ETH

Eidgenössische
Technische Hochschule
Zürich

Institut für Informatik
Fachgruppe
Computer Systeme

Beat Heeb

Design of the Processor–Board for the Ceres–2 Workstation

November 1988

Author's address:

Institut für Informatik
ETH–Zentrum
CH–8092 Zürich / Switzerland

Design of the Processor–Board for the Ceres–2 Workstation

Beat Heeb

Abstract

Ceres is a single user workstation based on the NS32032 microprocessor. The NS32532, a new, more powerful, but fully software compatible member of the NS32000 family made it possible to enhance the performance of the machine without much effort. Because of the modular structure of Ceres, only the processor board had to be changed, while the rest of the hardware and all software (except device handlers) remained unchanged. This paper describes the differences between the new processor board and the old one and presents the result of some performance measurements.

Contents

1 Introduction

The Ceres workstation is a single user computer, designed by N. Wirth and H. Eberle between 1984 and 1986. Its main parts are a NS32032 processor from National Semiconductor and the corresponding Floating-Point-Unit (FPU) and Memory-Management-Unit (MMU). It includes a high-resolution display (1024 * 800 dots), a 40 MByte Winchester disk, a floppy disk drive, keyboard, 3 button mouse and interfaces for V24 and a local network. The main properties of the machine are its simple design and its open architecture. The latter made it possible to add a high-resolution color display and an interface for a laser-printer without any problems. A technical description of the Ceres is given in [1], a more detailed analysis can be found in [2].

The modular structure of the machine makes it also possible to enhance its power without a complete redesign, more precisely, only the processor board has to be changed. This is especially fruitful when more powerful processors exist that retain most of the properties of the existing one. Until now, two such processors are available, the NS32332 and the NS32532. Both have the same programming model and are fully software-compatible with the NS32032 (on object code level). This is essential because it hides the change from the programmer. Started in 1987 a prototype board was built for each of these two processors. After successful completion only the superior one (that with the NS32532) was considered further. This paper documents the differences between this new processor board and the old one. It also includes results of some measurements made to compare the three variants and to quantify the reached progress.

2 Main Changes

2.1 The new Processor

The central part of the processor board is the processor itself, dictating most of the logic requirements. For the description of a new processor board it is therefore necessary to present the important properties of the new processor first.

The following features of the NS32532 are noteworthy:

1) The clock rate is increased from 10 to 25 MHz
2) The processor includes the memory-management-unit (MMU) and the timing-control-unit (TCU)
3) There is a separate 32 bit address bus instead of a multiplexed 24 bit address bus
4) The minimal number of cycles per memory access is decreased from 5 to 2
5) A 512 byte instruction cache and a 1024 byte data cache are included in the CPU chip
6) The size of the data bus can be changed between 8, 16 and 32 bit for each memory reference
7) There is a special mode for reading multiple words from consecutive addresses (burst-mode)
8) The size of the MMU pages is increased from 512 to 4096 bytes (a consequence of the wider address bus)
9) There are some new instructions for cache control

Details on the NS32532 can be found in [3].

Points 2) and 3) have the positive effect that they simplify the processor cluster considerably:

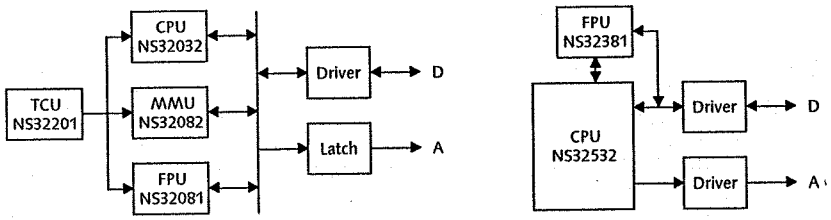


Figure 2.1 Processor clusters for the two processors

Beside the 32 drivers used for the connection from the local and the the global data bus, the old configuration needs another 16 drivers connected between the lower half of the local and the upper half of the global data bus. This was necessary because the MMU NS32082 was designed for the NS32016 and supports only a 16 bit data bus, but must be able to access the page tables in the main memory. With the integration of the MMU in the new CPU this complication disappeared.

The only slave processor not included in the CPU is the floating-point-unit (FPU). Corresponding to the processors, the old unit (a NS32081) is replaced by a new faster one (a NS32381). Apart from speed the only difference between the two versions is the data interface enhanced from 16 to 32 bit. Because the FPU connects directly to the CPU without any discrete logic needed, this change causes no further problems. The FPU is documented in [3].

Random logic is needed, however, for the bus signals *BE0..3* (byte enable), *R/W* (read/write), and *ILO* (interlocked operation), for the *RDY* (ready) input, and for the enable input of the data drivers. All these signals are produced by a single user-programmable logic device (PAL). This solution is not only fast and elegant but also flexible. The latter became important when it was discovered in first tests that the *BE* signals are invalidated too early by the processor to meet the requirements of the RAM board on highest speed. This problem could be solved by latching the BE signals until the end of the *RDY* signal. This change was done without modifying the hardware, just by reprogramming the device.

2.2 The new Bus Arbiter

The central part of the processor board (and also of the whole computer) is the arbiter which manages the bus and determines the actual bus master. The arbiter is conceptually independent of the CPU and all other bus masters, but, because the CPU is the bus master most of the time, the performance of the whole system strongly depends on a fast interaction between the CPU and the arbiter. It was therefore necessary to change the arbiter according to the requirements of the new processor.

The arbiter consists of two loosely coupled parts which could be named as the master control unit and the bus control unit. The former resolves bus request conflicts by a fixed priority scheme and determines the actual bus master, the latter controls the general bus signals and fixes the bus timing.

The master control unit on the old processor board consists of an asynchronous latch and a (combinatorial) priority-encoder. This asynchronous latch doesn't satisfy in two ways: first, it is difficult to generate a proper enable signal, particularly when the clock rate is increased to 25 MHz, and second it is nearly impossible to reason formally

about such a design. A simple solution of this problem is to replace the latch and the priority-encoder by a single synchronous state-machine. The corresponding state diagram looks as follows:

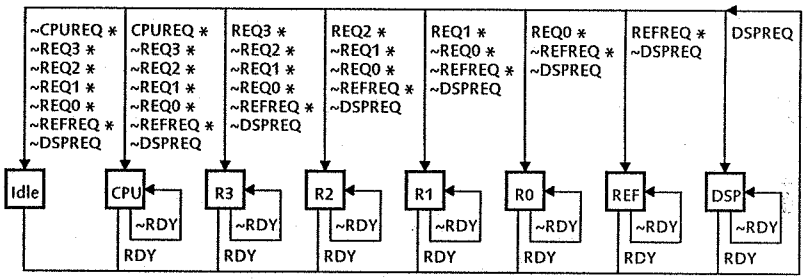


Figure 2.2 Simple master control logic state diagram

Each state means that the corresponding grant output is active, except in the Idle state where no output is active. An additional signal, active in each state except Idle, is needed to start the bus timing. The *RDY* signal is activated by the bus control unit during the last clock cycle of a memory access and is used as a start signal for the next access by the master control unit.

This solution, while being clean and elegant, doesn't solve the problem that motivated the asynchronous solution on the old board: The request signal available from the processor is valid too late to be latched synchronously without a wasted clock cycle. This fact can be seen as a mistake in the design of the NS32032, however on the NS32532 the late availability of the request signal is unavoidable because preceding each memory access the cache must be checked for a hit, in which case no request is to be activated. The NS32532 drives three signals at the beginning of a memory access, namely the *ADS* (address strobe), *BMT* (begin memory transfer), and *CONF* (confirm).

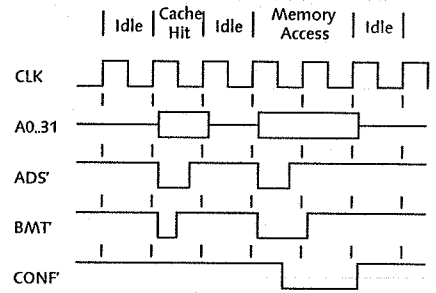


Figure 2.3 NS32532 memory access timing

ADS is active during the first clock cycle of each memory access, *BMT* is activated simultaneously with *ADS* but is deactivated immediately when a cache hit is detected, *CONF* is activated after the second half of the first clock cycle and is guaranteed to become active only when the memory access is really needed. *CONF* has the other important property of remaining active until the end of the access, which makes it an ideal request signal.

In order to use this late request signal without wasting the first clock cycle, the following changes are necessary: First the bus must be dedicated to the processor by default whenever no request signal is active. This was also done on the old board and can easily be implemented by merging the states Idle and CPU. But, although this was sufficient on the old asynchronous approach it is not on the new one; in a synchronous system, a start signal depending on the CPU request valid in the first cycle cannot be active before the second cycle. The only solution here is to make that the normal case. This is possible because the only bus signal activated during the second clock cycle is *DBE* (data buffer enable) which can be merged with the start signal. In other words, the master control logic provides the *DBE* signal in the second cycle instead of a start signal in the first one. But then the state machine must be changed such that the first cycle can be distinguished from the others. This requires doubling each state and gives the following final state diagram:

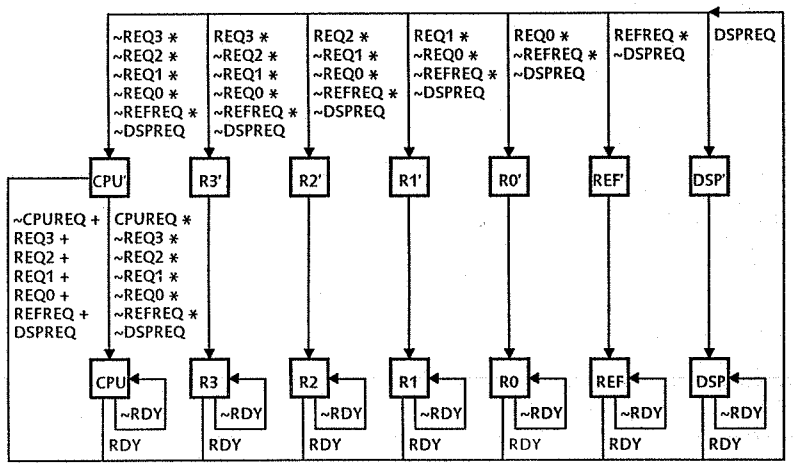


Figure 2.4 Final master control logic state diagram

The upper row of states corresponds to the first clock cycle, during the other states the *DBE* is active. The grant outputs are active in either of the two corresponding states, implying that exactly one grant is active in each cycle. The whole state machine is implemented in one PAL device (16R8).

The other state machine needed is the bus control unit. To hold compatibility with existing memory boards, while using the higher clock speed of the new processor, a memory access covers at least 6 cycles. Because the processor can fetch or store an operand in two cycles, at least 4 wait states must be inserted. In the case of an I/O device 12 additional wait states are required to achieve the desired delays. Since the first cycle is handled by the master control unit, the machine consists of 5 main and 12 wait states. Compared with the old board the total number of states is nearly the same; the reason lies in the fact that the clock speed and the number of states had to be doubled on the old board to reach the granularity required by the bus signals.

Beside the 12 wait states inserted when *IOEN* (I/O enable) is active, one or two wait states can be inserted by activating the signals *WAIT1* or *WAIT2*. In contrast to the old board, a third wait state is inserted when *WAIT1* and *WAIT2* are both active. This is necessary because one wait state on the new board is much less than one on the old. As before, an unlimited number of extra wait states take place as long as *CWAIT* (continuous wait) is active.

Figure 2.5 shows the resulting state diagram:

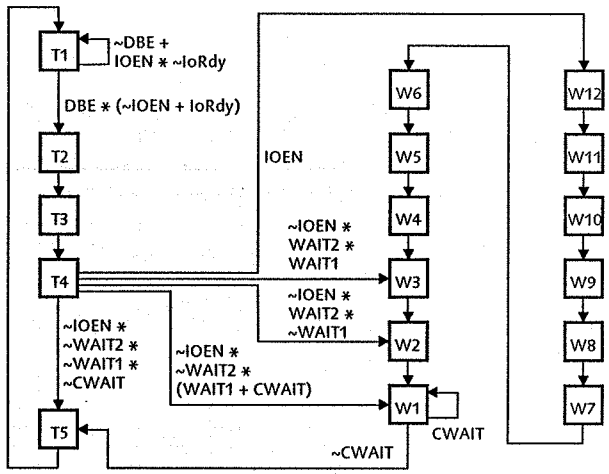


Figure 2.5 Bus control logic state diagram

The *CLR.REQ* signal, introduced as a reset signal for the request flip-flops of the various masters, is no longer needed, because, as a consequence of the synchronous master control unit, these flip-flops can be reset directly by the corresponding grant signal. For being compatible with existing boards, the signal is held active on the new processor board.

The signals *IoAcc* and *IoRdy* are added for communication with the I/O delay counter described latter.

Thanks to the smaller number of output signals needed and with some effort in finding an efficient coding for the states, it was possible to realize this state machine with another single PAL device (16R8). Because both parts of the arbiter are synchronous machines operating with the same clock, they can be seen as one single state machine too. However a description of this machine as a whole leads to an explosion of states and has no advantage over the presented two-part specification.

The following timing diagram contains the specification of all relevant bus signals:
(timing intervals in ns)

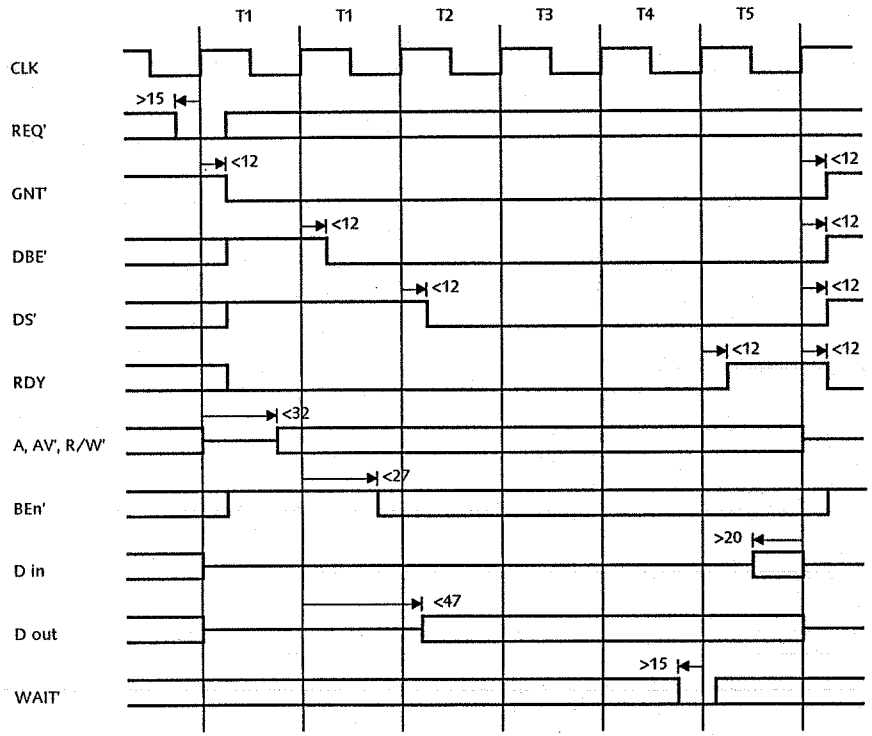


Figure 2.6 Bus signal timing

It is noteworthy that it was possible to reduce the time for a whole memory access from 500ns to 240ns using the same memory board, because only two of the five bus cycles are used for memory access on the old processor. The rest is wasted for transferring virtual and physical addresses over the data bus and for releasing the bus in the last cycle.

Finally we can conclude that the new arbiter is not only better adapted to the new CPU, but that also its behaviour can be described more precisely and its implementation is much simpler.

3 Miscellaneous Changes

3.1 The Mouse Interface

The function of the mouse interface is to make the physical position of the mouse available to the programmer. To do that, the mouse provides two pairs of phase coded signals, one for the horizontal and one for the vertical movement. The actual coordinates can be determined from these signals by a direction discriminator and an up/down counter for each direction. The values of these counters can be read by the processor through an I/O port.

On the old board the mouse interface was built with a PAL for the direction discriminators and standard chips for the counters. With a total of more than seven chips, this interface consumes a considerable part of the total space and power. Thanks to the availability of new, advanced programmable logic devices (EPLDs) the number of components on the new board could be reduced drastically. Because the interface consists of two identical, independent parts, two PLDs can be used with the same contents. Figure 3.1 shows the logic needed:

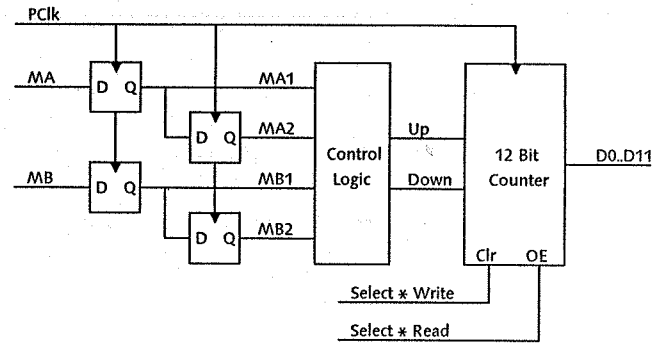


Figure 3.1 Mouse logic for one dimension

The mouse signals *MA* and *MB* are first synchronized and delayed by four D-flip-flops. By comparing the delayed with the undelayed signal, the control logic then detects the rising and falling edges and enables the counter for incrementing or decrementing. A read from the attached address enables the outputs of the counter while a write to the same address resets the counter to zero. The logic equations for the control logic contain all relevant cases:

```
Up   = MA1 * MA2 * ~MB1 * MB2 + MA1 * ~MA2 * MB1 * MB2 +
       ~MA1 * MA2 * ~MB1 * ~MB2 + ~MA1 * ~MA2 * MB1 * ~MB2;
Down = MA1 * MA2 * MB1 * ~MB2 + ~MA1 * MA2 * MB1 * MB2 +
       MA1 * ~MA2 * ~MB1 * ~MB2 + ~MA1 * ~MA2 * ~MB1 * MB2;
```

The chip chosen for the realization of this circuit is the EP600 from Altera Corporation [4]. This chip contains 16 independently configurable I/O macro cells, 4 additional inputs, and a logic array connecting all together. Of the 16 macro cells, 4 are used as D-flip-flops and the remaining 12 are configured as toggle-flip-flops with tri-state outputs for the counter. The control logic together with the logic needed to build the counter fit well into the logic array. The formal description of the whole chip is contained in appendix B.

Beside the two chips for the x and y coordinates, a single driver chip is necessary for the mouse button signals. An additional decoder present on the old board is avoided by presenting the three values (x, y, and buttons) in one double-word instead of assigning a separate I/O address to each of them. Figure 3.2 shows the location of the values in this double-word:

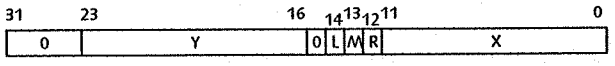


Figure 3.2 Mouse I/O port

Note that the x and y coordinates can still be read independently by using word-wide read operations.

3.2 The Refresh- and I/O-Delay-Counter

To refresh the dynamic RAM chips used in the Ceres periodically, a refresh-counter is used that requests the bus every 16µs. Instead of a data transfer, a refresh cycle is started by the *RFSH* signal which is simply the grant corresponding to the refresh request. On the old board the refresh-counter consists of a counter driven by the processor clock (10MHz) and a flip-flop which is set when the counter value reaches 160 and is cleared by the *RFSH* signal.

The delay-counter is not present on the old board; its introduction was motivated by the following problem: Some of the I/O devices (disk controller, SCC, and RTC) not only need a longer access time, which is guaranteed by the I/O cycle of the arbiter, but also a minimal time of about 1µs between two accesses. On the old board this delay had to be achieved by proper measures in software. With the new, considerably faster processor this is no longer feasible, because, on the one hand the violations are more frequent and less obvious, on the other hand the cache makes it nearly impossible to guarantee a delay by software. The only proper way is to guarantee the delay by hardware. This is done by a circuit consisting of a counter and a flip-flop, a circuit similar to the refresh counter.

To minimize the expense for the two counters, another EP600 is used for both circuits. A block diagram of the two parts is contained in Figure 3.3 and 3.4:

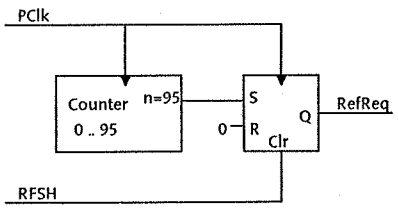


Figure 3.3 Refresh counter

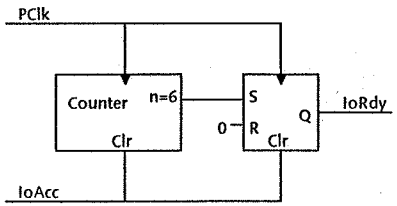


Figure 3.4 I/O delay counter

PClk is a 6MHz clock signal used by the SCC chip (serial communications controller). It was chosen instead of the processor clock because that simplifies the design and, more important, makes the design independent of the actual processor clock frequency. The only drawback is that the resulting signals have no relationship to the processor clock

and must be synchronized by a flip-flop. This is not hard, because these flip-flops fit into the chip already needed for synchronizing the master reset and interrupt request signals which are asynchronous anyway.

RefReq and *RFSH* are connections to the master control logic, which starts the refresh cycles. *IoAcc* is a signal generated by the bus control logic during each I/O cycle. *IoRdy*, which is fed back to the bus control logic, prevents the start of a new I/O cycle when it is inactive.

3.3 The Address Decoder

The main function of the address decoder is to deliver some bus signals like the *IOEN*, which is active whenever the address lies within the I/O domain, as well as the select signals for the individual I/O devices on the processor board. Because the wider address bus of the NS32532 leads to new I/O addresses anyway, the addresses are changed such that each of the major devices lies in a separate MMU page, which allows to protect devices individually against illegal accesses. Figure 3.5 shows the resulting memory map:

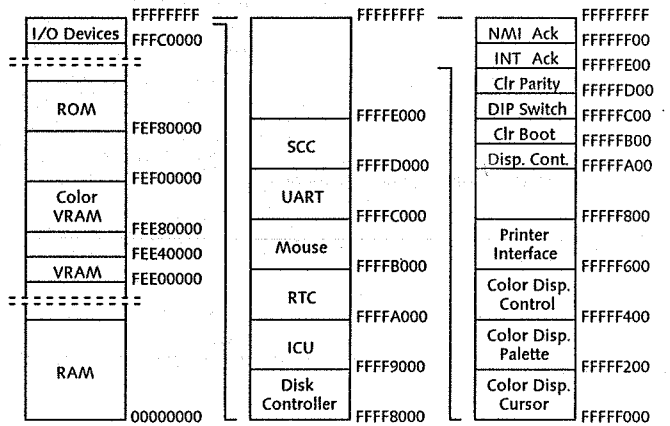


Figure 3.5 Memory Map

Like on the old board, the address decoder consists of a PAL device (16L8) for the coarse decoding and two decoders for the individual select signals. In contrast to the old solution, only a minimum of the address signals is directly connected to the PAL, the remaining signals (*A19..A23, A25..A31*), which must be high in all considered cases, are combined with a single AND-gate. This not only solves the problem of the increased number of address lines, but also leaves a part of the PAL unused, giving room for some random logic needed for the *CLR.PAR* signal on the bus and the *IoDec* signal used by the processor to handle I/O accesses correctly.

Another part related to the address decoder is the boot logic. During the boot phase (after CPU reset) the ROM space must start at address zero because the program counter is cleared by a system reset and the initial program must therefore start at address zero. The boot phase is identified by a special flip-flop, set by the *RESET* signal and cleared by an access of the 'Clear Boot' I/O address under software control.

On the old board the special address mapping is achieved by manipulating the processor address bus in a way such that the RAM space starting at zero is mapped to the ROM space. On the new board a simpler solution is used: Instead of manipulating the address bus, the behaviour of the *RomEn* (ROM enable) signal is changed during the boot phase in a way that every read cycle not belonging to an I/O device accesses the ROM. To prevent other devices like the RAM from beeing accessed simultaneously, the *AV* signal (address valid) is held inactive during these special ROM accesses. The additional logic needed for these signals as well as the boot flip-flop itself could be placed in the address decoder PAL.

3.4 The Boot ROM

The boot ROM holds the initial program which typically loads the final system programs from the disk. The usual 28 pin ROM chips are accessed byte by byte. To support a 32 bit data bus, four such chips are needed. This is very inefficient because even one chip is far too big for a typical boot program. To avoid this problem, the dynamic bus sizing feature of the new processor is used. Through an input signal of the processor, which is made available on the bus as the *BYTE* signal, it is possible to switch to an 8 bit wide data bus for each memory access. By activating this signal for ROM accesses, a single ROM chip suffices. The only drawback, the reduction of execution speed, is insignificant, because the ROM is used during system startup only and has no influence afterwards.

The ROM devices normally used are EPROMs (erasable, programmable read only memory) which can be programmed and erased by the user with a special hardware unit. This way of programming is well suited for the manufacture of small series, but is a time consuming task during the development of the boot program. A better way is to use an EEPROM (electrically erasable, programmable read only memory) which can be programmed without being removed from the machine. Because the used EEPROM (2864) is pin compatible to the corresponding EPROM (2764), the ROM logic could easily be extended to support both types. The only changes are the use of a bidirectional data driver and an additional logic for the write signal. To prevent the ROM from being overwritten accidentally, the write signal is protected by a switch on the backplane of the processor board. The same switch can be read by software as bit 8 of the dip-switch and it is used to select an alternate boot file source.

To meet the timing specifications of the EEPROM during programming, writes to the ROM area are treated as I/O accesses by the address decoder. Beside that, the programmer must ensure that the EEPROM is untouched for at least 10ms after each write to give the device time for the final programming of the memory cells.

4 Measurements

4.1 Performance

The main motivation for the design of a new processor board was an expected gain in performance. For a quantification of the gain, the execution times of some test programs were measured. To get a complete view over the available parts of the NS32000 series, the following Ceres configurations were compared:

- The old processor board with the NS32032 operating at 10MHz
- A prototype board with the NS32332 operating at 15MHz
- The new processor board with the NS32532 operating at 25MHz but without caches
- The new processor board with enabled data- and instruction caches

It is noteworthy that the measurements were made on the same physical machine with only the processor board changed. This ensures that the rest of the hardware, in particular the memory, as well as the programs are exactly identical and results in a fair comparison of the processors and their bus interface.

The benchmark programs used for performance comparison in [2] are not useful here because they execute a small loop with only a few instructions in it. The instructions as well as the variables of such a loop would fit entirely in the caches, resulting in a very fast execution, but the execution times are no longer related to those of real programs. To bypass this problem the following programs are chosen for comparison:

Quicksort
A program which sorts an array of 1024, 16 bytes long records with the quicksort algorithm. Initially the keys are distributed randomly over the array. Note that the total size of the array (16 kbyte) is far more than the size of the data cache.

Dhrystone
The Dhrystone benchmark defined in [5].

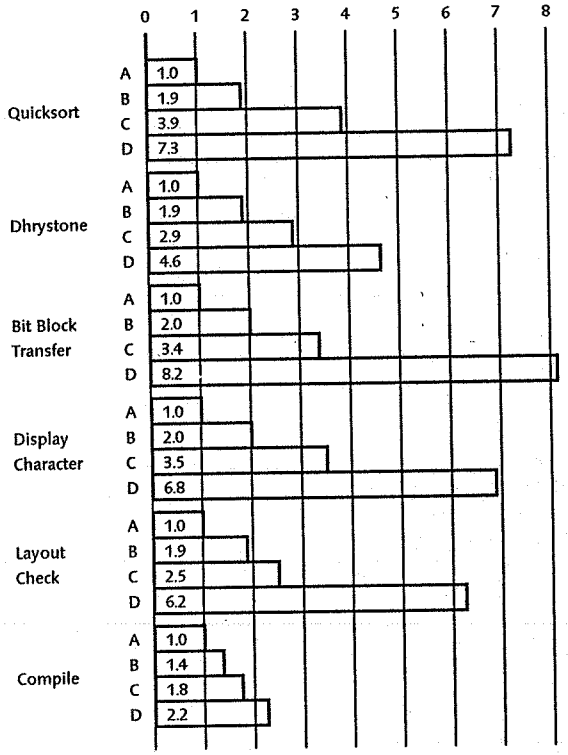
Bit Block Transfer
Bit Block Transfer moves a square of 512 by 512 pixels on the screen. The distance is chosen in a way that each word must be shifted during the transfer.

Display Character
A program which displays a 12 pixel high character on the screen. The average over all characters of the alphabet is chosen and the font patterns are already contained in memory.

Layout Check
This program is part of a layout editor for printed circuit boards. It verifies the final layout against a formally defined netlist. The program contains complex algorithms as well as large data structures holding both the layout and the netlist in the main memory.

Compile
The Modula 2 compiler. Because this program frequently accesses the winchester disk, its performance is more influenced by the disk controller than by the processor. It is included in this list for completeness.

A comparison of the execution times of these programs running on the different processors is shown in Figure 4.1:



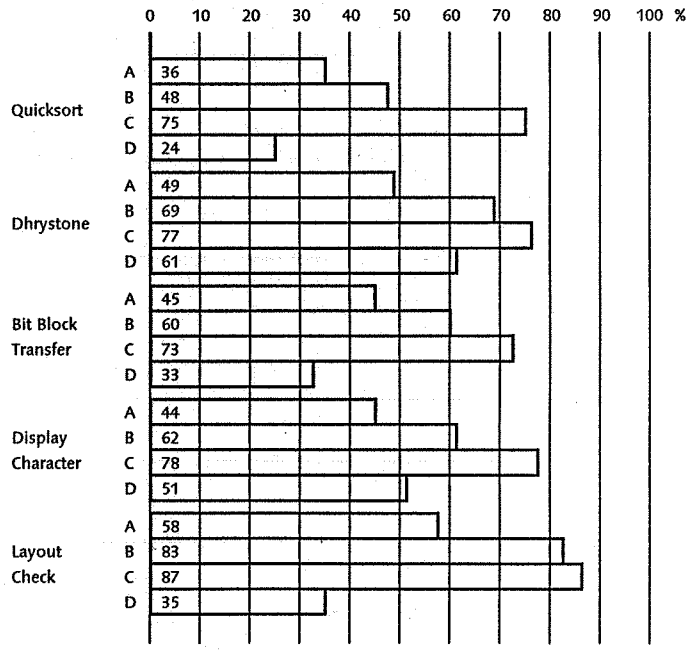
A: NS32032, 10MHz
B: NS32332, 15MHz
C: NS32532, 25MHz, no caches
D: NS32532, 25MHz, caches enabled

Figure 4.1 Relative performance

As expected the performance increases for each new processor. The gain is higher than the difference in clock speed, showing that the efficiency of the micro-code of the new processors is also improved. The most important increase, however, comes from the caches, which almost double performance.

4.2 Bus Utilization

Using a faster processor with equally rapid memory forces an increase of bus utilization. When this comes close to 100%, the memory interface becomes a serious bottleneck for the system. To see whether that happens, the bus utilization was measured for the same system configurations and programs as above. The measuring was done by counting the memory references with a simple frequency counter. The expected utilization can be calculated as the product of the access frequency and the memory cycle time. The results are shown in Figure 4.2:



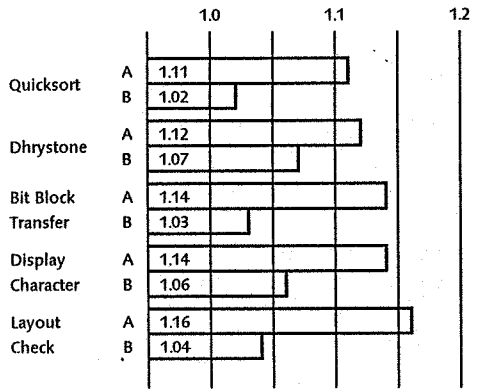
A: NS32032, 10MHz
B: NS32332, 15MHz
C: NS32532, 25MHz, no caches
D: NS32532, 25MHz, caches enabled

Figure 4.2 Bus utilization

Bus utilization of the NS32532 approaches 90%, which is considerably high. Enabling the caches, however, results in values mostly below that of the NS32032, which is surprisingly low. The conclusion is that, thanks to the caches, even an additional increase in the speed of the processor would be possible without a serious problem with the memory bandwidth.

4.3 Influence of Memory Speed

To analyse the interaction of the memory and the processor further, the influence of the memory speed on the system performance was examined directly. This was done by comparing the NS32532 board with a special version providing a memory cycle time of 5 instead of 6 clock cycles. To use this special board with the normal memory, the CPU clock frequency was lowerd from 25 to 20MHz. The measured performance was then corrected with a constant factor. Figure 4.3 shows the results:



A: NS32532, 25MHz, no caches
B: NS32532, 25MHz, caches enabled

Figure 4.3 Performance win due to the elimination of a wait state

Without caches the reduction of the memory cycle from 6 to 5 clock cycles results in a performance improvement of 11 - 16%, which is near the theoretical limit of 1/6. With enabled caches the win decreases to a few percent. Each further eliminated wait state brings a still worse result because the internal execution times of the processor become more dominant. In contrast, the cost of faster memory increases more than linearly with the number of saved wait states.

The results of this and the above measurements show that the comparatively slow memory board built for the NS32032 is still a good solution for the NS32532, although the four wait states look poor at first sight. Any effort for speeding up memory would not be jusified by the expected gain.

5 Conclusions

A new processor board was built for the Ceres workstation. The major changes are the replacement of the NS32032 CPU by a more powerful member of the software compatible NS32000 family and the adaptation of the bus arbiter which is the heart of the system. Some minor changes were done to make the design simpler and cleaner. Some parts were replaced by modern, flexible programmable logic devices (EPLDs).

The result is a board with a chip count decreased from 64 to 45, a current consumption reduced from 2.7 to 1.5 Ampere, but with a performance increased by a factor of 5 to 8. The board is compatible with all existing Ceres boards on the hardware side and needs only a few changes in system programs (I/O addresses) on the software side.

Measurements were made to examine the properties of the system. Beside the mentioned performance win, the results suggest that the existing memory board, although designed for a much slower processor, is still a good choice, because the effect of the higher frequency on the memory bandwidth is neutralized by the caches included in the processor.

A prototype on a wire wrap board was operating 4 month after the start of the project. The final version, built on a printed circuit board, took another 3 month. A series of 30 computers containing the new board is currently being completed.

The only substantial problem encountered during the design was an electrical interference between some wires on the prototype, which disappeared entirely on the final board. It seems that a processor operating at 25MHz reaches the limits beyond which a wire wrap prototype is no longer feasible.

Acknowledgements

I wish to thank N. Wirth for the suggestion and guidance of the project. I also wish to thank H. Eberle for the development and documentation of the Ceres, without which my work had never been possible. Finally I wish to thank I. Noack and A. Weiss for their untiring help in solving practical problems.

References

[1] H. Eberle: Hardware Description of the Workstation Ceres,
Institut für Informatik, ETH Zürich, report no. 70, January 1987

[2] H. Eberle: Development and Analysis of a Workstation Computer,
Ph. D. thesis no. 8431, ETH Zürich, 1987

[3] Series 32000 Microprocessors Databook,
National Semiconductor, 1988

[4] ALTERA Data Book,
Altera Corporation, January 1988

[5] R. P. Weicker: Dhrystone: A Synthetic Systems Programming Benchmark,
Comm. of the ACM, Vol. 27, No. 10, October 1984, pp. 1013–1030.

Appendix A: Summary of User–Relevant Changes

Software Differences

< old values >

Addresses

- 32 bit addresses < 24 bit >
- Disk controller: $FFFF8000..$FFFF801C < $FFFC00..$FFFC1C >
- ICU: $FFFF9000/$FFFF9004 < $FFFE08..$FFFE0C >
- RTC: $FFFFA000 < $FFFC80 >
- Mouse: $FFFFB000 < $FFFD00..$FFFD08 >
- UART: $FFFFC000..$FFFFC03C < $FFFD40..$FFFD7C >
- SCC: $FFFFD000..$FFFFD00C < $FFFD80..$FFFD8C >
- INT acknowledge: $FFFFFE00 < $FFFFE00 >
- DIP–switch: $FFFFFC00 < $FFFDC0 >
- Clear boot: read from $FFFFFB00 < write to $FFFDC0 >
- Clear parity: read from $FFFFFD00 < access of $FFFC40 >
- ROM: $FEF80000..$FEF81FFF < $F80000..$F87FFF >
- Display RAM: $FEE00000..$FEE3FFFF < $E00000..$E3FFFF >
- Color display RAM: $FEE80000..$FEEFFFFF < $E80000..$EFFFFF >
- other I/O: $FFxxxxxx < xxxxxx >

CPU

- new configuration register
- new debug registers
- V bit in the PSR for automatic overflow trap
- new instructions:

  CINV cache invalidate
  LPR CFG, x load configuration register
  SPR CFG, x store configuration register

MMU

- 4 kB pages
- new registers

FPU

- 8 LONGREAL–registers < 8 REAL or 4 LONGREAL > (F1 is still part of L0 !)
- new bits in the status register
- new instructions:

  SCALB x, y y := y * 2↑TRUNC(x)
  LOGB x, y y := TRUNC($\log_2$(x))
  DOT x, y L0 := L0 + x * y
  POLY x, y L0 := y + x * L0

SCC

Receiver driver enabled via DTR output (independant of transmitter driver)

Mouse

- X counter: bits 0..11
- Y counter: bits 16..23 (bits 0..11 of high word)
- buttons: ML: bit 14, MM: bit 13, MR: bit 12 (0 = pressed, 1 = released)
- A write to the mouse port resets both counters to zero

DIP–Switch

Bit 8: Boot switch (1: normal / 0: special)

Peripheral Cycles

A delay of 1us between two peripheral cycles is guaranteed by hardware

Hardware Differences

The following bus signals are changed:

BUS.ERR' pin Aa13

New signal. Raises a bus–error exception on the processor.

CLR.REQ' pin Aa15

No longer needed. Held active on the processos board for compatibility.

WAIT1' pin Ba9 and
WAIT2' pin Ba10

A third wait state is inserted when both are active.

IO.EN' pin Ba11

Active when the address lies in the new I/O domain: $FFFC0000 .. $FFFFFFFF.

BYTE' pin Ba16

New signal. Tells the processor to use a byte–wide bus for the actual memory access.

CLK pin Ba25

Processor clock increased from 10 to 25MHz.

FCLK pin Ba27

No longer supported.

A24 pin Bc25 to
A31 pin Bc32

New signals. Most significant byte of the 32 bit address bus.

Appendix B: PAL– and EPLD Listings

Master Control PAL

```
PAL prio: 16R8;

(* NS32532 Priority Encoder    B. Heeb, 8.3.88 *)

PIN  2: ~DSPREQ;    19: ~DSPGNT;
     3: ~RefReq;    18: ~RFSH;
     4: ~REQ0;      17: ~GNT0;
     5: ~REQ1;      16: ~GNT1;
     6: ~REQ2;      15: ~GNT2;
     7: ~REQ3;      14: ~GNT3;
     8: ~CpuReq;    13: ~CpuGnt;
     9: RDY;        12: ~DBE;

EQUATIONS

DSPGNT := RDY * DSPREQ
        + ~DBE * CpuGnt * DSPREQ
        + DSPGNT * ~RDY;

RFSH   := RDY * RefReq * ~DSPREQ
        + ~DBE * CpuGnt * RefReq * ~DSPREQ
        + RFSH * ~RDY * ~DSPGNT;

GNT0   := RDY * REQ0 * ~RefReq * ~DSPREQ
        + ~DBE * CpuGnt * REQ0 * ~RefReq * ~DSPREQ
        + GNT0 * ~RDY * ~RFSH * ~DSPGNT;

GNT1   := RDY * REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + ~DBE * CpuGnt * REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + GNT1 * ~RDY * ~GNT0 * ~RFSH * ~DSPGNT;

GNT2   := RDY * REQ2 * ~REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + ~DBE * CpuGnt * REQ2 * ~REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + GNT2 * ~RDY * ~GNT1 * ~GNT0 * ~RFSH * ~DSPGNT;

GNT3   := RDY * REQ3 * ~REQ2 * ~REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + ~DBE * CpuGnt * REQ3 * ~REQ2 * ~REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + GNT3 * ~RDY * ~GNT2 * ~GNT1 * ~GNT0 * ~RFSH * ~DSPGNT;

CpuGnt := RDY * ~REQ3 * ~REQ2 * ~REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + ~DBE * CpuGnt * ~REQ3 * ~REQ2 * ~REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + ~RDY * DBE * ~GNT3 * ~GNT2 * ~GNT1 * ~GNT0 * ~RFSH * ~DSPGNT
        + ~RDY * ~CpuGnt * ~GNT3 * ~GNT2 * ~GNT1 * ~GNT0 * ~RFSH * ~DSPGNT;

DBE    := ~RDY * DSPGNT
        + ~RDY * RFSH
        + ~RDY * GNT0
        + ~RDY * GNT1
        + ~RDY * GNT2
        + ~RDY * GNT3
        + ~RDY * CpuReq * ~REQ3 * ~REQ2 * ~REQ1 * ~REQ0 * ~RefReq * ~DSPREQ
        + ~RDY * DBE;

END prio.
```


Bus Control PAL

```
PAL time25: 16R8;
(* NS32532 25MHz Bus Timing State Machine   B. Heeb, 26.5.88 *)

PIN  2: ~DBE;        19: ~IoAcc;
     3: ~WRITE;      18: RDY;
     4: ~IOEN;       17: ~DS;
     5: ~WAIT2;      16: ~IORD;
     6: ~WAIT1;      15: ~IOWR;
     7: ~CWAIT;      14: ~d0;
     8: ~IoRdy;      13: ~d1;
                     12: ~d2;

(*           RDY d0 d1 d2 IoAcc
   T1:         0  0  0  0  0
   T2:         0  1  1  1  0
   T3:         0  0  1  1  0
   T4:         0  1  0  1  0
   W12:        0  0  0  1  0
   W11:        0  1  1  1  1
   W10:        0  0  1  1  1
   W9:         0  1  0  1  1
   W8:         0  0  0  1  1
   W7:         0  1  1  0  1
   W6:         0  0  1  0  1
   W5:         0  1  0  0  1
   W4:         0  0  0  0  1
   W3:         0  1  1  0  0
   W2:         0  0  1  0  0
   W1:         0  1  0  0  0
   T5:         1  0  0  0  0 *)

EQUATIONS

~RDY := ~d0
      + d1
      + IoAcc
      + CWAIT
      + d2 * ~IoAcc * IOEN
      + d2 * ~IoAcc * WAIT1
      + d2 * ~IoAcc * WAIT2;

d0   := ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * ~IOEN
      + ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * IoRdy
      + ~d0 * d1 * ~RDY
      + ~d0 * d2 * ~RDY
      + ~d0 * IoAcc * ~RDY
      + d0 * ~d1 * d2 * ~IoAcc * WAIT1 * ~IOEN * ~RDY
      + d0 * ~d1 * ~IoAcc * CWAIT * ~IOEN * ~RDY;

d1   := ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * ~IOEN
      + ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * IoRdy
      + ~d0 * ~d1 * d2 * ~RDY
      + ~d0 * ~d1 * IoAcc * ~RDY
      + d0 * d1 * ~RDY
      + d0 * ~d1 * d2 * ~IoAcc * WAIT2 * ~IOEN * ~RDY;

d2   := ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * ~IOEN
      + ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * IoRdy
      + d1 * d2 * ~RDY
      + ~d0 * ~d1 * d2 * ~IoAcc * ~RDY
      + d0 * ~d1 * d2 * ~IoAcc * IOEN * ~RDY
      + d0 * ~d1 * d2 * IoAcc * ~RDY;

IoAcc := ~d0 * ~d1 * d2 * ~IoAcc * ~RDY
      + d0 * IoAcc * ~RDY
      + d1 * IoAcc * ~RDY
      + d2 * IoAcc * ~RDY;

DS   := ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * ~IOEN
      + ~d0 * ~d1 * ~d2 * ~IoAcc * ~RDY * DBE * IoRdy
      + DS * ~RDY * ~IOWR
      + DS * ~RDY * ~d0
      + DS * ~RDY * d1
      + DS * ~RDY * d2
      + DS * ~RDY * IoAcc
      + DS * ~RDY * CWAIT;

IORD := d0 * d1 * d2 * IoAcc * ~WRITE
      + IORD * ~RDY;

IOWR := d0 * d1 * d2 * IoAcc * WRITE
      + IOWR * ~RDY * ~d0
      + IOWR * ~RDY * d1
      + IOWR * ~RDY * d2
      + IOWR * ~RDY * IoAcc
      + IOWR * ~RDY * CWAIT;

END time25.
```

Processor Control PAL

```
PAL proc: 16L8;

(* NS32532 Processor Control Logic      B. Heeb 9.6.88 *)

PIN     1: ~CpuGnt;
        2: ~be3;        19: ~ILO;
        3: ~be2;        18: ~BE3;
        4: ~be1;        17: ~BE2;
        5: ~be0;        16: ~BE1;
        6: ~ddin;       15: ~BE0;
        7: ~ilo;        14: ~WRITE;
        8: RDY;         13: ~CpuRdy;
        9: ~DBE;        12: ~BuffEn;
                        11: Slave;

EQUATIONS

IF CpuGnt THEN BE0 := be0 * DBE * ~RDY
                    + ddin * DBE * ~RDY
                    + BE0 * RDY;

IF CpuGnt THEN BE1 := be1 * DBE * ~RDY
                    + ddin * DBE * ~RDY
                    + BE1 * RDY;

IF CpuGnt THEN BE2 := be2 * DBE * ~RDY
                    + ddin * DBE * ~RDY
                    + BE2 * RDY;

IF CpuGnt THEN BE3 := be3 * DBE * ~RDY
                    + ddin  * DBE * ~RDY
                    + BE3 * RDY;

IF CpuGnt THEN WRITE := ~ddin * ~DBE
                      + WRITE * DBE;

IF CpuGnt THEN ILO := ilo;

IF TRUE THEN CpuRdy := RDY * CpuGnt + Slave;

IF TRUE THEN BuffEn := CpuGnt * DBE;

END proc.
```

Address Control PAL

```
PAL Addr: 16L8;

(* NS32532 Address Control Logic      B. Heeb 10.6.88 *)

PIN     1: ~CpuGnt;
        2: A16;         19: ~IoSel;
        3: A17;         18: ~boot;
        4: A18;         17: ~AV;
        5: ~HiAd;       16: ~WRITE;
        6: A24;         15: ~IOEN;
        7: ~RESET;      14: ~CLRPAR;
        8: ~ClrPar;     13: ~IoDec;
        9: ~ClrBoot;    12: ~RomEn;
                        11: ~IoInh;

EQUATIONS

IF TRUE THEN IoSel := A16 * A17 * A18 * A24 * HiAd * AV * ~IoInh;

IF TRUE THEN boot := RESET
                   + boot * ~ClrBoot;     (* RS Latch *)

IF CpuGnt THEN AV := ~boot
                   + WRITE
                   + A24;

IF TRUE THEN IOEN := A18 * A24 * HiAd * AV * ~IoInh
                   + ~A16 * ~A17 * ~A18 * ~A24 * HiAd * AV * WRITE * ~IoInh;

IF TRUE THEN CLRPAR := ClrPar + RESET;

IF TRUE THEN IoDec := A18 * A24 * HiAd * AV
                    + ~A16 * ~A17 * ~A18 * ~A24 * HiAd * AV * WRITE;

IF TRUE THEN RomEn := ~A16 * ~A17 * ~A18 * ~A24 * HiAd * AV * ~RESET
                    + CpuGnt * boot * ~A24 * ~WRITE;

END Addr.
```

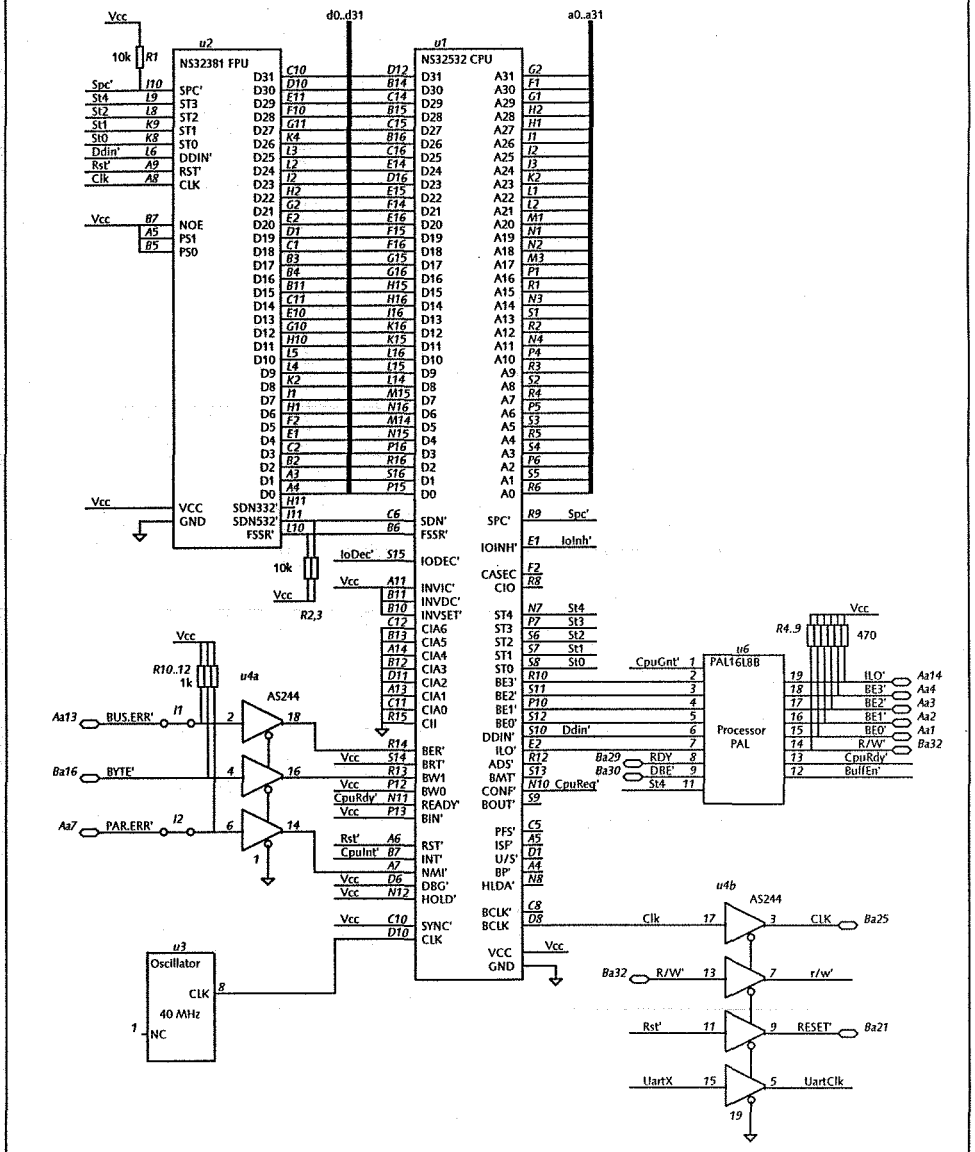

Mouse EPLD

```
B Heeb
ETH Zuerich
8/6/88
1.0
A
EP600
Ceres2 Mouse Counter

OPTIONS: TURBO = OFF

PART: EP600

INPUTS: Clk1@1, Clk2@13, MA@2, MB@11, Write'@14, Sel'@23

OUTPUTS: D0@3, D1@4, D2@5, D3@6, D4@7, D5@8, D6@9,
         D7@10, D8@15, D9@16, D10@17, D11@18

NETWORK:
Clk1 = INP(Clk1)
Clk2 = INP(Clk2)
MA = INP(MA)
MB = INP(MB)
nWrite = INP(Write') Write = NOT(nWrite)
nSel  = INP(Sel')  Sel  = NOT(nSel)
D0,D0 = TOTF(D0t, Clk1, Clr, , OutEn)
D1,D1 = TOTF(D1t, Clk1, Clr, , OutEn)
D2,D2 = TOTF(D2t, Clk1, Clr, , OutEn)
D3,D3 = TOTF(D3t, Clk1, Clr, , OutEn)
D4,D4 = TOTF(D4t, Clk1, Clr, , OutEn)
D5,D5 = TOTF(D5t, Clk1, Clr, , OutEn)
D6,D6 = TOTF(D6t, Clk1, Clr, , OutEn)
D7,D7 = TOTF(D7t, Clk1, Clr, , OutEn)
D8,D8 = TOTF(D8t, Clk2, Clr, , OutEn)
D9,D9 = TOTF(D9t, Clk2, Clr, , OutEn)
D10,D10 = TOTF(D10t, Clk2, Clr, , OutEn)
D11,D11 = TOTF(D11t, Clk2, Clr, , OutEn)
MA1 = NORF(MA1d, Clk2, , )
MB1 = NORF(MB1d, Clk2, , )
MA2 = NORF(MA2d, Clk2, , )
MB2 = NORF(MB2d, Clk2, , )

EQUATIONS:
MA1d  = MA;
MB1d  = MB;
MA2d  = MA1;
MB2d  = MB1;
Up    = MA1 & MA2 & /MB1 & MB2 + MA1 & /MA2 & MB1 & MB2 +
        /MA1 & MA2 & /MB1 & /MB2 + /MA1 & /MA2 & MB1 & /MB2;
Down  = MA1 & MA2 & MB1 & /MB2 + /MA1 & MA2 & MB1 & MB2 +
        MA1 & /MA2 & /MB1 & /MB2 + /MA1 & /MA2 & /MB1 & MB2;
OutEn = /Write & Sel;
Clr   = Write & Sel;
D0t   = Up + Down;
D1t   = Up & D0 + Down & /D0;
D2t   = Up & D0 & D1 + Down & /D0 & /D1;
D3t   = Up & D0 & D1 & D2 + Down & /D0 & /D1 & /D2;
D4t   = Up & D0 & D1 & D2 & D3 + Down & /D0 & /D1 & /D2 & /D3;
D5t   = Up & D0 & D1 & D2 & D3 & D4 + Down & /D0 & /D1 & /D2 & /D3 & /D4;
D6t   = Up & D0 & D1 & D2 & D3 & D4 & D5 +
        Down & /D0 & /D1 & /D2 & /D3 & /D4 & /D5;
D7t   = Up & D0 & D1 & D2 & D3 & D4 & D5 & D6 +
        Down & /D0 & /D1 & /D2 & /D3 & /D4 & /D5 & /D6;
D8t   = Up & D0 & D1 & D2 & D3 & D4 & D5 & D6 & D7 +
        Down & /D0 & /D1 & /D2 & /D3 & /D4 & /D5 & /D6 & /D7;
D9t   = Up & D0 & D1 & D2 & D3 & D4 & D5 & D6 & D7 & D8 +
        Down & /D0 & /D1 & /D2 & /D3 & /D4 & /D5 & /D6 & /D7 & /D8;
D10t  = Up & D0 & D1 & D2 & D3 & D4 & D5 & D6 & D7 & D8 & D9 +
        Down & /D0 & /D1 & /D2 & /D3 & /D4 & /D5 & /D6 & /D7 & /D8 & /D9;
D11t  = Up & D0 & D1 & D2 & D3 & D4 & D5 & D6 & D7 & D8 & D9 & D10 +
        Down & /D0 & /D1 & /D2 & /D3 & /D4 & /D5 & /D6 & /D7 & /D8 & /D9 & /D10;

END$
```

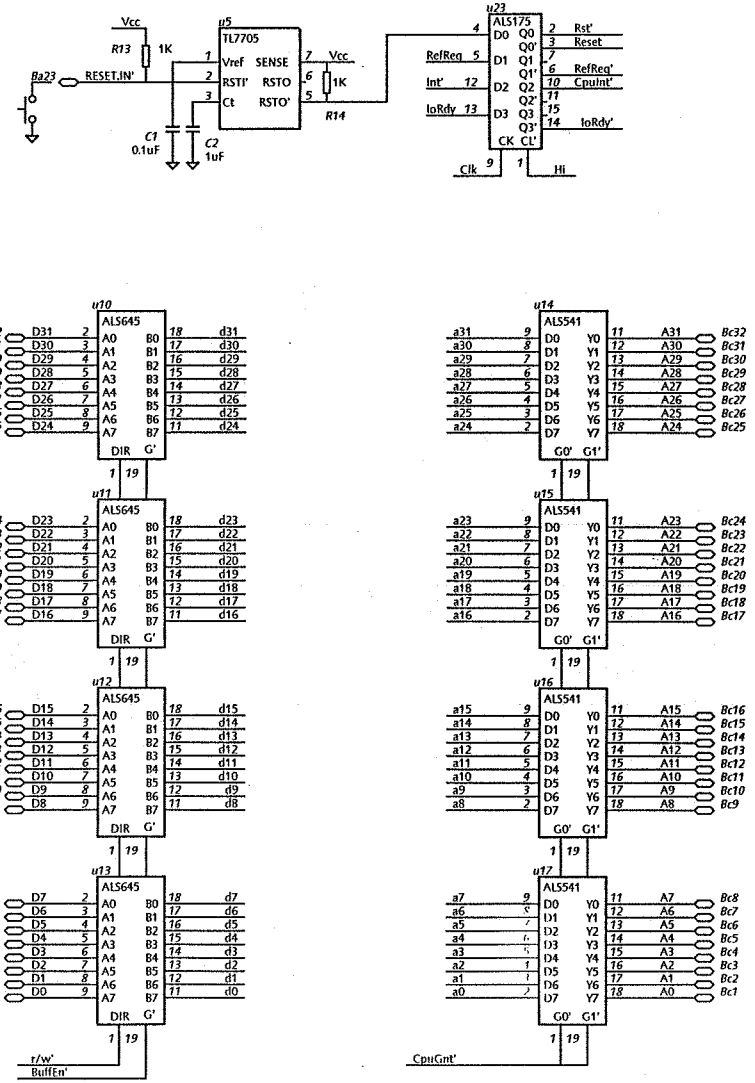
Timer EPLD

```
B Heeb
ETH Zuerich
8/6/88
1.0
A
EP600
Ceres2 IO & Refresh Timer

OPTIONS: TURBO = OFF

PART: EP600

INPUTS: RFSH'@2, IoAcc'@11, Clk

OUTPUTS: RefReq@3, IoRdy@4

NETWORK:
Clk = INP(Clk)
nRFSH  = INP(RFSH')  RFSH  = NOT(nRFSH)
nIoAcc = INP(IoAcc') IoAcc = NOT(nIoAcc)
RefReq = SONF(RefReqs, Clk, RefReqr, ClrRef, , )
IoRdy  = SONF(IoRdys, Clk, IoRdyr, ClrIo, , )
r0 = NOTF(r0t, Clk, , )
r1 = NOTF(r1t, Clk, , )
r2 = NOTF(r2t, Clk, , )
r3 = NOTF(r3t, Clk, , )
r4 = NOTF(r4t, Clk, , )
r5 = NOTF(r5t, Clk, , )
r6 = NOTF(r6t, Clk, , )
i0 = NOTF(i0t, Clk, ClrIo, )
i1 = NOTF(i1t, Clk, ClrIo, )
i2 = NOTF(i2t, Clk, ClrIo, )

EQUATIONS:

ClrRef = RFSH;       % Refresh Timer %
ClrIo = IoAcc;
r0t = VCC;
r1t = r0;
r2t = r0 & r1;
r3t = r0 & r1 & r2;
r4t = r0 & r1 & r2 & r3;
r5t = r0 & r1 & r2 & r3 & r4 & /r6;
r6t = r0 & r1 & r2 & r3 & r4 & (r5 + r6);
RefReqs = r0 & r1 & r2 & r3 & r4 & /r5 & r6;
RefReqr = GND;

i0t = VCC;           % I/O Timer %
i1t = i0;
i2t = i0 & i1;
IoRdys = i1 & i2;
IoRdyr = GND;

END$
```

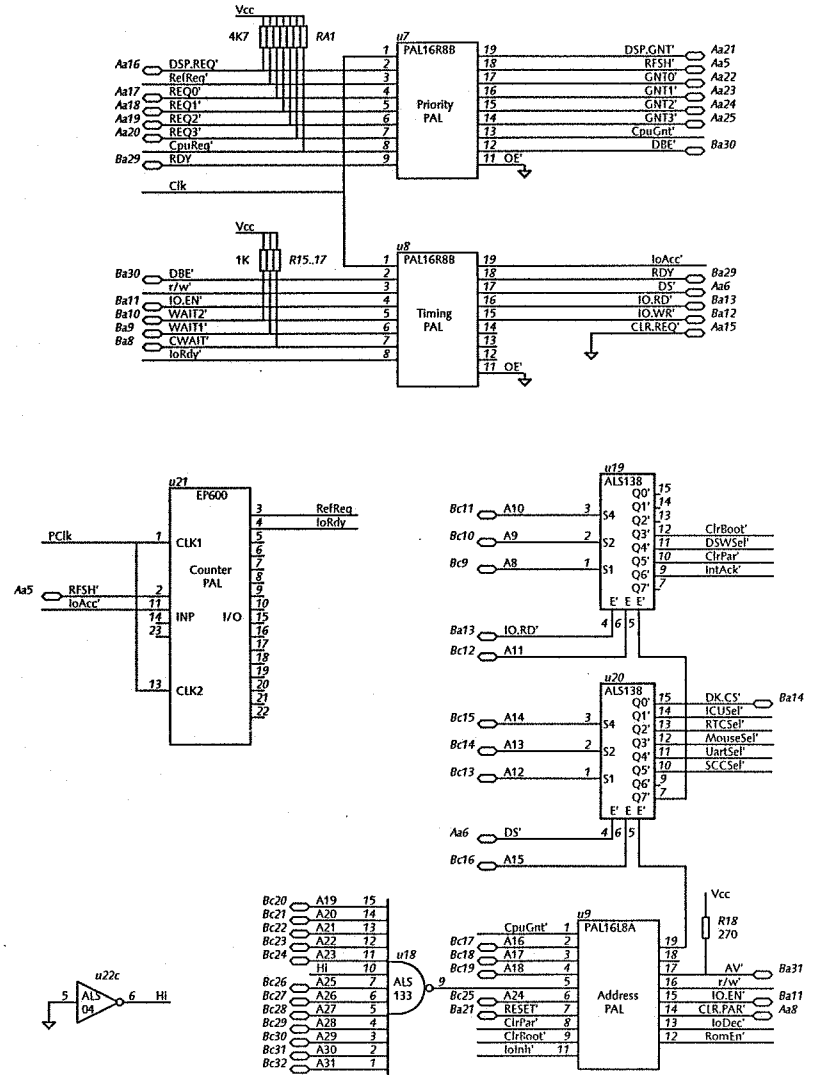

Appendix C: Circuit Diagrams



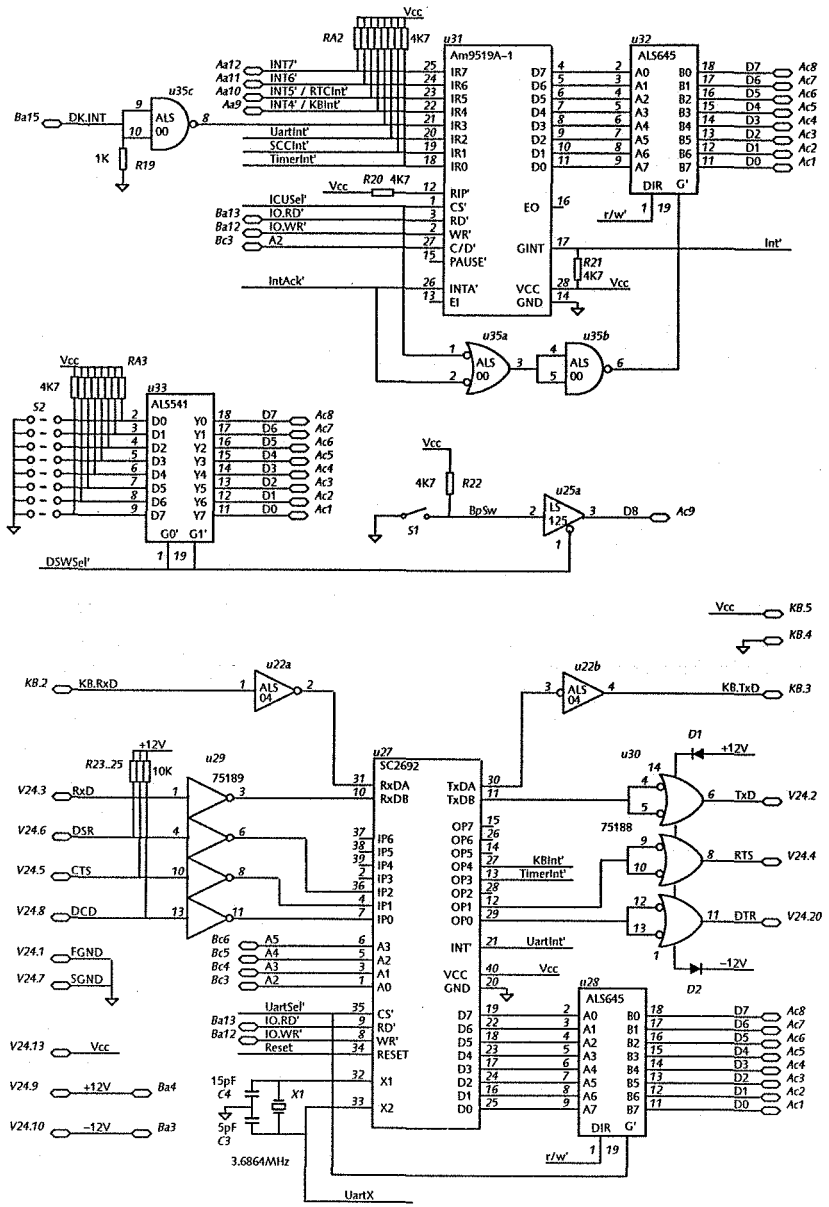
ETH Zürich	ns532.cpu1.SIL Processor Cluster	Author: B. Heeb	Date: 10.6.88



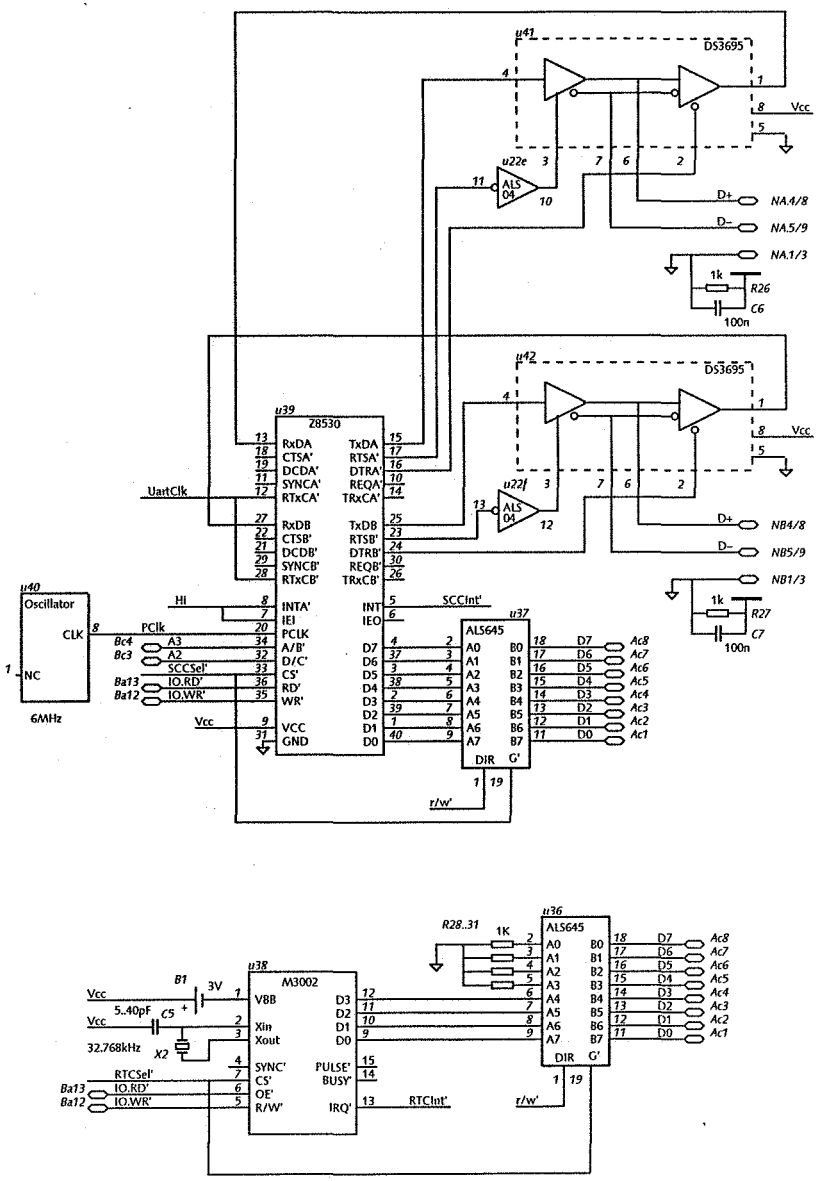
ETH Zürich	ns532.cpu2.SIL Reset–Logic, Buffers	Author: B. Heeb	Date: 10.6.88



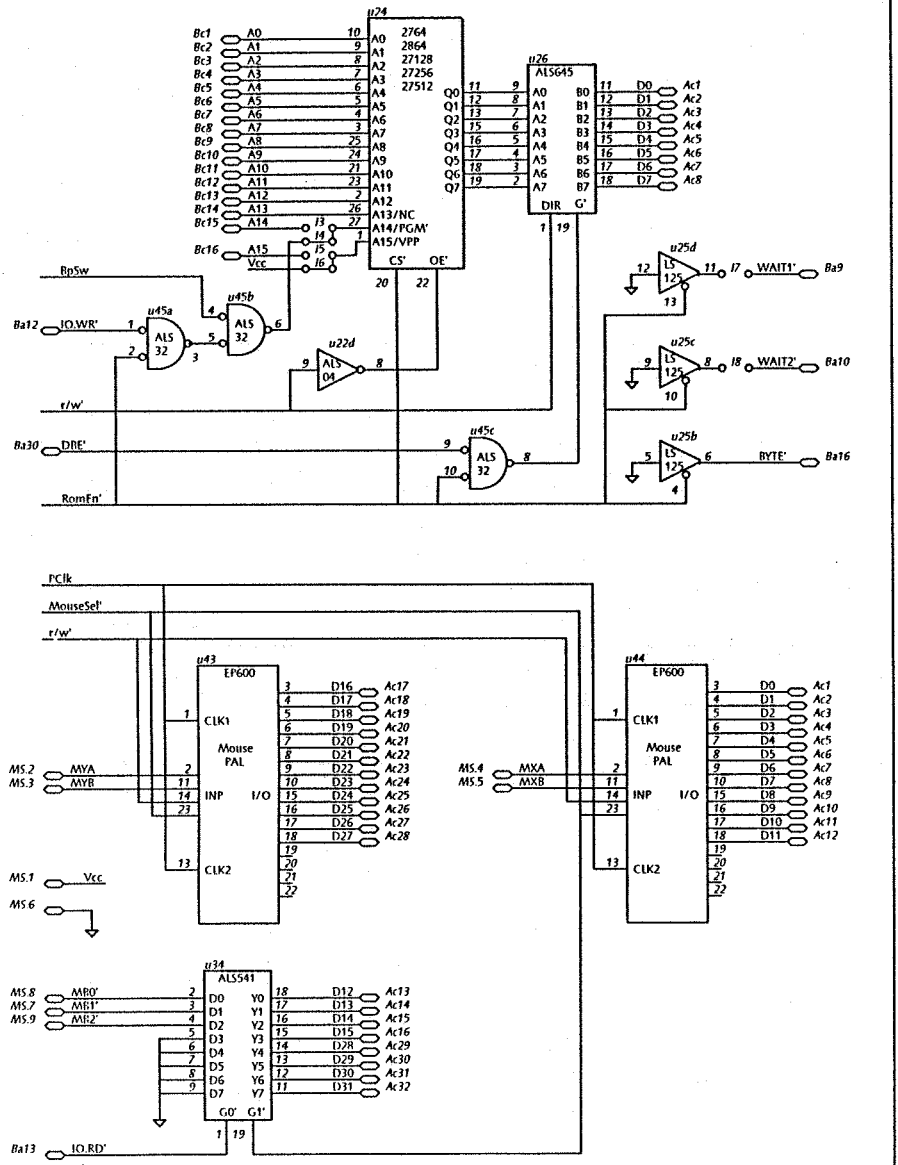
ETH Zürich	ns532.cpu3.SIL Arbiter, Ref. Counter Addr. Decoder	Author: B. Heeb	Date: 10.6.88



ETH Zürich	ns532.cpu4.SIL ICU / DSW / UART	Author: B. Heeb	Date: 10.6.88



ETH Zürich	ns532.cpu5.SIL SCC, RTC	Author: B. Heeb	Date: 10.6.88



ETH Zürich	ns532.cpu6.SIL Boot – ROM Mouse – Decoder	Author: B. Heeb	Date: 10.6.88