CSL Notebook Entry

To:	CSL Notebook	Date:	October 5, 1981
From:	Anthony West	Location:	PARC-CSL
Subject:	Auxiliary Hardware for D-Machines	File:	[Ivy]<CSL-Notebook>81CSLN-0050

XEROX

Attributes: technical, Alto, Computer organization, Communication, Database, Distributed computing, Dandelion, Dolphin, Dorado, DES, Cryptography, Encryption, Random Number Generators, Noise Diodes, Stable Storage, Atomic Transactions, WWVB, Broadcast Time Standards, Stable Storage, Universal Identifiers, Synchronizing Events, SLC Ethernet, I/O Interfaces, Intel/IEEE Multibus

References: [Needham] 81CSLN-0026; [Lampson] 81CSLN-0030;

Abstract: This note describes the evolution of the proposal to add to CSL's D-Machines hardware to support miscellaneous auxiliary functions, such as DES cryptography, random number generation, removable electronic stable storage, a receiver for the WWV broadcast time standard, and assorted external I/O interfaces.

1. Introduction

Earlier proposals suggested that CSL systems be augmented with hardware to enable research in a number of new areas. This note draws together the suggestions of a number of people on these issues and summarises the state of the design.

Under discussion is hardware for:

DES Cryptography

Random Number Generation

Electronic Stable Storage

Receiver for the WWV Broadcast Time Standard

External Interfaces

2. Strategy

The areas of research which the hardware would enable fall loosely under the heading of *Distributed Systems Research*. A realistic study environment requires that a significant number of machines will need the additional hardware, and this implies a considerable investment of funds and effort. Careful thought is required to identify the research benefit this hardware will yield compared with the investment CSL will have to make in building it.

2.1 What is the Purpose?

A major research issue in distributed computing is how to provide protection and controlled sharing of information.

The provision of high-performance *cryptography hardware* would enable us to study security issues in distributed systems. The adoption of the NBS DES algorithm as a federal standard and the availability of high-speed encryption chips effectively eliminates any other techniques from the

domain of discussion.

The availability of DES encryption hardware would enable us to build into Cedar facilities for *secure communication* (eg. in remote procedure calls) and *secure data storage* (eg. in databases and Alpine file servers). The project is timely with respect to Cedar's current state - in other words - the Cedar community is an attractive, small, closed user community which could be equipped with cryptography facilities in a uniform fashion.

An inherent characteristic of encryption is that it tends to be an all-or-nothing facility. Thus, although we can incorporate it into the new world of Cedar relatively painlessly, it is less simple to do so for services already in operation. It would be attractive to add secure communication to Grapevine, for example, but its large user community is based on Alto's, which are unlikely to have encryption hardware (ever?).

There are approaches by which client machines without the hardware can still use cryptography, however. A low-capability client could get the server to decrypt and send data in the clear for it. Although this approach gives up the prospect of achieving *secure communication*, cryptography is still used for *authentication*. Such machines could encrypt the shorter, less frequent messages needed for authentication and requests in software or microcode, and still achieve acceptable performance. [Mitchell]

The question of finding an evolution path to permit secure and unsecure environments to coexist is also an interesting research issue.

Known ways for providing reliable updating of (possibly replicated) information in distributed databases depend on the existence of stable-storage for storing critical information. To date, the properties of such storage have been modelled using discs, often leading to a drastic reduction in performance. It would be attractive to investigate the effects on performance and system organisation of providing high-speed, low-latency *electronic stable storage* for this purpose.

Similarly, proposals for managing the controlled updating of distributed databases requires stamping messages with unique identifiers for controlling the order of actions. The generation of these identifiers is often based on some function of the time-of-day. A number of complicated protocols have been discussed in the literature for overcoming the variations in each system's notion of what the time is. Instead of investing resources in synchronisation algorithms, it would be interesting to study alternative strategies based on the availability of a source of accurate time. A receiver for one of the broadcast time standards is a way of achieving this.

Finally, if we are going to build additional hardware for our systems anyway, we might as well avail ourselves of the opportunity to provide some inexpensive general-purpose experimental input-output ports to enhance the flexibility of our systems.

2.2 Which Machines?

CSL is gradually phasing out its investment in Alto computers in favour of D-machines. This applies both for personal workstations and servers, in the long run. Thus, the Alto's role in our future is clear: investment of further hardware effort in the Alto is counter-strategic.

In contrast, the role the Dolphin is to play in our future is not clear and CSL does not plan to acquire any more for use as workstations. Since there is no Trident disk controller for the Dolphin, the Alpine and Voice File Servers will have to be implemented on Dorado's. Thus, two major clients for auxiliary hardware will be Dorado-based, and the projected need for auxiliary hardware on Dolphins is limited.

Considering this, only the decision to equip the Dorado with auxiliary hardware can be justified in CSL at this time. Extension of the Dolphin is not precluded, of course.

It is conceivable that thought may need to be devoted to the issue of designing appropriate extensions for the DANDELION, but only if CSL actually acquires some for research purposes at some later date or if a clear need emerges from another direction (eg. SDD).

3. DES Cryptography

The ideas about DES encryption hardware derive almost exclusively from the Dolphin prototype based on the Fairchild chip set [Gifford].

The design will implement *memory-to-memory encryption*, so one set of hardware can be used both for *secure communication* and *secure data storage*.

Since the first prototype was built, AMD have announced a flexible, fast chip implementing DES, the Am Z8068. These devices are readily available and fairly inexpensive (~$135). The main advantages of this chip over the Fairchild chip set are:

Smaller:	one 40-pin package instead of four
Faster:	13.6 MBaud
More facilities:	multiple key registers, multiple encryption modes (ie. Cipher Block Chaining mode as well as Electronic Code Book mode)
Pipelining:	with independent input and output ports

3.1 Error Detection

Because the encryption of data is designed to be a transformation which is difficult to reverse, special consideration has to be given to the issue of how to handle hardware error conditions. If an error occurs during encryption, either by using an incorrect key or as a result of a hardware failure, recovery is impossible. Erroneous encrypted data cannot be scavenged!

Further, there seems to be a inherent difference between encryption for the purposes of *secure communication* and encryption for the purposes of *secure data storage* [Schroeder].

With encryption for secure communication, the *sink* end of a byte-stream decrypts and interprets the received data. There is a certain chance that errors will be detected quickly as a result of the decrypted garbage violating higher-level protocols. Maybe one could intentionally extend this idea to provide error control.

With encryption for secure long-term storage, however, there is no attempt to decrypt or interpret the semantics of the stored data until *retrieval-time*, by which time the clear text may have been discarded. Thus, the loose error check described above is not available.

This implies that users are going to feel much safer about cryptography for secure data storage if the hardware takes great care to detect errors at *encryption-time*.

The hardware paths out to the DES chip and in from it can be parity-checked in the usual way. However, no error-control is performed on data passing through the chip except for the *encryption keys*, which are actually 56 bits with 8 parity bits. All available DES devices indicate key parity errors.

However, simply *or*-ing the chip's *KeyParityError* output with other parity failure signals in the system's hardware is not appropriate. This is because other parity failures are considered to result from unrecoverable hardware failures and so they halt the Mesa emulator. Thus, it would be quite easy for a user to stop the machine by loading an incorrect key. We favour an alternative approach where the error signal is presented in a separate parity status register which microcode can read when it wants [Gifford].

Providing additional protection against DES chip errors or failures can be done by one of the following approaches:

a. *Decrypt blocks immediately after encryption and compare with the clear text*. This needs two buffers and twice as much time. We consider this approach unsatisfactory.

b. *Use 2 chips in series, one encrypting and one decrypting*. This would ensure that the clear text entering one chip emerges in a form which the second chip can decrypt back to the original. The comparison of the two versions of the clear-text is done in small units (4 words?) in hardware. Performance is reduced since two transformations have to be performed before the data can be trusted although pipelining is possible. The *phase difference* between the two versions of the clear text implies additional buffering hardware.

c. *Use 2 chips in parallel, both encrypting*. Assuming there are not any known or significant design bugs in the IC's, the failure modes we want to guard against are either random intermittant glitches giving rise to wrong bits, or hard chip failures. If the data outputs of the two chips are compared byte-by-byte, differences can be detected. If the comparison is done after parity regeneration, the Dorado parity protection and the protection gained from this scheme will overlap leaving no holes for errors to sneak in. Performance is better (parallelism) and comparison is done in hardware. No additional buffering required.

The current preferred favorite is (c) since the DES chips are fairly inexpensive. If a DES hardware failure is detected a fault wakes up the fault-handler which invokes retries, etc.

3.2 Performance

A comparison of the estimated performance of the proposed Dorado design (based on the AMD chip) with the performance of the existing Dolphin prototype (based on the Fairchild chip set) is presented here.

	Dolphin prototype	Dorado proposed
Devices used	Fairchild 9414	AMD Am Z8068
DES chips per encrypter	4	1
Cost per encrypter	$150	$135
Bandwidth	12.8 MBaud	13.6 MBaud
Encrypt/Decrypt 512 bytes	320 μS	300 μS
No. of 512 Byte blks per second	3125	3333

3.3 DES Key Generation

The security of the a cryptographic system is critically dependent on the unpredictability of the keys. To use a pseudorandom number generator, even seeded with a truly random number, would make keys unacceptably predicatable. We assume that encryption keys are derived from a truly random number generator. For a DES key, only 56 actual bits have to be generated, the additional 8 bits are odd parity bits, distributed 1 bit per byte in the low-order bit.

Note that the study of key distribution and authentication systems as a research topic does not require either the encryption hardware or the keys to be genuinely secure - almost any trivial scheme will do. However, it seems worth the trouble to invest the extra effort to strive for a genuinely secure environment using the best tools available at this time [McDaniel].

4. Random Bit Generation

The auxiliary hardware will include a noise diode and the necessary amplification and digitisation to produce a stream of *fairly random* bits. These bits are presented sequentially to the microcode (and the Mesa software) as a bit in an I/O device status register. An experimental version has been built and is being added to the Dolphin prototype [Gifford].

The distribution will only be fairly random because the output of the comparator will be naturally biassed towards ones or zeroes by the nature of the hardware. Software is responsible for *unbiassing* the distribution of 1's and 0's by considering the bits pairwise and calling the pair *01* a 1, and the pair *10* a 0 (say).

According to Gifford's measurements, random bits come out of the noise diode at about 20Kbaud. The unbiassing operation is non-deterministic so it will take an indeterminate amount of time to actually accumulate a random number. This is not worrying for DES work since keys do need to be *generated* very often, only *loaded*.

However, there may be other applications which need faster random number generation. It is proposed that these numbers will be generated out of a FIFO pool of random bits accumulated by a background Mesa task. It is possible to perform unbiassing and bit-accumulation in hardware, but there is no clear need to do so. Make your case soon please!

5. Electronic Stable Storage

A number of distributed system applications would benefit from the availability of removable electronic stable storage. One example is for bolstering up the weak properties of disk drives into a model of *strong atomicity*, namely, either all writes belonging to a transaction work or none do. Keeping state about remote procedure calls to help manage the *orphan problem* (keeping state about what is going on in a chain of remote procedure calls when one of the machines crashes) is another.

What these applications need most of stable storage is *negligable access latency* [Mitchell]. One model of how it would be used is that only small amounts of information will be written into it at any one time, so the overall *bandwidth* need not be very high [Lampson]. Thus, small records describing transaction state transitions, procedure call states, etc. can be written into stable storage with impunity without significant impact on system performance.

Operations on stable storage should be performed *synchronously* under the control of the Mesa emulator. Having the operation take place asynchronously, possibly resulting in task-switching, is unacceptable for performance reasons [Lampson]. This is simple if the blocks are small.

Usually, applications will only *write* data into the stable storage - reading is only performed for error recovery (extremely infrequent) or to flush the storage out onto disk occasionally. The *size* of the memory can be quite small - it only has to be large enough to reduce the flushing frequency to an acceptable level perhaps once every few seconds [Lampson].

5.1 Addressing

There are two approaches to integrating the stable storage into a system:

a. It is part of the memory space. It can be accessed directly using Mesa pointers.

b. It is an I/O device. Mesa device interface procedures are provided to access it.

The main advantage of making the memory appear in the address space of the system is that Mesa programs could manage *persistent* and *transient* variables and records uniformly. In particular, storage management of the stable storage is simplified.

Another advantage is that modelling the strong atomic property for disk drives is simplified. A disk page can be written into the electronic stable storage and the disk controller can read it directly from the storage to do the transfer [Mitchell]. This conflicts, however, with the conjecture that data will be written into stable storage in small units [Lampson].

Although intuitively appealing, the engineering of the address-space solution is difficult for D-machines. Even ignoring such issues as cache interactions, there is something to be said for isolating the stable storage from the main memory system somewhat for protection [Lampson]. Schemes have been proposed to guard against software failures, however [Needham & Mitchell].

It has been decided to make the memory appear as an I/.O device. This decision simplifies the packaging dramatically, as well as avoiding the problems which arise from memory-cache interactions.

5.2 Candidate Technologies

A number of technologies might be applicable for the memory design, namely:

a. CMOS RAM's

b. Magnetic Core Storage

Note that both CCD's and magnetic bubble stores would require external FIFO buffering to achieve the zero-latency property on writes which we require. The hardware is non-trivial if guaranteed behaviour in the case of machine failure is to be achieved. RAM's can achieve this effect with much less effort, *bearing in mind that the required size of the memory is small*.

NMOS RAM offers one advantage over CMOS RAM: higher density. However, this is achievable only at the expense of going over to a dynamic technology. We only require a modest amount of memory and would prefer to avoid the complications of refresh.

Static CMOS RAM comes in medium density I/C's (eg. Hitachi HM6116LP-4 = 2K by 8 bits), is fast (down to 120 nS), has very low standby power requirement (less than 50 μA per I/C) and are inexpensive (~$10). The construction of a 16K by 16 memory from 2K by 8 CMOS RAMS would require 16 I/C's, would consume less than 800 μA on battery standby, and would cost around $200. Small PCB-mountable trickle-rechargeable batteries have capacities of 100 to 2000 mAh, allowing memory data to be retained for between 100 and 1000 hours. This is more than adequate since our requirement is that the memory should be able to retain data for the longest conceivable holiday period (about 6 days) [Lampson].

Although CMOS seems so attractive, it is by no means clear that core storage is out of the running. There is a certain quaint attraction in finding a use for core memory in our systems. Several companies (eg. Plessey) make small removable core modules designed to be plugged into machines rather like 8-track cassettes. The price is unknown and an enquiry is being made. Core memory retains data indefinitely without battery backup.

5.3 Write Protection

It is desirable to incorporate hardware into the memory design to ensure that runaway software does not write into the stable storage. In order for software to write a location in stable store, it will have to first supply the current contents of the word in question. This is relatively easy to mechanise in hardware. Block transfers to stable store only need specify the current contents of the first word of the block.

5.4 Error Control and Redundancy

Operations on stable storage have to be safe. This means two things:

a. Getting data in and out must be safe operations.

b. It must be safe to keep data in the memory.

The path from the computer to the memory can be protected cheaply against random errors etc. by using standard parity techniques. The interface must have the property that write operations taking place whjen the power fails on the computer fail detectably, ie. recovery on power up can tell that the last write transaction on stable storage was not committed. One technique for doing this is to write a checksum out as the last word of a block, in much the same way as is used for disk drives.

To guard against gross physical failures, the memory should be (ie replicated. In the event of one checksummed copy of the information being incorrect, the other copy can be read for recovery. Fortunately, there is no reason why the two memories cannot be written in parallel, so no performance penalty is incurred. If hardware is added to compare the contents of words read back for consistency, there ought to be a way to read and write each memory individually.

The memory needs to be removable so that, in the event of system failure, it can be plugged into another server to effect recovery. It would be quite acceptable to have to throw a switch before starting to change any plugs [Mitchell].

5.5 Packaging

The proposed hardware structure is that the auxiliary hardware card carrying the encryption hardware will plug into the Dorado in the normal way. On this board will be an external bus interface capable of driving several external stable memories. These memories are packaged in small boxes and are linked to the Dorado with flat cable. The memory bus will permit block word-wide transfers over a multiplexed 16-bit wide address/data bus.

Packaging the stable storage in external boxes simplifies the transfer of storage from one machine to another. It also means that not every Dorado need actually have the storage installed.

6. Accurate Time-of-Day Services

There are a number of activities on the internet which could benefit from the availability of a source of accurate time-of-day information.

6.1 Why?

There are two motivations for introducing an accurate source of time into our internet environment:

a. A *service* need.

b. A *research* need.

The service need is easy to justify. The notion of time distributed about the internet drifts quite a bit. Time servers (which mostly happen to reside in Alto gateways) are polled for the time by a client, and the client is responsible for choosing the most accurate time (based on the average, or something).

Time servers derive the time from a free-running crystal clock in the machine. It is not unusual for the differences in the frequencies of these crystals to cause a variation of several minutes over a period of a month or so. Differences of minutes cause peculiar oddities at the user interface to such distibuted applications as maintenance of the distributed Grapevine registration database.

Protocols already exist to reset one server's idea of the time from another (GateControl). It is possible to extend these to cause an accurate time server to ram the accurate time down the throat of all the other free-running time servers on the internet at regular intervals. An outage would simply cause the internet to revert to the current situation.

The research need is not so easy to justify at this stage. In the earlier proposals, it was suggested that, if the hardware was simple, it should be thrown in "for free" to see how it got to be used. Actually, there are a number of research areas which would benefit from being able to generate unique distributed system identifiers. One possible way of generating these is based on knowing a fairly accurate time of day. The alternative seems to be making an investment in complicated distributed synchronisation protocols.

The intent is, however, to provide the facility based on the service needs and then see what research use it is put to.

6.2 How?

A number of systems exist which are designed to provide broadcast time standards, of which the most attractive is the WWV service broadcast by NBS. This information is broadcast in various forms on various frequencies fanging from 60kHz up to satellite transmissions in the UHF band. The 60kHz dialect of WWV is called WWVB and is particularly attractive because it is conveniently BCD encoded. Also, it appears that this is the WWV transmission which is both easiest to receive and the transmissions have uniformly high field-strength all over the US (unlike the short-wave versions). The satellite transmissions do not yield quite such accuracy (propagation distance further) and the siting of a suitable antenna is more critical.

Ignoring RF issues for a minute, it is possible to derive a time-of-day clock from the received digital signal which, after applying some simple corrections, is accurate within 500μS anywhere in the US portion of the internet. Doing better than that may be possible, but may require increased paranoia. The hardware necessary to do this is sufficiently cheap that one might equip every machine with an Auxiliary Hardware board with it.

Of course, it would be nice not to have to string up antennae all over the office even though it may not be necessary to have a full quarter-wave dipole!. Yet achieving good reception requires a reasonable antenna especially when one considers that this is a steel-frame building with very noisy contents. 60kHz is also a rather close harmonic of 60Hz, which might render the systems subject to interference from the mains power distribution [Thacker].

Commercial WWVB receivers are available which come packaged with good antennae, an RS-232 interface and thumbwheel switches to input corrections for propagation delay. These cost about $2500 and it seems adequate to consider equipping the internet with two or three of these to achieve internet-wide time synchronisation within one second. Higher accuracy is available on the Ethernets which the receivers are directly connected to, but gateways and leased lines introduce non-deterministic delays.

If greater accuracy were required (later), it is possible to contemplate a special path through the Ethernet microcode which would reach out to the WWV clock registers and copy the time into an Ethernet packet *as the packet is actually being transmitted*, ie. after deferral and acquisition of the ether.

Similarly, microcode at a client could recognise such a *SetTime* packet and reload the client's time-of-day clock directly. More accurate corrections for Ethernet and hardware propagation time could be derived from also implementing a microcode-based echo server in the time server allowing a client to measure its distance from the server.

One important requirement for distributed systems is that the unique ID's in the systems are generated in a *monotonically increasing* fashion. Assuming that a client's time of day clock is occasionally reset from a time server, it must be ensured that the reset cannot allow a duplicate number to be generated. One possibility is to receive the time packet into a safe place and (in hardware) reset a "microseconds-since-time-packet-received" counter incrementing at 1 MHz. Some protection is afforded by using a 0.9 MHz crystal. You might skip a few ID's, but you'll never have a duplicate [Mitchell].

It is currently proposed to purchase two WWVB receivers and plug them into spare Alto's which will do nothing other than run the time server protocol. This is to avoid having to dive into the code for the current Alto gateways, and is regarded as an interim solution! By connecting one of these receivers to the local Ethernet (Net #3), our time will be of rather better quality than elsewhere in the internet and thus, we may not run up against inaccuracy limitations for our research projects quite so soon.

7. External Interfaces

A number of suggestions have been received to add to the hardware we are building sundry external interfaces. These, together with their comments, include:

a. *SLC (1.5 Mbaud) Ethernet Interface*. Devices should be designed to connect to an Ethernet, not to a computer. Thus, one device design can be exploited by any of our machines. However, an SLC Ethernet is unlikely to be of interest to the audio project, and no other potential customers exist for it at the moment. It would be better to wait for a 10Mbaud Ethernet chip to come along and use that. Future systems should permit several 10 Mbaud Ethernet ports to be installed easily.

b. *IEEE 488 / GPIB Interface*. To allow our machines to control machinery and read instruments, they have to speak GPIB. However, the major candidate for this interface is ICL, and they are already committed to solving this problem with other specia-purpose hardware. They will surely not be using Dorado's for this either. If desired, a GPIB interface could be built up on an external parallel interface, see below.

c. *Audio Interface Components*. The major candidate for research using audio compeonents, the Voice project, has decided to solve this need with purpose-built hardware in Etherphones. No advantage can be perceived for putting audio components on the board over plugging them into an external parallel interface, see below.

d. *RS-232 Ports*. Numerous devices exist which require RS-232 interfaces. No major advantage can be perceived in providing this on the card over via an external parallel interface, see below.

e. *Intel Multibus Interface*. This is a serious contender. Just about any interesting piece of hardware, any microprocessor, or any disk controller exists in a Multibus incarnation. It would really enhance our ability assemble experimental equipment quickly if we could explot the Multibus and, instead of building hardware, simply *buy it and plug it together*.

   However, instead of making the Dorado (via the auxiliary card) look like a Multibus master, the preferred solution is to connect to Multibus systems via an Ethernet using either (a) a Stanford University 3 Mbaud Ethernet Multibus interface card, (b) an existing Intel 10 Mbaud 2-card Multibus interface or, eventually, (c) an Intel LSI 10 Mbaud Multibus interface card. Between these three options one can realise any configuration and this is compatible with the idea of making boxes that plug into the Ethernet, not into machines (see (a) above).

f. *Parallel I/O Bus*. Most of the above requirements can be met with a suitable external parallel I/O bus. It should be simple, reasonable fast, and should offer microtask *wakeup* facilities. The fact that the amount of hardware involved to make such an interface is so small is a very streong argument in its favour.

   Note that it is be possible to use the general parallel I/O bus also as the interface for the stable storage. However, it is not yet clear that this is a good idea. The current model assumes a 16-bit multiplexed address and data bidirectional bus running at about 100 nanoseconds.

8. Conclusion

From the discussion above, our design conclusion is that the board will have the following characteristics:

a. ***DES Cryptography***. Two AMD Am Z8068 encryption devices running in parallel will provide error-checked Electronic Code Book or Cipher Block Chaining encryption at 13.8 Mbaud. Keys will be generated from the random number generator, below.

b. ***Random Number Generator***. Actually a random bit generator. Based on a noise diode. Will generate random bits at about 20Kbaud. These will appear as a bit in a device status register to be read by the microcode.

c. ***Electronic Stable Storage***. Based on Hitachi 6116LP4 Static CMOS RAMs. 16K words of 16 bits will be packages in a small box and connected into an I/O device interface on the auxiliary card with a flat cable. The design of the interface will allow multiple boxes to be connected. Battery backup will be provided for at least 10 days of standby operation.

d. ***External Parallel I/O bus***. A simple I/O bus with multiplexed address and data lines will be provided to add on external interfaces. Audio, RS-232, DtoA/AtoD, microcomputer devices can all be added and addressed over this bus. The bus will provide microtask wakeup facilities to allow "interrupts" to be serviced directly by Mesa processes. It is unresolved whether to use ths bus as the interface for the stable storage or not.

e. ***Broadcast Time Standards Receivers***. More than one commercial receiver will be purchased to receive the WWVB broadcast time standard. These receivers will be interfaced to the internet initially via dedicated Alto-1 servers which do nothing else other than respond to time protocol Pups. One of the receivers will be sited on Ethernet #3 to ensure slightly more accurate time locally than elsewhere in the internet.