C *omputing*
S *urface*

Communications Processor Overview

S1002–10M100.04

meiko

Meiko's address in the US is:

Meiko
130 Baker Avenue
Concord MA01742

508 371 0088
Fax: 508 371 7516

Meiko's full address in the UK is:

Meiko Limited
650 Aztec West
Bristol
BS12 4SD

Tel: 01454 616171
Fax: 01454 618188

Issue Status:

Draft	
Preliminary	
Release	x
Obsolete	

Circulation Control: *External*

Contents

1. Overview 1

2. Inter-processor Communications 3

Latency and Bandwidth 4
Network Security 4
Virtual Addressing 5

3. Elan Functionality 7

Checking 7
Translation 8
Copying 9
Device Control 10
Thread Processor 10
Thread code 11
Events 11
Other Forms of Remote Access 12

4. Using the Communications Processor 13

DMA Transfers 13

5. Conclusions 15

Overview *1*

Effective cooperation between processing elements is a crucial factor in determining the overall performance of an MPP system. Maintaining effective interprocessor communication as a system scales in size is a vital aspect of preserving balance.

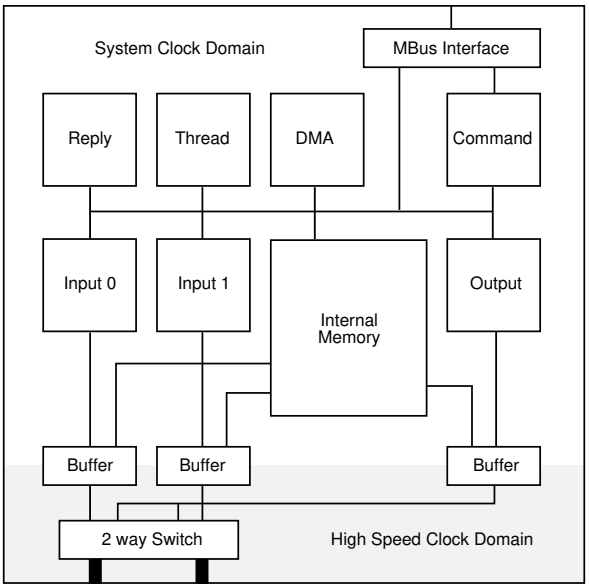
In designing the CS-2 architecture Meiko has concentrated on minimising the impact of sharing work between processors. The effect of this is to increase the number of processors that can be used effectively to solve a problem, improving the performance of existing parallel programs and making parallel processing efficient for a wider range of applications.

Every processing element in a CS-2 system has its own, dedicated interface to the communications network: a Meiko designed communications processor. The communications processor has a SPARC shared memory interface and two data links. Data links are connected by Meiko designed 8×8 cross-point switches. Data links are byte wide in each direction and operate at 70 MHz, providing 50 Mbytes/s of user bandwidth in each direction.

The communications processor supports remote read, write and synchronisation operations specified by virtual processor number and virtual address — both are checked in hardware. Latency hiding is supported by non-blocking instructions, instruction sequences and completion tests.

This document provides an overview of the design of the communications processor and its usage. For more information about the architecture of the data network see the *Communications Network Overview.*

Figure 1-1 The Elan Communications Processor



Inter-processor Communications 2

In a distributed memory system work is shared between processors by exchanging data over a communications network. The efficiency of data exchange controls the effectiveness of work sharing and hence the number of processors that can be used on a given problem.

Rather than design a new processor with built in communications capability Meiko chose to separate the issues in the design of the CS-2. Processing elements consist of a high performance RISC CPU (with optional vector processing capabilities) and a dedicated communications processor.

The interface between the communications processor and the rest of the processing element is central to the efficiency of the CS-2 network. It provides the following essential features:

- Low communication start-up latency.
- High bandwidth inter-processor communication.
- Security against corruption.
- Operation in a network-wide virtual addressing, virtual process environment.

Latency and Bandwidth

Efficient inter-processor communication requires both low latency and high bandwidth. While solutions to the bandwidth problem can be addressed by ever improving hardware technology, these improvements only exacerbate underlying latency problems.

To show that this is the case consider a system with a communications start-up latency of 10µs. To transfer a 100 byte message via a 1Mbyte/s network we will get an achieved bandwidth of 0.9Mbytes/s (90% efficiency). For the same transfer over a 50Mbytes/s network, the achieved bandwidth is just 8.3Mbytes/s (16% efficiency). Clearly the improvements in bandwidth for this example system have been severely limited by the start-up latency and the size of the data transfer.

By using a dedicated communications processor Meiko have reduced start-up latency by implementing in hardware the communications code that would normally execute on the main processor.

The data links joining communications processors and network switches are byte wide in each direction. Links are clocked at 70MHz. Their bandwidth after protocol is 50MBytes/s in each direction. The CS-2 data network is a fat tree with constant bandwidth between stages. It is capable of supporting full bandwidth transfers between all pairs of processors (see the *Communication Network Overview* for more details).

Moving communications code from the main processor to a communications engine does not in itself reduce latency. Performance improvements come from running the right code in the right places. In particular there are significant benefits to be had from moving the lightweight interrupt intensive operations associated with inter-process communication off a conventional microprocessor and onto a communications processor designed specifically for this purpose.

Network Security

The CS-2 communications network is shared by both user and system level communications so it is vital that a security mechanism is used to prevent unrelated communications from interacting. To relieve the burden of checking from the main processor and to reduce start up latency, the main processor is-

sues unchecked communication instructions to the communications processor, the communications processor then implements the security strategy in hardware. This mechanism is preferable to the more conventional use of kernel mapped devices, which use checked system calls to access the device, often with a significant performance impact (a checked system call in a 40MHz SPARC takes approximately 50µs). The CS-2 network protects processes from communications errors that occur within other unrelated processes, but does not protect a process from errors within itself. This is the same model as that employed for memory protection by the UNIX operating system — processes are protected from each other, but not from themselves.

Virtual Addressing

The communications processor uses separate page tables from the main processor. This means that a user process need not make its entire address space visible when it communicates, only the portion that contains the data need be mapped for communication. Secondly, separate page tables may be used to reduce the amount of cache flushing in non cache-coherent systems; in a write through cache only those pages that are mapped with write permission need be flushed.

The two sets of page tables are kept in step by a modified page out daemon and new page in code in the operating system. The modified page out daemon modifies both sets of tables, whereas the new page in code handles the asynchronous page faults from the communications processor.

Elan Functionality 3

The functionality of the communications processor was decided by drawing on experience from Meiko's CSTools/CSN communication software, used to create a programming environment over Transputer networks, and other message passing systems such as the Chorus Nucleus. This analysis showed that the start-up process consists of four components:

- Checking.
- Translation.
- Copying.
- Device control.

Each of which is important if start-up latency is to be minimised.

Checking

The CS-2 supports virtual memory addressing on each processing element, allowing it to implement a fully distributed store for operating system use, and permit it to implement the applications binary interface (ABI) for the base microprocessors. The communications processor therefore has two types of parameters to check: memory addresses and process addresses.

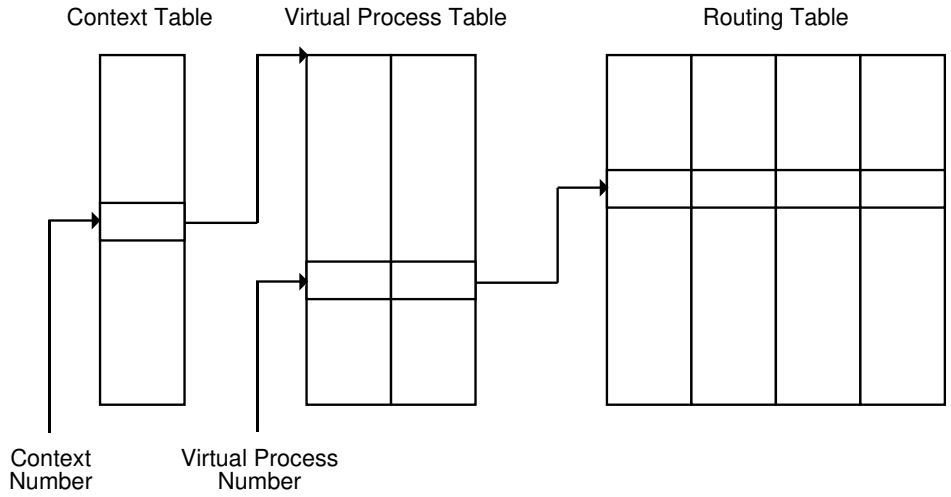
The communications processor receives unchecked virtual memory addresses from the main processor so it must incorporate a memory management unit (MMU). The MMU used within the Elan supports multiple simultaneous contexts allowing I/O to continue for suspended processes.

The checking of process addresses is analogous to the checking of memory addresses. It is implemented by a simple table look-up and exception mechanism. The communications processor is designed to handle the common case where a user is trying to communicate with other processes for which it has permission; an exception is generated whenever there is no permission. As checking is performed independently on each of the communications processors, failed processing elements can be removed from service by removing them from each communications processor's list of valid destinations.

Translation

Process and memory translation within the communications processor is implemented through the same mechanism as the checking, that is, by table look-ups. Memory address translation yields the same results as the main processor's translation mechanism. Dynamic process translation yields two components: a destination processor and a destination context. There are no physical processor or memory addresses in user space

Figure 3-1 Elan Process Translation



Virtual process IDs are translated through a per context virtual to physical processor translation which points at the route bytes needed to direct a message to this processor.

Copying

The communications processor supports a number of features to remove the requirement for copying of data. By using network wide virtual addressing there is no need to copy data into physically mapped output buffers, a common technique in distributed systems to overcome the problems of virtual address translation and page locking during communication. Furthermore, because the main processor and the communications processor share a common memory bus (a SPARC MBus) and the same cache coherency protocols, the problems associated with cache coherency are also avoided.

Clearly the avoidance of unnecessary copying contributes greatly to reduced start-up latency and efficient use of memory bandwidth. For messages that are copied once on sending, this adds $(messagesize \times 2) / (memorybandwidth)$ to the start-up latency, and consumes three times as much store bandwidth.

Device Control

The final requirement of message start-up code is in device control. This is setting up the communications parameters in store, signalling to the communication device, and responding to interrupts returned by the communications processor.

Control of the communications processor is via a command port which is normally mapped into the user address space. The command port consists of a range of memory addresses. The communications processor command is determined by extracting 5 bits from the address that is used. The data that is used by the communications processor command corresponds to the 32 bits of data that are written to that memory address. Commands sent to the command port are written in a single read-modify-write cycle and are acknowledged with the value that is read back (which will be non-negative if the command is accepted). The kernel can prevent the user issuing certain commands by mapping limited portions of the command port address space in to the user address space.

Exceptions generated by the communications processor may be handled by the communications processor's own thread processor, without direct intervention by the main processor.

Thread Processor

One of the objectives of the Elan communications processor is to reduce the number of interrupts and system calls that must be executed to perform message passing. As we have seen the combination of the user mapped command port and the Elan communication processor's security mechanisms allows user level code to initiate remote memory accesses without making a system call. In many cases, however, message protocols require higher level functions than simply the transfer of data. Other common requirements are for synchronisation between processes executing on separate processors, and allocation of global resources. To support these requirements the Elan communications processor includes a RISC processor which can execute user level code independently of the main node processor, and also create additional network transactions.

The hardware and microcode of the thread processor support an extremely lightweight scheduling mechanism. This allows lightweight processes (threads) running on the thread processor to be suspended and then rapidly rescheduled by the hardware when the relevant event has occurred.

The user level code in the main node processor can directly request the execution of a thread process through access to the appropriate command port. The thread code has no more privileges than the user code which initiated it. The Elan communications processor uses its page tables for the relevant user context whenever it makes a store access from the thread.

Thread code

Thread code can be written in ANSI C. An inlined library provides access to the Elan communication processor I/O instructions without the overhead even of a subroutine call.

Events

Events provide a general mechanism by which synchronisation may be achieved between lightweight threads running either in the same, or different, Elan communication processors. In addition an event can be used to cause an interrupt to the main node processor. An event is represented by a double word in store.

A thread can perform the following operations on either local or remote events:

Wait	If the event has already been set, then execution continues and the event is unset. Otherwise the thread is suspended on the event until the event is set, when it will be rescheduled.
Set	The event is set. If there was an action already present on the event then it is performed.
Clear	If the event was set it is cleared.
Test	Poll the status of an event without modifying or suspending on it.

There are various possible actions which can occur when an event is triggered, these depend on what has been suspended in the event structure:

A local thread	The thread is placed back on the thread run queue, so will resume execution.
A remote thread	The remote thread is rescheduled on its own processor.
A local interrupt	The main processor is interrupted.

Events also support queues of outstanding requests. When a queued event is set, the first action on the queue is executed, and the queue updated to point to the next action.

Other Forms of Remote Access

In addition to events, the Elan also supports other forms of remote store access. In particular thread code can generate network transactions to perform:

Atomic Swap	The word at the given remote address is returned, and overwritten with the word sent in the message.
Atomic Add	The word sent in the message is atomically added to the data at the remote address. The original remote data may optionally be returned.
Atomic test and store	The word at the remote address is compared with a test value sent in the message. If equal then a new value sent in the message is written to the remote store, otherwise the remote store is unchanged. The original remote value may optionally be returned.
Remote compares	The word at the remote address is compared with the given data using one of the operations ==, =, >= or <. The result of the comparison is returned as an acknowledge or negative acknowledge.

The broadcast capabilities of the Elite switch can be used to combine the results of a broadcast remote compare operation into a single result.

Using the Communications Processor *4*

In this section we show in outline how the communications processor is used to communicate with other processes via the data network. The example shows how to initiate a DMA transfer to remote store.

DMA Transfers

In the previous sections we have seen that a key factor in the design of the communications processor is that it offers low communication start-up latencies, and that communication start-up requires minimal intervention by the main processor. For a typical DMA transfer of data to a remote processor, the actions required by the main processor are as follows:

- User program creates a DMA structure in store identifying the characteristics of the transfer (source and destination addresses, amount of data, etc). This could be done in advance if the same access is to be made repeatedly.
- User program issues DMA command with RmW to command port. The address of the DMA structure is written to the appropriate address in the command port.
- User program checks command accepted; a value of greater than or equal to 0 in the command port indicates that the command was accepted.

The main processor is now free to continue with its work leaving the communications processor to transfer the data, and to ensure its integrity. The actions now required by the communications processor are:

- Command processor reads the 32bit data from the command port and uses this to locate the DMA descriptor. The descriptor is read into the communications processors DMA queue.
- DMA processor reads the queue item in.
- DMA processor performs destination process translation.
- DMA processor reads route information.
- DMA processor reads source data in and starts to send. The route information is prepended to the data, and is stripped off as it passes through the switch network.

If the main processor wanted confirmation that a DMA had completed it would include a pointer to an event in the DMA description. Polling this event (when there is no more useful work to do) would confirm completion of the transfer.

Conclusions *5*

Efficient inter-processor communications requires the right balance of latency and bandwidth. CS-2 uses Meiko's own communication hardware, developed from many years experience in the massively parallel processing field, to create a network with both high bandwidth and low start-up latency.

The Elan communications processor is key to minimising the network latency. It serves not just as a communications co-processor, but aims to minimise the amount of message start up code, and therefore minimise start-up latency. For simple communications the overhead on the main processor can be reduced to a single read modify write. More complex protocols require small fragments of code to be run on the communications processor. The requirement for copying of messages is removed by the ability of the communications processor to operate in virtual store. Protection is implemented by hardware table look ups of translation tables which impose low overhead on valid operations, and generate exceptions in the much less frequent error cases.