

# The Right Bus In The Right Place - A Tutorial

## Part I

*A lot of parallel busses are in use today. Some of them are competitive, others are complementary. The misuse of a bus may have dramatic results in a project. This paper explains different profiles of bus use. A system architect should always take into account the basic rules and reduce the risk of bad system behaviour.*

*By Dr. ir. M. Timmerman,  
Managing Director,  
Real-Time Consult.*

### Introduction

#### A - Roadmap

This paper subdivides bus use in different profiles or types of system architecture. Profile 1 is a simple single processor bus. Profile 2 is a cache supporting bus in a tightly coupled multiprocessor system. Profile 3 is a loosely coupled multiprocessor system with a master - slave Operating System structure. Profile 4 is identical to profile 3 except for a master - master Operating System architecture.

#### B - Single processor systems

Simple systems (Figure 1) can easily be implemented with one processor (Profile 1). If more performance is needed, faster processors can be used or you can enhance bus bandwidth by going from 8 to 16, 32 and even 64 bit data transfers. The use of cache may enhance overall system performance. However, there is an end to what technology can do for you today and if one processor is not enough, why not use several?

#### C - Multiple processor systems

Two very fundamental different solutions are in use to solve the problem of bus contention in multiprocessor systems. The first solution uses cache to reduce the bus load (Profile 2). The other puts enough memory on a particular board. All program code is then located there and no more code execution bus-traffic is needed (Profile 3 and 4).

Both solutions are used today. Each solution has its advantages and drawbacks. We will study these different architectural solutions and therefore look towards the type of traffic on the bus, the bus load produced by this type of traffic, and the board func-

tionality needed to support particular communication protocols between intelligent boards. We finish by giving some examples of commercially available backplane and motherboard busses.

(Remark: in all timing diagrams, the time scale is a relative one. An absolute time reference depends on the speed of the processor.)

### Profile 1 Bus: Simple Processor Bus

#### A - Board Bus and Backplane Bus

Each processor has a number of address, data and control lines to connect to memory and peripheral devices. If you design a board with all these components, you create a board bus (Figure 1). The traffic you will find on such a bus is code execution traffic, I/O traffic and interrupt handling.

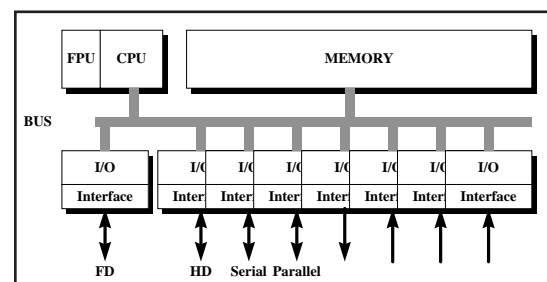


Fig 1. Simple computer architecture.

Code execution traffic means that some cycles on the bus will be for instruction fetch operations and others for instruction execution, which in most cases needs data transfers on the bus. Typically, each address of each consecutive cycle or transaction on the bus will have a random address.

You can now decide to propagate this board bus

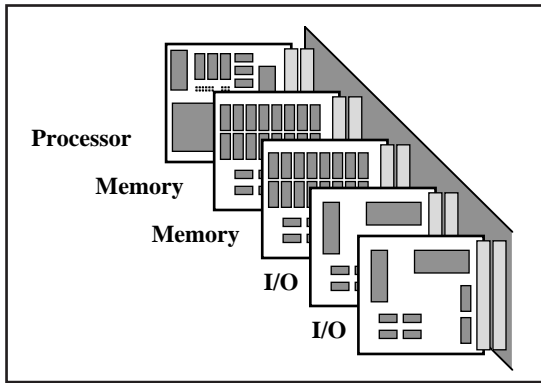


Fig 2. Simple backplane.

on a backplane or (flat)cable and extend your system in a modular way (Figure 2).

There is no difference in traffic compared to the previously described board bus if only memory boards and non intelligent I/O device boards are in use. The bus qualifies as a processor bus, a memory extension bus or a (non intelligent) I/O bus. (Figure 3)

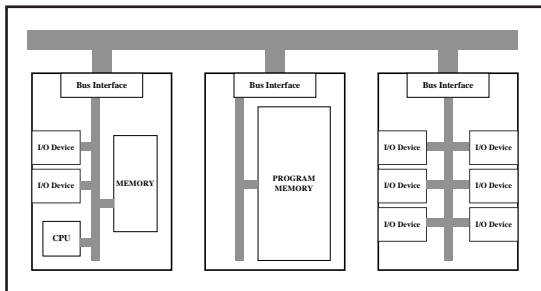


Fig 3. Profile 1 bus system.

Most systems just use one processor. Sometimes a Direct Memory Access Controller (DMAC) is added to enhance I/O to memory traffic. In this case there are two bus masters (the processor and the DMAC).

**B - Bus Definition**

A profile 1 bus is a bus characterised by code execution, I/O traffic including interrupt handling (Figure 4). Sometimes it has a simple arbitration mechanism to provide for a limited quantity (2) of bus masters. (Processor + DMAC).

In general, we call this a processor bus. We can call it a memory bus if the system contains only memory extension boards. In this case we will not find any interrupt handling capabilities.

If the extension boards are only simple I/O device boards, then we can call it a (non intelligent) I/O bus. Extensive interrupt capabilities are a characteristic here. The memory bus and I/O bus are often used today in mezzanine technology.

**C - Board functionality**

The boards on a profile 1 bus have no special functionality for handling higher level bus protocols like in the ones we will discuss later. We are in a low cost environment.

**D - Commercial products**

The implementation of a profile 1 bus takes all kinds of forms. Apart from on board and backplane busses, there are flatcable busses and now more and more mezzanine or piggy-back busses.

Almost all commercially available boards may be used in this mode. Examples are: ISA and EISA, MCA, Multibus 1, VMEbus, Sbus, PCI and PMC, IP, M-Modules Bus, STD, STE, etc.

**Bus Load: the problem.**

We define bus load as the percentage of time the bus is in use for actual transfers, cycles or transactions. During these bus activities, no other data transfers can take place. Depending on what a processor does, how it works (with pipe-lining or not), if it is using cache or not, a processor needs between 40 and 70% of the bus-time.

Suppose a processor wants to be bus master to execute code on the bus. Fetch and execute cycles will go in a random way over the bus. To make it simple, we assume here that such activities need 50% of the bus-time (Figure 4).

**A - From one to more processors.**

One processor never uses 100% of the available bus-time. Therefore we will study what happens if more processors want to use the same bus.

Case A: (Figure 4) Only one processor uses the bus. We observe that only 50% of the total available

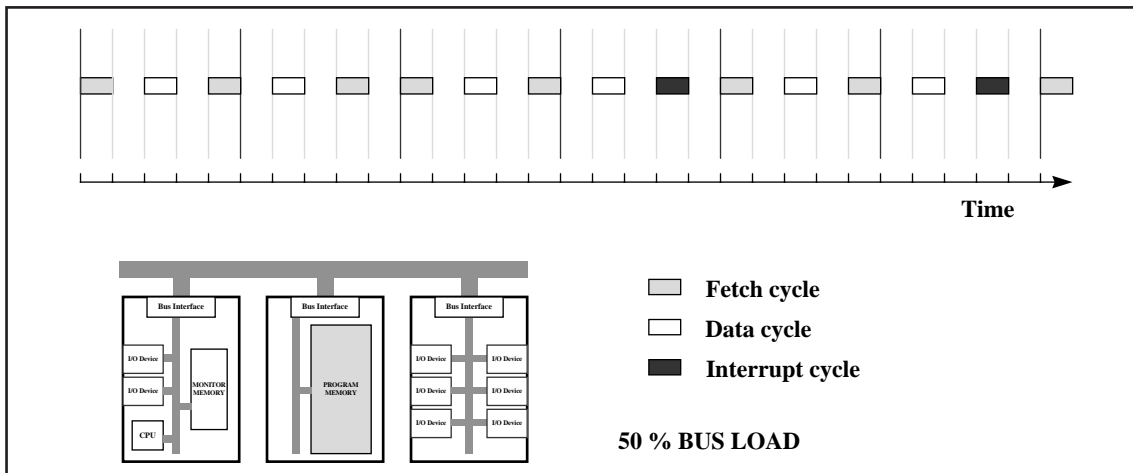
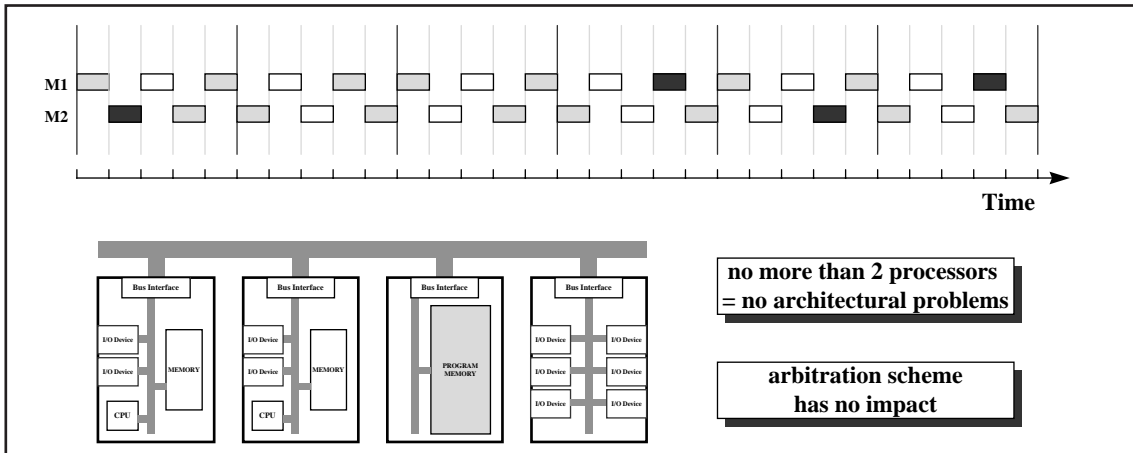


Fig 4. Profile 1 traffic.



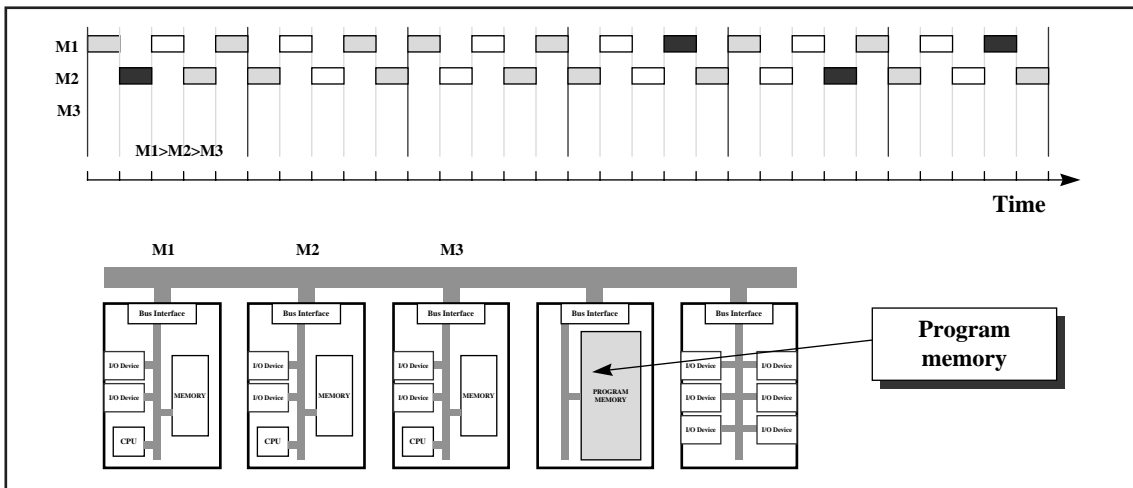
*Fig 5. Increase bus load: 2 processors.*

bus-time is in use what qualifies as ineffective use. The processor works full speed.

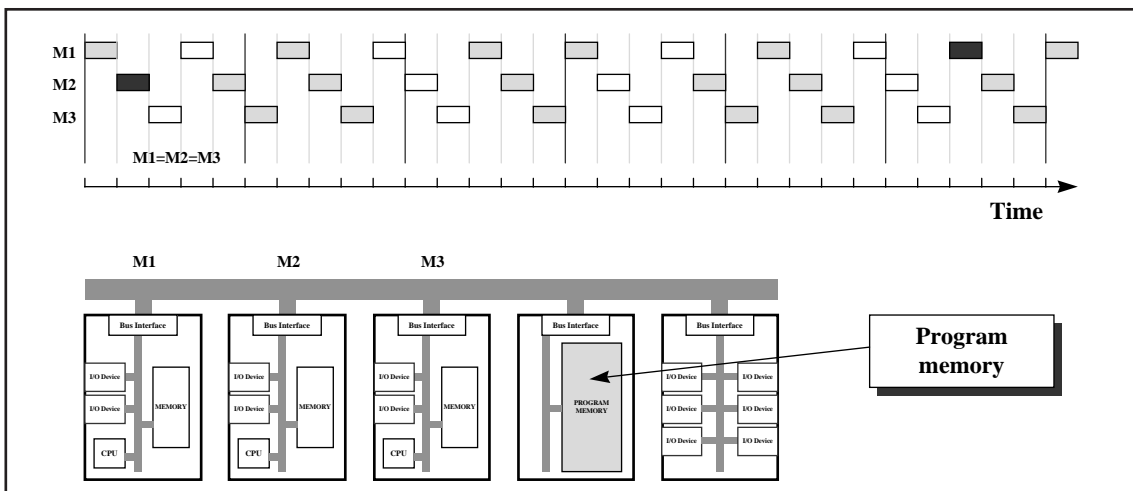
Case B: (Figure 5) Two processors, doing approximately the same activities (they both execute code on the bus), are competing for the bus. Suppose bus arbitration is done without losing time (which should be the case in parallel busses, but is rarely the case in practice (3)) then 100% bus-time is available for

these two processors. They will share the bus equally independently of the arbitration mechanism!

Case C: (Figure 6) Three processors are competing for the bus. Here the arbitration mode is starting playing a role. Let's assume here that we use in this case priority mode of arbitration. This means that one processor is given more priority bus access compared to others.



*Fig 6. 3 processors - priority arbitration.*



*Fig 7. 3 processors - round robin arbitration.*

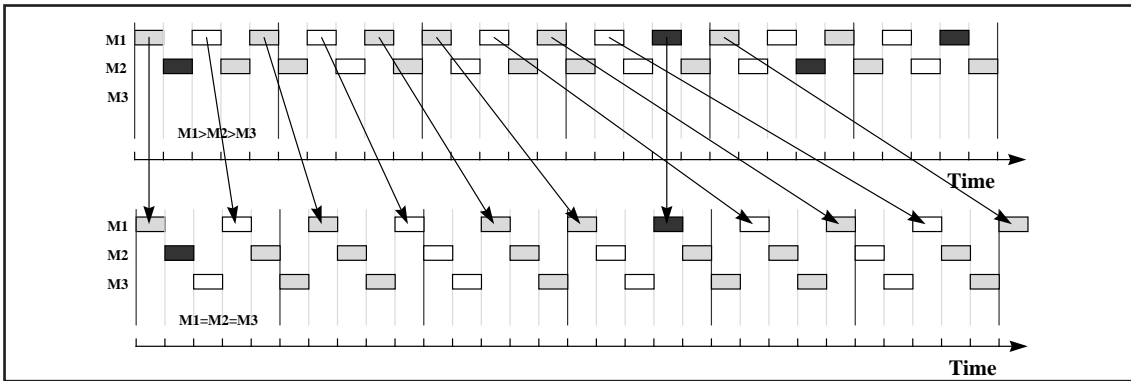


Fig 8. priority versus round robin comparison.

A fundamental problem occurs in this situation: the processor with the lowest priority will never get the bus. It will probably generate an on board bus error. (4)

Case D: (Figure 7) Three processors are competing for the bus with a round-robin bus arbitration mechanism. They get an equal share of the bus.

The system works perfectly but the processors slow down to 33% of bus use each. Each board will have to wait longer to have the bus available. During that time, the board generates on-board wait cycles. The performance of each board gets down (Figure 8). This may be a serious problem in Real-Time Systems.

**B - Conclusions**

Conclusion 1: Two processors competing for a bus will always share the bus without major problem. The chosen arbitration scheme has no importance.

Conclusion 2: From three processors on, there may be problems: Either one (or more) processors will never get the bus, or each board will drastically slow down (by increasing the bus-time out value (4)). There is definitely something wrong here. Indeed we are doing a fundamental mistake: executing code on a bus with more than 2 processors obliging each processor to ask for the bus for at least 50 % of the time.

**C - Problem solving**

We now will introduce two total different solutions for this problem. The first method of reducing bus load is to use a cache mechanism. The second one

is to put all program code on board. The first mechanism is used in number crunching applications, the second in Real-Time Systems.

**Bus Profile 2 : Cache Supporting Bus.**

**A - Introduction**

The use of cache can reduce the total bus load. We will not try to do any mathematical or statistical calculations here to determine the load reduction invoked by the use of cache. Let's keep it simple and just try to understand the principles. We just use one level of caching here. (Figure 9)

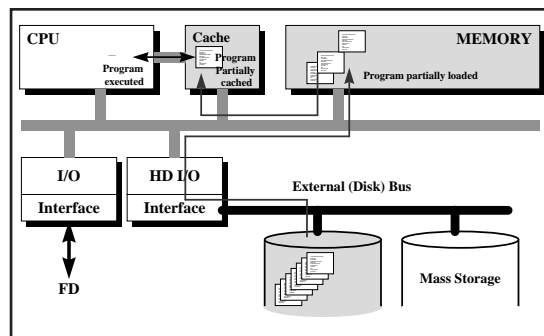


Fig 9. Cache use.

Case A: The processor loads its cache by doing consecutive "move" cycles on the bus. Once done, the processor executes code and does not need the bus anymore.

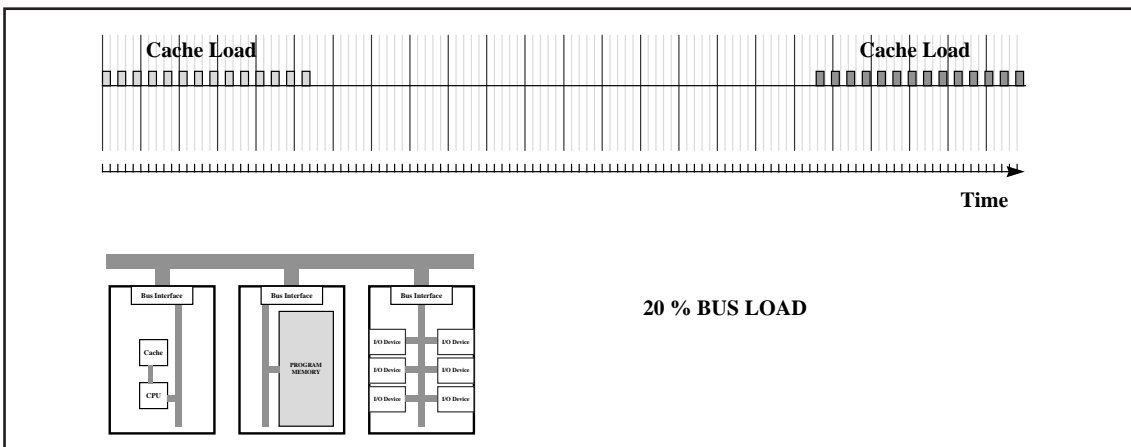
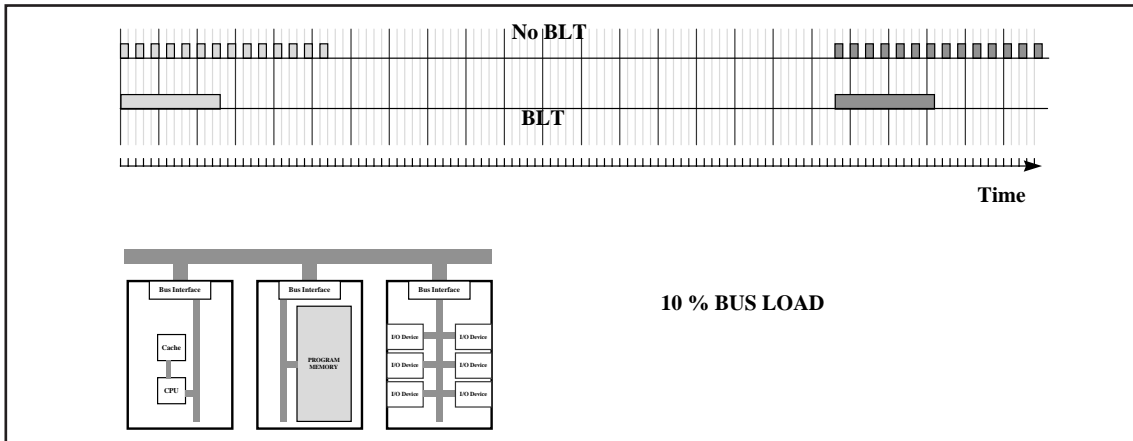


Fig 10. bus effect of cache use: lower bus load.

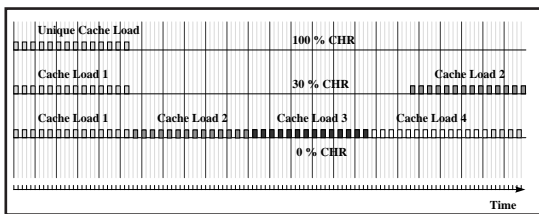


**Fig 11.** cache load without and with block transfers.

We observe no more bus activity (except for write through cycles: see later), thus freeing bus-time for other processors. (Figure 10)

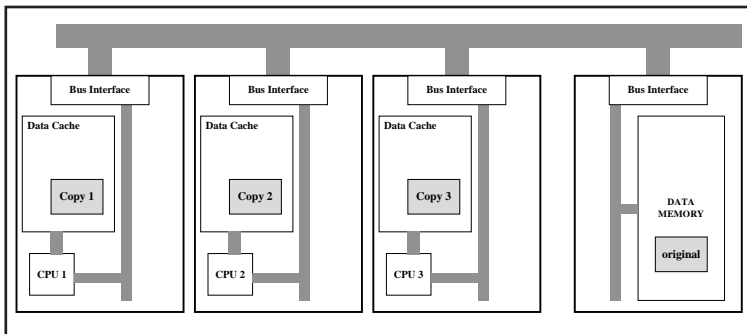
Case B: (Figure 11) The cache loads more effectively by a block transfer cycle. Even more time is available on the bus but the (atomic) bus cycles are longer. This introduces more interrupt latency that might create Real-Time System behaviour problems.

It is easy to understand that more free time is now available, however it is difficult to compute exactly how much. Indeed, the type of code executed, the characteristics of the processor and its cache, the number of data write-cycles which will involve write-through cycles (see later) are a selection of the many statistical factors involved.



**Fig 12.** cache hit rate examples.

The extreme case is where the program and data fit completely in the cache. In this theoretical case, you free the bus 100% (Figure 12). This means that no processor is using the bus, except for the initial cache load. In this case a very large number of processors can run on the bus, just because they do not use it. In fact this corresponds to an old criterion of mine saying that "the best way of using a bus is not to use it".



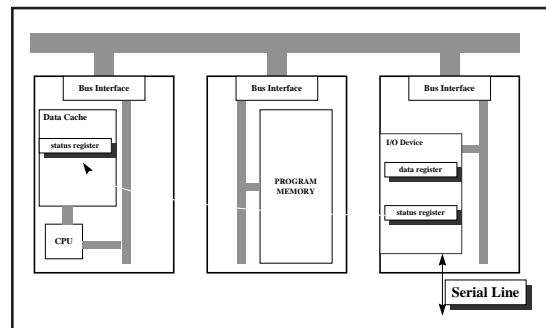
**Fig 14.** cache coherence - the problem.

This seems a ridiculous statement, but it means that the traffic on the bus should be kept as low as possible.

The use of cache reduces bus load (in a statistical way). However, there is a price to pay: bus complexity. The use of cache should be completely transparent to the programmer. The bus should provide special features like I/O register cache avoidance and cache coherence support.

**B - Do Not Cache I/O Registers**

Let's take a simple example to explain the problem: Suppose we have a non intelligent I/O board with a serial chip on it. (Figure 13) Instead of using interrupts, we will use a polling program to detect the arrival of a character.



**Fig 13.** I/O register caching - a problem.

If data-cache is enabled and no special measure has been taken, you will poll the contents of the copy of the I/O status register chip copied in cache during the first attempt to read it. If now the external event arrives, the original status bit will change in the chip, but the program will continue polling on the non-updated cached copy.

To solve this problem, the bus should have a provision to tell the processor what is cacheable or not. A cacheline signal is a solution. However some busses do not have such a line.

**C - VMEbus Example**

The VMEbus has no cache line

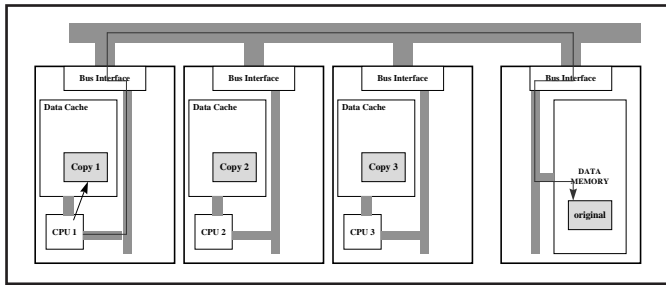


Fig 15. write through to update the original.

so the user or manufacturer has to take care and provide for a solution:

**Solution A:** Do not use data cache.

**Solution B:** Use of data cache and disable it via software before I/O transactions. The problem with this solution is that the software becomes totally hardware dependent.

**Solution C:** The board manufacturer provides a mechanism that disables the caching of I/O registers via a hardware scheme connected to the master address map of the board. For example: The "SHORT" address space in VMEbus should not be cacheable.

The user has to find out in the board manual how the board reacts. However, documentation is not always clear on that subject. Great care should be taken here. A little life-test, as the one described above, will clarify the board behaviour.

#### D - Cache coherence support

A problem may arrive if two or more processors are caching shared data.

Figure 14 indicates a configuration with three processors sharing the same data. The original is copied 3 times in the respective data caches. As long as the processors are reading the data, there is no problem. If one of them starts writing in the data cache then there is a first problem of updating the original. Usually a "write-through" cycle solves this problem (Figure 15). The value updates in the cache and via the bus in the original memory. (Every write-through cycle runs at the speed of the bus access!)

This action ensures the correctness of the original in a single processor environment. There is a problem however in multiprocessor systems. In our example we have a second and third copy on other processors.

The copies are not valid anymore. Several solutions are possible but it will take us too long here to treat them. Let's mention the simplest solution that invalidates the copies during the write-through cycle. This should happen transparently to the software. The bus interface and bus characteristics must provide for the solution.

If your bus has no provision for cache coherence, the user should take very great care. Do not cache shared data in that case by not using data cache or not using shared data.

#### E - Bus Definition

A profile 2 bus is one that provides for complete software transparent cache support.

#### F - Board and System Functionality

Global memory in which all code resides, is a characteristic in such an architecture. Each processor can execute any code. The group of processors (mostly 2 or 4) are the "processor engine" or "virtual processor". A symmetrical OS, like some UNIX's and one Real-Time OS like MTOS can do automatic load balancing. (5).

#### G - Commercial products

The choice of the cache-coherence support provides for a lot of different commercial solutions. Most manufacturers have therefore developed a proprietary solution for this architecture.

Futurebus+ fits in this category, however Futurebus+ has so many features that I consider it as a monster bus. Who wants to use an (expensive) monster?

The M-bus from Motorola is another example of such a profile 2 bus for parallel 88K use. The technology has been reused in a PowerPC environment. It supports the cache coherence techniques used by these processors.

Sun is also using an M-bus. However it is not the same M-bus as the one from Motorola. (Typical naming problem in the computer world). The principles are the same.

VMEbus is not suited for the use as a profile 2 bus.

For more information about cache coherence we refer to chapter 7 of (6). ■

**This article will be continued in issue 1Q97 of Real-Time Magazine.**

---

*Dr. ir. Martin Timmerman is graduated in Tele-communications Engineering at the Royal Military Academy (RMA) Brussels and is Doctor in Applied Science at the Gent State University (1982). He is giving general courses on Computer Platforms and more specific courses on System Development Methodologies at the Royal Military Academy Brussels. Outside RMA, Martin is known for his audits, reviews and workshops on system architectures, Real-Time Operating Systems and software engineering for Real-Time Systems. He is also editor-in-chief of Real-Time Magazine.*

#### References

1. P. KORTMANN and G. MENDAL, PERTS Prototyping Environment for Real-Time Systems Real-Time Magazine 1Q96 (ISSN 1018-0303)
2. T. NYGAARD, Using bus analysers to find and debug VMEbus problems. Real-Time Magazine 3Q94
3. M. TIMMERMAN, The Right Bus on the Right Place, Real-Time Magazine, 1Q 1993
4. M. TIMMERMAN, Bus error flaws in VMEbus Systems, Real-Time Magazine, 2Q 1992
5. M. TIMMERMAN, Workshops on RTOS - info at URL <http://www.realttime-info.be/realttime-consult>
6. J. ZALEWSKY, Advanced Multimicroprocessor Bus Architectures, IEEE Computer Society Press ISBN 0-8186-6327-8
7. J. ZALEWSKY, What every engineer should know about Rate-Monotonic Scheduling: A tutorial, in (6) and in Real-Time Magazine 1Q 95