Testing Approaches In the MC68020

John R. Kuban and John E. Salick, Motorola, Inc., Austin, TX

Functional testing of VLSI devices can require a prohibitively long engineering time, and produce poor fault coverage as well. On the other hand, fully structured testing can increase die size—impacting yield and increasing test time. In a chip as large and complex as the MC68020, functional testing is clearly impossible, and fully structured testing is undesirable. On the other hand, a judicious mixture of the two testing approaches makes the MC68020 a highly testable chip without significantly affecting die size.

This new 32-bit microprocessor, which is expected to provide four times the throughput of the MC68000, is designed to run at 16.67 MHz, and features a 32-bit data bus and cache memory (Figure 1). In order to provide a high degree of testing without unnecessarily increasing die size or test time, we chose a combination of functional testing—for the execution unit (EU)—and fully structured testing—for control logic that is usually difficult to test.

We had three goals for testing this microprocessor:

1. Keep testing development time to a reasonable percentage of actual chip design time;
2. Reduce the expense of simulation; and
3. Achieve a known fault coverage over a good portion of the chip, in order to ease the test grading burden.

We used a variety of techniques to produce structural tests of the control logic. In the first, following the natural partition afforded by the bus orientation, we separated PLAs, ROMs, and the cache from the rest of the circuitry, using some additional microcode and multiplexers (MUX). In order to cut test development time, we produced automatic test generation programs for the PLAs and cache RAM patterns.

Although the existing buses provided this natural partitioning, they were not sufficient to segregate some of the PLAs. For these, we created additional partitioning with a signature register. Consequently, these PLAs can also be tested exhaustively from this on-chip register. This signature register is the only built-in self test (BIST) on the chip.

Along with some "test management" microcode, partitioning accomplished our objectives quite well. Control section test generation seldom required whole chip simulations. Furthermore, some partitions were simulated at the system level. Finally, because our structural tests for the control section were generated algorithmically, most of that area has a known fault coverage.

The MC68020 comprises an execution unit, a micromachine to control the EU, and a cache memory (Figure 2). The micromachine has an input pipe and EU control logic implemented mostly in PLA and ROM. Either external memory or the cache feed instructions into this three-stage pipe, which consists of three 16-bit registers—IRB, IRC, and IRD.

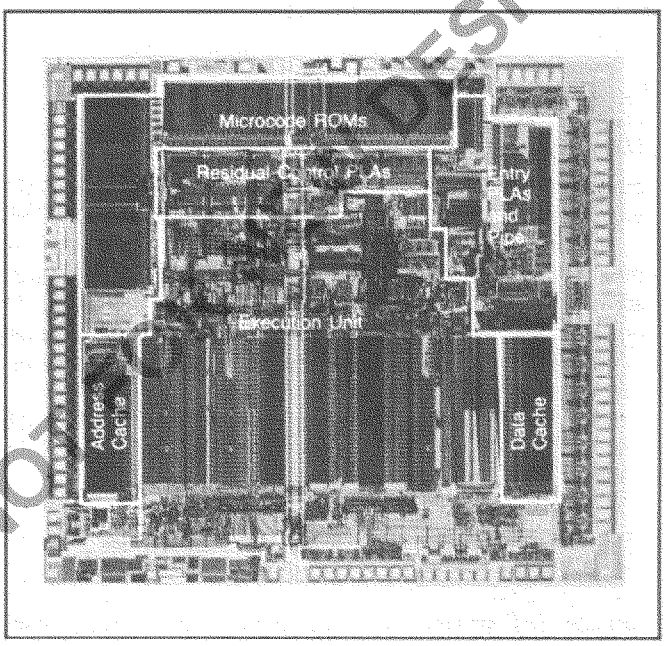


FIGURE 1. MC68020 32-bit microprocessor.

This pipe permits concurrent processing of three instructions. Register IRB receives the instruction opcode word and passes any embedded data operands into the execution unit. From IRB, the opcode moves to IRC for initial decoding by an entry PLA. Concurrently, the first extension word of the instruction—or the next opcode if there is no extension word—enter IRB. Finally, to make any subsequent microcode calls, or for residual control, the opcode moves to IRD, which holds the opcode of the instruction currently executing. A new opcode enters register IRD while the previous one executes its last microinstruction.

This instruction pipe drives four entry PLAs—A1, A2, A5, and A6—which, in turn, address the microcode ROMs. Input for A1 comes from the opcode waiting in IRC, while A1's output goes to the microROM. Thus A1 makes the first microcode access. Driven by the current opcode in IRD, A2 makes any subsequent microinstruction calls. In addition, IRC and IRD drive the other two PLAs, A5 and A6, which control microcode sequencing for the co-processor primitives supported by the MC68020. The outputs of all four PLAs are

This article is an extension of "Testability Features of the MC68020," by J. Kuban and J. Salick, appearing in the proceedings of the 1984 International Test Conference, Philadelphia, PA. © 1984 IEEE.

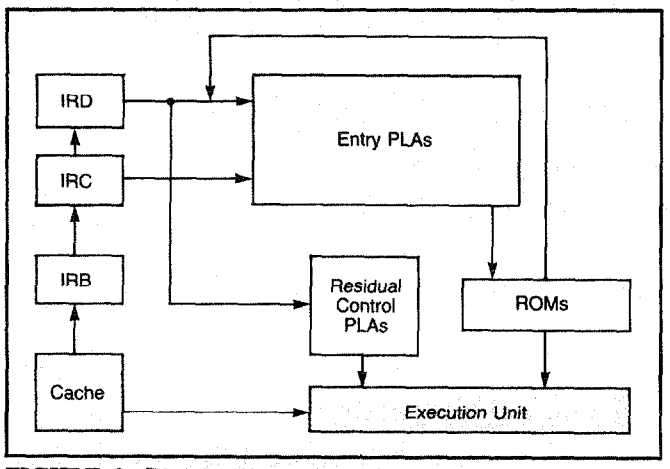


FIGURE 2. Block diagram of MC68020 micromachine.

bussed together to form the next microcode address, NMA, which is the 11-bit address given to the microcode ROMs.

Although there are logically two microcode ROMs—the microROM and the nanoROM—they reside in the same space addressed by NMA. The two ROMs have a total of 110 outputs. The 36-bit wide microROM handles the sequencing of micro operations by initiating an entry PLA access or a branch.

Micro operations take place in the EU and are controlled by the 74-bit wide nanoROM in conjunction with any residual control. Residual control involves attributes of instructions not directly related to instruction sequencing—such as operand size, condition code control, and execution register allocation. In the 68000 family, this control is generated by ecoding certain fields of instruction opcodes or extensions. Register IRD drives a set of PLAs to perform this decoding.

In the MC68020, two major PLAs handle ALU condition code and barrel shifter control, while several smaller PLAs perform the remaining tasks. The ALU and shifter control PLAs are actually three-level NOR structures with about 150 outputs. The remaining PLAs generate approximately 50 additional outputs. These output bits—along with the nano bits from the control ROM—manage the operation of the execution unit.

Because the MC68020's execution unit comprises three separate partitions, it can concurrently perform program-counter value generation, address computation, and data manipulation. Each section consists of user registers, temporary registers for virtual memory support, and an arithmetic unit. The data section of the machine includes additional structures, such as a 32 x 32-bit barrel shifter and a logical unit.

All registers are 32 bits wide and are accessed from the 32-bit address and data buses. Another bus, BC, shifts control section register values, constants, and test data into the execution unit where they can be read.

Testing ROMs

Previous schemes for testing 68000-family control structures used a special test mode in which test vectors (instruction opcodes) were fetched normally and the microcode was allowed to sequence. The address bus carried the nanoROM outputs off chip, and no execution unit activities were

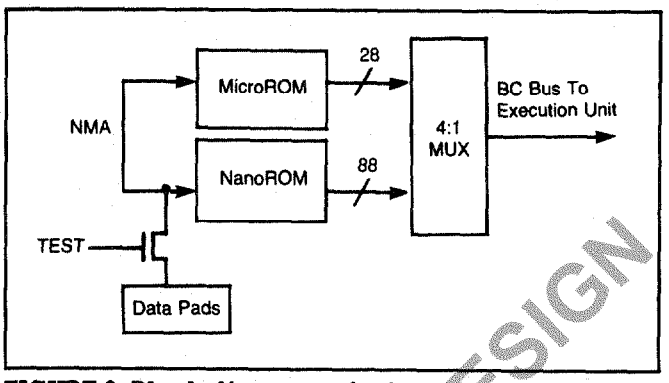


FIGURE 3. Block diagram of microcode test mode.

permitted to occur during this sequence.

This scheme, however, has several problems. Because the nanoROM output buffers are not sized to drive a bus, the tests cannot be run at maximum clock speed, requiring additional tests for speed paths. Furthermore, test results were difficult to observe. Only the nanobits were visible, which required the PLAs, microROM, and nanoROM to be tested as one large sequential machine, creating a test sequence that was lengthy and difficult to write. This test sequence was also microcode dependent and had to be rewritten if the microcode changed.

In designing the micromachine test features, we had two goals in mind. First, we sought to relieve the test generation problem. Second, in order to diminish the expense and difficulty of test grading, we wanted explicit fault coverage in this circuitry. Although our previous methods lay between functional and structural testing, these goals clearly pointed to a fully structural approach for micromachine testing.

Instead of permitting the entry PLAs to access the two ROMs, we use a test mode to force-feed the addresses to NMA. In this mode, the device is reconfigured as shown in Figure 3. Data pins 0-10 are wired directly to the NMA bus, while data pins 11-14 select a 1:4 MUX of both the microROM and nanoROM outputs. The BC bus carries the MUX outputs into the data section of the execution unit. From there, they move onto the address bus, into the address output register, and onto the output pads, where they can be observed.

In this piped operation, 32 bits of ROM output are visible every two clock cycles. Four passes of tests read the 1,10 outputs of the two ROMs. Because the inputs to the ROMs are not limited to the values generated by the entry PLAs, and because the number of ROM entries (2K) is small enough,the ROMs are exhaustively tested.

Testing PLAs

The method of testing the PLAs can be viewed in three parts: controllability, observability, and test generation. Once again, our objectives were to simplify the test generation and to obtain an *a priori* fault coverage.

The first step is to partition an individual PLA under test from the rest of the circuit. The PLAs are driven by opcodes, and we certainly do not want to run an entire instruction with each test vector. Furthermore, if we can select a PLA, feed it vectors without concern for the rest of the circuit's response, and collect its output, then we can meet both of our objectives.

Test microcode activates the test circuitry for this partitioning (Figure 4). Furthermore, in order to inhibit normal instruction sequencing, one of the functions of this microcode is to branch to itself. Two flavors of this microcode are available for A1 and A2. In one, the microcode accepts a deterministic test set from the tester. Alternatively, the other is a built-in self test, which produces an exhaustive test set from an EU register that increments with each test. On the other hand, because the inputs to A5 and A6 are a combination of IRC and IRD bits, it was not practical to implement exhaustive testing for these PLAs.

To access the test microcode, a return from exception (RTE) instruction must be executed. As the state of the machine is being reloaded, NMA is set to point to one of the test microcode sequences, and an EU register is set to select the PLA under test. In addition, in the BIST mode, the EU incrementing register is initialized.

Observability is quite different for the entry and residual control PLAs. The former—with only 11 outputs—lent themselves to data compression by signature analysis. Using the cyclic redundancy check (CRC) polynomial, a 16-bit signature register compresses these outputs (Kuban and Bruce 1983). This signature is read after each of the four PLAs is tested.

On the other hand, residual control PLA testing does not use data compression. Normally, the outputs of the two major residual control PLAs pass through a 7:1 MUX, which reduces their number to about 75. In the test mode, another 4:1 MUX further reduces the output width of all of the residual control PLAs to that of the bus. These outputs are brought off chip by the same path as used for testing the ROMs.

We test these PLAs in parallel, and observe the multiplexed outputs. Thus, for each test, we see 1 of 28 outputs of the major PLAs and 1 of 4 outputs of the remaining residual control PLAs—a situation that presents a significant problem for test generation.

Initially, we develop a test set for each PLA. After concatenating these sets, we sort them into 28 groups, and subsume within groups, to minimize test time.

Motorola has developed a test-pattern generator for PLA-like structures of an arbitrary number of levels, each of which can be any of the four main gate types. This test generator comprises two major components—the global algorithm and gate-level test generator.

In this test generator, we model the logic array as levels of gates, where each level consists of only one type of gate. Each gate has n inputs, where n is the number of transistors on that line. Starting at the first level and working to the last, we generate the $n + 1$ tests for each gate of a given level. As each test is generated, we attempt to subsume it under a previous test to reduce the total number of tests.

Gate tests are generated by setting the input under test to the appropriate value, while all other inputs to this gate are set to the noncontrolling value. Provided this is not a level-one gate, then previously accumulated data permits easy backward propagation to the primary inputs. Following this, the test generator evaluates the gate output.

Leaving all unspecified primary inputs as "don't care," we attempt to propagate the fault to a primary output. Because the array has only a few levels, the test will often propagate

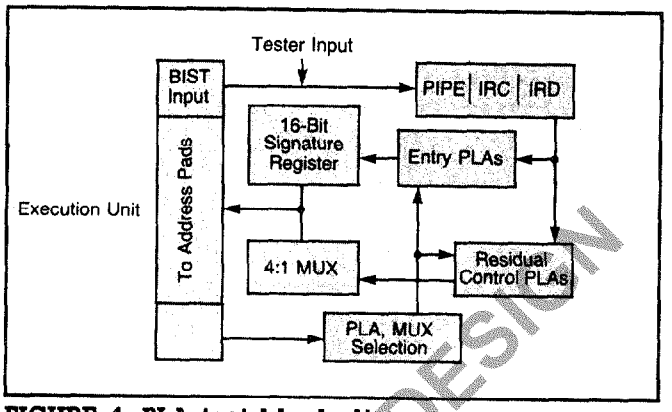


FIGURE 4. PLA test block diagram.

with "don't cares" intact. If the test does not propagate, we begin setting other primary inputs. At the start of the run, we select whether this will be done exhaustively or up to some number of random tries.

Initial use of the test generator on MC68020 logic arrays shows it produces concise patterns. Furthermore, fault simulation of test cases using these patterns confirms its accuracy.

Recently there has been some question as to the validity of the single stuck-fault model in PLA testing. Most of the PLA test literature uses the more general cross-point defect (CPD) model, which stems from the structure of the array. There are two possible defects at each of two types of cross points—spurious absence or presence of a transistor in the input or output arrays. Several researchers have proposed automatic test-pattern generation algorithms for this fault model (Smith 1979; Bose and Abraham 1982).

We have not found evidence that indicates a need for modeling added transistors. Because the stuck-fault model covers missing transistors, we feel it is an adequate model. The ability to run deterministic and exhaustive tests on this part will provide a means for evaluating the stuck- fault model for future PLA testing.

Testing the Cache

The MC68020's on-chip cache is a set-associative memory that stores instructions and extensions (Figure 5(a)). The function codes distinguish between program code and data and the latter is not cached. For cache entries, each memory location is represented as an element of one of 64 mutually disjoint sets. Bits 2-7 of the memory address determine set membership.

The cache consists of two fields—tag and data. The tag holds the upper 24 bits of an address in its set, while the 32-bit data field contains the contents of this location and three following it. The lower bits of the program counter (PC) are decoded and point to a tag and to the corresponding data-field entry. If the upper 24 bits of the PC match the tag, the opcode is moved into IRB and subsequently executed. Otherwise, an external fetch is made.

In order to simplify the hardware, the MC68020's replacement scheme calls for the accessed entry to be written when a miss occurs. Specifically, when a miss occurs, the opcode fetched from external memory is placed in IRB and in the cache. This forms the basis of the cache test scheme. If an access misses, the cache is written; if it hits, the cache is read (Figure 5(b)).

Upon entering cache-test mode—by means of an RTE instruction—a fetch is made. Input data is forced into a program-counter register to be used for the next cache access. If this access misses, the tag is written and the tester supplies another vector, which is written into the cache data field. This loop starts again with another vector from the tester. When the access hits, we read the data field.

This sequence allows reading and writing cache arrays independently of any instruction execution. More importantly, we can apply any pattern to the arrays, making RAM-type testing possible. A judicious choice of RAM patterns provides full testing of each cache location. This includes testing the tag, data field, and comparators.

Testing the EU

The Execution Unit is inherently controllable and observable, but at the expense of the overhead involved with instruction execution. Thus the problem is one of test generation.

No logic or microcode was added to assist in testing the EU or the random logic of the control unit. However, the structured testing of the residual control PLAs greatly simplifies EU testing. Functional testing of the EU and control sections normally requires testing each instruction under several different conditions (such as addressing modes, operand size, and condition-code settings). This testing is minimized since the PLAs that determine these conditions have already been tested.

Conclusions

Partitioning the sequential micromachine substantially saved time and money. Where possible, we used automatic test generation and simulated modules, rather than the entire chip. Our ROM and PLA test generators used system-level functional definitions. The module simulation model for these tests consisted of the array being tested and either its output MUXes or the signature register.

We developed the cache RAM patterns with a full-chip simulation of only four cache accesses. Our chief concern in these simulations was to establish bus timing.

We used test generation programs to produce structured tests for the micromachine. These test patterns took less than three man-months to generate. The bulk of that time was spent developing the necessary software. We estimate that generating functional tests for the micromachine could require as many as four man-years to achieve similar fault coverage. Also, when microcode changes do occur—as they inevitably do—new tests will be easier to generate.

Fault grading can be one of the most expensive activities in designing a chip. The tests generated algorithmically for ROM, PLA, and cache RAM have a known fault coverage without fault simulation. ROM is 100% tested, while PLAs are tested for all detectable faults, and the cache is tested with standard RAM patterns.

Because the MC68020 is to be a high volume production device, overall die size needed to be minimized. Sharing test routing with existing routing kept chip area to minimum. The entire test logic—16-bit signature register, MUXes for the ROMs and PLAs, routing, and "test-only" logic—accounted for less than 3% of the total chip area. Furthermore, using general "test management" microcode, rather than actually

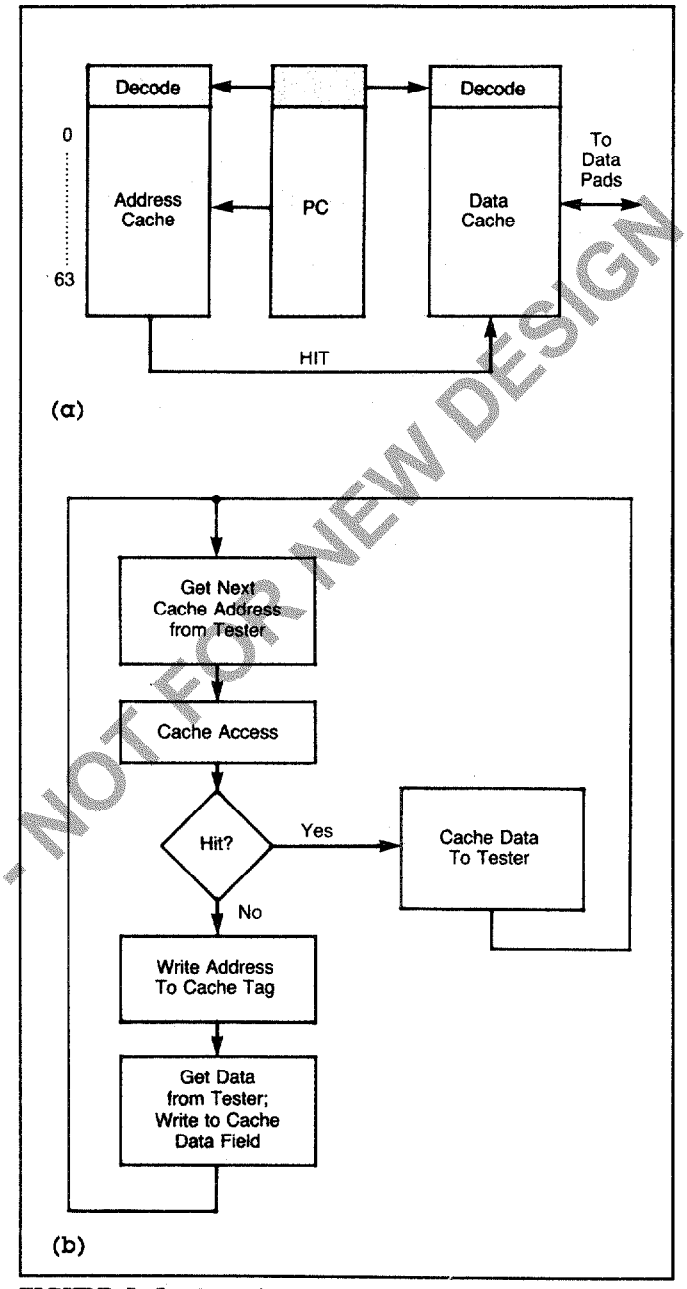


FIGURE 5. Instruction cache block diagram (a) and test flow chart (b).

generating test cases or test conditions, minimized our ROM requirements. We use less than 2% of the total code, while maintaining needed flexibility.

Test partitioning reduces the micromachine—a huge sequential test problem—to a number of combinational blocks. Inputs to these combinational blocks typically can be stimulated from the data pads or from easily accessible registers. Outputs are generally redirected to a bus and then on to primary outputs. Test generation for these combinational blocks can be automated.

This approach achieved the same goals as scan design over the problem areas of the chip. However, scan tends to increase die size far more than does our method (Williams and Parker 1983). The effect of increased die size on yield loss is well known (Murphy 1964).

The disadvantage of our design *vis-a-vis* scan is the time needed to produce *ad hoc* test logic. Due to the bus structure of the part, however, it was relatively easy to create test partitioning for the embedded structures. We feel that the expected higher yield justifies this extra time. Nevertheless, this trade-off must remain a crucial question in the commercial semiconductor industry.

The addition of the residual control test feature dramatically reduced the amount of functional testing required for the EU. If only functional testing had been used for this section, an additional six man-months would have been required to achieve the same fault coverages. Similarly, if previously used partitioning schemes had been used for the micromachine, another six man-months would have been required. Furthermore, unlike functional testing, our vectors are easy to generate and have a known fault coverage. The disadvantage of our methods is that the tests require more vectors—and thus more test time.

Although the actual production test time has not been established, indications are that even with exhaustive PLA testing, test time will be acceptable. The savings in test generation costs and the known fault coverage more than outweigh the disadvantage of increased test time. □

References

Bose, P. and J.A. Abraham. 1982. "Test Generation for Programmable Logic Arrays," *19th Design Automation Conference*, Las Vegas, NV.

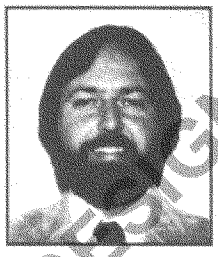
Kuban, J. and B. Bruce. 1983. "The MC6804P2 Built-In Self Test," *IEEE International Test Conference*, Philadelphia, PA.

Murphy, B.T. December 1964. "Cost-Size Optima of Monolithic Integrated Circuits," *Proceedings of the IEEE*.

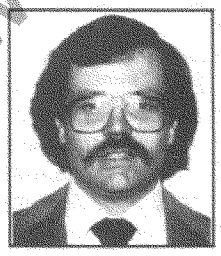
Smith, J.E. November 1979. "Detection of Faults in Programmable Logic Arrays," *IEEE Transactions on Computers*.

Williams, T.W. and K.P. Parker. January 1983. "Design for Testability—A Survey," *Proceedings of the IEEE*.

About the Authors



John R. Kuban is a senior staff engineer with Motorola, Semiconductor Products Sector, MOS Microprocessor Division, Austin, TX. During his four years at Motorola as a member of the Design Test Group, he has been involved with the testability features used on several Motorola designs, including the 6804 and MC68020. His current interests include testability of commercial VLSI, BIST, and CAD test tools. Kuban received his BS in Electrical Engineering from the University of Texas at Arlington in 1974.



John E. Salick is a staff engineer with Motorola, Semiconductor Products Sector, MOS Microprocessor Division, Austin, TX. His current interests include testability of commercial VLSI, PLA testing, signature analysis, CAD test tools, adventure games, and his Macintosh computer. Salick received his BA in Journalism from the University of Wisconsin in 1978 and his BS in Electrical Engineering from the University of Texas at Austin in 1983. He is currently pursuing an MS degree in Electrical and Computer Engineering at the University of Texas.