

The Level-1 cache (which is typically on the processor chip) provides a small amount of very fast (single cycle) cache memory. Level-2 cache (which is typically external to the processor chip, and is accessed over the main memory bus) consists of fast static RAM. While the Level-2 cache is slower, its capacity is much larger.

This L2 cache is accessed when context switches are done, or when data accesses become widely dispersed. Each time this happens, the speed of execution slows for awhile, until the instructions (or data) being accessed from the L2 cache are copied into the L1 cache.

In systems where context switches occur frequently (or where data is widely scattered) the *speed* of the L2 cache can make a significant difference in the overall system performance. While the speed of the L2 cache can be increased somewhat with faster static RAM chips, the chip-to-chip signal propagation delays between the processor chip and the static RAM chips limit the maximum L2 cache speed.

One solution is to move the L2 cache onto the processor chip. However, a large L2 cache can take up considerable chip real estate. Fortunately, there is an alternative. Most critical signal timing is between the processor and the L2 cache controller logic. If these signal propagation delays can be reduced, access times to the L2 cache can be reduced.

The best way to do that is to move the L2 cache's SRAM memory controller logic onto the processor chip. In addition to reducing the signal propagation delays between the processor and the memory controller, this allows the controller circuitry to be synchronized with the processor's clock.

However, that only solves part of the problem. The address bus and the data bus signals must still be routed between the processor chip and the external SRAM chips. Given the limited drive capability of the processor chip and the memory chips, this chip-to-chip signal propagation delay depends largely on the capacitive loading of the signal lines between them. If the L2 cache is accessed over the same bus that is used to access main memory (a caching technique known as a *look-aside* caching) the capacitive loading (and the propagation delay) on that bus will be significant.

The signal loading between the processor chip and the L2 cache memory can be minimized if the L2 cache is accessed through a *different bus* than the main memory, as shown in Figure 2. This architecture provides many advantages:

- This point-to-point address/data bus can operate at a much higher clock rate.

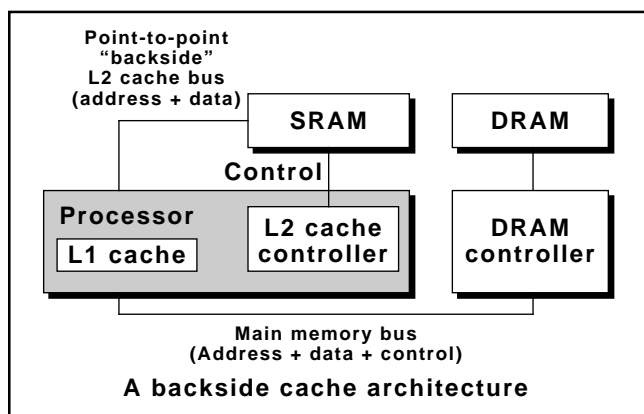


Figure 2

- The main memory bus can run faster, since it does not have the capacitive loads of the L2 cache.
- The traffic on the main memory bus is greatly reduced, because L2 cache hits don't produce any activity on the main memory bus. This allows concurrent DMA transactions, as long as the processor continues to execute from the L2 cache.

Choosing and maximizing the right DRAM technology

Of course the contents of both L1 and L2 caches are initially loaded from main memory, and whenever main memory is accessed the processor must slow down to main memory speed. If the L1 and L2 caches do their job, main memory is accessed relatively infrequently.

However, with the very high clock speeds of today's RISC processors, each main memory access uses up a lot of processor clock cycles. Taken together, main memory accesses account for a large portion of the processor's clock cycles, and any reduction in main memory access time will provide substantial performance benefits.

Fortunately, with the L1 and L2 caches providing instructions to the processor during loop execution, accesses to main memory tend to be mostly sequential, as instructions are loaded into the caches on the first iteration through each loop. DMA transfers are also done sequentially.

Using Extended Data Out (EDO) DRAM

These sequential accesses provide an opportunity for optimizing the design of the main memory. Extended Data Out (EDO) DRAMs have latches on their data outputs that maintain the data output signal levels, as the DRAM begins its next access. Because the DRAM can begin its next cycle earlier, this can reduce the memory cycle time by as much as one third. By combining this reduced memory cycle time with 2-way interleaving, data transfer rates for sequential accesses can be quite good.

For example, Motorola's Falcon memory controller chip set can be used to implement a 2-way interleaved memory architecture with a 427-Mbyte/sec data transfer rate, using standard Extended Data Out (EDO) DRAM. This level of performance is provided even with error detection and correction.

Using synchronous DRAM

Synchronous DRAM (SDRAM) is optimized for *burst* accesses. It requires multiple clock cycles to open one of its 4 banks of memory. However, once a bank has been opened, each subsequent access to that bank requires only one cycle.

This is actually an extension to the "page" concept. However, page mode DRAM has only one bank while SDRAM has 4 banks, increasing the chance that a "bank hit" will occur. In addition, an SDRAM controller will usually support multiple blocks of SDRAM memory, so the chance of hitting an opened bank is fairly high.

In order to take advantage of the reduced latency time for subsequent access to "open" banks, an SDRAM controller must keep track of which banks are open. This complicates the design, but when interfaced to a processor with a pipelined bus interface, SDRAM can provide impressive data transfer rates.

For example, when used with a 100 MHz processor clock, current SDRAMs can use 4-read bursts of 2.5-1-1-1 clock cycles to do sustained data transfers at 582 Mbytes/sec. Burst writes are a little faster at 640 Mbytes/sec. Figure 3 compares the PowerPC burst read bandwidth for various DRAM technologies.

Main memory performance can make a big difference

In many applications, the main memory performance can be the determining factor in the overall system performance. Figure 4 shows the specINT95 and specFP95 benchmarks for various PowerPC single board computers, including:

- the MVME2300
- the MPX100
- the MVME3604

The importance of the main memory speed is indicated by the fact that the MVME2300 (which has no L2 cache) performs very well against the MPX100 (which has 1 Mbyte of L2 cache). To be specific:

- the SPECint95 performance of the MVME2300 is almost as good as the MPX100
- the SPECfp95 performance is considerably *better* than the MPX100

The reason for this is that the MVME2300 has a much faster main memory.

In fact, higher performance can actually be obtained with a *smaller* L2 cache, as long as the main memory is fast. As shown in Figure 4, the MVME3604 board (with a cache only half as large as the MPX100) greatly outperforms the MPX100 because its main memory is faster.

Providing a peripheral bus

On today's single board computers the peripheral chips typically reside on a local PCI bus. For example, (as shown in Figure 5) a PCI local bus might be used to support:

- an Ethernet controller chip
- a SCSI controller chip
- serial ports
- PMC mezzanine sites

Many (if not most) of these I/O chips have built-in DMA controllers that can transfer data at full speed over the PCI bus. For example, a 33 MHz DMA controller which can transfer 32-bit words would be able to transfer data at:

$$(33 \text{ Mtransfers/sec}) * (4 \text{ bytes/transfer}) = 132 \text{ Mbytes/sec}$$

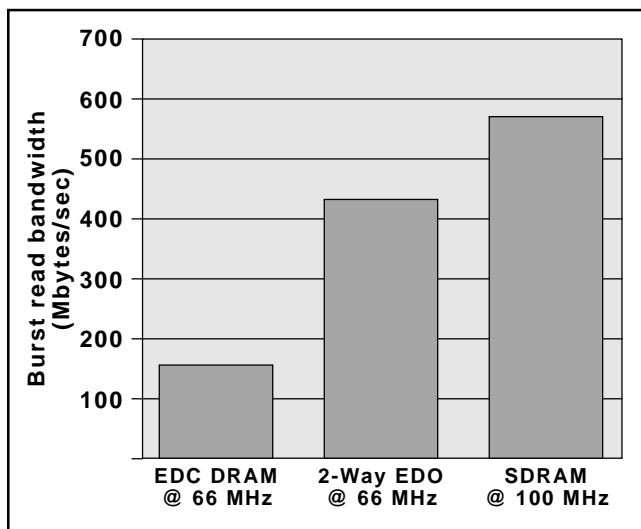


Figure 3

Of course, this data transfer rate can only be achieved if the PCI-to-memory bridge can accept (or provide) a 32-bit word on *each and every* PCI clock cycle. Unfortunately, off-the-shelf PCI bridge chip sets can only pass data at rates up to about 70 Mbytes/sec. Thus, the only way to transfer data into DRAM over the PCI bus at 132 Mbytes/sec is to design a custom bridge ASIC.

DRAM latencies

Writing data into (or reading data from) DRAM, there is a latency on the first access. Thus, the behavior of a DRAM is described as n-1-1-1-1... where "n" is the initial latency and -1-1-1-1... indicates that the DRAM can provide (or can accept) data on every clock beat thereafter.

If the DRAM resides on a bus where a processor also has access to it (as shown in Figure 5) there will also be some *bus arbitration* latency that must be taken into account in computing the average throughput of the DMA device.

Thus, even if a PCI DMA device is able to provide (or accept) data at the maximum 33 MHz rate, the initial latency will add

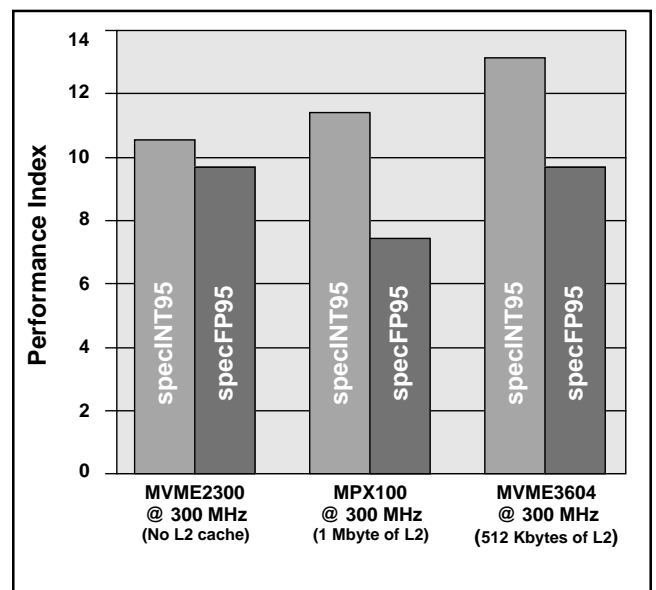


Figure 4

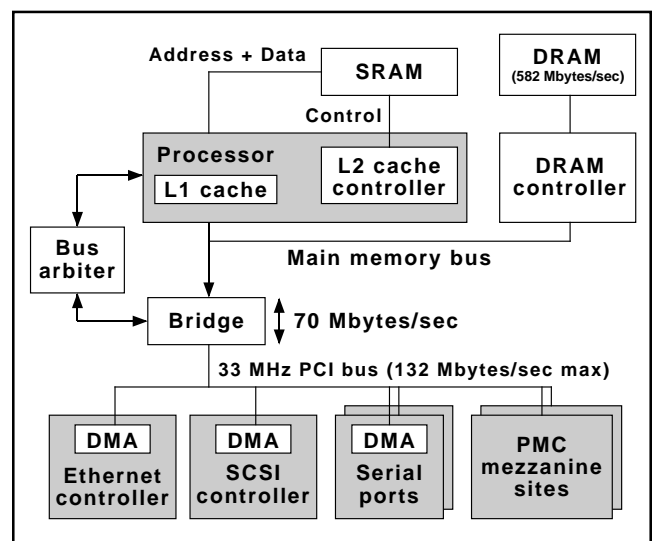


Figure 5

some overhead to each DMA transfer, thus lowering its average throughput.

Making optimum use of the PCI bus bandwidth and the DRAM bandwidth

One way to increase the PCI bus throughput is to provide *data buffers* in the bridge chip, between the PCI bus and the memory bus. These buffers would accept data from the PCI bus at 33 Mtransfers/sec while requesting the memory bus.

The PCI DMA device would not need to wait for the initial latency, and could transfer its data stream to the bridge, even if the memory bus is not immediately available. When the memory bus eventually becomes available, the bridge would empty the contents of its internal data buffer into the DRAM, over the memory bus. Since this buffer could be emptied at the maximum speed of the DRAM (582 Mbytes/sec) most of the memory bus bandwidth would still be left for the processor.

A way to further increase the PCI bus throughput is to double the width of the PCI bus (and the PCI bridge ASIC) to 64 bits. That would provide a PCI bus bandwidth of:

$$(33 \text{ Mtransfers/sec}) * (8 \text{ bytes/transfer}) = 264 \text{ Mbytes/sec}$$

The MVME2400 single board computer

Figure 6 shows the block diagram of the MVME2400 single board computer. The MPC750 processor chip has 32 Kbytes of instruction cache, and 32 Kbytes of data cache. Both of these are 8-way set associative caches, which allows them to handle widely dispersed memory accesses without continuously overwriting their contents.

The data cache can be configured to work in either of two modes:

- copy-back mode
- write-through mode

Copy-back mode

In copy-back mode the data cache only writes data back out to DRAM when the cache line holding that data must be overwritten, or when a DMA access to DRAM is attempting to read the contents of a location corresponding to new data which has been stored in the cache.

Write-through mode

In write-through mode the DRAM is always kept updated – every write operation updates both the cache and the DRAM. (In most applications write-through mode is not used because it can be very inefficient, especially if data is repeatedly written to the same location, as in a tight loop.)

The back-side L2 Cache

The MPC750 processor chip on the MVME2400 board has a built-in L2 cache controller that supports a 2-way, set associative cache, and a dedicated *back-side L2 cache port* for connecting its internal cache controller to external SRAM devices, and the MVME2400 board interfaces 1 Mbyte of SRAM to this port.

The main memory and I/O architecture

In addition to its L1 and L2 caches, the MVME2400 board includes the Hawk ASIC, which provides:

- an ECC SDRAM controller that supports up to 256 Mbytes of 100 MHz SDRAM

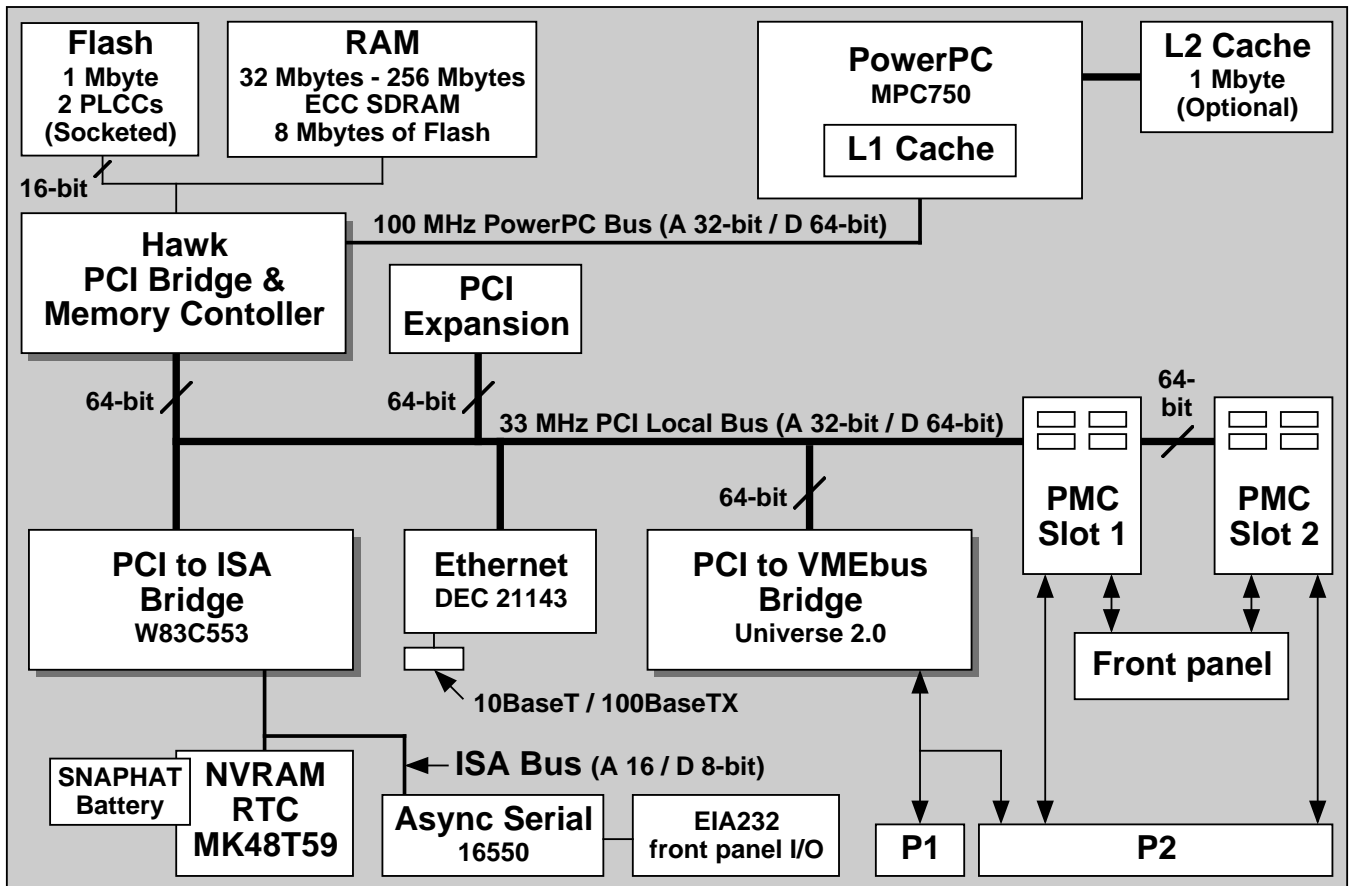


Figure 6

- a Flash controller that supports 9 Mbytes of Flash memory
- a PCI host bridge
- an interrupt controller

Throughput on the MVME2400

As mentioned earlier, there is a latency on the first access when bursting data into (or out of) the SDRAM. However, because this initial overhead is paid only once (at the beginning of each burst) longer bursts provide higher effective throughput.

Figure 7 shows the throughput of both the local PCI bus and the memory bus on the MVME2400, as the size of the PCI DMA transfers to DRAM are varied.

The lower curve in Figure 7 shows how the PCI bus throughput varies as a PCI device performs DMA transfers of various sizes. As shown earlier, the maximum theoretical throughput for a 33 MHz, 64-bit-wide PCI bus is 264 Mbytes/sec. Figure 7 shows that, as the size of the DMA transfers is increased to 1024 bytes, the throughput of the PCI bus on the MVME2400 comes very close to this theoretical maximum.

The PowerPC processor on the MVME2400 board accesses the memory through a 100 MHz, 64-bit-wide memory bus, using 2.5-1-1-1 burst accesses. We can calculate the average number of clock beats per transfer as follows:

$$(2.5+1+1+1) / 4 = 1.375 \text{ beats/transfer}$$

Using this value, we can then calculate the effective data transfer rate over the memory bus:

$$(100 \text{ Mbeats/sec}) * (8 \text{ bytes/beat}) / (1.375 \text{ beats/transfer}) = 582 \text{ Mbytes/sec}$$

Whenever a DMA device on the PCI local bus accesses the SDRAM through the Hawk PCI bridge, it takes control of the memory bus, and forces the processor to wait. This reduces the available bandwidth between the processor and the SDRAM. The upper curve in Figure 7 shows how the remaining bandwidth available to the processor varies as the size of the DMA transfers is varied.

Without the data buffering in the Hawk chip, the DMA transfers over the memory bus would be taking place at the speed of the PCI bus (264 Mbytes/sec) and the DMA controller would need to control the memory bus virtually all of the time. However, with the data buffering there is still plenty of PCI bus bandwidth for the processor to access main memory.

The bridge in the Hawk ASIC between the local PCI bus and the memory bus also allows both buses to operate independently

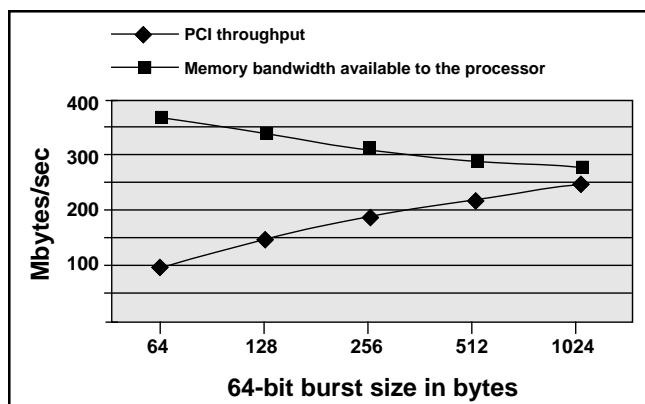


Figure 7

at full speed, except when a PCI DMA device accesses the DRAM.

Throughput with 32-bit PCI devices

While 64-bit PCI chips are becoming more common, most PCI bus traffic is still done as 32-bit transfers. Figure 8 shows how the burst size affects:

- throughput over a 32-bit-wide PCI bus
- available memory bandwidth for the processor

System latency

Burst transfers are a good way to provide high throughput over the PCI bus. However, real-time system designers must balance system throughput against system *latency*.

With longer bursts, system throughput and efficiency is higher. However, the effect is opposite for the system latency, which is particularly important for real time applications. System latency can be improved by:

- using ultra-fast main memory
- providing read and write buffers between the PCI bus and main memory

Figure 9 shows the time required for the MVME2400 board (which employs both) to complete bursts of various sizes using both 32-bit-wide and 64-bit-wide burst transfers. As shown, a burst of 256 bytes would take about 2 μsecs to complete, when done with 32-bit-wide transfers. The same trans-

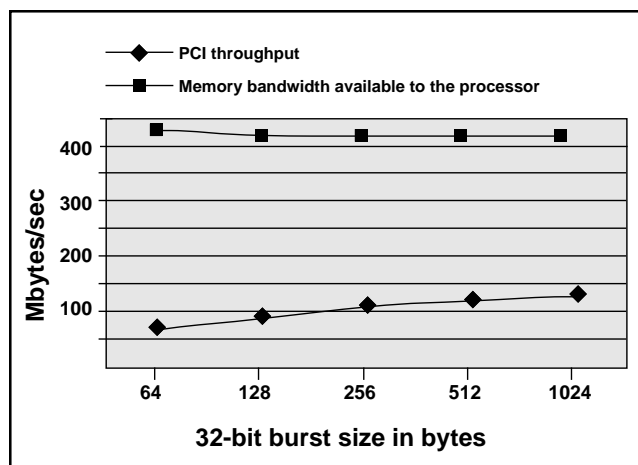


Figure 8

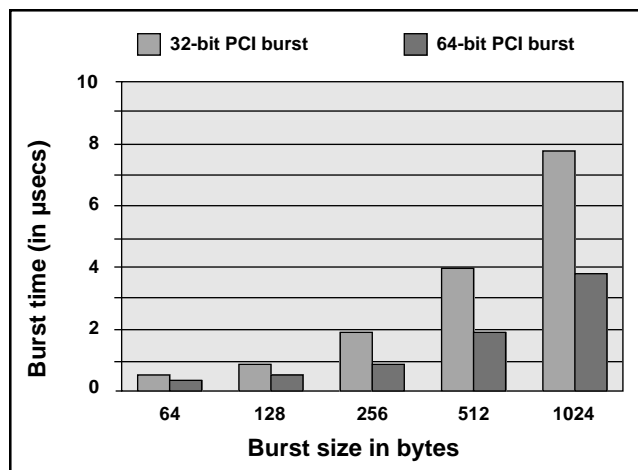


Figure 9

fer would only take half as much time when done in 64-bit-wide bursts.

Summary

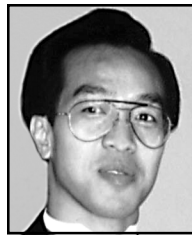
The MVME2400 offers an unprecedented level of system performance by:

- optimizing the processor/cache interface for performance
- designing the main memory architecture to provide high throughput
- maximizing the system I/O throughput

With its 1-Mbyte L2 cache, its 256 Mbytes of high-bandwidth ECC SDRAM, and its Hawk ASIC, the MVME2400 takes the fullest possible advantage of the 350 MHz MPC750 PowerPC processor chip. Its two 64-bit PMC mezzanine sites, and its 64-bit-wide onboard PCI bus also make it ideal for demanding real-time applications. Ω

References

- [2] MPC750 RISC Microprocessor User's Manual, MPC750UM/AD
- [3] PCI Local Bus Specification Revision 2.1



Chau Pham is the director of Advanced Product Development for the Cross Industries Business Unit at Motorola Computer Group. Chau was one the main contributors for the VME64/VME64X Standards and the 2eVME/2eSST Protocols. He also served as the vice chairman for the PCI Mezzanine Card (PMC) Standard. Prior to joining Motorola Computer

Group in 1984, Chau was a design engineer at Space Data Corporation, where he worked on various NASA related projects. Chau earned his Bachelor Degree in Electrical and Computer Engineering at Arizona State University.

If you have questions about this article or if you would like more information about the MVME2400 single board computer, you can contact the author at:

Motorola Computer Group
2900 South Diablo Way
Tempe, AZ 85282 USA
Email: chau@mcg.mot.com