Functional verification of a frequency-programmable switch chip with asynchronous clock sections

B. Hoppe
B. Arthur-Mensah
E. W. Chencinski
S. Joseph
H. Kumar
J. F. Silverio

An integral part of the IBM eServer™ z990 I/O subsystem is the self-timed interface (STI) switch chip. The STI switch is an application-specific integrated circuit (ASIC) designed to provide high I/O connectivity and high bandwidth within the system. The complexity of the functional verification of the STI switch chip is inherent in the implementation of seventeen logical clock domains and the support of six different STI interfaces with programmable frequencies. The logic within these clock domains is connected via asynchronous interfaces. This paper describes the methodology to verify the functionality of the switch chip with various STIs by introducing a combination of verification techniques. This involves random biased stimulus generation, automated result prediction checking, and the use of cycle simulation to stress the logical design. The cycle simulation required new techniques to model equivalent behavior in order to verify the correct integration of nondigital components on the chip. Advanced methods were implemented to ensure correctness of the frequency-dependent design units and functionality across the asynchronous interfaces. A single verification environment was developed, providing the flexibility to seamlessly support the different levels of design abstraction and uncover the design errors at the appropriate level.

Introduction

The z990 eServer* I/O subsystem comprises an architecture providing I/O connectivity, networking connectivity, and intersystems connectivity. Since the introduction of the z900 system [1], significant improvements have been implemented in the I/O subsystem architecture. The higher I/O connectivity that is supplied breaks the limit of 256 channels, providing faster and more efficient transmission between systems and to the networking attachments. As depicted in **Figure 1**, an integral part of the new and improved z990 I/O subsystem is the STI switch chip.

The STI switch chip provides high-speed connections between the memory bus adapter (MBA) and the I/O attachments [2] and to other systems within the Parallel Sysplex* [3]. The switch chip is also called the multiplexor/demultiplexor chip. The connection to the MBA can be via the enhanced self-timed interface (eSTI), which is configured to operate at 2 GB/s, or via the multispeed self-timed interface (mSTI). The mSTI can be configured to operate at a data rate of 333, 500, or 1000 MB/s. This configuration dictates whether the switch chip operates as a level-one or as a cascaded level-two device. As a level-two device, the switch chip connects to a

0018-8646/04/$5.00 © 2004 IBM

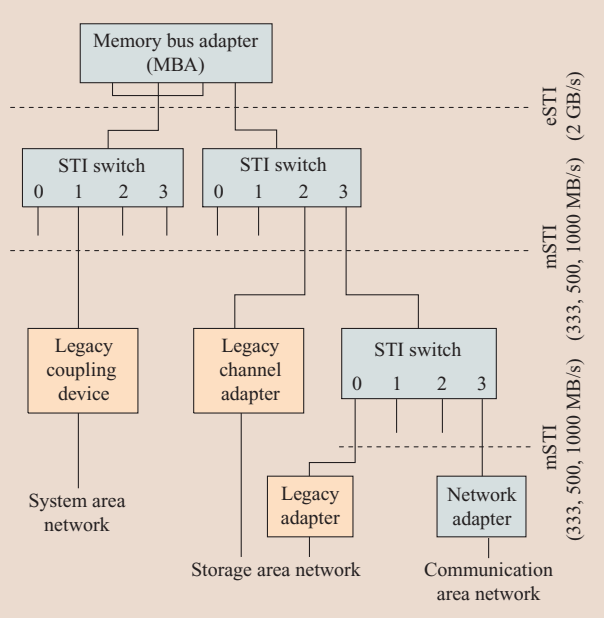

Figure 1

IBM z990 I/O subsystem overview.

level-one STI switch via the mSTI link. This configuration provides higher connectivity. Four mSTI ports are implemented to support the downstream I/O connections.

The mSTI and eSTI support data is transferred via electrical cables in a serial fashion in which the clock is transmitted with the data across the physical link [4]. To support this interface, the switch chip integrates a physical adaptation layer and a logical adaptation layer for each of the interfaces. The physical adaptation layer serializes and deserializes the transfers of data, while the logical adaptation layer assembles the information packets and supports a parallel interface with the chip host logic.

The STI links and the supporting adaptation layers can be programmed to operate at multiple frequencies. In addition, the host logic integrates several clock domain units. In all, the switch chip integrates a total of seventeen different clock domains. As a result, the interfaces between these domains and link boundaries are asynchronous over a wide frequency range. Information-packet-forwarding logic algorithms were implemented to ensure arbitration fairness between incoming and outgoing packet flow and to optimize this flow as a function of data length and clock frequency.

To ensure a high-quality design and a fast time-to-market, the majority of the ASIC logic is verified using a functional cycle simulation environment. With 9.6 million transistors integrated in the switch chip, the objective was to utilize a high-performance cycle simulator [5] as much as possible where appropriate to maintain high performance across the various levels of simulation. The functional verification methodology [6, 7] implements a random biased approach to stimulate the designs with automated checking using cycle simulation.

Some of the logic units contain nondigital designs integrating analog components, differential interfaces, and subcycle wire delays. In this case, an event simulator is used to ensure the functional correctness of these components. Also, the approach is to verify these components in a standalone-unit environment so that these units need not be verified exhaustively at the chip level.

Register-transfer-level (RTL) design models are generated to verify the function within the chip as well as the internal and external interface protocols. This includes verifying the interactions between the new designs and the integrated intellectual property. These RTL models are referred to as one-cycle simulation models. In turn, full chip gate-level design models are created to verify the functionality and integration of error recovery, clocking structures, and test logic. Some of the error-recovery design features are verified using the partial RTL model.

Very often during the design development cycle, IP components, nondigital components, and the rest of the chip logic are designed in parallel. For example, this was the case with the development of the physical and logical adaptation layers within the switch. This creates unique challenges to identify the necessary verification environments, the appropriate methodology, and the verification tasks to be covered for each environment. Also, emphasis must be placed on reusing verification components for stimuli generation and prediction checking across the various verification levels.

This paper identifies the verification tasks for ASIC chips. The overall verification methodology used to verify the STI switch chip is described, and the various models and levels of verification are highlighted. A special modeling of non-cycle-simulation-compliant logic is addressed. A detailed description of the verification environments is also provided. Special consideration is given to a new method developed to verify chip designs with multiple asynchronous clock domains using a stress-testing mechanism in a simulation cycle environment. The verification of the asynchronous interfaces and the handling of the clocks within a cycle simulator are discussed.

ASIC verification tasks

The verification tasks of an ASIC design can be divided into the following subcategories: mainline, error detection, reporting and recovery, and pervasive verification tasks.

Special focus is required on verifying the asynchronous interfaces and the design features being influenced by the frequency variation.

Mainline verification tasks

In transaction-based I/O ASIC chips, mainline verification is commonly referred to as information packet flow. The objective is for the interface to provide random biased stimulus of defined information packets, posted and nonposted, at both external and internal interfaces, and predict the appropriate outcome and responses as a result of the generated stimulus, thereby ensuring the data integrity and correct behavior of internal registers. Correct packet conversion, packet routing, packet destinations, and data payload lengths have to be checked. In addition, during this verification task, interface protocol checking must be performed according to the chip interface specifications. This incorporates packet transmission, packet acknowledgment, packet timeouts, and correct handshaking. It incorporates performance verification to prove the predicted latency and the effective usage of existing resources. During mainline verification, deadlock conditions can easily be detected with timeout checking mechanisms. When a timeout occurs, the chip function has reached a stalled state, or deadlock, and packet flow stops. Live-lock conditions, which occur when internal resources are not freed up, must be monitored in order to detect design bugs.

Error detection, reporting, and recovery tasks

To verify error detection, reporting, and recovery features of the design, an error injection mechanism must be in place to force error scenarios. The objective is to ensure that the error is recorded correctly. This is accomplished by introducing internal monitoring mechanisms. Subsequently, error reporting of the error conditions, such as interrupts or checkstops, is also verified. Assertion checking is used to ensure that the error conditions are properly reported. Depending on the masking implemented for each error condition, an error can be classified as a soft error or as a hard error. The simulation environment integrates a structure to randomly set the error-masking bits and verify that each error condition is classified properly. A detailed description of the error injection and error recovery functionality and simulation environment to verify these tasks is outside the scope of this paper.

Pervasive verification tasks

Overall, nonfunctional tasks fall into this category, including the power-on-reset (POR) sequence, logic built-in self-test (LBIST), clock generation and distribution, register accesses with the system clocks running, and physical link initialization.

The LBIST verification task involves connection checking between the pseudorandom pattern generator (PRPG) and the multiple input shift register (MISR) [8]. In addition, LBIST signatures generated from function simulation test cases must be compared with signatures generated in the testbench test-pattern generation model.

The physical link interface initialization and calibration sequence must be followed prior to ASIC link operation. The simulation environment checks that the proper initialization sequence is followed and that the required bit pattern is used to reach the link operational state. LBIST and physical link initialization tasks are integrated in overall ASIC POR sequence verification tasks.

In any ASIC device, the phase-locked loop (PLL) programmability is verified, and the locking functionality is checked to ensure that accurate clocks are generated to the various partitions of the chip to reset its logic. Owing to the multiple clock sections and numerous configurations, an important task included in the overall pervasive verification realm is the clock generation and distribution verification.

Another aspect of pervasive verification is the ability to access the design internal registers while the system clocks are running. Basically, various checks are implemented to verify the read and write accessibility of available internal registers via the specified service interface. This is done while other mainline information packet flow is being exercised. In the process of checking register accessibility, the actual register functionality is verified to ensure that the proper operation occurs as a result of setting specific register bit values.

Asynchronous interface verification tasks

Because of the increasing number of configurations of the I/O chips and the external interfaces running in a different, asynchronous clock section, the issue of verifying the correct function of asynchronous boundaries becomes a more complex issue. The first problem is to identify the asynchronous boundaries in a specific design. If special design rules are applied, a static check can be applied that the design actually follows the rules of the asynchronous boundary. This is possible only if the design is newly developed and is able to follow the given rules. In most cases either a legacy design or an external intellectual property is used as the implementation on one side of an interface that does not follow those rules. This makes the static verification of the asynchronous boundary impossible, and requires functional verification.

Design abstraction to perform verification tasks

Typically, separate verification environments and tests are developed to implement the various verification tasks. These environments include random biased test-case generation for mainline verification tasks, deterministic

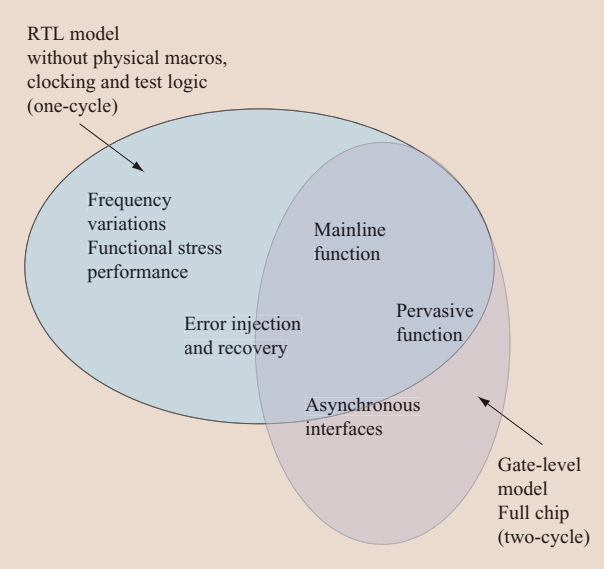

Figure 2

Model requirements vs. verification tasks.

tests for error injection, recovery simulation, and pervasive function verification tasks. This typically limits the simultaneous execution of the different tasks, and it also means that the specific tasks can be performed only on specific abstraction levels of the design. In most cases, the mainline functional stress tests are performed on an RTL model to efficiently eliminate the functional bugs. Gate-level design sources are previously verified or are integrated in a unit environment as part of the overall verification effort. Pervasive functions, such as the power-on-reset sequence, initialization of the chip via the service interface, and changing the frequency of the clocks, must be verified on a gate-level model, since it requires all clocking, scan rings, and test logic connections, which are introduced as part of the design-for-test process [9]. If the environments are disjunctive, the sequence to initialize the chip via the service interface and perform functional stress tests after the start of the clocks does not validate the initialization function.

Figure 2 shows the optimum abstract model levels for performing the specified verification tasks. To fulfill those requirements, maximize the efficiency, minimize the turnaround time of hardware bugs, and structure the verification environment along with the hardware design development while providing a complete verification, a multistaged verification methodology has been deployed to verify the switch chip. This methodology supports partial chip RTL as well as full gate-level-cycle-based simulation environments.

Using the same environment for all verification stages and ensuring that the verification efforts keep pace with the design development offers the flexibility to reassign the verification model level to the appropriate verification task. Also, this provides the capability to stimulate the service interface while executing regular mainline functions and in the process exercise multiple logic functions simultaneously.

Verification methodology

A very efficient methodology is required to minimize the turnaround time of design bugs, which is a tradeoff between the amount of resources and development time. The complexity of the chips requires focused unit verification for certain partitions, leaving the full chip to be tested at the higher levels. It is desirable to verify as much as possible on an RTL using abstract cycle-simulation models. In addition to the unit- and chip-verification efforts, the integration of the entire chip was tested on a subsystem and system level. Legacy chips or newly developed chips were connected to the STI switch executing tests in order to uncover potential specification problems in the implementation of the interface protocols.

Unit simulation

The function of the physical macros is to transmit data over a cable with a high frequency. The functional implementation of the physical macros is on the gate level and has a very well-defined interface. The alignment of data within a clock cycle and multiplexing data controlled by wire delays represent logic which cannot be verified using a cycle simulator, since the abstraction of the model leads to elimination of all delays other than clocked latches. Therefore, the physical macros supporting the mSTI and eSTI protocols were verified in a standalone testbench using event simulation with the correct timing information. Slow, nominal, or fast timing information of the appropriate ASIC technology was loaded into the event simulator, and link initialization and functional operations were performed to verify the proper behavior of the design. Also, a unit-verification environment was created for the logical eSTI and mSTI macros to ensure appropriate initial design quality.

Chip-level modeling

Cycle simulation does not allow full timing verification. However, event simulation with the full signal delay information usually occurs late in the verification process, and with the large designs, achieving full timing simulation is impractical because of performance degradation. This is overcome by using cycle simulation coupled with the use of static timing analysis and checking tools such as EinsTimer* [10], ensuring correct function.

Typical I/O ASIC designs have multiple clock partitions that are mostly asynchronous to one another; hence, special modeling must be applied to reflect an accurate relationship between the different clock sections and their numbers of simulator cycles per clock cycle of the respective domain. Algorithms must be applied to evaluate logic only in clock sections in which the clock edges actually occur.

Chip model stages

The logic design used in simulation was modeled under two main categories—one-cycle and two-cycle models. The following section describes the model stages used for the switch chip.

C++ models replaced the hardware design of the physical macros in the one-cycle logic design model. These models are referred to as mSTI and eSTI behaviorals in **Figure 3**. This allowed the simulation to mimic the actual mSTI link initialization sequence and information packet formatting, thus resulting in a shortened round-trip time for data transfer. Test cases that required the rigorous examination of packet forwarding between the STI logical macro and host logic were conducted under this abridged version of the switch design. A model implementing the behavior of the universal service interface (IF behavioral) is used to exercise base maintenance functions. Each clock domain is fed by a simulation-only programmable oscillator macro to allow frequency variation.

The two-cycle model, which is a full representation of the design of the switch gained from the gate-level netlist, underwent variations to accommodate pervasive function testing. This full chip gate-level model (also known as the two-cycle model), whose characteristics are augmented in subsequent versions, uses the service interface behavioral to set up the PLL, LBIST, logic reset, flush reset, and latch initialization logic. For a complete analysis of end-to-end data transfer, additional mainline and recovery testing was performed on the full chip gate-level model shown in **Figure 4**. Each instance on an analog PLL was replaced by simulation-only logic (PLL behavioral) which was the equivalent of a real PLL in a cycle simulator. The content of this logic existed as described in [11]. It was used to verify the surrounding controls of the PLL to program the clock delay, pulse width, and frequency of the PLL.

To verify the correct function of the pervasive function required a multivalue representation of the logic, which led to the development of a multistate full chip gate-level cycle-simulation model. Apart from a binary zero or one, a value "X" can be assigned to every net to indicate *not initialized* or *don't care*. This first variant deployed an unknown initialization sequence on the design latches for additional testing of flush reset. Unlike the full chip gate-level model, which required a 10% duty cycle for clocks, a second variant was developed to exhibit a 50% duty cycle for its clocks. The latter model was used for LBIST and signature matching verification. Asynchronous boundary testing utilized a third variant, shown in **Figure 5**, which incorporated additional simulation-specific random delay

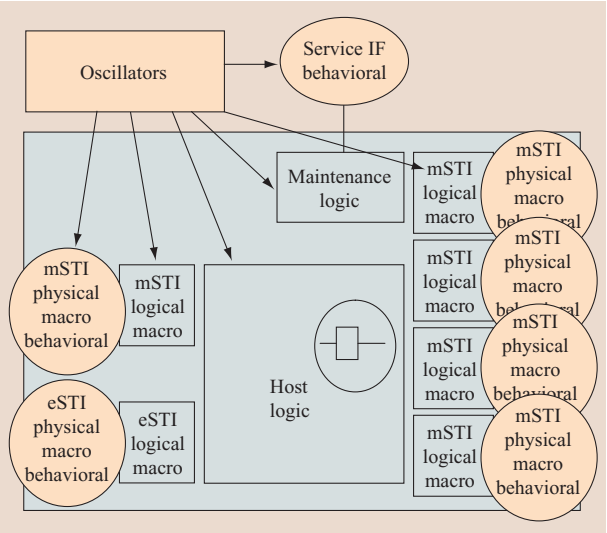

Figure 3

Core RTL model with explicit clocking of each clock domain.

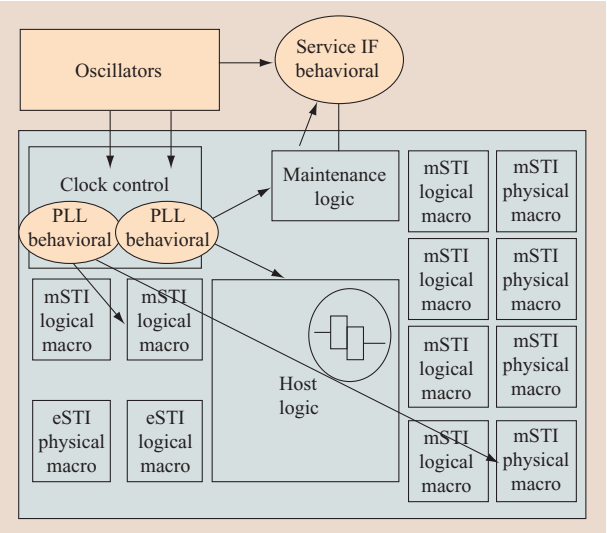

Figure 4

Full chip gate-level model.

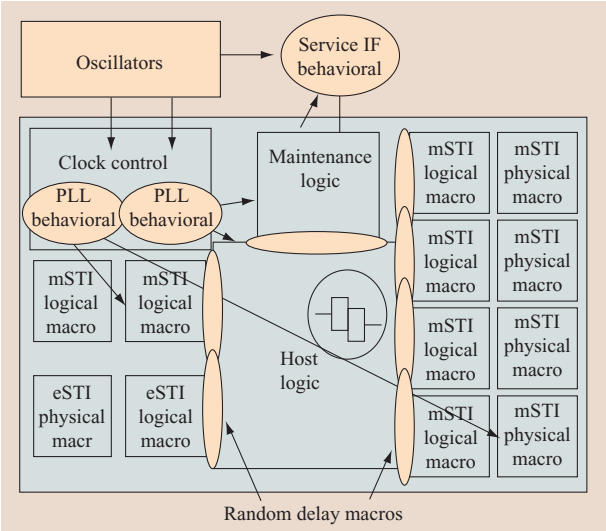

Figure 5

Full RTL chip model with modeling of asynchronous interfaces.

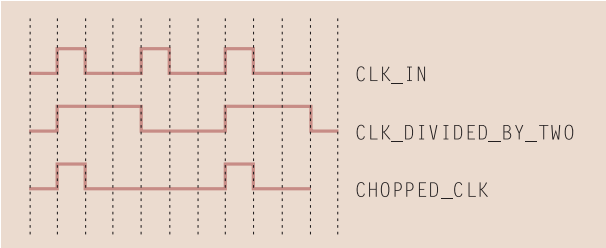

Figure 6

Fifty percent clock duty cycle altered and divided by 2.

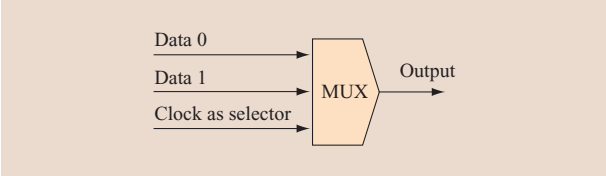

Figure 7

Clock selecting the input of a multiplexor.

macros to provide simulation with programmable delays at critical asynchronous paths. This third model was periodically modified as the identification of asynchronous crossings within the design continued to evolve.

Modeling accurate behavior of non-cycle-simulation-compliant components for cycle simulation

To enable a design for functional verification, nondigital design components, which do not naturally behave correctly in a cycle simulator, must be modified. For example, physical layers to drive chip interfaces, wire delays, and specific behaviors of the drivers on the chip boundary require timing information to reflect accurate logic behavior. This required timing information can, however, be modeled and incorporated in a cycle-simulation model to provide equivalent behavior. The gain in simulation performance justifies the additional effort to model this behavior. Special simulation-only logic driven by the simulator clock, instantiated within the chip, can provide behavior equivalent to a certain level in a cycle simulator, which can be used to ensure functional correctness within its context. However, race conditions, clock jitter, temperature drift, and skew cannot be reflected in a cycle simulator to ensure accurate functionality.

The physical macros were incorporated in the two-cycle model generated from the gate-level netlist of the chip. Physical macros used unique latches and state machines, which are based on both the rising edge and falling edge of the cycle. The complex clock and data paths of the macro malfunction when they are clocked with 50% duty cycle clock inputs. In cycle simulation, the input value of the slave latch is captured if the clock input is accordingly high. Since the switch chip deals with multiple asynchronous clock domains, a 50% duty cycle clock, which could be high for multiple simulation cycles, can cause incorrect values to be propagated through the latch models; this happens if data is captured from an asynchronous boundary which is running at a different clock frequency. This phenomenon causes two different data sets to be launched, while the capturing latch processes both data sets in one clock cycle, creating false simulation results in the model used. Hence, the edge-triggered latches had to be altered into a different form in order to be correctly modeled in a cycle simulator. An example is illustrated in **Figure 6**, where the upper waveform shows the modeling of clocks for edge-triggered latches as an input to the clock divider, and the middle waveform the output of the clock divider, which can cause false simulation results. The waveform at the bottom represents the result of the modified clock. Furthermore, logic was added for multiplexors, on which clock inputs were used as data signals. **Figure 7** shows an example in which a clock was used to output two sets of data at twice the speed without defining another clock; this is used as a two-to-one serial converter. For proper behavior, this logic required alteration in the simulation model, since clocks on the data inputs cause a delay of one simulation cycle the output. The delay of one simulation cycle causes the logic path to capture inaccurate data from the asynchronous

boundaries. To reach functionality, the clock inputs to these latches were ensured to be 50% duty cycle and aligned with altered clocks for latches represented in Figure 6. The unique clocking structure of the physical macro required both 50% duty cycle clocks used as data inputs and altered clocks for conventional clock inputs.

Chip verification methodology and tools

This environment utilizes the C++ language and object-oriented constructs to implement the expected behavioral. It also takes advantage of existing simulation libraries developed in C++ to support biased random stimulus generation using parameter tables and simulation run control. The C++ simulation environment interacts with the cycle simulator, utilizing the function sets provided by the simulation application interface (SimAPI) [12] to control the simulation and to access facilities within the design. For each of the major design interfaces, a combination of a stimulus generator, interface driver, and interface monitor represents a behavioral. Internal facilities are monitored to determine the internal state of the model and to check the content of internal registers. Boolean equivalence checking is done on the various stages and abstraction levels of the design using Verity [13].

Functional coverage

Coverage measurements were obtained with the implementation of coverage models that collected occurrences of interesting events, including both successful events and specific failure events. Two types of coverage models were implemented. One type focused on detecting and recording events internal to functional units within each ASIC. The other type essentially monitored the ASIC external interfaces for all specified commands and protocols. For both coverage model types, the collection of events was processed to provide a picture of the overall functional verification coverage. The analysis from the coverage reports provided feedback for test-case generation and coverage model development stages. This is an iterative process carried out throughout the ASIC development.

Functional verification environment

The environment consists of packet generators, drivers, monitors, result prediction, and checking to verify mainline, recovery, and error injection. **Figure 8** shows the code structure. The packet generator generates a variety of information packets and link control words (LCWs), destined for devices both downstream and upstream from the switch. This also includes configuration information packets for internal facilities of the switch, and broadcast messages containing device status information such as interrupts. The packet generator can operate in a deterministic or random mode. Deterministic mode allows

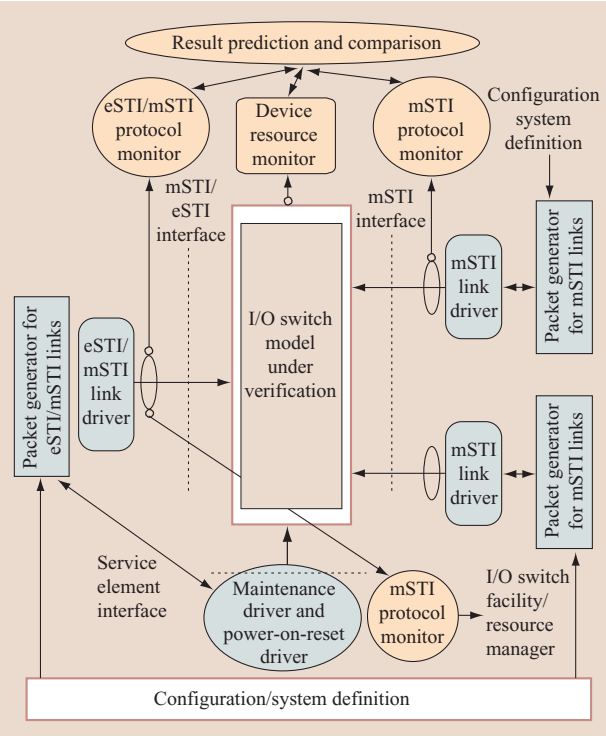

Figure 8

Switch chip verification code structure.

specific definition of stimuli, while random mode utilizes probability tables to bias random values in information packets and link control messages. The elements generated by these tables include information packet lengths, source/destination ports, different types of read/write commands, and link control messages indicating the link state. Error injection is also randomly exercised by using information from similar probability tables to generate erroneous stimuli to the chip and also to misinterpret information transmitted from the chip. This includes errors such as corrupt bits on the data link and faults in the decoding/encoding process of bits. The eSTI/mSTI link drivers stimulate the external mSTI and eSTI data links with the information generated by the packet generator based on the STI protocol, whereas the maintenance and power-on-reset drivers respectively stimulate the service wire interface and the power-on-reset state machine to allow access to the internal facilities of the switch that are required by the service element. On all mSTI and eSTI links, protocol monitors record incoming and outgoing packets and perform protocol checking and report transactions. A device resource monitor is also present to watch and record any internal changes within the switch chip.

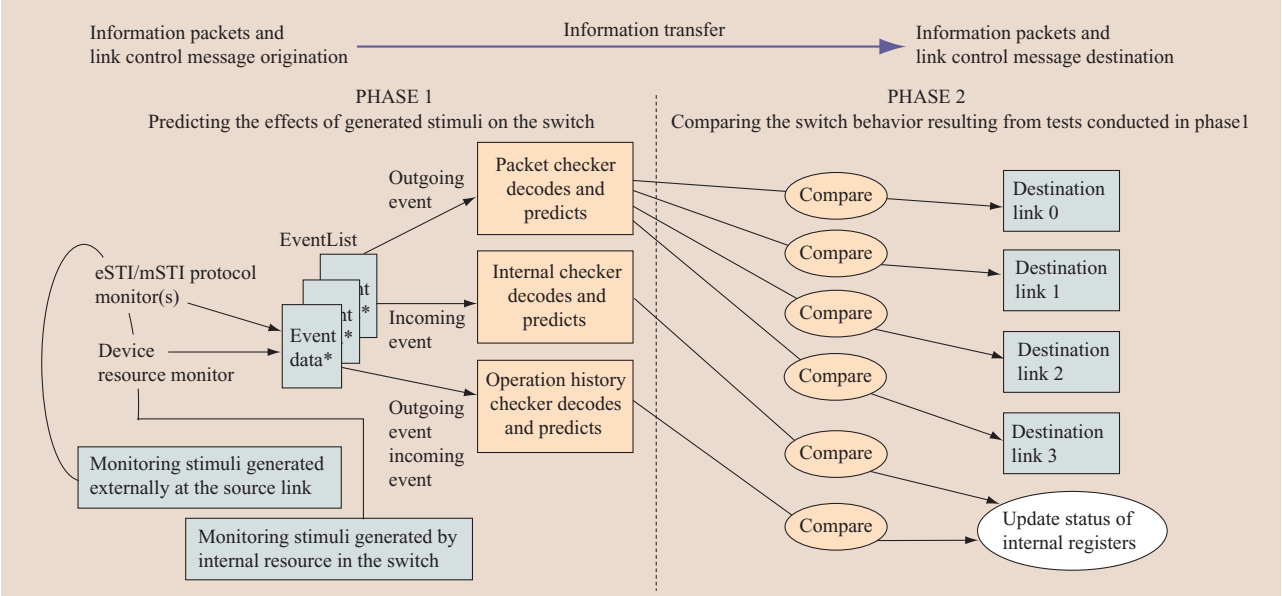

Figure 9

Checking structure and result comparison in verification code for the switch chip.

The checking code illustrated in **Figure 9** predicts and compares all actions performed on the external interfaces and within the chip. In the checking protocol, outgoing packets and packets generated as a result of internal chip changes are referred to as outgoing events. Packets targeted to a resource within the chip are referred to as incoming events. There are three main types of checking code: 1) the packet checker, which decodes, predicts, and compares all packets targeted to each of the mSTI and eSTI ports; 2) the internal checker, responsible for decoding, predicting, and comparing all packets targeted to internal chip facilities; 3) the operation history register checker, also in charge of decoding, predicting, and comparing all traffic (information packets and general link control messages) that is recorded within the switch. Reports from the monitors and checkers are used by the I/O switch facility/resource manager to maintain the device status database and manage event processing.

Verification of multiple asynchronous clock frequency domains and interfaces

Clock handling and stress testing across multiple asynchronous frequency interfaces using a random cycle simulation environment

As shown in **Figure 10**, the STI switch chip supports a layered design structure which integrates multiple clock frequency domains. The I/O ports and link adaptation layers (LALs) are configured to operate at various frequencies. The host adaptation layers (HALs) operate at slower frequencies relative to the I/O port. The host logic (HL) contains multiple clock domains as well.

The clock logic within the ASIC chips is verified more accurately with the use of a full chip gate-level model, which makes it possible to introduce accurate subcycle delays in the logic paths. However, the overall simulation performance associated with these large-size models considerably limits the number of simulation runs. With the proper modeling of analog and timing-dependent components, the use of simulation-cycle-based models can improve the overall simulation performance considerably. In this case, an abstraction of the design components is developed and then integrated with the rest of the ASIC design to generate an RTL model.

In order to handle the verification of the various clock domains, the clock behavior is driven from multiple sources. The source of the clocks differs, whether a single-cycle simulation environment or a two-cycle simulation model is being implemented. In a one-cycle model, the input to the oscillator macro is toggled. The intent is to verify the clock paths and the clock gating logic within the chip. The clocking structure logic integrates an on-product clock generation (OPCG) logic, which is verified as well. When a two-cycle simulation environment is used, the clocks are driven by toggling the outputs of the phase-locked-loop (PLL) logic. This approach provides simulation coverage of the overall clocking structure on the chip, including the clock splitters and clock dividers.

In this case, the actual clock logic is used to drive the various clock frequency domains.

Checking mechanisms are implemented as part of the random environment to ensure that all domains are clocked at the appropriate frequency so that the overall clock logic is structured properly.

In a simulation cycle environment, the multiple frequency clock domains are represented as simulation cycles per system clock cycle relationships. With the use of these simulation cycle representations, frequency stress testing can be performed using a random cycle environment. This is the basis for the method described in [14]. The method involves the calculation of multiple frequency simulation cycles using an automated greatest common factor (GCF) method; the asynchronous interface stress testing is then performed by skewing the cycle relationships using a random cycle simulation environment.

As shown in **Figure 11**, a mathematical approach is used to generate the default discrete simulation cycle (sim cyc) values in order to represent the different frequency

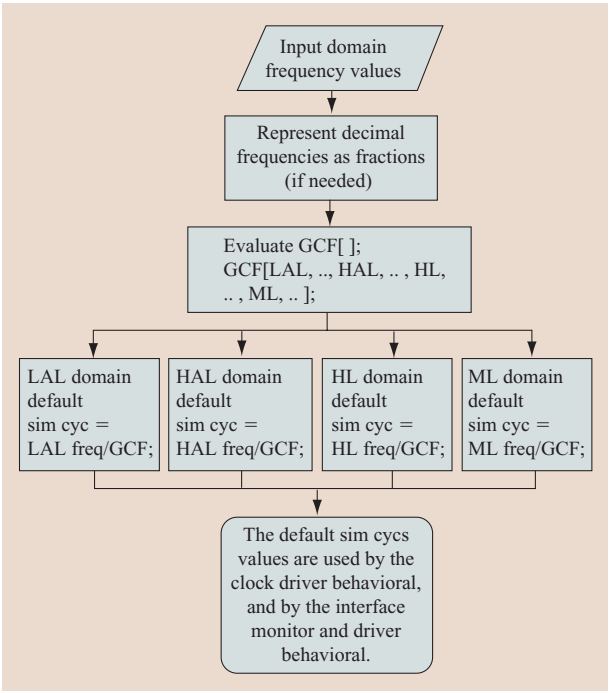

Figure 11

Flowchart representing frequency default values as simulation cycles.

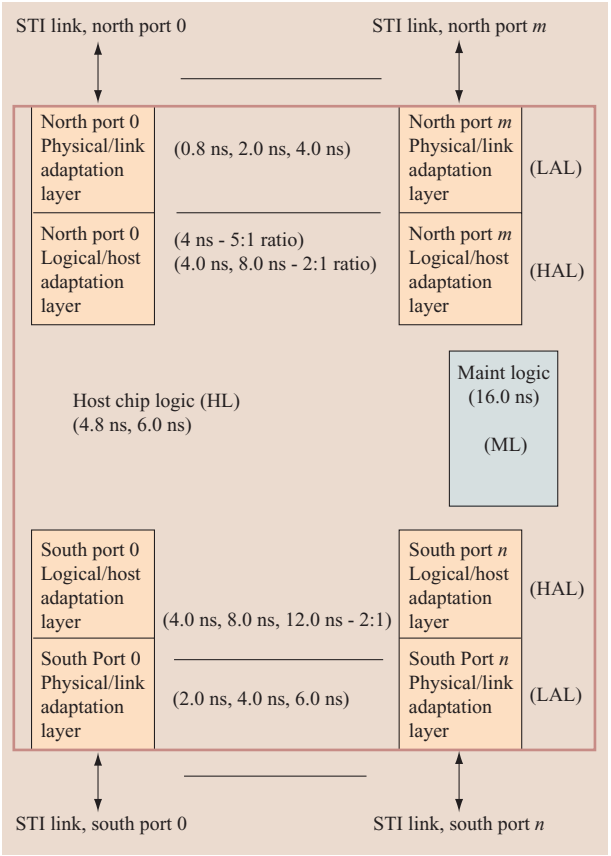

Figure 10

Clock domains of the STI switch chip.

domains. First, a unit of time must be identified so that one simulation cycle can be equated to this unit of time. The unit of time is determined as the GCF between the various frequency domain values. This can be represented as follows:

$$\mathrm{GCF}[\mathrm{NorthLAL}(0), \mathrm{NorthHAL}(0), \ldots, \mathrm{HL}, \mathrm{ML}, \ldots, \mathrm{NorthLAL}(m), \mathrm{NorthHAL}(m), \mathrm{SouthLAL}(0), \mathrm{SouthHAL}(0), \ldots, \mathrm{SouthLAL}(n), \mathrm{SouthHAL}(n)],$$

where ML stands for maintenance logic.

Note that for clock domains with decimal frequencies, the GCF is calculated on the basis of all numerator values, with the decimal values represented as fractions (i.e., 0.8 ns = 8/10 ns; factoring the numerator, we have 1, 2, 4, 8). After the GCF between all numerators is determined, it is divided by 10 to obtain the value of the unit of time. The default simulation cycles for each frequency domain can then be calculated using the formula [*domain simulation cycles* = (*domain frequency*/*GCF*)]. The GCF value ensures that the most efficient unit of time (represented in simulation cycles) is determined. This guarantees that the appropriate default simulation cycle values are selected to represent the different clock domains and to provide the most efficient model performance.

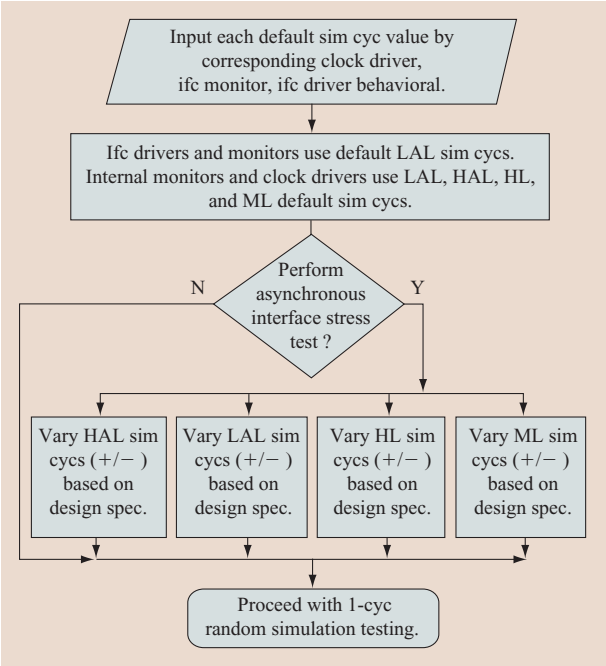

Figure 12

Flowchart representing clocking for environment for asynchronous stress testing. (ifc: interface.)

The behavior of the clock drivers, internal monitor, and interface monitor and driver behaviorals is depicted in the flowchart diagram of **Figure 12**. The default simulation cycle relationships are input by each driver or monitor behavioral corresponding to each frequency domain. The interface driver and monitor behaviorals make use of the LAL simulation cycle values. This is true for the north and south interfaces. The clock drivers and internal monitor behaviorals use the default LAL, HAL, HL, and ML simulation cycle values.

If asynchronous stress testing is not required, the one-cycle random simulation can proceed, with the default simulation cycle values representing the various frequency domains. On the other hand, if asynchronous stress testing is required, the representation of simulation cycles is varied for each domain. This variation is based on the design specifications, and it can be varied a certain percentage above and below the default simulation cycle value. The minimum variation is one simulation cycle. The simulation cycle variation is selected at random on a per-test-case basis. Once the new stress test values are set, the one-cycle random environment simulation can proceed.

The approach of skewing the starting of the clocks in a specific domain relative to the clocks in another domain uncovers potential buffer underrun conditions (i.e., data is read out of the buffer before it is written). Extreme clock boundary scenarios can be tested as well. This also introduces a checking mechanism to ensure that the rising edge of one clock domain can be captured by the asynchronous clock of another domain.

Asynchronous interface functional verification

The switch and the bridge chip contain many thousands of signals that are transmitted from latches that are clocked with one frequency, are potentially combined through logic gates with other signals, and are later captured by latches that are clocked at a different frequency with no guaranteed phase relationship to the transmitting clock. These "asynchronous interface crossings" require extreme care in their modeling and verification, since relevant timing information is not captured within the event or cycle simulator models.

This type of clock section crossing is quite distinct from a second type of crossing that also exists in great numbers on these chips: synchronous communications between latches of differing frequency. The macros implementing the physical layers of eSTI and mSTI employ synchronous interfaces internally between the custom-designed, very-high-speed logic that makes up the serial timed interface and the logic that interfaces with the host using a parallel port and a much slower frequency of the chips. These synchronous clock section crossings are correctly modeled by the cycle simulation models and the static timing analysis; consequently, they do not require the special modeling described below for signals crossing asynchronous clock sections. This section discusses the complexities of the asynchronous design, and the modeling approach taken to stress the design.

The first difficulty encountered in modeling the asynchronous crossings is to identify all signals crossing an asynchronous boundary requiring special focus in verification. The switch chip contains legacy macros (both updated and unaltered), as well as entirely new macros, all clocked and connected in ways that are different from previous designs developed for the IBM eServer z900 I/O subsystem. Furthermore, different designers picked up most of the legacy logic, so having those crossings identified by the designers is not an option. EinsTimer, a tool used for static timing calculations, is employed after setting up the clock transition times in the control files in a way that enables the identification of all crossings as highly negative "slack." *Slacklist* is a term for the output of a static timing tool. The slacklist lists the elements in a path from the launching latch to the capturing latch, with all gates in between. A slacklist shows the signal arrival time and the timing slack at every point along each path. The timing slack is the difference between the time when the signal arrives and the time when it would be required to arrive in order to barely meet the timing requirement at the end of all paths it feeds. (An example timing

requirement is that a signal must arrive before the falling clock at the capturing latch.)

Another configuration of the timing tool entirely prevents timing of paths between latches that are clocked from the same oscillator source. This produced large slacklists containing only those paths that were launched by a latch in a clock section that was asynchronous to the receiving latch. Studying the paths and their structures resulted in identification of logical signals to be considered as the "crossing point" for each path, and created a timing "cut point" at that signal. (A *crossing point* is a point conveniently chosen in the logic path that a signal traverses when it has been launched by a latch clocked by one frequency and captured by a latch clocked with another frequency. For the simplest and most common case, this is simply a wire or buffer between the two latches. In more general cases, it could be a point in the path after logic has combined a small set of signals which share the same launching clock frequency and are being combined before being captured by a latch with another clock frequency. A *cut point* is a control command which instructs the timing tool that this path should not be timed. That is, no signal arrival time is carried through the cut point, and therefore paths crossing this point do not have negative slacks.) After this effort was completed, a slacklist with no negative slacks and a list of cut points for every asynchronous crossing signal on the chip were available.

The list of asynchronous crossing signals serves as the input for creating a simulation model that stresses the asynchronous crossings. A simple logic model for a signal delay randomizing function (SDRF) is shown in **Figure 13**. The logic has an input, a random number generator, and a switch controlled by the least significant bit of the random number. The switch selects the input signal or the output of a latched version of the signal with one cycle of delay. The only output of this logic is the output of the switch. A script loads the logic design file of the chip. Using the cut-point list that contains all of the crossings, the script cuts out each of the asynchronous crossings one by one and reconnects each cut pair with a customized copy of the SDRF element. The script writes the resulting altered chip logic to a file, which is used as an input to the cycle simulator.

The cycle simulation test suite is run on this altered model containing the randomized crossing delays. We find that many asynchronous crossings were designed in such a way that they can tolerate this level of randomization. For those that can tolerate it, we consider the signals to have successfully passed the verification criteria at this point in the process. However, experience shows that this method is overly pessimistic, and some of our crossings have failed because of this unrealistic pessimism. The pessimism springs from the act of determining a new random delay

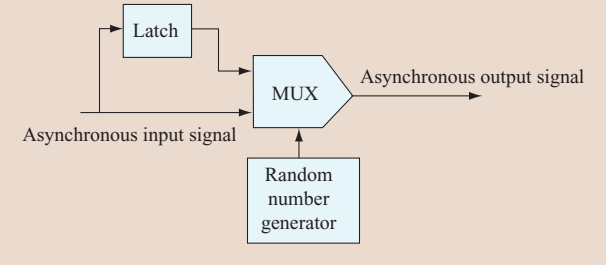

Figure 13

Delay macro insertion in asynchronous interface for cycle simulation.

for a signal every cycle. A signal can experience a delay of one cycle on the crossing during a given cycle, and then may have no delay the very next cycle. In effect, this shortens a signal by one cycle. Some of the crossings were designed to guarantee delivery of a single cycle pulse regardless of the clock alignment, by relying on edge detection to transmit the signal. For that type of design approach, the shortening of a signal could make the signal disappear completely. After a thorough review of such an edge-detection crossing design indicating that the design is sound, a second type of randomizing algorithm for the crossings that used this design has been created. The delay value of every such crossing signal is randomized once at the start of every test case, and the random delay selection remains unchanged throughout the execution of that test case. Over time, this allows a rich set of combinations of delays across the different crossing signals, since the test-case suite is regressed repeatedly on a daily basis. This is still pessimistic, since it assumes that any signal crossing can pass or fall behind any other signal crossing, because the randomizer sets the delay of all crossings. It turns out that the asynchronous designs implemented on the switch chip can tolerate this pessimism, so the logic does not require any change.

One unexpected benefit from implementing this particular verification process was that the portion that identified the crossings could be done very early, because of the simplicity of setting up the timing analysis tool to produce the slacklists. The manual steps involved in selecting the best crossing cut points required designers to review each crossing individually. The slacklists showed the tree structure of the path. This has been shown to be a good method for allowing designers not familiar with the legacy logic to become familiar with the crossing design techniques and to discover unintentional crossings and crossing design mistakes very early. Many problems were discovered early in the design cycle, well before the asynchronous model was built, and this proved very

valuable in the intense period leading up to the final verification and release of the design to manufacturing.

Summary and concluding remarks

The key to first-time success of the overall verification effort of the switch chip was appropriate planning. Identifying the tasks and incorporating error injection and pervasive verification up front, as well as designing the verification environment to enable exercising and checking all of these functions on the appropriate level, allowed limitations to be discovered before the critical phase of the hardware verification. The flexibility to switch hardware abstraction levels between RTL and gate level for specific verification tasks using the same environment enabled the team to provide early feedback to the designer and permitted the full chip design to be tested as it became available. Enabling the design for cycle simulation by applying special implementation techniques of non-cycle-simulation-compliant logic provided the ability to exercise all functionality on a full chip using the same simulator maintaining a high simulation performance. This must be accompanied by an event simulation of the components using those non-cycle-simulation-compliant macros on a unit level. Since this happens on smaller design pieces, it is a negligible effort compared with verification of the complete logic of the chip in an event simulation environment. The use of cycle simulation on the full chip provides a tremendous performance improvement. However, full automation of the process in converting components containing non-cycle-simulation-compliant logic requires additional effort and is not possible at this time.

Functional coverage was an integral part of the verification process, ensuring high quality of the design to be released to manufacturing. The implementation of coverage models provided a high level of confidence upon completion of the verification efforts and helped quantify the stress on the logic designs.

The asynchronous interface verification techniques described in this paper were critical to the success of bringing up the real hardware. Varying the frequencies of the asynchronous clock domains early on the RTL model allowed the verification of critical design function, which was related to the frequency. Verification of the functionality of the asynchronous interfaces themselves involved the use of a timing tool to identify the asynchronous crossings, and the insertion of special timing delay logic at the identified crossings. This logic generates a random delay at the asynchronous interfaces, thus emulating real behavior using the random cycle simulation environment. These techniques for verifying the function of asynchronous interfaces in the specified frequencies proved to be a success when the prototypes were tested in a real machine. All asynchronous crossings operated as intended. These techniques can generally be applied to verify all ASIC chips with asynchronous boundaries.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. D. J. Stigliani, Jr., T. E. Bubb, D. F. Casper, J. H. Chin, S. G. Glassen, J. M. Hoke, V. A. Minassian, J. H. Quick, and C. H. Whitehead, "IBM eServer z900 I/O Subsystem," *IBM J. Res. & Dev.* **46,** No. 4/5, 421–445 (July/September 2002).
2. E. W. Chencinski, M. J. Becht, T. E. Bubb, C. G. Burwick, J. Haess, M. M. Helms, J. M. Hoke, T. Schlipf, J. M. Turner, H. Ulland, M. H. Walz, C. H. Whitehead, and G. Zilles, "The Structure of Chips and Links Comprising the IBM eServer z990 I/O Subsystem," *IBM J. Res. & Dev.* **48,** No. 3/4, 449–459 (May/July 2004, this issue).
3. C. L. Rao, G. M. King, and B. A. Weiler, "Integrated Cluster Bus Performance for the IBM S/390 Parallel Sysplex," *IBM J. Res. & Dev.* **43,** No. 5/6, 855–862 (September/November 1999).
4. J. M. Hoke, P. W. Bond, R. R. Livolsi, T. C. Lo, F. S. Pidala, and G. Steinbrueck, "Self-Timed Interface of the Input/Output Subsystem of the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 447–460 (July/September 2002).
5. J. Darringer, E. Davidson, D. Hathaway, B. Koenemann, M. Lavin, B. Lee, J. Morrell, S. Ponnapalli, K. Rahmat, W. Roesner, E. Schanzenbach, and L. Trevillyan, "EDA in IBM: Past, Present and Future," *IEEE Trans. Computer-Aided Design, Integrated Circuits & Syst.* **19,** 1476–1497 (December 2000).
6. B. Wile, M. P. Mullen, C. Hanson, D. G. Bair, K. M. Lasko, P. J. Duffy, E. J. Kaminski, Jr., T. E. Gilbert, S. M. Licker, R. G. Sheldon, W. D. Wollyung, W. J. Lewis, and R. J. Adkins, "Functional Verification of the CMOS S/390 Parallel Enterprise Server G4 System," *IBM J. Res. & Dev.* **41,** No. 4/5, 549–566 (July/September 1997).
7. J. M. Ludden, W. Roesner, G. M. Heiling, J. R. Reysa, J. R. Jackson, B.-L. Chu, M. L. Behm, J. R. Baumgartner, R. D. Peterson, J. Abdulhafiz, W. E. Bucy, J. H. Klaus, D. J. Klema, T. N. Le, F. D. Lewis, P. E. Milling, L. A. McConville, B. S. Nelson, V. Paruthi, T. W. Pouarz, A. D. Romonosky, J. Stuecheli, K. D. Thompson, D. W. Victor, and B. Wile, "Functional Verification of the POWER4 Microprocessor and POWER4 Multiprocessor Systems," *IBM J. Res. & Dev.* **46,** No. 1, 53–76 (January 2002).
8. B. L. Keller and T. J. Snethen, "Built-In Self-Test Support in the IBM Engineering Design System," *IBM J. Res. & Dev.* **34,** No. 2/3, 406–415 (March/May 1990).
9. P. S. Gillis, T. S. Guzowski, B. L. Keller, and R. H. Kerr, "Test Methodologies and Design Automation for IBM ASICs," *IBM J. Res. & Dev.* **40,** No. 4, 461–474 (July 1996).
10. L. Stok, D. S. Kung, D. Brand, A. D. Drumm, A. J. Sullivan, L. N. Reddy, N. Hieter, D. J. Geiger, H. H. Chao, and P. J. Osler, "BooleDozer: Logic Synthesis for ASICs," *IBM J. Res. & Dev.* **40,** No. 4, 407–430 (July 1996).
11. G. A. Van Huben, T. G. McNamara, and T. E. Gilbert, "PLL Modeling and Verification in a Cycle-Simulation Environment," *IBM J. Res. & Dev.* **43,** No. 5/6, 915–925 (September/November 1999).
12. G. G. Hallock, E. J. Kaminski, Jr., K. M. Lasko, and M. P. Mullen, "SimAPI—A Common Programming Interface for Simulation," *IBM J. Res. & Dev.* **41,** No. 4/5, 601–610 (July/September 1997).
13. A. Kuehlmann, A. Srinivasan, and D. P. LaPotin, "Verity—A Formal Verification Program for Custom

CMOS Circuits," *IBM J. Res. & Dev.* **39,** 149–165 (January/March 1995).
14. J. Silverio, H. Kumar, S. Joseph, and B. Hoppe, "Method for Verification of Various Asynchronous Frequency Domains in a STI Switch Chip, by Implementing a Greatest Common Factor Mathematical Approach to Generate the Discrete Simulation Cycle Values and Randomly Select the Simulation Cycle Percent Variation Across the Multiple Domains," patent filed 2003.

Received October 1, 2003; accepted for publication January 28, 2004; Internet publication April 27, 2004

Bodo Hoppe *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (bohopp@de.ibm.com).* Mr. Hoppe is a member of the IBM global verification team in the German Research and Development Laboratory. He received a degree in electrical engineering, with emphasis on telecommunications, from the Berufsakademie Stuttgart, Germany, in 1993. At that time, he joined the IBM Research and Development Laboratory in Germany, where he has worked on hardware system verification of the IBM S/390 G3 Parallel Enterprise Servers. He worked on several generations of systems in microprocessor and system verification, and is currently working on hardware verification of an I/O adapter chip for future eServer systems. Mr. Hoppe received an IBM division award in 1996 for the G3 bringup and the hardware system validation effort of the IBM S/390 G3 Parallel Enterprise Servers. In 2000, he was awarded the IBM Outstanding Technical Achievement Award for his verification work on the IBM z900 Server. Mr. Hoppe is an author or coauthor of several patents and technical papers.

Bridgette Arthur-Mensah *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (bfosu@us.ibm.com).* Ms. Arthur-Mensah is a member of the IBM Advanced Hardware Verification team in Poughkeepsie, New York. She received her B.S. degree in electrical engineering from Rutgers University in January 2001, at which time she joined the IBM I/O subsystem development team focusing on the verification of high-speed I/O switches for the z990 servers. Ms. Arthur-Mensah is currently working on the next generation of the switch for future zSeries servers while pursuing an M.S. degree in electrical engineering at Columbia University.

Edward W. Chencinski *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (chencins@us.ibm.com).* Mr. Chencinski received a B.S. degree in electrical engineering from Lehigh University in 1980, joining IBM that same year. He was involved in the ES/3090 SCE and expanded storage hardware design, as well as the logic support element design of the ES/9000. In the early 1990s, Mr. Chencinski joined the G3/G4 CMOS cryptographic hardware processor design team, focusing on pervasive functions, simulation, timing, and modular exponentiation. He is currently a Senior Engineer leading the STI switch chip design team.

Sabina Joseph *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (sabinaj@us.ibm.com).* Ms. Joseph is currently an Advisory Manager of the eServer Advanced Hardware Verification Team. In 1999 she joined IBM in Poughkeepsie, New York, after receiving an M.S. degree in computer science from West Virginia University and a B.A. degree in mathematics/computer science from Seton Hill University. From 1999 to 2002, she held technical positions in zSeries CP and I/O verification. In 2002, she became a manager, responsible for zSeries and pSeries I/O verification.

Haresh Kumar *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (hkumar@us.ibm.com).* Mr. Kumar is a Staff Engineer. In 2000 he joined IBM in Poughkeepsie, New York, after receiving a B.S. degree in electrical and computer engineering from the University of Rochester. He has held various technical positions in the I/O hardware verification group of the zSeries eServer and currently leads the verification team of a switch chip.

Jose F. Silverio *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (silverio@us.ibm.com).* Mr. Silverio received a B.S. degree in electrical engineering from the University of Hartford, Connecticut, in 1990, joining IBM that same year as a logical designer for an advanced I/O processor project. In 1995 he received an M.S. degree in computer engineering from Syracuse University. He joined the Advanced I/O Connectivity Hardware Verification team in 1996 and is currently a Senior Engineer. In 1996, Mr. Silverio received an IBM Team Award for his work on S/390 G3 fast internal bus design. He received IBM Outstanding Technical Achievement Awards for his design verification work on the STI-to-PCI bridge chip for the S/390 G4 (1999), Multiprise 3000 (2000), and z900 eServer (2001) projects. Mr. Silverio recently received an IBM Invention Achievement Award for his work on the STI switch chip for the z990 eServer.