Testability Features of AMD-K6^{TM} Microprocessor

R. Scott Fetherston, Imtiaz P. Shaik and Siyad C. Ma
California Microprocessor Division
Advanced Micro Devices
One AMD Place, Sunnyvale CA 94088-3453 USA

Abstract

This paper describes the testability features and test pattern development methodologies for the AMD-K6^{TM} Microprocessor. The embedded **Design for Testability** *(DFT) structures and test strategy provide high quality manufacturing tests.*

1 Introduction

Testability features and pattern development methodologies adapted for the AMD-K6^{TM} are described in this paper. To provide support for both lab debug and pattern generation for wafer sort testing, it was decided at the outset of the project to make the AMD-K6^{TM} design fully scannable. I/O cells are equipped with scan to support boundary scan and JTAG compatibility. BIST is utilized on all major internal full-custom functional blocks. A novel approach for IDDQ pattern generation is used.

Section 2 provides an overview of the processor architecture. Section 3 describes the *Design for Testability* (DFT) features. Section 4 presents scan test generation. Section 5 describes cell library development for path delay-fault test generation. We also summarize path delay-fault results. Section 6 outlines BIST for cache memory arrays and the emcode ROM. In Section 7 we briefly discuss the IEEE 1149.1 JTAG boundary scan implementation for AMD-K6^{TM}. Section 8 details issues related to IDDQ testing of the processor. Finally we present conclusions and future work.

2 Overview of The AMD-K6^{TM} Architecture

The AMD-K6^{TM} processor is based on a RISC core known as the *Enhanced RISC86 microarchitecture.* This architecture implements the X86 instruction set by internally translating X86 instructions into RISC-like operations called *RISC86 Ops* [4]. The processor decodes up to two X86 instructions per clock, most of which are decoded by hardware into one to four RISC86 Ops, whereas the uncommon instructions are mapped into ROM-resident RISC sequences. The instruction scheduler buffers up to 24 RISC86 Ops and up to six Ops are issued out-of-order to seven parallel execution units, speculatively executed and retired in order (Figure 1). The Branch Resolving unit uses two-level branch prediction based on an 8192-entry branch history table, a 16-entry branch target cache and a 16-entry return address stack. The processor has an MMX unit that incorporates the *multi-media extensions* (MMX) to the X86 instruction set. The MMX unit uses a *Single-Instruction Multiple-Data* (SIMD) technique to perform highly-parallel and compute intensive algorithms involved in multimedia applications.

Figure 2 shows a rough floor plan of the AMD-K6^{TM} processor. The processor implementation contains large embedded memory arrays. The level-one instruction and data caches are 32KB each and are 2-way set associative. The data cache allows simultaneous load/store operations every cycle. The instruction/data TLB's store 64/128 entries. The instruction tag RAM stores 512 20-bit physical tags and logically

INTERNATIONAL TEST CONFERENCE

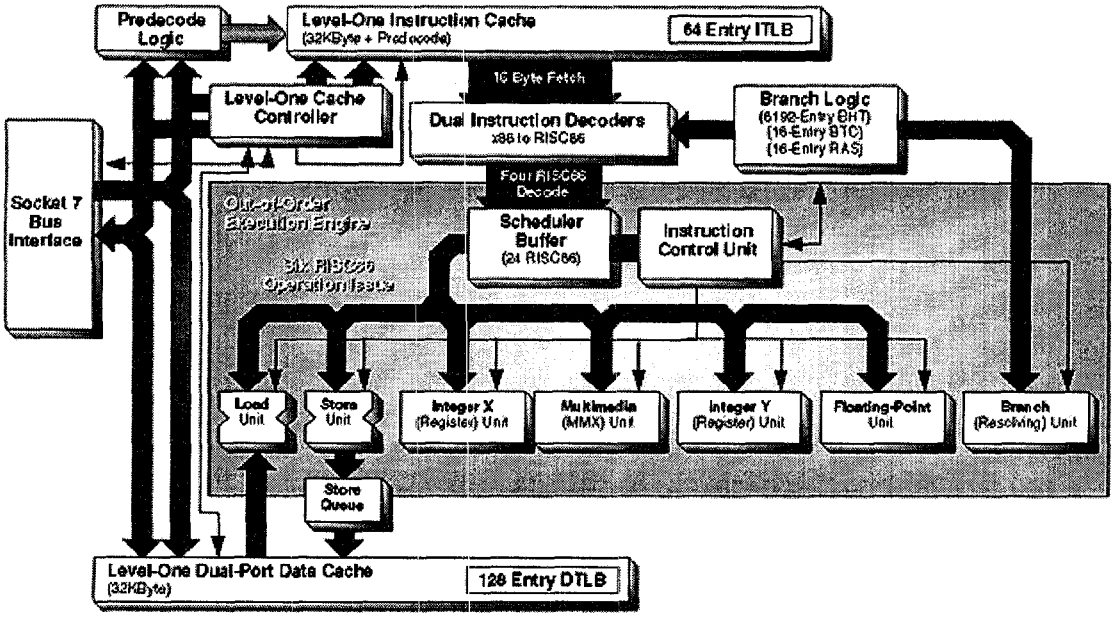

Figure 1: AMD-K6^{TM} Processor Block Diagram.

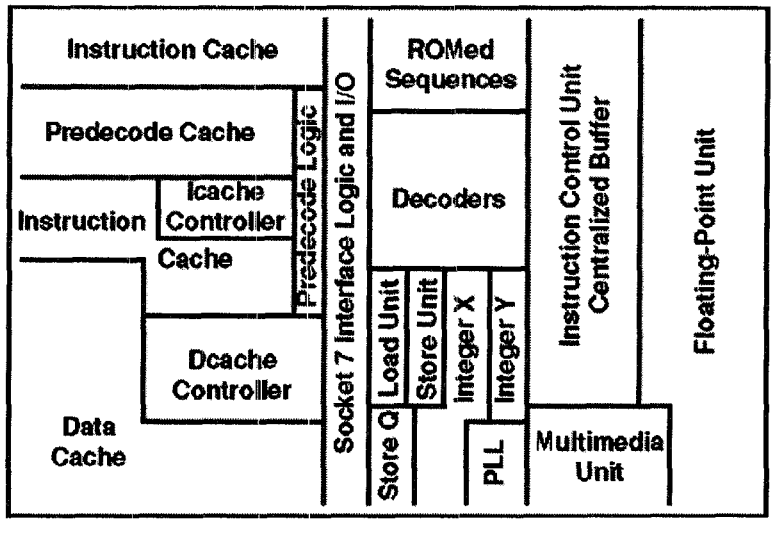

Figure 2: Floor plan of AMD-K6^{TM} Processor.

is 2-way set associative. However, it is physically constructed with eight sets of TAG-TLB comparators and eight sets of snoop comparators, with eight tags being read each cycle. This allows all possible synonyms to be checked in a single cycle, at the expense of layout complexity and area. The Instruction Predecode cache stores up to 8K 40-bit predecoded instruction words. The floating-point unit is implemented separately and runs on its own ϕ_1/ϕ_2 clock grid, synchronous with the main processor clock (PCLK), and is interfaced with the core logic through the *Numeric Processor Interface* unit. The processor speaks to the outside world through the standard Pentium^{TM} socket-7 interface shown in the center of the Figure 2.

The processor is implemented, at this writing, using a 0.35μm 5-metal layer CMOS technology with shallow trench isolation and tungsten local interconnect. The die contains 8.8M transistors, about a third of which are in the cache arrays. C4 solder bump flip-chip technology is used to assemble the die into a ceramic 321-pin PGA.

3 Design For Testability Features

3.1 Full-Scan Design

To provide support for both lab debug and pattern generation for wafer sort testing, it was decided at the outset of the project to make the AMD-K6^{TM} design fully scannable. Therefore, all internal storage elements are scannable and I/O cells are equipped with scan to support boundary scan and JTAG compatibility. A logic diagram of the underlying functionality of the internal storage element, depicted in Figure 3, illustrates a *shared master* rising edge triggered scan flip-flop. In normal operation, it operates as a flip-flop. For scan operation, the clock is held low and data is scanned using non-overlapping pulses on the shift clock pins SC1 and SC2 as shown in Figure 4. Like the Shift Register Latch [5], this design eases the impact on the routing constraints of the shift clock signals and insures uncorrupted scans by increasing the non-overlap time of the shift clocks. This design does not support the use of a narrow interval between a shift pulse and a clock edge for path delay testing. However, scan test patterns have been produced for path delay testing of the AMD-K6^{TM} design by scanning in a vector and clocking with a narrow interval between 2 clock edges (see Section 5.3). Recent evidence suggests that this provides a more effective path delay test [6].

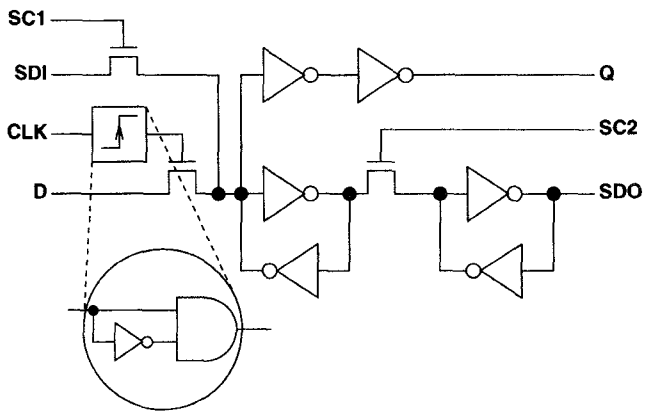

Figure 3: Logic Diagram of Scan Storage Element.

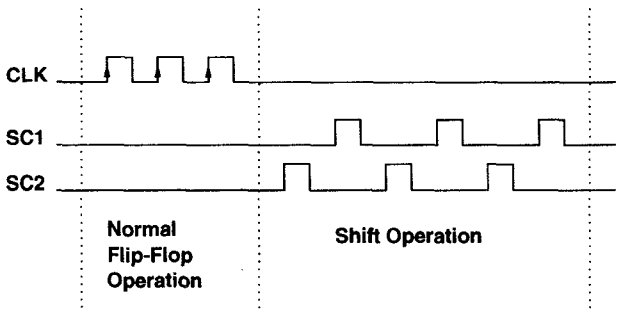

Figure 4: Clock Waveforms for Normal and Shift Operations.

Scan chain stitching of the gate level netlist is performed with an intelligent stitch order extracted from a placed netlist. The design utilizes 4 scan chains with over 37,000 total scan storage elements.

3.2 Embedded Clock Control

The design uses an on-chip phase locked loop (PLL) to generate the distributed internal clock.

The PLL produced clock is a programmable multiple of the external clock signal applied to the device. The PLL can be bypassed with an external clock to support scan and other debug testing. To better support debug of speed failures the PLL can be put into special test modes to utilize the dedicated high-speed operation, rather than using the bypass clock. Therefore, control of the PLL implements the following different modes of operation:

1. Cycle Stretch
2. Stop PLL
3. Pulse PLL

In Cycle Stretch mode the device can operate at high speed to a desired pipeline cycle, double the period of the next one or more cycles, and then resume at the normal high rate. In Stop PLL mode the PLL can be halted on a desired cycle, after which a scan out of the design can be performed. In Pulse PLL mode the PLL can provide a selective number of high-speed clock pulses. This supports scan in, clock twice, scan out operation in speed testing.

4 Scan Test Patterns

Scan stuck-at test patterns have been created by a conventional ATPG tool. The AMD-K6^{TM} design contains just over one million simulator primitive elements, as reported by the gate level fault simulation tool. The full scan implementation makes combinational ATPG possible and thereby provides a realistic opportunity to push the stuck-at fault coverage goal for this large design close to 100 %.

To characterize the test quality of the AMD-K6^{TM} product, the final fault coverage number will be determined by several metrics, including the stuck-at. One of the metrics, being implemented at this writing, is an inter-cell bridging fault simulation algorithm. The bridging fault simulation scheme is much like that described in [3] but with an improvement to account for feedback paths. Current plans are to fault grade the stuck-at scan patterns with the bridging fault simulator and generate additional patterns to improve the bridging fault coverage should it prove to be insufficient. Future plans include fault grading and pattern generation of scan patterns for stuck-open faults.

5 Path Delay-Fault Test Patterns

Due to the high operation speed of the AMD-K6^{TM}, AC tests are necessary to ensure that the chip operates at the target frequency. Besides at-speed parallel patterns, separate scan-based AC tests were generated using a path delay fault ATPG tool. Path delay fault patterns were generated for two purposes: silicon timing debug and delay fault detection for production testing. The flow of generating path delay fault test patterns starts with cell library modeling, followed by static timing analysis, and finally test pattern generation. Each of these three tasks are described below.

5.1 Cell library development

Most of the cell library models used for producing single stuck-at test pattern generation can be used for path delay fault ATPG. However, some cell models need to be adapted for the path delay fault ATPG methodology. For example, redundancy may be introduced by the circuit designer for reasons such as performance enhancement or hazard removal. This may create some paths that are functionally redundant, but can physically propagate a transition. Since redundancy will introduce untestable single stuck-at faults, such paths are typically removed in gate level models used for single stuck-at ATPG. For path delay fault ATPG, functionally redundant paths are preserved, since a delay fault may actually cause such a path to fail. For example, consider the function

$$z = ab + \overline{a}\,c \qquad (1)$$

An additional AND gate may be inserted for hazard protection as

$$z = ab + bc + \overline{a}\,c \qquad (2)$$

Although gate bc is redundant, a slow-to-fall fault on the gate bc may be sensitized to z by setting $a = 1$ and $c = 1$. Special remodeling was also needed for some full custom blocks to produce meaningful path delay fault patterns.

5.2 Static Timing Analysis

Paths were generated using a static timing analysis tool that produced the longest 5000 paths in the chip, as well as 5000 other long paths distributed among various parts of the chip. This strategy was adapted to target paths with little slack as well as other paths covering a wide distribution across the die. Known false paths in the design were declared as an input to the static timing analysis tool, and were not reported among the 10,000 long paths. Runtime for static timing analysis was approximately 3 days on an Sun UltraSparc 2 workstation.

5.3 Path Delay-Test Generation

Path delay fault tests were generated using an ATPG tool. Each path is tested with a two-pattern sequence as shown in Figure 5. The first pattern, applied through scan, sets up the initial condition on the path. The second pattern, applied through pulsing the system clock, launches a transition along the target path. The transition is captured on the destination scannable flip-flop by pulsing the system clock again shortly after the first clock pulse.

The ATPG tool provides an option of randomly filling the unassigned inputs for the test patterns. The random fill option was enabled when tests were generated (for the purpose of production testing), which may result in some fortuitous detection of delay or transition faults on non-targeted paths. When debugging explicit paths the random fill option was disabled and all outputs were masked except for the path destination. This proved helpful in debugging the path of interest.

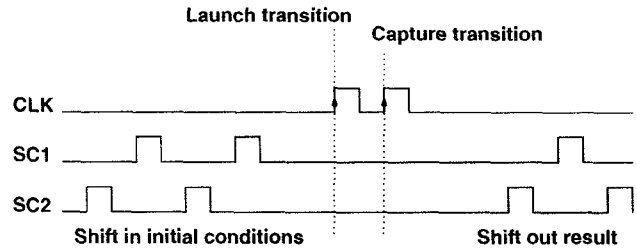

Figure 5: Clocking for Delay-Fault Testing of a Path.

5.4 Path Delay-Fault Coverage Results

Table 1 shows the statistics for path delay fault ATPG. The total path coverage is the percentage of total paths that are detected. The testable fault coverage removes the untestable paths, which include many false paths, from the total paths. Aborted paths are paths that the ATPG tool was not able to generate a test for, nor determine if they were untestable within the given time limit. Notice that each vector is a two-pattern sequence.

Table 1: Path Delay-Fault Results.

Total number of paths	10000
Detected paths	4211
Untestable paths	4831
Aborted paths	958
Total Path Coverage	42.11 %
Testable Path Coverage	81.46 %
Number of Vectors	1088
Runtime	3 days & 20 hrs

Path delay test patterns will also be fault graded for transition fault coverage to determine areas on the die that are not covered by AC tests. If the transition fault coverage is proven to be insufficient, a set of transition fault tests will be generated to cover those undetected transition faults.

6 BIST

There are 16 large full custom functional blocks in the design for which a gate level netlist was created and verified against results from a transistor level simulation. The full custom macros implement a scheduler array, a PLA, the L1 instruction and data caches, translation look-aside buffers and microcode ROMs. BIST is employed on the L1 instruction and data caches and translation look-aside buffers and the CPU microcode ROM. BIST is run during engineering evaluation and production testing. In the following subsections we describe BIST for RAM and ROM arrays.

6.1 BIST for Cache Memory Arrays

Each RAM array in the AMD-K6^{TM} processor has its own BIST controller that runs a 13N BIST algorithm [2]. At power-on the BIST controllers are initialized to a safe reset state. The *StartBIST* register in each of the BIST controllers is set using *a single* scan operation to start the BIST. The BIST execution time for each controller depends upon the size of each individual cache. However, the processor is clocked for a fixed (known) number of clock cycles that ensures the completion of BIST for the largest cache. The results are scanned out to check if all cache arrays passed BIST.

Some of the cache RAMs implement a 2-row redundancy. After evaluation it will be enabled for future revisions of the product. With redundancy enabled, the *reset emcode* runs BIST in a special mode at power-on. In this mode, the BIST controller executes the 13N algorithm in two passes. In the first pass it records up to two bad rows (if any) which are then available to the respective cache for reprogramming the redundant rows. In the second pass the BIST controller reruns the 13N algorithm on a reprogrammed RAM array to ensure the fix. Had there been three or more errors in the first pass OR any errors in the second pass, the BIST controller signals a fatal error. At the end of BIST the controller sends out a *DoneBIST* signal for the *reset emcode* to take appropriate action depending upon the outcome of the BIST. As the cache arrays can perform a read and write in a single clock cycle, the complexity of the BIST algorithm reduces to 9N.

6.2 BIST for emcode ROM

The ROM BIST controller contains an address generator and a MISR. The address generator is a standard up/down counter that generates addresses once in a forward and then in a reverse direction. This avoids error cancellation arising from diagonal errors that are sensitive to the the output MISR shift direction [7]. The ROM BIST controller's *StartBIST* is set using a scan operation. This initializes the address counter and the MISR and starts BIST in the subsequent cycle. The contents of each address are compacted into a *signature* which is then scanned out and compared with the known *good* signature. During production ROM BIST can be run simultaneously with RAM BIST using a single scan operation.

6.3 BIST Controller Implementation

The BIST controllers were implemented using standard verilog HDL and synthesized. The basic controller design was utilized for all instances of RAM BIST controllers by appropriately changing the address and data generators. The area invested for BIST is 3-4 % the area of the cache array.

7 JTAG Boundary Scan Implementation

AMD-K6^{TM} has the standard IEEE 1149.1 JTAG Boundary scan implementation on all of its I/O pins except a few reserved test and debug pins. The JTAG instruction register is 5-bits wide and supports the following public instructions: IDCODE, BYPASS, HIGH-Z, EXTEST and SAMPLE/PRELOAD. No other private/public instructions have been implemented.

8 IDDQ Tests

8.1 Design Considerations

To help minimize background currents from sub-threshold leakage when IDDQ testing, separate power planes to the large custom arrays in the AMD-K6^{TM} design can be turned off at dedicated C4 bumps that distribute power to the arrays.

8.2 IDDQ Test pattern Development

The IDDQ test pattern development for the AMD-K6^{TM} design extracts a subset of scan patterns from the set of total scan patterns based on considerations of both inter- and intra-cell shorts. Extraction of scan patterns based on the popularly used pseudo stuck-at model [1] selects patterns targeted to cover shorts within a cell by checking for the appearance of sensitizing values at the inputs of the cell(s). Our method also considers the possibility of the appearance of a short between nets in the interconnect between cells by using a list of netname pairs of candidate bridging fault sites extracted from the physical database. The netname pairs are used by the automated IDDQ pattern extraction tool by scoring a successful detection of the bridging fault site when a 1/0 difference appears on the netname pair for a given scan pattern. This analysis is performed in concert with the pseudo stuck-at model, thereby grading the patterns for effectiveness of coverage of shorts appearing between the cells as well as within the cells.

To avoid exponential runtime in extracting the candidate bridging fault sites from the physical database, all wire segments are resolved into ordered lists sorted by the regular intervals, routing tracks, into which they were routed. After collecting all segments into the ordered lists, it becomes a simple task of traversing the lists (one for the X direction, one for the Y direction) to determine which nearest neighbor segments share a common side-by-side run and report the accumulated amount of runs from all segments for the netname pair(s). The lists need to be traversed only once to extract the data for all possible netname pairs.

The gate level IDDQ fault grading method accommodates a technique to scale die area exposure to the aggregate critical area, or total amount of area occupied by nearest neighbor conductors on same levels of interconnect, and is totaled for all levels of interconnect on the silicon for the 2 nets. The value, in microns, of the total side-by-side runs of netname pairs are normalized to an average length of side-by-side polygons (driven by different sources) found within all cells in the standard cell library. The normalized value, or weight, for the netname pair is placed in the list with the netname pair and provided to the simulator. The simulator then scores a successful pseudo stuck-at detection of weight 1 and a successful netname pair bridge detection with a weight from the user supplied netname pair list. A highest sum of weights when injecting all faults yields a test pattern with highest bridging detection. If a user supplies no weights in the netname pair list, the tool defaults to a weight of 1 for each netname pair. If a user supplies no netname pair list, the tool defaults to a pseudo stuck-at fault graded result.

9 Conclusions & Future Work

The above described testability features and pattern development methodologies provide high quality structural tests to minimize manufacturing costs of the AMD-K6 microprocessor design. The scan design for testability feature implemented in the design supports: (1) static voltage level testing for wafer sort and debug testing; (2) the application of 2 pattern sequences for production testing; (3) scan based BIST; (4) JTAG. The metrics employed for pattern generation and fault simulation include, stuck-at, transition fault, path delay, and a novel IDDQ fault grading technique that incorporates physical design information in consideration of both intra-cell and inter-cell bridging faults. Future work will (a) employ algorithms for scan pattern fault grading and pattern generation of bridging and

open faults, (b) device methods for fast data retention test of the cache arrays.

References

[1] Robert C. Aitken. A Comparison of Defect Models for Fault Location with IDDQ Measurements. In *Proceedings of the International Test Conference*, pages 778–787, November 1992.

[2] Rob Dekker, Frans Beenker, and Loek Thijssen. Fault Modeling and Test Algorithm Development for Static Random Access Memories. In *Proceedings of the International Test Conference*, pages 343–352, November 1988.

[3] Chennian Di and Jochen A. G. Jess. An Effective CMOS Bridging Fault Simulator: With SPICE Accuracy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1071–1080, September 1996.

[4] Donald A. Draper, Matthew P. Crowley, John Holst, Greg Favor, Albrecht Schoy, Amos Ben-Meir, Jeff Trull, Rajesh Khanna, Dennis Wendell, Ravi Krishna, Joe Nolan, Hamid Partovi, Mark Johnson, Tom Lee, Dhiraj Mallik, Gene Frydel, Anderson Vuong, Stanley Yu, Reading Maley, and Bruce Kauffmann. A X86 Microprocessor with Multimedia Extensions. In *Proceedings of the International Solid-State Circuits Conference*, pages 172–173, February 1997.

[5] E. B. Eichelberger and T. W. Williams. A Logic Design Structure for LSI Testing. In *Proceedings of 14th Design Automation Conference*, pages 462–468, June 1977.

[6] P. Franco, S. Ma, J. Chang, Y. Chu, S. Wattal, E. J. McCluskey, R. L. Stokes, and W.D. Farwell. Analysis and Detection of Timing Failures in an Experimental Test Chip. In *Proceedings of the International Test Conference*, pages 353–361, November 1996.

[7] Yervant Zorian and André Ivanov. An Effective BIST Scheme for ROM's. *IEEE Transactions on Computers*, 41(5):646–652, May 1992.