Synopsis V5.0
Heavy Ion Single Event Effects Testing of the Intel Pentium III (P3) Microprocessors[†]

Jim Howard[1], Ken LaBel[2], Marty Carts[3], Ron Stattel[3], Charlie Rogers[3] and Tim Irwin[4]
1. Jackson and Tull Chartered Engineers, Washington DC 20018
2. NASA GSFC, Greenbelt, MD 20771
3. Raytheon ITSS, Greenbelt, MD 20771
4. QSS, Inc., Greenbelt, MD 20771

Test Dates: October 12-15, 2001. Report Date: January 21, 2002.

Introduction

As part of the Remote Exploration and Experimentation Project, work was funded for the "Radiation Evaluation of the INTEL Pentium III and Merced Processors and Their Associated Bridge Chips." As a continuing step in the completion of this work, Intel Pentium III processors were tested at the heavy ion facility at the Texas A&M University Cyclotron Facility (TAMU). There were three main objectives for this second heavy ion test. The first was to do a thorough investigation of single-event-induced functional interrupts (SEFI) and the single event upsets (SEU) in comparing the Pentium III processors in the PGA370packaged form. The second objective was to investigate the event rate for the various components of the processors, under a variety of software and hardware configurations and do a better job of getting per bit error rates for the caches. Finally, the third objective was to continue to look at the state-of-the-art processors and investigate the impact on rates with Device Under Test (DUT) clock speed. The remainder of this report details the test process, methodology and results from this second heavy ion testing.

Devices Tested

Pentium III devices with rated clock speeds of 933 and 1000 MHz were tested. Pentium III devices were manufactured by Intel. All devices were characterized prior to exposure. A listing of all devices used in this testing is given in Table I below.

Test Facility

Facility: Texas A&M University Cyclotron Facility
Heavy Ions: 55 MeV/amu Argon
Flux: 3.54 x 10^2 to 1.42 x 10^5 ions/cm^2/s.

Test Methods

Temperature:

The test was conducted at room temperature. Intel P3 junction temperature was monitored using an on-die diode.

Test Hardware:

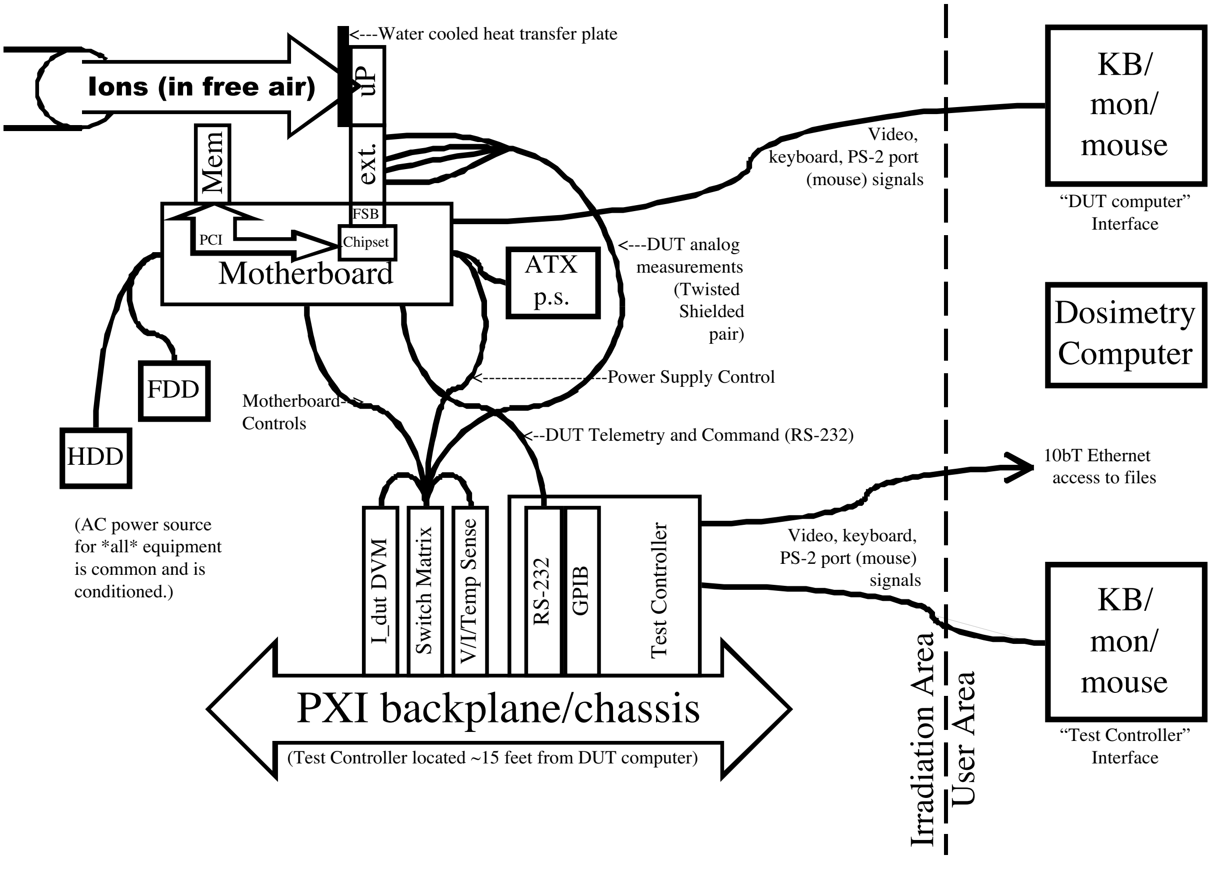
The system hardware consists of two subsystems, the Test Controller and the DUT computer subsystems (DUT cooling is discussed later as another subsection under this,

[†] This work was performed at NASA/GSFC for the Remote Exploration and Experimentation Project. This project is part of NASA's High Performance Computing and Communications Program, and is funded through the NASA Office of Space Sciences.

Test Hardware, section). Any cabling not required for the operation of the DUT computer is considered part of the Test Controller subsystem. These subsystems are described below. Following that is a description of the DUT processors. Figure 1 illustrates the overall test configuration. This has not changed dramatically from previous tests, but has evolved. The current (as of TAMU testing in October 2001) state of the test is described.

TABLE I
Device Under Test (DUT) Table

Device	Vendor	Speed	DUT Number	S-Code/ CPUID	Serial Number
Pentium III	Intel	933 MHz	933_FC2	SL4ME/686	0001F2EF0AB303EC
Pentium III	Intel	933 MHz	933_FC3	SL5DW/68A	000007DC594C62CE
Pentium III	Intel	1 GHz	1000_FC4	SL5DE/68A	000181200F7F92E9
Pentium III	Intel	1 GHz	1000_FC5	SL5DE/68A	0000B61C6F9A15FE



<---Water cooled heat transfer plate
Ions (in free air)
uP
ext.
Mem
FSB
PCI
Chipset
Motherboard
ATX p.s.
FDD
HDD
Video, keyboard, PS-2 port (mouse) signals
KB/ mon/ mouse
"DUT computer" Interface
<---DUT analog measurements (Twisted Shielded pair)
Dosimetry Computer
<------------------Power Supply Control
Motherboard--> Controls
<--DUT Telemetry and Command (RS-232)
10bT Ethernet access to files
(AC power source for *all* equipment is common and is conditioned.)
I_dut DVM
Switch Matrix
V/I/Temp Sense
RS-232
GPIB
Test Controller
Video, keyboard, PS-2 port (mouse) signals
KB/ mon/ mouse
"Test Controller" Interface
PXI backplane/chassis
(Test Controller located ~15 feet from DUT computer)
Irradiation Area
User Area

Figure 1. Block diagram illustrating the overall test configuration.

Test Controller (PXI) Subsystem

The Test Controller hardware is based on the PXI specification. A description of the Test Controller, or PXI Computer, subsystem follows.

The PXI subsystem, as shown in Figure 2, resides outside the irradiation area and is connected to the DUT at the irradiation point by cabling ~40′ long. It consists of the PXI components, the PXI Computer <-> DUT Computer cabling and the user interface.

The PXI components include the PXI chassis, which contains an embedded controller (running Win98, Labview (LV) environment and a custom LV application), a signal switch matrix, and two digital multimeters (DMMs) in the voltage measurement mode. The switch matrix provides two functions--The multiplexing of analog signals to one of the DMMs, and contact closures (pulling signal levels to ground). One DMM measures all analog values except the value read most frequently or as most important to not be delayed by switch settling time. The other DMM is dedicated to monitoring that value.

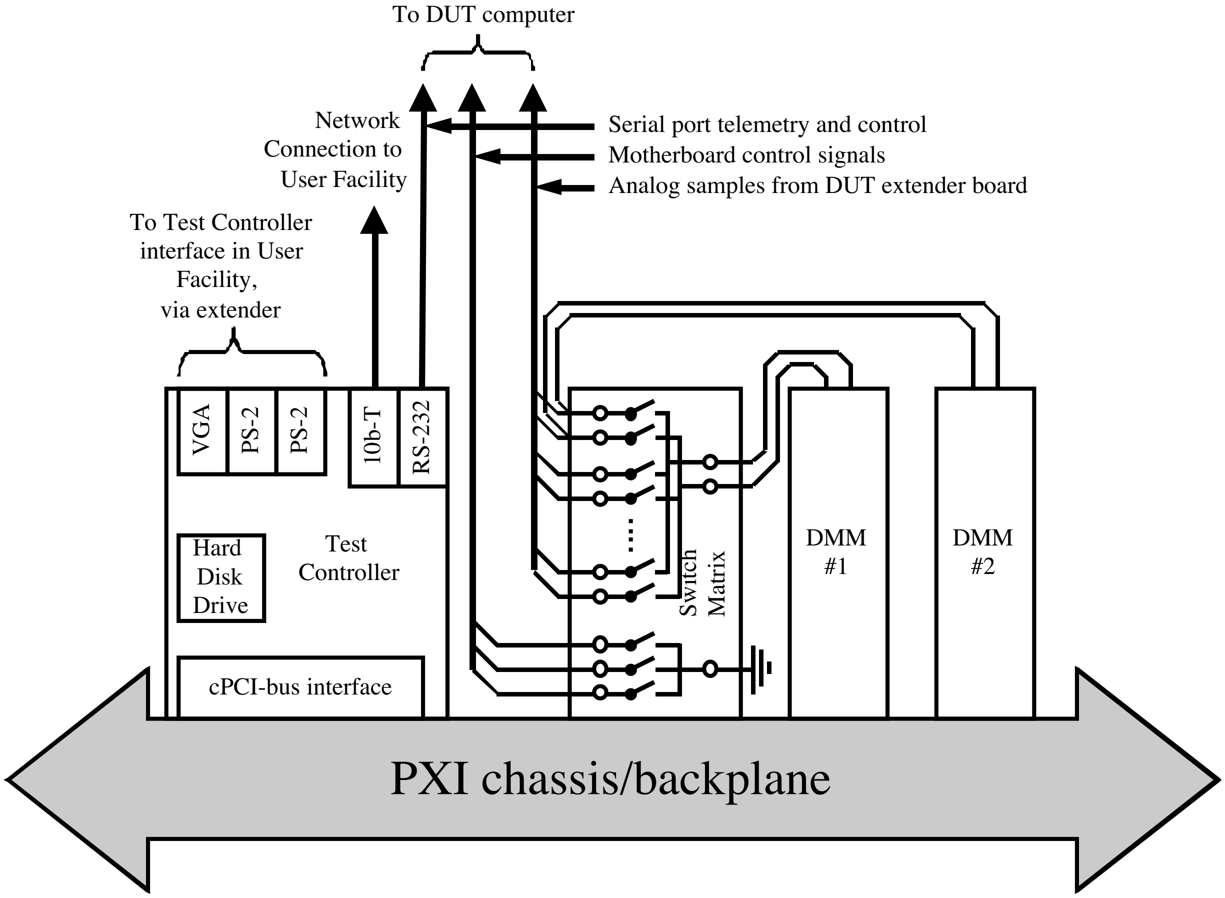

To DUT computer
Network Connection to User Facility
Serial port telemetry and control
Motherboard control signals
Analog samples from DUT extender board
To Test Controller interface in User Facility, via extender
VGA
PS-2
PS-2
10b-T
RS-232
Hard Disk Drive
Test Controller
cPCI-bus interface
Switch Matrix
DMM #1
DMM #2
PXI chassis/backplane

Figure 2. Block diagram of the PXI subsystem.

The PXI Computer's user interface, network connectivity (for data file access) and AC power feed are also components of the PXI Computer. An extended (via a CAT5 cable based extender from Cybex, Inc.) keyboard/monitor/mouse user interface provides user control of the PXI computer from the user facility (which in this instance is located in the hallway outside the regular, restricted access, user area).

Most of the PXI Computer <-> DUT Computer cabling leaves the PXI subsystem from the switch matrix (described further below). Exceptions are the AC power cable to power the DUT computer and a serial (RS-232) cable for telemetry/command of the

DUT computer (telemetry originates within the DUT Computer, commands originate within the PXI Controller).

The Texas A&M University Cyclotron Facility has a chain link fence separating the irradiation point and the restricted access user area. A scissors gate separates the restricted access user area from the hallway where control of this test was conducted.

DUT Computer Subsystem

The DUT computer subsystem consists of the components immediately connected with the operation of the DUT computer, including components mounted directly to the motherboard, components located nearby (e.g. disk drives) and connected via cables, and the user interface.

Directly attached to the commercial motherboard are the DUT processor (described later in the Pentium III section), a RAM module, and a PCI-bus memory board (added on the hypothesis that even the simple RS-232 telemetry was subject to SEE. This board and the serial port receive identical copies of the telemetry feed; the contents of the memory board will survive a soft reset of the DUT computer for later readout through the RS-232 port in case the RS-232 port or the entire processor crashes. This has proven useful.).

The DUT computer motherboard resides in the test chamber, positioned directly in front of the particle beam but so that only the DUT processor is irradiated. These DUTs are all flip-chips and their die are parallel to the surface of the board (unlike the original slot-version P3s tested); an aluminum base holds the motherboard and DUT die perpendicular to the particle beam.

Located nearby (~6 feet) are a modified standard PC ATX power supply (PS), a floppy and/or hard disk drive, and a Cybex user interface extension identical to the one used to extend the PXI computer.

Again, no extender is inserted between the DUT and the motherboard, and thus voltages but no currents were monitored. The motherboard is modified to monitor two voltages (Vcc and Vtemp). Each of these and a local ground sample for each are transmitted over twisted-shielded pair for measurement at the PXI DMMs.

Intra-DUT Computer cabling includes the disk drive cables, an extension cable for the ATX PS including a tap for test controller on/off control and the keyboard/monitor/ mouse extension (a CAT5 cable based extender from Cybex, Inc.).

The motherboard is modified to allow connection to two controlling signals, both momentary contact closures. The motherboard front panel power on/off (MotherPonoff) input signal is controlled by the PXI switch matrix, as is the motherboard front panel soft reset (MotherSR) input signal.

ATX PS on/off state is normally controlled by a constant signal from the motherboard (The ATX SP supplies a standby +5V to power such motherboard functions). This signal (PS_ON#) is, approximately, a latched toggle of the front panel signal, MotherPonoff. This motherboard PS_ON# signal is disconnected from the ATX power supply's PS_ON# input so that that can be controlled directly from the PXI. This additional control is necessary because the computer can hang to the extent of not responding to the normal on/off commands. The ATX PS AC power is extended back to the user facility.

The DUT Computer runs a real-time operating system and an autoboot software application residing on either floppy or hard disk, which periodically transmits a telemetry stream to the PXI through the motherboard's serial port via a null modem cable (see the section, Test Software, for details). Commands are also input from the PXI Computer over this cable.

The PCI memory board is a PCI plug-in card that makes memory available on the PCI-bus. During testing, the DUT software writes the same telemetry stream to this PCI memory as is written to the serial port. Unlike main memory, this PCI memory persists through soft resets and is retrievable by the main menu of the software application. A comparison of the serial port telemetry and the PCI board telemetry allows some insight to SEE failure locus and improves the general reliability of data gathering.

Intersystem Cabling

Three previously described digital control signals, MotherPonOff, MotherSR and PS_ON#, leave the DUT Computer subsystems via shielded cable. Two twisted shielded pairs carry the voltage samples from the DUT processor extender card to the PXI subsystem. The length of this cabling can be at least as long as 45′ without compromise of signal quality.

DUT Cooling

DUT cooling is described here instead under the DUT Description section because it is now less of an integral component of each DUT and more of another component of the DUT Computer. These DUTs dissipate upwards of 10W from one die and thus require cooling. The DUT die is face-bonded with the substrate up, so both the heat and the particle beam must traverse the entire thickness of the die. Whereas the cooling method for earlier proton testing on the Slot version of P3 allowed a thicker (~100 mil, 2540 micron) Al heat transfer plate to be used (with heat sink and fan), three factors made this impractical for these heavy ion tests. First, the limited heavy ion range makes a thinner heat transfer plate necessary, and the thinner the plate the greater the thermal resistance of the plate (and therefor the higher the DUT die temperature, all else remaining equal). Second, higher speed (and thus higher power dissipation) DUTs have become available since the earlier tests and the fan-based cooling solution has become impractical. Third, the flip-chip form factor of these DUTs would have made the lateral transfer of the heat via a heat transfer plate much more cumbersome.

The solution implemented here (See Figure 3) combines direct water-cooling of a metal plate instead of heat sink and fan, and involves superior attachment of the plate to the die.

The plate is machined out of Al stock and the finished product is approximately 0.5″ thick. Its size is slightly larger than the DUT. The metal immediately above the die is thinned to 10 mil thickness. One side of this rectangular hole is angled upwards in order to allow beam access from angles up to 60 degrees from normal incidence. When the motherboard is mounted vertically for irradiation, this slope is on the DUT's left side; angled irradiations are achieved by rotating the entire motherboard about a vertical axis penetrating the center of the DUT die.

Two parallel water channels and a perpendicular channel connecting the two (this third channel is then stopped up, resulting in a U-shaped water path) were drilled through the plate. 1/8″ NPT threads were fit with quick disconnect connectors. The channel is roughly shown as marks in the blue layout die in Figure 3; the brass and green plastic quick disconnects are to the right of the plate. The beam access hole is shown with the angle to the right.

Also shown in Figure 4 is the attachment system. #4 allthread is bent to a square-bottom U, with the threads ground smooth at the bottom to reduce abrasion of the white plastic socket's heat sink attachment hooks (The top hook is visible in the picture). A pair of holes in the plate allows the mounting of each of these U's. One pair of holes is round; the other is a rounded slot to allow for the rotation of its U during mounting/dismounting of the heat sink. The springs allow force-balancing ensuring that the cooling plate lies flat and evenly loaded, upon the DUT die. The four nuts are adjusted until the desired spring compression is achieved. The springs have a constant of about 25 lb/inch, so a compression of 0.2" on each of the four screws provided the rated compression force of 20 lb well centered on the die.

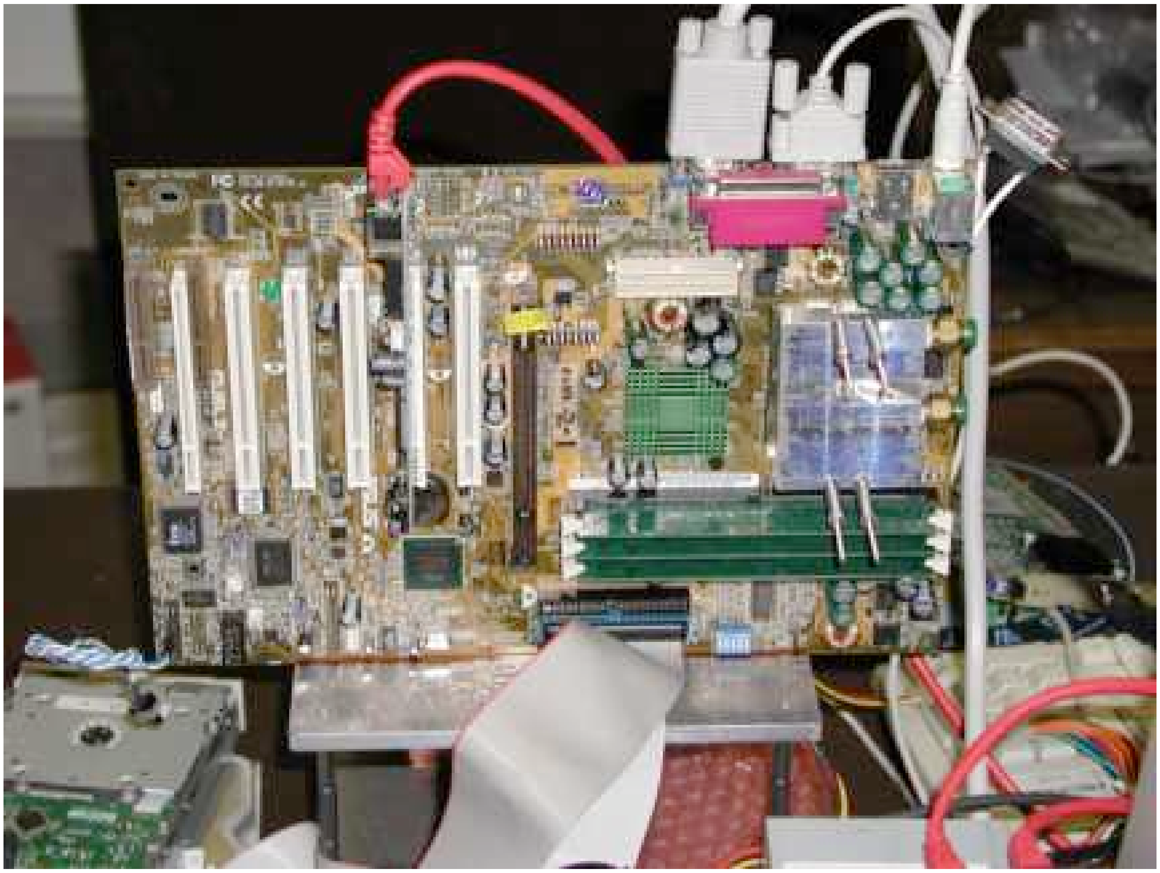


Figure 3. Cooling plate for the PGA370 package.

Figure 4, the die-side of the plate, shows two cutouts to avoid motherboard components and a shallow recess in which the top of the die makes contact with the plate.

Water flow from the source (described below) circulates through the U-shaped channel (the photo shows the connectors at one end without tubing for illustration purposes). The heat capacity of water and the transfer efficiency from the plate to the water are such that a low flow rate would result in only 10 °C rise, but a higher rate of approximately 1 gal/minute at ~30 psi drop across the entirety of the plumbing was selected.

Two water sources are available. The accelerator magnet cooling system uses Low Conductivity Water (LCW) for cooling to avoid eddy current losses in the magnets' high magnetic fields. The LCW system can have a very high source pressure and the return (back) pressure is about 30 psi. The LCW system water is a few degrees hotter than outside ambient air temperature, since it is cooled only through evaporative coolers. The amount of water flow available for experiment cooling is uncertain, but it is potentially limited.

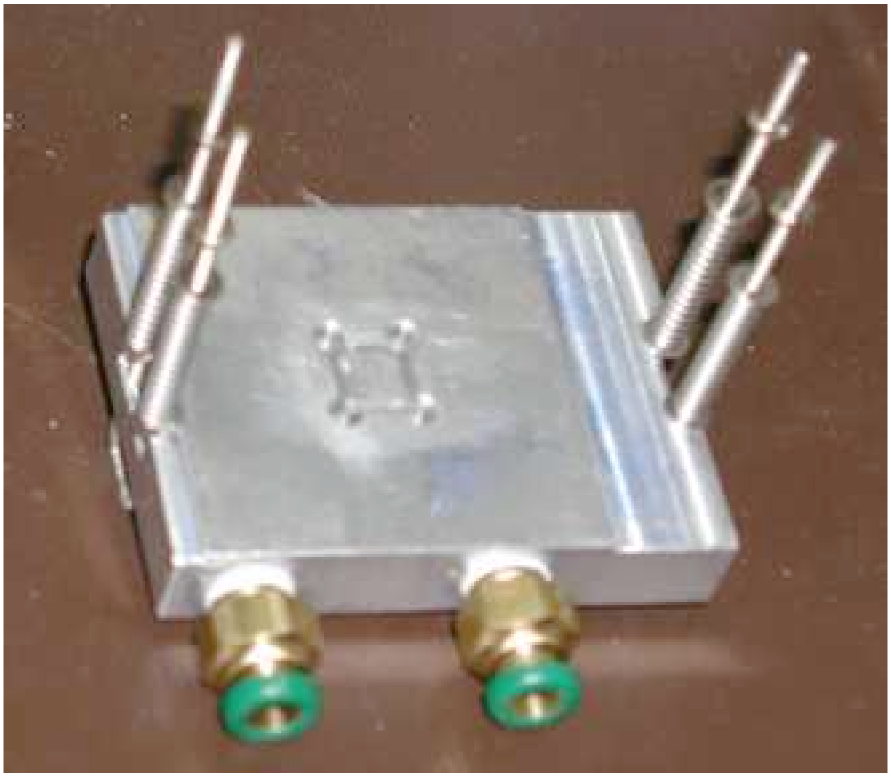


Figure 4. The heat transfer plate, die-side (with springs unfortunately mounted on the wrong side).

The alternative source of water is tap water, which is available at a relatively low (but quite useable) pressure, has zero back pressure (it exits the cooling plate down to a floor drain), is at ground temperature (which is somewhat cooler than ambient most of the year), and any conceivable flow rate is not an issue. In addition to these benefits, a simple setup of plumbing components can be used to provide this water source, in preparation and proofing of the water-cooling system; thus, tap water was chosen as the source. Figure 5 shows the plumbing used for setup. The input has a shutoff valve and a pressure regulator in series that feeds a cross. The three other ports of the cross hold a pressure gauge, an overpressure relief valve (set to ~10 psi over the regulator pressure), and the output to the DUT cooler. The return flow from the cooler passes back by this plumbing and travels with the overpressure valve output tube to the floor drain.

The tubing is common polypropylene tubing having a pressure rating of 150 psi but is fairly rigid, having a bend radius of as small as ~2″ only with coercion, and allows no twist, making it difficult to work with. The quick disconnects, visible in the figures, allow not only for quick setup and change, but also for free rotation of the tubing that greatly ameliorates the tubing rigidity problem.

At the accelerator facility, this plumbing is not needed, and the accelerator tap water source is available as a shutoff valve and two quick disconnects nearby the irradiation point, allowing easy switchover between water cooler systems.

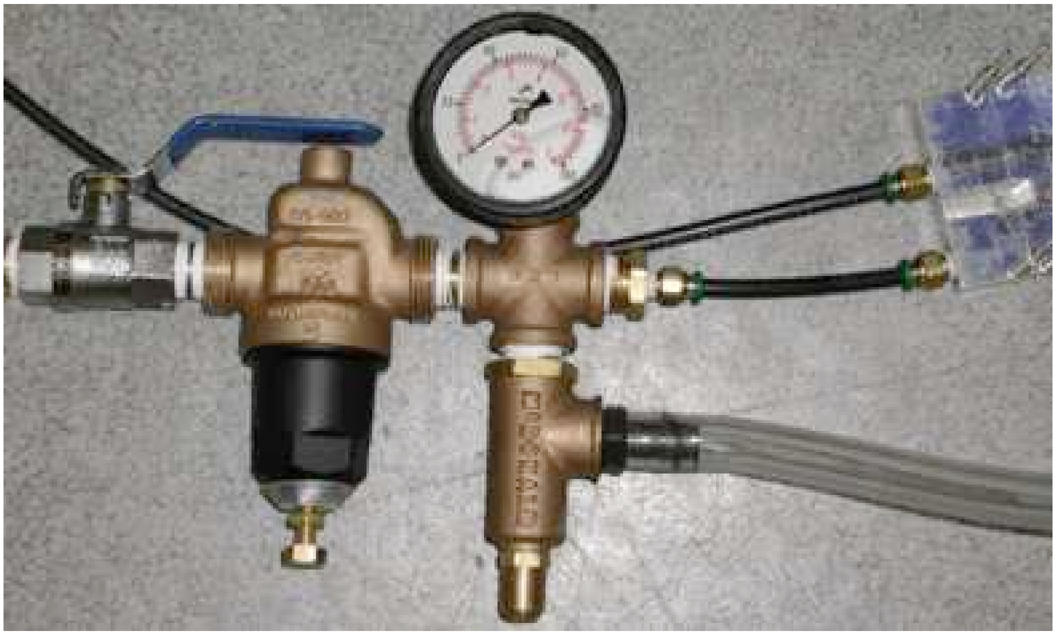


Figure 5. Water pressure regulator with overpressure relief valve.

For DUT operation without irradiation, as during setup, the OEM fan/heat sink/sprinclips solution can be used.

Pentium III (FC-PGA370)

The DUT processor is an unmodified standard PGA370 (FC-PGA) form factor Pentium III (P3) device manufactured by Intel Corporation. The SC242 (card-edge contacts module) form factor P3 would have been preferred because it was used in earlier tests and because an extender card exists for it (which allows samples of device currents), but the manufacturer migrated away from the more expensive SC242 form factor when on-die L2 cache became manufacturable.

The PGA370 P3 (see Figure 6) is a flip-chip device. The P3 die is ~0.45″ x 0.37″ x 35 mil thick, mounted active layers down, contacting ball bonds contacts on a small (~2″ square) board which also holds (on the reverse side) 370 pins and power supply decoupling capacitors. The board has something like green soldermask on its surface. A fillet of epoxy surrounds the edge of the die, probably to reduce stresses on and exclude contaminants from the ball bonds. No modifications are made to this device. The OEM cooling solution, a clip-on heat sink and fan, come (and are) separable. During setup, they are more convenient than the water-cooling solution and are used. 256 Kbytes of L2 cache reside on die and the system bus operates at 133 MHz.

An FC-PGA370 DUT mounted into its motherboard socket is shown in Figure 7. White thermal grease is visible around the edges of the DUT die, at the center of the green pin-spreading PCB. The ZIF socket lever is visible immediately to the right of the DUT. Tabs extending from the side of the white ZIF socket on the two sides adjacent to the ZIF lever are used by the cooling solution attachment, in this case, the sprung U-shaped clamps. Mounted to the underside of this PCB is a dense packed 370 pins. The center of the underside, immediately under the DUT die, does not have pins; a few supply bypass capacitors reside there. These are not expected to be affected by any irradiations.

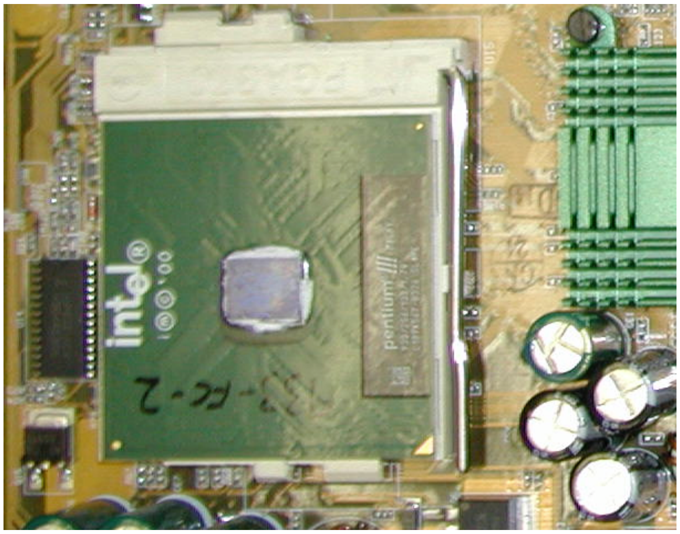


Figure 6. FC-PGA form factor P3, in a ZIF socket, without heat sink.

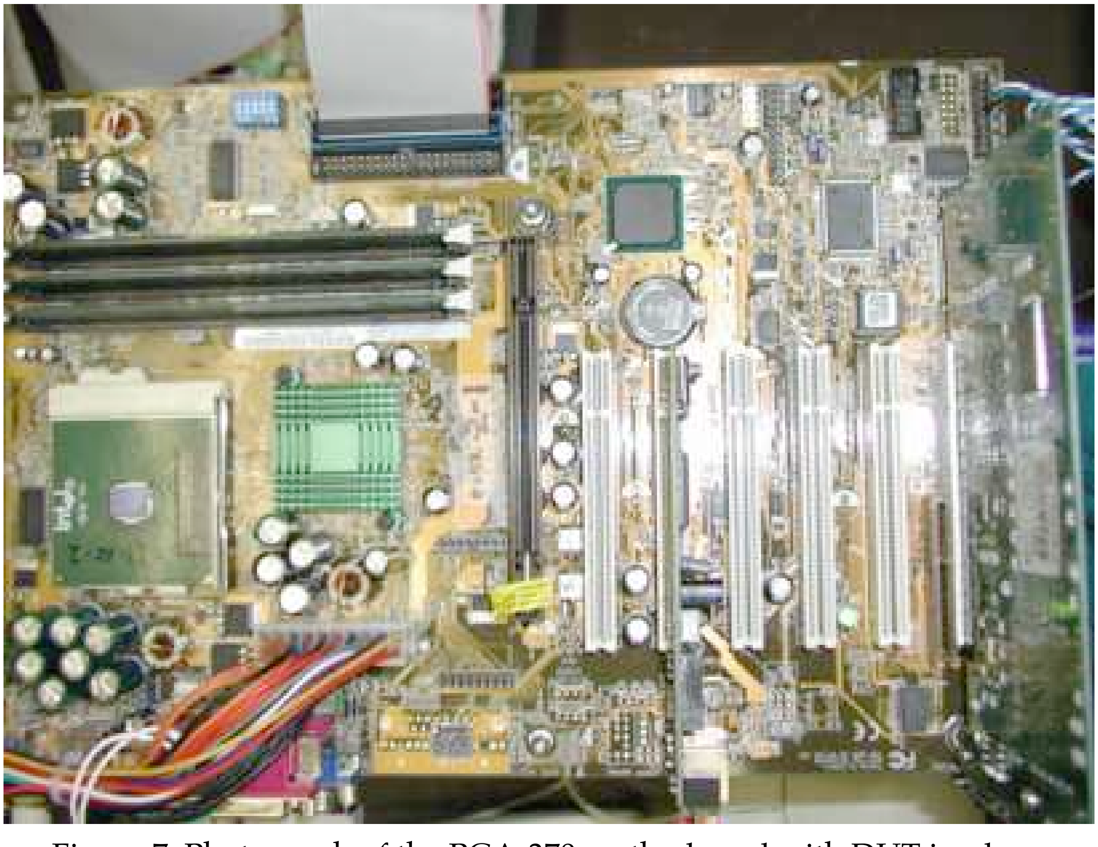


Figure 7. Photograph of the PGA-370 motherboard with DUT in place.

DUT computer signals that are controlled by the PXI subsystem, as described above, or by the user from the user facility are:

Name	Destination	Description

PS_ON#	ATX Power supply	Hold low (0 V) for PS on;
		Open = High = Off

MotherPonoff	Motherboard power switch connector	Pulse low (0 V) to toggle power on and off
MotherSR	Motherboard reset switch connector	Pulse low (0 V) to initiate reset
Command/ Telemetry	COM1	RS-232 carrying commands to the DUT subsystem (this also carries Telemetry to the PXI subsystem).
Keyboard	PS-2 keyboard port	

DUT computer signals that are monitored by the PXI or directly by the users in the user facility are:

Name	Source	Description
Vcc_core	Mother board	Voltage sample of the DUT core voltage. Twisted shielded pair (TSP): One side of pair is Vcc_Core; Other side and shield are ground.
V_temp	Extender board	Voltage sample of the on-die temperature sensing diode. TSP.
Telemetry/ Command	COM1	RS-232 carrying telemetry from the DUT subsystem (this also carries Command information from the PXI subsystem).
Telemetry and OS output	VGA card	Video carrying telemetry and OS/BIOS boot output to the user facility.

Test Software

The DUT software is compiled in Microsoft Visual C++ with a Pharlap Linker Add-in. The tests are written in a combination of C and assembly language. The software is executed using the Pharlap Real-Time Operating System. Pre-emptive task switching is turned off. The DUT communicates to the user through a VGA/keyboard interface and a serial port to the test controller system. All telemetry is echoed to a memory area that is not destroyed upon soft-reset or short power cycles. This memory is dumped through the serial port to the test controller after a reboot following a crash of the system. This is performed in an attempt to recover information lost due to the crash.

There are ten tests available for testing the Pentium III components. Each test sends a keep-alive at approximately a one hertz rate (0.1 Hz for test "H") and sends error results as they happen. One test is selected for each exposure to the beam and the software repeatedly performs the test until a pre-determined dose is reached or the software stops communicating to the user.

Test "A" checks the general-purpose registers of the CPU. There are eight general-purpose registers. Three registers are used by the program to keep track of execution parameters. The ESI register is used to point to a data area in memory where the values of the five registers under test are mirrored. The EAX register is used to keep track of the number of executions so that the keep-alive can be sent at proper intervals. The ESP register keeps track of the stack, which is used to store information when logging errors.

This test sets five CPU registers (EBX, ECX, EDX, EBP, and EDI) to a baseline value of 0AAAAAAAA Hex, then continuously checks to see if any of the register values change. If any values change, an error is reported to the user and an attempt is made to reset the register to its baseline value. The register is read again to form a new baseline value. The error report includes the following: the name of the register that changed, the value it changed to, the baseline before the error and the baseline after the error. The test then continues. At each keep-alive the baselines are reset to 0AAAAAAAA Hex.

There are four errors that this test is designed to catch: "Bit error: 1 changed to 0", "Bit error: 0 changed to 1", "Miscompare: data different by >1 bit", and "Miscompare: data identical". At TAMU the first two error types were reported as "Miscompare: data different by 1 bit". To determine how to classify this error the telemetry is examined to compare the values read and expected. The following is an example of the telemetry output:

```
A.A.A.A.A.A.A.A.A.A.A.A.A.A.A.A.
FAIL Sat Mar 24 03:44:14 2001

00000004
AAAAAABA
AAAAAAAA
AAAAAAAA
END
A.A.A.A.A.A.A.A.A.A.A.A.A.A.A.A.
```

The first line indicates that the error occurred in the ECX register. The second line is the value read from the ECX register after the miscompare. The third line is the expected value for ECX and the fourth line is the value of ECX after trying to reload ECX with the expected value. In this case the value read from ECX has a bit that is 1 where a 0 was expected. This error was classified as "Miscompare (data different by 1 bit)". If the second line were "FFFFFFFF" then it would be classified as "Miscompare (data different by >1 bit)".

In test "A", an error in one of the registers is expected to result in a "Bit error: 1 changed to 0" or a "Bit error: 0 changed to 1". A hit that causes a register to be reset is expected to result in a "Miscompare: data different by >1 bit". An error in performing the compare operation is expected to result in a "Miscompare: data identical".

Test "B" checks the Floating Point Unit (FPU) with a maximum of data transfer to the FPU. A buffer is loaded with the arguments and expected results for the five operations tested (fadd, fsub, fmul, fdiv, and fsqrt). For each cycle through the test, the following sequence is followed for each operation: The first operand is transferred to the FPU from memory. The second operand is transferred to the FPU from memory. The operation is performed. The result is transferred to memory. The result is transferred back to the FPU and compared with the expected result. Any errors are saved to a log in memory and reported to the user at the next keep-alive. The EBX is used to keep track of the number of cycles run. When EBX passes a constant number of cycles a keep-alive is sent and any errors are reported.

An error consists of three quadwords: The first is the function identifier (Hex 0:fadd 18:fsub 38:fmul 48:fdiv 60:fsqrt), the second is the result of the operation and the third is the expected result.

There are four errors that this test is designed to catch: "Bit error: 1 changed to 0", "Bit error: 0 changed to 1", "Miscompare: data different by >1 bit", and "Miscompare: data identical". The classification is determined in the same manner as that of test "A". An error in the performance of an operation, the transfer of operands, or a bit change in the operand registers is expected to result in a "Miscompare: data different by >1 bit". An error in the transfer of the result or a bit change in the result register is expected to result in a "Bit error: 1 changed to 0" or a "Bit error: 0 changed to 1". An error performing the compare function is expected to result in a "Miscompare: data identical".

Test C performs a memory test or cache test. If the cache is turned off, it performs a memory test otherwise, it performs a cache test. If the cache is on, then the cache is turned off, the memory is loaded with an incrementing pattern and the cache is turned back on before entering the test. The entire range is loaded with a baseline of 0AAAAAAAAH. The range is 131072 words for the memory test and the L1&L2 cache test and 8192 words for the L1 only test. The memory is checked word by word. After each word is checked, its value is changed to the bitwise complement of the baseline. If the value is not as expected then an error is reported and an attempt is made to reset the value to the baseline. The error report includes: the location address, the value read, the baseline and the value after attempting to reset to the baseline. If the value cannot be restored to the baseline, then checking is disabled for that location until the next keepalive. After the entire range is checked, the baseline is changed to its bitwise complement and checking starts again from the beginning.

Two types of errors occur during the cache tests. Tag errors cause 16 consecutive words to be in error and bit errors cause one or two bits to be in error in a word. We have seen three tag error scenarios. The first is that a tag miss occurs and the pattern from memory is read. The second is that the wrong baseline is read. In addition, the third is that a random pattern is read. The output from the first two scenarios has been compressed so that there is one line output that describes what happened for all 16 words. Most tag errors occur once but we have seen some tag errors recur several consecutive times within a run when testing with L1&L2 cache on.

When the cache test is started, the user is prompted for a test size. The available sizes are 100%, 50%, 25%, and 1%. The actual size of the test is computed as full cache (256K for L2 and 16K for L1) times this percentage. When testing the L2 cache, 1% is not an option since this would reduce the size such that only the L1 cache is utilized.

Because of the high number of bits checked in the cache test, it has the high number of reported errors. The number of bits tested in the L1&L2 cache test is 2097152*x% data bits plus 180224*x% tag bits (2277376*x% bits). This test is designed to test the L2 cache by always retiring data in the L1 cache when it overflows. The data flows through the L1 cache. The data is transferred to the L1 cache across a 64-bit data bus. The data then stays in the L1 cache for an average of .008% of the time it spent in the L2 cache. The L1&L2 cache test tests the entire bit range in 0.464*x%-0.928*x% seconds on a 1GHz part and divided by x/1GHz for an x GHz part.

The number of bits tested in the L1 cache test is 131072*x% data bits plus 12800*x% tag bits (143872*x% bits). This test tests the entire bit range in 0.0288*x%-0.0577*x% seconds on a 1GHz part and divided by x/1GHz for an x GHz part.

In all cache tests, the data is transferred to the execution unit from the L1 cache across a 64-bit bus. In order to determine whether transfer errors are occurring as opposed to bit flips in storage, we shall add cache tests that test smaller areas of the cache. We will find error rates at three different sizes for each cache. From this data we will determine the part of the rate that is independent of the size (transfer errors) and the part that is proportional to the size (storage errors).

For previous test "C" data, there were two possible classifications for a memory/cache error: "memory error", "recurring memory error". To determine how to classify this error the telemetry was examined for repeating entries. If an error repeated more than 50 times between keepalives it was classified as "recurring memory error". All other errors were classified as "memory error".

For this testing improvements were made to test "C" to give more visibility into memory/cache errors and to lessen the number of errors from instruction cache hits. The software was placed in a non-cacheable segment of memory and the telemetry output was expanded to include the location and contents of memory found in error. Also test "C" was changed from a process of loading all memory, checking all memory, loading all memory… to a process of checking one location loading that location checking the next location… This allowed the data to sit in its location (cache or memory) for the entire time cycle between checks. In the previous version, the data was loaded at about the halfway point between checks.

In addition, some data output reduction was added. If several consecutive locations read the wrong baseline or the ram pattern (for cache tests), then the telemetry output was reduced to one error message followed by "repeated xx times". Most tag errors now show up as follows:

```
FAIL Thu May 24 22:05:57 2001
 memory test error 01750 1750 repeated 16 times END
```

For the current test "C" data, there are four possible classifications for a memory/cache error: "recurring memory error", "tag error", "Bit error: 1 changed to 0", "Bit error: 0 changed to 1". If an error repeated more than 50 times or in more than 50 consecutive locations, it was classified as "recurring memory error". If an error appeared in 16 consecutive locations (1 line in the cache), it was classified as "tag error". If an error showed that a bit was 0 but was expected to be 1 then it was classified as "Bit error: 1 changed to 0". If an error showed that a bit was 1 but was expected to be 0 then it was classified as "Bit error: 0 changed to 1". There were several cases where one line of telemetry showed more than one bit error or tag error as in the following example:

```
FAIL Wed May 23 04:51:55 2001
 memory test error 149EE 08AEA 0AAAA 0AAAA END
FAIL Wed May 23 23:37:36 2001
 memory test error 02010 2010 repeated 32 times END
```

The first report shows 2 bit errors and the second shows 2 tag errors.

Test "D" launches seven subthreads each with a counter that is reset to zero. Each thread increments its counter if the counter is less than 11 and then passes control to the next thread. The main thread then checks after 50 milliseconds to see if all of the counters have reached 11. If not an error is reported to the user. The test repeats continuously.

The "Task Switch Error" is the only error that this test is designed to catch. This error will occur if any of the subthreads stop executing.

Test "E" runs through 16K of instructions repeatedly. The instruction sequence is to increment the eax register from 0 to 3 checking in between each increment to see if the value is as expected, then to decrement the eax register 3 times and check to make sure it returns to zero. Any errors are reported to the user and the cache is invalidated. Thirty of forty-six bytes in the code are continually used during proper execution; 65% of the 16K are exercised. The ECX register is used to keep track of the number of times the instructions are repeated. When ECX passes a constant number of cycles, a keep-alive is sent. The cache is flushed and the test is restarted whenever an error is detected.

Because the test is extremely sensitive to the value in EAX, the test will report a cache error as soon as the proper sequence is interrupted. The following example shows how the program responds to such interruption:

This is the error free operation of the test "E" (eax starts at 0):

```
40              inc eax
3D01000000      cmp eax,1
0F85E03F0000    jnz erout       ;jump not taken
40              inc eax
3D02000000      cmp eax,2
0F85D43F0000    jnz erout
40              inc eax
3D02000000      cmp eax,3
...
```

The following two scenarios show how one bit change can affect the operation of test "E" (eax starts at 0):

```
41              inc ecx
3D01000000      cmp eax,1
0F85E03F0000    jnz erout       ;jump taken
...
003D0100000F    add [ecx][eax]+0F000000H,bh
85E0            test esp,eax
3F              aas             ;adjust al after subtract
0000            add [eax],al
40              inc eax
3D02000000      cmp eax,2
0F85D43F0000    jnz erout       ;jump taken
…
```

This test is designed to report only one type of error: "cache error". Any sequence that causes a jump to "erout" will report a "cache error".

Test "F" checks the Floating Point Unit (FPU) with a maximum of operations in the FPU. A buffer is loaded with the arguments and expected result. The operation tested is:

cos(cos(cos(sin(sin(sin(sqrt(sqrt(sqrt(sqrt(sqrt(sqrt(a*b)))))))))))) where a=0.123456789 and b=0.987654321. For each cycle through the test the following sequence is followed: The first operand is transferred to the FPU from memory. The second operand is transferred to the FPU from memory. The operations are performed. The result is transferred to memory. The result is transferred back to the FPU and compared with the expected result. Any errors are saved to a log in memory and reported to the user at the next keep-alive. The EBX is used to keep track of the number of cycles run. When EBX passes a constant number of cycles a keep-alive is sent and any errors are reported.

An error consists of three quadwords: The first is the function identifier (Hex 78), the second is the result of the operation, and the third is the expected result.

This test has the same expected errors and causes as test "B". However, since there are fewer transfers and more time spent with data sitting in the FPU, the frequency of errors should be different.

Test "G" checks the Matrix Math Extensions (MMX). A buffer is loaded with the arguments and expected results for the four operations tested (pxor, por, pmul, pmulh, padds, addps, divps, and mulps). For each cycle through the test the following sequence is followed for each operation: The first operand is transferred to the FPU from memory (FPU registers are used by the Pentium for performing MMX functions). The second operand is transferred to the FPU from memory. The operation is performed. The result is transferred to memory. The result is compared with the expected result. Any errors are saved to a log in memory and reported to the user at the next keep-alive. The EBX is used to keep track of the number of cycles run. When EBX passes a constant number of cycles a keep-alive is sent and any errors are reported.

An error consists of three quadwords or five quadwords depending on the function: The first is the function identifier (0:pxor 1:por 2:pmul 3:pmulh 4:padds, 5:addps, 6:divps, and 7:mulps). For functions 0 through 4, the second quadword is the result of the operation and the third is the expected result. For functions 5 through 7, the second and third quadwords are the result of the operation and the fourth and fifth quadwords are the expected result.

There are four errors that this test is designed to catch: "Bit error: 1 changed to 0", "Bit error: 0 changed to 1", "Miscompare: data different by >1 bit", and "Miscompare: data identical". The classification is determined in the same manner as that of test "A". An error in the performance of an operation, the transfer of operands, or a bit change in the operand registers is expected to result in a "Miscompare: data different by >1 bit". An error in the transfer of the result or a bit change in the result register is expected to result in a "Bit error: 1 changed to 0" or a "Bit error: 0 changed to 1". An error performing the compare function is expected to result in a "Miscompare: data identical".

Test "H" measures the passage of time in CPU cycles against the ISA bus clock of 8.33MHZ. A timer board has been set up to provide an interrupt every 16383 cycles of the ISA bus clock. 5000 samples of the number of CPU cycles between interrupts are recorded and sent to the user after the test is complete in about 9.8 seconds. Then another test is started.

This test outputs a distribution of the number of CPU cycles between interrupts. The format of this output is as follows (each entry is in hex followed by a comma):

- the number of cycles for the first value in the list
- the number of samples below the lowest value in the list
- the number of samples beyond the highest value in the list
- the increment in number of cycles between entries in the list

- the list of values (comma separated) where a series of zeros is output as 0:v (v being the hex number of times to repeat the 0 entry). There is no comma after the last value in the list.
- The keep-alive: "H"

Test "L is used to verify that the CPU is still operating properly after changes in the setup or after a reboot. It loops through tests "A" through "H" repeating each test through two keepalives.

Test "N" is used in total dose testing. It continuously loops through tests "A" through "H" stopping every ten minutes for the test controller system to take voltage and current measurements. For these measurements, the test outputs "MEASURE <time>" to the telemetry signaling the test controller to take its measurements. It then waits 30 seconds for these measurements to be taken. Then it outputs "END" and continues looping through the tests until the next measurement time.

In addition to the errors that the tests are designed to catch, several errors can occur in any of the tests. The most common of these is the "General Protection Fault" (GPF). This error occurs when the program is not executing as designed. There are several events that could cause this:

- The Global Descriptor Table could have been corrupted. We now run all tests with the GDT stored in un-cacheable memory to minimize the occurrence of a corrupt GDT.
- The instruction could have been corrupted. We now run all tests with instructions stored in un-cacheable memory to minimize the occurrence of an unintended instruction being executed.
- A register could have been corrupted.
- The Arithmetic Unit could have computed an illegal address.

Other errors that can occur in any test are: illegal instruction, divide-by-zero exception, debug exception, non-maskable interrupt, one byte interrupt (INT3), interrupt on overflow, bound interrupt, device not available exception, double fault, coprocessor segment overrun, invalid TSS, invalid segment exception, stack fault, page fault, coprocessor error, hang, bomb, self reset. The last three of these errors are classifications that are determined by the manner in which a test terminates. A hang is the classification for when the DUT stops sending keepalives and stops responding to the user. A bomb is the classification for when the DUT display is loaded with a random pattern. A self-reset is for when the DUT reboots from the disk.

Any of the exceptions can be caused by an unintended instruction sequence due to a corrupt instruction or the execution of instructions from the wrong location (corrupt instruction pointer, segment register or GDT). This is very unlikely however since the instruction sequence for the exception is one in 65536 possible instruction codes.

The "illegal instruction" exception can also be caused by execution of an illegal instruction but is less likely because most of the values are valid instructions and executing an unintended instruction is more likely to end up in a GPF.

The "divide-by-zero exception" can also be caused by a divide by zero but is very unlikely because the event would have to load a zero into the divisor immediately before a divide instruction or an unintended divide would have to be executed with a divisor value of zero. Test "H" is designed to perform two divides per keep-alive. Test

"A","B","F", and "G" are designed to perform only one divide per keepalive. Test "C","D","E" are designed not to do any divides.

The "Debug exception" has no additional causes.

The "non-maskable interrupt" can also be caused by the NMI input signal being set.

The "one byte interrupt (INT3)" can also be caused by an INT3 instruction but is less likely because there is one code in 256 that performs the instruction and it would have to be an unintended instruction.

The "interrupt on overflow" can also be caused by an INTO instruction but is less likely because there is one code in 256 that performs the instruction that triggers this and the overflow flag must be set when it is executed.

The "BOUND interrupt" can also be caused by a BOUND instruction but it is less likely because there is one code in 256 that performs a boundary check and it would have to be an unintended instruction. In addition, the actual boundary check would have to fail.

The "Device not available exception" can also be caused by a failure in detecting the math coprocessor. This would have to be a transient that occurs at precisely the same time that the CPU is trying to detect the coprocessor.

The "Double fault" can also be caused by an exception in the exception handler. Since we are using the same exception handler for the "double fault", the exception in the exception handler is likely to cause an infinite number of calls to the handler and either continuously report an incomplete failure message or hang. The following combinations can cause a "Double fault":

- Contributory Exception followed by Contributory Exception.
- Stack Fault followed by either a Contributory Exception or a Stack Fault.

The "Coprocessor segment overrun" has no additional causes.

A corrupt GDT or TSS can also cause the "Invalid TSS" or "Segment not present exception". Since these are stored in memory it is less likely to occur.

The "Stack Fault" can also be caused by a corrupt GDT, TSS, stack segment register or stack pointer. This is more likely than "Invalid TSS" and "Segment not present exception" because if the stack is accessed outside of valid memory this fault is generated instead of the GPF that is generated for other segments.

A corrupt GDT or TSS can also cause the "Page Fault". There are three causes of a page fault: Writing to a read only page, Accessing a page which is not in physical memory, or Accessing a page to which the current program does not have access. Additional information is present in the error code:

- Bit 0 indicates whether (0) the page was not present or (1) there was an access rights violation or use of a reserved bit.
- Bit 1 indicates whether (0) the program was attempting a read or (1) the program was attempting a write.
- Bit 2 indicates whether (0) the program was executing in supervisor mode or (1) the program was executing in user mode.
- Bit 3 indicates whether (0) the fault was not caused by a reserved bit violation or (1) the fault was caused by a reserved bit violation.

During this testing we recorded one "Page Fault". It was caused by a write to a page that was not present.

The "coprocessor error" has no additional causes because the coprocessor is disabled from generating this error.

Table IV, below, shows possible errors reported for each test and possible causes. By correlating the expected susceptibility of the components within each test to the reported errors, we can determine a statistical probability for each of the possible causes, shown on the right side of the table.

TABLE IV

Pentium Test Error Interpretation Table

Test	Error Reported	Possible Causes
A	Miscompare report showing that the data is actually different	Specified register bits flipped
		Alias/Scratch register bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped
	Miscompare report showing that the data is actually the same	Instruction cache bits flipped
		Arithmetic unit bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
B	Miscompare report showing that the data is actually different by only one bit	Data cache bits flipped
		Floating point register bits flipped
	Miscompare report showing that the data is different by several bits	Data cache bits flipped
		Floating point register bits flipped
		FPU bits flipped
		Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
	Miscompare report showing that the data is actually the same	Instruction cache bits flipped
		FPU bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
C	Memory Error	Data cache bits flipped
		Register bits flipped
		Alias/Scratch register bits flipped
		Instruction cache bits flipped
		Arithmetic unit bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
	Recurring Memory Error	Register bits flipped
		Alias/Scratch register bits flipped
		Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
	Tag Error	Data cache bits flipped
D	Task Switch Error	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped

		Data cache bits flipped
		Register bits flipped
		Alias/Scratch register bits flipped
E	Cache Error	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Register EAX bits flipped
		Alias/Scratch register bits flipped
F	Miscompare report showing that the data is actually different by only one bit	Data cache bits flipped
		Floating point register bits flipped
	Miscompare report showing that the data is different by several bits	Data cache bits flipped
		Floating point register bits flipped
		FPU bits flipped
		Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
	Miscompare report showing that the data is actually the same	Instruction cache bits flipped
		FPU bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
G	Miscompare report showing that the data is actually different by only one bit	Data cache bits flipped
		Floating point register bits flipped
	Miscompare report showing that the data is different by several bits	Data cache bits flipped
		Floating point register bits flipped
		FPU bits flipped
		Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
	Miscompare report showing that the data is actually the same	Instruction cache bits flipped
		FPU bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
All	General Protection Fault	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Register reset to FFFFFFFF
		Register bits flipped
		Alias/Scratch register bits flipped
	General protection fault where CS:EIP = 18:FFFFFFFF	Instruction pointer reset to FFFFFFFF
	Illegal instruction	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
	Divide-by-zero exception	Instruction cache bits flipped

		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Debug Exception	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Non-maskable interrupt	NMI input signal set.
	One-byte interrupt (INT3)	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Interrupt on overflow	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	BOUND interrupt	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Device not available exception	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Double fault	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped

		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Coprocessor segment overrun	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Invalid TSS	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Segment not present exception	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Stack fault	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Page Fault	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Coprocessor error	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped

		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
	Hang	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
		Functional Interrupt
	Bomb	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
		Functional Interrupt
	Kernel	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
		Functional Interrupt
	Self-Reset	Instruction cache bits flipped
		Instruction pool (ROB)/Centralized Scheduler bits flipped
		Instruction pointer bits flipped
		Arithmetic unit bits flipped
		Data cache bits flipped (Page Table corruption)
		Alias/Scratch register bits flipped
		Functional Interrupt
		Reset input signal set

Test Methodology

Heavy Ion Single Event Effects Test Issues

LET Determination

The SEE testing utilized a beam of 55.1 MeV/amu Argon ions (giving an energy of 2204 MeV). Unlike the proton testing, all materials in the beamline must be considered. Therefore, these materials and the depth into the silicon die must be taken into account to properly determine the LET in the ions in the sensitive regions of the device.

The material conditions used in the TAMU testing were 25.4 microns of aramica (output window of the cyclotron), 80 millimeters of air (testing was done in air for thermal considerations), 10 mils of Aluminum (water-cooled jacket) and 900 micron of silicon (P3 die thickness). The first two of these were stationary, whereas the latter two would rotate with the device (for getting effective LETs). Figure 8 shows a plot of the LET in silicon as a function of the depth in silicon (the aramica and air degradations have been accounted for as well as a normal incident ion through the Aluminum).

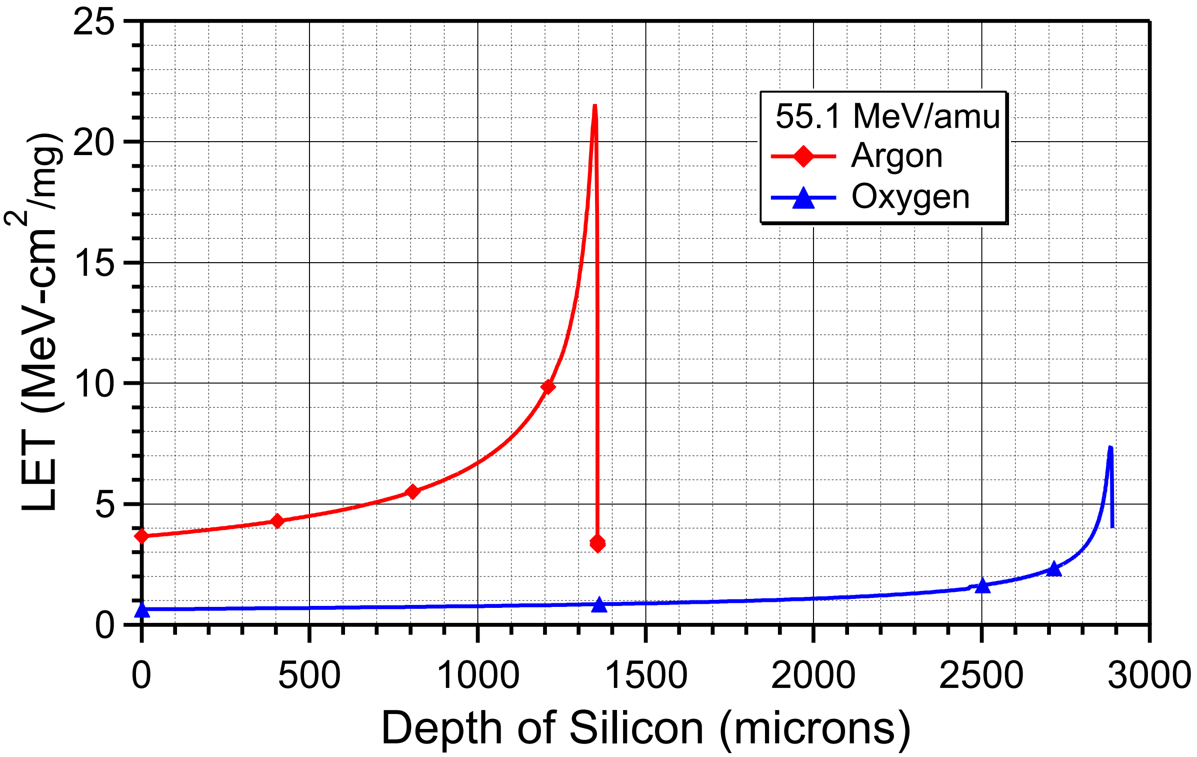


Figure 8. Plot showing the two ions used at TAMU for heavy ion testing. The zero point on the x-axis represents the Silicon surface on the backside of the die.

As can be seen from Figure 8, at the 800-micron depth into the die, the LET curve for Argon curve is heading towards the Bragg peak. Therefore, for the Argon ion, the LET will not be constant over the sensitive regions. That, combined with not knowing exactly where the sensitive regions begin, makes the determination of the LET in the sensitive regions problematic. For the remainder of this report, it has arbitrarily been chosen to take the 800-micron depth point as the defining depth for LET. It should be noted that this value should have associated error bars based on this depth and location on the Bragg curve. It would be best, though to utilize experimental information for a more accurate determination. This can be done using the roll off in cross sections curves with increasing LETs and through the use of charge collection spectroscopy (currently under investigation). Therefore, for the Argon beam, effective LETs of approximately 5, 8, 11, and 15 were used.

Thermal Considerations

The Pentium III, when operating under normal conditions with the caches enabled, will draw in excess of 20 watts of power. If left in that state with no cooling, the processor will not even boot. Also, the Pentium III die, as procured as a COTS parts, are flip chip solder bubble bonded die to the DUT daughter cards. Since the beam must hit the die directly, the packaged heat sink and cooling fan must be removed. In its place a water-cooled jacket, which is thinned to 10 mils over the die, is used.

The large thermal source is also the reason that the die cannot be thinned, as has been done with other flip chip parts. The thick substrate is the main thermal path for removal of heat from the junctions. Thinning this would place excessive thermal stresses on the die and most likely lead to structural failures.

Die Thickness Issue

As pointed out above, these parts are flip chip solder bubble bonded die. This places the sensitive regions of the processor approximately 900 microns deep in the silicon die with respect to the heavy ion incidence point (See Figure 9). Thermal issues compound this by requiring cooling material in the beam line, as well. Therefore, only high energy and high Z beams are capable of penetrating and giving higher Linear Energy Transfer (LET) values in the sensitive regions.

If the beam energy is such that normal incidence the ions do not have penetration issues into the sensitive region but when the beam is either degraded in energy or the part rotated (to give a larger effective LET), there are penetration issues, then roll-off of the cross section curves may be observed. This would be seen as a high effective LET data point having a lower cross section than the previous lower effective LET point. For the beam used in this testing and the P3 die tested, this roll-off point is expected to occur somewhere between an effective LET of 10-20 MeV-cm^2/mg.

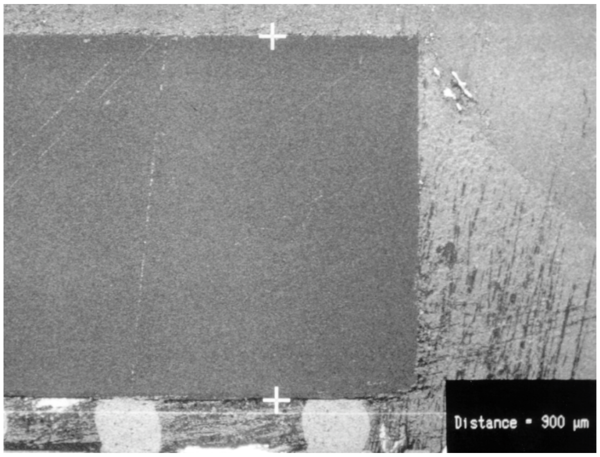
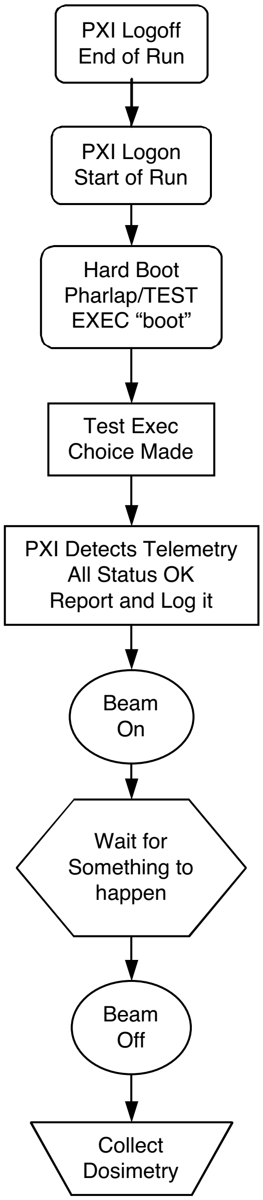


Figure 9. Photograph showing a close up cross section of the P3 die. Depicted here is the die thickness (900 microns) between the backside of the die and the front edge where the solder bubbles are clearly visible.

Single Event Effects Test Process

The SEE test process must include methods to test for all aspects of single event effects (latchup, functional interrupts, upsets, etc.). As a number of these effects are sensitive to the software being run and may be sensitive to numerous other conditions, detailed control of the DUT is required. To this end, an extensive operating system would serve no purpose other than a software overhead that is uncontrollable. Therefore, the boot process is done into a minimal operating system (Pharlap) and a test executive is run that allows testing of the DUT at a very low level. A flow diagram of the main testing process is shown in Figure 10.

The main part of this flow process, after the DUT is placed in a known operating state, is waiting for something to happen and then dealing with it. The flow describing this process is shown in Figure 11. Four general categories of events are expected: functional interrupts, latchup, system resets (radiation induced), and non-fatal errors (some error is produced but it does not immediately induce a functional interrupt or system reset. The remainder of the flow for all of these conditions shows the steps required to gain information about what exactly happened and to recover the DUT to a known state. The "Reboot Testing" in this routine is simply trying a soft reboot three times. If at that point the system has not recovered, a hard boot is then done. If a soft reboot is successful, then a "Memory Grab" routine is executed that recovers the telemetry stored in the Memory Card.



Previous run is completed

Logging is started and a new test run begins.

The test processor is hard-booted (so that a known startup condition exists), the minimal OS and test software is started.

Choice is made as to which test routine is to be run.

Test hardware detects that P3 is up and running test software correctly. This is logged and operator is informed that test is ready to go.

Proton Beam is turned on.

Software loop executes while we wait for something to happen. This is then handled in the "Something Happened Routine".

When the run is completed, the beam is turned off.

Beam dosimetry is collected and logged with the test results.

Figure 10. Flow diagram and description for main testing loop.

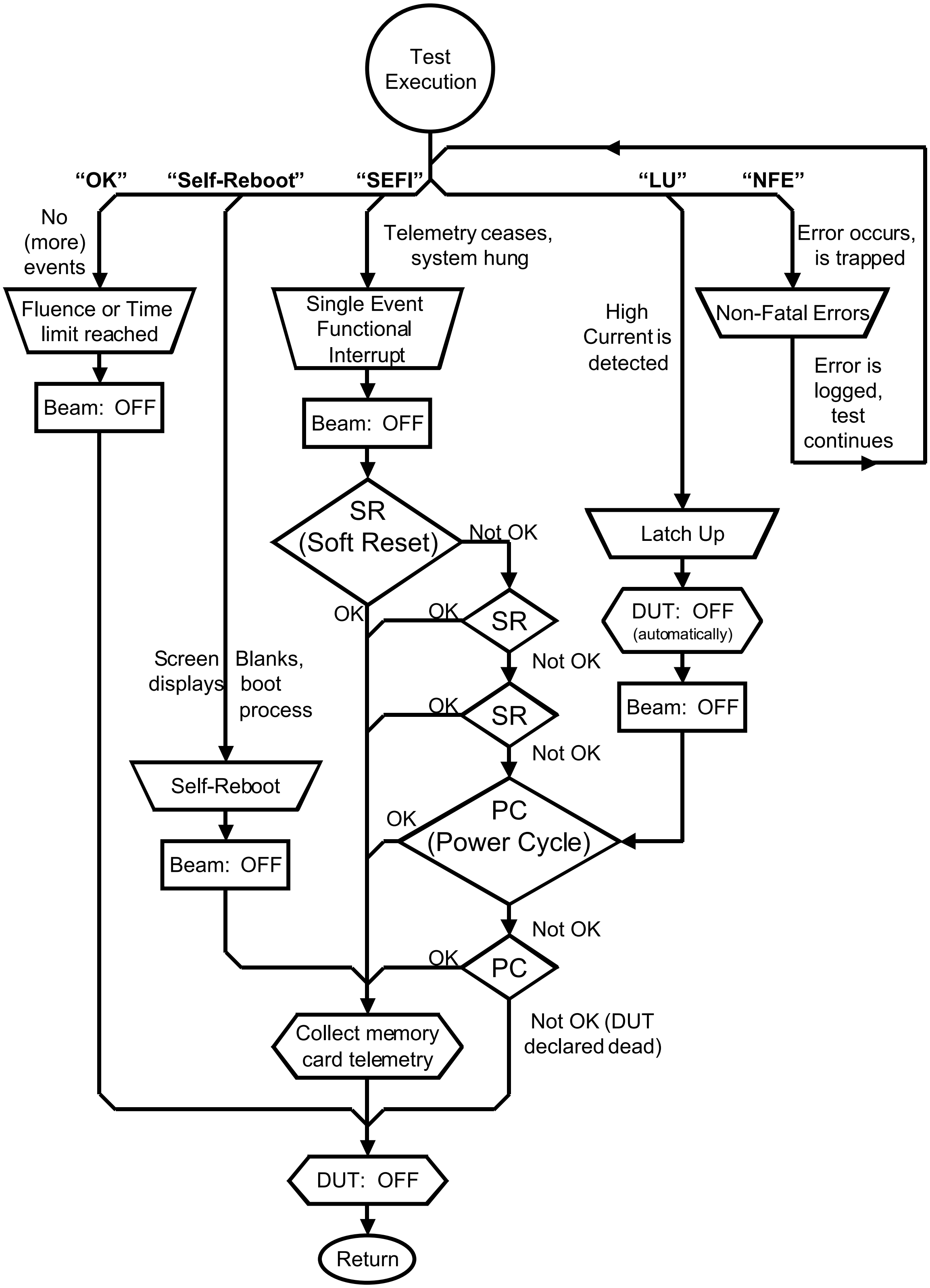


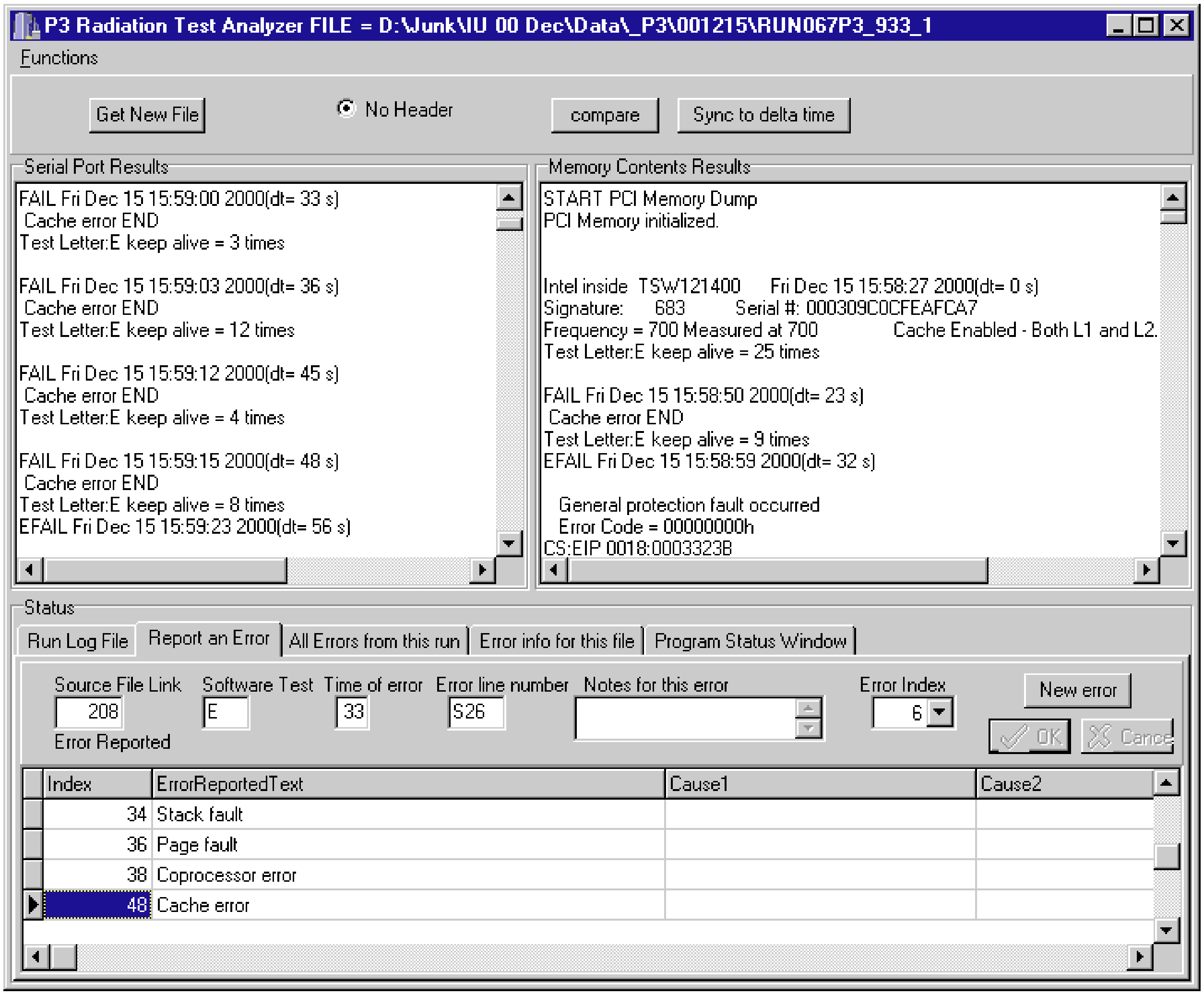
Figure 11. Flow process after an event has been detected.

Analysis Methodology

The data sets that are generated from Single Event Testing, due to the large test matrix, contain enormous amounts of data. It became necessary to generate software to deal with these large data sets to analyze the data under all the test conditions. A database form was chosen as the best form for this analysis. The database initially reads all of the test conditions, then allows to user to scan through the telemetry files. The user marks locations within the telemetry files with error annotations that are then stored in the database with those associated test conditions (See Table II, previously).

After each telemetry file has been annotated, the database can be queried via Structured Query Language (SQL) commands to extract only those conditions wishing to be analyzed. Depending on the detail of the SQL commands, either the event rate (or cross section) can be calculated directly or the selected data from the database can be exported in tabular format for other software to continue the analysis.

Figure 12 and Figure 13 show a sample screen shot for the telemetry file analysis and the SQL data extraction process, respectively.



Serial Port Results

```
FAIL Fri Dec 15 15:59:00 2000(dt= 33 s)
 Cache error END
Test Letter:E keep alive = 3 times

FAIL Fri Dec 15 15:59:03 2000(dt= 36 s)
 Cache error END
Test Letter:E keep alive = 12 times

FAIL Fri Dec 15 15:59:12 2000(dt= 45 s)
 Cache error END
Test Letter:E keep alive = 4 times

FAIL Fri Dec 15 15:59:15 2000(dt= 48 s)
 Cache error END
Test Letter:E keep alive = 8 times
EFAIL Fri Dec 15 15:59:23 2000(dt= 56 s)
```

Memory Contents Results

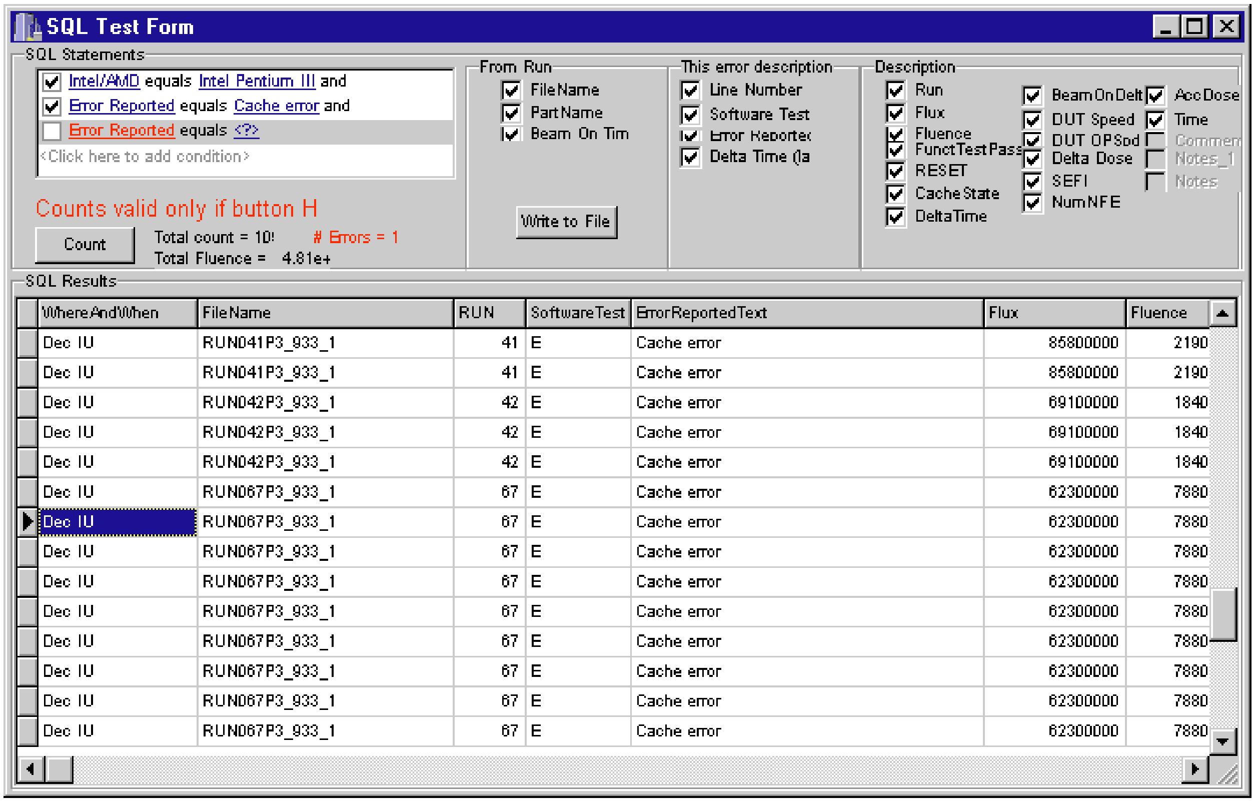
```
START PCI Memory Dump
PCI Memory initialized.


Intel inside  TSW121400     Fri Dec 15 15:58:27 2000(dt= 0 s)
Signature:     683         Serial #: 000309C0CFEAFCA7
Frequency = 700 Measured at 700            Cache Enabled - Both L1 and L2.
Test Letter:E keep alive = 25 times

FAIL Fri Dec 15 15:58:50 2000(dt= 23 s)
 Cache error END
Test Letter:E keep alive = 9 times
EFAIL Fri Dec 15 15:58:59 2000(dt= 32 s)

   General protection fault occurred
   Error Code = 00000000h
CS:EIP 0018:0003323B
```

Source File Link	Software Test	Time of error	Error line number	Notes for this error	Error Index
208	E	33	S26		6

Error Reported

Index	ErrorReportedText	Cause1	Cause2
34	Stack fault		
36	Page fault		
38	Coprocessor error		
48	Cache error		

Figure 12. Screen shot of the telemetry analysis software.



SQL Test Form

SQL Statements

☑ Intel/AMD equals Intel Pentium III and
☑ Error Reported equals Cache error and
□ Error Reported equals <?>
<Click here to add condition>

Counts valid only if button H

Count Total count = 10! # Errors = 1
Total Fluence = 4.81e+

From Run

☑ FileName
☑ PartName
☑ Beam On Tim

Write to File

This error description

☑ Line Number
☑ Software Test
☑ Error Reportec
☑ Delta Time (la

Description

☑ Run
☑ Flux
☑ Fluence
☑ FunctTestPass
☑ RESET
☑ CacheState
☑ DeltaTime
☑ BeamOnDelt
☑ DUT Speed
☑ DUT OPSpd
☑ Delta Dose
☑ SEFI
☑ NumNFE
☑ AccDose
☑ Time
□ Commen
□ Notes_1
□ Notes

SQL Results

WhereAndWhen	FileName	RUN	SoftwareTest	ErrorReportedText	Flux	Fluence
Dec IU	RUN041P3_933_1	41	E	Cache error	85800000	2190
Dec IU	RUN041P3_933_1	41	E	Cache error	85800000	2190
Dec IU	RUN042P3_933_1	42	E	Cache error	69100000	1840
Dec IU	RUN042P3_933_1	42	E	Cache error	69100000	1840
Dec IU	RUN042P3_933_1	42	E	Cache error	69100000	1840
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880
Dec IU	RUN067P3_933_1	67	E	Cache error	62300000	7880

Figure 13. Screen shot of the SQL data extraction software.

Results

Single Event Latchup

Four different P3 processors (two 933 MHz and two 1 GHz) run at the two different clock speeds (No attempt to collect data at clocked down processor speed was made in this testing as compared to previous tests. Previous testing indicated no difference in sensitivity, so the limited test time was used to further investigate the cache sensitivity.). During these tests, the processors were running one of the tests in the test executive (tests were varied) and exposed to heavy ion fluences (per run) that varied from 3.4 x 10^2 to 3.9 x 10^5 ion/cm^2. The P3 parts were tested in 407 conditions (effective Linear Energy Transfer (LET), cache on/off, and software executing). In all of the testing, no evidence of latchup was observed.

Single Event Functional Interrupts (SEFI)

Figure 14 shows the SEFI cross section for all the parts tested as a function of the effective LET. Shown are four curves representing the four cache conditions that were tested: All cache on, All cache off, L1 Data on only, and L1 and L2 Data cache on only. The first observation to make is that to first order the cross sections for all cache conditions are in the same order of magnitude. While the "All Cache Off" case is consistently below the other cases, it is not significantly below. This data seems to indicate that the primary mechanism that leads to a SEFI event is not generated with an error event in the cache. The cache may play a role in a second order effect, however.

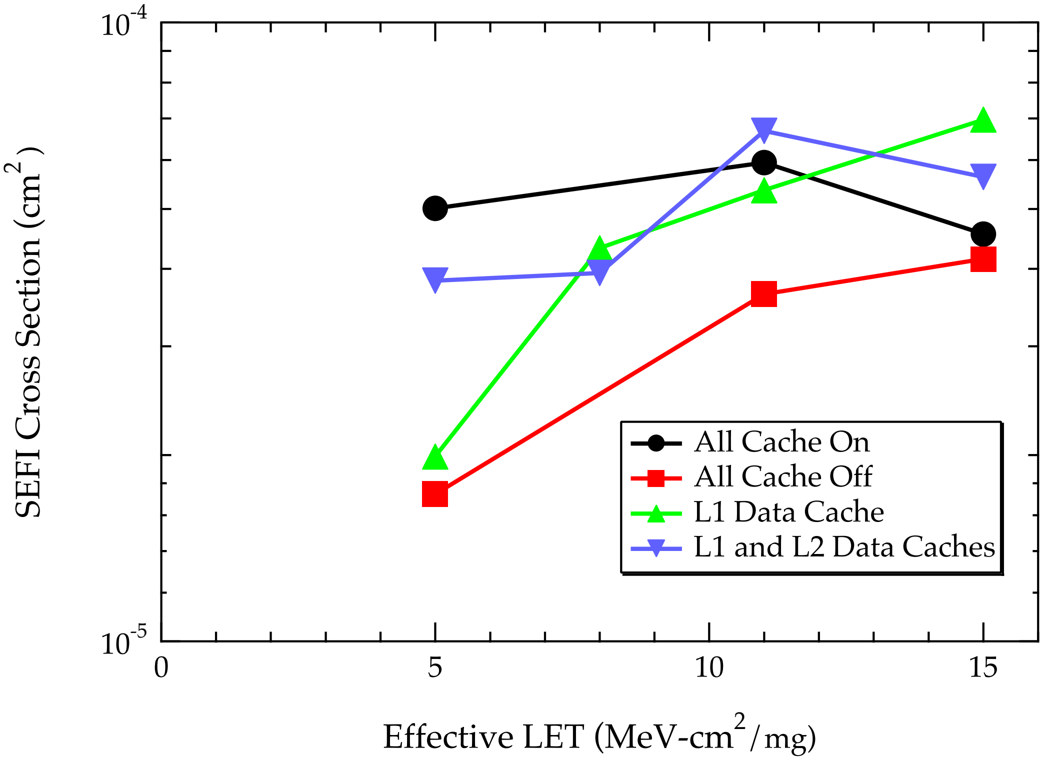


Figure 14. SEFI cross section as a function of effective LET for varying cache states.

Single Event Upsets – Non-SEFI

Figure 15 shows the cross section data for the first type of non-SEFI event, the exception. An exception is an event where the processor is capable of determining that something went wrong and possibly, what went wrong. If coded correctly, the processor can report this information to the software running and let it handle the event rather than stop the process. In the software used for this testing all exceptions are handled by stopping the process that is currently running, resetting all parameters for the test case being run, and restarting the process.

Unlike the SEFI cross section, it can be seen in Figure 15 that there is a strong dependence on the cache state for the exception rate. There is over an order of magnitude difference between the cases of "All Cache Off" and "All Cache On". There is a small difference when the data caches are enabled, but the primary effect is seen when the instruction cache is enabled.

One final item to note from Figure 15 is that all of the cross sections decrease between the LET 11 and 15 points. While variation can be expected in this type of testing, this is also the LET regime where roll-off is expected

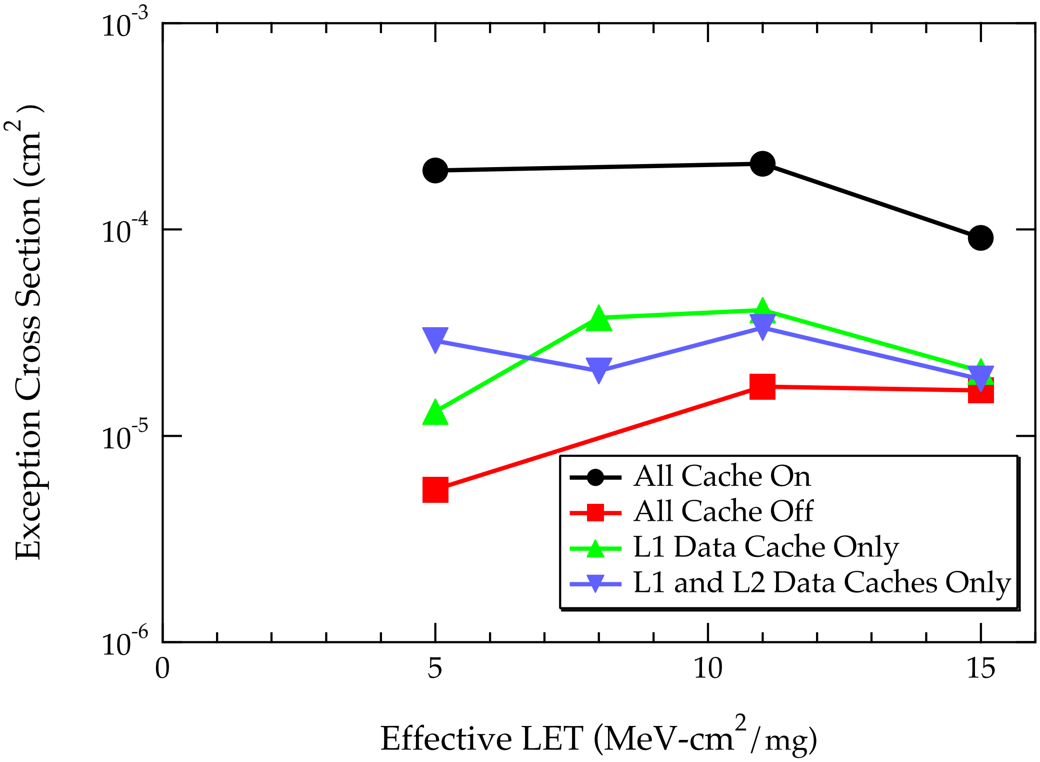


Figure 15. Exception cross section as a function of effective LET for varying cache states.

The next in line to consider for non-SEFI events are errors during the individual software tests (Tests A-G, see Software Section). Tests A, B, D, F, and G will be considered first as in all these cases very few to no errors were observed. Fluences during these tests were typically in the range of 10^5 to 10^6 ions/cm^2. Therefore, the registers, the floating point unit registers and combinatorial logic and the MMX unit have very small cross sections, on the order of a few time 10^{-6} cm^2.

Tests C and E test the data and instruction caches, respectively. These areas of the die do show a high sensitivity to upsets and will be dealt with separately. The first to be considered in the instruction cache test, Test E. Figure 16 shows the per-bit cross section for the instruction cache as a function of effective LET for the two device speeds tested. It is easy to observe in Figure 16 that the cross section is in the saturation regime over the LET range tested. The possible exception to this is for the 1000 MHz parts, roll-off may be coming into play for the effective LET 15 point. As in the SEFI and exception cases, it is difficult to say whether this is truly roll-off or just statistics. It should be noted, though, that the difference in the 933 and 1000 MHz parts shown in Figure 16 is not seen in the exception or SEFI data. That is why that data was not plotted as separate device speeds.

We can say with some confidence, though, that the threshold LET will be less than one and the saturation cross section is on the order of 2 x 10^{-8} cm^2/bit.

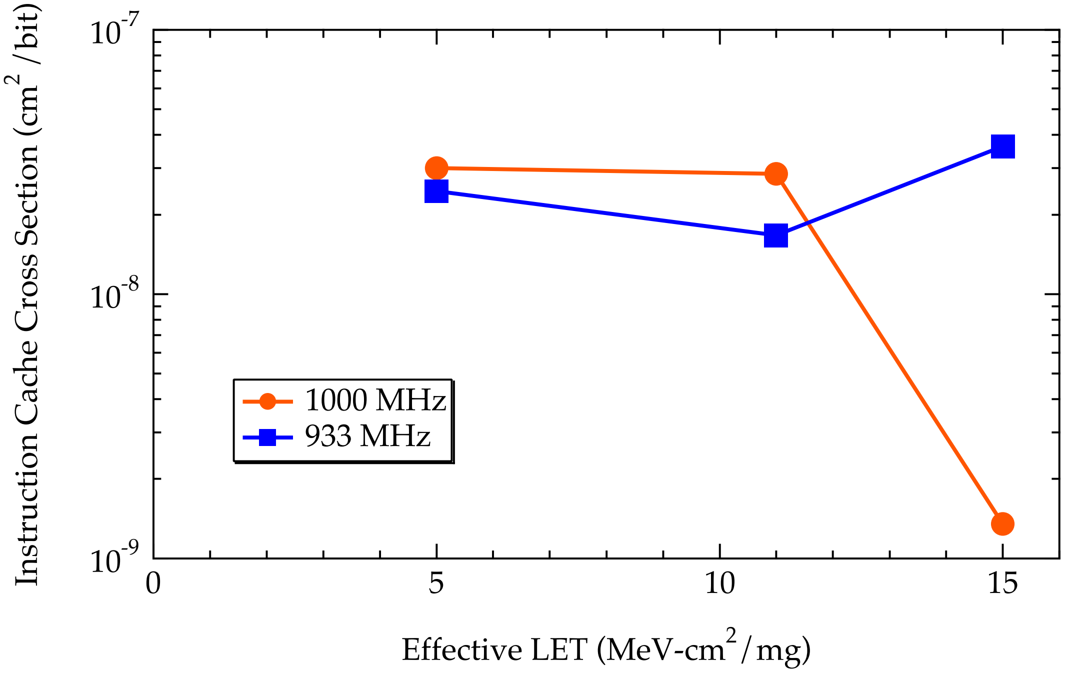


Figure 16. Data from Test E showing the instruction cache per–bit cross section as a function of the effective LET. Shown is the data for both the 933 and 1000 MHz parts.

The final test to be discussed, Test C, tests the data caches. As in the previous proton testing the data caches were controlled individually so that two cases were possible, testing with the L1 Data cache only and testing with both the L1 and L2 Data caches. Unlike previous testing, these cases were able to be further divided into percentages of the total cache size (either L1 only or L1 and L2). These percentages were 100, 50, 25 and 1% (except for the L1 and L2 case where the 1% run was not possible as this would make the L2 cache tested small than the L1 cache and the L2 cache would therefore not be sampled).

As stated in the software section, there are three types of errors that can occur in Test C. They are recurring memory error, tag error, and bit error (both 1 to 0 and 0 to 1 cases are tracked). With this improved software, recurring memory errors were not observed in this testing. We believe that previously observed recurring memory errors were an artifact of the software rather than a true stuck bit.

In addition, in this testing, the observation of different types of tag errors arose. The categories observed were single tag errors, multiple tags in error within the same test loop, and multiple occurrences of the same tag error across consecutive test loops. The mechanisms for the latter two of these tag errors is not yet understood or why they almost exclusively only occur for the case of the L1 and L2 data cache (only two events occurred for all the L1 only cases as compared to 139 events for the L1 and L2 cache cases, with similar fluence levels of 10^5 to 10^6 cm^{-2}). Therefore, for the L1 Data cache only, the cross section for these multiple tag errors is ≤ 10^{-10} cm^2. For the L1 and L2 Data cache case, the cross section for each of these multiple tag errors is approximately 2 – 4 x 10^{-10} cm^2.

The final tag error case to consider is the single tag error. Figure 17 shows the per-bit cross section for a tag error as a function of effective LET for the two cache cases tested. The error bars on this graph are the one–sigma statistical variation across the parts tested (both part-to-part and speed differences). The error bars on the L1 only case are significantly larger than the L1 and L2 case (they almost fall within the plot symbol). This is due mainly to the much smaller number of errors seen in the L1 only case.

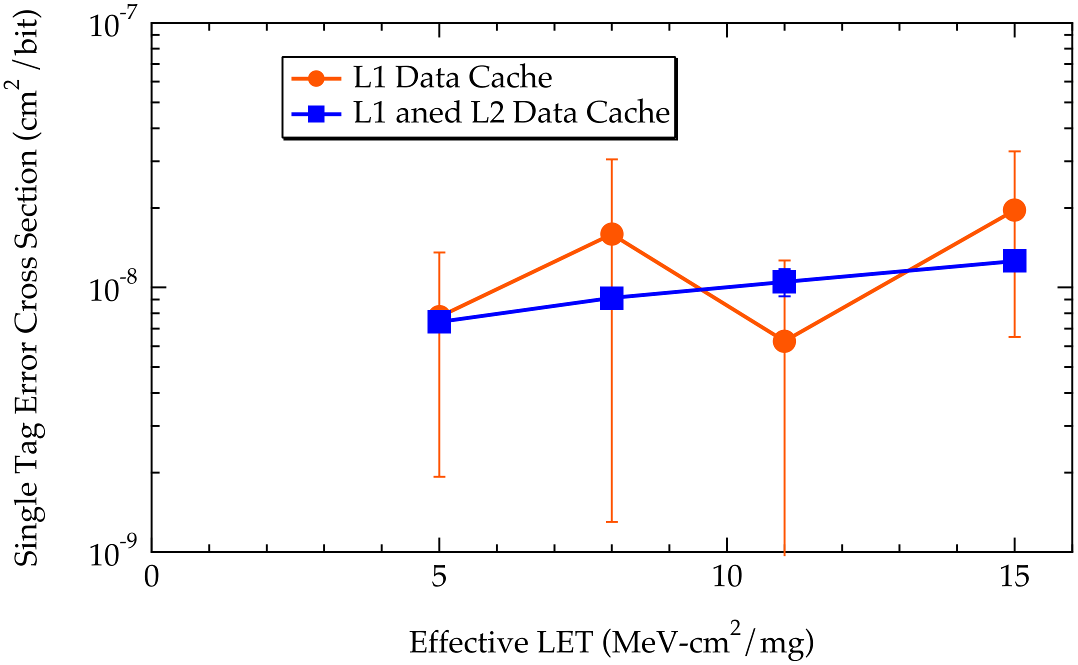


Figure 17. Data from Test C showing the per–bit tag cross section as a function of effective LET. Shown are the two cases of L1 data cache only and the L1 and L2 data cache.

It can also be seen in Figure 17 that the cross section is near, if not at, saturation levels across the entire range of LETs used. This would indicate that the threshold level will be small (< 1) and that the saturation cross section is in the range of 1 – 2 x 10^{-8} cm^2. These numbers are consistent with the data from the instruction cache test. Since the tag bits are physically the same technology and manufactured within the same array as the actual caches we should expect to see similar numbers between the actual cache bit error rate (both instruction and data) and the tag bit error rate.

The final result to be considered for Test C is the cache bit upset cross section. When the data is collected during the testing, bit upsets are classified as either a 0 to 1 or a 1 to 0 upset depending on their initial state. These results collected during this testing showed that there is no statistically significant difference between a 0 to 1 and a 1 to 0 upset (i.e., they occur at the same rate). It should also be noted here that the total number of upsets allowed during a single loop through the cache being tested was much less than 1% of the total cache size being tested, thus the possibility of a bit upsetting out of and back to its original state is extremely unlikely. With no difference in the upset sensitivity due to initial state, data presented from this point will be for the total bit upset cross section (i.e., the sum of the 0 to 1 and 1 to 0 cross sections).

The next issue to discuss is the process of the cache bit testing and its time implications. After the appropriate portion of the cache is loaded with the pattern (alternating 1's and 0's), each bit is read, its complement written back and read again. The purpose of this sequence is to first determine if the bit was upset since the last visit to the bit. The second reason is to determine if the bit is possibly stuck by writing a different value and making sure that the write actually happened. The disadvantage of this scheme is the amount of time it takes to do these three steps. While this may seem to occur very quickly, when it has to be done 2,097,152 times (for the 100% L1 and L2 case), the loop time can be significant. This is especially true considering the rate at which the processor can have an exception or SEFI event.

To determine how significant this timing issue is, a correction factor was determined based on a number of factors. For each run at the accelerator, besides collecting the number of bit upsets, exceptions and whether a SEFI occurred, the time that run took is also recorded. Using a typical time for a loop through the portion of the cache being tested (as determine in bench-top testing), a number of loops and errors per loop can be determined. The correction factor comes in whenever a loop is interrupted. This will occur every time either an exception or a SEFI occurs. The case for the SEFI is obvious but for the exceptions, it must be stated that the process control for an exception is to deal with the exception by resetting the processor to a known state at the beginning of a run and restarting the process. While this does not take a large amount of time, it does interrupt the loop that was currently running.

When a loop is interrupted, there is, in effect, an entire loop that has not been done. This is due to all the bits logically after the bit tested when the loop stopped and all the bits logically prior to the bit being tested that have been reset and may have already upset again. Therefore, a correction factor for each case is determined based on the average upset per loop for that case and on the total number of SEFI and exception events that occurred for that case. As would be expected, the cases were the number of bit being tested is small (all L1 only cases and the 25% L1 and L2 case), the correction factors were completely negligible (<< 0.1%). For the larger percentage cases for the L1 and L2 cache, the correction factor was still small (on the order of 1 to 3%). While not significant for this data set, this process can become important as the cache sizes continue to increase. Currently available P3 and P4 processors are being distributed with 512KB of L2 cache and Motorola PPC processors are being distributed with 2MB of L3 cache. Even though little difference is shown in the data, the correction factor has been kept for the remainder of this report.

The final issue to consider is the number of bits for each of the cases. We had thought that this would not be any issue and the correct number of bit would just be the fraction of the total for each of the different percentage cases run. While this seems to be a good assumption for the L1 only case, Figure 18 seems to indicate differently for the L1 and L2 cache cases.

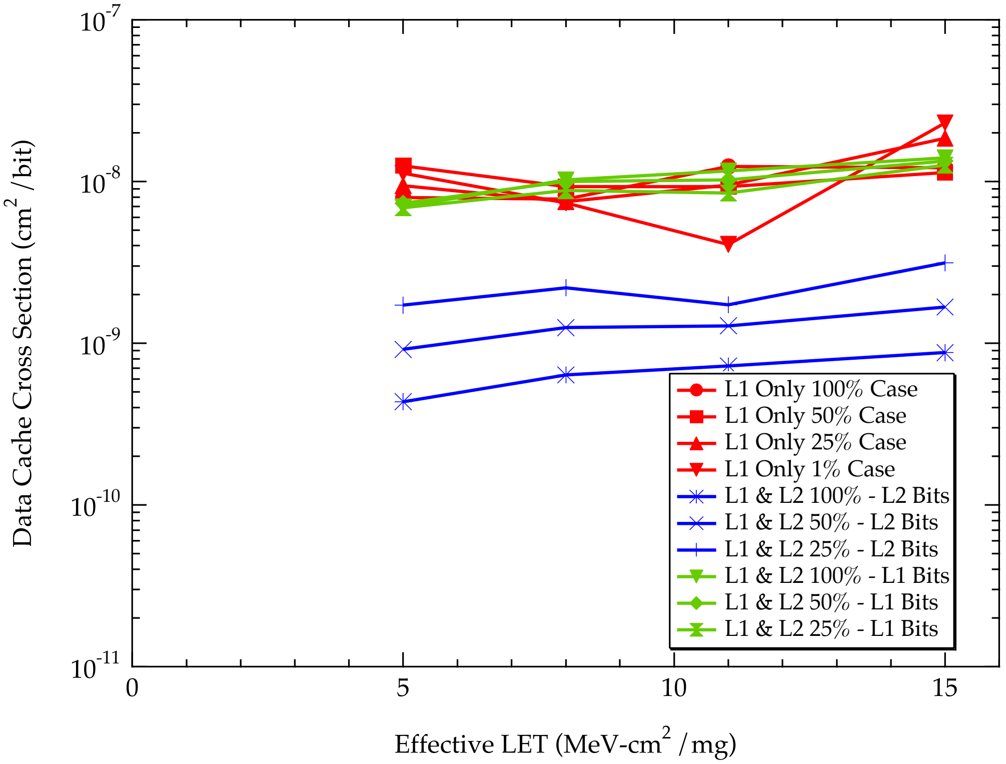


Figure 18. The per-bit cross section for the data caches is shown as a function of the effective LET. The data for L1 and L2 case is shown with two bit cases, the proportional L2 bits and the 100% L1 bits.

The four red traces represent the four percentage cases for the L1 only case. As can be seen in Figure 18, these show a saturation cross section trend across the effective LET region tested and a saturation cross section of approximately 1 – 2 x 10^{-8} cm^2/bit. This number is in line with the results from Test E on the instruction cache and the results from the tag errors from Test C. This is what one would expect if all the caches are of the same technology and layout (as is typically done).

The problem arises when the per bit cross section is calculated for the L1 and L2 cases using the proportional number of bits based on 100% of the L2 cache test. These three percentage cases are shown in blue in Figure 18. As can be seen, the curves fall below the L1 only case by an order of magnitude or more. More significantly is the spread in the curves as a function of the percentage of the L2 cache tested. This could possibly be explained by having two mechanisms that lead to upsets, one that is not related to the number of bits tested (e.g., in the periphery circuits) and the bit cell upsets. An attempt was made to determine what single curve could be subtracted from each of the three total cross section curves for the L1 and L2 percentage cases that would lead to the three per-bit percentage cases falling on top of each other. While there is no physical basis for this methodology, for the postulate given above, this curve must exist. The best attempt at finding this curve and the resultant per bit curves is shown in

Figure 19. In this figure, the red curves are the same as in Figure 18. The single green curve is the mystery mechanism that has not bit number dependence. It is shown here at the same level as the L1 data for convenience only. The actual cross section values are five orders of magnitude larger, as it is a per-device cross section rather than a per-bit. The three blue curves in Figure 19 are the remainders of the three L1 and L2 percentage cases after subtracting out the mystery mechanism curve and dividing by the proportional number of L2 bits for the percentage case. While the curves have grown somewhat closer together, the grouping is still not very tight and the values are almost two orders of magnitude lower than for the L1 cases (and for that matter the instruction cache and tag bits). This difference does not seem to make a good fit or reasonable explanation.

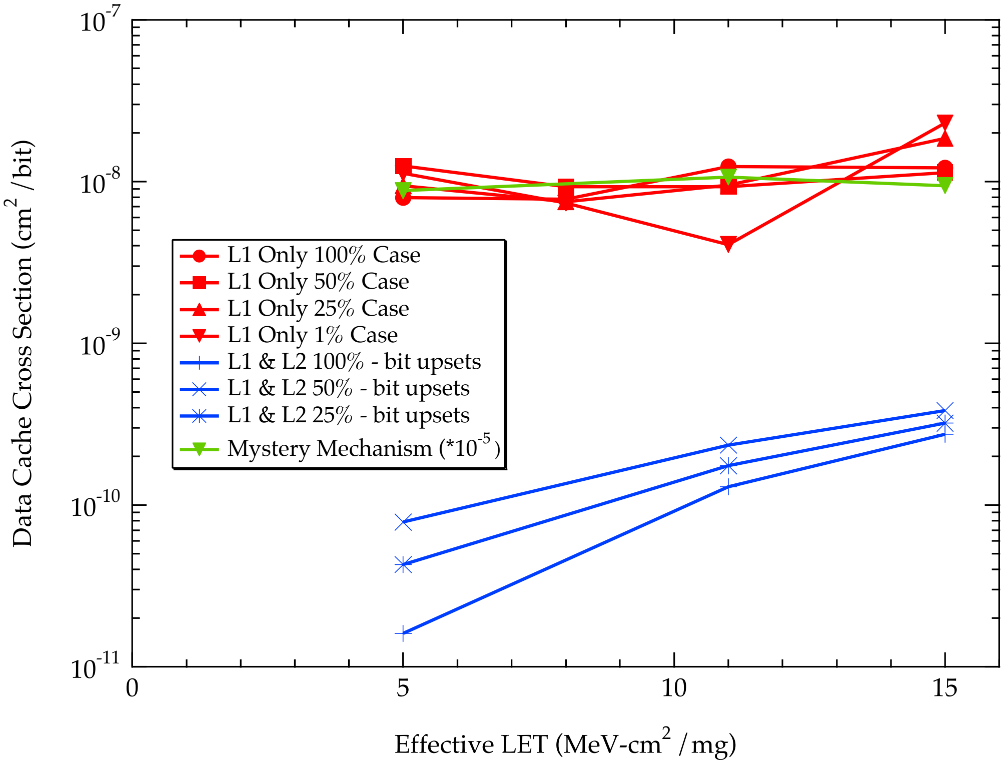


Figure 19. Data cache per-bit cross section curves showing possible two upset mechanism for the L1 and L2 cases.

To investigate an alternative explanation, we can return to Figure 18. In this figure the three green curves represent the three percentage cases for the L1 and L2 data cache that have been made per-bit by dividing the cross section by the total number of bits in the L1 cache (L1 Data 100% case). It is easy to see that these curves now fall directly on top of each other and on top of the L1 only curves, indicating the same per-bit cross section for all of the caches, under all percentage cases and their respective tag bits as well. This situation makes the most sense. The problem is trying to explain why the 100% L1 data cache size is the correct number of bits to be dividing by. At present, we have no explanation for this but it is still under investigation.

Assuming this is the correct situation, the per-bit cross section for the data cache is the same as the instruction cache and the data cache tag bits at approximately 1-2 x 10^{-8}

cm^2/bit. All of these curves are relatively flat over the entire effective LET range tested, indicating that the threshold should be small. A good estimate from this and earlier testing is that threshold will be at an effective Let of less than one.