CGaAs PowerPC FXU

Alan J. Drake, Todd D. Basso, Spencer M. Gold, Keith L. Kraver, Phiroze N. Parakh,
Claude R. Gauthier, P. Sean Stetson, and Richard B. Brown
University of Michigan
1301 Beal Ave
Ann Arbor, MI 48109
(734) 763-4207
ajdrake@umich.edu

ABSTRACT

The development of a PowerPC™ fixed-point execution unit (FXU) in a resource limited, radiation-hard technology is described. Detailed architectural studies led to a design which maximizes performance in a small transistor count implementation. Manufactured in Motorola's 0.5–µm Complementary Gallium Arsenide process, the device operates from 0.9 to 1.9 V with a nominal frequency of 25 MHz at 1.3 V, dissipating 274 mW.

Keywords

Gallium Arsenide, Microprocessors, Design Methodology, Testing Methodology

1. INTRODUCTION

The loss of the Galaxy IV satellite in May, 1998 demonstrated the increasing dependence of our society on space systems and the magnitude of the disruption when those systems fail. Communications with the satellite were lost when the navigation system failed, disabling pagers, credit card transactions, and TV satellite feeds. Components must be resistant to the harsh space environment in order to prevent recurrences of the Galaxy IV debacle. It is estimated that thousands of new communications satellites will be launched over the next few years, fueling a need for more space-tolerant components [2]. To this end, the PUMA processor, an MCM-based, multi-chip PowerPC implemented in the Motorola Complementary Gallium Arsenide (CGaAs™) process, was designed at the University of Michigan. This paper will briefly describe CGaAs technology and then present the development of the low transistor-budget Fixed Point Unit (FXU), which is the nucleus of the PUMA project. The development, verification and test environments for this exotic technology are described, and test results are presented.

2. CGaAs OVERVIEW

The Complementary Gallium-Arsenide process provides high electron mobility of the undoped Indium Gallium Arsenide channel, as well as a P-device for low power complementary designs. The three-layer metal (Al) process produces threshold voltages of ±0.55 V and allows logic gates to operate at voltages from 0.9 to 2.0 V. In addition, CGaAs is radiation hard, with resistance to single event upset (10^{-10} upset/bit-day for complementary logic), total dose radiation (more than 10^8 rads), and latchup (more than 10^{12} rads).

CGaAs has some design challenges, including higher gate and source-drain leakage currents than comparable CMOS processes [3]. As in CMOS, many logic styles can be realized in CGaAs, including Complementary, Source-Coupled FET Logic (SCFL), Domino, Dual-Rail Domino, Differential Cascode Voltage Switch Logic (DCVSL), and Pseudo-Direct Coupled FET Logic (P-DCFL) [3]. We evaluated Domino logic through the design of a PowerPC ALU [1], and DCVSL through the design of a multiply-accumulate unit [7]. Because of its low power dissipation, excellent radiation hardness, and compatibility with available design tools, we selected Complementary as the main logic family for the PUMA FXU. P-DCFL logic was used in the SRAM address decoders because of its superior speed. The greatest challenges presented by CGaAs are the high threshold voltages (which limit speed), and the comparatively low integration level of this immature process, which provided the impetus for the architectural study.

3. ARCHITECTURE

The architectural studies described in this section were performed in the early stages of the PUMA project. At the inception of the project, the CGaAs roadmap predicted that million-transistor circuits would be practical by the time the PUMA chips were fabricated (a small transistor budget by CMOS standards). However, over the course of the project, process development delays due to a lack of investment in needed equipment reduced this estimate. The FXU that was finally implemented consisted of only 383,000 transistors. The studies described in this section evaluated various aspects of the microarchitecture, with the objective of optimizing the performance of processors that are constrained to implementation using comparatively few resources.

The PowerPC instruction set was modified slightly for this implementation; the 133 PUMA instructions include most PowerPC arithmetic operations (add, subtract, shift, rotate, logical and compare), all of the branch instructions, load and store byte, half-word and word instructions, load and store with update, and cache management instructions. Floating-point, integer multiply/divide, 64-bit arithmetic operations, load-store-multiple, and string instructions are not supported in this small-budget processor.

Some PowerPC instructions perform multiple operations and/or modify several registers. To avoid the control complexity, additional register file resources, and long critical paths that this would impose, the PUMA processor translates complex instructions into simple unit operations, each of which executes one operation and modifies at most one architectural register. This approach is used in modern X86 machines to execute the native CISC instructions on a RISC core. In the PUMA processor, the unit operations keep the processor simple and help balance the critical paths, while enabling this small processor to execute complex instructions.

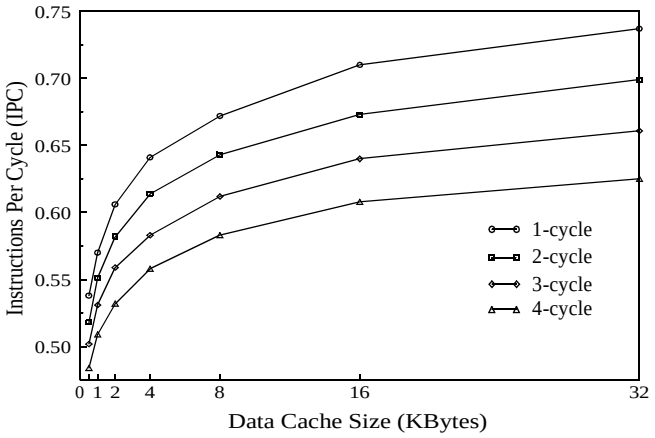


Figure 1: Effects of primary data cache latency on overall performance.

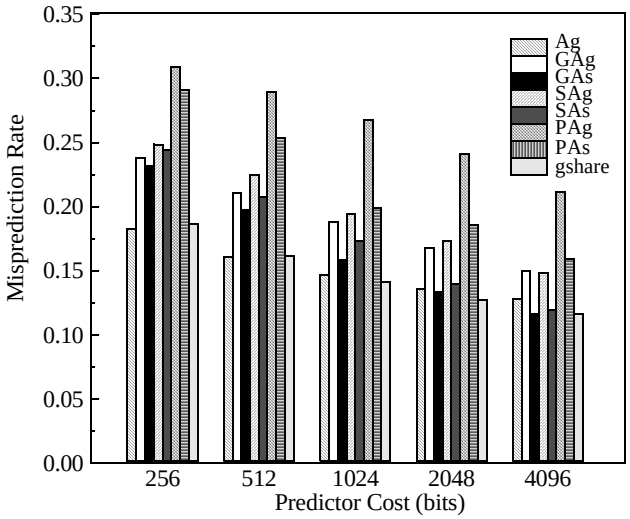


Figure 2: Effects of predictor cost on misprediction rate.

3.1 Simulation Methodology

A generic superscalar microprocessor similar to the one described by Johnson [6] served as the baseline model. This baseline processor model is two-way superscalar, with on-chip 4 K-B instruction and data caches. The baseline fetch mechanism neither prefetches instructions nor predicts branches; performance improvement of this model over a scalar machine is based solely on the ability of the superscalar execution engine to extract parallelism from the instruction stream.

A simulation environment was developed to facilitate rapid evaluation of architectural features. Johnson's superscalar simulator, ssim [6], was used as the foundation for a cycle-level simulator. Extensive modifications were required to convert the simulator from MIPS to PowerPC architecture, and additional extensions were added, such as instruction prefetch and branch prediction. The Spec95 integer benchmark suite was used as the basis for evaluating performance. This simulation environment was used to study the performance effects of various cache parameters (size, associativity, latency and line size), stream buffer size, superscalar width, reservation station size, reorder buffer size, and various sizes and types of dynamic branch predictors. Brief discussions of cache and branch prediction analysis are included here to illustrate this aspect of the design flow.

3.2 Architectural Analysis

The key features of the micro-architecture were simulated independently to determine how they would affect the performance of the baseline model; the metric for comparison was execution rate, as measured in instructions per cycle (IPC). Only enhancements that could be implemented with a relatively small number of transistors were considered.

3.2.1 Data Cache Optimizations

The simplest method of improving memory performance is to make the data cache as large as possible, but constraints of the CGaAs technology prohibit the implementation of a large on-chip data cache.

Figure 1 plots the performance of the baseline machine with various data cache sizes and access latencies. An on-chip data cache configuration will have a single-cycle latency, whereas an off-chip data cache may have a three-cycle latency: one cycle each for transmitting the index, accessing the array, and returning the data. From these simulations, a general rule is seen; to maintain a given level of performance, the data cache must double in size for every additional cycle of latency incurred. An off-chip data cache must be 16 KB to provide the same performance as a 4 K-B on-chip cache. Microprocessors that are unable to incorporate a data cache on the main processor die can place the data cache off-chip provided that the cache is made large enough to compensate for the additional communication latency.

3.2.2 Dynamic Branch Prediction

Branch instructions are control dependencies that must be resolved to determine the correct path of instruction execution. The delay between instruction fetch and resolving the control dependency is referred to as the branch delay. This delay may result in several cycles of inactivity, or wasted execution slots, in a pipelined microprocessor.

While the branch problem can be solved with enough hardware, such an approach is impractical in a small microprocessor design. Figure 2 shows the relationship between prediction accuracy (in terms of misprediction rate) and predictor cost. The configurations studied range from a simple one-dimensional array of two-bit counters [9] (Ag) to the more complicated two-level adaptive schemes [10]. The one-level scheme outperforms all two-level schemes in very small predictors (below 1 K-bits). However, for configurations above 1 K-bits, the two-level schemes are generally superior. A 1 K-B predictor implementing global sharing (gshare) [8] can achieve about 90% accuracy, making a simple and efficient enhancement to a small microprocessor design. The gshare predictor is basically a GAg scheme (global branch history and global pattern history) where the Pattern History Table (PHT) is indexed using a hash of the program counter and the Branch History Register (BHR). GAs (global branch history and per-set pattern history) and SAs (per-set branch history and per-set pattern history) also achieve approximately 90% prediction accuracy, but they require multiple pattern history tables, complicating the implementation [1].

3.3 Architectural Summary

The architectural studies guided the development of the CGaAs PUMA micro-architecture to an efficient (in terms of instructions per cycle/million-transistors) implementation. A CMOS prototype of the PUMA FXU was implemented which incorporated on-chip instruction and data cache, branch prediction, out-of-order execution, and prefetching. In order to stay within the transistor budget in the CGaAs version, neither instruction prefetching nor branch

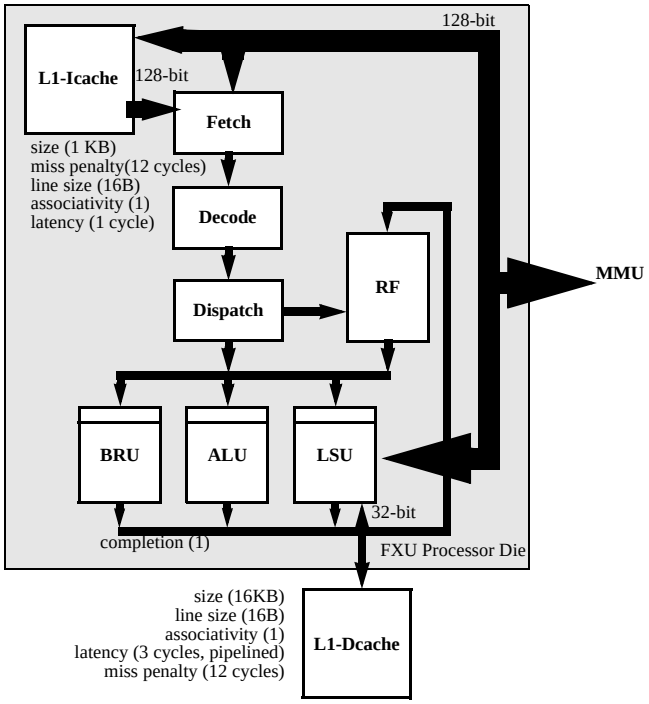


Figure 3: Processor block diagram.

prediction were included, a simple execution scheme was employed, and the data cache was moved off-chip. The on-chip, single-cycle instruction cache is 1 KB in size, and the off-chip 16 K-B data cache has a three-cycle memory access latency, the effects of which are minimized through the use of a pipelined load-store unit.

A block diagram of the CGaAs PowerPC processor is shown in Figure 3. Instructions fetched from the L1 I-Cache or from off-chip memory are cracked into unit operations in the decoder; most PowerPC instructions remain single-word, single-cycle instructions, but some become multiple unit operations. Only the 'rotate and mask' instructions are longer than two unit operations. The dynamic instruction count growth, based on the Spec95 integer benchmark suite, was 15%. In a PUMA processor that could issue multiple instructions per cycle, the superscalar core would be able to extract parallelism from the stream of unit operations, resulting in a degradation in overall performance due to this code growth of only 2.8% [1]. To stay within the CGaAs transistor budget, the pipeline was implemented with a single-issue in-order execution policy, which is not able to take advantage of this parallelism.

The instruction cache, fetch unit, and load-store unit (LSU) are connected to the memory subsystem by a 128-bit data bus. The Load-Store Unit communicates with the off-chip primary data cache over a 128-bit bus (a full cache line). Despite the Micro-architecture limitations, the CGaAs PowerPC achieves an execution rate of 0.50 instructions per cycle.

4. DESIGN METHODOLOGY

The processor design and implementation were carried out by six graduate students in less than one year. The team created a gate-level RTL model of the processor and ran exhaustive random tests to verify the design. The team developed a library of CGaAs standard cells with which the chip was designed, and assembled them using automated placement and routing tools.

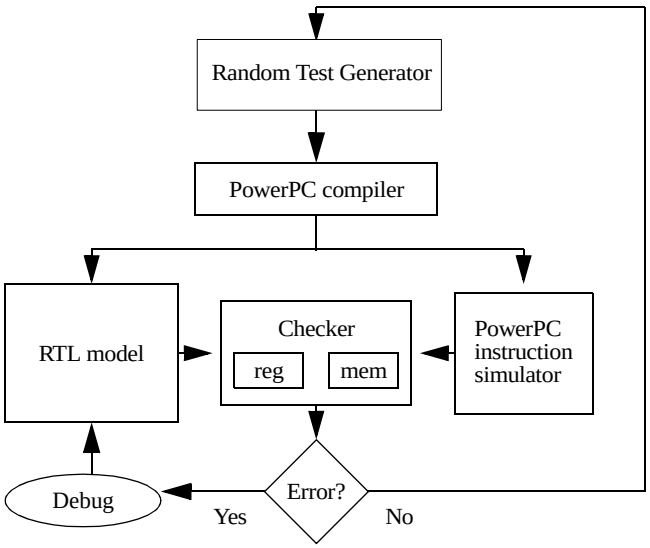


Figure 4: RTL verification environment.

4.1 Verification Environment

An automated verification environment was developed based on a PowerPC instruction simulator [4] and a random test generator. The verification environment is pictured in Figure 4. The random test generator (RTG) can be parameterized, allowing the user to specify the test length, number of branches, and loop structure. Other options build on the basic program structure. For example, the RTG can generate tests that focus on certain instruction types, restrict register usage to create more data and naming dependencies, and allow a percentage of misaligned memory accesses and unimplemented instructions in the random instruction mix to stress the exception-handling mechanisms. The RTL model of the design was simulated in parallel with the PowerPC instruction-level simulator. The checker compared the architectural register values and memory contents as each instruction was retired.

This environment was first used to verify a highly abstract behavioral model of the processor. Over time, the RTL was distilled into a gate-level RTL representation of the design. The verification environment allowed some of the functional blocks to be behavioral while others were gate-level models. This verification environment was used to simulate the execution of over two billion instructions.

4.2 CAD Tools

The CAD environment was customized around layout and extraction tools from Mentor Graphics, and the Epoch™ design compiler from Cascade Design Automation, which was used for library generation, static timing analysis, and automated placement and routing.

4.2.1 Standard Cell Library

HSpice™ was used to determine the optimum transistor sizes for logic gates having one to four inputs. Using this information, Epoch generated a basic CMOS cell for all gate types and desired drive strengths. IC Station™ from Mentor was then used to modify the layout for CGaAs compliance. The PUMA CGaAs standard cell library consists of thirty full-complementary logic gates, including most boolean functions of two and three variables, assorted complex functions, multiplexors, standard buffers, tri-state buffers, and flip-flops. Most gates are available in drive strengths from 1x to 6x, while the inverters, standard buffers, and

tri-state buffers are available with drive strengths of up to 64x. The standard cell height was fixed at 48.05 µm, corresponding to the vertical dimension of a 6x inverter. The width varied depending upon the complexity of the logic gate. Care was taken with commonly-used cells such as the D flip-flop to ensure that the width was kept to a minimum. Gate delays ranged from 137 pS to 676 pS, with a typical gate delay being approximately 350 pS.

4.2.2 RAM Compiler

An optimizing RAM compiler [5] was developed to generate SRAM macrocells. This tool was designed to be process-independent, allowing it to quickly adapt to design-rule changes and device model updates. Optimized SRAM macrocells were generated by this tool in two phases. First, a spice-based, heuristic-guided parallel optimization algorithm was used to iteratively explore transistor sizes. This algorithm provides the data necessary to define the achievable power-delay design space for a user-specified SRAM configuration given a specific set of layout design rules and device models. This design space defines the minimum power dissipation that can be achieved for a range of operating frequencies. MasterPort™, a layout compaction tool from Cascade, and commercial extraction tools were used to generate optimized leaf cell layout masks and provide their parasitics to the optimizer.

In the second phase, the optimized SRAM layout is generated. To accomplish this, the user selects a point within the achievable power-delay design space for implementation. The tool then generates the leaf cells that correspond to this design point and tiles them into the user-specified array configuration. The tool performs power rail sizing and automatically generates corresponding spice netlist and GDSII layout files.

4.2.3 Place and Route

The standard cells and memory modules were imported into the Epoch design environment, where the Epoch tools performed automated placement and routing of these blocks. The Floorplanner tool was used to manually optimize the top-level of the design. The automated routing tool was then used to route the top-level and size the power rails appropriately.

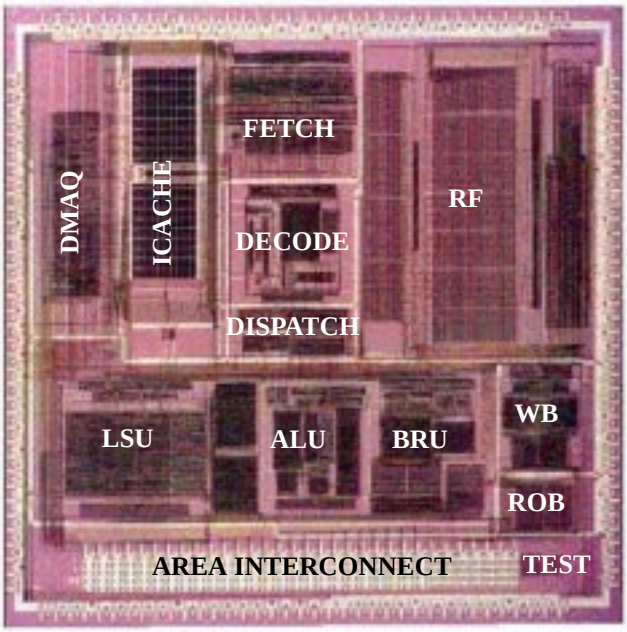


Figure 5: Photo of 13.1 x 11.4 mm CGaAs FXU die, showing 288 peripheral pins (228 I/O) and 250 area I/O pins.

The entire clock grid was extracted and simulated using HSpice. The clock buffers were manually sized to equalize the clock skew at each functional block with the chip. The manufactured chip is shown in Figure 5.

4.3 Testability

Due to low integration levels, test structures had to be efficient. Furthermore, there were few I/O pads to devote to testing. A test block, three scan paths, an instruction cache disable, and a data cache disable were added to the design to maximum testing with a minimum amount of hardware.

The test block consists of a pad ring, a 32-bit shift register, an inverter ring-oscillator, a NAND gate ring-oscillator, and a clock tree output. These units facilitate testing of the clock, I/O pads, basic gates, and registers independent of the microprocessor.

The cache disable lines force the microprocessor into a miss condition on every instruction and data access. This puts the microprocessor into a state in which its operation is easily predicted, and allows testing to be performed independently of the caches.

Complexity, controlability and observability were the primary considerations in deciding which functional units would have scan paths. Since the number of paths to be supported was limited, only the decoder, dispatch unit, and ALU were given scan paths. These functional units are the most complex and most critical to the operation of the microprocessor. The remaining functional units are not scannable because they are fairly simple, consisting of registers and a small amount of logic.

These test structures provide an incremental test flow that begins with device level testing, moves to testing the microprocessor in a very simple state, and finishes with detailed testing of the full ISA and scan paths.

5. TESTING

FXU testing was performed on an HP82000 IC Development System which has 240 individually configurable test channels. To use the tester, it was necessary to package the FXU for testing and then generate test vectors. Each of these issues provided interesting challenges.

5.1 Packaging

Testing the FXU was a potentially expensive proposition. As shown in Figure 3 there are two off chip buses, one to the Memory Management Unit (MMU) and the other to the L1 data cache. The MMU interface is routed through the peripheral pins, while the L1 cache interface is made through an array of bumps that would be flip-chip connected on an MCM. The total number of I/O pins (peripheral and area interconnect) exceeded the capabilities of the HP82000. Rather than purchase an expensive die interface with too many pins for the tester, we decided to initially package the FXU in a pin grid array package and test everything but the data cache interface. Data cache testing would have to wait for system level testing. The FXU was packaged in a 391 pin, cavity-down PGA purchased from Kyocera Corporation. Ten chips were packaged by Norsk Engineering. A packaged FXU is shown in Figure 6. A Device Under Test (DUT) board was also purchased for the HP82000.

5.2 Test Vector Generation

The test environment consists of the hardware performing the testing and the software used to generate the tests. The generation of test vectors was a daunting task. The external FXU pins consist of a 30-bit address bus, a 32-bit data-out bus, a 128-bit data-in bus,

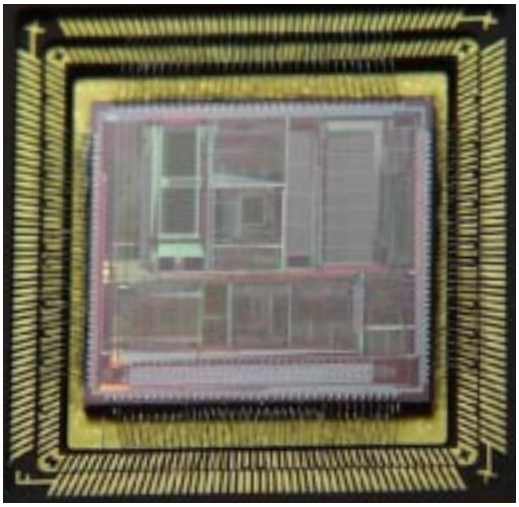


Figure 6: Packaged FXU

and various other control signals amounting to 228 I/O pins. Manually entering test vectors was impossible and random generation was difficult because input vectors had to be legitimate instructions from the ISA. Fortunately, the verification environment described earlier provided the answer to test generation.

The TDS Software System by Fluence Technology (formerly Summit and TSSI) converts vector files from various simulators to test vectors usable on many digital testers, including the HP82000. The generation cycle is shown in Figure 7. The verification environment was modified to dump FXU I/O line changes to a Verilog variable change dump (VCD) file. TDS converts the VCD file to a vector file usable by the HP82000. Our HP82000 has 80 400-MHz channels and 160 200-MHz channels. TDS supports only the 100-MHz mode of the 200-MHz channels. Since the buses were mixed between 200-MHz and 400-MHz boards, they all had to be run in 200-MHz mode. A Perl script was written to convert the TDS generated vectors into the format required by each channel, making the output compatible with our HP82000 configuration.

This test process provided flexibility to quickly generate both simple and complex tests, providing a unified environment for all levels of testing.

5.3 Test Cycle

Testing began with process and device characterization, which was done by Motorola. Next, tests were written manually on the HP82000 to verify the I/O pads, ring oscillators, clock tree, shift register, and a reset of the FXU. The next group of tests were written using the test environment to verify each of the functional units with the caches disabled to reduce the amount of hardware being exercised. These tests included: a string of no-ops to verify pipeline execution; immediate adds followed by stores to test the registers; loads and stores to verify the load/store unit; adds, shifts and subtracts to test ALU functionality; and branches to check the branch unit and special purpose registers. Next, the voltage/frequency characteristics were tested by exercising the critical path found from simulations. This path is the branch target address calculation. A test was written in which a conditional branch must be taken with a negative offset, forcing a full 32-bit add, the worst case scenario. Power characteristics were measured with a test that exercised stores, loads, branches, and ALU instructions in an infinite loop. The last test involved turning on the instruction

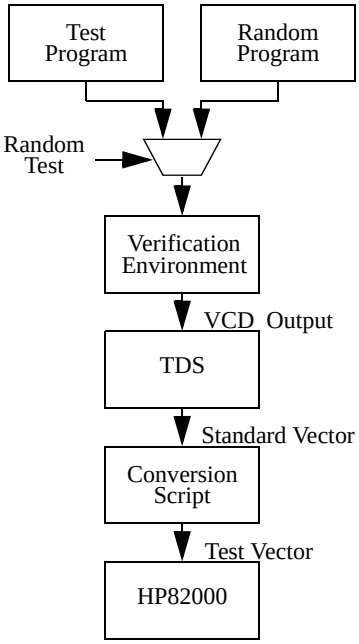


Figure 7: Test Generation Cycle.

cache and re-running the branch tests. Additional cache tests were done on a 2 K-B SRAM chip fabricated on the same process run as the FXU.

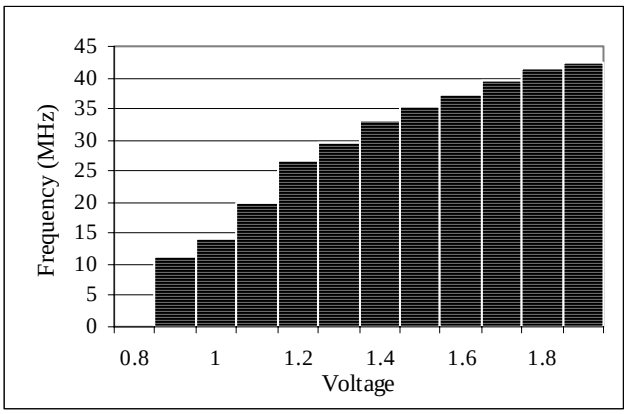
5.4 Results

For a first run in a new technology, on a chip that was pushing the integration levels for that technology, the results are remarkably good. Of the ten microprocessors packaged, all of them had functional test blocks but only two passed the no-op instruction tests. The most common problem with the failing chips was incorrect address sequencing.

The two chips that passed had varying degrees of success with the remaining tests. None of the devices passed all the tests completely. In general, the data output bus, which stores data to memory, was unpredictable, with some bits oscillating and others not switching at all. Functionality of most units in the FXU had to be determined by monitoring address sequencing. It was determined that immediate instructions and program address sequencing worked; however, instructions that manipulate register data failed. Functionality of the ALU, load/store unit, and the branch unit can be inferred from these tests.

The branch instructions worked properly. Using branches, the critical path of the FXU could be tested, despite unpredictable output data. Figure 8 shows the results of voltage vs. frequency testing performed using a sequence of branches. The FXU operated at a maximum frequency of 42 MHz at 1.9V. These results must be viewed with some reservations, since only the branch unit's critical path could be measured with certainty.

Figure 8 shows the power curves at varying frequencies for both chips. There is not much difference in power between operating frequencies, indicating that most of the power is dissipated as static power. The core dissipates 18% of the power, with the remainder being dissipated in the pads. At a nominal operating voltage of 1.3 V, the FXU can be run optimally at 25 MHz, dissipating 274 mW.



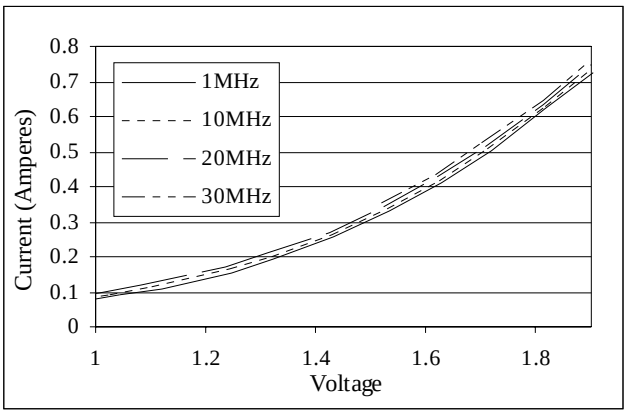


Figure 8: Voltage-Frequency and Voltage-Current Plots.

None of the devices passed the instruction cache tests, indicating non-functional caches. More detailed cache testing was performed on a separate 2 K-B SRAM chip. It used the same SRAM design as the FXU caches. These chips also failed; data-out always followed the data-in, indicating that the decoder was not working correctly. The decoder uses P-DCFL NOR gates. The ratios of these gates were not sufficient to provide a low enough output low over process corners, which proved much wider than anticipated. Process data showed that the beta values, drive currents, and leakage currents of the N and P transistors, as well as the threshold voltage of the P devices, had a much wider distribution than anticipated. The variation of P device threshold indicated by process data could also explain the other testing failures. Leakage currents would be higher, and some gates may not turn off at all, adding to static power and data errors. Further testing of the scan path and circuit simulations with the measured process corners should help identify the exact problems.

6. CONCLUSION

In this paper a radiation-hard, low-transistor-budget microprocessor has been presented. Commercial CAD tools were used in conjunction with university-generated tools to optimize the architecture, verify the design, implement the layout, and generate test vectors. Compromises had to be made to stay within the CGaAs integration level, that left some architectural features underused. Wide variation in transistor parameters on this immature process rendered some circuits in the prototype non-functional. Still, it was possible to verify functionality of two of ten chips, which run optimally at 25 MHz and 1.3V. The architecture is currently being translated to SOI for circuit studies.

7. ACKNOWLEDGMENTS

The authors gratefully acknowledge the funding for this project provided by DARPA under contract ARO DAAH04-94-G-0327, and thank Motorola Semiconductor for fabricating the FXU and providing the information which facilitated the PUMA design. We thank Cascade Design Automation, Mentor Graphics, MetaSoft and Fluence for the use of their CAD tools, which were critical to this project.

8. CONTACT INFORMATION

Todd Basso, Spencer Gold, and Claude Gauthier are currently at Sun Microsystems and may be contacted at tbasso@East.Sun.com, sgold@East.Sun.com, and Claude.Gauthier@Eng.Sun.com. Phiroze Parakh works for Monterey Design Systems. Sean Stetson is at Texas Instruments and can be contacted at stetson@ti.com. Keith Kraver and Richard Brown are both at the University of Michigan and may be reached at kkraver@umich.edu and brown@umich.edu, respectively.

9. REFERENCES

[1] T. Basso, “A Micro-architecture for Resource Limited Superscalar Microprocessors”, 1999 Ph.D. Dissertation, University of Michigan, pp. 22-23.

[2] J. Benedetto, “Economy-Class Ion-Defying ICs in Orbit,” *IEEE Spectrum*, vol. 35, no. 3, March 1998, pp 36-41.

[3] R. Brown, B. Bernhardt, M. LaMacchia, J. Abrokwah, P. Parakh, T. Basso, S. Gold, S. Stetson, C. Gauthier, D. Foster, B. Crawforth, T. McQuire, K. Sakallah, R. Lomax, T. Mudge, “Overview of Complementary GaAs Technology for High-Speed VLSI Circuits,” *IEEE Transactions on VLSI Circuits*, March 1998, vol. 6, no. 1, pp 47-51.

[4] A. Cagney, PSIM: PowerPC Simulator, ftp://ftp.ci.com.au/pub/psim, 1994.

[5] S. Gold, B. Bernhardt, and R. Brown, “A Quantitative Approach to Nonlinear Process Design Rule Scaling,” *Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI*, March 1999, pp 99-112.

[6] M. Johnson, *Superscalar Microprocessor Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[7] M. Kelley, M. Postiff, T. Strong, R. Brown, T. Mudge, “Complementary GaAs (CGaAs) 32-bit Multiply Accumulate Unit,” in *Conf. Rec. 31st Asilomar Conference on Signals, Systems, & Computers*, Nov 2-5, 1997, pp 1507-1511.

[8] S. McFarling, “Combining Branch Predictors,” *WRL Technical Note TN-36*, Digital Equipment Corporation, June 1993.

[9] J. E. Smith, “A Study of Branch Prediction Strategies,” *Proceedings of the 8th Annual International Symposium on Computer Architecture*, May 1981, pp 135-148.

[10] T. Yeh and Y. N. Patt, “Two-Level Adaptive Training Branch Prediction,” *Proc. of The 24th ACM/IEEE International Symposium and Workshop on Microarchitecture*, November 1991, pp 51-61.