Design and Test of the PowerPC^{TM} 603 Microprocessor

E. Kofi Vida-Torku, Charles H. Malley**, Sung Park*, Rowland Reed**
** International Business Machines Corp., ** Motorola Inc.*
Somerset Design Center
9737 Great Hills Trail
Austin, Texas 78758

Abstract

The PowerPC 603 [1] microprocessor is a powerful low-cost implementation of the PowerPC Architecture^{TM} specification. The structured design, logic verification and test data generation methodologies of the 603 are presented in this paper. The success of these methodologies has been demonstrated by meeting the 603's aggressive time-to-market goals.

1.0 The PowerPC 603^{TM} Microprocessor

The 603 is the second member of the PowerPC microprocessor family[1]. The 603 is a powerful low-cost superscalar implementation of the PowerPC Architecture specification[2]. The 1.6 million transistor, 85 mm^2 chip features on-chip 8Kbyte instruction and data caches coupled to a high performance 32/64-bit system bus. Power dissipation is under 3 watts at 80Mhz. Board and system test are supported using the JTAG IEEE P1149.1 port.

A simplified block diagram of the 603 is shown in Fig 1. A maximum of two instructions per clock cycle are fetched from the instruction cache into the instruction buffers and branch unit. The branch unit executes any branch in the prefetch buffer and redirects the prefetch unit accordingly. The dispatcher decodes two instructions at a time from the instruction buffer and dispatches them if possible to available execution units: System unit, Integer unit, Floating point unit and Load/Store unit. The dual 8K-byte data and instruction caches have associated memory management units (MMUs) which implement the PowerPC Virtual Enviroment Architecture[2]. The processor bus interface unit (BIU) accepts bus requests from the instruction and data caches and places the requests on the 603 external bus. A common on-chip processor (COP) and JTAG controller are used to control test features of the chip.

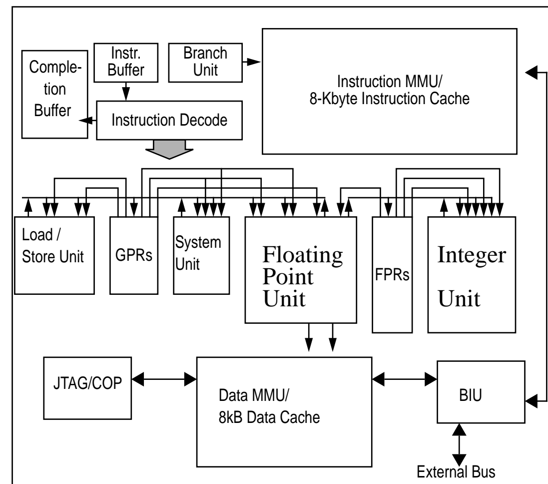

Figure 1: 603 Block Diagram

Some of the power reduction features are discussed in the next section followed by the 603 design methodology. Section three covers the functional, logic and physical verification process. Finally, the logic and array design for test methodology is presented.

1.1 Low Power features

The goal of the 603 design was to optimize for performance, power consumption and cost. One hard limit was set by the portable computer systems, which required the typical processor power dissipation be no more than 3W. Therefore, many design choices were made to minimize power consumption[3].

The 603 implements a Phase Lock Loop (PLL) for synchronizing the internal clock to the system clock (bus clock). The PLL provides the internal clock frequency to be 1-4X integer multiples of the bus clock up to the

---

1. In this document, the terms "PowerPC 603 Microprocessor" and "603" are used to denote the second implementation of the PowerPC microprocessor family. IBM, PowerPC, PowerPC 603, PowerPC Architecture and PowerPC 601 are trademarks of International Business Machines Corporation.

internal clock frequency limit of 80MHz. The multiple internal clock to bus clock frequency ratios allow system design flexibility, but also potential power savings by being able to lower the internal clock frequency by changing the ratio when necessary.

The output of the PLL is routed through an H-tree which provides a very low clock skew of +/- 70ns and an average power dissipation of 100mW at 80MHz. The single phase H-tree clock feeds clock regenerators that regenerate two phase clocks for clocking the internal logic. The clock regenerator also enables clock control functions for power management and testability. In power management, the clocks to the logic are disabled at the clock regenerator, while the H-tree is still clocking. This feature allows selective functional unit shut-down where idle units are shut down while other busy units are clocking.

For power reasons, the 603 is implemented with full static logic. The static logic also simplifies the design to preserve the logic state in the event the input clock to the processor is disabled which happens often in portable computers for power savings. The logic elements were designed to optimize the speed-power product. The cache on the 603 has been designed to save power. Features such as subarrays, and word line pulsing are used to minimize power consumption[3].

For further power savings, the 603 provides two active power management methods; static and dynamic power management. Static power management has three modes; Doze, Nap and Sleep which are software controllable through a control register. The static power management mode is used by the system software to lower the power dissipation of the 603 when a system idle condition is detected. Power saving gets progressively better going from Doze to Sleep. Doze mode is used in situations where no cpu activity is required, but cache coherency must be maintained. In Doze mode, the 603 maintains bus snooping and services a snoop hit on a dirty cache line. If the system does not require bus snooping during idle, then Nap and Sleep modes may be used for further power savings. Doze and Nap mode require the input clock and PLL configuration to be unchanged allowing a quick recovery to the full-on mode. Sleep mode can accomodate changes in the input clock or the PLL configurations, but requires a PLL relock time. Once the 603 enters a static power management mode, the clocks to most of the functional units are disabled. An external event such as an external interrupt will “wake up” the 603 from a static power management mode to the full-on mode, and the 603 will resume normal operation. An example of typical power dissipation numbers are; 366 mW in Doze, 135 mW in Nap and 47 - 105 mW in Sleep at 80MHz internal clock speed.

In contrast to static power management, the Dynamic power management requires no software intervention except for the one-time invocation of the mode. Dynamic power management system saves power on a cycle by cycle basis by turning on functional unit clocks only when specific instructions are dispatched to the unit. Benchmark tests show a typical power savings of 8.5% - 16% [3].

2.0 The PowerPC 603 Design Methodology

The various elements of the 603 design methodology are shown in Figure 2. The goal is to guarantee that the 603 that was designed at the functional level is what gets shipped for production. Some of the elements of the methodology are identical to that used for the PowerPC 601 TM design [4].

The first input to the process is the functional specification for each chip design defined in terms of a Register Transfer Level (RTL) description. The RTL model is the primary (golden) model used to determine functional accuracy. This model, which is code-compiled and verified using a cycle simulator, is state equivalent with the physical implementation; that is, every latch component is individually specified within the RTL description. The second input to the design process is the logical and physical descriptions of custom and library elements designed in the Cadence environment. The final input is Rules. A Rule is an abstraction of some aspect of a design put into a form that the design tools can understand. An example of a rule would be information about the timing paths through a design block in a form appropriate for the timing analysis tool.

The RTL model is constructed to be hierarchically identical to the actual partitioning used in the chip floorplan. The only difference being that some artificial levels within the RTL model are removed during the creation of the chip level netlist that represents the chip’s physical structure. The logic design of the PowerPC 603 microprocessor is partitioned within functional boundaries for which various design strategies are applied. One such partition, known as a Random Logic Macro (RLM), is a control logic structure which instantiates functional components. RLMs are implemented using techniques that range from automated logic synthesis [5] to full custom design.

Cycle simulation is used for functional verification. The cycle simulator used, TEXSIM, is very fast--simulating as many as 90 cycles per second --and allows billions of cycles to be executed in the verification of the design. Cycle simulation can be performed at both the unit and the chip level. Cycle simulation does not perform any analysis of the timing characteristics of the design.

Timing analysis is accomplished by a static timing analysis tool. Static timing analysis algorithmically examines all possible timing paths through the design, providing more complete coverage than that achieved by timing-based simulation. Two static timing tools are used : IDSP-Timing and STEP. The IDSP-Timing static timing tool is tightly coupled with the IDSP-Synthesis tool and is used for the timing analysis of synthesized RLMs.

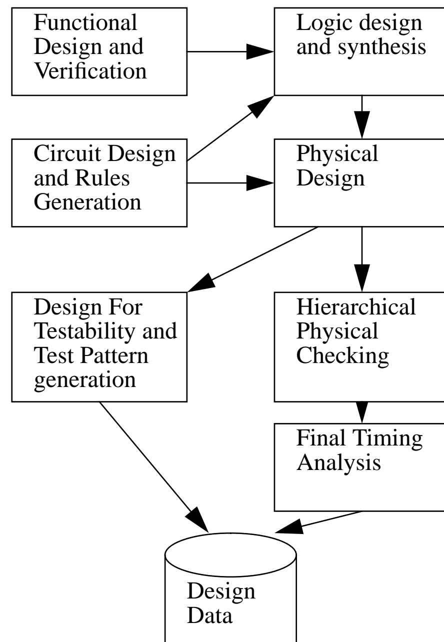

Fig. 2 603 Design Process Flow

The IDSP-Synthesis tool, in conjunction with IDSP-Timing, is used to automatically generate and fine-tune random logic macros (RLMs). STEP is used for full chip timing.

A boolean comparison tool, is used to algorithmically compare the functional description of the design with a logic implementation developed manually or via synthesis. Design-for-test and automatic generation of test patterns is provided by the use of Level-Sensitive Scan Design techniques and the TPG test pattern generation tool.

Automatic physical placement and wiring is performed using the stack-based paradigm of the IBM VIDAS tool suite.

Hierarchical Design Verification (HDV) is used for physical verification. Hierarchical design verification allows the physical design to be checked one level of hierarchy at a time, which requires much less compute time and space than does flat checking.

A fundamental concept within the PowerPC microprocessor design methodology is the use of library elements called building blocks. A building block is the lowest level element that can be referenced within the RTL model hierarchy. It is a self contained unit which has a functional specification, a logic schematic implementation, and a physical layout. The circuit design and construction of building block libraries takes place in parallel with the development of the RTL model. Building block circuits represent simple functions such as Boolean logic gates, latches, and multiplexors as well as more complex datapath elements including adders, shifters, rotators, and arrays. A building block may be used just once or it may be used a number of times. Building blocks are assembled in the construction of RLMs and are instantiated in higher level hierarchical structures which make up the chip design. The logic schematic for the building block is used to represent two distinct views. One is called a gate-level model and the other is called a switch-level model. For those cells in the hierarchy that do not contain transistors, the gate-level model and switch level model refer to the same schematic.

The gate-level view is a Boolean equivalent logic representation of the stuck-at fault model for these transistor circuits. The switch-level model refers to the cells within the building block schematic hierarchy which represent the transistor network that implements the functional model.

2.1 Early Design, Early Floorplanning, and Library Development

At the beginning of the design three activities take place concurrently--early design, early floorplanning, and library development. Critical timing path analysis is done at this stage using the transistor models and process parasitics delivered from the technology group, and the circuit simulation tool MSPICE . Some high-level ‘what-if’ design work using the logic capture and functoinal simulation tools is also done. Early floorplanning assigns area budgets to the various logic partitions, evaluates various placements of these partitions on the chip, and determines what global routing constraints the layout would have to conform to. This is currently done using the floorplanning portion of the VIDAS tool suite.

The library development process created reusable logic components to be used primarily in the semi-automatic design process (synthesis, floorplanning, routing). The most basic library elements, are called books or standard cells. The inputs to the process of developing these books are:

a) The types of functions for which books need to be developed, i.e. nand, nor, etc. b) Technology information, i.e. transistor models, process parasitics, design rules. c) Global routing constraints.

Examples of the outputs of this process are :
a device-level netlist, a layout database, a description of the logic function in text format, timing information and physical information indicating blockages, etc.

3.0 Design Verification

The design process consists of extensive design verification to guarantee first pass function and performance. The verification is done at the logic and functional level. Simulation vectors are generated using a Random TestPattern Generator (RTPG) [6] and are supplemented with designer specified functional patterns. These patterns are simulated on the RTL description and the results are compared against the expected response from an architectural model which is presumed to be correct.

3.1 Logic Verification

Since the RTL model is used to determine functional verification, the goal of the logic verification methodology is to determine equivalence between the functional specification (RTL model) and the switch-level implementation. The switch-level implementation is represented by the combination of the structure interconnect within the chip level netlist, the building block interconnect contained within RLMs, and the logic schematic which represents the building block's circuit design. This verification is accomplished hierarchically in three steps:
The first step is to determine the equivalence of the three building block level models: functional, gate-level, and switch-level. Next, the RLM's logic equations are verified to match the synthesized or customized technology mapped implementation. Finally, the interconnect of RLMs and building blocks within the RTL description are verified to match the interconnect of these blocks described by the chip level logic netlist.

3.1.1 Formal Verification using Boolean Comparison

A Boolean Equivalence Checker known as BEC is a design verification tool which compares static Boolean networks for logical equivalence using binary decision diagrams (BDDs) [7]]. In general, given enough time and space, BEC can compare any two networks of logic; but, because Boolean comparison is an NP complete problem, there is no guarantee that a comparison will be completed within the specified time and space constraints.
BEC is used to verify building blocks, RLMs, and structures. In the case of building blocks and RLMs, the simulation model (RTL) is translated into a technology independent Boolean logic network. During this translation, heuristics are applied to transform complex or high level RTL expressions into a logic structure containing low-level primitive functions.
The first application, known as building block BEC, verifies that the functional model and gate-level model are equivalent. The second application, RLM BEC, verifies that the functional specification for RLMs matches the technology mapped netlist created either by logic synthesis or by custom design. The third application called structure BEC, verifies that the interconnect within the upper levels of the hierarchy in the technology mapped chip netlist matches the interconnect specified within the RTL simulation model.

3.1.2 Building Block Verification

The verification of the three views: functional model, gate-level model, and switch-level model, is accomplished using one of three methods depending on the type of building block. Method 1 makes use of the transitive property: if A=B and B=C, then A=C. In this application, the verification methodology states that if the switch-level model can be proven equivalent to the gate-level model using exhaustive pattern simulation (A=B), and the gate-level model can be proven to be equivalent to the functional model using BEC (B=C), then the switch-level model is equivalent to the functional model (A=C); that is, all three are equivalent. This method is limited to combinatorial building blocks whose transistor cells have sixteen or fewer inputs.
Method 2 is used for building blocks that contain either sequential logic or transistor cells with more than sixteen inputs. This method uses test and functional patterns to verify the switch-level and gate-level models. BEC is still applied to verify that the gate-level and functional models match.
One type of building block called an array represents functional units which read and write memory structures containing multiple data locations. Circuits classified as arrays include caches, registers, block address translators, and content addressable memories. This type of building block does not lend itself to verification methods 1 or 2 because it is typically very difficult to describe the actual logic inside of arrays in terms of a functional and gate level model. The cycle simulator, ATPG fault simulator, and BEC support a high level behavioral construct called a RAM primitive which models the address decoding, read/write clocking, and output data latching. When this primitive is used in the functional and gate-level models of the array, the real logic implemented in the switch-level schematic is not represented. Method 3 is used to verify arrays. This method simulates a set of verification vectors on all three models (functional, gate-level, switch-level). These patterns are a collection of test patterns, functional patterns, and patterns derived from RTPG. Figure 3 shows the logic verification methodology for building blocks.

3.1.3 Switch-Level model vs. Gate Level Model

The switch-level model is compared to the gate-level model by applying patterns to the transistor network in an event driven switch-level simulation. Three types of patterns may be applied. Exhaustive patterns are used to compare the switch-level model to the gate-level model for transistor circuits which have a relatively small number of inputs. For this purpose, an exhaustive set of input patterns (2^n, where n = number of inputs) is simulated on the gate-level model in *good machine mode* to obtain the expected outputs. These patterns are then simulated on the switch-level model and the results compared to the expected outputs.

When exhaustive simulation is not suitable for a cell, two other types of patterns may be applied. Test patterns are created by executing an ATPG tool which uses the gate-level model as input. By simulating ATPG patterns on two representations of a design and comparing the results, a large class of errors can be detected[8]. Using ATPG to verify the switch-level and the gate-level models insures that a correct set of production test patterns will be created for the circuit. To achieve a higher degree of verification assurance, a set of functional patterns is simulated on the gate-level and switch-level models in addition to the test patterns.

3.1.4 Gate-Level Model vs. Functional Model

The next step is the verification of the gate-level model. This is determined by comparing it to the functional model using BEC. The logic represented by the gate-level model defines the building block's Boolean relationship in terms of its inputs and outputs. The functional model is compiled and transformed into a Boolean logic network and compared to the gate-level representation. Non-Boolean functions such as tri-state drivers and latches are converted to Boolean equivalents that allow networks containing these components to be compared.

3.1.5 Switch-Level Model vs. Functional Model

As a final verification step, the functional model can be equated to the switch-level implementation directly by comparing the results of pattern simulation at the switch-level and functional level. This is only necessary when exhaustive verification of the switch-level vs. gate-level or gate-level vs. functional models is not possible.

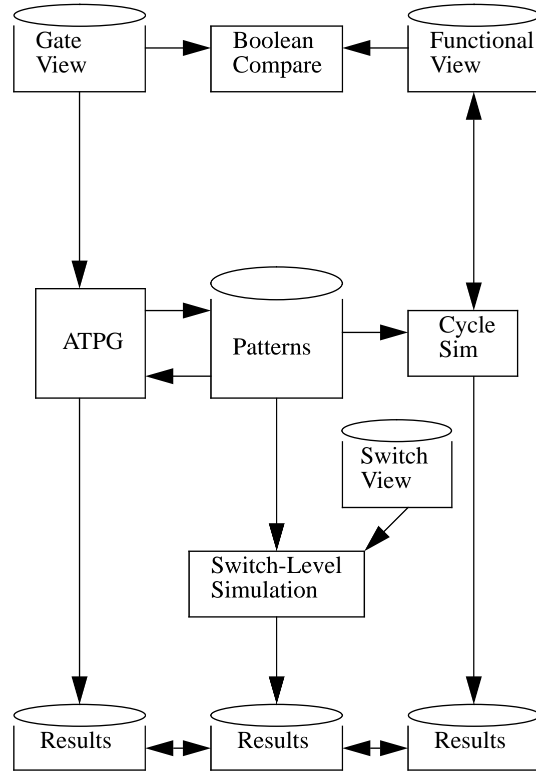

Fig. 3. Building Block Logic Verification Methodology

3.2 System Level Verification

After logic verification of all sub-blocks, each unit of the 603 is verified using the cycle simulator. The code compiled RTL mode of a unit is subjected to random and non-random patterns. The verified units are assembled into the 603 design model for system verification. This process is accomplished in two steps: Implementation Verification Programs (IVP) used to test the hardware design features. Next, Architectural Verification Programs (AVP) are used to verify the instruction set implementation against the PowerPC Architecture specification.

4.0 PowerPC 603 Design for Test

The 603 is being manufactured by IBM and Motorola, two companies that traditionally have different methods of generating manufacturing test data. A single test methodology that combines IBM's formal Level scan design (LSSD) and Motorola's adhoc design for test was agreed on [9]. Logic circuits are tested with full scan LSSD design. A DC stuck fault coverage of greater than

95% was established as a minimum target. Hard to test areas of the logic are tested by functional patterns using the 603 assembly language. Functional patterns are also used to test all small arrays. Large memory arrays are tested by Array Built-in self test. Quiescent current measurement is used as a supplement to the scan and functional patterns.
About 30% of the 1.6 million transistors are in random logic on the chip. However, the random logic takes up roughly 70% of the 85 mm^2 die area. This area of the chip is tested primarily through automatic test pattern generation (ATPG) based on Level Sensitive Scan Design (LSSD). The 603 was designed adhering to the basic tennants of LSSD. All chip latches are implemented as scannable Shift Register Latches (SRL).
Test clocks were designed with LSSD controls to provide independent control of the SRL's master and slave latch components during LSSD testing. Chosing a full scan methodology provided a cost effective means of providing high quality manufacturing test data using automatic test pattern generation programs. Full scan also provided us with facilities which aid in defect analysis, chip debug and characterization.

The first step in the scan test pattern generation process consists of ensuring that circuits in the building block OTS level are 100% testable. In this step, we had to ensure that the test model response to non-functional test patterns would match the actual circuit implementation. Pass gate muxes and embedded RAM models received the most attention. All macro models used for test generation were compared against the actual transistor circuit via simulation as described in section three.
The 603 master logical model was translated into the standard EDIF language for LSSD scan and clocking rules checking. Clock rules violations were checked and designed out to ensure test data that was race free. The next rule violation was to ensure that no tri-state burnout condition could occur during the scanning process or due to non-functional patterns generated by the ATPG program. If this could occur, the act of testing the chip could actually damage it. The final step in the scan test pattern generation process is to submit the test procedure, fault list and the EDIF design file for test pattern generation.
To provide the control for scan based design, three chip pins were dedicated to LSSD testing: two test clock pins are used to provide complete, independent control of the master and slave components of each SRL during scan-based testing. A third pin, is activated to put the 603 into LSSD scan-based testing mode. In addition, a shared pin enables shift register operation in LSSD test mode. A second shared pin prevents array content corruption by inhibiting write operations during scanning. All scan input pins and scan output pins are shared with functional pins. None of the LSSD control pins is available to system designers for LSSD test on a board.
The overhead incurred in the LSSD design consists of the extra scan port on each flipflop (SRL) and the die area dedicated to routing the scan string wires and the LSSD control signals

5.1 Memory test

The memory arrays in the 603 account for abot 70% of the transistors and occupy 30% of the chip area. These arrays can be classified into two groups:
1) Small arrays consisting of six arrays in the Memory Management Unit, the General Purpose Register (GPR) and the Floating Point Register (FPR).
2) Large arrays consisting of the dual 8K-byte data and instruction caches together with the two TAGs (512 bytes each).
The small arrays are tested with functional patterns using the 603 assembly language. Specific memory algorithms are written to read from and write into all locations in memory. After the assembly code is compiled, the compiled code is simulated using a cycle simulator. The output of the simulator is then combined with the 603 timing specification to produce a converted functional pattern set for the tester.

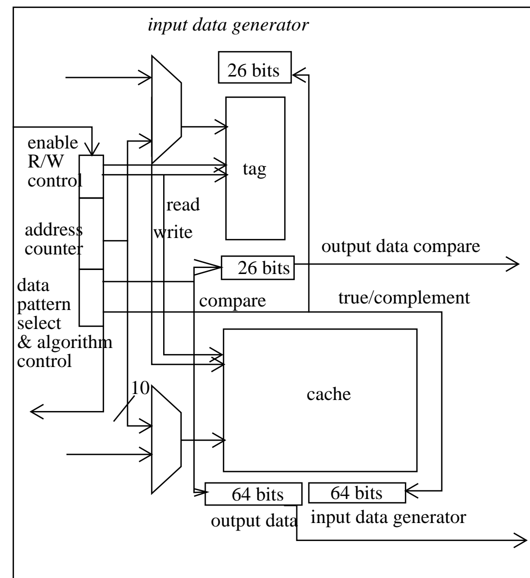

Figure 5: ABIST Block Diagram [9]

The data and instruction caches together with the two TAGs are tested with array built-in self-test (ABIST). The ABIST structure [9] is shown in Figure 5. A deterministic memory algorithm operating concurrently for pattern generation and output data comparison was implemented to provide 100% coverage of all cells.

The self-test hardware is integrated for each cache and-TAG. A common address generator feeds both the Cache and the TAG. However, each array has independent data input and output registers. The complete ABIST sequence is initiated through and results obtained via the JTAG/COP test port. The overhead of the ABIST is about. 0.4% of the total chip area, occuping less in less than 2% of the CACHE/TAG array

5.2 System Test Design Support

The 603 has a unique functional unit called the Common On-chip Processor (COP) that provides debug and various other test support. The major debug and test support functions of the COP are:

- IEEE 1149.1 JTAG standard boundary scan.
- JTAG compliant private instructions.
- LSSD test control.

The IEEE 1149.1 JTAG public instructions Bypass, Sample/Preload, and Extest, are implemented in the COP.

The debug functions of the COP are controllable by 39 JTAG compliant private instructions. The COP operation will, typically, be controlled by an external hardware debugger which communicates with the COP via the standard JTAG interface pins.

The three JTAG public instructions and the 39 private COP instructions are implemented as 8 bit JTAG instructions. With these instructions, the user is able to take control of the 603 for debug operations. A typical debug session would involve stopping the processor, scanning out all the internal states and dumping out the contents of any internal arrays, such as caches. The scannability allows the user to alter the state of any latch or array content to initialize the 603 to a particular state or to simulate an error condition.

The silicon overhead for the COP is 0.8% of the die area which is minute compared to the extensive debug capability provided.

5.0 Summary and conclusion

The PowerPC 603 microprocessor is a low-power RISC microprocessor designed using the structured design, verification and test methodologies outlined in this paper. These methodologies enabled first silicon to boot operating systems. Production test and system debug features were designed into the chip thus enabling quick turn around from silicon foundary to computer system design factories.

Acknowledgements

We acknowledge the valuable contribution of Johnny Leblanc, Ron Walther for contributions on IBM design for test, Craig Hunter, Tom Munns and Javier Prado for Motorola functional test contribution. The efforts of the tools group and 603 design teams are appreciated.

References

[1]] B. Burgess, M. Alexander, Y. Ho, S. Litch, S. Mal lick, D. Ogden, S. Park, and J. Slaton, “The PowerPC™ 603 Microprocessor: A High Performance, Low Power, Superscalar RISC Microprocessor,” Proceedings of COMPCON 1994, February 1994.

[2] E. Shila, “The PowerPC Architecture”, IBM RISC Syatem/6000 Technology: Volume II, IBM Corporation, 1993

[3] S. Gary, C. Dietz, J. Eno, G. Gerosa, S. Park, H. Sanchez, “The PowerPC 603™ Microprocessor : A Low-Power Design for Portable Applications”, Proceedings of COMPCON 1994, February 1994

[4] T. Brodnax , M. Schiffli and F. Watson ,”The Power PC™ 601 Design Methodology”, Proceedings of the International Conference on Computer Design, Oct. 3-6, 1993.

[5] J. Darringer, D. Brand, J. Gerbi, W. Joyner, L. Trevillyan, “LSS: A System for Production Logic Synthesis,” IBM Journal of Research and Development, Vol. 28, No. 5, pp. 537-545.

[6] A. Aharon, A. Bar-David, B. Dorfman, E. Gofman, M. Leibowitz, V. Schwartzburd, “Verification of the IBM RISC System/6000 by a dynamic biased pseudo-random test program generator”, IBM Systems Journal, Vol 30, No 4, 1991.

[7] S. J. Friedman and K. J. Supowit, “Finding the Optimal Variable Ordering for Binary Decision Diagrams,” 24th Design Automation Conference, ACM/IEEE, June 1987, pp. 348-355.

[8] M. Abadir, J. Ferguson, T. Kirkland, “Logic Design Verification via Test Generation”, IEEE Transactions on Computer-Aided Design, Vol 7 No 1, January 1988.

[9] C. Hunter, E. K. Vida-Torku, J. Leblanc,”Balancing Structured and Ad_hoc Design for Test: Testing of the PowerPC™ 603 Microprocessor”, Proceedings of the IEEE International Test Conference, Oct. 3 - 5, 1994.