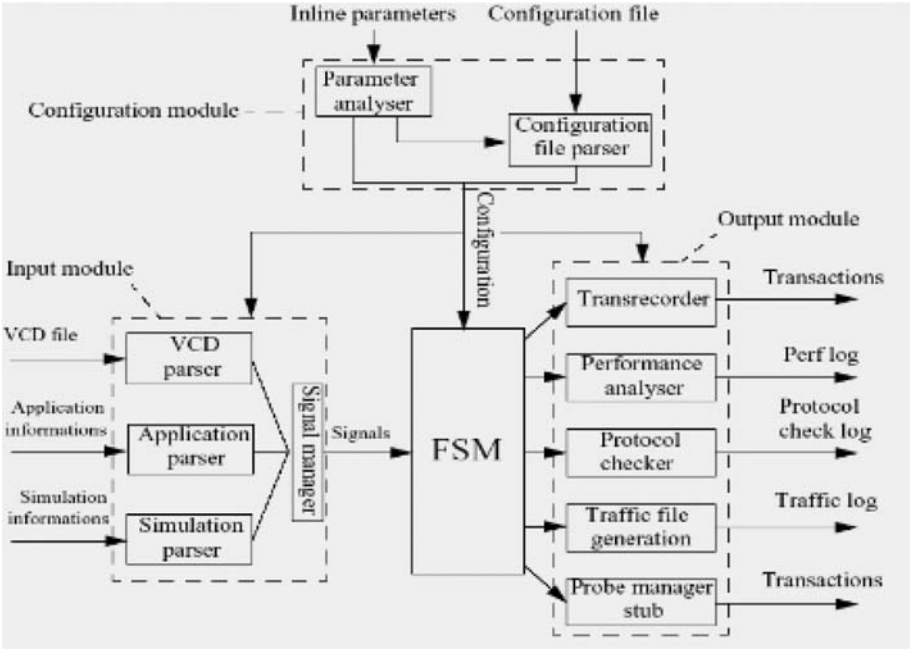


Figure 6-2. Cycle-Accurate Monitoring Structure

As illustrated in Figure 6-2, the FSM is implemented in C++. Note that the recording mechanism is implemented using SystemC Verification[2] (SCV) library. This standard improves the inter-operability between ATs by providing APIs for the transaction-based recording. Monitors used in the transaction recording allow targeting all database formats (whose recorders implement the SCV transaction dumping API with the same code through the unified API provided by SCV). To manage a new SCV compliant database format, the only action to perform is to link the new recording library with the existing probe. Thus, designers can use their own analysis environment such as text based, Cadence Incisive [4], or Novas Verdi [5].

2.5 Analysis Environment

The transactional debugging analysis environment consists of two parts:

- waveform viewer;
- user plug-in with query and add-on debugging features.

Further details on both parts are provided in the following sub-section.

[2] SCV is the extension of SystemC for verification.

2.5.1 Transactional Viewer

The real-time debugging is fully linked with the high capabilities from the AT to display transactions along with traditional signals. For this reason, all the traditional operations applicable to the signal display should also be applicable to the transactional display. Typical examples of such include comparison, search, splitting transaction attributes (analogous to splitting signals in a bus explosion), and expanding all transaction events occurring at a particular time instant.

On top of these basic display functionalities, a transactional viewer must take into account various aspects of transactional structure. It means that the viewer should provide the capabilities of displaying transaction overlaps and transaction attributes (with flexible control over which attributes to display). Cadence Incisive and Novas Verdi are two powerful tools that support the transactional display with high efficiency. Figure 6-3 shows the example of SimVision transactional display from Cadence Incisive.

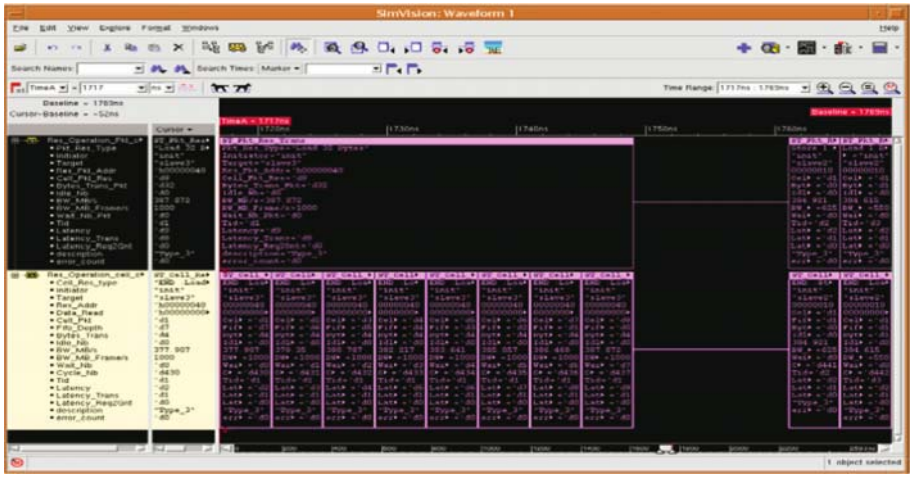


Figure 6-3. SimVision Transactional Display

2.5.2 Viewer Statistics Plug-in

The transactional monitor is delivered with a set of predefined queries, SysProbe Analysis Generator (SPAG). These queries are automatically generated for a given design based on a configuration file that depends on the communication structure, COM. All of the COMs supported by the AT

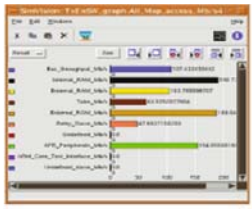
integrate a set of SPAG facilities. The results collected by these queries serve as the basic traffic statistics for that design. Three groups of SPAG queries are available as shown in Table 6-1.

Table 6-1. SPAG Query

Query Type	Query Function
Generic Query	Queries independent of COM.
COM Generic Query	COM-dependent queries for full COM analysis.
Project Query	Project-dependent queries.

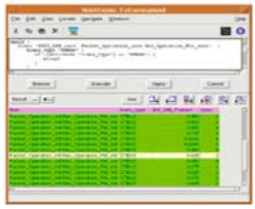
Based on the viewer statistics, various performance evaluations can be examined. As an example, typical analyses obtained through the Cadence Incisive environment are (see Figure 6-4):

1. COM bandwidth;
2. COM opcode distribution;
3. COM memory map access;
4. COM memory map bandwidth;
5. COM latency statistics;
6. Initiator COM map access;
7. Link between query database table and wave viewers.

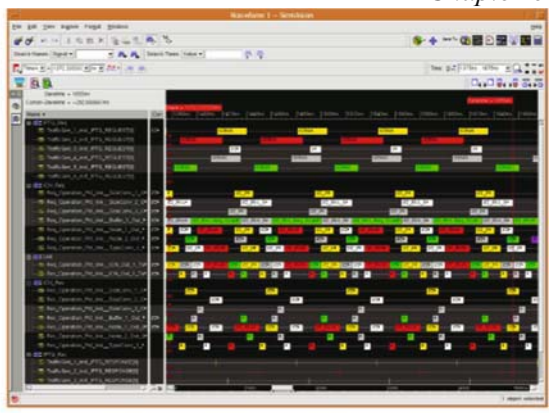


COM Map Bandwidth

COM Opcode

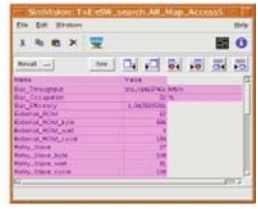


COM Access



Initiator May Access

COM Bandwidth



Latency Statistics

Figure 6-4. Cadence Incisive Statistics Plug-in Environment

2.5.3 User-defined Statistics

In addition to the SPAG query set, users are allowed to define their own set of queries. The user-defined query is based on the Cadence Incisive tool, Transaction Explorer (TxE) [6]. This tool provides users with an easy way to create specific queries by using the "browse button" where options are proposed at every step of a query creation.

2.5.4 Embedded Software Plug-in

SysProbe is delivered with a software-profiling plug-in called SysProbe Embedded Software (SPES). This plug-in associates the embedded software with the program counter (PC) signal of a given design to allow performing the hardware-software analysis. Results collected by SPES serve for creating a correspondence between the software execution and the hardware transactions recorded on the system.

Based on the disassembled code, SPES creates a correspondence between source codes and assembly codes during the debugging process to enable following the code execution by tracking the PC value. Although these

principles seem similar to those used in common debuggers, SPES provides complementary results that give additional benefits to post-mortem analyses such as capabilities of moving at arbitrary execution time, going back during execution, and software profiling.

SPES has two key features:

1. *Software Execution Display*
   Note that SPES is not a debugger. As illustrated in Figure 6-5, SPES provides a post-processing tool that allows hardware designers to understand the software execution without adding an ISS.
2. *Early Software-Profiling*
   SPES is used for profiling early software execution, particularly in analyzing the functioning of the interrupt request (IRQ) for a given SoC platform.

Typical services offered by SPES include:

1. Correspondence between the time cursor and executed source codes.
2. Correspondence between C and assembly codes.
3. Display of function calls as signals.
4. Replacement of bus opcode signals by corresponding function names.
5. Duration and frequency of function calls.
6. Execution number of a specific code line.

Instead of providing just a simple probe, the capabilities listed above can be extended to provide a system view that relates the embedded software to the whole system. Thus, SPES makes it possible to track the execution of software commands in a system. Such ability allows analyzing the software performance according to the system architecture, and determining the arbitration influence on the speed of software execution.

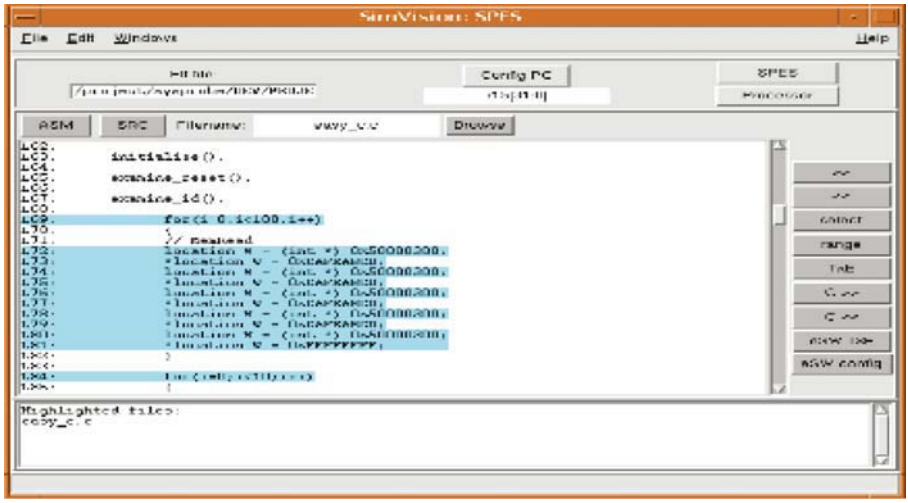


Figure 6-5. Embedded Software Plug-in of Cadence Incisive Environment

2.5.5 Transactional Link Plug-in

Transactions are characterized by particular relationships among them. The analysis environment of the transactional debugging must support features that describe the inter-transactional relationship.

The transactional viewer presented in section 2.5.1 supports such feature. The transactional linker in the transactional monitor works very well in a system where all the monitors can communicate together through the same bus protocol.

This feature, however, is not supported in certain cases where several heterogeneous systems coexist. A typical example is the simulation with mixed abstraction levels. A specific plug-in is therefore developed to handle this situation. As depicted in Figure 6-6, this transactional link plug-in creates the virtual link between transactions using a post-processing engine. This added feature is particularly useful to follow up the life cycle of transactions. By tracking the transactional life cycle, any resultant transaction during the simulation will be traced from its creation to its ending. A currently unavailable feature is a tracking from transactions to the resultant signals.

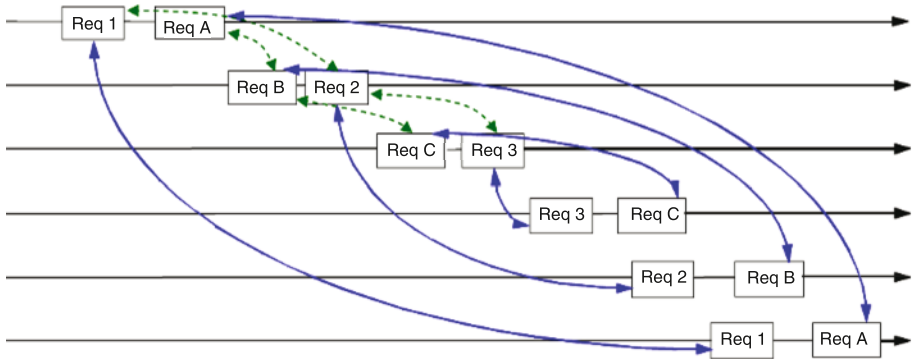


Figure 6-6. Example of Transactional Flow Link

2.6 Verification Role of Transactional Monitors

On top of its role as an analysis tool, the transactional monitor also serves as a verification tool for TLM IPs. The principle of such verification methodology is described hereafter.

To begin with, RTL signals of an IP under test are extracted and converted into transactions from an RTL test bench Based on this information, the transactional monitor will generate a set of IPTG configuration files.

Subsequently, a platform comprising an IPTG, COMs, and the TLM model of the IP under test is constructed for validation. According to the IPTG configuration files generated earlier by the transactional monitor, the IPTG will generate the same traffic as monitored at the RTL level. Through the comparison facility of the AT, the simulation results of the TLM IP are compared to those of the RTL IP for verification purposes.

2.7 Comparison of Abstraction Levels

Communication structures become similar in a certain sense as they approach the transactional level. At this point, including a subset of similar information in the communication structure will help to detect easily the discrepancy between different levels of abstraction.

A specific tool, *TransCompare*, is developed based on this concept. This tool computes the divergence percentage and lists all the discrepancy points of two traces. Such analyses can be purely functional or timed. Indeed, the engine of *TransCompare* ignores the timing information. The timing information is actually treated the same as any other transaction attributes.

By allowing the user to select on which attributes a computation is performed, *TransCompare* provides a direct access to both a pure functional and timed comparison tool. In addition to this key role, *TransCompare* is able to align the different naming conventions of transaction attributes from different database. This feature allows the transaction attributes or parameters to be correctly identified for comparison.

The main advantage of *TransCompare* is that it considers the transaction as a data flow by extracting timing information as parameter or attribute of the transaction. For this reason, this tool permits computing the functional convergence of different transactions even if their timing is completely irrelevant. An interesting added value of *TransCompare* is its transaction-filtering mechanism. Considered as data flows, transactions are easily filtered according to their attributes. Through this filtering mechanism, transactions traced from an IP can be compared to its reference even if it is in the integration phase. This method is also fully applicable to the emulation traces using the VCD input features of the monitoring tool.

3. TRAFFIC GENERATOR

As introduced in section 1.2, the intellectual property traffic generator (IPTG) is a critical component in a given SoC architectural platform. The IPTG is a SystemC block that reads a traffic characterization file (i.e. IPTG configuration file) as input, and subsequently re-generates the corresponding traffic as output on the platform communication structure.

3.1 Principles

The IPTG is instantiated in a SoC platform following the same manner of instantiating any other components. The ultimate goal of having an IPTG instantiated for a given IP is to generate the traffic specific for that IP on a SoC platform.

A typical SoC platform incorporated with the IPTG could include the components at any of the abstraction levels listed below:

1. timed transactional level modeling (timed TLM);
2. bus cycle accurate (BCA);
3. register transfer level (RTL).

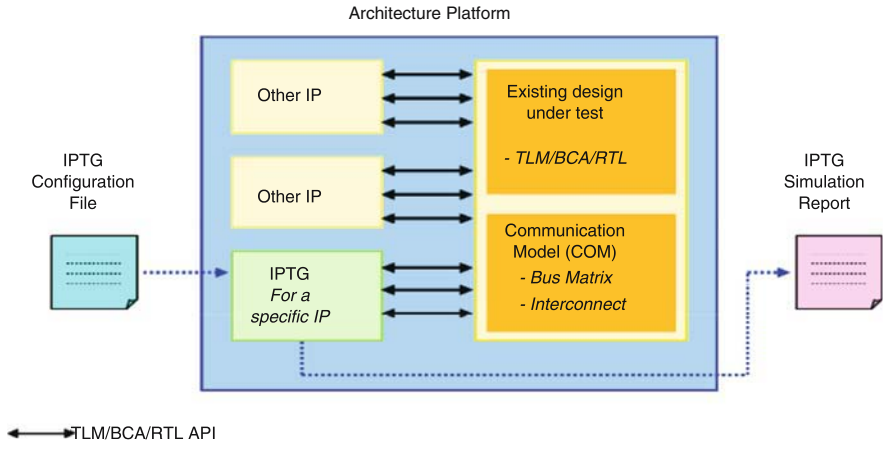


Figure 6-7. SoC Platform with IPTG Instantiation

The structure of a SoC platform with an IPTG instance is depicted in Figure 6-7. Note that the design under test shown in the figure represents an IP or a subsystem under test. Once instantiated in a SoC platform, the IPTG generates traffic on the ports of the communication model, COM. The COM ports are coded at one of the three different abstraction levels mentioned earlier: timed TLM, BCA or RTL.

As shown in Figure 6-7, the input to the IPTG is a configuration file that holds the following information:

- full statistical traffic;
- optional refinement;
- opcode sequence list;
- IP characterization parameters such as frequency and data size.

According to the information of the configuration file, the IPTG re-generates the IP traffic as the output. Another interesting feature of the IPTG is that a simulation report of the traffic generation could be produced by the IPTG for observation. In addition, a synchronization mechanism is implemented in the IPTG to model the dependency between system events.

3.2 Core Implementation

The building concept of the IPTG is based upon the standard of Open SystemC Initiative (OSCI). It is therefore fully compatible with the tools of the mainstream EDA providers. Furthermore, the IPTG is equipped with the randomization capability founded on SCV, which is an extension of SystemC for verification. Since both OSCI and SCV are open sources, the IPTG is a tangible solution totally free of charge.

3.3 Traffic Characterization

A given IP can be considered as a succession or a series of synchronized processes. The IPTG considers any single process or any group of these processes as *behavior*. An IP, therefore, is described by the IPTG as a series of behavior where each of them represents a particular type of IP traffic.

There are two approaches to define and model the characteristics of the IP traffic:

- *Traffic Modeling*. Define an IP by a set of behavior where each behavior represents specific bus traffic as seen from the external world. The IP is therefore viewed as a black box by users. The overall bus traffic of the IP could be considered as different specific traffic pieces that represent different IPTG behavior. The detailed information to configure these traffic characteristics is specified in an IPTG configuration file. In addition, there is a rather simple block to ensure a good consistency for all the behavior switching and overlapping.
- *IP Modeling*. Define an IP by a set of behavior where each behavior represents a specific internal IP traffic. This internal traffic is managed by a bus plug-in interface to subsequently create the bus traffic. The bus plug-in interface is represented by a FIFO with a threshold value and an opcode list. As illustrated in Figure 6-8, the bus traffic generated by the IP is split into two parts: (i) IP traffic that fills the FIFO, and (ii) FIFO traffic on the bus.

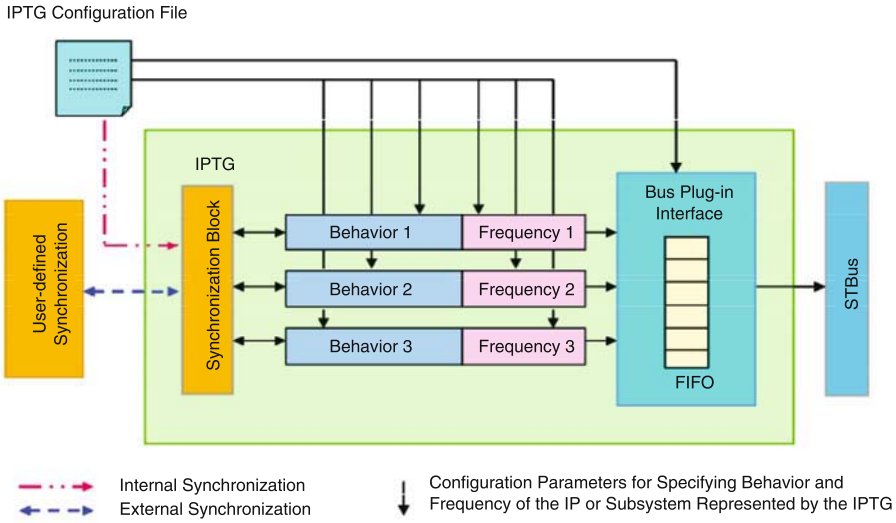


Figure 6-8. IP Modeling of IPTG

3.4 IPTG Configuration File

The IPTG configuration file is the key role of the IPTG methodology, which serves to model the behavior of a given IP in terms of its traffic.

Indeed, the IPTG configuration file is a text file with a set of parameters. Each parameter or more precisely, each keyword, is assigned a specific value as an argument. These values are the essential pieces of information to describe the IP traffic.

To ensure the development effectiveness and simplicity, users only need to define a subset of the parameters in the IPTG configuration file. Other parameters are kept optional. This flexibility allows not only a quick traffic definition but also a later traffic refinement during the project development.

An IPTG configuration file is divided into two sections:

1. *Header Section.*
   This section contains general description of an IP.
2. *Behavior Section.*
   This section provides specific characteristic descriptions of an IP.

Each section holds a list of keywords that are either compulsory or optional. The IPTG configuration file is written up by choosing the proper keywords and assigning them with the corresponding argument values. A particular grammar must be followed to develop both sections.

An IPTG configuration file could be manually written by architects or IP developers. The analysis tool, SysProbe, can also generate such a

configuration file for a given IP. It monitors the RTL/TLM simulation of the IP and generates the corresponding IPTG file as illustrated by Figure 6-9.

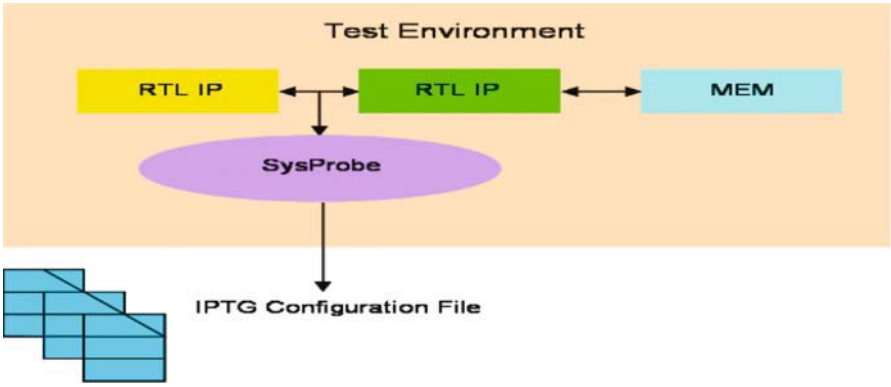


Figure 6-9. Generation of IPTG Configuration File by SysProbe

3.5 Synchronization

In order to manage synchronization issues, the IPTG incorporates a mechanism where the IPTG behavior and the bus interface FIFO are synchronized to get all the possible traffic combinations. Two approaches are distinguished for implementing synchronization in IPTG methodology:

- Configuration file-controlled synchronization;
- User-defined synchronization.

Recall that there are two synchronization blocks depicted in Figure 6-8. The synchronization block residing within the IPTG is controlled by a configuration file while the user-defined synchronization block is external to the IPTG.

3.5.1 Configuration File-Controlled Synchronization

The IPTG configuration file is extended to include the information of timing constraints specific for each behavior of an IPTG. Such information is characterized by a set of configurable parameters, which will be allocated to the current transfer in a system. Controlled by these timing parameters of the configuration file, the synchronization of an IPTG based platform can be adequately respected. Two approaches can be distinguished in handling the synchronization controlled by the IPTG configuration file:

- Linked Synchronization.
- Event-driven Synchronization.

The linked synchronization is intended for "linking" certain IPTGs together according to a set of predefined configuration rules. There are links based upon various criteria such as:

a) Process-based synchronization: A given IP is modeled by a set of processes (also called behavior). This mode assures synchronizing the different processes within a given IPTG. It is also a synchronization mode used to release time synchronization between processes coming from different IPTG.
b) FIFO-based synchronization: As illustrated in Figure 6-8, an IPTG can include a FIFO. Thus, several basic synchronizations have been developed to guarantee the synchronization of such FIFOs between different IPTGs. This feature is normally used to represent an IP that includes several bus ports. Each port is representing by an IPTG. Through such mechanism, the IP can be created by grouping all these IPTGs together.
c) Block-based synchronization: a set of traffic generators consuming data on a block-based policy; they are synchronized according to the end of each block.
d) Others: other synchronization policies are available but will not be described here.

Essentially, the linking synchronization coordinates the synchronization between all the processes within an IPTG parameterized by the IPTG configuration file. It also manages the synchronization between different linked IPTGs that correspond to the same IP. Note that these are both implementations for the "internal" synchronization of the IPTG blocks that represent the same IP. The configuration file is responsible for coordinating the different parts of the IP traffic. It is in charge of starting and stopping different behavior pieces that correspond to that IP. The main behavioral attributes parameterized in the configuration file for this purpose include:

1. random behavior succession;
2. randomization with increments or basic constraints;
3. single simulation for each behavior, i.e. no synchronization;
4. FIFO synchronization among different IPTGs.

On the other hand, the event-driven synchronization implements the system synchronization by coordinating the different IPTGs that correspond to the different IPs based upon some event-driven conditions. Such synchronization mechanism is directly included in the traffic definition of the IPTG configuration file.

Using the linked synchronization helps to obtain groups of IPTGs that represent the IPs with several bus interfaces. However, a much more

complex synchronization mechanism is needed to represent the real system synchronization between these IPs. For this reason, an event-driven synchronization mechanism is required.

One of the constraints to implement the event-driven synchronization was the lack of the ability to change the synchronization mode without recompiling the system synchronization policy. To solve this problem, such functionalities are directly embedded into the configuration file of the IPTG.

During the creation of the IPTG configuration file, a synchronization keyword (e.g. GEN and WAIT with an event name) can be embedded in each process. If the system synchronization is enabled, then the overall synchronization common to all IPTG will take care about these event during the runtime.

By bringing together both the linked and event-driven methods, the configuration file-controlled synchronization can implement quite complete but rather basic system synchronization. This approach involves all the IPTGs instantiated in a SoC architecture platform to create an overall traffic of the system. It works well if all the major synchronization aspects are independent of the routing system.

3.5.2 User-defined Synchronization

The user-defined synchronization is an alternative of refining the system synchronization of the SoC architecture platform. This mechanism is implemented in the form of an "external" block where the IP behavior or process is programmed using several IPTG-specific C++ APIs. The event occurrences related to the synchronization issues are managed by these APIs. To do so, users simply need to develop single or multiple control blocks to control the IP behavior. SystemC is strongly recommended as the programming language for this purpose because it offers the built-in synchronization blocks.

Although it allows users to fully program the desired synchronization, the user-define synchronization necessitates a good command of SystemC from the SoC architect and hence induces a significant coding cost. Furthermore, the user-defined SystemC block cannot be overloaded during the simulation runtime. A re-compilation is therefore unavoidable to adapt for this change.

Given the time and effort expenses in programming the user-defined synchronization block, the untimed TLM SoC platform could be an interesting alternative. Since the untimed TLM platform is indeed a fully functional platform with the system synchronization implemented within, it can thus be reused as some sort of "timing agent" to help defining and describing the IP traffic on the platform. Based on such "functional" descriptions, the corresponding IPTG configuration files are prepared by

splitting the different behavior pieces according to the "functional" synchronization. These descriptions are then connected to the matching untimed TLM models on the untimed TLM platform in order to build a "timed" TLM platform. As the untimed TLM platform implements very complete system synchronization, the resultant IPTGs manage to cover the most advanced parts of the IPTG synchronization. A rather comprehensive study of the SoC architecture can be realized through the management of the overall synchronization and data dependency by this method.

3.6 IPTG Simulation Report

The IPTG generates a simulation report at the end of each simulation. If there are multiple IPTGs, a single simulation report is generated for all of them. Two key roles of the IPTG simulation report are explained hereafter:

- *Verification of Expected Traffic*
  The resultant traffic from a simulation will be compared to the expected traffic as described in the IPTG configuration file for verification. If there are any violations of the expected traffic, the simulation report will list them out as warnings. The warnings will be shown at different levels according to the degree of severity. The type of violation will be listed as well, for instance, non-achieved bandwidth.
- *Tracing Effectiveness of FIFO*
  The FIFO in an IPTG bus plug-in interface is traced by the simulation report to study its effectiveness. First, a Value Change Dump (VCD) file is traced. The VCD file contains the information of the FIFO traced against time during the simulation. Second, a set of general statistical information is computed for the FIFO throughout the simulation, for instance, the maximum/minimum value of the FIFO. Figure 6-10 shows a screen snapshot of the FIFO traffic effectiveness analyzed by the tool of Cadence SimVision. Here, users can have a direct understanding of the generated traffic with transactions and of the FIFO evolution with analog signals.

The ultimate goal of producing an IPTG simulation report is to help the platform architects to observe, verify, and eventually optimize the effectiveness of a system.

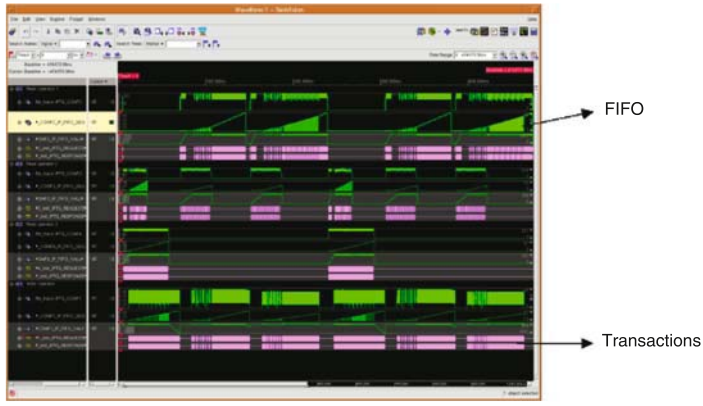


Figure 6-10. Studying Effectiveness of IPTG FIFO using Cadence SimVision

4. ISS INTEGRATION

An Instruction Set Simulator (ISS) is often required to complete the architecture analysis of a SoC platform. Considering the complexity growth of the current SoC design, the use of micro-processors has become compulsory in most of the SoC design.

By using a timed TLM wrapper and the BFM library, the ISS can be integrated into a SoC platform at the relevant level of the architecture study. The ISS is utilized for three purposes (which will be detailed in this section):

1. COM programming;
2. interrupt analysis;
3. traffic generation.

Contradictory enough, the pitfall of using the ISS is actually driving the complexity of the SoC platform much higher. The dependency on the ISS core and the associated tool-chain are additional aspects to deal with. Sometimes, the ISS could be the bottleneck of the simulation speed unless the architecture exploration is conducted at the cycle accurate level. The ISS will however become less accurate if the architecture analysis is performed at the cycle accurate level.

Considering the irremediable tendency of using the ISS in the current SoC architecture analysis, this section will briefly discuss the three main purposes of integrating the ISS in a SoC platform.

4.1 ISS for COM Programming

Most of the communication models (COM) and memory controllers of a SoC platform require appropriate programming to assure the optimal system performance. The system micro-processor is frequently held accountable for this important task.

The IPTG can be used easily to program all the required registers of the hardware IPs for this purpose. However, this method cannot guarantee the same programming of the COM for the architecture validation and for the real software delivery. This is the main reason why the ISS is still necessary in running the SoC simulation. Therefore, SoC architects have to provide the routines to configure the COM and other critical architectural components.

Another reason to include the ISS in the SoC simulation is the potential need for updating the COM arbitration dynamically. The routines of the COM arbitration may occur upon some interrupts. Unless the whole system synchronization mechanism is successfully implemented by the IPTG, such dynamic configuration can only be achieved by applying the ISS.

4.2 ISS for Interrupt Analysis

The second typical purpose of using the ISS in a SoC architectural platform is to validate the correct execution of interrupts based on the real-time constraints.

The SoC architectural platform tailored for this purpose focuses on the ISS and the peripherals that generate interrupts. Other IPs (in the form of IPTGs) are included on the platform only for generating the noise on the interconnect and memory controllers, which assures the execution of the interrupt codes according to real traffic constraints.

The analysis based on the noise generation serves as a preliminary study of the platform interrupt and traffic. A more advanced study can be carried out by using the IPTG of all IPs involved in the platform to generate a real system-level traffic.

4.3 ISS for Traffic Generation

To better analyze a system, the SoC architectural platform should consider the traffic due to the code fetching and the cache filling. In addition, the architectural platform should also take into account the functionalities executed by the system processor core such as the MP3 treatment in a multimedia platform.

Preferably, the ISS is used to execute the code to get the real traffic for a given application. To simplify the simulation platform, however, the ISS can

be replaced by an IPTG to simulate the cache refill accesses. Excluding the ISS will certainly eliminate the dependency on the ISS-specific tool-suite and debugger. The IPTG replaces the ISS by providing a generic trace that includes all of the cache refill accesses.

Before replacing the ISS by the IPTG, a simple platform consisting of the ISS and a memory is constructed to run the code. A monitoring tool is used to probe the simulation traces to create the according traffic file. This traffic file will be re-injected into the substituting IPTG. Then, a new configuration file will be created for that IPTG so that the IPTG can replace the ISS in the platform for any simulation.

5. GETTING READY ARCHITECTURE PLATFORM

This section describes briefly of how to get ready a SoC architecture platform, covering the generic SoC architecture platform, communication model (COM), memory structure model (MEM), and the accuracy trade-off.

5.1 Generic SoC Architecture Platform

Speaking of the SoC performance analysis, the SoC platform itself would be the first thing to come across one's mind. A SoC platform is typically composed of several model blocks aimed for different purposes, for instance, the communication model (COM), memory structure model (MEM), IPTGs and other IP models.

All of these blocks could be modeled at any of the three different levels of abstraction: timed TLM, BCA or RTL. These model blocks could coexist in the same SoC platform though they might be modeled at the different levels of abstraction. Bridges are used to enable the communication among these blocks. Figure 6-11 gives a better picture of a SoC platform using the IPTG methodology to perform the architecture analysis.

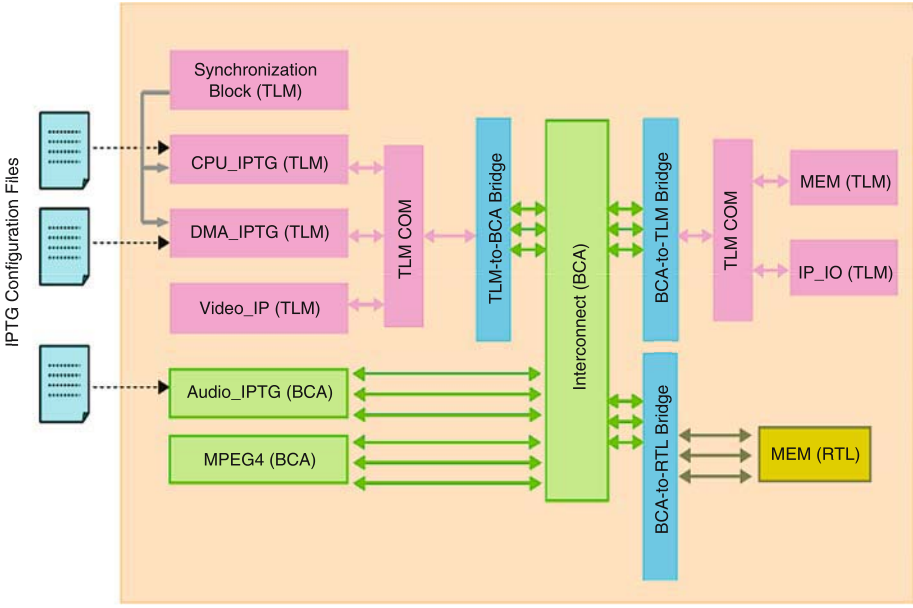


Figure 6-11. Example of Generic IPTG Platform

5.2 Communication Model (COM)

The COM is the structural backbone of the SoC platform intended for defining the communication micro-architecture features such as topology, arbitration, and FIFO size. This communication backbone can be modeled at any abstraction levels of timed TLM, BCA, or RTL, to embrace the associated communication protocol of the SoC platform.

Considering the exponential growth of SoC design today, the architecture of a typical SoC platform can easily involve around fifty initiators and tens of targets. The results of such platforms could be undesirable. Hundreds of incorrect or non-optimized routing systems may be produced along with thousands of signals holding very different programming arbitrations. For this reason, the very powerful analysis tool becomes a must in the current SoC architecture analysis.

According to the requirements of simulation accuracy and speed, a timed TLM or cycle accurate routing system is used in a given SoC project. The trade-off between the different abstraction levels for the routing system is on the account of SoC architects. Of course, the final choice is certainly dependent upon the model availability.

A timed TLM simulation aims at the early SoC architecture exploration. In this analysis phase, a very high number of simulation iteration loop is

required to increase the coverage of architecture exploration up to the whole system. Then, with the known inaccuracy percentage, it helps designers to draw an initial routing structure by selecting the best suited COM type and platform architecture.

To carry out the SoC micro-architecture validation or optimization, the COM parameters need to be programmed accordingly. Thus, using the cycle accurate model of the COM becomes compulsory in this phase. To avoid wasting time in re-coding the COM into cycle accurate SystemC models, various tools such as Tenison Vtoc [7] or Mentor H2C [8] are used to translate HDL blocks into SystemC codes.

5.3 Memory Structure Model (MEM)

The MEM is a collective name designated for all models representing the memory controllers and memory modules in the SoC platform. It can be modeled at any different abstraction levels of timed TLM, BCA, or RTL, by respecting the common rule of giving enough details to model the access latency accurately.

To perform the SoC architecture analysis correctly, the TLM MEM must be configured from an ASCII file extracted from the memory specification or RTL simulation. The reason of configuring the TLM MEM is to model the "real" timing of accesses. The delay induced inside the MEM is computed based on several parameters such as previously-accessed address, type, current access, etc.

The memory is often the bottleneck of a SoC due to the memory contention. For this reason, it is recommended to model the MEM at its fullest possible accuracy. This model can be the cycle accurate SystemC model translated from HDL. It can also be the non-functional but cycle accurate blocks, which does not respect the data consistency but the cycle accuracy of transfers. This is indeed a cycle accurate memory controller without implementing the functionality of memory accesses.

5.4 Accuracy Trade-off

The proposed methodology can ensure the compatibility of analysis at different levels of abstraction. Nevertheless, this method is suggested as a complementary solution to the spreadsheet study.

As illustrated in Figure 6-12, several studies are executed according to the accuracy requirement during a project life. However, as an incremental method, the analysis is always refined. Starting from a spreadsheet study, the HW/SW partitioning as well as the basic COM and MEM choices are realized. The spreadsheets required during this initial step are reused to

program the IPTG configuration file. According to the model available, timed TLM or cycle accurate simulations can finally be executed.

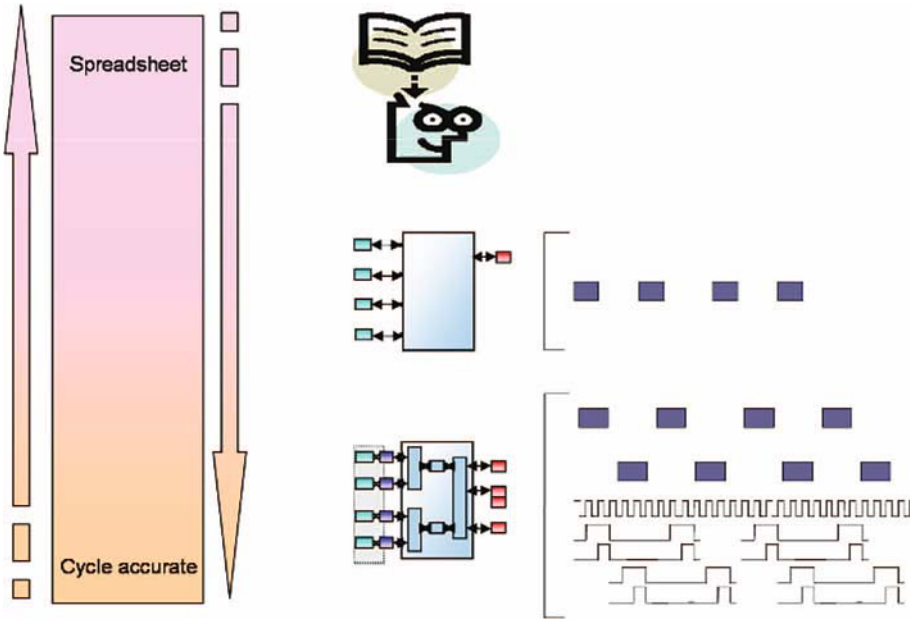


Figure 6-12. Accuracy Trade-off

6. EXAMPLE OF USING IPTG METHODOLOGY

This section provides a practical example of the SoC architectural analysis through the IPTG approach. The same methodology is used across several families of SoCs. One of this chip is the STB7100, a High Definition Low Bit-Rate Video Decoder, developed by STMicroelectronics.

6.1 Functional View of STB7100[3]

The STB7100 is the world's first single-chip Set Top Box (STB) solution supporting the High Definition H.264/AVC and VC1 specifications, which are poised to enable the next generation of high quality consumer video

[3] The information in this section is extracted from the website of STMicroelectronics at http://www.st.com.

systems and broadcast services. It also supports the H.264/AVC advanced video decoding standard, Microsoft's VC1 standard and high definition MPEG-2. The STB7100 can be used in:

- cable, satellite, terrestrial and IP set-top box;
- DVD in consumer and automotive.

The STB7100 demultiplexes, decrypts, decodes and outputs HD and SD video streams with associated multi-channel audio. A dual display compositor provides mixing of graphics and video with independent composition for TV/monitor and VCR outputs. SATA and USB interfaces are provided to enable low-cost connectivity to hard-disk drives and low-cost system expansion. The functionalities of STB7100 are summarized in Figure 6-13.

The STB7100 can simultaneously decode multiple HD streams and output the resultant video to two television sets, or display picture-in-picture. Its CPU core is a high-performance 300MHz ST40, ST's 32-bit RISC family based on the SuperH™ architecture and widely used across digital consumer applications. It supports all of the current STB operating systems and middleware, with power to spare for software enhancements in the future.

The new device is based on an innovative video decoding architecture which combines hardware and software techniques to allow systems to be upgraded in the field to support new standards as they become available. For Digital Video Recorder (DVR) applications it features embedded peripheral interfaces - including serial, ATA and USB 2.0 - to allow external devices to be added easily to an STB or DVD player, either during manufacture or by the viewer, in order to provide additional functionality. Viewers increasingly use digital video recording for program time shifting. Other peripherals that could be connected to a set-top box through the USB interface include digital cameras, printers, and memory cards.

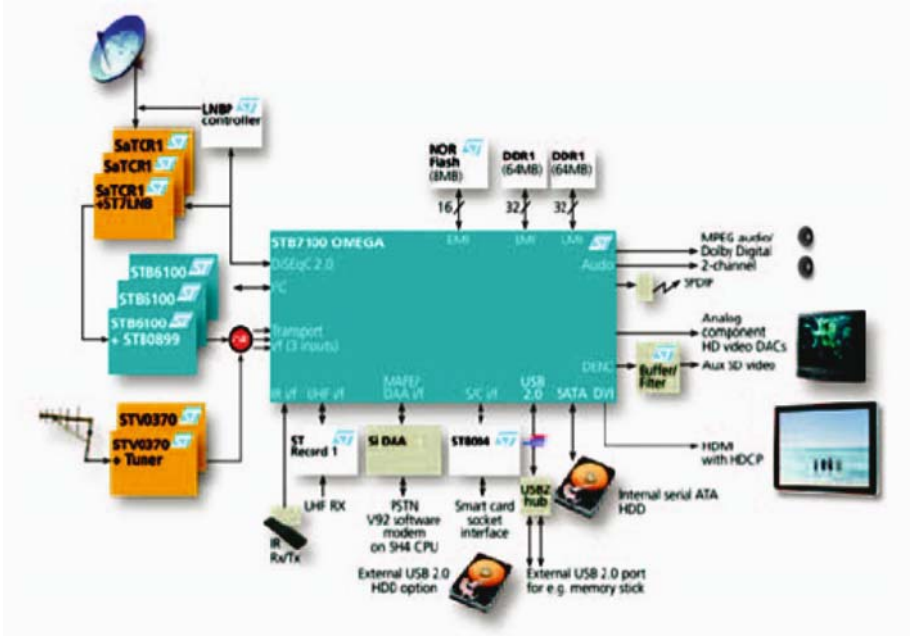


Figure 6-13. Functionalities of STB7100

6.2 Architecture Analysis of STB7100

As stated above, a typical Set Top Box or DVD SoC is built using:

- several CPUs: a host and several dedicated cores for audio and video processing;
- hardware IPs: such as hardware assists, graphic processors, and peripheral interface controllers, each of them behaving as an initiator and/or a target on the routing system;
- One or several DDR memory controllers called LMI hereafter.

The conception of the communication model for such a complex SoC starts with a spreadsheet-based analysis. The different working modes of the system are listed and characterized. For each scenario, the requirements of all the initiators are detailed then summed up in order to choose the memory buffers locations and to size the memory interfaces. Then, in order to design and validate in advance the interconnection between on-chip IPs and to configure the traffic of the IPs, the whole system is modeled in SystemC at transaction level.

This platform consists in several tens of IPTGs describing the behavior of the CPUs and the IPs' initiator side. The communication model is based on simple switches and links available both in BCA and RTL. The MEM is

made of the LMIs and of basic memories modeling the IPs' target side. An example of such platforms is shown in Figure 6-14.

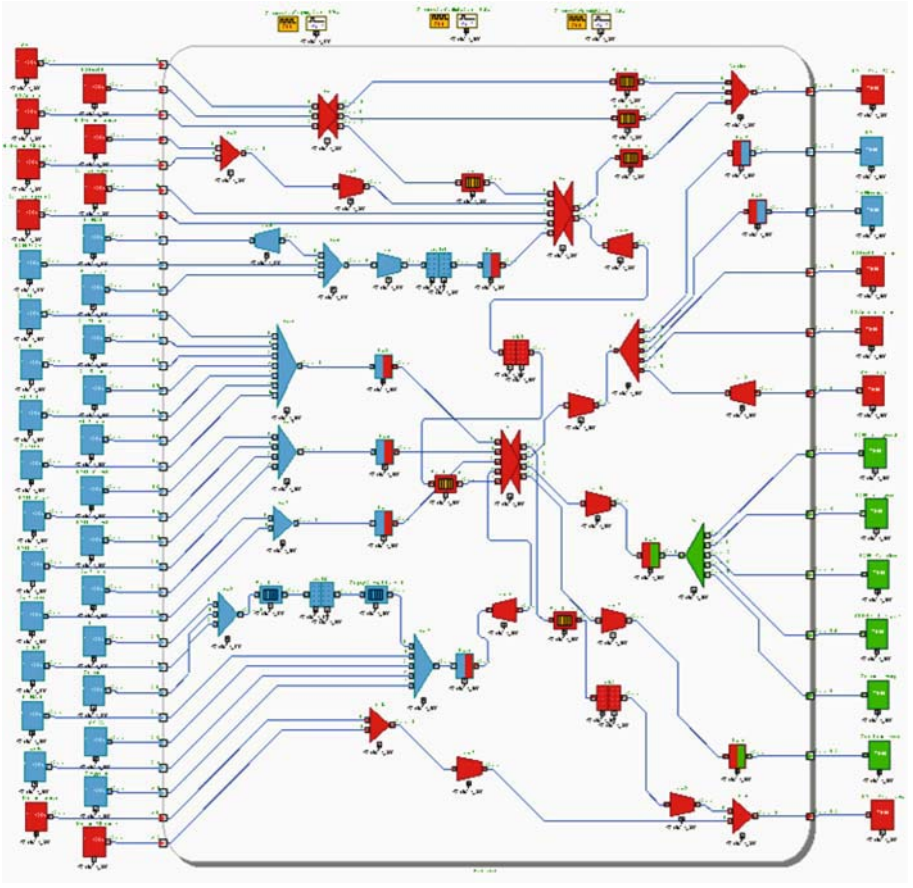


Figure 6-14. Schema of a Communication Model

The IPTGs are modeled at TLM level and the COM and basic memories at BCA level. The RTL model of the LMIs is used to obtain cycle accurate behavior for these key components, which are the bottleneck of the system. A TLM-to-BCA translator is then associated to each IPTG, and a BCA-to-RTL translator to each LMI port.

The CPUs are, in a second step, replaced by their associated ISS. The CPUs are configured in traffic modeling mode. The host is in charge of the communication model and the memory controllers.

The other IPTGs are set in IP modeling mode. An important feature of the IPTG is that it enables to model the dependencies between plugs of the

same IP, thanks to the synchronization mechanism. Consider an IP that works from memory to memory, the pipeline is stopped whenever a write plug is full or a read plug is empty.

The analysis of a simulation performed on such a platform is straightforward because the IPTG FIFO level is monitored. Any over/underflow in a real-time IP is flagged and the percentage of pipeline stopped time in decoder IPs is reported.

For each working mode of the IPs, a set of IPTG configuration files is defined to model the worst case in terms of bandwidth consumption. Then, scenarios of the spreadsheet analysis are reproduced, gathering the IPTG configuration files of all the IPS, and a simulation is run for a portion of an image.

When the performances are not met, the SysProbe transaction debugger allows to observe directly internal nodes of the communication model and to analyze the root cause of the performance drop off. A side advantage of this approach is that the verification of the communication model's RTL can be done in the SystemC environment, using meaningful scenarios.

7. CONCLUSION

The methodology proposed in this chapter enables outlining a plug-and-play architecture environment based on the platform assembly that requires no new language learning.

The IPTG approach is put forward as a complementary solution to the conventional architecture analysis on paper, which takes into account the inadequacy of simulating a real system's scenario at an accurate level. By adopting this method, IP designers create directly the IP configuration file that will be reused across various projects with high flexibility to update various products. In brief, the very rewarding result of this methodology is an analysis environment that is powerful yet easy-to-maintain.

REFERENCES

[1] OCP Specification, Available on the OCP Website: http://www.ocpip.org

[2] STBus Functional Specifications, Available on STMicroelectronics Public Support Website: http://www.stmcu.com/inchtml-pages-STBus_intro.html, April 2003.

[3] ARM AMBA 3.0 Specification, Available on ARM Website: http://www.arm.com

[4] Cadence Incisive (SimVision), Information available on Cadence website: http///www.cadence.com

[5] Novas Verdi, Information available on Novas website: http://www.novas.com

[6] Cadence TxE, Information available on Cadence website: http//www.cadence.com

[7] Tenison Vtoc, Information available on Tenison website: http://www.tenison.com

[8] Mentor Graphics H2C, Information available on Mentor Graphics website: http://www.mentor.com