Diss. ETH No. 8431

Development and Analysis of a Workstation Computer

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Technical Sciences

---

presented by
Johann Jakob Eberle, Dipl. El.-Ing. ETH
born August 18, 1959
citizen of Häggenschwil (St.Gallen)

accepted on the recommendation of
Prof. Dr. N. Wirth, examiner
Prof. Dr. W. Fichtner, co-examiner

1987

Acknowledgements

I am deeply indebted to Professor N. Wirth for giving me the opportunity to develop the hardware of the workstation Ceres and for supervising this thesis. His profound competence and insight are invaluable.

I thank Professor W. Fichtner for his willingness to be my co-examiner and for advising me on this thesis.

I am grateful to my colleagues for many valuable discussions and for their help. In particular, thanks go to the colleagues and friends who participated in the Ceres project: Frank Peschel and Matthias Wille, who shared the project from the beginning and worked on the system software for Ceres; Immo Noack, who helped in the construction of the Ceres hardware, and thanks to his engagement the Institut für Informatik is now equipped with 50 Ceres workstations. Their efforts were indispensable. I also thank Hans Hablützel for proof-reading the thesis paper.

On behalf of the project team I acknowledge the support by National Semiconductor Corporation and the helpful cooperation of its representative in Switzerland, H. Eisenring.

Contents

Abstract

The workstation Ceres is a stand-alone computer for a single user. The design is an example of a simple system architecture reflected by a careful implementation with minimal costs. Ceres is based on the 32-bit microprocessor NS32032, which is oriented to the use of high-level and modular languages. A key feature is the high-resolution bitmapped graphics display which is attractive for applications such as program development or document processing. The arbitrated memory bus and the modular system organization are open to future hardware extensions.

This thesis documents the hardware development of the workstation Ceres. The design objectives of the raster graphics interface and of the bus structure are discussed in detail. Finally, processor-memory communication of two prototype versions is analysed, which differ only in the width of their data paths to memory.

The raster graphics interface of Ceres contains an integral frame buffer memory, which is directly addressable by the CPU. The frame buffer is based on video RAM technology which ideally meets the high video bandwidth requirements of the 1024 x 800 non-interlaced display. An inexpensive and flexible solution is retained by dispensing with dedicated hardware support for image manipulation.

The backbone of the Ceres computer is the memory bus, which is shared by multiple master devices. The bus is controlled by a centralized arbiter. Short response times are ensured in that the shared memory is re-allocated for every memory cycle according to fixed priorities. A default assignment strategy prevents the processor from being significantly slowed by arbitration delays.

The analysis of processor-memory communication is motivated by the observed small benefit in performance gained by replacing the NS32016 CPU with the NS32032 CPU and thus doubling the memory bus bandwidth. Measurings show that the bus capacity of the NS32032-based Ceres is only used to a small degree. Therefore, the additional costs can hardly be justified. This contrasts with the frequently heard claims of the superiority of 32-bit computers.

Kurzfassung

Ceres ist ein Arbeitsplatzrechner für einen einzelnen Benützer. Der Entwurf ist ein Beispiel einer einfachen Systemarchitektur, welche sich in einer sorgfältigen Implementierung mit minimalem Aufwand widerspiegelt. Ceres basiert auf dem 32-bit Mikroprozessor NS32032, der auf die Verwendung von höheren, modularen Programmiersprachen ausgerichtet ist. Eine Besonderheit ist der hochauflösende Rastergrafik-Bildschirm, welcher Anwendungen wie die Programmentwicklung oder das Bearbeiten von anspruchsvollen Dokumenten attraktiv gestaltet. Der arbitrierte Speicherbus und der modulare Systemaufbau erlauben künftige Erweiterungen der Hardware.

Die Dissertation dokumentiert die Hardware-Entwicklung des Arbeitsplatzrechners Ceres. Ausführlich werden die Entwurfskriterien der Rastergrafik-Schnittstelle und der Busstruktur besprochen. Schliesslich wird die Prozessor-Speicher-Kommunikation zweier Prototypen-Versionen untersucht, die sich lediglich in der Breite ihrer Datenpfade zum Speicher unterscheiden.

Die Rastergrafik-Schnittstelle von Ceres enthält einen separaten Bildschirmspeicher, welcher von der CPU direkt zugegriffen werden kann. Der Bildschirmspeicher ist mit Video-RAMs aufgebaut, die sich auf ideale Weise für die benötigte hohe Video-Bandbreite des ohne Zeilensprungverfahren arbeitenden Bildschirms eignen. Eine kostengünstige und flexible Lösung ist gewährleistet, indem auf spezielle Hardwareunterstützung der Bildmanipulationen verzichtet wurde.

Das Rückgrat des Ceres-Rechners ist der arbitrierte Speicherbus, der von mehreren sendenden, als auch empfangenden Teilnehmern gemeinsam benutzt wird. Der Bus wird von einem zentralen Arbiter verwaltet. Indem jeder Zugriff auf den gemeinsamen Speicher gemäss festen Prioritäten einzeln vergeben wird, werden kurze Antwortzeiten ermöglicht. Liegen keine anderweitigen Busanforderungen vor, so kann die CPU dank einer bevorzugten Behandlung ohne Verzögerung auf den Speicher zugreifen.

Die Analyse der Prozessor-Speicher-Kommunikation wurde durch den geringen Leistungsgewinn veranlasst, welcher beobachtet wurde, nachdem die NS32016 CPU durch die NS32032 CPU ersetzt wurde und damit die Speicherbus-Bandbreite verdoppelt wurde. Messungen zeigen, dass die Buskapazität der NS32032-basierten Ceres nur wenig ausgelastet ist. Die zusätzlichen Kosten sind deshalb kaum zu rechtfertigen. Diese Feststellung steht im Widerspruch zur oft gehörten Ansicht der Überlegenheit einer 32-bit Rechnerarchitektur.

1 Introduction

This thesis documents and analyses the design and implementation of the 32-bit workstation Ceres. The design sets an example of a simple system architecture reflected by a careful implementation. The development was made in an environment free from commercial restrictions such as compatibility with existing products or industrial standards. This was both the chance and the obligation to develop general and powerful concepts.

The machines currently being designed are based on principles that have been known for years. Technology is the motive force for new designs, more than new principles. Old principles are reapplied by using new technologies, they are analysed and, if necessary, improved. Learning about computer hardware engineering means designing and implementing. In this sense, the thesis shall be a contribution to systematic computer hardware engineering.

Personal computing has its roots at the Xerox Palo Alto Research Center, where in 1973 the Alto computer was developed [Thacker 79]. Rather than providing a centralized computing facility, computing power was distributed to its users. Further innovative provisions were the bitmapped raster display and the mouse pointing device.

Under the influence of the Alto computing environment, the Lilith computer was developed in the years 1977 until 1979 [Ohran 84]. With it, personal computing made its entry into the Institut für Informatik of the Swiss Federal Institute of Technology (ETH). The architecture of Lilith is optimized for the development and execution of Modula-2 programs. The processor is realized as a microprogrammed stack machine based on bit-slice technology. While the Lilith processor architecture still compares favourably with today's microprocessors, its implementation is getting on in years.

The rapidly evolving VLSI technology has provided the motivation to design a new workstation. The project was stipulated by Professor N. Wirth in his efforts to systematically develop useful tools for research and education. Notable, already mentioned results of these efforts are the workstation Lilith [Wirth 81a] and the programming language Modula-2 [Wirth 82]. The new workstation has been named CERES, an acronym for Computing Engine for Research, Engineering, and Science. In the old Italian and later Greek mythology, Ceres is the name of the goddess of fertility.

The project started in early 1984 when the basic concepts of the hardware architecture were proposed by Professor N. Wirth. A first prototype was finished a year later in the spring of 1985. The prototype was based on the 16-bit processor NS32016 from National Semiconductor. At that time, future developments of integrated circuits were seen to be concentrated on 32-bit processors. Therefore, a second prototype based on the 32-bit processor NS32032 was developed, which is software compatible with its family member NS32016. In fall 1985, the redesign was complete. By the end of 1986, a series of 30 computers was built. Another series of 20 computers is currently being finished.

The venture to develop a personal computer was shared with Frank Peschel and Matthias Wille, who have ported the Lilith operating system Medos-2 [Knudsen 83, Peschel 87]. A one-pass Modula-2 compiler was developed by Professor N. Wirth [Wirth 86b]. The two prototypes were implemented and debugged by the author alone. Roger Burlet developed the computer cabinet. Immo Noack designed the layouts for the printed circuit boards and was in charge of manufacturing the two series.

The following goals were set for the development of the workstation Ceres:

- The computer, particularly the processor has to efficiently support the execution of high-level language programs. The architecture should incorporate one of the recent microprocessors that claim to be oriented to the use of high-level languages.
- A high-resolution, flicker-free display and a mouse pointing device have to be included in order to provide flexibility and comfort for the human interaction with the computer.
- The design of the computer has to be simple and systematic. The realization has to be modular and extensible.
- Development and manufacturing costs have to be minimized. Only available standard components must be used.

The thesis consists of seven chapters. Chapter 2 contains a detailed hardware description of the workstation Ceres. Chapter 3 adds some design considerations made during the development of the raster graphics interface for Ceres. Chapter 4 provides a classification of microcomputer buses, and based on it discusses the bus structure of Ceres. Chapter 5 examines processor-memory communication of Ceres in order to find an explanation for the observed low performance benefit gained by the doubled memory bus bandwidth of the second prototype. Chapter 6 describes the experiences with complex integrated circuits made during the period of development. Finally, Chapter 7 summarizes the results presented in this thesis.

2 Hardware Description of the Workstation Ceres

2.1 Introduction

The hardware of Ceres sets an example of a simple system architecture reflected by a clean implementation. The concise description is presented within this chapter and is appreciated in particular by the hardware designer who wants to add his own extensions and by the software designer who wants to write system programs. Hardware and software have to be designed together in order to attain efficiency and reliability. The basis is a brief and comprehensible documentation of the design.

The entire machine is implemented with 250 integrated devices ranging from SSI up to VLSI. Nearly half of these components are storage devices. Random logic is provided by standard TTL devices, mostly ALS and AS; where a desired function could not appropriately be realized with available fixed TTL functions, programmable logic was used. The circuits are mounted on four printed circuit boards that are connected by a backplane board. All printed circuit boards are fabricated with four metal layers allowing separate power and ground planes in order to minimize electrical problems.

The following sections contain a textual description of the Ceres hardware. The corresponding schematic circuit drawings are contained in Appendix A. The functional specifications of the programmable logic devices are listed in Appendix B.

2.2 Hardware Structure

The Ceres hardware consists of a 32-bit processor based on the National Semiconductor Series 32000 chip set, primary memory, secondary memory, and miscellaneous input and output devices. These include a high-resolution display, a serial keyboard, a mouse pointing device, an RS-232-C serial line interface, and an RS-485 serial line interface. Figure 2.1 shows a block diagram of the hardware structure. This section gives a brief description of the main hardware characteristics.

Processor

The switchboard of the Ceres computer is a National Semiconductor NS32032 32-bit microprocessor. Two slave processors add capabilities for virtual memory management and floating-point arithmetic. The processor operates at a clock rate of 10 MHz, resulting in a memory cycle time of 400 ns. It has an addressing range of 16M bytes. Its repertoire includes 83 basic instructions with 9 addressing modes [NS 86a].

Primary Memory

The primary storage of Ceres consists of 2M bytes of dynamic RAM, 256K bytes of video RAM, and 32K bytes of ROM. The former is implemented with 256K-bit dynamic RAM chips. Parity checking makes it possible to detect single bit errors within a data byte. A special type of dynamic RAM, a 64K-bit video RAM, is used to store the display bitmap. 64K-bit chips form the ROM memory for bootstrap and diagnostic software.

Secondary Memory

The secondary storage of Ceres consists of a Winchester hard disk drive and a floppy disk drive. The 5 1/4" hard disk has a formatted capacity of 40M bytes, an access time of 40 ms, and a data transfer rate of 5 Mbits per second. For backup, a 3 1/2" floppy disk is available with a formatted capacity of 720K bytes, an access time of 94 ms, and a data transfer rate of 250 Kbits per second.

Input/Output Devices

The display is a high-resolution 17" raster scan monitor. It can display 819'200 dots which are stored in a matrix called bitmap that is 1024 dots wide and 800 dots high. The picture is refreshed at a rate of 62.15 frames per second (non-interlaced) which results in a nearly flicker-free image. The bitmap information is stored in a separate, dedicated memory implemented with video RAMs.

In addition to a standard serial ASCII keyboard, an opto-mechanical, three-button mouse is provided with a resolution of 380 counts per inch.

The standard RS-232-C serial interface works with asynchronous data transfer rates from 50 to 38'400 bits per second. A higher transmission speed can be obtained with two RS-485 serial ports for data transfer rates up to 230.4 Kbits per second. In a multipoint configuration, this interface allows the implementation of a low-cost computer network.

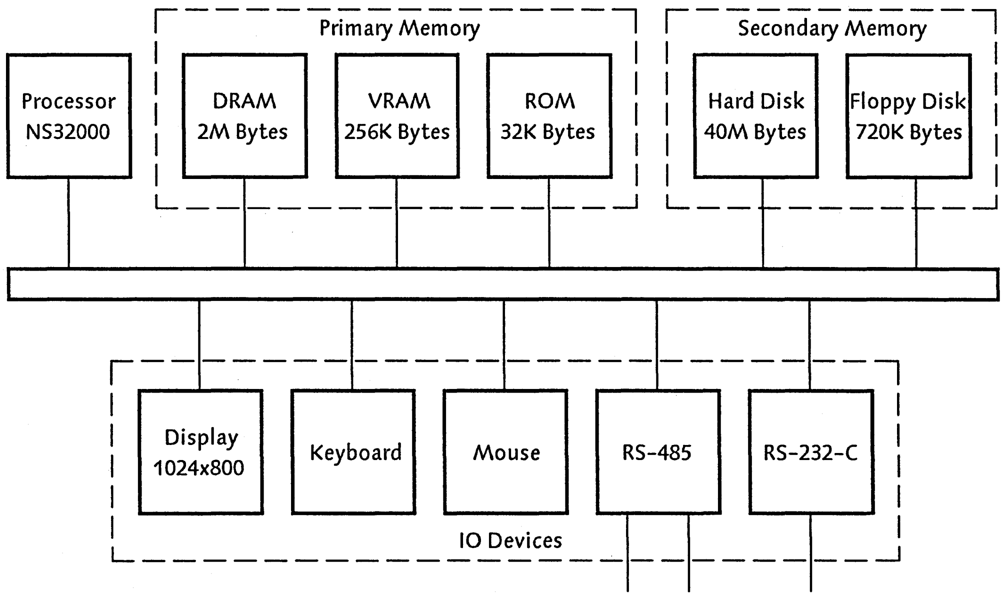


Figure 2.1 Hardware structure of the Ceres computer.

2.3 Hardware Implementation

The Ceres computer is packed in a 465 mm x 190 mm x 365 mm cabinet small enough for desktop application. Display monitor, keyboard, and mouse are separate and individually connected to the computer cabinet. The cabinet houses the power supply, the hard disk drive, the flexible disk drive, and the card cage, which can hold up to six circuit boards. The board dimensions are the extended double-Eurocard format, which is 220.0 mm x 233.4 mm. Packaging and the parallel interconnection structure of the backplane make it possible to access every signal with a scope probe for debugging and maintenance without providing additional facilities such as a bus extension card.

The power supply is rated for a wattage of 130 W. Excluding the display monitor, the basic configuration of Ceres consumes 78 W during power-up and 57 W during normal operation (typical values). The capacity of the power supply, therefore, will still be sufficient if hardware extensions are added. It must be noted that the hard disk drive and the disk controller consume about as much as half of the figure given for normal operation.

The hardware of Ceres is physically divided into several boards which are connected by the memory bus:

- the *processor board* contains the processor chip set, the memory bus access and timing controller, the boot ROM, and various IO devices
- the *memory board* holds the dynamic RAM memory
- the *display controller board* comprises the video RAM memory for the displayed bitmap and the logic to serialize the bitmap data into the video refresh data
- the *disk controller board* combines a controller for both the hard disk and the floppy disk drives
- all boards communicate via the *motherboard* which contains the memory bus

Based on the circuit diagrams in Appendix A, the hardware of Ceres is explained in the following sections. The specifications of the integrated circuits used are contained in the referred data sheets.

2.3.1 Processor Board

Processor

The NS32032 central processing unit (CPU) has a uniform linear 16M-byte addressing range and a full 32-bit architecture and implementation [NS 86a]. Internal working registers, internal and external data paths, and ALU are all 32-bit wide. There are eight general purpose registers which provide local, high-speed storage for the processor, such as holding temporary variables and addresses. Eight dedicated registers are used to store address and status information. The register set, the supported data types, and the instruction set are fashioned after high-level language instructions [NS 84a]. Code generation is made easier by a high degree of symmetry. (Note: A processor's architecture is said to be symmetrical if every supported data type is provided with a complete set of operators and if each operator can use any addressing mode to access operands.) An analysis of the NS32000 architecture in respect of the code generation by a compiler is contained in [Wirth 86a].

A slave processor is an auxiliary processing unit which operates in coordination with the CPU. The NS32082 memory management unit (MMU) performs address translation, virtual memory management, and memory protection [NS 86a]. The NS32081 floating-point unit (FPU) operates on two floating-point data types: single precision (32 bits) and double precision (64 bits). Arithmetic operations include Add, Subtract, Multiply, Divide, and Compare. Several Move and Convert instructions are also available.

The structure of the processor and its memory bus interface are illustrated in the block diagram of Figure 2.2. The following blocks may be distinguished:

- the *processor cluster* consists of the NS32032 CPU, the NS32082 MMU, and the NS32081 FPU
- the *timing control unit* generates the clock and reset signals for the slave processor and the memory bus
- the *memory bus interface* connects the address, data, and control signals of the local, multiplexed slave processor bus to the demultiplexed memory bus

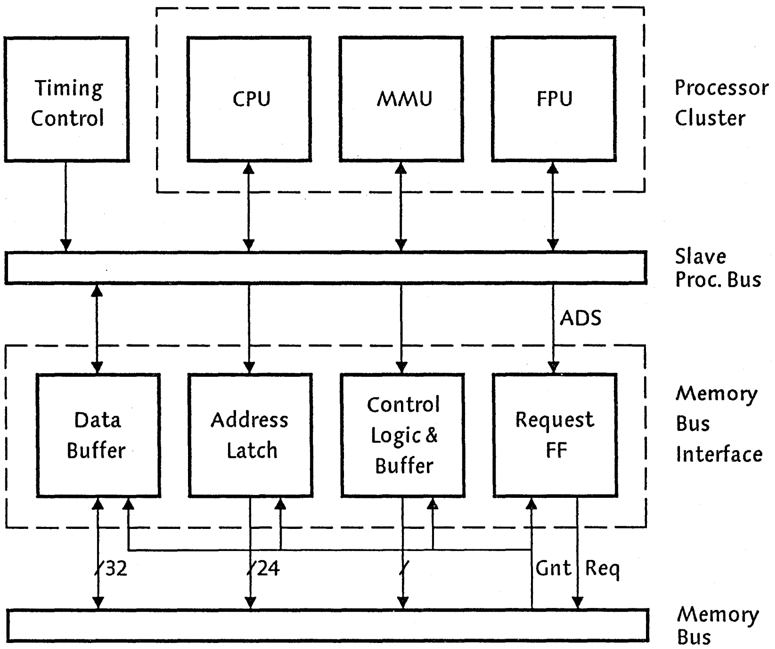


Figure 2.2 The processor and its memory bus interface.

The *processor cluster*, i.e. the CPU (u24) and its slave processors (u23, u25) are connected to a local, multiplexed address and data bus (ad0-ad23, d24-d31) that combines 32 bits of data with 24 bits of address. The local bus is required either for memory access (or access to IO devices which are memory mapped) or slave processor communication. In the former case, the memory bus is transparent to the slave processor bus. In the latter case, only the two least significant bytes of the data bus are used. Note that the CPU is solely responsible for memory access, i.e. operands of a slave processor instruction are always fetched from memory by the CPU.

The NS32201 *timing control unit* (TCU, u36) provides a two phase, non-overlapping 10 MHz clock (TCU.PHI1, TCU.PHI2), which is used by the processor chips [NS 86a]. In addition, a 10 MHz and a 20 MHz TTL compatible clock (TCU.CTTL, TCU.FCLK) are generated. The timing waveform of these clock signals is shown in Figure 2.3. The TCU also provides circuitry that meets the reset requirements of the processor chips. If the reset input line RSTI' is pulled low, the TCU asserts TCU.RST' which resets the processor chips. The RSTI' input signal is provided by the TL7705 (u35) which contains a power voltage sensor and a debounce circuit. It is activated at power-up or when the externally mounted reset button has been pressed. The reset and clock signals on the memory bus (RESET' and CLK, FCLK) are buffered versions of the corresponding TCU signals (u7).

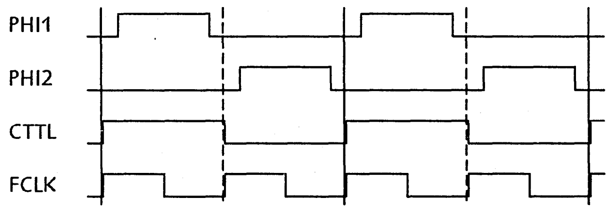


Figure 2.3 TCU clock signals.

The *memory bus interface* consists of a 32-bit wide data buffer, a 24-bit wide address latch, a buffer for several control lines, and a circuitry that requests a bus cycle when the processor wants to access the memory. The data buffer is made up of four 74ALS645 octal bus transceivers (u3-u6) [TI 83c]. Two additional 74ALS645s (u1, u2) are needed for the MMU with its 16-bit wide data bus to access the higher data word of the main memory. When the MMU accesses an odd memory word (A1=1), the higher data word of the memory bus (D16-D31) has to be mapped onto the lower word of the processor bus (ad0-ad15). The signals GW' and GD', required for the buffer enable inputs, are generated by part of a PAL16L8A PAL device (u21) [MMI 78]. The MMU has to access the memory in order to update its internal address translation cache from page table entries in memory or to update certain status bits within them.

The address latch uses three 74ALS573 octal D-type transparent latches (u9-u11). Using the address strobe signal MMU.ADS', the information of the multiplexed address/data bus is retained by the latches at the beginning of a bus cycle. At power-up or after a system reset, a flip-flop (u40a) with the output signal name BT.UP' is set. If BT.UP' is asserted and the CPU is accessing memory, the address information of the signal lines ad19-ad23 will not be gated to the memory address bus lines A19-A23; instead, another ALS573 (u8) sets these lines to high as long as BT.UP' enables this latch. This maps address locations 000000-07FFFF (hex) to F80000-FFFFFF (hex) where the boot ROM and IO devices are located. Note that the processor starts program execution with a PC value zero after reset. The boot flip-flop is reset irreversibly under software control.

The following control signals are provided: ILO', AV', R/W', and BE0'-BE3'. ILO' is a buffered version of the corresponding CPU signal CPU.ILO' (u7), which indicates that an interlocked instruction is being executed. It is made available to external bus arbitration circuitry in order to implement the semaphore primitive operations for resource sharing. This signal is, however, not used in the present circuits. AV' marks a valid address on lines A0-A23 and is activated when a processor request has been granted (u34b, u7). R/W' indicates the direction of the data transfer as seen from the processor (u38c, u7). BE0'-BE3' facilitate individual byte accessing on the 32-bit data bus. Any data item, regardless of size, may be placed starting at any memory address; therefore, the 24-bit address A0-A23 is a byte address. While the data bus always transfers double-word data, the memory uses BE0'-BE3' to select the appropriate bytes. A PAL16L8A device (u21) contains the necessary logic to generate the byte enable signals. During a memory write cycle, these signals are defined by either the CPU (CPU.BE0'-CPU.BE3') or the MMU (A1). A CPU memory access can contain one, two, three, or four bytes, while the MMU always accesses words. An MMU memory cycle can be identified if MMU.MAC' is low. During a memory read cycle, BE0'-BE3' are all active. This precaution must be taken to prevent floating data buffer inputs caused by non-selected memory devices.

The cycle request circuitry consists of a 74AS74 flip-flop (u22a). It is set by the address strobe signal MMU.ADS', which signals that the processor is starting a bus cycle. A CPU memory cycle request (signalled by a low CPU.REQ' signal) is acknowledged by the bus arbiter with an active CPU.GNT' signal. CPU.GNT' is used as an output enable of the buffers and latches of the memory bus interface. The RDY signal is used to extend the current processor bus cycle. This is necessary if the CPU bus cycle request cannot be acknowledged immediately (u39b) or in case of a slow access (u34a).

Memory Bus Arbiter

Processor, display controller, and DRAM refresh timer share access to the main memory and the memory bus. The device that controls the bus is known as the bus master. The transfer of bus control from one device to another is defined by a set of bus request and bus grant signals. The circuit is outlined in Figure 2.4. The arbiter consists of a *priority register* and a *bus control* unit that controls the timing of a memory cycle. The priority register is made up of a PAL16L8A (u16) and a 74AS573 octal D-type transparent latch (u15). The bus control signals are generated by a finite state machine (FSM) built from two PAL16R8As (u13, u14). A detailed description of the bus control FSM including a state diagram is contained in Appendix B. Note that the state machine is clocked by the fast clock (f = 20 MHz) in order to achieve higher granularity.

The sequence of events during a read and a write memory cycle is shown in Figure 2.5. A full speed memory cycle is performed in four cycles of the processor clock CLK, labeled T1 through T4. Clock cycles not associated with a memory cycle are designated Ti (for "idle"). In order to acquire control of the bus, the device asserts its bus request signal that is fed into the priority register. The highest-order signal applied at a request input is transferred to the appropriate grant output. If any request has been submitted to the priority register, ANY' becomes low, thereby informing the bus control FSM that a memory cycle has to be started. The FSM responds with a low G signal causing the state of the request lines to be latched by the priority register. At the end of a memory cycle, the signal CLR.REQ' clears the processed request. The following bus master devices are provided (listed in descending priority):

DSP.REQ'	DSP.GNT'	Display refresh controller
REF.REQ'	REF.GNT'	DRAM refresh timer
REQ0'	GNT0'	not used

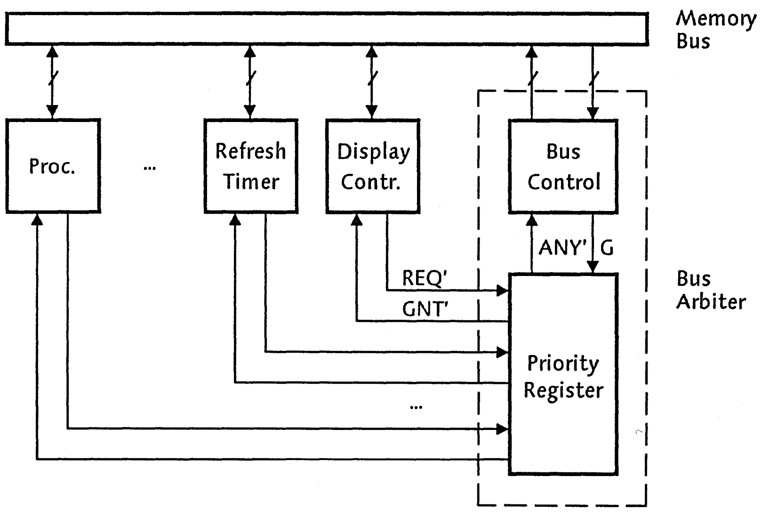


Figure 2.4 Memory bus arbiter.

REQ1'	GNT1'	not used
REQ2'	GNT2'	not used
REQ3'	GNT3'	not used
CPU.REQ'	CPU.GNT'	Processor

The bus control FSM provides further control signals that are specifically introduced to suit the NS32000 processor chips, but are general enough to serve other master devices as well. The data buffer enable signal DBE' is used to control the data bus buffers. The leading edge of DBE' is delayed a half clock period during read cycles to avoid bus conflicts between data buffers and either the CPU or the MMU. As the multiplexed slave processor bus holds the address until shortly after the end of T1, conflicts occur if data buffers are opened too early. DBE' goes inactive in the middle of T4, having provided the necessary data hold times. If the processor is performing a read cycle, the data bus is sampled at the end of T3. The data strobe DS' signals the beginning of a data transfer. This signal is used by the control circuitry for the dynamic RAMs. The leading edge of DS' is delayed a half clock period during write cycles to guarantee the appropriate data setup time for the DRAMs. DS' returns to the high level at the beginning of T4. During a write cycle, the processor presents data from the beginning of T2 to the end of T4.

To allow sufficient strobe widths and access times for any speed of memory or peripheral device, the bus control FSM provides cycle extension. As explained in Section 4.3.2 the arbitrated memory bus does not allow the use of the cycle extension capabilities of the TCU. The FSM uses the following wait input signals (listed in descending priority):

- a low IO.EN' during T2 causes the FSM to perform a so-called peripheral cycle, which is characterized by four added wait states (TW4, TW3, TW2, TW1). In addition, a read or write strobe signal (IO.RD', IO.WR') is generated which meets the setup and hold timing requirements of slower peripherals. IO.RD' and IO.WR' are decoded from R/W'
- if WAIT2' is sampled low during T2, two wait states (TW2, TW1) are inserted

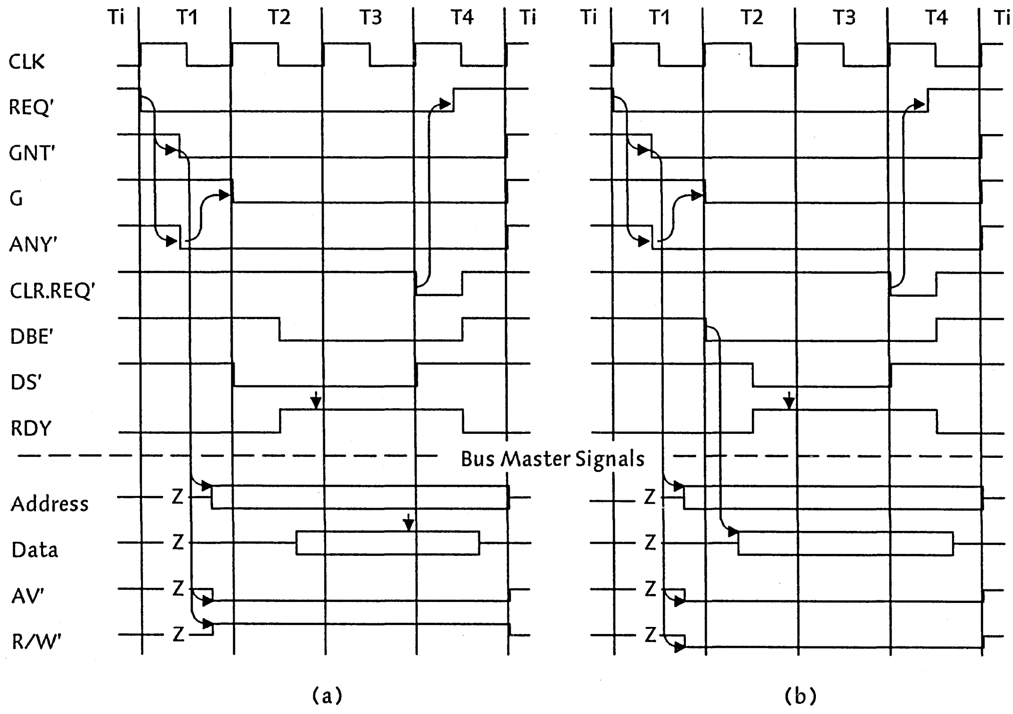


Figure 2.5 Read (a) and write cycle timing (b).

- if WAIT1' is sampled low during T2, one wait state (TW1) is inserted
- CWAIT' initiates a continuous wait. As long as sampled low during T2 and TW1, one wait state (TW1) is inserted

Examples of cycle extension are shown in Figure 2.6. The processor is informed of an extended bus cycle by means of the RDY signal. At the end of T2, the RDY signal is sampled by the CPU or MMU. If RDY is high, the next T-states will be T3 and then T4 ending the bus cycle. If RDY is low, then another T3 state will be inserted after the next T-state, and the RDY line will again be sampled during the next T-state.

Although the processor has the lowest-order priority and thereby looses competition with any other bus masters, it is treated in a privileged way. Whenever no other master requests the bus, the processor is given control over the memory bus by default; as a result, there is no arbitration delay in case of a memory access by the processor.

The introduction of the address valid signal AV' is necessary for the following reasons. Since the processor also controls the memory bus during idle times, AV' is used to indicate a valid memory address during a memory bus cycle. Furthermore, bus masters such as the refresh timer for the DRAMs request so-called "dummy" bus cycles in order to prevent other devices from simultaneously accessing the memory bus. AV' is then set to inactive, hindering any bus slave in decoding the address.

The mentioned DRAM refresh timer is placed on the processor board. It is assumed that a central timer is responsible for refreshing all dynamic memory devices. The refresh timer

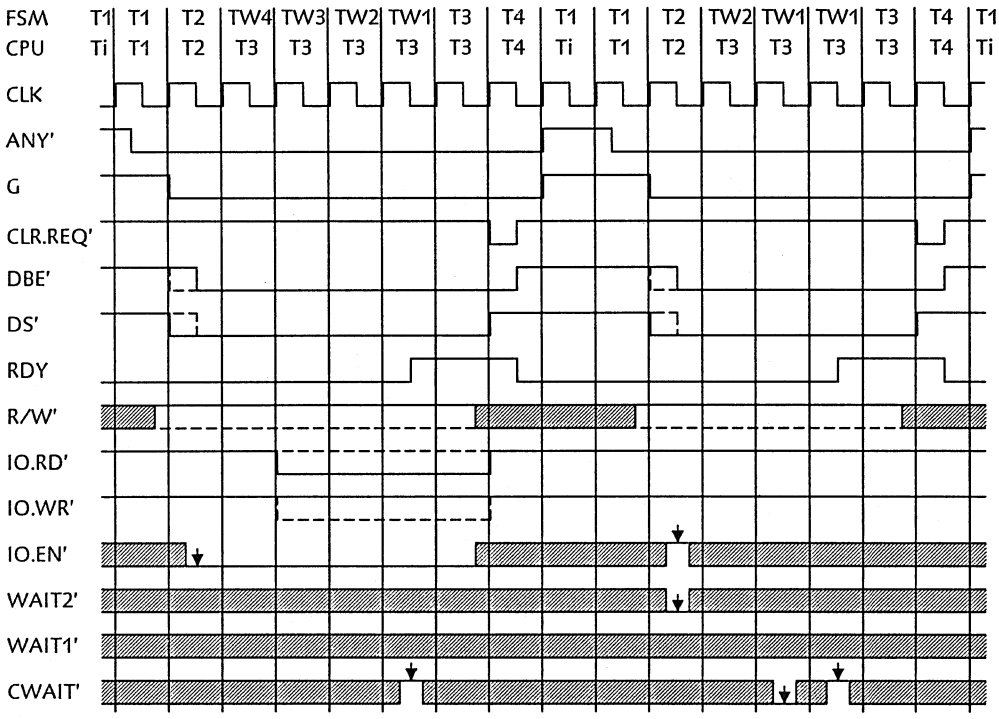

FSM T1 T1 T2 TW4 TW3 TW2 TW1 T3 T4 T1 T1 T2 TW2 TW1 TW1 T3 T4 T1
CPU Ti T1 T2 T3 T3 T3 T3 T3 T4 Ti T1 T2 T3 T3 T3 T3 T4 Ti
CLK
ANY'
G
CLR.REQ'
DBE'
DS'
RDY
R/W'
IO.RD'
IO.WR'
IO.EN'
WAIT2'
WAIT1'
CWAIT'

Figure 2.6 Cycle extension.

consists of a 74LS393 dual 4-bit counter (u45) which divides the system clock CLK by 160. The refresh request line REF.REQ' is, therefore, asserted every 16 µs. A memory refresh cycle is indicated by a low RFSH' signal.

Boot ROM and Standard Input/Output Devices

The *boot ROM* and several *standard IO devices* are also on the processor board. Part of the address space is assigned to IO ports. This strategy is called memory-mapped IO with devices residing in the reserved IO address space loosely called IO devices. An *address decoder* provides the appropriate select signals. As can be seen in Figure 2.7, the IO devices include:

- a dual universal asynchronous receiver/transmitter (*UART*) that interfaces the serial keyboard and offers an additional RS-232-C serial port
- a dual-channel serial communications controller (*SCC*) providing two RS-485 serial interfaces
- a *mouse interface*
- a battery-backed real time clock (*RTC*)
- a *DIP-switch* holding system parameters
- an interrupt control unit (*ICU*) supporting up to eight interrupt sources

The width of the 32-bit data bus is not fully used by the peripheral devices. Their data paths are 4-, 8-, or 16-bit wide. The data bus interfaces are aligned with the 32-bit data bus using

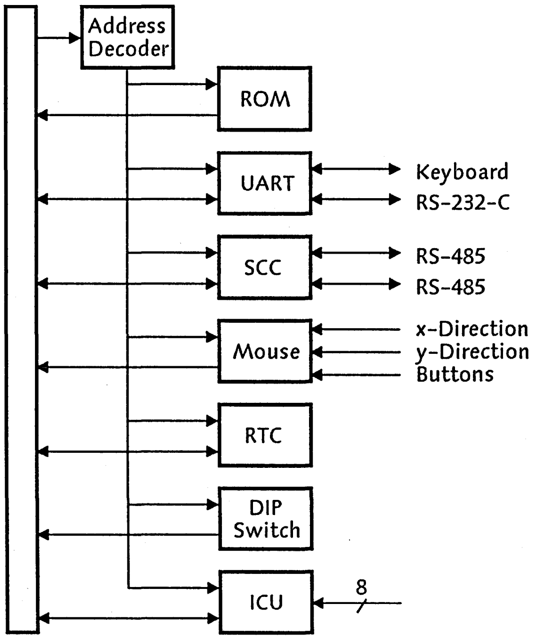


Figure 2.7 Boot ROM and standard IO devices.

the lower-order data bits. An 8-bit peripheral unit, for example, is connected with data bits D0–D7. All IO devices are accessed with addresses modulo 4 equal 0, i.e. device register addresses are double-word addresses and address bits A0 and A1 are ignored.

A PAL20L8A (u12) and a 74ALS138 3- to 8-line decoder (u57) implement the *address decoder*. The PAL device provides the ROM and IO device enable signals ROM.EN' and IO.EN'. The reserved memory locations for ROM and IO devices are shown in Figure 2.8. To simplify future IO expansions, IO.EN' is also available on the memory bus. This signal further causes the arbiter to perform an extended, peripheral cycle. For the two uppermost 512-byte-sized IO pages, additional select signals are generated (IO.PG0', IO.PG1'). The ICU resides in the uppermost IO page (IO.PG0'). This is required by the fact that the CPU reads the interrupt vector from the fixed address FFFE00 (hex) [NS 86a]. The ICU chip select signal (ICU.CS') must not be activated when the CPU reads a dummy byte from address FFFF00 (hex) during a nonmaskable interrupt sequence; therefore, ICU.CS' is disabled if A8 is high (u62d). The next lower IO page (IO.PG1') is reserved for the other standard IO devices. A 74ALS138 (u57) provides eight select signals each having an address range of 64 bytes.

The *boot ROM* is made up of four EPROM devices (u41–u44). The corresponding sockets can be configured for different ROM types (2764, 27128, 27256, 27512) with a range in total memory capacity from 32K bytes to 256K bytes. 150 ns parts are required in order to avoid wait states. The ROM data outputs are connected to the memory data bus with four 74ALS541 octal unidirectional buffers (u29–u32). The ROM address inputs are connected to the address lines A2–A17 (double-word address).

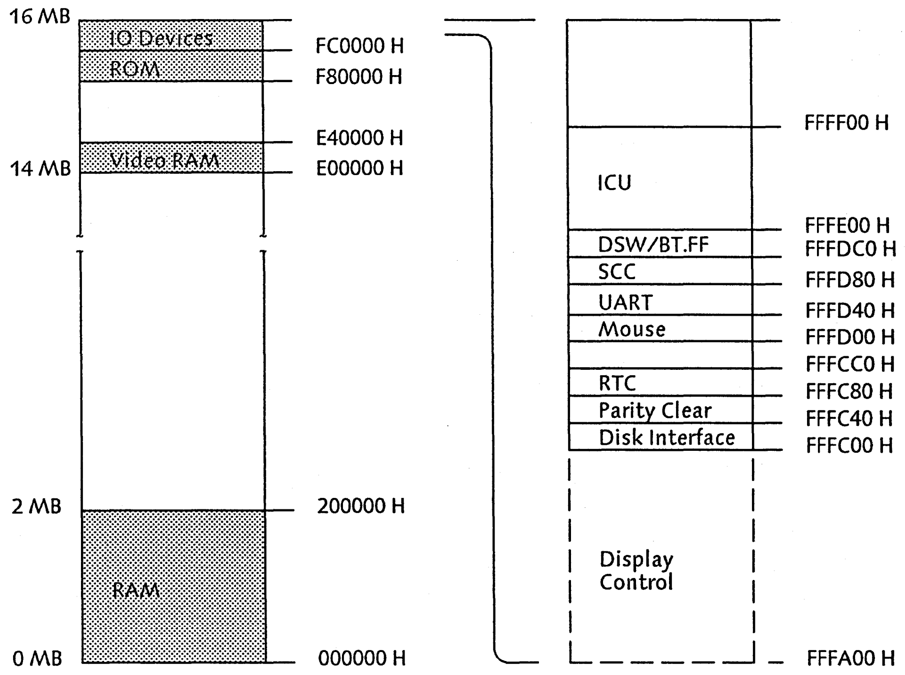


Figure 2.8 Memory map.

A SC2681 *UART* (u47) provides two independent, full-duplex, asynchronous receiver/transmitter channels with software selectable baud rates up to 38'400 bits per second [Philips 83]. One channel is used for the keyboard. The receive and transmit data signals of the keyboard interface (KB.TxD', KB.RxD') are TTL compatible; the other channel implements an RS-232-C interface. A standard RS-232-C line driver (75188, u59) and a line receiver (75189, u60) [TI 77] are used to provide the data transmission and the most common modem control signals: TxData, RxData, Request to Send, Data Terminal Ready, Clear to Send, Data Carrier Detected, and Data Set Ready. Also provided on the UART chip is a programmable 16-bit counter/timer. Individual interrupt signals are output by the UART for the keyboard interface (KB.INT'), the RS-232-C interface (UART.INT'), and the counter/timer (UART.C/T). The crystal oscillator of the UART requires an external 3.6864 MHz crystal. A buffered version of this clock signal is also used by the SCC.

The Z8530 *SCC* (u50) is a dual-channel, multiprotocol data communication peripheral [Zilog 82a, Zilog 82b, Zilog 85]. The SCC can handle asynchronous and synchronous formats including SDLC. In the latter case, data rates up to 230.4 Kbits per second are possible. Each of both channels constitutes an RS-485 serial line interface using DS3696 high-speed differential tristate line transceivers (u61, u64) [NS 83]. The SCC's "request to send" output (RTSA', RTSB') defines the data transmission direction. The "clear to send" input (CTSA', CTSB') is used to detect a line fault condition (LFA', LFB'), which is reported by the transceiver in case of bus contention or fault situations that cause excessive power dissipation within the device. The SCC requires an external 6 MHz clock oscillator (u51). The 3.6864 MHz clocking signals for the receiver/transmitter channels are derived from the UART's oscillator circuit.

The *mouse interface* keeps track of the relative mouse position and holds the state of the three mouse buttons. A direction discriminator controls the up/down counter for the x- and y-directions. The three switches can be directly read on a parallel port and polled by software. The mouse interface is composed of the following components. A 74ALS138 3- to 8-line decoder (u56) provides select signals for the x-register (RX'), y-register (RY'), and button state register (RB'). All registers are read-only. The state of the mouse buttons (MB0', MB1', MB2') is isolated from the data bus (D0-D2) by a 74ALS244 octal buffer (u54), which is enabled by RB'. For each direction the mouse generates two phase-shifted signals (MXA, MXB and MYA, MYB). This information is evaluated by the direction discriminator which is realized with a PAL16R8A (u55). This device generates the necessary control signals for the x- and y-counters. Each counter is made up of two cascaded 74F779 8-bit counter chips (u17-u20) [Philips 84]. A built-in tristate IO port reduces the part count of the data bus interface.

The M3002 *RTC* chip (u66) contains a time of day clock and a calendar [MEM 84]. The register address and data are multiplexed over four data lines; therefore, no separate address lines are needed. External components include a 32.768 KHz crystal for the on-chip oscillator and a battery back-up to keep time and date when no external power is supplied. Because of the low power consumption of this device, the lithium cell provided has a lifetime of more than 10 years.

The *DIP-switch* (u27, u28) holds 8 bits of information (read-only). The off-position corresponds to a logic 1. The switches can be used to set the processor configuration, the size of installed memory, or a machine number in a network.

The Am9519A-1 *ICU* (u52) accepts up to eight maskable interrupt request inputs, resolves priorities, and supplies programmable response bytes for each interrupt [AMD 80, AMD 84]. The latter feature allows the CPU to acknowledge interrupt requests in the so-called vectored mode, interpreting the ICU's response byte as a vector value. Depending on the applied address, an additional circuit (u62) distinguishes between a "normal" access to the ICU's register (icu.cs') and an interrupt acknowledge cycle (icu.inta'). The group interrupt output ICU.INT' is synchronized with the rising edge of TCU.CTTL (u22b) in order to minimize the possibility of metastable states as recommended in [NS]. The ICU inputs the following interrupt signals (listed in descending priority):

INT0'	counter/timer (UART.C/T)
INT1'	two RS-485 channels (SCC.INT')
INT2'	RS-232-C interface (UART.INT')
INT3'	disk controller (DK.INT')
INT4'	keyboard (KB.INT')
INT5'	real time clock (RTC.INT')
INT6'	not used
INT7'	not used

Interrupt lines INT4'-INT7' are available on the backplane bus. In particular, INT6' and INT7' are provided for future IO device expansion.

The address decoder further generates the signals BT.CS' and PAR.CLR'. Any write access to an IO address assigned to BT.CS' clears the boot flip-flop (u40a). The parity error flag is reset during a hardware reset (u34c) or by accessing an address assigned to PAR.CLR' (u39c).

2.3.2 Memory Board

The Ceres memory board contains 2M bytes of dynamic memory and is occupied by 72 DRAM devices organized with a 36-bit wide data bus which allows for 32 bits of data plus byte parity. The memory is designed to accept 256K-bit 120 ns dynamic RAM chips, which operate with the processor at 10 MHz without wait states. Memory can be expanded by additional memory boards.

The organization of the memory is shown in the block diagram of Figure 2.9. In addition to the *memory array*, the following components are needed:

- the *board selection* logic allows the board to be activated for different address ranges
- the *memory control* logic takes care of the proper sequence of a memory cycle. Derived from the original address, the row and the column address together with the appropriate row and column address strobe signals are generated successively.
  Further, the memory control logic is periodically forced to refresh the dynamic memory chips. The contents of a refresh counter are then sent as the address to the memory array
- the *error detection* unit generates parity bits (write-cycle) and checks the read information (read-cycle)
- the bidirectional *data line buffers* connect the data paths of the memory array and the memory bus

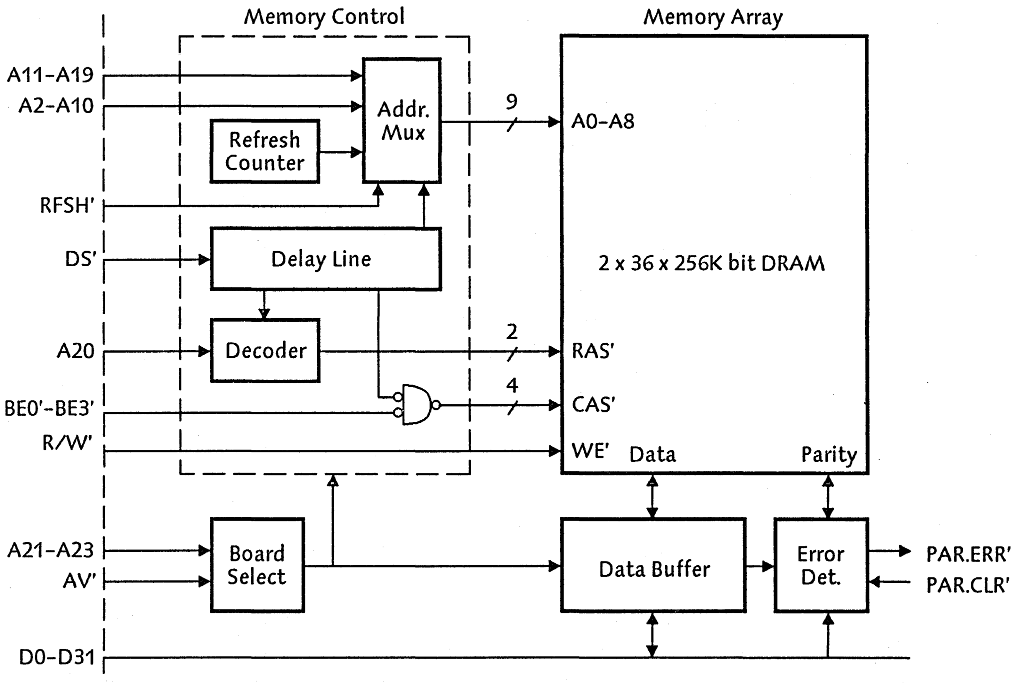


Figure 2.9 The 2M byte dynamic memory.

The *memory array* is divided into two banks each consisting of 36 DRAM devices. Address lines A2-A20 provide a double-word address. Individual byte accessing is controlled by the byte enable signals BE0'-BE3'.

The *board selection* logic uses a 74ALS138 3- to 8-line decoder (u10). If the signal AV' indicates a valid address, the three most significant address bits A21-A23 are decoded and assign an address range of 2M bytes to each decoder output. One of these is chosen as the board select signal MCS' by closing the appropriate jumper (u9).

Most functions of the *memory control* logic are provided by the DP8419 DRAM controller (u12) [NS 86a]. The higher-order address bits A11-A19 serve as the row address and the lower-order address bits A2-A10 as the column address. These addresses are output sequentially on the address lines a0-a8 that drive the memory devices. The address strobe signals RAS' and CAS' are generated by an internal delay line induced by the signal DS'. The corresponding timing diagram is shown in Figure 2.10. Individual control lines for each memory bank (RAS0', RAS1') and for each byte (CAS0'-CAS3') are generated by using the signals A20 and BE0'-BE3', respectively. A20 is used as an input to the internal bank decoder of the DRAM controller. BE0'-BE3' ORed with CAS' yield CAS0'-CAS3'. The write enable signal WE' is a buffered version of the bus signal R/W'. The DRAM controller uses high output current drivers for all address and control lines. External damping resistors reduce both overshoot and undershoot on these signal lines caused by the high-capacity load of the memory devices. The DRAM controller performs a "normal" memory cycle, if the CS' input driven by the MCS' line is activated and the mode input M0-M2 is set to the so-called auto access mode. This implies that the signal RFSH' must be inactive. If RFSH' is active, a refresh memory cycle takes place. The state of the address lines A2-A23 including MCS' is not relevant in this mode.

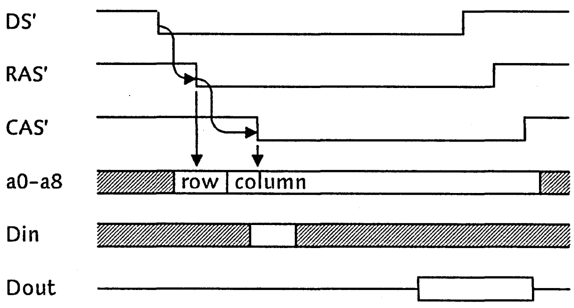


Figure 2.10 DRAM timing.

Error detection and *data line buffers* are made up of four Am29833 9-bit parity bus transceivers (u1-u4) [AMD 85]. The bidirectional tristate buffers need separate output enable signals for each direction. They are obtained through combination of the signals R/W', DBE', and MCS' (u6). The error detection circuit contains a parity generator and checker. The result of the parity checker is latched in an internal flip-flop which is triggered by the trailing edge of the signal DS' at the end of a read cycle (u6c). In case of a parity error, the open-collector output signal PAR.ERR' becomes active. The flip-flop can be reset by the signal PAR.CLR'. PAR.ERR' is connected with the nonmaskable interrupt signal of the CPU.

2.3.3 Display Controller Board

The design of the display refresh controller has mainly been influenced by the use of so-called video RAM devices (VRAM) that have been developed specifically for video applications. The multiport VRAM combines a standard 64K-bit DRAM with an on-chip 256-bit shift register and the necessary controls to transfer data between the memory array and the shift register. The two ports (that is the memory array and the shift register) can be accessed simultaneously except during a data transfer between the memory array and the register.

The display controller board houses 256K bytes of *display memory* and the *display refresh controller.* The display memory is made up of 32 VRAM chips organized with a 32-bit wide data bus as seen from the processor and a 8192-bit wide video data bus as seen from the display refresh controller. 64K-bit 150 ns VRAM chips [TI 83b] are used which are accessed with one additional wait state. The display memory accommodates two bitmaps that can be displayed alternatively.

Display Memory

The organization of the display memory is shown in Figure 2.11 and is very similar to the one of the memory board with the exception of the error detection circuit which has been omitted. The following explanation of the implementation, therefore, concentrates on the differences.

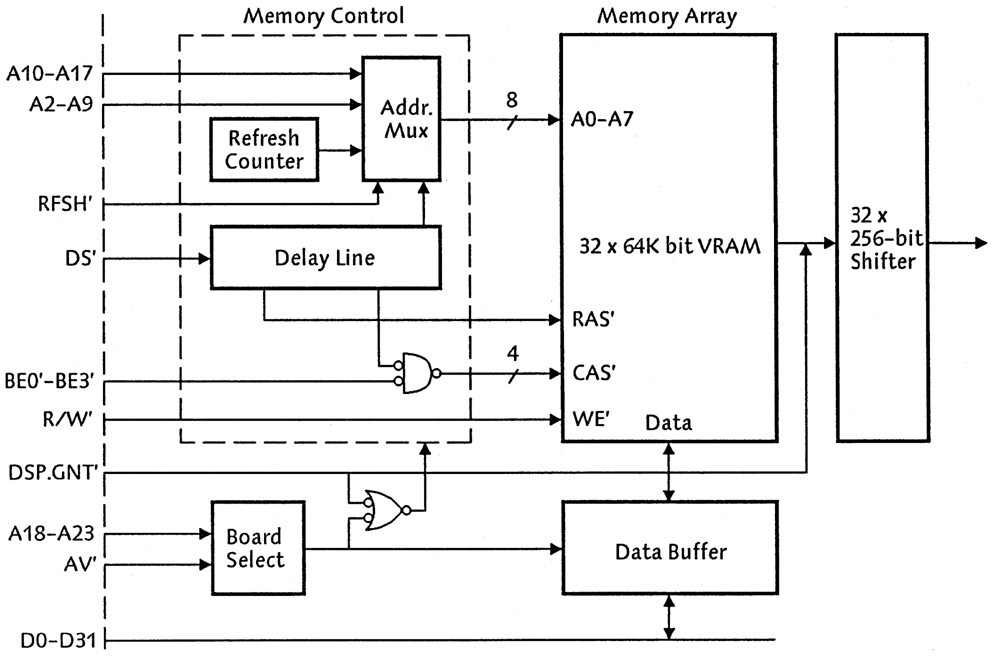


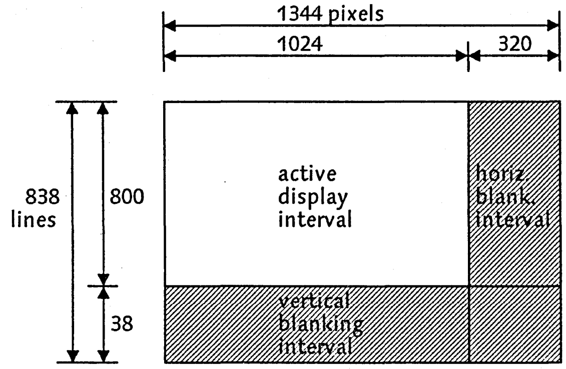
Figure 2.11 The display memory.

The *memory array* contains one bank only. The *board selection* logic is done through a PAL16L8A device (u7). Using address bits A18-A23 the decoder assigns the address range at E00000-E3FFFF (hex) to the display memory. When the display refresh controller accesses the display memory (DSP.GNT'=0) the address decoder is omitted (u11a). As no data are transferred on the memory bus in this case, the data line buffers are not enabled. The *memory control* logic is again realized with a DP8419 DRAM controller (u8). Because of the smaller address range, fewer address lines are needed (A2-A17). The *data line buffers* are made up of four 74ALS645 8-bit bus transceivers (u1-u4).

Display Refresh Controller

In order to better understand the display refresh controller, the display parameters of the high-resolution CRT-monitor are explained first. As can be seen in Figure 2.12, the total frame time consists of the active display interval, the horizontal blanking interval (horizontal retrace), and the vertical blanking interval (vertical retrace). The pixel clock frequency is the product of pixels per line, lines per frame, and frames per second:

$$f(\text{pixel}) = 1344 \cdot 838 \cdot 62.15\ s^{-1} = 70\ \text{MHz}$$



Pixels per line	1344
Lines per frame	838
Displayed pixels per line	1024
Dislpayed lines	800
Frames per second	62.15
Pixel clock frequency	70 MHz
Pixel time	14.3 ns
Data transfer rate	6.4 Mbyte/s

Figure 2.12 The display parameters of the high-resolution CRT-monitor.

The structure of the *display refresh controller* is shown in a further block diagram (Figure 2.13). The following blocks can be distinguished:

- the *clock generator* circuitry provides the clock signals for the video shifter and the horizontal and vertical counters
- the *horizontal and vertical counters* keep track of the position of the displayed picture element (or pixel). Derived from the state of the counters, control signals such as the ones for the synchronization of the display monitor are generated; furthermore, the counters determine the memory array address of those display lines which have to be refreshed next
- the *display memory,* which is arranged as a three-dimensional array of 32 memory devices each being organized as 256 words of 256 bits
- the *video shift register* transforms the data which are transmitted by the VRAMs as 32-bit entities into the bit-serial video data signal

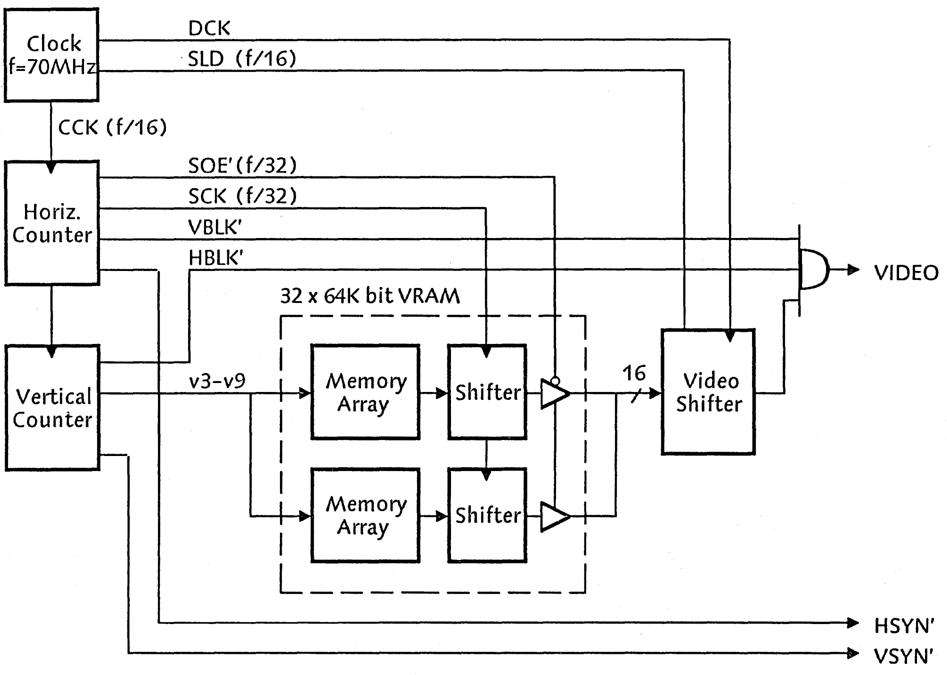


Figure 2.13 The display refresh controller.

Not shown in the block diagram is the *display control register* and the *memory cycle request* circuitry needed to gain access to the "video port".

The *clock generator* circuitry uses a hybrid 70 MHz oscillator chip (u29). Its output provides the pixel or dot clock DCK. A 74AS163 4-bit counter (u27) acts as a divider of the dot clock frequency. It produces the clock signal CCK for the horizontal counter and the load signal SLD for the video shift register. The timing relationship of these signals is shown in Figure 2.14.

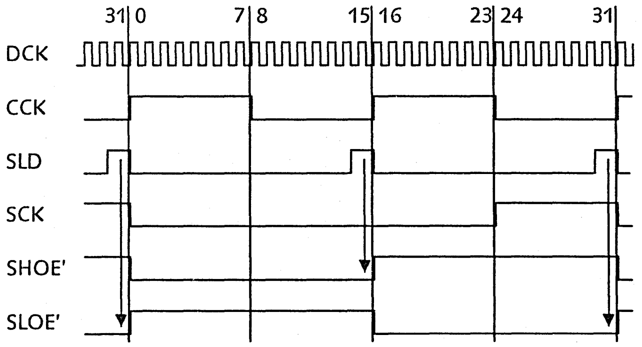


Figure 2.14 Clocking signals of the display refresh controller.

The *horizontal* and *vertical counters* are made up of two, respectively three 74ALS163 4-bit counters (u17, u19, u20, u30, u32), a 27C64 EPROM each (u18, u21), and a 74F378 6-bit latch each (u22, u33). Horizontally, the counter represents the pixel position divided by 16, while the vertical counter state corresponds to the line position. The two EPROMs generate the waveforms of the horizontal and vertical control signals based on the counters state (Appendix A.3). The following signals are provided:

- HBLK' and VBLK' deactivate the video outputs during the horizontal and vertical blanking intervals
- HSYN' and VSYN' are responsible for the line and frame synchronization of the video beam
- HRQ and VRQ cause the VRAM shift register to be reloaded with a new bitmap block every 8 display lines, thus allowing the counter output signals v3-v9 to be used as the bitmap block address
- VCK is the clock signal of the vertical counter
- HCLR' and VCLR' initialize the horizontal and vertical counters to the zero state at the end of a line or a frame, respectively

The horizontal counter also controls the clock signal SCK and the output enable signals SLOE' and SHOE' of the VRAM shift registers. The timing relationship can be seen in Figure 2.14. The 16-bit video shift register is loaded, alternating with the lower and the higher 16 bits of the VRAM shift register data outputs.

The *video shift register* is realized with a 74F676 16-bit shift register (u24) [Philips 84]. Its output, the serialized video data SOUT, and the blanking signal BLK', first have to be synchronized with the dot clock DCK by a 74AS175 quad D-flip-flop (u26), before SOUT can be masked by BLK' (u0). The display control register provides the signal INV which, when set to 1, inverts the video data signal (u0).

In order to access the display memory, the display refresh controller periodically requests a memory cycle from the bus arbiter. The *memory cycle request* flip-flop (u14) asserts the DSP.REQ' line when the horizontal and vertical counters activate MRQ (MRQ = VRQ AND HRQ, u23c) and the display control register bit DSP.EN is set to 1. Another flip-flop is needed to synchronize the request signal with the system clock CLK. A granted memory cycle is indicated by an active DSP.GNT' signal. At the end of a memory cycle, the signal CLR.REQ' clears the request.

DSP.GNT' serves as the output enable of two 74ALS541 octal buffers (u5, u6) that gate the address and control signals to the memory bus. The address is made up of the vertical counter outputs v3-v9 and the display control register output a17. Signals v3-v9 define the display line that has to be scanned next; a17 determines in which half of the display memory the displayed bitmap is located. This address information is directed to the address lines A10-A17 which define the memory row of each VRAM that is to be loaded into the internal shift register. If the two address bits A8 and A9 equal 00 during this register transfer cycle, a total of 256 bits can be subsequently read out (it is possible to transfer 64, 128, 192, or 256 bits of a memory row into the shift register). All other address bits are neglected (A0-A7, A18-A23). As address decoding now has to be inhibited (AV'=1), the display memory also has to be selected (DCS') when DSP.GNT' is active (u11a).

The signal T'/OE', which is input to the VRAMs, has two functions that are shown in Figure 2.15. First, it selects either shift register transfer or random-access operation when RAS' falls; therefore, during a memory access of the display refresh controller, T'/OE' equals DSP.GNT', which is already low as RAS' falls. Second, if a random-access operation is performed, it functions as an output enable after CAS' falls. For this reason, it can then be identical to CAS' (u11b).

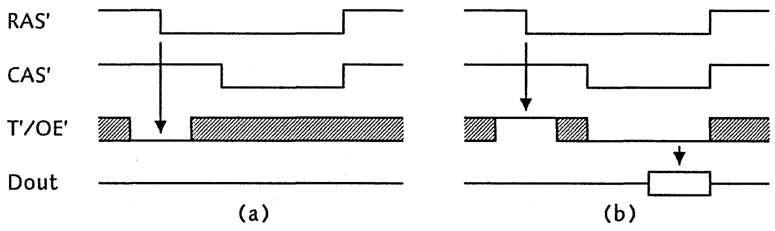


Figure 2.15 VRAM shift register transfer (a) and random access operation (b).

The *display control register* is implemented with a 74ALS175 quad D-flip-flop (u12). The flip-flops are reset by a RESET' pulse. The address decoding is performed by half a PAL16L8A (u7). The register is located at FFFA00 (hex). The meaning of the three write-only bits is as follows:

Bit 0	0	Display Enable (initialized value)
	1	Display Disable
Bit 1	0	A17=0 (initialized value)
	1	A17=1
Bit 2	0	Normal Video (initialized value)
	1	Inverse Video

DSP.EN set to 0 (bit 0 of the control register) prevents any display requests. Nevertheless, the display is refreshed with the contents of the VRAM shifters which will no longer be loaded. The shifter input signal SI now defines the video data signal. In order to guarantee a blank screen, SI is connected to INV (bit 2 of the control register). In the inverse mode SI is set to 1 in order to get an inverted video data signal of 0.

The design of a display refresh controller with a pixel frequency of 70 MHz requires special care. At a clock period of 14.3 ns, gate delays of 5 ns are of considerable significance. The following provisions have been made:

- all registers generating critical signals are clocked by the same clock signal (u22, u33, u24, u26)
- synchronizers adjust different signal delays (u25, u26)
- all paths in a combinatorial circuit are of the same length (u0)

2.3.4 Disk Controller Board

The Western Digital WD1002-05 disk controller board [WD 83] contains a Winchester interface (Seagate ST506 compatible) and a floppy interface (Shugart SA450 compatible). The controller holds all of the logic required for a variable sector length (up to 1K bytes), ECC correction, data separation, and host interface circuitry. The latter consists mainly of an 8-bit bidirectional parallel bus and appropriate control signals. Programmed IO is used to transfer sector data to and from an on-board sector buffer. Except for the board select signal DK.CS' and the interrupt request signal DK.INT, all signals of the host interface are "standard" memory bus signals. Additional circuitry for the signals DK.CS' (u57) and DK.INT (u63b) resides on the processor board.

2.3.5 Motherboard

Physical extensibility is obtained by placing the circuitry on several boards that are connected by a backplane (motherboard). The Ceres motherboard offers slots for six boards connected with a common, parallel backplane bus. Three slots are occupied by the already explained standard boards. Packaging flexibility is provided by requiring that the physical card position on the motherboard has no effect on the functioning of the system. This is accomplished by avoiding the use of daisy chain signals, which would require that there be no empty slots between boards and by having all signals independent of the backplane position. To avoid floating values, pullup resistors are provided for the address and data signals. The backplane bus contains the following lines:

Address	A0-A23
Data	D0-D31
Control	
Data Transfer	AV', BE0'-BE3', DS', DBE', R/W'
Bus Arbitration	REQ0'-REQ3', GNT0'-GNT3', DSP.REQ', DSP.GNT', CLR.REQ'
Cycle Extension	CWAIT', WAIT1', WAIT2'
IO Devices	IO.EN', IO.RD', IO.WR', DK.CS', DK.INT
Interrupts	INT4'-INT7'
Clock	CLK, FCLK
Miscellaneous	RESET', RESET.IN', RDY, ILO', PAR.ERR', PAR.CLR', RFSH'

2.4 Hardware Extensions

The modular, extensible multiboard arrangement invites not only the addition of more memory but, in particular, hardware which extends the versatility of the Ceres computer. The memory bus provides all necessary signals to either add new bus master or bus slave devices. In Figure 2.16, bus interfaces for both types are proposed. Note that the slave must be "synchronous" in the sense that it is always available and does not provide a completion signal.

The timing specification of a basic memory cycle is presented in Figure 2.17. Based on a processor clock period of 100 ns, the unit of value is the nanosecond. For a peripheral cycle, four wait states are inserted between T2 and T3.

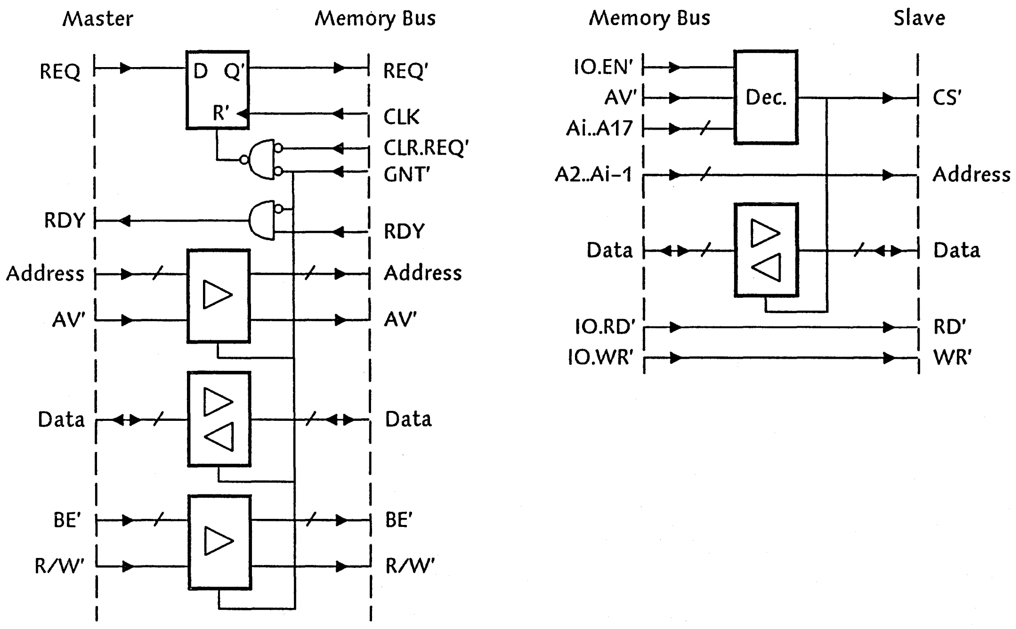


Figure 2.16 Interfaces for a bus master and a bus slave.

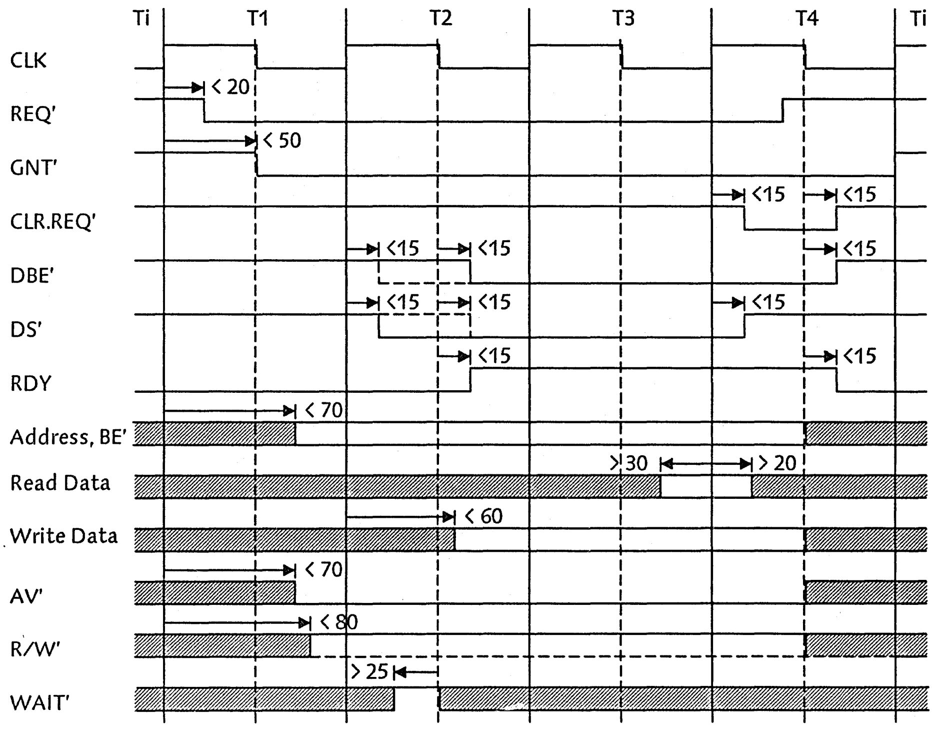


Figure 2.17 Timing specifications of a basic memory cycle.

3 Raster Graphics Interface Design

3.1 Introduction

In early computers, limited storage capacity led to a display output of little flexibility. A dense encoding of display information allowed a restricted set of symbols mostly consisting of alphanumeric characters. Progress in technology has removed these restrictions: cheap and large semiconductor memories can be used to store high-resolution display images in an unencoded form, thus providing maximum flexibility. As a consequence, large amounts of memory data have to be accessed in order to create the image and to refresh the display monitor. While the concern of earlier designs was to reduce the required storage capacity, recent designs are concerned with how to provide enough processing performance to manipulate and refresh video data.

As technological constraints of storage and processing performance become less severe and various VLSI-solutions for graphics applications are offered, the designer of a raster graphics system has to give thought to what the software designer really needs and to concentrate on clean and clear concepts. Hardware and software have to be designed as an entity in order to complement each other.

The graphics system of Ceres is a typical example of a simple and straightforward design. In contrast to many other workstations, the graphics functions of Ceres are completely software-based, i.e. no hardware assistance for image manipulations is provided. A detailed description of the display controller board is contained in Section 2.3.3. This chapter adds some design considerations that are in particular applicable to raster graphics with mostly stationary objects as found in document processing, for instance. The design of the raster graphics interface for Ceres is aimed at using only available integrated circuits and at economizing on the hardware expense.

The first part of this chapter outlines raster graphics principles and shows some possible realizations. The explanations are made with respect to the design of the raster graphics interface for Ceres which is discussed in the second part.

3.2 Raster Graphics Principles

A typical raster graphics system has the organization of the block diagram in Figure 3.1. The image is stored in a memory, called the *frame buffer*, as a matrix of intensity values, known as the *pixmap*, where each unit of storage corresponds to a picture element, or a *pixel*. In the simplest case of a monochrome image each pixel can be represented by a single bit: the pixmap is then called *bitmap*. Multiple bits per pixel are required to display greyscale or colour images. The subsequent discussion concentrates on monochrome raster graphics only. The frame buffer is accessed by the *display processor* in order to manipulate the image data, and by the *display refresh controller* in order to route the image data to the *raster-scan display*. Raster scanning implies that the image is scanned onto the display screen surface in a raster sequence as a fixed succession of scan lines, made up of pixels.

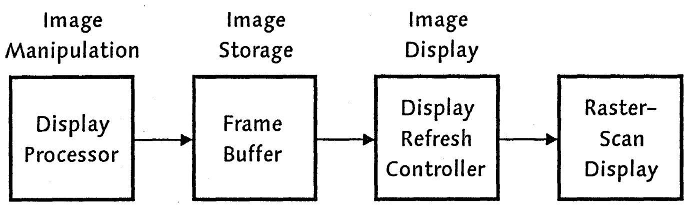


Figure 3.1 Structure of a raster graphics system.

Significant architectural characteristics of a display system are:

- the *update performance*, i.e. the time required to generate a new image in response to a user request, and
- the *refresh performance* that determines the resolution and stability of the displayed image.

Both the update and refresh process compete for the frame buffer as a shared resource. Because the implementation of the frame buffer normally does not allow for simultaneous access, the display processor and the display refresh controller compete for a finite number of available memory cycles. Since the display refresh controller has to supply the video data to the raster display according to strict timing requirements, it always goes ahead of the display processor, which has to put up with the remaining memory cycles. Note that the bandwidth requirements of the display refresh controller can be precisely quantified, while the bandwidth requirements of the display processor vary significantly with each application.

As cheaper and larger semiconductor memories have become available, higher resolution bitmaps can be stored. However, the display processor and the display refresh controller have to access the frame buffer more frequently, making the contention problem even worse. The main concerns of designing a raster graphics system are to increase the bandwidth of the port to the shared frame buffer, to speed up the operations for manipulating bitmap data, and to reduce memory access conflicts caused by the display refresh controller.

3.2.1 Image Storage

The development of image storage technology can best be illustrated by studying representative workstations such as the Alto, Lilith, and Ceres.

The Xerox Alto was one of the first personal computers that incorporated a raster graphics display [Thacker 79]. The 608 x 808 bitmap is displayed with a refresh rate of 30 frames/s. The bandwidth required by the serial video data comes to 20 MHz. The frame buffer is single-ported and can reside anywhere in memory. The unique aspect is that the display refresh action is programmed in microcode and executed by the CPU. During the active line interval the CPU fetches double-words in 1.05 μs, and the 32 bits are displayed in 1.6 μs. A full screen bitmap occupies about half of the main memory of 128K bytes and displaying it consumes about 48.36% of all available memory cycles. In the lack of a large memory the displayed bitmap is pieced together by a list of smaller bitmaps, so that white spaces do not consume any memory space.

Raster graphics technology was further refined for the design of the Lilith computer [Ohran 84, Wirth 81b]. The bitmap has an increased resolution of 704 x 928 pixels, displayed with a refresh rate of 30 frames/s resulting in a video data bandwidth of 27 MHz. In contrast to the Alto, the display refresh of Lilith is performed by a separate hardware unit. A quad-word is fetched in 0.375 μs, and the 64 bits are displayed in 2.0 μs. The displayed bitmap occupies a fourth of the main memory of 256K bytes and refreshing the display consumes 11.48% of all memory cycles.

The raster graphics interface for the Ceres computer has been developed even further. The 1024 x 800 bitmap is now refreshed at a rate of 62.15 frames/s. The video data bandwidth reaches 70 MHz. The display refresh controller accesses 8192 bits in 0.400 μs, and these are displayed in 154 μs. If the display refresh controller could only access 32 bits in 0.400 μs, then all memory cycles would be absorbed because the 32 bits would be displayed in 0.457 μs only. The displayed bitmap occupies about a twentieth of the main memory of 2M bytes and refreshing the display consumes as little as 0.25% of all memory cycles.

	Resolution [pixels]	Bitmap Size [K bytes]	Refresh Rate [frames/s]	Video Bandwidth [MHz]	Memory Cycles [%]
Alto 1973	608 x 808	60	30	20	48.36
Lilith 1978	704 x 928	80	30	27	11.48
Ceres 1986	1024 x 800	100	62.15	70	0.25

Figure 3.2 Examples of raster graphics interfaces.

The technological trends can easily be seen in Figure 3.2. The resolution of raster displays is steadily increasing; likewise the stability of the displayed image is improved by higher refresh rates. Although the frame buffer has to supply a significantly larger amount of video data, the table shows that techniques have been found to reduce drastically the memory cycles required for refreshing the display.

Different techniques have been developed for the implementation of frame buffers. Some typical architectural concepts shall be discussed in this section. A tutorial in memory design for raster graphics displays is contained in [Whitton 84]. Reference to [Baecker 79] can be made for a history of early frame buffer devices.

With available dynamic RAMs (DRAM), it is not possible to realize a true multi-port frame buffer that can be accessed simultaneously by the display processor and the display refresh controller; they both have to use the same port at different times. To suit the update and refresh process, techniques have to be found that equip the frame buffer with a considerably higher access bandwidth than required for accessing normal main memory.

A common method is to widen the data bus for accesses of the display refresh controller. Figure 3.3 illustrates a *wide-word* memory architecture: while the display processor has an n-bit wide access path to the frame buffer, the display refresh controller can access m bits in one memory cycle, whereby m is an integral multiple of n; typical values are n = 16 and m = 64. The implementation can be facilitated by using wide-word memory devices. Typical organizations of such DRAM units are 16K x 4 bits or 64K x 4 bits allowing a reduction in part count, savings in power, and improved reliability. The disadvantage of this design is the expense due to the m-bit wide multiplexer and shift register. Furthermore, electrical problems are likely to arise due to the expensive interconnection structure: an m-bit wide bus connects the memory devices, multiplexer, and shift register. This layout was used for several computers, such as the Xerox Dorado [Lampson 80] and the Lilith [Ohran 84].

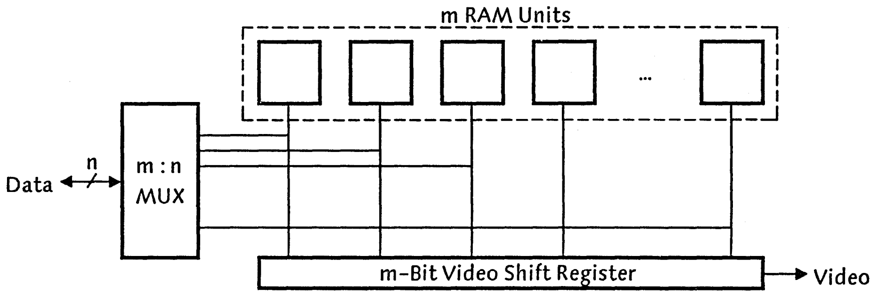


Figure 3.3 A wide-word memory organization of a frame buffer.

However, the application of this method is limited. Figure 3.4 lists memory cycle requirements of the display refresh controller during the active display interval as a function of the number of pixels that are fetched per memory access. Assumed display parameters are a resolution of 1024 x 800 pixels and a refresh rate of 30 Hz (interlaced) and 60 Hz (non-interlaced), respectively. The memory cycle time is assumed to be 400 ns. Especially the figures shown for the non-interlaced mode demand alternative methods.

	1024 x 800, 30 Hz 25 ns Pixel Time		1024 x 800, 60 Hz 12.5 ns Pixel Time	
Pixels per DRC Access	Time between DRC Accesses [ns]	Memory Cycles for DRC [%]	Time between DRC Accesses [ns]	Memory Cycles for DRC [%]
16	400	100	200	cannot be supported
32	800	50	400	100
64	1600	25	800	50

Figure 3.4 Memory cycle requirements of the display refresh controller (DRC) for different frame buffer access widths.

Another method to meet the high bandwidth requirements is to use a special, fast access mode as provided by most DRAMs. The modes are called page mode, nibble mode, and ripple mode. They allow multiple bits of data to be sequentially read or written within an extended cycle and are applicable, if data are accessed that are located at successive addresses. A detailed analysis of using *burst modes* for the realization of a frame buffer is contained in [Whitton 84]. However, the achieved reduction of memory cycles required for refreshing the display is comparable with a wide-word memory architecture.

Recognizing the need for significantly higher data transfer rates chip designers developed a DRAM unit with an internal shift register [Pinkham 83]. A block diagram of this memory device is shown in Figure 3.5.

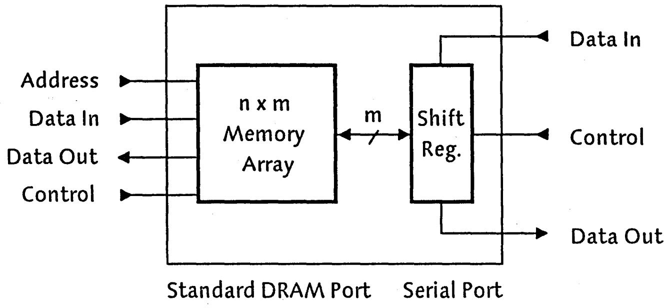


Figure 3.5 The organization of a VRAM.

The device, also known as *video RAM* (VRAM), contains a standard dynamic memory internally organized as a matrix of n x m memory cells and an m-bit wide shift register. Both ports, the standard DRAM port and the serial port can be accessed simultaneously. The contents of a complete memory row (m bits) can be transferred between the memory array and the shift register in one memory cycle. During that time it is not possible to access any port of the VRAM. As Figure 3.6 shows, a typical frame buffer contains n N x 1 bit VRAMs, whereby n is determined by the width of the bus connecting the display processor and the frame buffer. The serial output ports of the n memory devices feed in parallel to an external n-bit video shift register. With a maximum clock rate f at which the shift register of the VRAM can be operated, video data rates of up to n · f can be achieved with n VRAMs. Current VRAM implementations specify a clock rate f of 25 MHz.

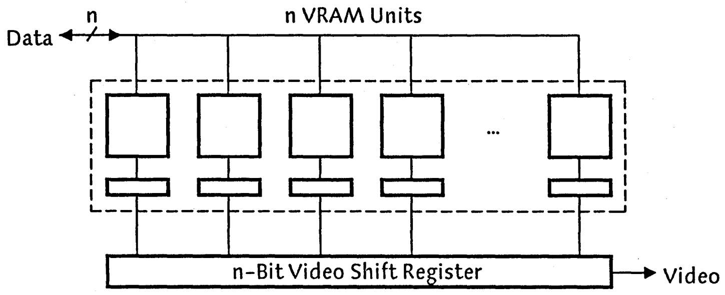


Figure 3.6 Frame buffer organization based on VRAMs.

VRAM technology appears to be ideal for frame buffer design. The video bandwidth is no longer limited by accessing the frame buffer. Furthermore, no significant delay of display processor operations is caused by interfering frame buffer accesses of the display refresh controller.

3.2.2 Image Creation

The most important bitmap operator is *BitBlt*, which stands for bit block transfer as designed by Ingalls in 1975 [Ingalls 81]. A detailed discussion is contained in [Newman 79] where it is called *RasterOp*, short for raster operator. RasterOp works on rectangular regions within a bitmap. The operator takes two rectangles, called the source s and the destination d and modifies d using values of s:

$$d \leftarrow F(d,s)$$

Of the sixteen possible Boolean functions, four appear to be useful for monochrome image manipulations: F(d,s) typically is s (replace), d OR s (paint), d XOR s (invert), or d AND NOT s (erase). The importance of RasterOp and its generality in application have been discussed extensively in [Newman 79, Ingalls 81, Gutknecht 83]. Graphics primitives are either implemented in software and executed by the CPU, or assisted by special hardware.

A software-based implementation is simple, flexible, and inexpensive. However, executing RasterOp on a general purpose microprocessor is expected to be slow. An improvement can be obtained by providing RasterOp as a single machine instruction, as it is possible with a microprogrammable processor. This is advantageous because the instruction sequence, which is repetitively executed by a processor with a fixed instruction set, can be encoded in a single instruction thus reducing the time needed to fetch and decode the instructions and because lower level optimizations are possible. A nice example of a purely software-based implementation for an MC68000 CPU is the AT&T graphics terminal Blit [Pike 85]. Microcoded RasterOp can be found in the Xerox Alto and the Lilith computers.

Various hardware assistance for raster graphics has been proposed, ranging from devices that cooperate with a general purpose microprocessor in manipulating single words, up to display processors that execute high-level graphics functions. Depending on the kind of hardware assistance, there are differences for the CPU in accessing the frame buffer, as shown in Figure 3.7. In an *integral* frame buffer design, the frame buffer is an integral part of the CPU's memory address space. In a *peripheral* frame buffer design, the frame buffer is not directly accessible from the CPU, but is controlled instead by a display or graphics processor.

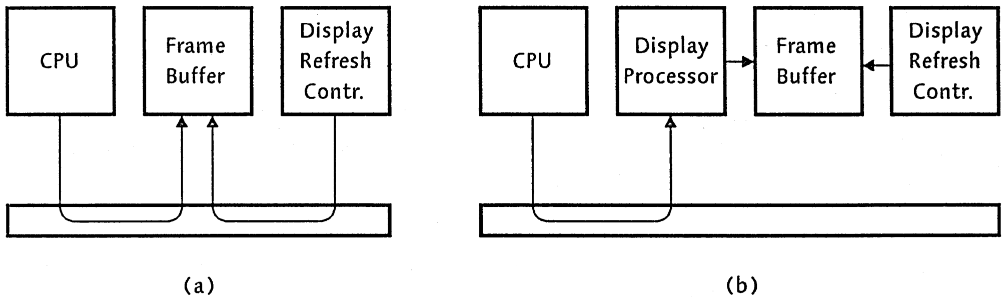


Figure 3.7 An integral frame buffer (a) and a peripheral frame buffer (b).

A modest solution is to add hardware assistance for shifting, masking, and Boolean operations as it is shown in Figure 3.8. The processor's duty is then reduced to simply move data from one memory area to another. By executing a read-modify-write memory cycle a destination word obtains a new value given by the logic operation between the destination word (D_{out}) and the source word (D_{in}). A barrel shifter is provided in order to align the source words with the destination words. A mask register is useful if the destination words have to be modified only partially.

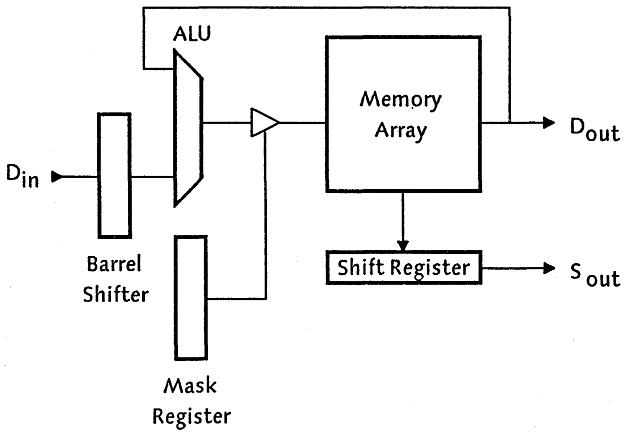


Figure 3.8 Frame buffer with hardware assistance for RasterOp.

While these provisions could be easily implemented with standard components, an even less expensive solution is provided by a special VRAM. The HM53462 from Hitachi already contains an ALU and a mask register [Hitachi 1986]. The device offers the ability to internally interpret a write cycle as a read-modify-write cycle modifying the data according to a specified ALU-operation. Similar concepts have been followed up in the design of VLSI-circuits such as the MergeOp Unit described in [Kronfeld 85] or the RALU (RasterOp ALU) from VTI [VTI]. In addition to an ALU and a mask register, these devices also contain a barrel shifter. Unfortunately, both devices are only 16-bit wide and are not cascadable.

The CPU can be completely freed from image manipulations by providing a separate display processor, which is equipped with a dedicated instruction set that allows to efficiently execute high-level graphics functions. Furthermore, the normally required peripheral frame buffer design allows the display processor and the CPU to work in parallel. Display processors are welcome VLSI-applications and are or will become available from most larger semiconductor manufacturers. A collection of descriptions of current display processors can be found in [CG&A 86]. All these devices show similar architectures that not only contain the display processor itself, but also a DRAM controller/driver, and a display refresh controller.

A disadvantage associated with display processors is their lacking flexibility due to a predefined set of instructions. It will be difficult, or even impossible, to extend the display processor's firmware. This limits the exploration of novel raster operations, e.g. drawing of spline curves or filling of bitmap areas.

A severe problem of most hardware assisted raster graphics systems is that RasterOp cannot be uniformly applied to both the bitmap memory and the general memory. Bitmaps have to reside in a dictated memory area that in the case of a peripheral frame buffer is not even transparent for the CPU. However, modern raster graphics software requires non-visible bitmaps of which there can be a large number and which have to be processed as efficiently as the displayed bitmap [Pike 83]. Note that every non-uniformity of the address space complicates the software.

Hardware support for raster graphics is best illustrated by the SUN workstation family. The SUN1 provides a two-dimensional peripheral frame buffer and hardware assistance of the type shown in Figure 3.8 on a separate Multibus card [Bechtolsheim 80]. The CPU communicates with the graphics hardware through a number of registers. One drawback of this implementation is that the 32-bit CPU has to access the frame buffer through a 16-bit interface. A general RasterOp implementation is expensive because it must deal with four cases depending on the location of the bitmap: frame buffer to frame buffer, frame buffer to general memory, general memory to general memory, and general memory to frame buffer. The SUN2 has an integral frame buffer, directly addressable by the CPU. Optional hardware assistance is provided. RasterOp is therefore reduced to two cases of frame buffer to frame buffer and all else.

3.3 Raster Graphics Interface for Ceres

The graphics hardware for Ceres is characterized by the premise to retain as much flexibility as possible. This excludes the use of a separate, special purpose display processor. The implementation contains an integral frame buffer based on VRAMs and a display refresh controller. They serve purposes where little or no flexibility is required.

The raster graphics hardware of Ceres is an example of the simple and straightforward design applied to this workstation. Graphics hardware and software were designed together. It was postulated that the graphics functions would be based on RasterOp. Before the hardware was implemented, RasterOp was written in assembly language in order to evaluate its performance [Wanner 84]. Only slightly slower execution times have been estimated for RasterOp on Ceres compared with the microcoded versions on Lilith. Therefore, for the basic configuration of Ceres, hardware assistance could be left out of consideration. The final implementation of RasterOp on Ceres is reported in [Peschel 87].

In addition to the detailed description of the frame buffer and display refresh controller in Section 2.3.3, the following discussion contains an evaluation of the raster graphics hardware developed for Ceres.

3.3.1 Frame Buffer

The frame buffer of Ceres is one of the early applications of VRAM technology. The prototype was implemented with samples of VRAM devices that were provided by Texas Instruments. The used memory device type has the part number TMS4161 and contains a 64K x 1 bit DRAM and a 256-bit wide shift register [TI 83b, Pinkham 83]. A block diagram of the frame buffer is shown in Figure 2.11 of Section 2.3.3. The merits of VRAM technology have already been discussed in Section 3.2.1. The display refresh process consumes as little as 0.25% of all available memory cycles. Thus, the image update process has nearly unrestricted access to the integral frame buffer, i.e. the CPU is no longer blocked by frame buffer accesses of the display refresh controller. There is no need for a local frame buffer bus in order that memory accesses of the display refresh controller are uncoupled from the global memory bus as it is one of the motivations of choosing a peripheral frame buffer design.

The forms of hardware assistance that help the image update process most are found in increasing the availability of the memory or speeding up the image update processor itself. In this respect building the frame buffer for Ceres out of VRAM devices is an improvement upon earlier implementations such as the Lilith and the Alto, where any CPU activity including the image update process could use 88.52% and 51.64%, respectively, of all available memory bus cycles.

The use of VRAMs requires that the displayed bitmap has to reside in a reserved memory area. The location of the displayed bitmap is further restricted in that it was laid down that its base address had to be fixed although the available capacity of the frame buffer memory would have allowed a relocation within certain bounds. This restriction certainly is of no importance, but simplifies the implementation of the display refresh controller.

Whereas main memory is equipped with a parity error detection circuit so that single bit errors within a data byte can be detected, a similar provision for the VRAM memory has been omitted. The reason is that only bitmap data are assumed to reside in the VRAM memory and with that memory errors only affect visual output but not program flow.

The VRAM organization and the width of the memory data bus necessitate 32 memory chips resulting in a capacity of 256K bytes, whereas with the given display resolution 100K bytes of VRAM would have been sufficient. However, the available video memory capacity can be used for a technique called *double-buffering*: the displayed bitmap can reside either in the lower or the higher half of the 256K byte-sized video memory. A typical application of double-buffering is the continuously visible movement of graphical objects as required for interactive positioning [Kohen 85]. With double-buffering, image updates called for by the moving object are not executed in the displayed bitmap. Instead a new image is prepared in a background bitmap and when finished, the display refresh controller is switched to it. In this way, the display is prevented from flickering because the displayed object is never temporarily blanked. This is an improvement upon other methods that use one display bitmap only and first delete an object before it is redrawn at a new position.

3.3.2 Display Refresh Controller

A block diagram of the display refresh controller is shown in Figure 2.13 of Section 2.3.3. The display refresh controller is based on the principle of a synchronous sequential circuit of the Mealy-type. A sequential circuit consists of a state register and a combinatorial circuit. The state register of the display refresh controller is realized with several cascaded, synchronous 4-bit counters. The combinatorial circuit represents the state transition function and is realized with ROMs. (More precisely, part of the transition function is already provided by the counter devices.) The output function of the sequential circuit is responsible for the synchronisation of the video beam with the video data signal and for loading the video shift register with the frame buffer data. The states are coded in a way that they represent the actual horizontal and vertical position of the displayed pixel. Thus, the counter outputs can be directly used to address the frame buffer for loading the video shift register.

The design of the display refresh controller is guided by the need for flexibility. Therefore, the transition function is not hardwired. Instead erasable ROMs are used. Without any hardware modifications, the display parameters such as the resolution of the display monitor can be altered. This proved to be valuable during the period of development.

A careful design has allowed to use standard TTL devices. The most critical device is the video shift register which is clocked at 70 MHz. The video shift register is loaded with the output data of the shift registers built into the VRAMs. The output enable line of the VRAM's serial port offers the possibility to reduce the size of the video shift register and to time-multiplex data to it. In the present design the outputs of 32 VRAMs are input to a 16-bit shift register. Actually, even an 8-bit shift register would have been sufficient because the resulting 8.75 MHz data rate, at which data would be loaded into the shift register, is still far below the 25 MHz maximum frequency of the VRAM shift clock.

The design employed allows to load the VRAM shift registers during the blanking interval only. If all accessed data are to be displayed, the line length has to be an integral divisor of the total number of bits that are loaded into the VRAMs' shift registers during one memory cycle. With the given parameters, the line length l is: $l = 2^n$, $16 \leq l \leq 8192$. This restriction allows an efficient realization of the display refresh controller and the frame buffer. If a line length is specified that does not fulfill the above criteria, the design would have to be extended in that the frame buffer would consist of two VRAM banks that could be alternately accessed by the display refresh controller: while video data would be accessed from the serial ports of one bank, the shift registers of the other bank could be loaded.

As will be shown in Section 4.3.2, the arbitration of the shared memory bus is optimized for short response times. Memory requests of the display refresh controller have to be processed during the horizontal retrace of the video beam. As specified in Section 2.3.3, the horizontal retrace time and with it the latency time, i.e. the time permitted until a memory request has to be granted, corresponds to 320 pixels or 4.5 μs. It can easily be met by an arbiter that re-allocates the shared memory for every memory cycle. Note that significant complications would be caused if non-interruptable bus transfer sequences of variable length would be allowed.

3.3.3 Why no Hardware Support for RasterOp?

Afraid of being blamed for what has not been provided, the hardware engineer tries to add most, if not all, possible features to his design, thus losing a simple and regular concept. Irregularities are introduced by having a frame buffer organization different from other main memory or by adding hardware support for raster graphics that can only be applied to a reserved memory area. This leads to more complicated and less reliable raster graphics software. A raster graphics system, therefore, has to be based on a *uniform* memory and processor architecture. Two structures for hardware to support RasterOp-style raster graphics can be accepted: either provide a simple, uniform memory structure and let software do the rest, or if utmost speed for a particular application is required, provide hardware support in a way that enables general software to be written.

The need for a regularly designed raster graphics system shall be illustrated with two typical applications of *off-screen bitmaps*. Windowing is a well-established technique that allows to have several virtual screens assigned to different objects or tasks. Off-screen bitmaps may be used to hold non-visible windows partly or entirely, as it may occur when windows are allowed to overlap [Pike 83]. Off-screen bitmaps may also be valuable for storing image parts that can be repeatedly used for creating an image. The time required to create an image can then be shortened if these are drawn once off-screen and then copied to the screen bitmap for each appearance. Applications are temporarily displayed command menus of interactive programs or macros of a graphics editor. Depending on the complexity of the image, this technique can easily outperform a peripheral frame buffer design with a dedicated display processor that has to redraw such image parts in full for each update.

The memory space required for off-screen bitmaps can be considerably large and cannot be fixed. An economic implementation of a peripheral frame buffer provides a local memory capacity that typically is only a fraction of the main memory capacity. If the peripheral frame buffer cannot hold all off-screen bitmaps, bitmap manipulations get complicated and inefficient. The programmer, however, expects that off-screen bitmaps can be kept anywhere in main memory and that the efficiency of RasterOp does not depend on where in memory the bitmap is located. The demand for a uniform memory is even more evident if virtual memory is provided. Nearly any number of off-screen bitmaps could reside in a uniform virtual memory space, an off-screen bitmap neither had to be resident in the physical memory nor did a resident off-screen bitmap require a physically continuous memory space.

In contrast to many other workstations the structure of Ceres complies with the explained requirement of a uniform memory. It is to be admitted that the displayed bitmap has to reside in a reserved memory area. This has to be considered when allocating bitmaps. However, the salient point is that RasterOp can be uniformly applied to any bitmap independent of its memory location. The strength of this concept can be illustrated best with the minimal effort that was required for the addition of a laser printer interface. The structure of the interface is comparable with the one of the display refresh controller. The bitmap to be printed is allocated in normal main memory, i.e. no local VRAM frame buffer is required, because the video data rate of the laser beam printer is slow compared with the one of the display monitor. As bitmaps can reside anywhere in memory, standard RasterOp functions are used to create the printer bitmap. Thus, printer and screen bitmaps are treated identically.

While the chosen concept is clear and simple, its realization has to prove that the performance is sufficient. Figure 3.9 gives measured times for some bitmap operations on Ceres, Lilith, SUN2, and SUN3. The numbers measured for Ceres are certainly acceptable particularly for graphics applications as found in document processing.

There are, of course, sophisticated graphics applications that demand for utmost speed. However, it has been shown that even a general purpose microprocessor can supply a considerable graphics performance. Thanks to the generality of the uniform memory and processor architecture, minimal costs have been required for the development of the raster graphics system for Ceres. As the performance of microprocessors will further increase, the simplicity of the presented design will attract other applications making hardware support, e.g. in form of a display processor, superfluous.

	Block Transfer Aligned 512 x 512 [ms]	Block Transfer Dealigned 512 x 512 [ms]	Scroll Vertically [ms]	Display 12-point Character [ms]	Display 24-point Character [ms]
Ceres NS32032-10	114	132	84	0.23	0.32
Lilith Am2901	71	73	168	0.13	0.27
SUN2 MC68010-10	81	190	170	0.60	0.77
SUN3 MC68020-16.7	18	33	37	0.24	0.27

Figure 3.9 Bitmap operations on Ceres, Lilith, SUN2, and SUN3.

4 Microcomputer Bus Design

4.1 Introduction

A digital system is a collection of elements that can process and store information. By connecting these elements with communication paths, so-called buses, a higher-level system is composed. Depending on the level of the system hierarchy, several buses are found: an integrated circuit such as a microprocessor contains internal buses that connect registers and arithmetic logic units; other pathways are used to transfer data between the processor and the memory of a computer; computers again are connected by buses that can even span long distances.

Performance and cost of a computer system are decisively influenced by its bus structure. The advance of semiconductor technology continuously optimizes the performance/cost ratio of computer components. Therefore, the bandwidth and the interconnection cost of a bus system become increasingly significant. As an example, Intel Corporation's 8008 8-bit microprocessor, which was introduced in 1972, had an average instruction execution time of 30 μs, while todays 32-bit microprocessors typically require less than 1μs for the execution of an instruction. In order to satisfy the microprocessor's hunger for data, high-performance buses are required. Interconnection structures have therefore to be recognized as an important topic of computer architecture.

The first part of this chapter gives a classification of bus parameters that can be taken as a basis for bus design. An illustration is presented with the second part that contains a discussion of the design and implementation of the Ceres bus structure. Ceres is a single-processor microcomputer in the sense that memory and IO devices are placed around one single microprocessor. Other processing elements may exist, but are dedicated to a specific function and are not user-programmable. In the following they are called controllers.

4.2 Classification Criteria

For a long time the importance of bus design has been underestimated, which is also reflected by the existence of few comprehensive publications. A classical paper was written by Thurber et al. which proposes "A systematic approach to the design of digital bussing structures" [Thurber 72]. Although it was the first paper that exclusively handled this subject, its classification is still valid and shall therefore underlie this section. [Levy 78] contains a practical discussion of bus design based on the history of the PDP-11 family. [Corso 86] is the most recent publication. It is a whole book that treats "Microcomputer Buses and Links". More general introductions can also be found in [Hayes 79], [Gustavson 84], and [Färber 84].

First of all the term "bus" has to be defined. A *bus* is an interconnection structure between n participants (n >= 2) of which s are able to act as the *source* (1 <= s <= n) and d as the *destination* (1 <= d <= n) during a bus cycle. A *bus cycle* spans the time taken to transfer an elementary item of information. The participant that is able to initiate a bus cycle is known as the *master*, while the responding participant is called *slave*. At the same time only one source (i = 1) can send information to j destinations (1 <= j <= d). Note that with the given definition a point-to-point link is the minimum configuration of a bus (n = 2, s = 2 and d = 2 for bidirectional communication, s = 1 and d = 1 for unidirectional communication).

As Figure 4.1 shows, two types of bus cycles can be defined: during a write cycle the master acts as the source, while the slave is the destination of the information transfer; during a read cycle the slave acts as the source, while the master is the destination.

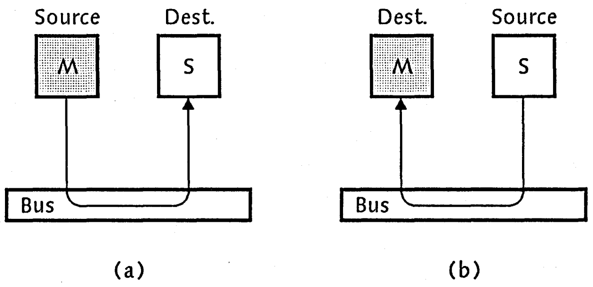


Figure 4.1 A write (a) and a read (b) bus cycle.

Figure 4.2 summarizes the terminology used for the following classification of bus parameters.

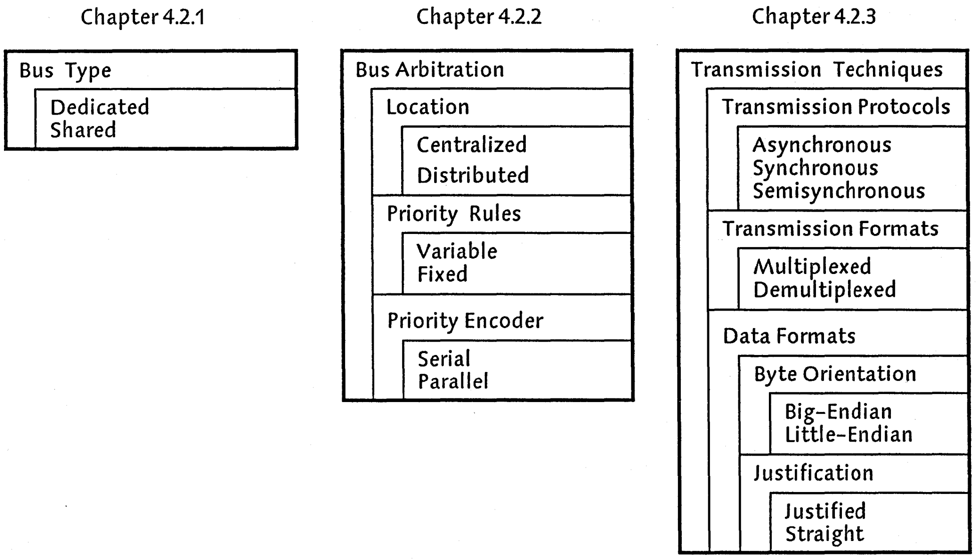


Figure 4.2 Classification Criteria.

4.2.1 Bus Topology

In this section the interconnection structures, i.e. the type and number of buses of a computer system, are considered. The buses can be separated into two types: *dedicated* and non-dedicated or *shared*. [Thurber 72] defines a dedicated bus as permanently assigned to either one function or one physical pair of devices, whereas a non-dedicated bus is shared by multiple functions and/or devices. Note that the given definitions introduce a further distinction between functional and physical interconnection structures.

The following example illustrates the proposed classification. The Unibus originates from Digital Equipment Corporation and was designed for the PDP-11 computer family [Levy 78]. It is a single bus to which all system components, i.e. the processor, several controllers, main memory, and IO devices are attached. The bus is physically shared by the processor and the controllers which are all capable of initiating data transactions. The Unibus contains separate address, data, and control lines. The address and data bus can therefore be seen as functionally dedicated. However, this description is not complete. The data bus is used for the transfer of instructions, operands, or interrupt vectors. In this view, the data bus is shared by several functions. Besides the dichotomy of functional and physical structure, the hierarchy of bus functions makes a classification even more difficult. For this reason, the classification shall be eased by the following definitions. A bus type is called dedicated, if at the same time only one pair of devices wants to communicate with one another. A bus is said to be shared, if at the same time more than one pair of devices want to exchange information. Usage conflicts are inherent to shared buses and are resolved by a mechanism called *arbitration*.

Some common bus topologies for computer systems are now presented and analysed. As will be seen, the most basic trade-off considerations are normally data transfer rate versus interconnection cost. The interconnection structure can be represented as a graph whose nodes correspond to data processing units or switching elements and whose edges represent data communication paths [Hayes 79]. A system may contain as many dedicated buses as logical connections are required. A fully connected system with n units that all need to communicate with each other requires $n \cdot (n-1)/2$ dedicated buses. At most n/2 different data transfers can take place simultaneously. An example for n = 4 is shown in Figure 4.3a. The other extreme is a system with one single shared bus, that is used for all communications of n units, as seen in Figure 4.3b. Only one transfer may be performed at the same time.

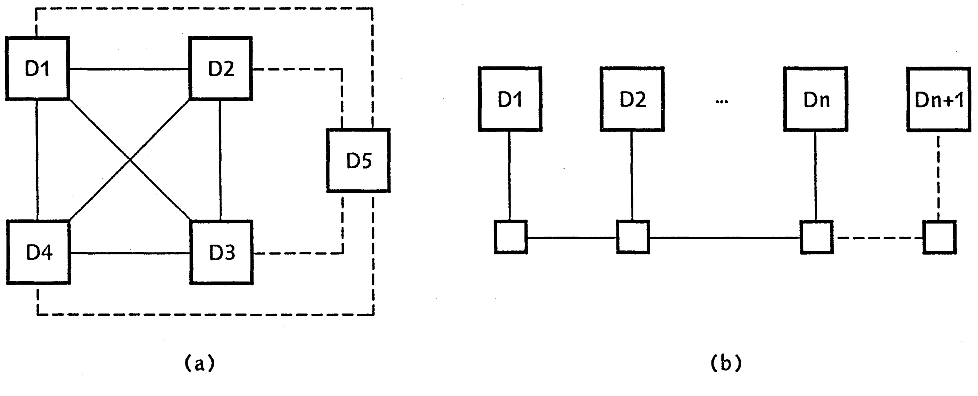


Figure 4.3 A dedicated (a) and a shared bus structure (b).

The advantage of a dedicated bus is its high data transfer rate, because there is no delay caused by bus contention and because parts of the communication protocol, such as the source or the destination address, can be implicit. In a system with several dedicated buses high transfer rates are achieved as information can be exchanged simultaneously. The disadvantages of such a system are its high interconnection cost and lack of flexibility. Adding a new unit requires that buses and interfaces are added to the existing units (Figure 4.3a).

The main advantages of a shared bus are modularity and low interconnection cost. [Thurber 78] defines modularity as the incremental cost of adding a device to the system. Adding a new unit to a shared bus causes only little cost, because the interconnection structure of the existing units has not to be altered (Figure 4.3b). The dedicated bus structure of Figure 4.3a cannot be said to be modular, since an extension may require the addition of n buses (for a system with n units).

Bus contention causes shared buses to be slow, since access to the bus is delayed when the bus is occupied. Further delays can be created, if bus allocation and data transfer are not processed in parallel. Besides delays, expense in logic has to be paid. Because only two devices may be connected at the same time, switching elements are required that allow to connect and disconnect devices to and from the bus. A supervisor is needed to control bus connections, i.e. to solve usage conflicts and allocate bus cycles. Requested bus cycles that cannot be processed immediately may require additional logic that allows to restart the cycle at a later time. The higher universality of a shared bus further causes more complex communication protocols. Addressing and synchronization, for example, have to occur explicitly.

The presented structures are extreme examples of a system using only dedicated buses and one having a single shared bus. Besides the outlined considerations a fully connected system is normally not required because some of the devices have their own specific functions which are used only by part of the devices. A compromise may be a tree-like interconnection structure. Figure 4.4 shows an illustration: a combination of a global, shared bus and several local, shared or dedicated buses that allows to execute several simultaneous data transfers.

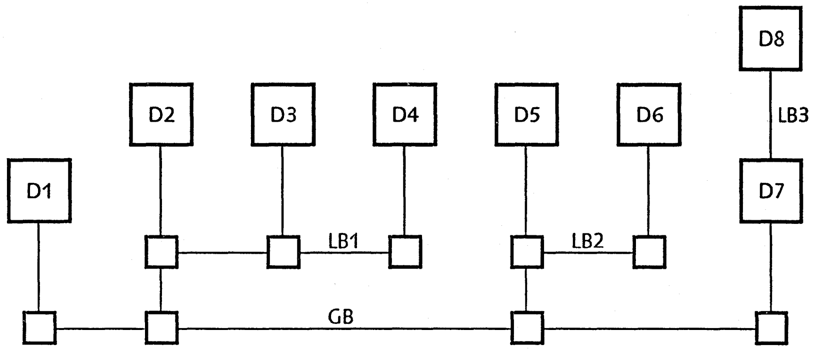


Figure 4.4 A tree-like bus structure.

Multiple buses not only increase bandwidth, but also reliability. A failure of a path in a dedicated bus structure (Figure 4.3a) only affects the two units that are connected to it. Nevertheless the two units may still re-route data transfers through other units which are employed as relays. Loop-free interconnection structures are less reliable. A bus failure stops all communication of the attached units since no alternative paths exist. While a failure in a tree-like bus system obstructs one branch (Figure 4.4), the same incident makes all communication of a system with a single shared bus impossible (Figure 4.3b).

4.2.2 Bus Arbitration

The economy of shared buses is attractive for many computer implementations, especially for smaller ones. The major problem of this interconnection method are usage conflicts: contention arises when the bus is requested by several devices at the same time; collisions are caused if several requesters gain access to the bus simultaneously. In order to guarantee a correct, error-free data transfer, the following methods exist:

- collision avoidance by contention avoidance
- collision avoidance by contention resolution
- collision detection and re-transmission of damaged transfer items

The mechanism to resolve bus contention is called *arbitration* and is the most suitable for parallel buses. [Chen 74] defines arbitration as a matter of assigning a single resource to one of a number of requesters. The shared resource is the bus and the resource requesters are the master devices.

A complete bus cycle of a shared bus consists of three phases: first the arbiter performs the allocation of the bus by selecting one of the requesting master devices, which then addresses the slave it wants to be connected with; finally, the actual data can be transferred. Although allocation, addressing, and data transfer of the same bus cycle have to be executed sequentially, the throughput of the communication path can be increased by pipelining these operations. Figure 4.5 shows the sequence of bus operations using two- and three-level pipelining.

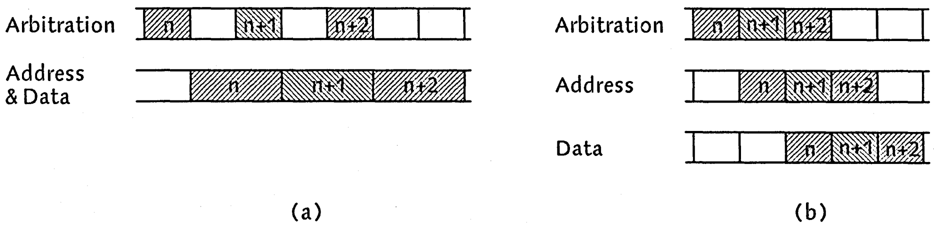


Figure 4.5 Two- (a) and three-level pipelining (b) of bus operations.

Before the examination of some arbitration methods, some classification criteria are considered. The location of the arbitration logic is said to be *centralized* or *distributed*. Centralized arbitration uses a single hardware unit to process bus requests. The location of the hardware could be within one of the bus master devices or it could be separated. In distributed arbitration the control logic is dispersed over all bus master devices. In spite of this distinction of the physical implementation, an arbiter is logically a single unit assigned to the shared resource.

Conflicts of competing requesters are resolved according to allocation rules. The rules use either *fixed* or *variable* priorities. Fixed priorities means that, as a result of a prespecified ordinal relation of requesters, always the same requester goes ahead of another. The priority of a requester is chosen according to the latency time that it can tolerate, i.e. high priorities are assigned to requests that require short response times. However, a monopolizing use of the bus is possible. If many high priority requests are continuously issued, low priority requesters are starved, i.e. excluded from accessing the bus. No specification of an upper limit to the service time can then be given. If real-time constraints have to be fulfilled, the behaviour of all requesters has to be analysed. A starvation-free, fair allocation algorithm uses variable priorities that allow to dynamically adapt the selection criteria to the given circumstances. Fairness is achieved by raising the priority of waiting requesters or by lowering the priority of already granted requesters. A modified fixed priority scheme can also guarantee fairness through the simple provision that new requests are inhibited until all pending requests have been processed.

An arbiter with fixed priorities uses a priority encoder that can be implemented either with a *serial* or a *parallel* logic circuit. The corresponding circuits are shown in Figure 4.6 and are only different implementations of the same logical function. The serial circuit has a regular structure but is slower, while the parallel circuit has a irregular structure and is faster. The same implementation choices are also known from other circuits as, for example, binary adders. Figure 4.6 further suggests a distributed or centralized implementation of the arbiter circuits. Not shown are additional provisions that have to be made in order to synchronize the requesting devices and defining the instances when the allocation of bus control can take place.

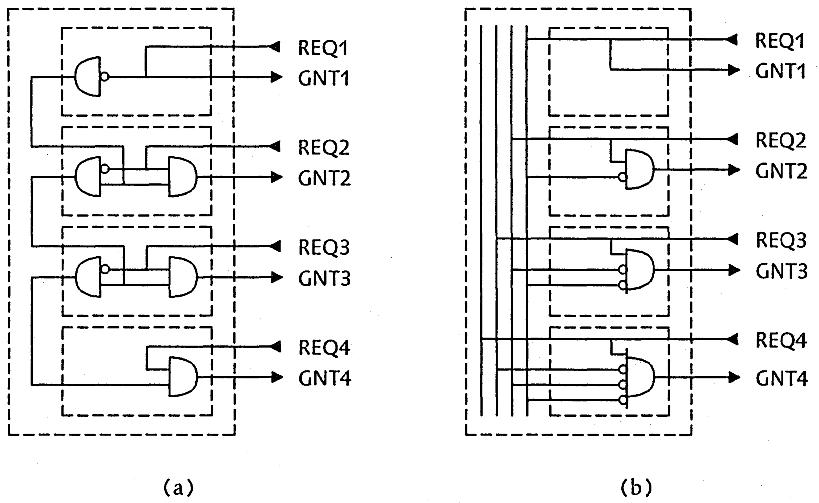


Figure 4.6 Priority encoders for a serial arbiter (a) and a parallel arbiter (b).

In the following paragraphs some common arbitration configurations are discussed. Bus arbitration methods are handled in [Thurber 72] and [Thurber 78]. Detailed explanations are also given in [Chen 74] and [Seck 83]. Note that similar methods are also applied to situations in which the problem of contention arises. An example is the handling of interrupt requests where several devices might be sending simultaneous interrupt requests to a processor. The basic criteria of an arbitration method are the allocation speed and the number of bus lines.

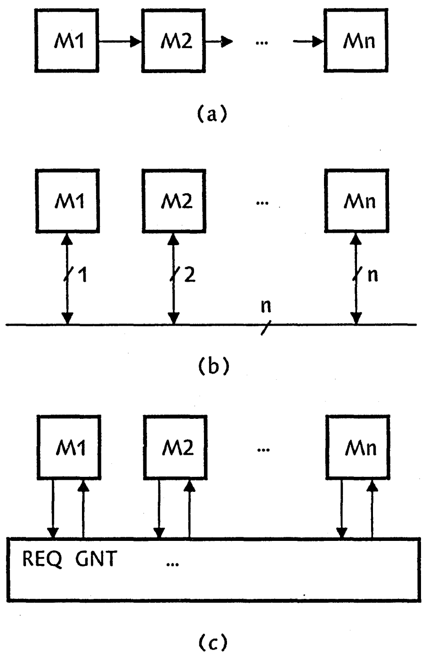


Figure 4.7 Distributed serial arbiter (a), distributed parallel arbiter (b), and centralized serial or parallel arbiter (c).

The distributed serial arbiter is also known as *daisy chaining* and is shown in Figure 4.7a. Its realization can contain the priority encoder of Figure 4.6a. The grant line is daisy chained through all devices. When the bus is available, the grant signal propagates through the chain passing all devices which are not requesting the bus. The first requesting device that gets the grant signal does not further propagate it and may access the bus. The priority is fixed by the position of the device in the chain. Daisy chaining uses only one control line (independent of the number of devices) and simple allocation logic. However, arbitration speed depends on the number of devices and can be quite slow. Furthermore, the addition or subtraction of devices is hindered by the demand of a continuous chain of requesters that may not be broken.

Figure 4.7b shows a distributed parallel arbiter. The bus contains n bus request lines assigning a distinct line to each master. Thus, each bus master has the ability to observe the requests of all competitors. The allocation rule says that a master device may only take the bus if no other device with a higher priority requests the bus. The described scheme can be implemented efficiently with the circuit of Figure 4.6b and offers short response times, since requests are processed in parallel. The disadvantage is that each master requires an individual request line. The number of lines can be reduced to $\log_2$ n if the request identifications are encoded. However, open-collector lines have then to be used resulting in a slower priority resolution.

The centralized implementation of a serial or parallel arbiter results in the same scheme as shown in Figure 4.7c. This method is also called *independent requesting* because it uses a separate pair of bus request and grant lines for each master. Favourably, parallel arbitration is preferred: the centralized arbitration logic can immediately identify the requesting device and respond with the corresponding grant signal. The main disadvantage of independent requesting is the number of control lines. 2n wires are required for n devices.

The Unibus illustrates a combination of a centralized parallel arbiter (independent requesting) and a distributed serial arbiter (daisy chaining) [Levy 78]. There are five groups of bus masters, each group has a fixed priority and its own bus request and grant line. Each group can contain several masters that are connected in daisy chain fashion. The CPU program execution priority allows to mask out certain requests. Therefore, the Unibus arbitration method contains a combination of the classification criteria discussed above: centralized and distributed location, fixed and variable priority, serial and parallel priority encoding. The distributed parallel arbitration method has become very popular and has been adopted for several recent bus standards as for IEEE's Futurebus [Taub 84], Texas Instruments' NuBus [TI 83a], and Intel's Multibus-II [Intel 84].

4.2.3 Transmission Techniques

Transmission Protocols

Three transmission protocols can be distinguished: *synchronous*, *asynchronous*, and *semisynchronous*. Examples of these protocols are given in Figure 4.8.

Synchronous transmission is characterized by a transmission frame of fixed length that is known to all participants (Figure 4.8a). The frame is divided into intervals that define when address and data have to be handed over or taken over, respectively. Synchronization is normally achieved by a globally generated clock signal that is distributed to all communicating devices. The main attraction of synchronous transmission is its simple implementation that requires no synchronizers. The disadvantage is the lacking flexibility: the slowest device determines the frame length. No advantage can be taken by a possibly increased performance of future devices. Further difficulties arise as a bus master gets no feedback whether a data transfer has been successfully completed.

In asynchronous transmission the communicating devices are synchronized by separate, interlocked signals (Figure 4.8c). In literature these are also referred to as handshake or sync/acknowledge control lines. Each data transfer is accompanied by a sync signal issued by the source to indicate that data is present, and an acknowledge signal issued by the destination to notify that data has been accepted. The popularity of asynchronous transmission is due to the fact that the transfer time can be varied with the devices. However, this advantage has to be paid with a longer transmission protocol caused by the propagation delays of the interlocked sync/acknowledge signals, and auxiliary hardware required because each device has to be able to control these signals. While additional control logic also has to be provided in order to abort a non-acknowledged bus transfer, asynchronous transmission is safer since faulty situations are detected, which can happen when the hardware is broken or a non-defined address is issued.

Semisynchronous transmission combines the advantage of the described techniques (Figure 4.8b). As in synchronous transmission a global clock is supplied to all devices. In addition, a wait signal is provided, comparable with the acknowledge signal in asynchronous transmission. By asserting the wait line the slave can request the insertion of wait cycles in order to extend the current transaction. Still, semisynchronous buses do not achieve the flexibility of asynchronous buses, because a bus transaction can only be extended by a discrete amount of time, that is an integral number of the basic clock period.

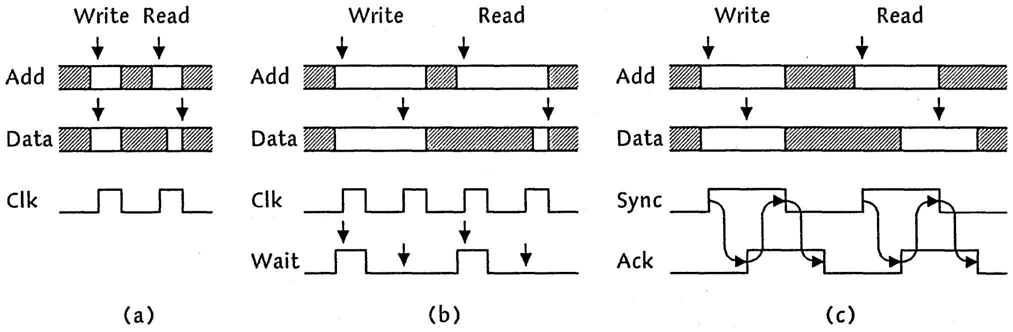


Figure 4.8 Synchronous (a), semisynchronous (b), and asynchronous transmission (c).

An alternative to semisynchronous transmission is synchronous transmission with a so-called *split-cycle* protocol. This technique allows variable-length transactions, but eliminates wait cycles that obstruct the bus mainly during read operations. For that purpose read operations are split into two cycles. During the first one, the requesting device transmits a unique identifier of itself and the address of the requested data accompanied by the read command. When the requested device has prepared its reply, it initiates the second transaction cycle by transmitting the identifier of the requesting device and the requested data accompanied by a response status. The time between the cycles can be used for other transactions. The cost of the gained versatility is expensive logic required for the requesting and responding devices which both have to be able to play the role of a master and a slave.

The Unibus, introduced in 1970, illustrates the flexibility of asynchronous data transmission: several subsystems with progressively increasing speeds have been introduced without changing the Unibus specifications [Levy 78]. However, now that buses operate near their physical speed limits, little advantage can be gained from asynchronous buses. The latest developments such as the NuBus [TI 83a] and the Multibus II [Intel 84] favour synchronous transmission. A synchronous split-cycle protocol was already used in 1977 for the SBI-bus (Synchronous Backplane Interconnect) of DEC's VAX-11/780 computer [Levy 78].

Transmission Formats

In this context, the transmission format of bus signals may be of interest. Bus lines transport address, data, and control signals. A *demultiplexed* bus has separate lines for address and data, whereas a *multiplexed* bus uses the same lines for both signal types at different times (Figure 4.9). Properly, time-multiplexing is a general technique that divides a transaction into phases which are assigned to parts of the information and are sequentially transferred on the same set of lines. The extreme is the partition into single bits that are transmitted on a single line. This case is called a *serial bus*. Multiplexing allows to reduce the number of lines and interfaces. As a consequence physical space and connectors are saved. Furthermore, power consumption and signal switching noise are reduced. However, slower data rates are the consequence as address and data are now transferred successively. The speed penalty is less than one might expect, because a device first has to be addressed before data can be transmitted.

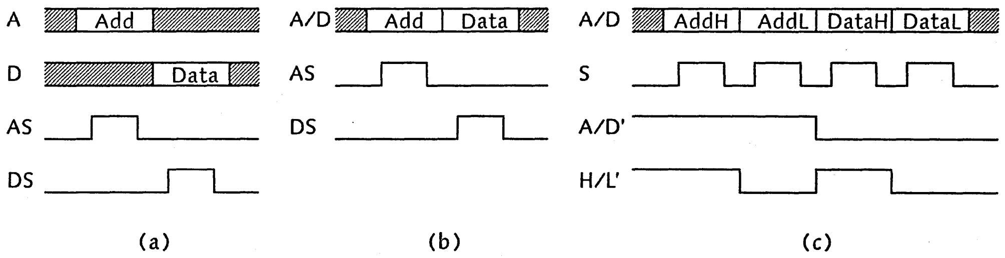


Figure 4.9 A bus cycle using a demultiplexed (a) and a multiplexed bus (b, c).

Other techniques may be applied to reduce the number of bus lines. An elementary method is the encoding of information, as it is common for the transmission of addresses. It may also be used for the reduction of control lines (Figure 4.9c). The cost for multiplexing and encoding are more expensive interfaces: multiplexers and encoders have to be provided for the source of the transmission, while demultiplexers and decoders are required at the destination.

Data Formats

Much confusion is caused by the existing conventions of data formats. Whole papers are dedicated to this subject [Cohen 81], [Kirrmann 83]. The unnecessary disorder usually starts with the numbering of bits and bytes. The memory can be assumed to be a linear array of bits organized into bytes, words, and other higher-level data structures. A hierarchical organization assigns each unit as a subunit to the next higher abstraction level. A consistent order numbers bits, bytes, words, etc. all starting from the same end of the memory array. While most people agree that the numbering of bits starts with the least significant bit (LSB), this is not the case for the numbering of bytes. Following Cohen's notation [Cohen 81], two conventions can be distinguished (Figure 4.10):

- In a *little-endian* format the least significant byte of a data unit is stored at the lowest memory address.
- In a *big-endian* format the most significant byte of a data unit is stored at the lowest memory address. This order is opposed to that used for the numbering of bits, which is unfortunate.

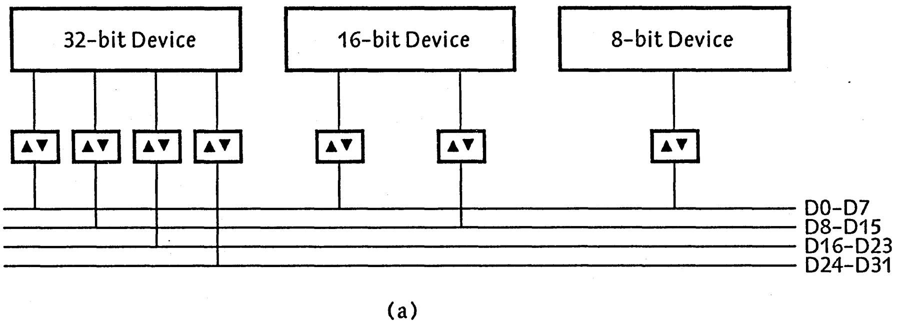
If the width of a bus is not fully used for all data transfers, specifications have to define which lines are used. An example is a 32-bit wide bus to which not only 32-bit devices, but also 16- and 8-bit devices are attached. A parallel bus is said to be either *justified* or *straight*

MSB LSB

31 24	23 16	15 8	7 0	
Byte 3	Byte 2	Byte 1	Byte 0	Little-Endian
Byte 0	Byte 1	Byte 2	Byte 3	Big-Endian

Figure 4.10 Bit and byte ordering.

(unjustified). With the justified bus all data that do not make use of the full bus width are aligned to the left or the right of the path. In a straight bus data can be aligned to any boundary that is given by forming groups of data lines that are as wide as the smallest accessible data item. Interfaces for a justified and a straight 32-bit bus are shown in Figure 4.11. Note that the examples are little-endians. On the straight bus 16-bit devices get only half, 8-bit devices only a quarter of the addresses, if no additional multiplexers are provided. On a justified bus all devices can use all addresses. It is obvious that complex interfaces may be required if processors, memories, IO devices, and buses are connected that use different data formats.



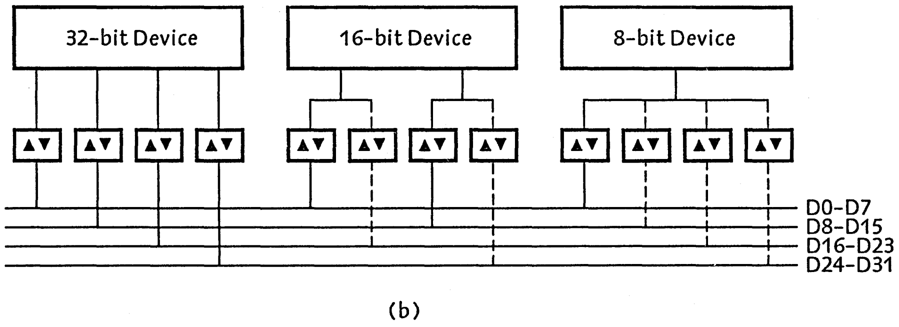


Figure 4.11 Interfacing a justified (a) and a straight (b) bus.

4.3 Design of the Ceres Bus System

The preceding section has introduced a classification of bus parameters. The given terminology and the basic trade-off considerations help to define the communication requirements between the computer's system components and to choose the appropriate bus structure. However, cost and technology constraints will further influence the bus implementation. The dependencies of bus parameters can cause the conception and implementation to pass through several iterations.

The Ceres bus structure is shown in Figure 4.12. The shared *memory bus* is realized as a global backplane bus. Other, mainly dedicated buses such as the *slave processor bus* may be implemented locally. In the following sections, the characteristics, design, and implementation of the slave processor bus and the backplane bus are explained.

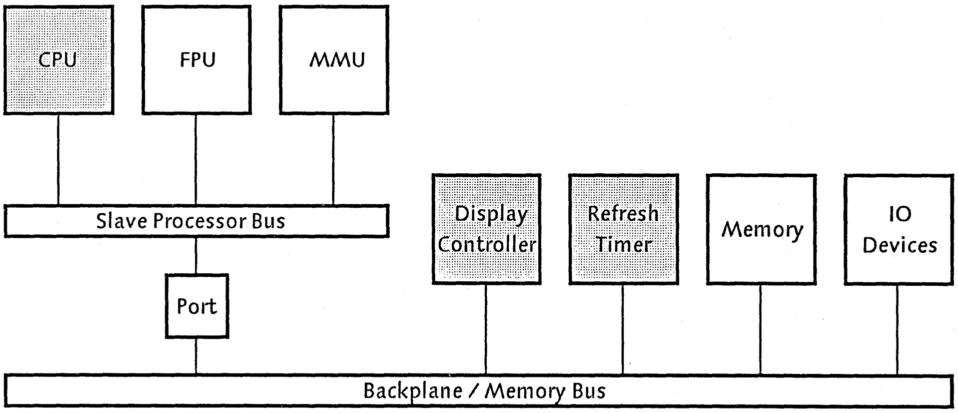


Figure 4.12 The Ceres bus structure.

Designing the Ceres bus was a most instructive task and proved the architectural importance of bus structures. Much work went mainly into the design of the backplane bus, its bus allocation techniques, its interfaces, and not to forget the realization of the printed circuit board.

4.3.1 Slave Processor Bus

Due to technological limitations, but also the need for modular design, the NS32000 processor functions are distributed among a chip set of three members: the central processing unit (CPU), the memory management unit (MMU), and the floating point unit (FPU) [NS 86a]. The MMU and FPU are optional and are not required for the functioning of the CPU. In addition, the CPU can support user-defined custom slave processors. All units are connected by the so-called slave processor bus. Although there is no freedom in designing this bus, it is a good illustration of a high-performance, dedicated bus of little flexibility.

The detailed interface specifications are given in [NS 86a]. Only few connections are needed consisting of sixteen data lines and three control lines that determine the type of transfer and the validity of the data signals. The slave processor bus requires no additional interface logic. Transmission is synchronous and takes two clock cycles for one bus cycle resulting in a data transfer rate of 10M bytes/s. If a 32-bit wide data bus could be used, the data rate would be doubled to 20M bytes/s.

Slave processor communication follows a simple protocol. The CPU first broadcasts an identification code in order to establish a connection with one of the slave processors. It then transfers the operation code of the instruction to be performed followed by the required operands. The slave processor is now able to start execution of the instruction. While the slave processor is performing the operation, the CPU is only allowed to prefetch instructions, but not to process instructions in parallel. Upon completion, the CPU reads the computed results. As seen, all transfers are initiated by the CPU, which acts as the only master. This also implies, that the CPU is solely responsible for fetching the source operands and storing the destination operands.

The outlined sequence is a good example of having implicit protocol parts: once the CPU has installed a connection, the slave processor has no longer to be addressed explicitly, it stays activated until the end of the protocol.

However, dedicated buses are inflexible. This can be illustrated by both the implementation and the specification of the local slave processor bus. Because the bus is specifically accommodated to the used processor chip set, it is not worth providing mechanical interfaces, such as bus connectors, in order to offer the facility of future extensions. Moreover, the NS32032 CPU supports only the 16-bit wide slave processor interface of the first generation NS32082 MMU and NS32081 FPU, which have originally been designed together with the NS32016 CPU [NS 86a]. The faster 32-bit wide slave processor bus of the next generation NS32382 MMU and NS32381 FPU cannot be supported [NS 86b].

4.3.2 Memory Bus

The backbone of the Ceres computer is the global backplane bus. It has both an electrical and a mechanical function as it provides the communication paths, power distribution, and a mechanical support. According to the classification criteria of Figure 4.2 its characteristics can be summarized as follows. The *shared bus* is controlled by a *centralized arbiter* with *fixed priorities* that uses a *parallel priority encoder*. A *semisynchronous transmission* protocol is used to transfer data in a *little-endian* order on a *demultiplexed, straight* bus. These characteristics are now handled in more detail.

Bus Type

The memory bus is shared by several master and slave devices. The master devices typically consist of the processor and several controllers. The number of masters is restricted by the arbiter that can manage up to seven devices. The slave devices comprise memory and IO devices. Their number is determined by the addressable memory space which is 16M bytes.

The basic arrangement contains three masters, that are shown in Figure 4.12 (shaded boxes). Besides the processor, two controllers can be recognized: the display refresh controller, which has to access the video frame buffer in order to generate the video signal, and the DRAM refresh timer, which periodically has to perform a memory refresh cycle so that the dynamic memory chips do not loose their stored data.

The two mentioned controllers are special for the reason that they request a bus cycle not because they need the bus communication facilities but rather because they have to prevent other masters from simultaneously accessing the memory. In both cases the data path is not used. During a memory access of the display controller data are transferred into internal registers of the memory chips. In the course of a memory refresh cycle no data are transferred at all, because only a dummy read operation has to be performed.

Although both operations could be performed locally, i.e. hidden from the global memory bus, this simple solution is preferred particularly because the loss of memory bus bandwidth is negligible. The display controller has to access the video memory once per 8 display lines during the active vertical display interval, which corresponds to a period of 154 μs or 0.25% of the total bus bandwidth (the display parameters are specified in Section 2.3.3). The refresh timer has a period of 16 μs implying that it consumes 2.5% of all possible bus cycles.

Bus Arbitration

A centralized control logic is responsible for the allocation and timing of bus cycles. A centralized concept is simple to implement and to test. Furthermore, simple interfaces can be used to attach devices to the bus making local control logic superfluous. The arbiter uses an independent requesting method, because it is suitable for a centralized implementation, but also allows very fast allocation speed.

The arbiter employs fixed priorities for resolving conflicts of competing master devices. The implementation consists of a simple combinatorial circuit, namely the parallel priority encoder of Figure 4.6b. This scheme is well suited for a single-processor system in which devices with highly divergent characteristics and requirements are attached to the bus: while the processor makes most use of the bus, the controller devices are normally infrequent consumers; but in contrast to the processor their bus requests normally underlie more severe real time constraints. Therefore, latency tolerance is a more important criterion than fairness.

The display controller can serve as a typical example. It requests video data during the horizontal blanking interval (horizontal retrace of the video beam). During this interval the request has to be processed because the data to be fetched are already needed for the next display line. As this example shows, a controller normally is also short of expensive, local buffer memory. Therefore, a request to access the memory has to be acknowledged within a fixed and short time bound.

Additionally, short response times are achieved by a *cycle-by-cycle arbitration* which means that the arbiter allocates only single bus cycles. Other methods allow variable-length transfer sequences controlled by the same master: after a master has gained access to the bus, it is its decision when to relinquish control of the bus. Although the overhead of arbitration delays is reduced, the problem of starvation arises, if a momentary master ignores higher priority requesters.

Nevertheless, starvation remains a problem of the applied method. It is possible that low priority requests are excluded from bus allocation by higher priority requests. Although it is assumed that controller devices in the average make only little use of the bus, the interplay of master devices may not be left out of consideration.

A major problem was induced by the mentioned asymmetry between the master devices. In a single-processor computer the processor obviously plays a dominating role. Therefore, the backplane bus is normally optimized for its most frequent user. This cannot only affect the characteristics of the bus signals but also the allocation algorithm. In order to avoid allocation delays, the bus may be assigned to the processor by default, in that the bus is not released until a request of another master is issued. On the other hand symmetric solutions are to be preferred because no distinction of cases is necessary. Case distinction complicates the design and is usually a source of mistakes. In view of these contrary requirements several versions of the arbitration and bus timing control logic have been implemented, of which two concepts can be distinguished.

Influenced by the NS32000 processor architecture the scheme of Figure 4.13 was implemented first. Due to its shortcomings this concept was later given up. By default the bus is assigned to the processor. An active hold request signal (HOLD) informs the processor that some other master requests access to the bus. Thereafter, the processor relinquishes the bus either when it is idle on the bus or after the current bus cycle is finished. On receipt of an active hold acknowledge signal (HLDA) from the processor, the arbiter grants the request with the highest priority.

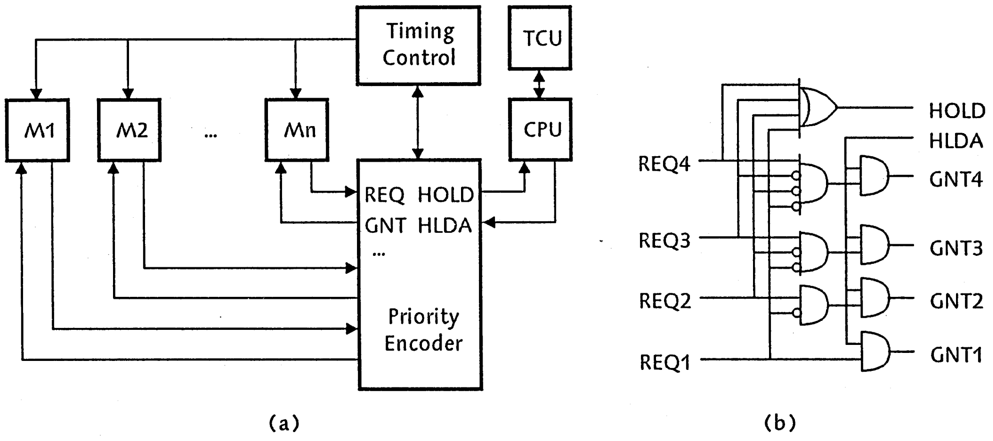


Figure 4.13 The first arbitration concept (a) with a schematic of the priority encoder for n = 4 (b).

Note that the timing of bus cycles is controlled by separate circuits. The processor uses a timing control unit (TCU) that belongs to the Series 32000 microprocessor family [NS 86a]. It contains a clock oscillator and a cycle control logic. Another, centralized timing control circuit is responsible for any bus cycle of the controllers. A separation is necessary as the processor's TCU is not intended to be shared by several master devices. This restriction leads to an unnecessary complex solution: separate circuits are required, which also have to be synchronized with each other.

The main disadvantage of the presented arbitration concept originates from the way the processor can be forced to release the bus. The method with hold request and hold acknowledge signals was already used for the early 8-bit microprocessors, although it is now said to be used not only for DMA but also for multiprocessing purposes. It is inefficient because the processor is preventively halted also in situations where no bus collisions occur. This is the case for slave processor communication that is performed on the local, dedicated bus which is isolated from the arbitrated backplane bus (Figure 4.12). Moreover, the processor performs no internal bus cycles after it has acknowledged a hold request [NS 86a]. Further delays are caused by the comparatively slow response time: two clock cycles, i.e. 200 ns are lost for every hold request - hold acknowledge handshaking. Especially when using a cycle-by-cycle arbitration method, this can cause an inadequate loss of bus bandwidth.

The second, more uniform concept is outlined in Figure 4.14 and represents the final solution. Now not only the controller devices but also the processor has to request a bus cycle explicitly. Further, a single circuit controls the timing of bus cycles for all master devices. However, the processor is still treated differently: although it has the lowest priority, it has the privilege that the bus is assigned to it whenever no other controller requests access to the bus. This strategy is called *default assignment* and is illustrated with the diagram of the priority encoder (Figure 4.14b). A similar method was already used for the VAX-11/780 computer [Levy 78]: the CPU is given the lowest priority with the privilege that it may transmit without asserting a request signal of its own whenever no other connection uses the bus.

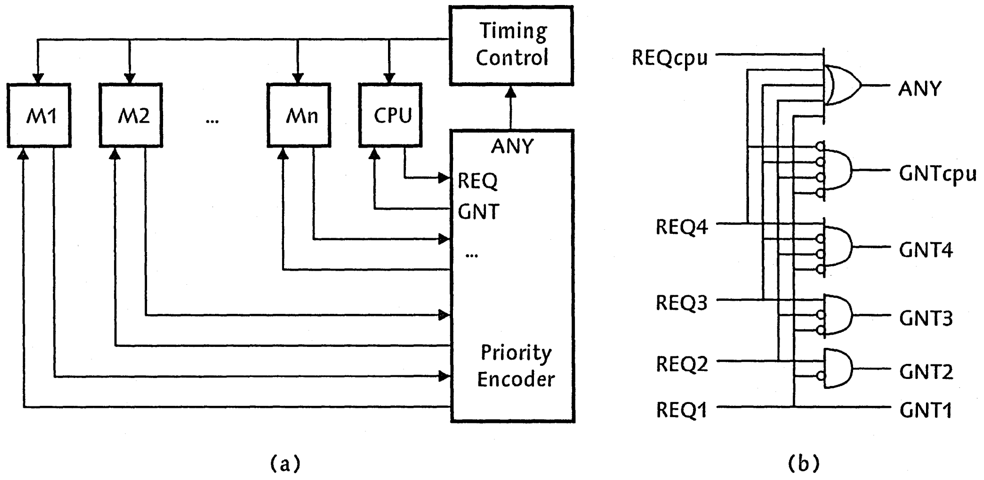


Figure 4.14 The second arbitration concept (a) with a schematic of the priority encoder for n = 4 (b).

Provided that the bus is not occupied by any controller, a request from the processor can be processed immediately and is not slowed by any delay for arbitration or activating the switches for the bus lines. With the aid of Figure 4.15 the timing of bus cycles as seen from the priority encoder is explained. A bus cycle takes four system clock cycles, labeled T1 through T4. It is presupposed that the request signals (REQ1..n) of all controller devices are synchronized with the leading edge of the system clock (CLK). As soon as the bus is idle, i.e. no controller sends a request to the priority encoder, the processor is selected (GNTcpu). Before it is known, whether the processor makes use of the bus, the buffer devices of its bus interface are enabled. Only in the middle of T1 the processor address strobe signal indicates the beginning of a bus cycle and issues a bus request (REQcpu). Because no request conflicts have to be resolved, the processor can proceed without being slowed down by any arbitration delay, i.e. without a loss of a clock cycle. The timing controller starts a bus cycle if at the end of T1 any request (ANY) including the one from the processor is pending. If the processor issues a bus request and the bus is already occupied, the processor is halted by using its cycle extension facility.

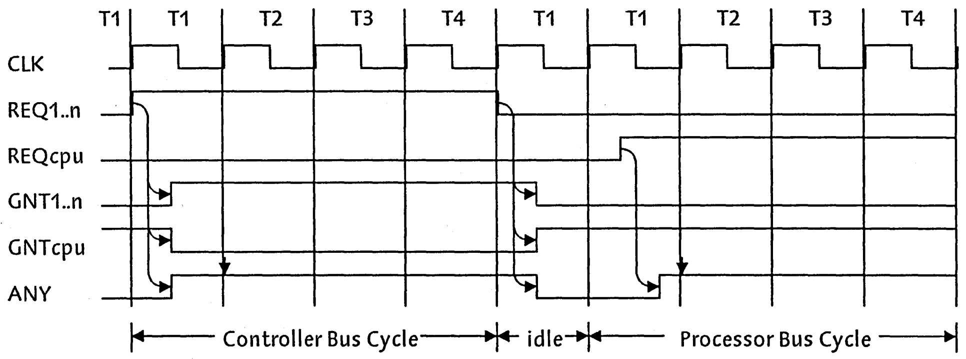


Figure 4.15 Bus timing as seen from the priority encoder.

The main advantage of this scheme is that the processor is no longer preventively halted, but only in situations where there is a bus conflict. Traffic on the backplane bus does not affect slave processor communication any more. Even more important, internal execution is not hindered as long as the processor does not need access to the bus.

The presented method still remains unsatisfactory. The NS32000 processor indicates too late when it wants to begin a bus cycle. If it would be known earlier, the arbiter circuit would be much less time critical. It might be even possible to pipeline bus operations by overlapping bus allocation with the previous bus cycle as shown in Figure 4.5a. Actually, the processor provides status information that precedes the corresponding bus cycle. However, in conjunction with the MMU the address strobe signal, more precisely the physical address valid signal (generated by the MMU) is the only reliable indication that stands in a fixed timing relation with the following bus cycle. A further improvement could be gained if the address information would be presented earlier. If the address could be decoded before the request has to be issued, local and global bus requests could be distinguished. Access to resources that are exclusively utilized by the processor could be granted on the local bus without interfering with global bus communication.

The demonstrated deficiency of the NS32000 processor of interfacing a shared bus might be overcome with a pipelining technique. Due to the pipelined architecture of modern microprocessors, the address information of the next bus cycle is mostly already known during the current bus cycle. By revealing internal pipelining to external bus interface logic time is gained that allows to decide whether the local or global bus has to be accessed and further permits bus arbitration to take place overlapped with the current bus cycle.

The single timing control circuit of Figure 4.14a does not make any use of the Series 32000 TCU. An implementation with two programmable array logic chips was chosen because it seemed impossible to share the TCU among several masters. The TCU is degraded in that it only serves as a clock oscillator for the processor.

Transmission Techniques

A semisynchronous transmission protocol is used for the memory bus because it is simple to implement and is supported by the NS32000 processor. The basic bus cycle takes four clock periods or 400 ns and can be extended through insertion of *wait states*. Multiple control signals allow the insertion of a fixed or variable number of cycles and eases interfaces of slow memories and IO devices.

During the development of the Ceres computer both possible transmission formats, a multiplexed and a demultiplexed backplane bus have been implemented. Figure 4.16 shows the required bus interfaces for both cases. The advantage of a multiplexed bus are lower interconnection cost and reduced signal switching noise. This gets more significant as the development of microprocessors produces continuously wider address and data paths. The multiplexed bus seems to be even more attractive when the master and slave devices already multiplex address and data lines. For Ceres this is true of the NS32000 processor and the main memory. The concept of the multiplexed bus was therefore implemented, but showed weakness when the above described, second arbitration method was implemented.

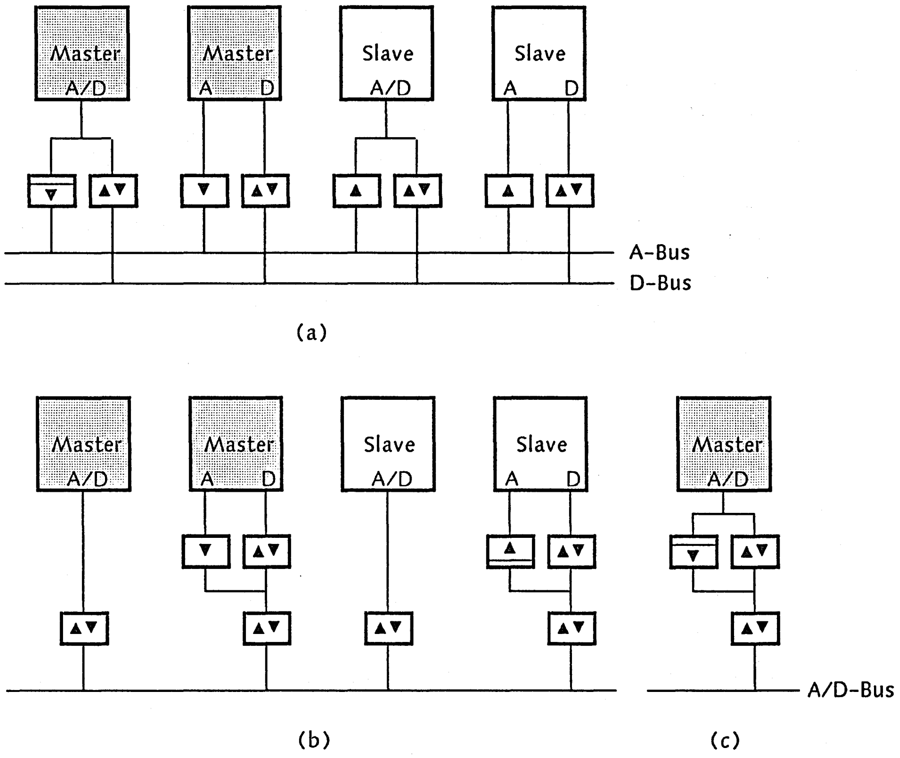


Figure 4.16 Interfacing a demultiplexed (a) and a multiplexed bus (b, c).

If in case of a bus conflict, a master such as the NS32000 processor which provides multiplexed address/data information, can be halted only during the data transfer and not before or while it issues the address, then the bus interface gets much more complicated. Figure 4.16c shows the required logic. The address has to be temporarily stored until the bus is granted. The bus interface then has to generate the correct timing sequence of the address followed by the data information. Therefore, in order to avoid expensive interfaces, the backplane bus was redesigned with separate address and data lines.

Having a straight bus the data path of the processor and the memory can be connected straightforward. The data lines are grouped as four bytes that can be individually controlled by byte enable signals. As Figure 4.17 shows, data can be transferred on the bus in 10 different access ways depending on data width (one, two, three, or four bytes) and path position. The NS32000 processor is able to access any item, regardless of size, from any byte address. Access to a double-word with an address modulo 4 equal 1, for example, is accomplished with two subsequent bus cycles of type 9 and type 1. This capability is questionable and will be analysed in Chapter 5. Devices with a smaller data path are connected to the least significant data lines and can only be accessed with addresses modulo 4 equal 0. This restriction has been accepted in order to avoid expensive interfaces that use multiplexers as shown in Figure 4.11.

Byte 3	Byte 2	Byte 1	Byte 0	Type	A1 A0	Data Width
			▨	1	0 0	8-bit Access
		▨		2	0 1	
	▨			3	1 0	
▨				4	1 1	
		▨	▨	5	0 0	16-bit Access
	▨	▨		6	0 1	
▨	▨			7	1 0	
	▨	▨	▨	8	0 0	24-bit Access
▨	▨	▨		9	0 1	
▨	▨	▨	▨	10	0 0	32-bit Access

Figure 4.17 Bus access types of the NS32000 processor.

Modularity

Modularity was one of the main bus design concerns, because it not only allows future hardware expansions but also supports a simple and testable hardware implementation. Expandability makes the following demands:

- the bus has to be well specified and documented
- the bus interfaces have to be simple, avoiding complicated control logic
- the cost of the bus connections have to be commensurate with the anticipated role of the extensions
- the limitations such as the memory address space and the bus bandwidth have to be carefully chosen allowing sufficient reserves. Other essentially physical quantities that are to be considered are the number of available free card slots, the bus length, the bus driving capabilities and the admissible maximum power consumption

Although modularity can be measured with the incremental cost for the addition of new devices, the undertaken investment may not be omitted: the mentioned reserves have to be paid for before they are used.

With the following provisions the interface logic for bus additions can be simplified:

- most signals are transmitted in decoded and demultiplexed form which avoids encoders and multiplexers for the controlling device and decoders and demultiplexers for the responding device
- control signals are generated centralized in order to avoid distributed control logic

An illustration can be given with the proposed interfaces for a Ceres bus master and a bus slave that are contained in Section 2.4. The centralized arbiter is solely responsible for the timing of bus cycles and therefore relieves the masters of generating the corresponding timing control signals. Note that the interfaces contain no sequential circuits. Decoded control signals are transmitted, i.e. the phases of a bus cycle can be identified with individual control signals. Actually, the bus cycle phases of synchronous transmission could be recognized with the help of a finite state machine. Interfacing peripheral devices is further eased by the presence of a *peripheral cycle*. Memory accesses that fall into the reserved address area for IO devices are automatically extended in order to allow sufficient setup and hold times for address and data of slower peripherals. However, the proposed techniques need more signal lines and have to be paid for with increased interconnection cost.

The extensibility of the Ceres backplane bus has been proven by the addition of interfaces for a colour monitor and a laser beam printer. The former uses a slightly modified version of the existing display controller board. Without any interference the standard black/white monitor and the colour monitor can be refreshed simultaneously. The structure of the printer interface is again similar to the display controller with the difference that it contains no on-board bitmap buffer. Its main task is to fetch data from main memory, serialize it, and send it to the printer.

During the system development modularity and transparency have been of great value. As shown in Section 2.3, the Ceres hardware is partitioned into the processor board, memory board, and the display controller board, that are all connected to the common backplane bus. The boards have been implemented and tested in the same sequence. The partition of functions and distribution to the boards had been done in a way that each level of implementation was working autonomously and completely testable. System development is further simplified through transparency, which means that all system resources, such as memory and IO devices, are made global and therefore accessible from the backplane bus.

4.3.3 Why not a Standard Bus?

Using a standard bus allows to compose a complete computer system by using board-level products. The results are shorter development time and therefore lower development cost. In applications in which some custom design is required, it might be still advantageous to incorporate a standard bus in order to have the possibility to use standard boards for future expansions. However, the flexibility of standard buses exact more expense due to the greater component count, which again requires additional board space and increases power consumption. For that reason standard buses are mostly used for systems that are dominated by development costs or need configuration flexibility, while less flexible systems that are dominated by manufacturing costs will use specialized configurations to avoid the costs of unnecessary components and connections.

Although a product with standardized interfaces can be commercially more attractive, the designer of a computer architecture has to answer whether the available standard buses are appropriate and do not impose unnecessary complexity, which causes lower reliability and a loss of performance.

With the availability of 32-bit microprocessors also several 32-bit bus standards have emerged. The best known are the *VMEbus, Multibus II, NuBus,* and *Futurebus.* A comparison of standard buses can be found in [Corso 86] and [Borrill 85]. The buses have many similarities starting with the physical specifications: all boards have Eurocard dimensions and one or two 96-pin connectors (DIN 41612-C96). With the terminology introduced in Section 4.2 the enumerated buses can be characterized as follows.

The VMEbus (Versa Module European) was the first available 32-bit standard bus and was released by Motorola in 1980 [Motorola 85a]. The bus can be shared by up to 20 masters. The arbiter is realized as a centralized allocation logic with independent requests whereby each request resembles a distributed daisy chain. The priority of each chain can be fixed or variable. An asynchronous transmission protocol is used to transfer data on a demultiplexed bus. Justification is irregular. (Considering Figure 4.17 the bus is 16-bit justified for accesses of type 1, 2, 3, 4, 5, and 7, but straight for accesses of type 6, 8, 9, and 10.)

The Multibus II was issued by Intel in 1983 [Intel 84]. Up to 20 masters can share the bus. A distributed parallel arbitration method with fixed priorities controls access to the bus. A semisynchronous transmission protocol using a 10 MHz clock provides data transfers on a multiplexed, 16-bit justified bus.

As depicted in Figure 4.18 the VMEbus and the Multibus II actually are multiple bus structures. The VME and iPSB buses are global system buses and have been described above. They provide data movement and inter-processor communication functions. Besides, optional buses can be supplied for specific functions. The VMX and iLBX II buses are local buses mainly intended for accessing local memory and peripheral devices. The VMS and iSSB buses are serial buses purposed for inter-process communication. In addition, the Multibus II provides two further buses, an IO expansion bus called iSBX bus and the Multichannel DMA bus, that both have been carried over from its predecessor, the Multibus I, which is a 16-bit standard bus.

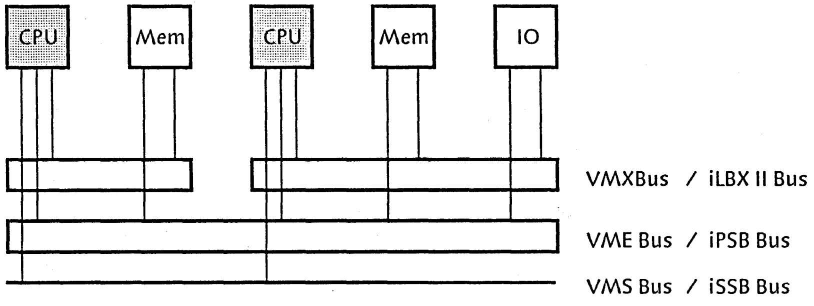


Figure 4.18 VMEbus and Multibus II bus architecture.

The NuBus was originally designed at the Massachusetts Institute of Technology in 1980 and later taken over by Texas Instruments [TI 83a]. The Nubus specifications are based on coherent concepts. Up to 16 masters share the bus and control access to it by a distributed parallel, fair arbiter scheme that uses fixed priorities. The transmission protocol is semisynchronous and uses a 10 MHz clock. The bus is multiplexed and straight.

The Futurebus is mainly propagated by the IEEE. The final specifications are on the way. Special attention is given to the electrical behaviour of transmission lines. Severe electrical specifications reduce signal noise and lower signal delays. The transceivers are non-standard devices with optimized parameters for the bus environment. The bus can be shared by up to 21 masters. Again, a distributed parallel arbiter with fixed priorities is used, whereby fairness is guaranteed. An asynchronous protocol transfers data on a multiplexed, straight bus.

The evaluation of standard buses has to consider in particular buses that are supported by a wide supply of board level products. At the given time, only the VMEbus and Multibus II fall into this category. Neither of these or the other standard buses has been chosen as a backplane bus of the Ceres computer for the following reasons:

- All 32-bit standard buses are intended for <u>multiprocessor applications</u> providing more or less sophisticated mechanisms for bus locking, multi-destination interrupts, fair arbitration, and cache-memory support:

    - bus locking allows indivisible sequences of bus operations in order to provide mutually exclusive access to shared memory variables
    - interrupts originate not only from multiple sources, in a multiprocessor environment they also have to be allotted to multiple processors
    - the arbitration algorithms are optimized for fair allocation of bus cycles and less for short latency tolerances of devices that have to comply with real-time constraints
    - modified read and write operations allow to maintain consistency of local cache-memories that store shared data

  The shown multiple bus structure of the VMEbus and the Multibus II resemble the same basic multiprocessor architecture: interprocessor communications are handled over the main system bus, while program access and execution are handled via the processor's local bus.

  Ceres is a single-processor computer. It never was intended to incorporate multiple processors nor to be part of a multiprocessor system. A multiprocessor bus as provided by the available standard buses is therefore conceptually not an appropriate backplane bus.

  (Note: A multiprocessor system uses multiple homogeneous processors that all have the same available resources. The partition of tasks among the processors is done dynamically.)

- Mainly the popular VMEbus and Multibus II are <u>processor-dependent</u>. The VMEbus reflects the signal timing of Motorola's 68020 microprocessor [Motorola 85b], while the Multibus II is tailored to Intel's 80386 microprocessor [Intel 86]. Transmission protocols and data formats of these buses and processors are tuned to each other. In general, connecting processors, memories, IO devices, and buses with different characteristics can cause a loss of performance and the necessity for more complex interfaces. The analysis of transmission protocols and data formats in Figure 4.19 illustrates this problem and calls in question, whether a processor-independent bus can be defined at all.

Processor	Data Width	Byte Orientation	Justification	Transmission Protocol	Transmission Format
NS32032	8, 16, 24, 32	Little-Endian	Straight	Semisynchr.	Multiplexed
MC68020	8, 16, 24, 32	Big-Endian	Straight	Asynchronous	Demultiplexed
80386	8, 16, 24, 32	Little-Endian	Straight	Semisynchr.	Demultiplexed

(a)

Bus	Data Width	Byte Orientation	Justification	Transmission Protocol	Transmission Format
VMEbus	8, 16, 24, 32	Big-Endian	Irregular	Asynchronous	Demultiplexed
Multibus II	8, 16, 24, 32	Little-Endian	16-Bit Justified	Semisynchr.	Multiplexed
NuBus	8, 16, 32	Little-Endian	Straight	Semisynchr.	Multiplexed
Futurebus	32	Not Specified	Straight	Asynchronous	Multiplexed

(b)

Figure 4.19 Transmission techniques of microprocessors (a) and standard buses (b).

- Due to their versatility standard buses are complex and so their implementation may even require support chips. VLSI-chips are available for both VMEbus and Multibus II providing functions for interrupt handling, bus arbitration, and timing control. The most complex of these chips implements a hardware-based message-passing protocol for Multibus II, that allows to transfer data structures as required by interprocess communication. Without processor intervention messages are transferred using a "pass by value" protocol that can consist of several bus cycles. An application of this is the transfer of interrupt information. No dedicated interrupt-control lines are needed, instead, interrupt information is passed as a special message.

Summing up, standard buses are multiprocessor-oriented, processor-dependent, and may require complex interfaces. For these inexpediencies the Ceres computer incorporates a private backplane bus that is optimized for its single-processor architecture. Nevertheless, an adapter to a standard bus could still be provided. If only IO devices are added, an attractive alternative is a SCSI interface adapter [NCR 85]. It is an 8-bit parallel bus (thus eliminating data format incompatibilities) that can be shared by up to 8 devices allowing asynchronous or synchronous transfers with data rates of 1.5M bytes/s and 4M bytes/s, respectively.

5 Analysis of Processor–Memory Communication

5.1 Introduction

As described in Chapter 4, cost and performance of a computer system are significantly influenced by its bus structure. An optimum interconnection structure can only be found with a good knowledge of the system's communication requirements. In this respect the design of a single processor microcomputer system such as the Ceres workstation seems to be a simple task: if the characteristics of the processor's memory interface and the memory bus match, an economic solution appears to be obvious.

As mentioned, the first version of Ceres contained a NS32016 CPU and a 16-bit wide memory data path. The redesign finally contained a NS32032 CPU with a 32-bit wide data bus. Contrary to expectations, the performance benefit gained by doubling the memory bandwidth was minor resulting in a worse cost/performance ratio for the NS32032-based Ceres than for the NS32016-based Ceres. But also the development of the NS32032-based Ceres required a disproportionate effort: Mainly because of the bus width, additional electrical problems appeared; even worse, faulty CPU-chips have delayed the development progress by months. With this experience, the motivation was given to analyse the processor's utilization of the memory bus in detail.

The redesign of Ceres, i.e. the replacement of the NS32016 CPU with the NS32032 CPU, was based on the assumption that mainly the performance of the bus which connects the processor and the memory would affect processing performance. Like most other available computers, Ceres has the structure of a classical von Neumann computer. Due to its simplicity and flexibility this concept has been kept alive for over forty years. The limitations of this architecture lie in the connection of the CPU and memory. Backus has called this connection the *von Neumann bottleneck*. In [Backus 78] he wrote:

[Quote] In its simplest form a von Neumann computer has three parts: a central processing unit (or CPU), a store, and a connecting tube that can transmit a single word between the CPU and the store (and send an address to the store). I propose to call this tube the von Neumann bottleneck. The task of a program is to change the contents of the store in some major way; when one considers that this task must be accomplished entirely by pumping single words back and forth through the von Neumann bottleneck, the reason for its name becomes clear. Ironically, a large part of the traffic in the bottleneck is not useful data but merely names of data, as well as operations and data used only to compute such names. Before a word can be sent through the tube its address must be in the CPU; hence it must either be sent through the tube from the store or be generated by some CPU operation. If the address is sent from the store, then its address must either have been sent from the store or generated in the CPU, and so on. If, on the other hand, the address is generated in the CPU, it must be either generated by a fixed rule (e.g., "add 1 to the program counter") or by an instruction that was sent through the tube, in which case its address must have been sent ... and so on. [Unquote]

The memory bus bandwidth B is defined as the maximum rate in bits per second at which information can be transferred to or from memory and depends on the bus width W, which is the number of bits that can be transferred simultaneously, and the cycle time T_M, which is the minimum time that must elapse between the initiation of two different memory accesses:

$$B = W \cdot T_M^{-1}.$$

The memory bandwidths of the NS32016-based and NS32032-based Ceres are:

$$B_{16} = 16 \text{ bit} \cdot (400 \text{ ns})^{-1} = 40 \cdot 10^6 \text{ bit/s},$$

$$B_{32} = 32 \text{ bit} \cdot (400 \text{ ns})^{-1} = 80 \cdot 10^6 \text{ bit/s}.$$

The time T_E required to execute an instruction is determined by the total size of transferred memory data M, the memory bus bandwidth B, and the processing overhead P:

$$T_E = \frac{M}{B} + P$$

Both, the instruction and data stream flow through the von Neumann bottleneck and contribute to M. Note that M also includes useless data that are transferred if not the full bus width is used. The processing overhead P combines the time required to decode the instruction, to calculate the addresses of the instruction and of the operands, and to actually execute the instruction. While P was dominant for the early processors, i.e.

$$P \gg \frac{M}{B} \text{ and } T_E \simeq P,$$

technological advance has lowered P, in that

$$P \rightarrow 0 \text{ and } T_E \simeq \frac{M}{B}.$$

The limitations of the von Neumann bottleneck can be mitigated by increasing B or decreasing M. B can be increased by increasing the width of the transferred word or by decreasing the cycle time of a word transfer. M can be decreased by adding a level to the memory hierarchy, in that frequently accessed instructions or data are kept in a memory local to the processor. Memory allocation can be done either by software or by hardware. In the former case the local memory is known as a set of registers, in the latter case it is known as a cache memory. Conceptually, these techniques only shift the place of the bottleneck. However, local communication or even on-chip communication can be faster and cheaper.

While today's 32-bit microprocessors promise to offer improved performance mainly because of the wider data bus, it seems that the performance of next generation microprocessors can only be increased by adding complex memory bus interfaces. Therefore, performance of announced microprocessors such as the NS32532 or the MC68030 is improved by integrating on-chip cache memories. A further improvement seems to be only possible by doing without the von Neumann structure. A continuation of the development may be the separation of the path used for the transfer of the data stream and the instruction stream by using separate buses. An actual representation of this concept is the Am29000. It may be added that in contrast to the enumerated CISC-processors recent developments of RISC-processors interestingly handle M without much care. A simple instruction format is used which increases M, but decreases the instruction decoding time and with that P.

Few publications are available that examine the bus traffic of von Neumann computers, both quantitatively and qualitatively. Quantitative specifications are required in order to choose an interconnection structure with an optimum cost/performance ratio. Qualitative specifications, i.e. the knowledge of the transferred information contents, indicate how the communication between processor and memory can be improved. It seems, that the rapid development of semiconductor technology does not dedicate any time to refine concepts, i.e. to analyse a concept and thereafter improve its realization. Instead, features are added that fill up a chip die. Nevertheless, a contribution to this subject are the studies made by Wirth and Heiz, which compare microprocessor architectures based on code generation by Modula-2 compilers [Wirth 86a, Heiz 87]. With the data given in their papers an estimation of bus traffic is possible. Considering the communication requirements of microprocessor architectures, it can be judged which architecture has the highest performance potential disregarding the technology taken for its realization.

The following measurings shall give an explanation of the low performance benefit gained by widening the data bus of the Ceres workstation. It shall also throw light on the dynamic behaviour of accessing memory of representative state-of-the-art 32-bit microprocessors, namely the NS32016 and NS32032 CPUs. It is not the intention of this chapter to compare microprocessor performances or even to enumerate benchmark results. Rather, a detailed analysis of bus utilization of a selected microprocessor family is given in order to judge the economical use of system resources.

5.2 Experimental Methods

The experiments use a simple test program that is contained in Appendix C. The original program was proposed by Wirth [Wirth 81a] and contains a selection of test sequences that measure various specific features of the language Modula-2. The experiments also include the well-known Dhrystone program [Weicker 84].

The experiments have been performed on two versions of Ceres, of which one contains a NS32016 CPU and the other a NS32032 CPU. Subsequently, they shall be denoted as C16 and C32, respectively. Both CPUs realize the same 32-bit architecture with the same full 32-bit internal implementation; they differ only in the widths of their data paths to memory [NS 84b]. The rest of the computer hardware is identical. Both machines are completely software-compatible: the same programs are executable on both machines without any adaptation or recompilation. Therefore, a comparison of the two versions involves only one variable: the width of the memory bus.

The measurings can be grouped into two categories. First, the performance of both versions is compared. The measurings are performed by simply counting the number of times the statements of the above mentioned test program are executed. Values have been determined for C16 and C32 with and without the inclusion of the memory management unit (MMU). If the MMU is present, a bus cycle takes five instead of four periods of the system clock. The additional period is required for the virtual to physical address translation. However, during the experiments the virtual addresses are interpreted as physical addresses, i.e. the MMU never has to access memory in order to get page table information.

Furthermore, the frequency and type of memory transfers are analysed. In order to monitor the memory bus operations, provisions in hardware have been added to both versions of

Ceres. A block diagram of the bus monitor hardware is outlined in Figure 5.1. A control register allows to select one of four signal groups to be examined, i.e. to be compared with a test vector that is also defined by the same control register. If the patterns match, a counter is incremented by one. Not shown is a free-running reference counter. The control register and the counters are software-programmable. The following conditions can be identified:

S=0: The number of system clock cycles during which the bus is assigned to the CPU, the display controller, and the refresh timer. Actually, only the measuring of CPU cycles is of interest, the other numbers can be calculated.

S=1: The width of data transferred during a memory read cycle (8 or 16 bits for C16; 8, 16, 24, or 32 bits for C32).

S=2: The width of data transferred during a memory write cycle (8 or 16 bits for C16; 8, 16, 24, or 32 bits for C32).

S=3: The bus cycle status code.

The measurings for S = 1, 2, 3 consider only bus cycles controlled by the CPU.

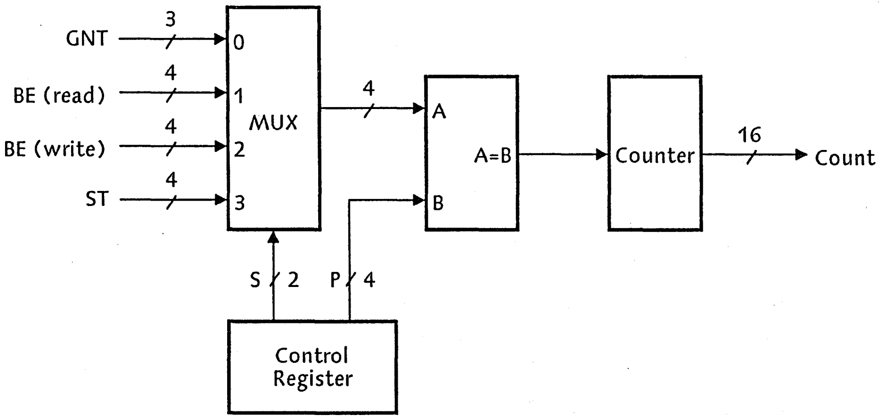


Figure 5.1 The bus monitor hardware.

5.3 Experimental Evaluations

Performance Comparison of C16 and C32

A first experiment compares the processing speed of C16 and C32. Figure 5.2 shows proportional numbers obtained by dividing the absolute numbers counted for C32 and C16. At first sight it is surprising that the speed advantage of C32 is only small. In general, slightly better results are obtained if the MMU is inserted. Programs that involve a lot of arithmetic calculations show a poor improvement of 6% to 15%. A remarkable improvement can only be determined for memory-intensive programs such as pointer handling which shows an improvement of 55%. It is of course naive to expect a doubled processing performance by doubling the bandwidth of the processor's data path. The processor uses the data path for fetching the instructions and for reading and writing information of the instruction's operands. The CPU's instruction look-ahead mechanism prefetches instructions into a queue and would only benefit of the wider data path, if the bus was heavily loaded, so that the queue is often little filled. Also, many operands are only 16-bit wide and do not make full use of the data bus width. Nevertheless, a better improvement is expected for programs that make exclusive use of 32-bit operands as is the case for copying arrays or LONGINT arithmetic, for which an improvement of only 39% and 15%, respectively, has been measured.

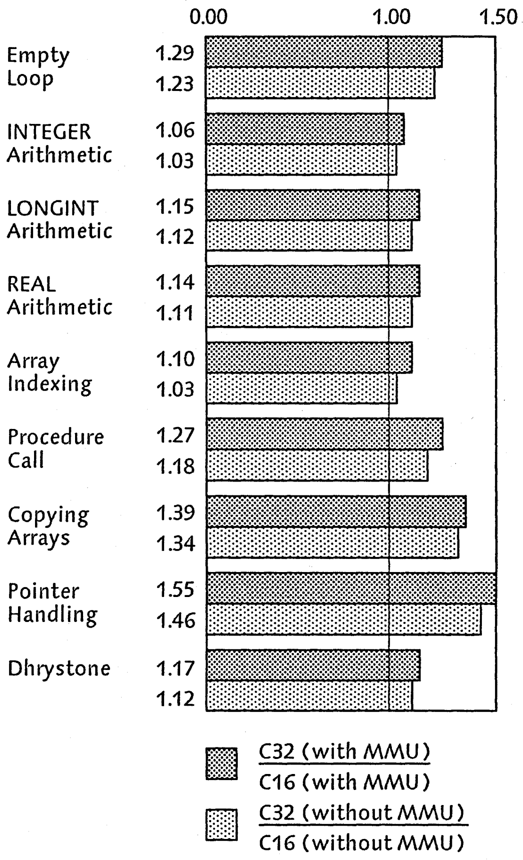


Figure 5.2 Performance comparison of C16 and C32.

Performance Loss due to Address Translation

The influence of using an MMU has also been measured and is shown in Figure 5.3. This figure can also serve as an answer to the question: What decrease of performance is to be expected when memory cycles are extended by one additional system clock period? It is not the subject of this section to discuss the functional impact of providing memory management as seen from the system programmer. For C16, performance is lowered by 7% to 17%, while for C32 only a decrease of 4% to 11% is noticed. Mainly for C32 this is far below the worst case of 20%. Based on the following studies, an explanation of this observation is provided: especially the CPU of C32 puts a light load onto the bus; about half of all these bus cycles are instruction fetches that fill the instruction queue. As is its purpose, the instruction queue lets the instruction stream become decoupled from the bus bandwidth including the length of a bus cycle. Therefore, only the few transfers of operand information can be slowed down by an extended memory cycle.

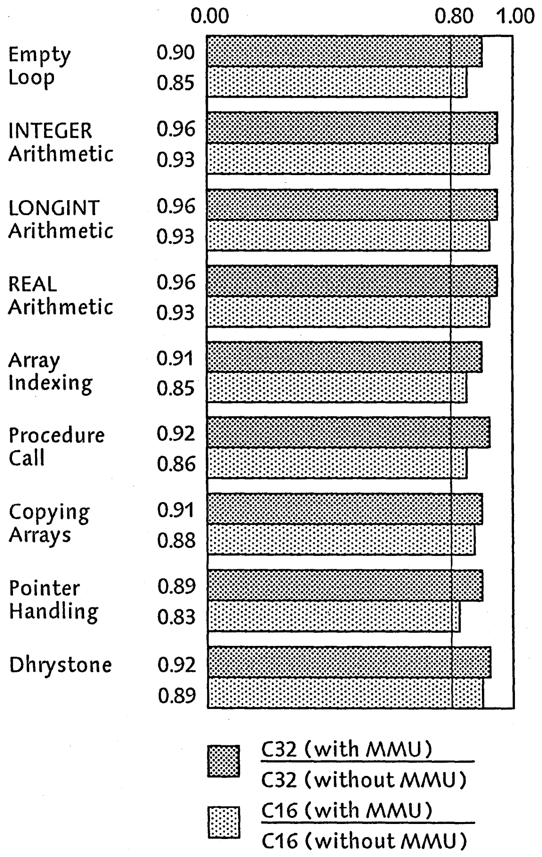


Figure 5.3 Performance loss due to address translation.

The next generation NS32332 CPU has the ability that the timing states T2 and TMMU (address translation cycle) are overlapped [NS 86b]. Consequently, a bus cycle can be performed in four system clock periods. Higher expense has to be paid for a memory with an access time of one clock period, which is 66.7 ns at 15 MHz; the performance advantage can be estimated with the shown figures. The use of this feature is questionable and suggests that designers should evaluate these "improvements" before advertising them.

Bandwidth Requirements

A more detailed analysis is obtained by using the described bus monitor hardware. The bus bandwidth is shared by several master devices. The portions taken by the refresh timer and the display controller can be calculated and have been verified by measurings. For the refresh timer it amounts to 2.5%, for the display controller it amounts to 0.25% on C32 and to 0.50% on C16, where the video memory has to be accessed twice as often as on C32. For the following considerations these contributions are negligible.

The portion absorbed by the CPU is shown in Figure 5.4. As presumed earlier, the bus of C32 is badly utilized: it is occupied by the CPU during 19% to 71% of time. A low utilization is given by programs that involve arithmetic calculations, while a higher utilization is achieved by programs that cause numerous memory references as in the case of pointer handling. Taking into account, that the bus width is not fully used for all transfers, the actual bus utilization mainly of C32 is further decreased. The corresponding figures are also shown in Figure 5.4.

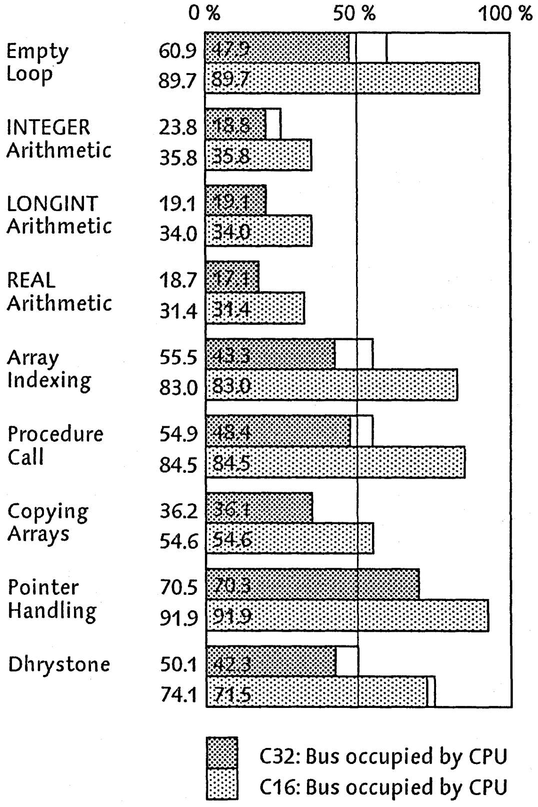


Figure 5.4 Bus utilization of C16 and C32.

Utilization of Bus Width

A table of the widths of bus transfers is shown in Figure 5.5 and was also included in the calculations of Figure 5.4. Instruction fetches count as accesses that use the full bus width, which is true for sequential instruction fetches but not for non-sequential instruction fetches resulting in a slight falsification of the statistics. Accesses that do not use the full bus width are explainable by the size of program variables as, for example, the programs for INTEGER (16-bit operands) and LONGINT (32-bit operands) arithmetic document. The bigger the difference between the widths of the smallest accessible data item and the data bus, the less efficient the utilization of the bus.

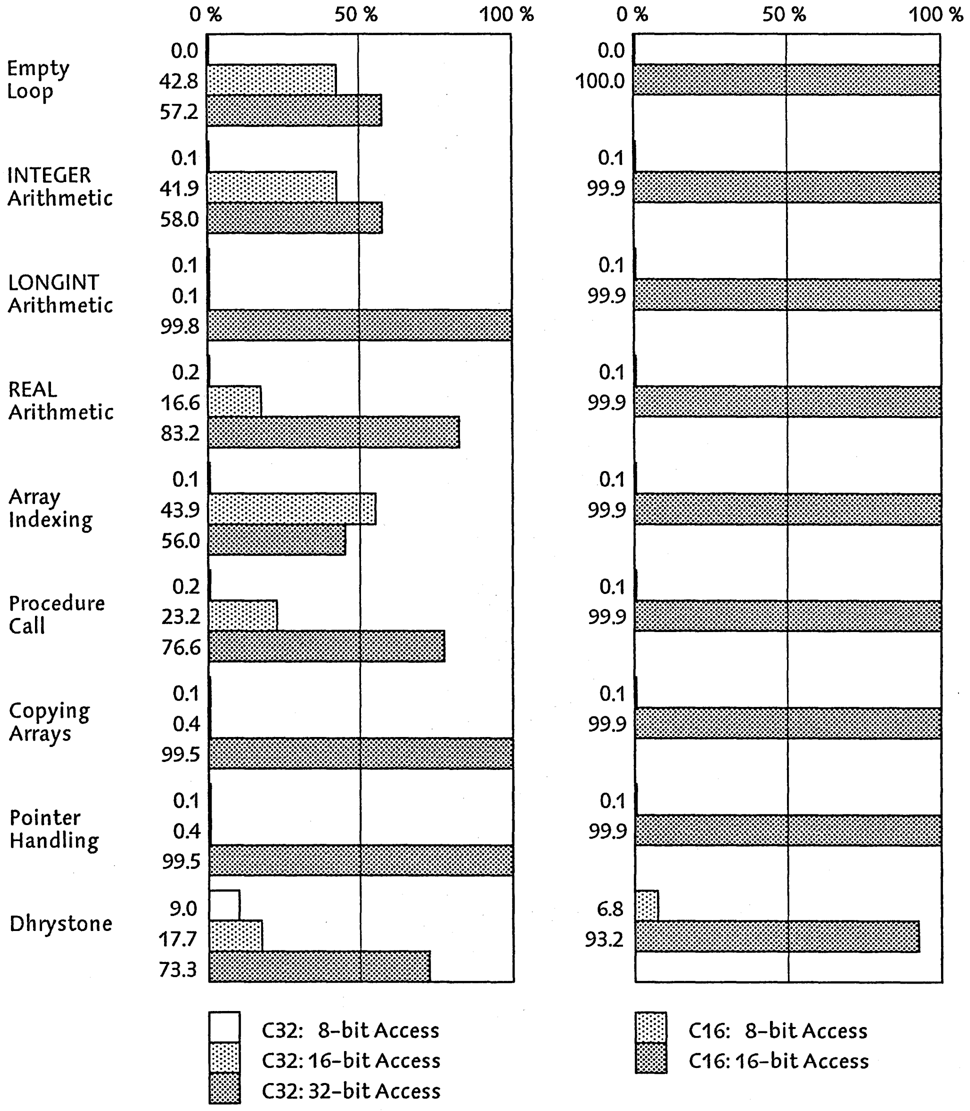


Figure 5.5 Utilization of bus width.

Type of Transferred Data

By interpreting the processor's status code, which accompanies every bus cycle, the type of bus cycle can be determined. The table of Figure 5.6 shows the following bus cycle types [NS 86a]:

- Instruction Fetch: The CPU is reading a word from the instruction stream (16 bit for C16, 32 bit for C32).
- Data Transfer: The CPU is reading or writing an operand of an instruction.

- Read RMW Operand: The CPU is reading an operand which will subsequently be modified and rewritten.
- Read for Effective Address Calculation: The CPU is reading information from memory in order to determine the effective address of an operand. This will occur whenever an instruction uses the Memory Relative or External addressing mode.

Except for copying arrays, the figures show similar values for all test programs. The CPU of C32 uses about 50% to 60% of all bus cycles in order to fetch instructions, while for C16 values between 53% and 73% have been measured. All other cycles are used for data transfers. Address information of an operand is read from memory for accessing pointer variables.

An explanation has to be added for the test program that copies arrays. As seen in Appendix C, the generated code contains a move string instruction, which makes it possible to copy a whole block of data, in our case consisting of 128 elements of 32-bit length, with one single instruction. Therefore, nearly all bus cycles can be used to transfer data. Because the full data bus width is used, it may be expected that this example fully exposes the bandwidth advantage of C32. Still, Figure 5.2 has shown a performance benefit of only 34% (without MMU). The explanation has already been presented in Figure 5.4, where it can be seen that in this case the CPU of C32 uses about one third of the available bandwidth, more exactly only 36% of all possible bus cycles are used by the CPU.

Direction of Bus Transfers

In Figure 5.7 the direction of bus transfers is analysed. Most read cycles are due to instruction fetches. Their proportion has been presented in Figure 5.6. Thus, the number of cycles can be determined that are used to either read an operand or read information from memory to determine the operand's address. While studying these figures it must be remembered that the task of a program is to change the state of the machine which is represented by the computer's store. The write cycles can be looked at as essential, while most other bus traffic is computation overhead required to calculate the address and value of a memory datum [Backus 78]. However, the dramatic discourse of Backus cannot be confirmed by comparing the numbers of read and write cycles, especially, if instruction fetches are neglected.

Utilization of the Instruction Queue

A comparison of the program code length and the total length of the actually fetched instruction stream shows that at the end of a program loop, i.e. when the branch instruction is executed, the 8-byte instruction queue of both, C32 and C16 is filled for most of the time. As the corresponding measurings in Figure 5.8 show, the instruction look-ahead mechanism works satisfactorily independent of the bus width. The wider bus of C32 certainly lowers the number of cycles needed for fetching instructions, but the freed cycles are not used elsewhere. Increasing the size of the instruction queue as done in the NS32332 CPU is a questionable enhancement, especially because the average total instruction length of compiler generated code is 3.6 bytes [Wirth 86a].

An optimization of bus usage due to instruction fetching may be appropriate for several reasons. Addresses emitted in order to fetch instructions can be characterized as being

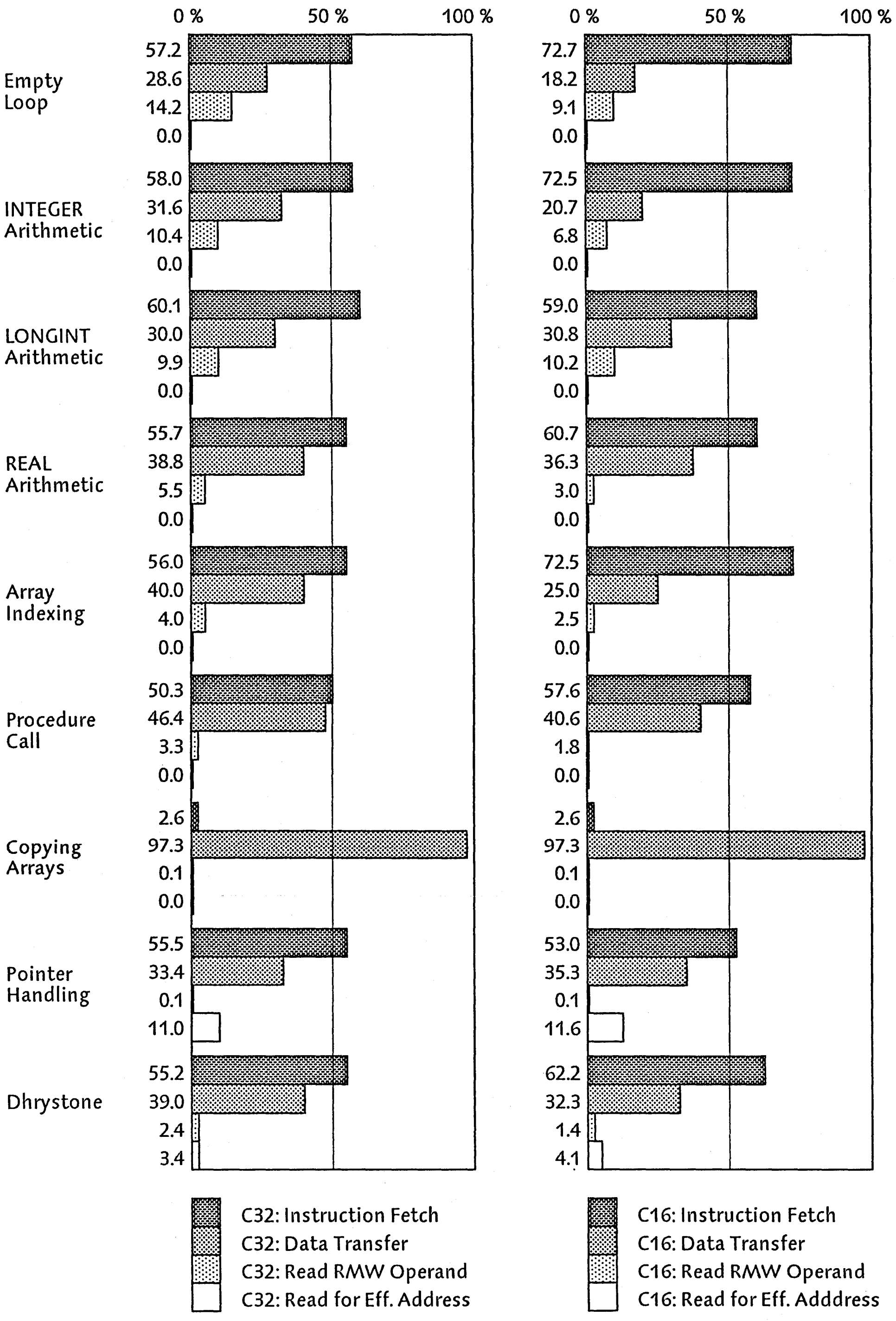


Figure 5.6 Type of transferred data.

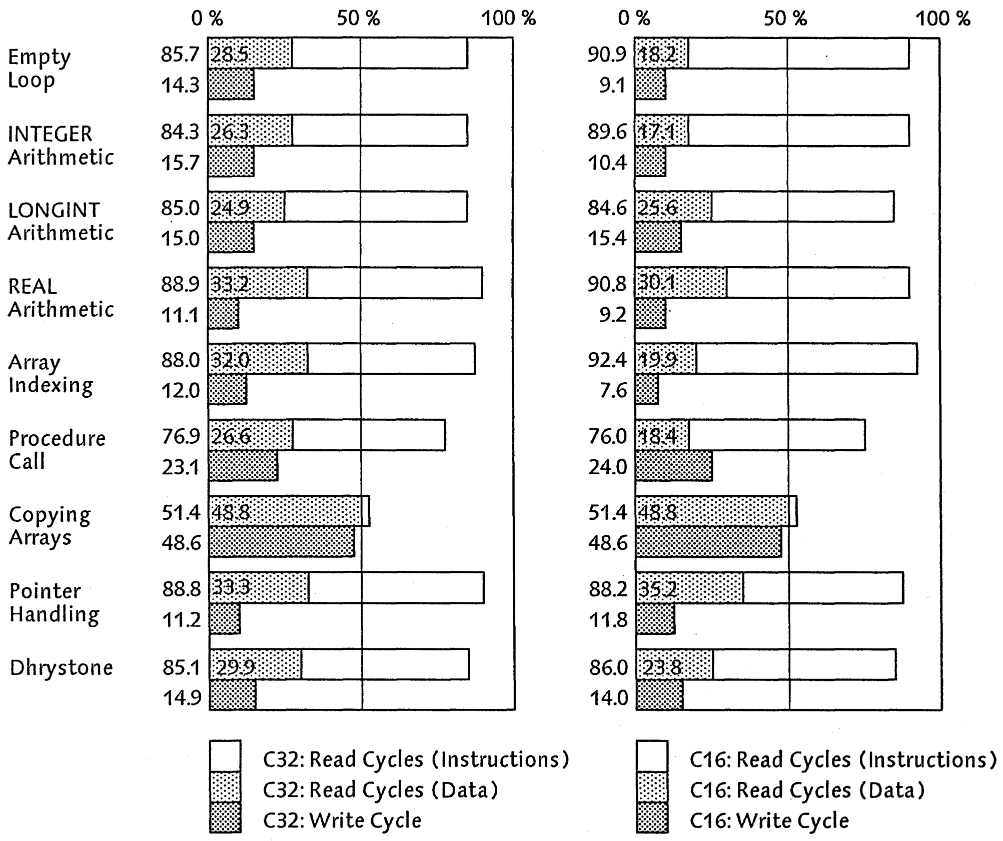


Figure 5.7 Direction of bus transfers.

sequential and often repetitive. A significant reduction of bus usage could be achieved by placing an instruction cache between the CPU and the main memory. Although an instruction queue can be looked as a cache, its look-ahead mechanism only derives benefit from the sequential character of instruction addresses in that the execution time of sequential instructions is not delayed by instruction fetches. With a real cache, also the

	C16	C32
Empty Loop	4	4
INTEGER Arithmetic	6	8
LONGINT Arithmetic	6	8
REAL Arithmetic	8	8
Array Indexing	6	4
Procedure Call	6	6
Copying Arrays	7	6
Pointer Handling	6	8

Figure 5.8 Remaining instruction bytes at the end of a program loop.

repetitive character of instruction addresses is considered thus reducing delays caused by program branches and reducing the number of memory references. An instruction cache can be implemented efficiently by using a direct mapping technique [Hayes 79]; also, the CPU only has to see a read-only memory, supposing that self-modifying code is not permitted. For the shown test programs more than half of all bus cycles could be saved. Note that local memory for data storage is already supplied by a set of registers. As data references are less predictable, the allocation of registers is best done by the compiler with the knowledge of the program context. Adding another level to the memory hierarchy in form of a data cache with a hardware controlled allocation strategy is conceptually contradictory at the least.

Exemplary Analysis of Instruction Execution Times

The presented figures can also be corroborated by the data given in [NS 84a], which can be used to calculate the instruction execution time. The specified data are average values based on certain given assumptions: the specifications mainly ignore instruction fetches in that it is assumed that the entire instruction is present in the instruction queue; it is further assumed that operand transfers do not overlap other operations. This is a pessimistic statement, as under certain circumstances overlapping is possible. It is noteworthy that it is not possible to determine the best and worst case of the execution time of an instruction sequence. In order to better understand the previously criticized results, those test programs shall be examined that make use of 32-bit operands, which applies to LONGINT arithmetic, copying arrays, and pointer handling. The respective relevant instructions are analysed in Figure 5.9. The timing characteristics are given in clock cycles and are listed according to the formula given in Section 5.1. The shown numbers for the MOVS instruction are related to the copying of one string element. Again, these figures confirm that for the NS32016 CPU and NS32032 CPU the processing overhead P is not negligible. Therefore the equation $T_E \simeq M/B$ is not even approximately accurate. The portion of execution time needed to access memory operands is also shown. For C16 it comes to $M/B = 0.11 \cdot T_E \dots 0.71 \cdot T_E$ and for C32 to $M/B = 0.09 \cdot T_E \dots 0.55 \cdot T_E$. Once more, these numbers show that doubling B on C32 can only have little impact, as also the comparison of the execution times of C16 and C32 in the table of Figure 5.9 illustrates.

A further note may be added. The assumption that on C32 32-bit operands can be used at the cost of 16-bit operands is, of course, not true: Mainly arithmetic operations such as division and multiplication are dependent on the operand length, as the comparison of the execution times for MULW/MULD and DIVW/DIVD show.

Data Alignment

The instruction execution time depends on the number of memory references, the memory bandwidth, and the processing overhead. The number of memory references due to instruction fetches can be kept low with a dense encoding of the instructions, i.e. of both the opcode and the displacement. The NS32000 instruction set achieves a high code density by encoding every instruction as a byte stream with a length that varies from one byte to 22 bytes in increments of one byte. Comparing available microprocessor architectures the code density achieved with the NS32000 instruction set is far ahead of other available microprocessors [Wirth 86a]. In order to further improve the encoding efficiency, the NS32000 CPUs allow not only instructions but also data (no matter of what type of data) to

		C16				C32				
		$\frac{M}{B}$	P	T_E	$\frac{M}{B} \cdot \frac{1}{T_E}$	$\frac{M}{B}$	P	T_E	$\frac{M}{B} \cdot \frac{1}{T_E}$	$\frac{T_E\ (C16)}{T_E\ (C32)}$
LONGINT Arithmetic	MULW (1)	12	54	66	0.18	12	54	66	0.18	1.00
	MULD (1)	24	86	110	0.22	12	86	98	0.12	1.12
	DIVW (1)	12	97	109	0.11	12	97	109	0.11	1.00
	DIVD (1)	24	129	153	0.16	12	129	141	0.09	1.09
Copying Arrays	MOVSW (1)	8	11	19	0.42	8	11	19	0.42	1.00
	MOVSD (1)	16	11	27	0.59	8	11	19	0.42	1.42
Pointer Handling	MOVD (2)	24	10	34	0.71	12	10	22	0.55	1.55

Addressing Modes: (1) instruction s(FP), d(FP)
(2) instruction s2(s1(FP)), d(FP)

Figure 5.9 Examples of instruction execution times.

be aligned to any byte boundary. As a consequence, depending on its size and its alignment an operand may be split and transferred with several bus cycles. This feature is open to question as it increases the number of memory references due to data transfers. Therefore, the Modula-2 compiler for Ceres aligns 16-bit operands to word boundaries (address modulo 2 equal 0) and 32-bit or 64-bit operands to double-word boundaries (address modulo 4 equal 0). The impact of data alignment onto program execution speed is illustrated in Figure 5.10: the execution of the test programs is between 8% and 35% slower for C32, and between 7% and 27% slower for C16, if program variables are not aligned (assuming the worst case, where all variables are dealigned).

While a dense encoding of instructions increases the efficiency of both the memory and bus utilization, this does not apply to the encoding of data. Allowing to align data on any byte boundary results in a compact utilization of the available memory space, mainly significant for storing large data structures, but increases the number of bus cycles and delays program execution. Therefore, an alignment restriction is preferred, although a slightly more complicated memory model for the allocation of memory data by the compiler is caused. (Both requirements, an efficient bus and memory utilization could be achieved with a fully aligned bus, where data are transferred by using the full bus width for any byte address. However, this scheme is too complicated to be realized.)

5.4 Cost Analysis of the Memory Bus

This section tries to estimate the costs that are due to the width of the memory bus. Because the course of development of C16 and C32 has not reached the same state - C16 was only realized as a prototype, while C32 has been produced in small quantities - a detailed cost analysis cannot be given.

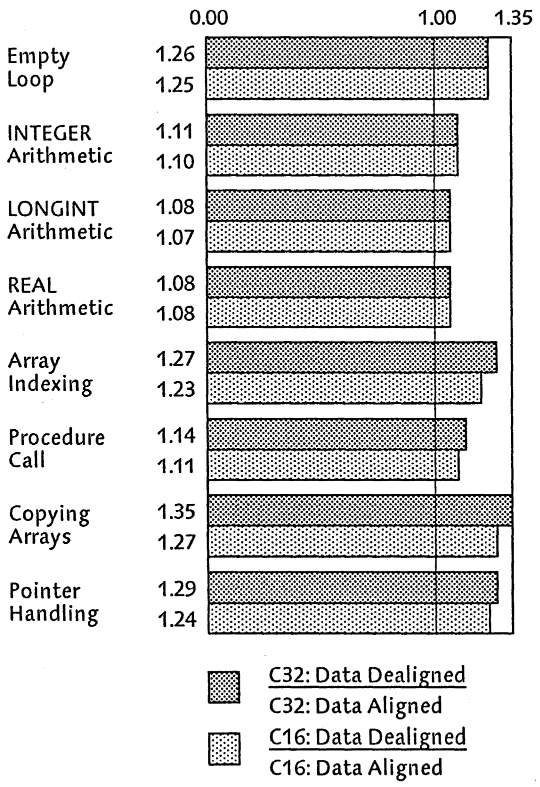


Figure 5.10 Data alignment.

Costs are caused during development and manufacturing. The Ceres project is certainly dominated by development costs. Furthermore, the development of C16 to C32 initially seemed to be a trivial task. Conceptually this is true, however, the realization produced unforeseen difficulties. The simultaneous switching of 32 data line drivers induces noise on power and ground distribution lines. This problem can be remedied, among other precautions, by lowering the inductance of power distribution lines. Therefore, the printed circuit boards had to be redesigned, now using two complete layers for power distribution. The main trouble, however, was caused by faulty CPU-chips, which delayed the development process by months. It must be stated openly, that if the NS32032 CPU had been incorporated into the first design of Ceres, the whole project would have exhausted our patience and would have failed.

Manufacturing costs are influenced by the number of parts and connections. Comparing the buses of C16 and C32, 20% more signal lines are required for the wider data bus of C32. To the same extent, the component count of the bus interfaces is increased. Additional line drivers and receivers are required, which again require additional board space and increase power consumption.

The width of the memory bus also determines the organization of main memory. Memory ICs are organized as n x m bit arrays, where n is the number of addressable word-storage locations and m is the word size. If an N x M bit memory array is to be built from n x m bit memory ICs, then at least

$$k = \frac{N \cdot M}{n \cdot m} \quad (N \geq n, M \geq m)$$

components are needed. Figure 5.11 shows the organization of commonly used memory ICs and memory modules that are constructed from a minimum number of ICs. The values chosen correspond to the ones of C32. The minimum capacity of a memory module is dictated by the bus width and may be more than basically needed. This is true for the VRAM- and the ROM-memory modules. A solution of this problem may be a feature called dynamic bus sizing, which supports operation with 8- or 16-bit buses and is found for the MC68020, I80386, and NS32332.

	n	m	N	M	k	N · M
DRAM	256K	1	256K	32	32	1M
VRAM	64K	1	64K	32	32	256K
ROM	8K	8	8K	32	4	32K

Figure 5.11 Organization of memory ICs and memory modules.

5.5 Conclusions

Comparing C16 and C32, only a low performance benefit has been gained by doubling the memory bus bandwidth. The analysis of processor-memory communication has explained the reason. The memory bus, in particular the one of C32, is only lightly loaded. The instruction execution time is dominated by internal processing overhead. Lowering the time needed to access instructions and operands from memory can, therefore, result in a low speed advantage only. Furthermore, the instruction stream cannot gain advantage from a wider bus because it is decoupled from the memory bus bandwidth by an efficiently implemented prefetch queue built into the CPU. Although the limitations of the von Neumann bottleneck have to be faced, the implementations of the analysed CPUs give room to much improvement. The badly utilized memory bus of C32 demands a reduction of internal processing overhead. Unfortunately, instead of offering internally re-worked CPU versions, so-called next generation CPUs are offered such as the NS32332 and soon the NS32532 with different system interfaces that are not compatible with older versions and, therefore, require considerable hardware changes to be made.

With this knowledge the increased manufacturing and development costs of C32 are disproportionate. The interpretation of the presented measurings shows that the redesign of Ceres, i.e. the replacement of the NS32016 CPU with the NS32032 CPU, cannot be justified and should have been omitted. The upshot of the foregoing is that it must be the application engineer's duty to critically and carefully evaluate not only his design, but also the built-in components.

6 Experiences with Complex Integrated Circuits

Packaging of computing functions plays a dominating role in implementing workstations. Highly integrated circuits allow reductions of the chip count, of board space, in power consumption, and, as a consequence, a lowering of the overall manufacturing costs. A landmark in the development history of integrated circuits is the one-chip microprocessor. It reduces the essentials of a computer to one inexpensive component and is the prerequisite for personal computing and the widespread distribution of computing facilities.

However, the costs of actually employing complex VLSI chips into a complete system are often neglected or underestimated. System designers can have hard times with the application of complex VLSI chips. Chip specifications are incomplete and obscure and often consist of pidgin-English-like formulations. While the hardware designer normally gets clear and precise information in the traditional form of timing diagrams and electrical characteristics, the software designer mostly faces unstructured and abstruse information. With the help of "application notes" and "technical notes" he tries to understand the datasheets. Serious complications arise if, after all, the chip does not operate as specified. To localize the problem can be extremely difficult as the chip appears as a black box that hides the explanation of its incorrect operation.

The development of the Ceres computer was delayed by several problems of the kind mentioned above. Three examples are given. We do not intend to criticize the named manufacturers in particular, but want to document our experiences with complex chips. Further examples can also be found in [Lyon 85]. Additionally, [Wirth 87b] is well worth reading.

- The NS32202 interrupt control unit (ICU) is a member of the National Semiconductor Series 32000 family. The ICU is an example of an unnecessarily complex chip: it features several programmable modes and options controlled by 32 registers. Although we could use a small subset of all possible functions only, this chip was initially chosen to be built into the prototype in order to have an interface for servicing interrupts that is consistent with the one of the CPU. However, the ICU did not work. After several days lost studying the specifications, installing the chip, and tracing the problem, the ICU was finally replaced. We decided on a different type of interrupt controller, the Am9519A, which is a comparatively simple chip. Months later, our observations were confirmed with the following user information note (User Information, NS32202 ICU, Revision E, June 6, 1984):

  [Quote] It has been found that the upper nibble of register 1 (software vector) gets spuriously transferred to register 16 (mode control). In particular, the bits which are set (1's) in the upper nibble of register 1 set the corresponding bits in register 16 to 1. This creates some undesirable effects in a NS32202 based system ... [Unquote]

  It must be added that the affected mode control register sets the operation mode of the ICU and is the first register that is accessed during the initialization sequence of the ICU. This explains that the ICU appeared to be completely uncontrolled.

- Revision F of the NS32032 CPU contains the following bug (User Information, NS32032 CPU, Revision F1, July 24, 1985):

[Quote] In executing the RETT or RETI instructions, the CPU will sometimes read the MOD register value from the wrong address, and also with incorrect byte order and/or missing bytes. The incorrect address will be offset by a small amount (+/- 3 bytes) from the correct address. The instruction continues by attempting to read the SB value from the incorrect address in the MOD register. This problem is associated with a specific set of timing sequences on the bus, involving HOLD/HLDA DMA and/or WAIT states. It can be bypassed by aligning the RETT instruction so that it can be fetched in one memory cycle, and disaligning the stack containing the return address so that the CPU must pop the return address in two (or more) memory cycles. Doing both of these together prevents the sequences that lead to this failure. This also applies to all previous revisions. [Unquote]

Revision F of the CPU was used in the second Ceres prototype and replaced the NS32016 CPU of the first prototype. We spent about two toilsome months in locating the described problem. While most smaller programs were executed without problems, larger programs including the operating system Medos-2 failed occasionally. At that time, we even had knowledge of the quoted error description, but had not seen the connection. We finally found that the mentioned timing sequence was given, if a memory access of the CPU was delayed through the insertion of WAIT states, as was the case if either the DRAM refresh timer or the display refresh controller was accessing memory.

- The realization of a disk controller even had to be abandoned. An inexpensive design seemed to be possible by using a single-chip disk controller, the HDC9224 from Standard Microsystems [SMC 85]. However, we were not able to make the controller operate as specified. We only succeeded in moving the head of the disk drive, but neither in reading or writing disk data. During the search for the problem, we received the following corrections (Addendum to Technical Note 6-5, August 14, 1985):

  [Quote] The Desired Sector register (register 3) uses standard binary notation rather than 1's complement ... The Sector Count register (register 6) uses standard binary notation rather than 1's complement ... [Unquote]

  The wrong notation was not specified by the original datasheet, it was later introduced by the technical note. Although the failure in building a disk controller probably has other reasons, the quotation is an example of the quality of datasheet specifications.

Note that the citations are only extracts of longer lists. Designating these lists with "user information" or "addendum to technical note" makes the circumstances look harmless. The wording of the bug reports appears cynical when the lost development time is considered.

The lesson to be learned by the system designers is that the application of complex chips can be troublesome. Caution is especially recommended if recently introduced chips are to be used. The complexity of a system is not reduced by the usage of highly integrated circuits, it is moved from the board level to the chip level only. Beyond it, the universality of most chips adds complexity. In conclusion, chip designers and manufacturers are requested to spend more time on design verification and to provide simple and comprehensible chip specifications in spite of chip designs for a market that appears feature-crazy and gadget-hungry.

7 Results

7.1 Summary of the Thesis

This thesis documents and analyses the design and implementation of the workstation Ceres. Following the table of contents the results can be summed up as follows.

Chapter 2 gives a hardware description of the workstation Ceres. The concise description of the implementation presents an example of a simple system architecture. The complete hardware of two Ceres prototypes was realized by the author in two years. A simple design is easy to test: the main debugging tools were a 100 MHz scope and a simple 10 MHz logic state analyzer. The hardware structure is easily manageable and comprehensible. This is especially appreciated by the software designer who writes system programs [Wirth 87a] and the hardware designer who adds hardware extensions.

Chapter 3 discusses possible realizations of raster graphics systems in general, and the raster graphics interface of Ceres in particular. The frame buffer of Ceres was one of the early applications of video RAM technology which proves to meet ideally the memory bandwidth requirements of high-resolution display monitors. Graphics functions on Ceres rely on the bitmap operator RasterOp. Their implementation is purely software-based, i.e. no hardware support is provided. The solution is inexpensive and offers maximum flexibility. The realization is based on the conceptual insight that graphics functions must be able to be applied to a uniform memory address space, where both the bitmap memory and the general memory are located. This concept is violated by many current designs that incorporate display processors equipped with local, isolated frame buffers.

Chapter 4 gives a classification of microcomputer buses, which is subsequently used for a discussion of the Ceres bus structure. The backbone of the Ceres computer is the memory bus. A semisynchronous transmission protocol is used to transfer data in a little-endian order on a demultiplexed and straight bus. The shared bus is controlled by a centralized arbiter with fixed priorities. The introduced arbitration method is optimized for short response times. The latency time until a request is granted is reduced by re-allocating the shared memory for every memory cycle. Furthermore, a default assignment strategy provides the most frequent bus requester, i.e. the CPU, with the privilege to initiate transfers on an idle bus without any arbitration delay.

Chapter 5 analyses processor-memory communication with respect to the NS32016-based Ceres and the NS32032-based Ceres. These versions differ only in the width of their data paths to memory. Timings show that in particular the bus capacity of the NS32032-based Ceres is only used to a small degree. Of the requested memory cycles, the most part is due to the instruction stream, which is, however, efficiently decoupled from the memory bandwidth by a prefetch queue built into the CPU chip. The disillusioning result of this analysis is that the performance benefit gained by doubling the memory bus bandwidth is small. The area where the wider bus shows most performance improvement is the generation of displayed data (RasterOp). The conclusion is that the increased manufacturing and development costs of the NS32032-based Ceres can hardly be justified. Positively formulated, much room would be given to improve the implementation of the NS32032 CPU without necessarily changing its system interface specifications.

Chapter 6 discusses our experiences of employing complex integrated circuits. The development of Ceres was delayed by several problems caused by faulty chips and incomplete or wrong chip specifications. Therefore, an improvement of chip design verification and the provision of simple and concise chip specifications are requested.

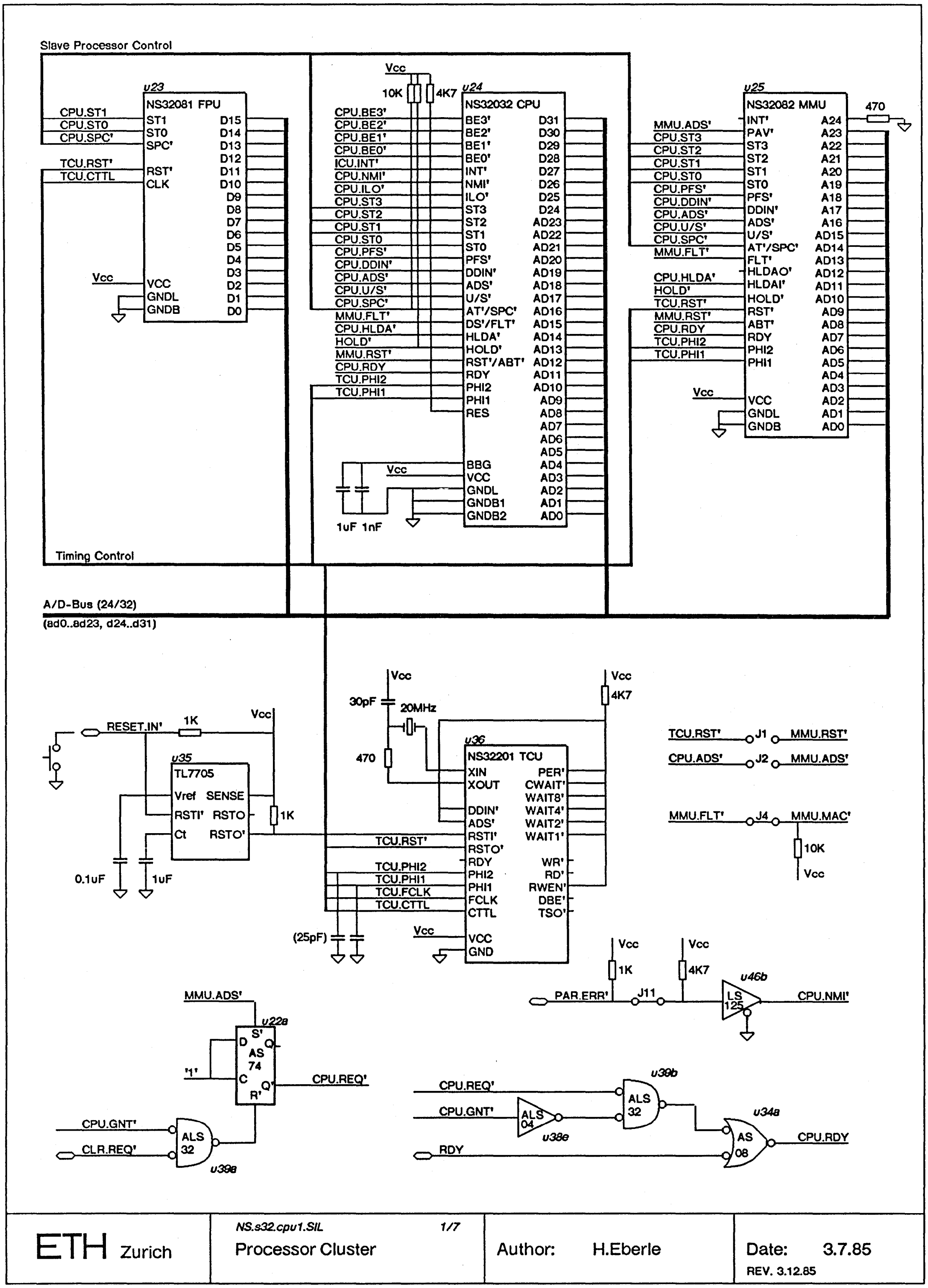
In summary, with this thesis we demonstrate that computer engineering is a systematic discipline providing the means to translate demanding design problems into precise implementations. It forces the designer to employ simple and clear concepts. The resulting design concentrates on essentials rather than on embellishments which do not contribute to the problem's solution. The ultimate test of an engineering project is its accurate implementation. The rewards of these efforts are reliable and efficient computer systems.

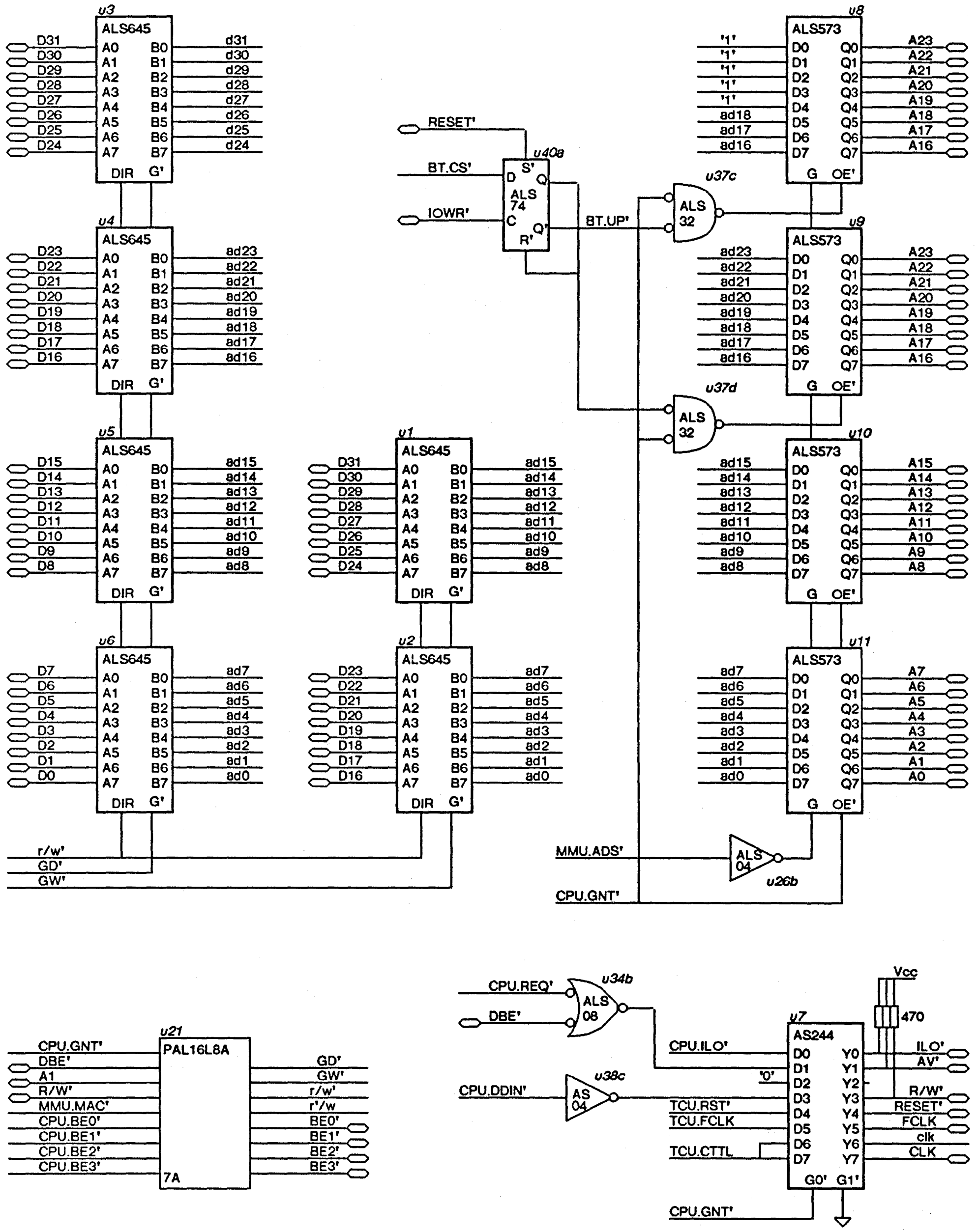
7.2 Practical Results

The Ceres project is the continuation of the effort to develop usable tools that support other research activities or the students' education. A first series of 30 computers has been in use since the end of 1986. Another series of 20 computers is currently being completed. The reliable operation of the machines is much appreciated. Hardly any repair or maintenance effort has been needed so far. No hardware modifications have been necessary, also the specifications of the first series have been maintained for the second one. The extensibility of Ceres has been demonstrated by the easy addition of a laser printer interface and a colour monitor interface.

Ceres is now taking over the place of its predecessor Lilith. In comparison with Lilith, Ceres is favoured by a halved component count. Further, Ceres requires only a fourth of the power consumed by Lilith. These savings add to both, lower manufacturing costs and higher reliability. The costs for material and manufacturing amount to approximately SFr. 10'000.- per machine in the produced quantities. Compared with Lilith, the main functional improvements or extensions are the larger memory address space, hardware support for virtual memory and real arithmetic, and the high-resolution, flicker-free display.

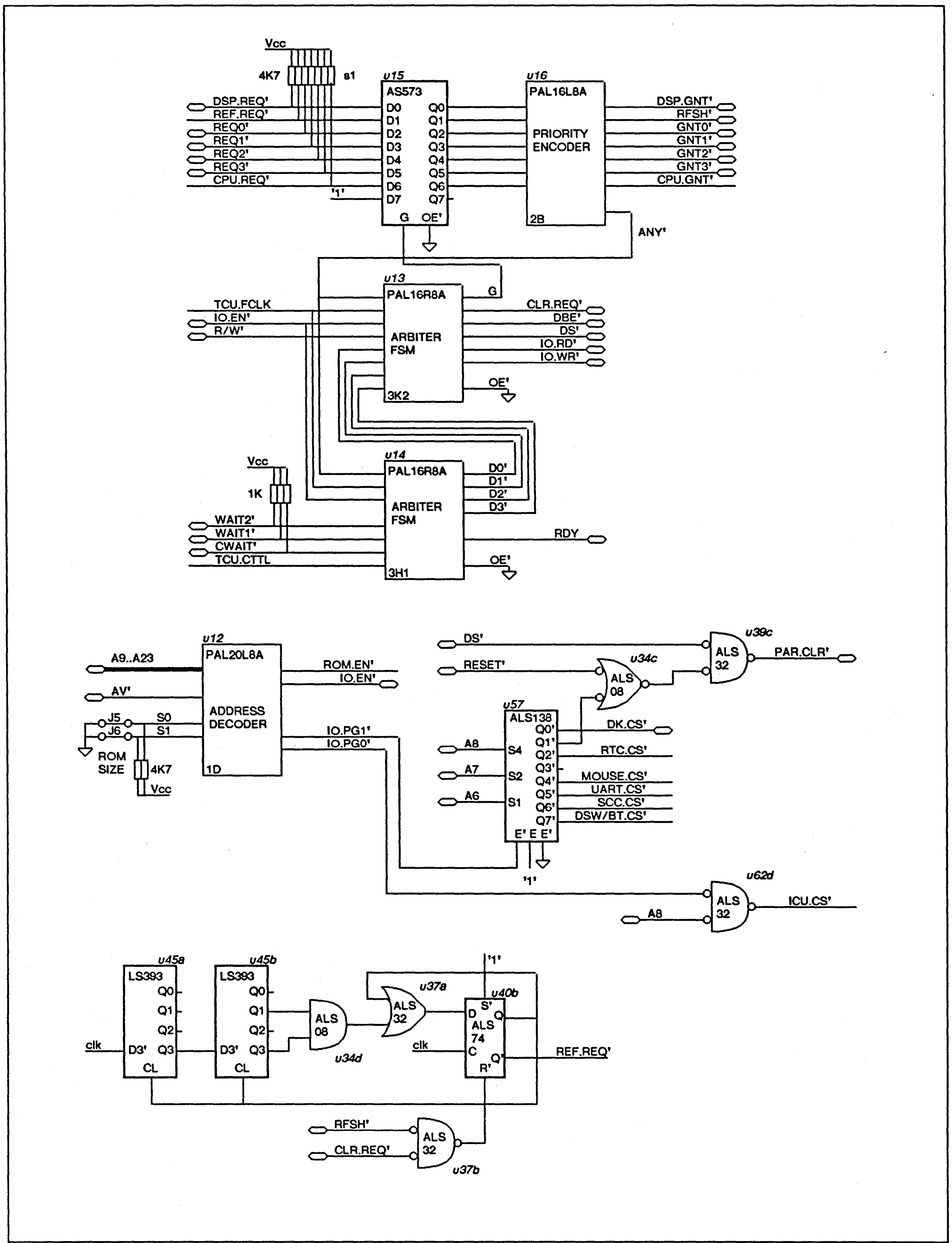
A.1 Processor Board



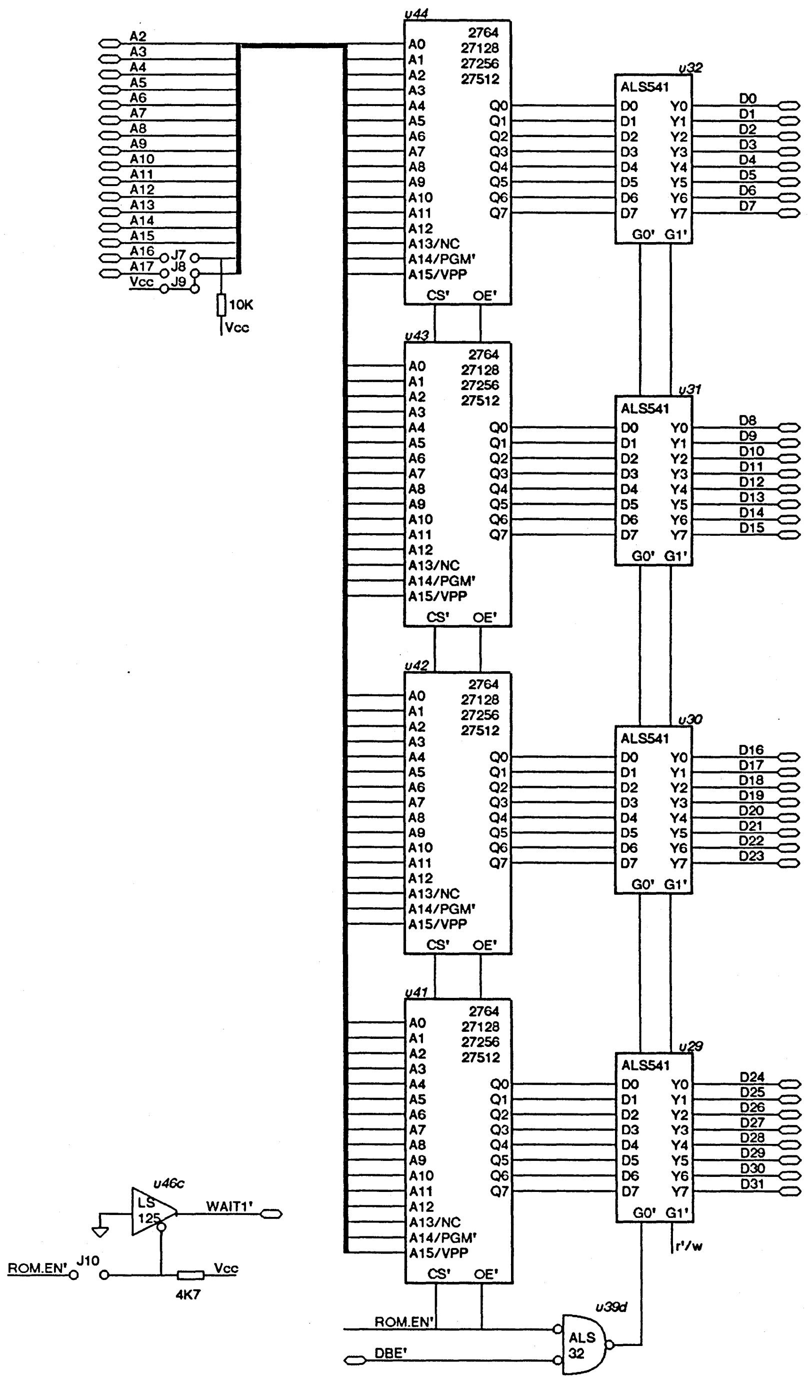


Termination resistors (270/560) are provided
for TCU.FCLK, TCU.CTTL, FCLK, CLK and clk.

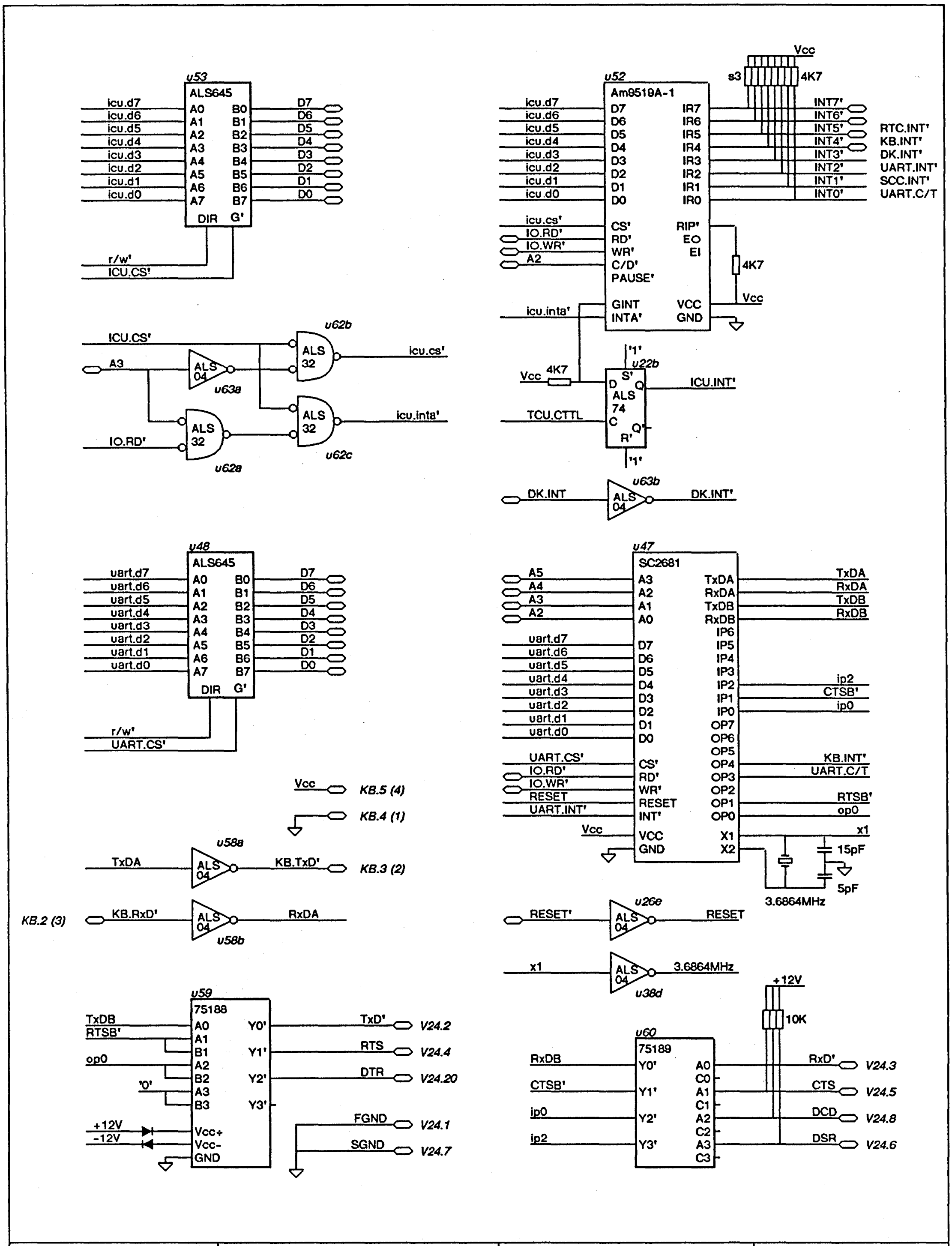
ETH Zurich	*NS.s32.cpu2.SIL* *2/7* Data Buffer, Address Latch Bus Control	Author: H.Eberle	Date: 3.7.85 REV. 21.5.86



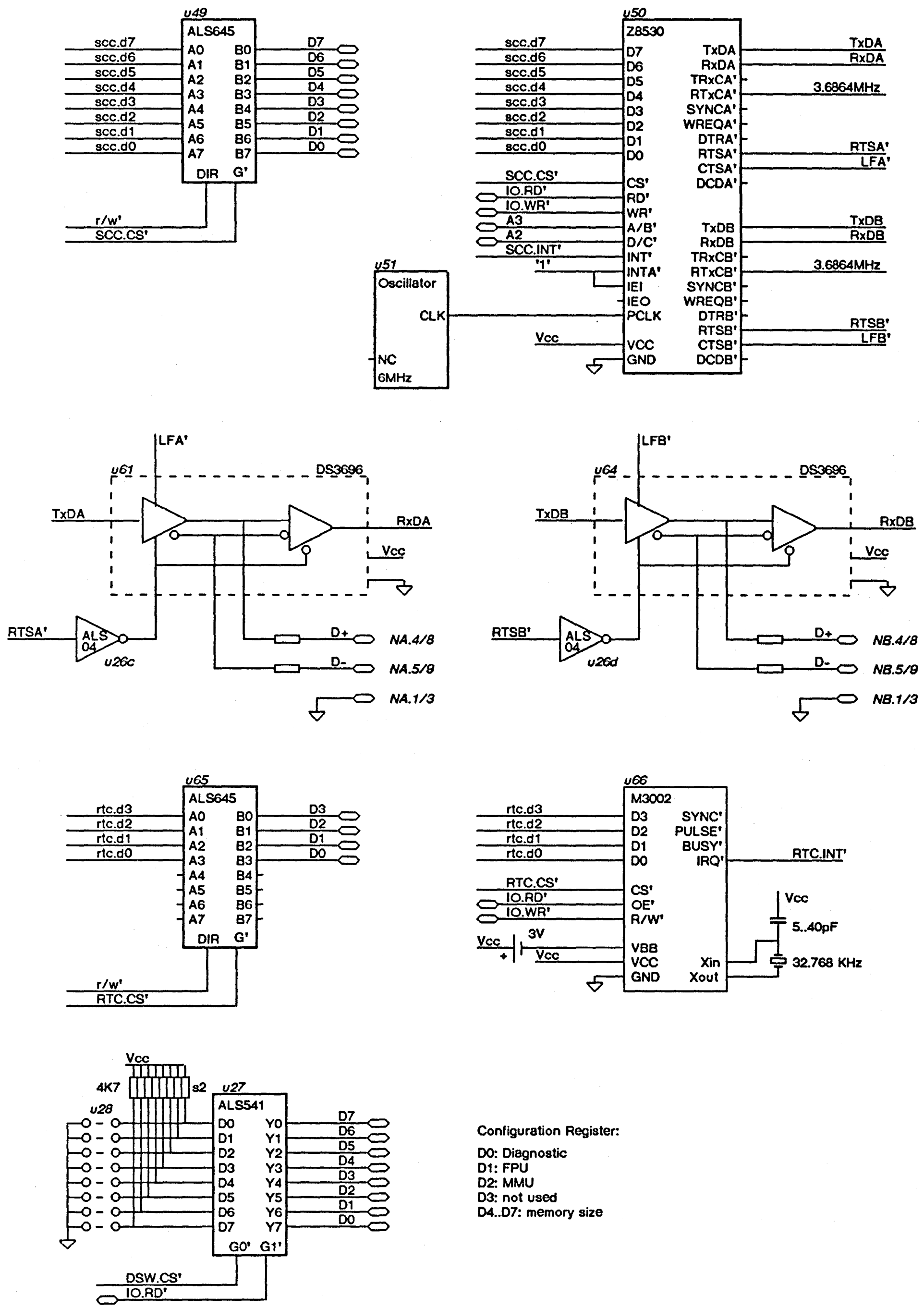
ETH Zurich	NS.s32.cpu3.SIL 3/7 Bus Arbiter, Addr. Decoder Refresh Timer	Author: H. Eberle	Date: 3.7.85



ETH Zurich	*NS.s32.cpu4.SIL* *4/7* ROM	Author: H.Eberle	Date: 3.7.85 REV 3.12.85

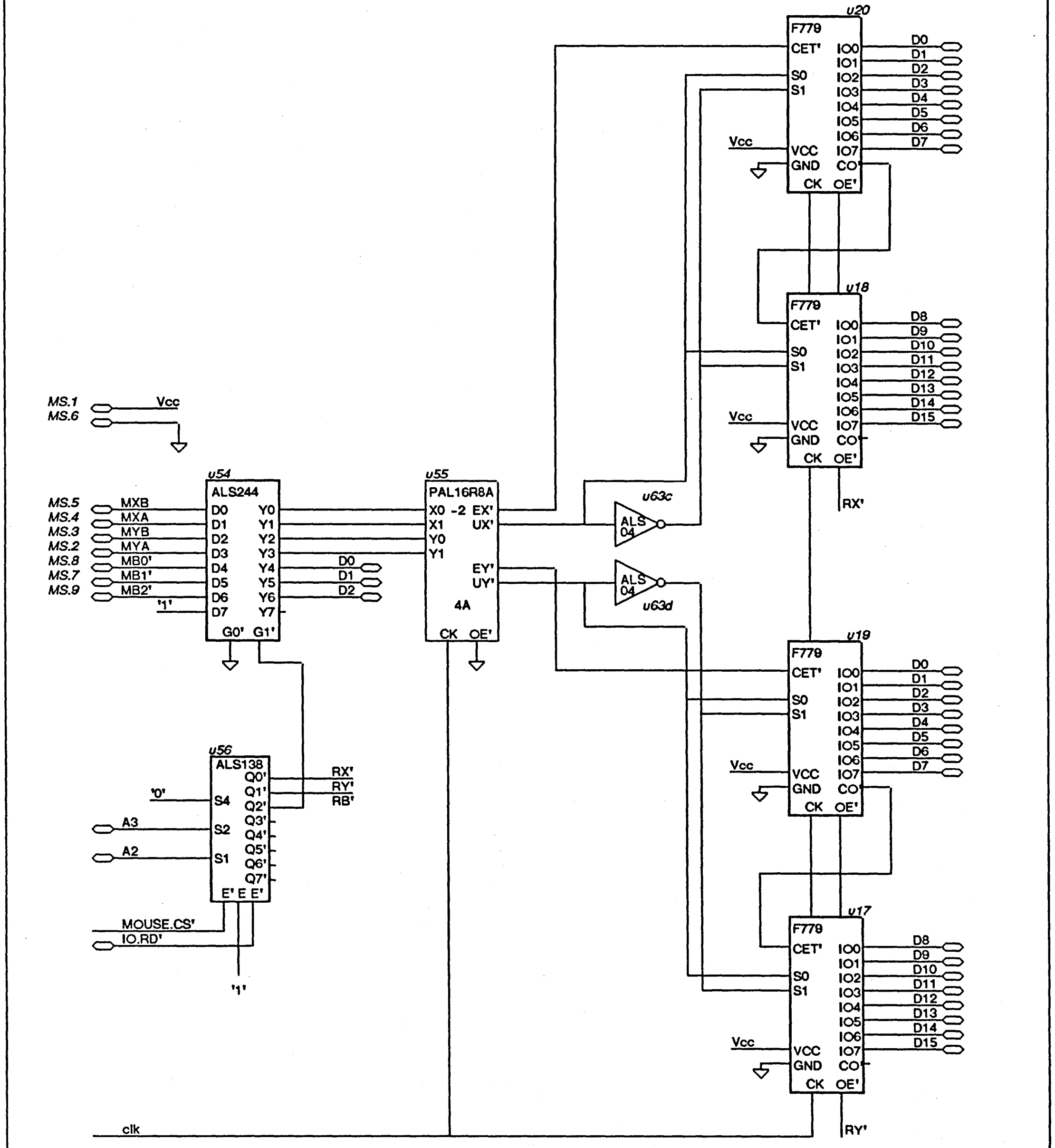


ETH Zurich	NS.s32.cpu5.SIL 5/7 ICU UART	Author: H.Eberle	Date: 3.7.85 REV. 21.5.86



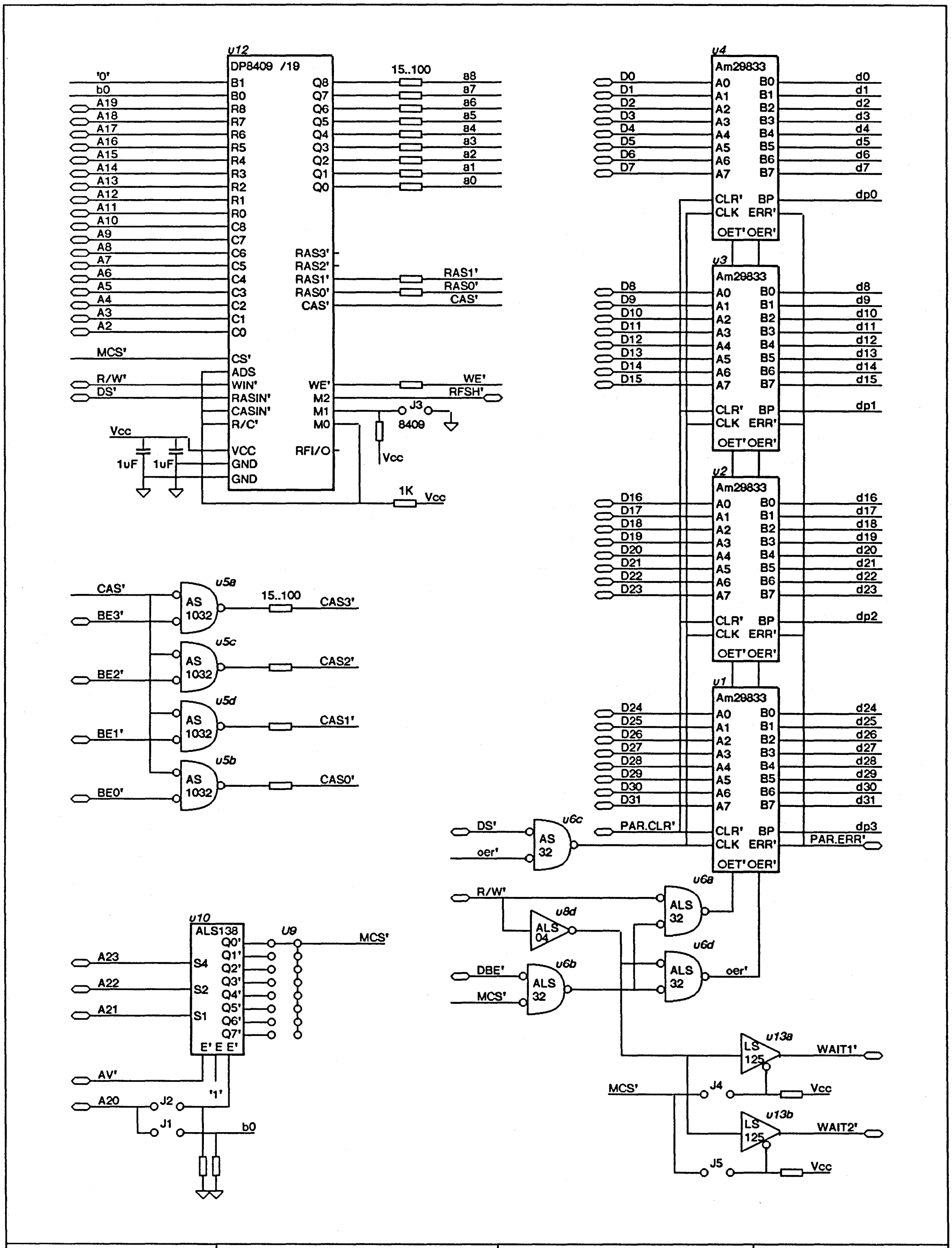
Configuration Register:
D0: Diagnostic
D1: FPU
D2: MMU
D3: not used
D4..D7: memory size

ETH Zurich	NS.s32.cpu6.SIL 6/7 SCC, RTC, DIP-switch	Author: H. Eberle	Date: 3.7.85 REV. 3.12.85

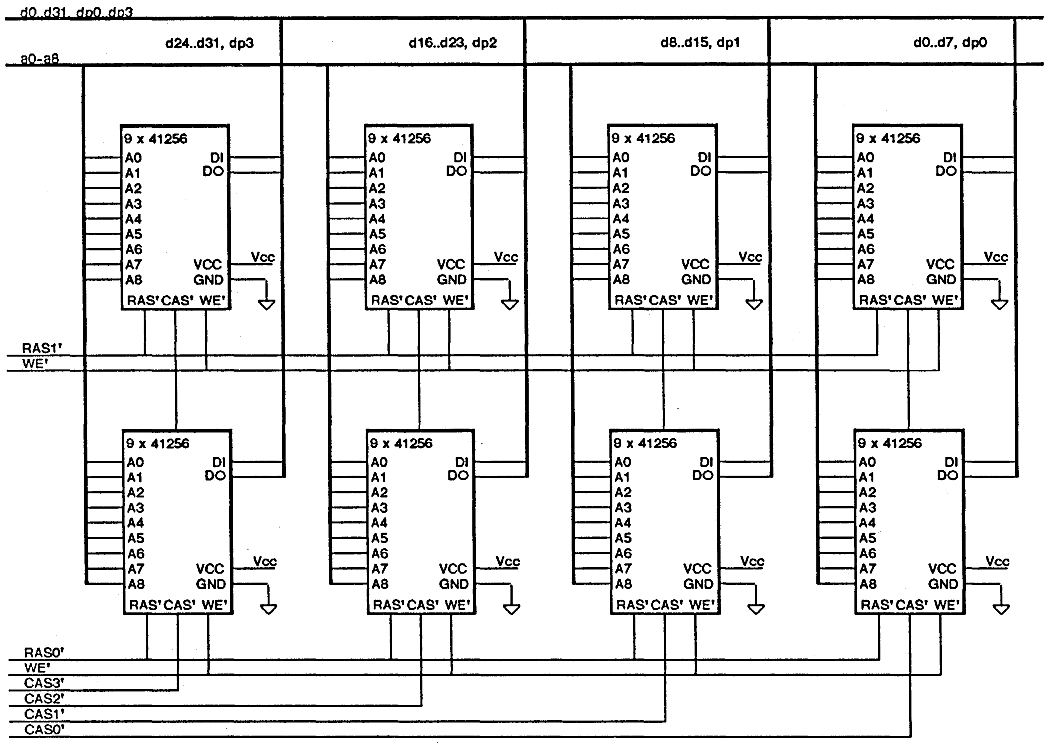


ETH Zurich	*NS.s32.cpu7.SIL* *7/7* Mouse-Interface	Author: I.Noack/H.Eberle	Date: 29.4.85 REV. 3.12.85

A.2 Memory Board

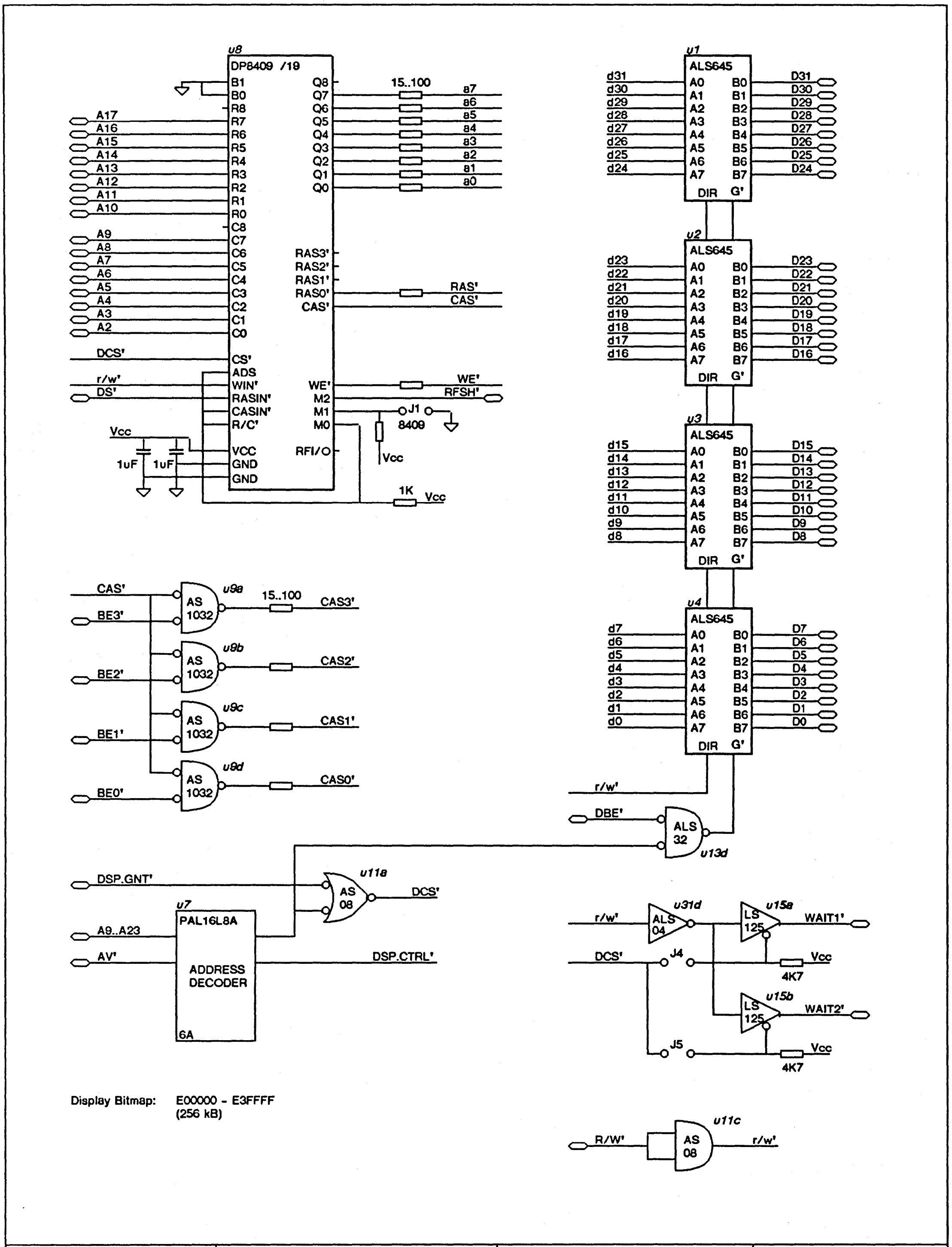


ETH Zurich	*NS.s32.mem1.SIL* *1/2* DRAM Controller Bus Buffers	Author: H. Eberle	Date: 8.4.85 REV. 9.10.85

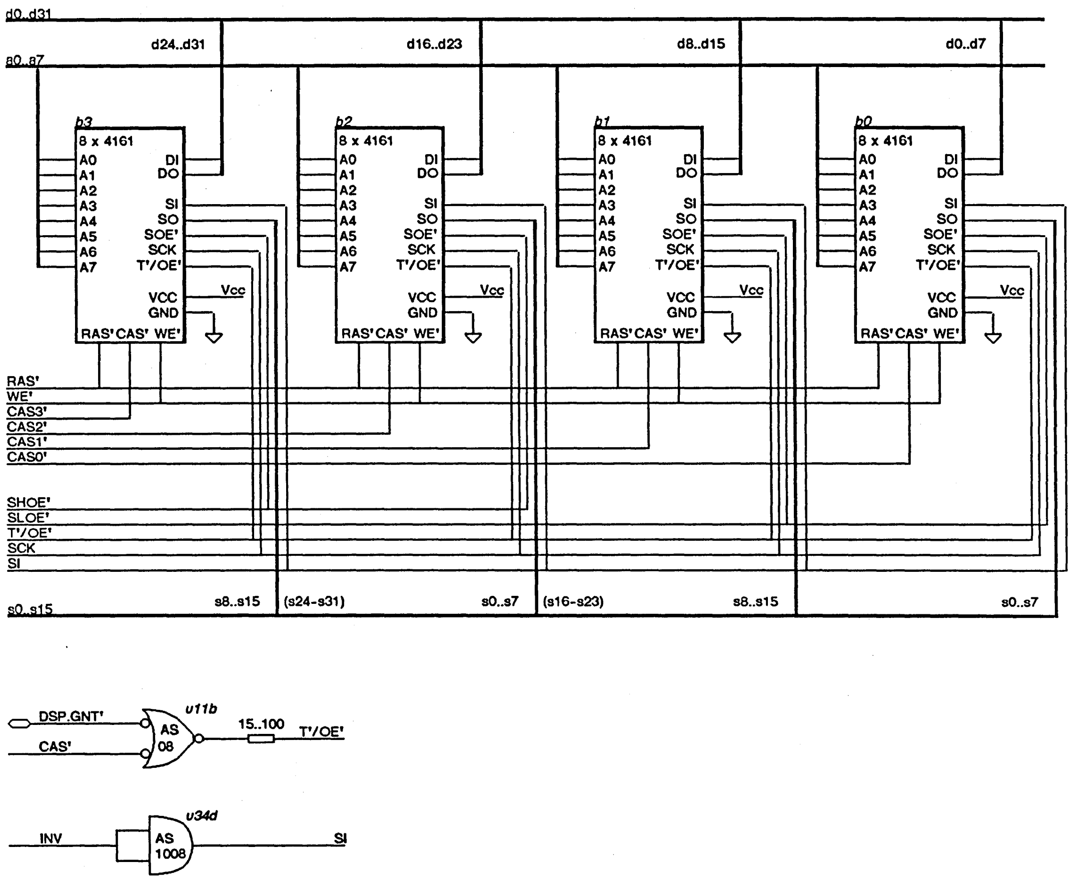


ETH Zurich	*NS.s32.mem2.SIL* *2/2* Memory (2 x 32 x 256 KBit)	Author: H. Eberle	Date: 29.3.85

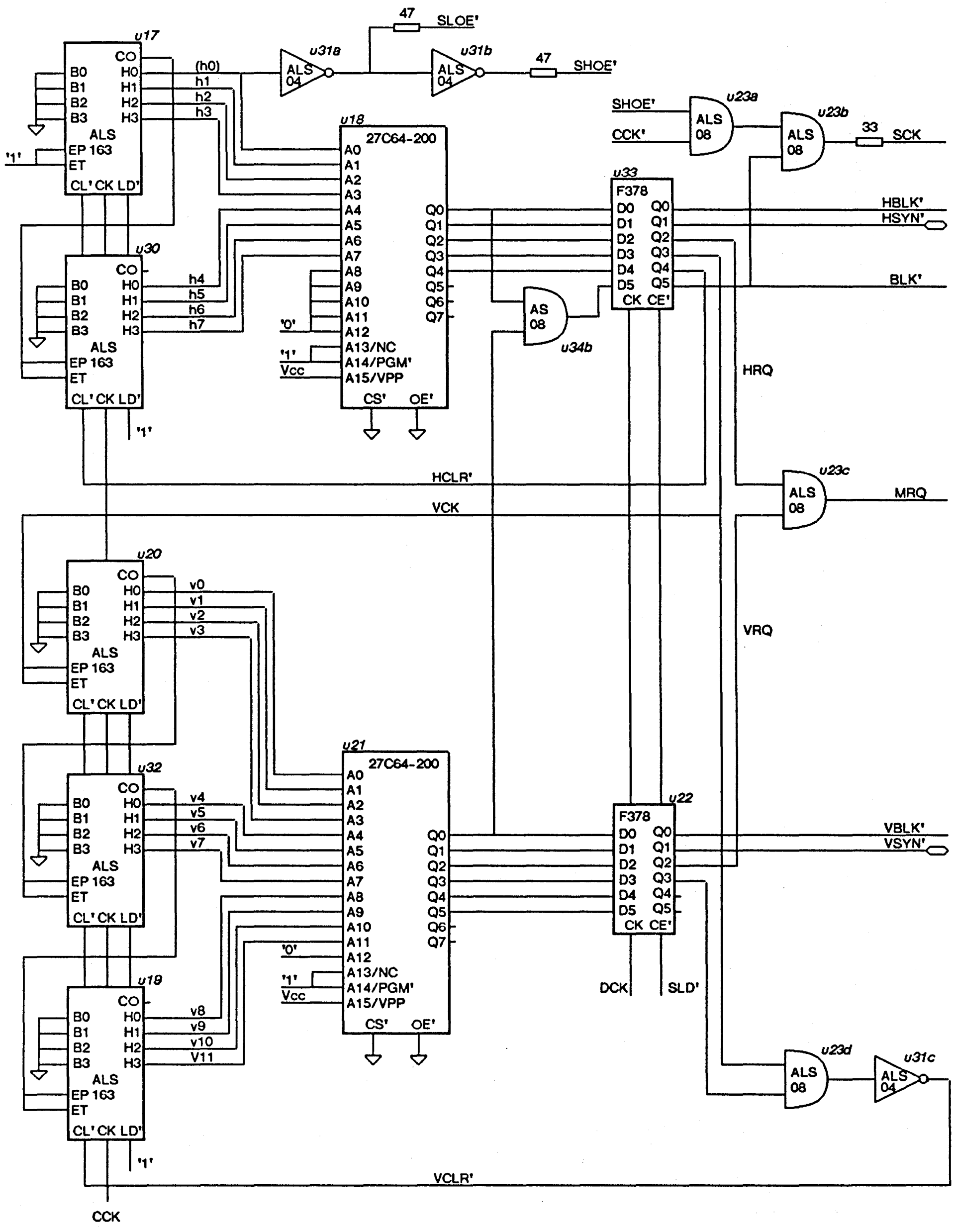
A.3 Display Controller Board



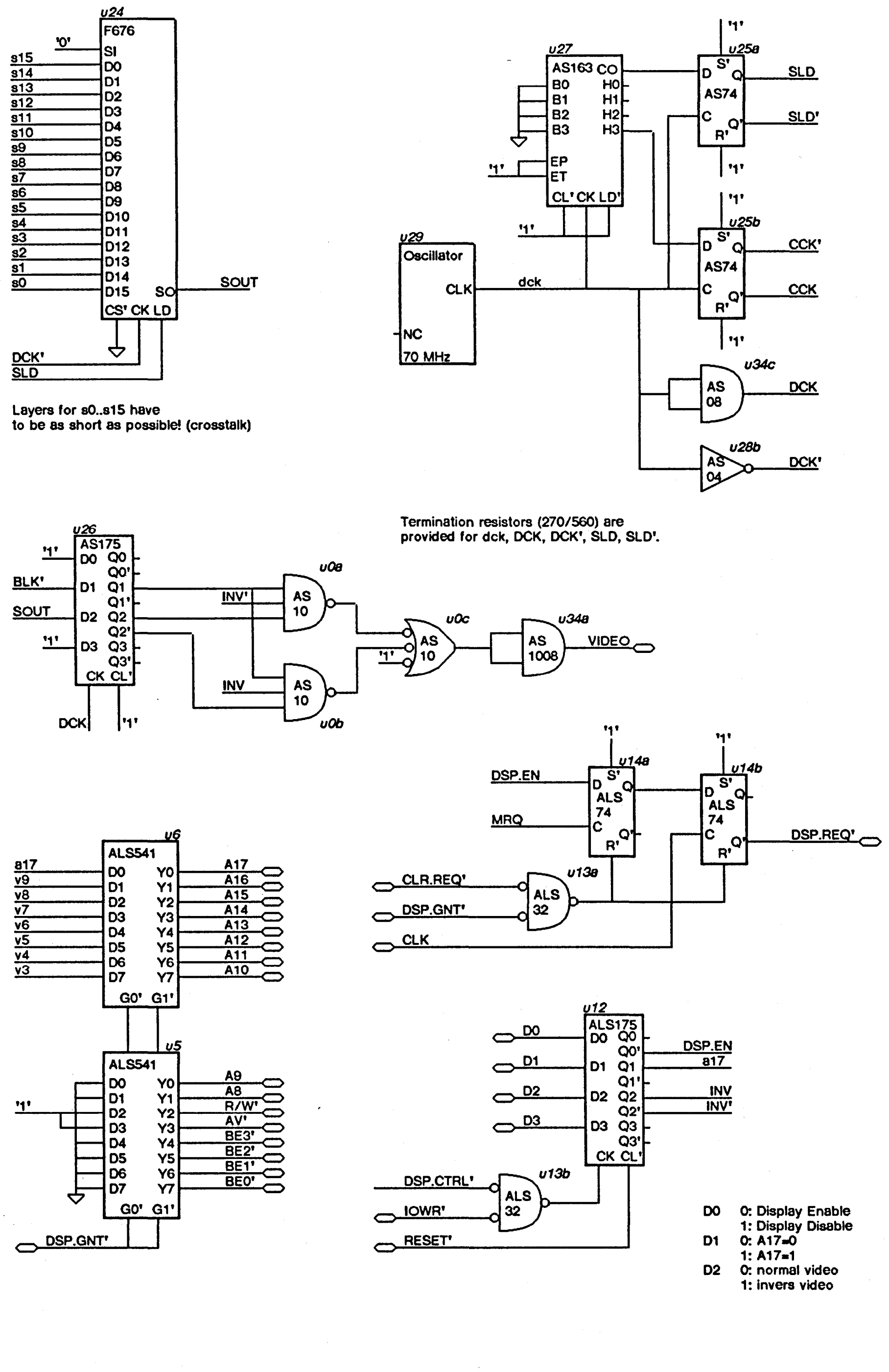
ETH Zurich	*NS.s32.dpl1.SIL* *1/4* DRAM Controller Bus Buffers	Author: N. Wirth H. Eberle	Date: 27.8.85 REV. 9.10.85



ETH Zurich	*NS.s32.dpl2.SIL* *2/4* Display Memory (1 x 32 x 64 KBit)	Author: N. Wirth H. Eberle	Date: 27.8.85 REV. 13.6.86



ETH Zurich	*NS.s32.dpl3.SIL* *3/4* Counters	Author: N. Wirth H. Eberle	Date: 27.8.85 REV. 21.5.86



Layers for s0..s15 have
to be as short as possible! (crosstalk)

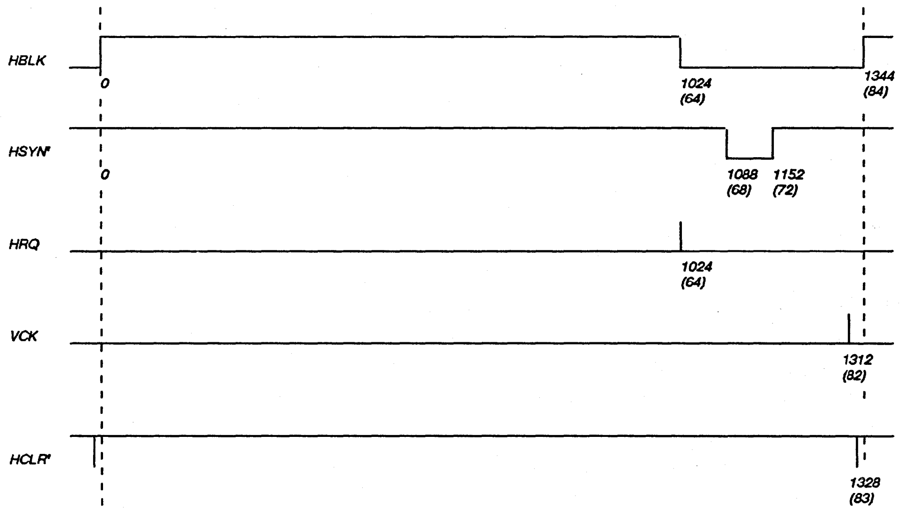
Termination resistors (270/560) are
provided for dck, DCK, DCK', SLD, SLD'.

D0	0: Display Enable
	1: Display Disable
D1	0: A17=0
	1: A17=1
D2	0: normal video
	1: invers video

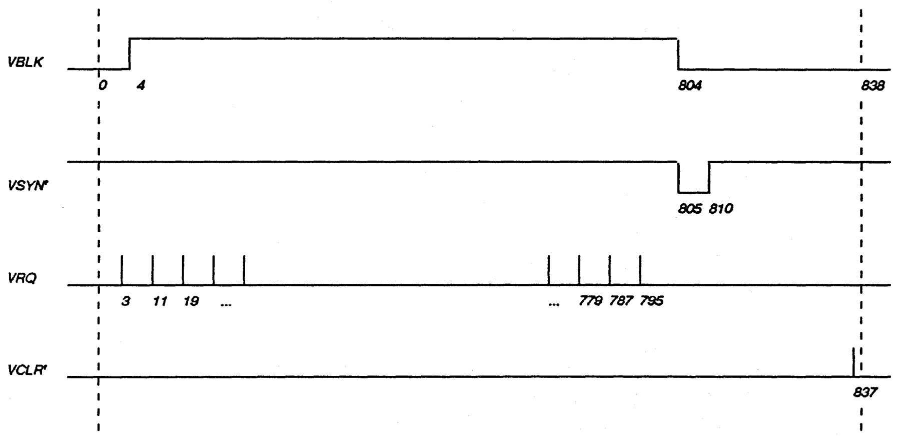
ETH Zurich	*NS.s32.dpl4.SIL* *4/4* Shifter, Clock Generator, Address Buffer, Status Register	Author: N. Wirth, H. Eberle	Date: 27.8.85 REV. 13.6.86

Horizontal timing *(actual H-ROM addresses are 1 less)*

Resolution: 1024 x 800
Fh = 52.08 kHz
Fv = 62.15 Hz
Fp = 70 MHz



Vertical timing



ETH Zurich	NS.s32.dpltiming.SIL Display Timing	Author: H. Eberle	Date: 4.2.86

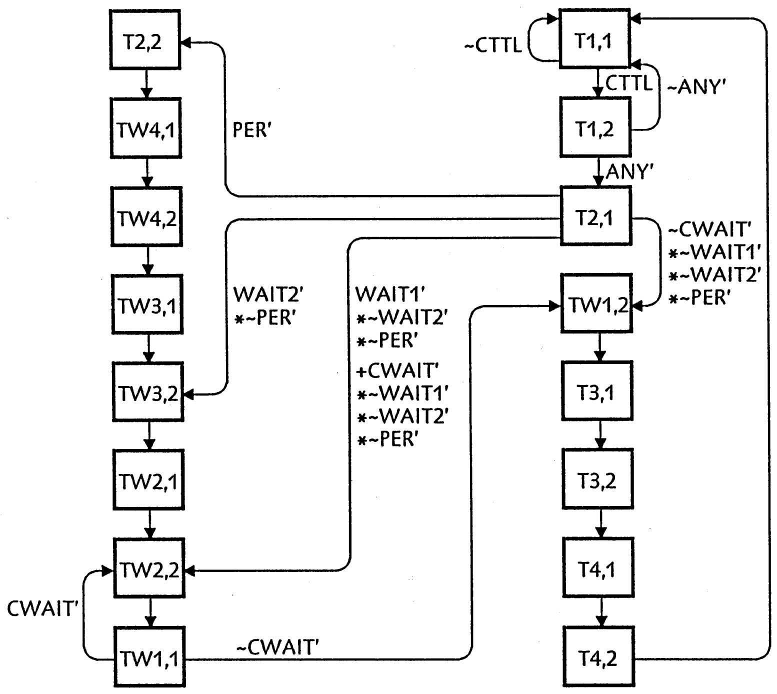
B PAL Logic Equations

```
PAL NSPAL1D: 20L8; (* Address Decoder for ROM and IO Devices *)
  IF TRUE ROMEN   := AV * A23 * A22 * A21 * A20 * A19 * ~A18 * ~A17 * ~A16 * ~A15 * SEL1 * SEL0
                   + AV * A23 * A22 * A21 * A20 * A19 * ~A18 * ~A17 * ~A16 * SEL1 * ~SEL0
                   + AV * A23 * A22 * A21 * A20 * A19 * ~A18 * ~A17 * ~SEL1 * SEL0
                   + AV * A23 * A22 * A21 * A20 * A19 * ~A18 * ~SEL1 * ~SEL0;
  IF TRUE IOEN    := AV * A23 * A22 * A21 * A20 * A19 * A18;
  IF TRUE IOPG0   := AV * A23 * A22 * A21 * A20 * A19 * A18 * A17 * A16 * A15 * A14 * A13 * A12
                   * A11 * A10 * A9;
  IF TRUE IOPG1   := AV * A23 * A22 * A21 * A20 * A19 * A18 * A17 * A16 * A15 * A14 * A13 * A12
                   * A11 * A10 * ~A9;
END NSPAL1D.

PAL NSPAL2B: 16L8; (* Priority Encoder *)
  IF TRUE ANY    :=  DSPREQ + REFREQ + REQ0 + REQ1 + REQ2 + REQ3 + CPUREQ;
  IF TRUE DSPGNT :=  DSPREQ;
  IF TRUE REFGNT := ~DSPREQ *  REFREQ;
  IF TRUE GNT0   := ~DSPREQ * ~REFREQ *  REQ0;
  IF TRUE GNT1   := ~DSPREQ * ~REFREQ * ~REQ0 *  REQ1;
  IF TRUE GNT2   := ~DSPREQ * ~REFREQ * ~REQ0 * ~REQ1 *  REQ2;
  IF TRUE GNT3   := ~DSPREQ * ~REFREQ * ~REQ0 * ~REQ1 * ~REQ2 *  REQ3;
  IF TRUE CPUGNT := ~DSPREQ * ~REFREQ * ~REQ0 * ~REQ1 * ~REQ2 * ~REQ3;
END NSPAL2B.

PAL NSPAL3H1: 16R8; (* Arbiter Finite State Machine 1 *)
  D0          := ~D3 * ~D2 * ~D1 * ~D0 *  CTTL
               + ~D3 * ~D2 * ~D1 *  D0 *  ANY
               + ~D3 * ~D2 *  D1 *  D0 *  PER
               + ~D3 * ~D2 *  D1 *  D0 * ~PER * ~WAIT2 * ~WAIT1 * ~CWAIT
               +  D3 * ~D2 * ~D0
               +  D2 *  D1 * ~D0
               + ~D3 *  D2 * ~D1 *  D0 * ~CWAIT;
  D1          := ~D3 * ~D2 * ~D1 *  D0 *  ANY
               + ~D3 * ~D2 *  D1 *  D0 * ~PER
               +  D3 * ~D2 *  D0
               + ~D2 *  D1 * ~D0
               + ~D3 *  D2 * ~D1
               +  D3 *  D2 *  D1 * ~D0;
  D2          := ~D3 * ~D2 *  D1 *  D0 *  PER
               + ~D3 * ~D2 *  D1 *  D0 * ~PER * ~WAIT2
               +  D2 * ~D1 *  D0
               +  D3 * ~D2 *  D1 *  D0
               + ~D3 *  D2 *  D1
               + ~D3 * ~D2 *  D1 * ~D0
               +  D3 *  D2 *  D1 * ~D0;
  D3            := ~D3 * ~D2 *  D1 *  D0 *  PER
                 + ~D3 * ~D2 *  D1 *  D0 * ~PER * WAIT2
                 +  D3 * ~D1
                 + ~D2 *  D1 * ~D0
                 +  D3 *  D2 *  D1 * ~D0;
  ~RDY          := ~D3 * ~D2 * ~D1
                 + ~D3 * ~D2 *  D1 *  D0 *  PER
                 + ~D3 * ~D2 *  D1 *  D0 * ~PER *  WAIT2
                 + ~D3 * ~D2 *  D1 *  D0 * ~PER * ~WAIT2 *  WAIT1
                 + ~D3 * ~D2 *  D1 *  D0 * ~PER * ~WAIT2 * ~WAIT1 * CWAIT
                 +  D3
                 + ~D3 *  D2 *  D1 * ~D0
                 + ~D3 *  D2 * ~D1 *  D0 *  CWAIT;
END  NSPAL3H1.

PAL NSPAL3K2: 16R8; (* Arbiter Finite State Machine 2 *)
  ~G            := ~D3 * ~D2 * ~D1 *  D0 *  ANY
                 + ~D2 *  D1
                 +  D3 * ~D1
                 + ~D3 *  D2
                 +  D3 *  D2 *  D1 * ~D0;
  CLEAR         := ~D3 * ~D2 *  D1 * ~D0;
  DS            := ~D3 * ~D2 * ~D1 *  D0 *  RD *  ANY
                 + ~D3 * ~D2 *  D1 *  D0
                 +  D3 * ~D1
                 +  D3 * ~D2
                 + ~D3 *  D2;
  DBE           := ~D3 * ~D2 * ~D1 *  D0 * ~RD *  ANY
                 +  D3 * ~D1 * ~RD
                 + ~D2 *  D1 * ~RD
                 + ~D3 *  D2 * ~RD
                 + ~D3 *  D2 *  D1 *  D0
                 + ~D3 *  D2 * ~D1 * ~D0
                 + ~D3 * ~D2 *  D1 * ~D0;
  IORD          :=  D3 * ~D2 *  RD *  PER
                 + ~D3 *  D2 *  RD *  PER;
  IOWR          :=  D3 * ~D2 * ~RD *  PER
                 + ~D3 *  D2 * ~RD *  PER;
END  NSPAL3K2.
```



(a)

Zn	Q3..0	CTTL	ANY'	PER'	WAIT2'	WAIT1'	CWAIT'	Zn+1	D3..0	G	CLEAR'	DBE'	DS'	RDY	IORD'	IOWR'
T1,1	0000	0	X	X	X	X	X	T1,1	0000	1	1	1	1	0	1	1
		1	X	X	X	X	X	T1,2	0001	1	1	1	1	0	1	1
T1,2	0001	X	1	X	X	X	X	T1,1	0000	1	1	1	1	0	1	1
		X	0	X	X	X	X	T2,1	0011	0	1	R/W'	~R/W'	0	1	1
T2,1	0011	X	X	0	X	X	X	T2,2	1101	0	1	0	0	0	1	1
		X	X	1	0	X	X	Tw3,2	1010	0	1	0	0	0	1	1
		X	X	1	1	0	X	Tw2,2	0110	0	1	0	0	0	1	1
		X	X	1	1	1	0	Tw2,2	0110	0	1	0	0	0	1	1
		X	X	1	1	1	1	Tw1,2	0111	0	1	0	0	1	1	1
T2,2	1101	X	X	X	X	X	X	Tw4,1	1100	0	1	0	0	0	1	1
Tw4,1	1100	X	X	X	X	X	X	Tw4,2	1000	0	1	0	0	0	1	1
Tw4,2	1000	X	X	X	X	X	X	Tw3,1	1001	0	1	0	0	0	Y1	Y2
Tw3,1	1001	X	X	X	X	X	X	Tw3,2	1010	0	1	0	0	0	Y1	Y2
Tw3,2	1010	X	X	X	X	X	X	Tw2,1	1011	0	1	0	0	0	Y1	Y2
Tw2,1	1011	X	X	X	X	X	X	Tw2,2	0110	0	1	0	0	0	Y1	Y2
Tw2,2	0110	X	X	X	X	X	X	Tw1,1	0101	0	1	0	0	0	Y1	Y2
Tw1,1	0101	X	X	X	X	X	0	Tw2,2	0110	0	1	0	0	0	Y1	Y2
		X	X	X	X	X	1	Tw1,2	0111	0	1	0	0	1	Y1	Y2
Tw1,2	0111	X	X	X	X	X	X	T3,1	0100	0	1	0	0	1	Y1	Y2
T3,1	0100	X	X	X	X	X	X	T3,2	0010	0	1	0	0	1	Y1	Y2
T3,2	0010	X	X	X	X	X	X	T4,1	1110	0	0	0	1	1	1	1
T4,1	1110	X	X	X	X	X	X	T4,2	1111	0	1	1	1	0	1	1
T4,2	1111	X	X	X	X	X	X	T1,1	0000	1	1	1	1	0	1	1

Y1 = ~(R/W'∗/PER')
Y2 = ~(/R/W'∗/PER')

(b)

Figure B.1 Bus control state diagram (a) and truth table (b).

```
PAL NSPAL4A: 16R8; (* Mouse Direction Discriminator *)
  X0N         := X0;                X1N        := X1;
  Y0N         := Y0;                Y1N        := Y1;
  EX          := ~X1N * ~X1 * ~X0N *  X0 + ~X1N * ~X1 * X0N * ~X0
               + ~X1N *  X1 * ~X0N * ~X0 + ~X1N *  X1 * X0N *  X0
               +  X1N * ~X1 * ~X0N * ~X0 +  X1N * ~X1 * X0N *  X0
               +  X1N *  X1 * ~X0N *  X0 +  X1N *  X1 * X0N * ~X0;
  EY          := ~Y1N * ~Y1 * ~Y0N *  Y0 + ~Y1N * ~Y1 * Y0N * ~Y0
               + ~Y1N *  Y1 * ~Y0N * ~Y0 + ~Y1N *  Y1 * Y0N *  Y0
               +  Y1N * ~Y1 * ~Y0N * ~Y0 +  Y1N * ~Y1 * Y0N *  Y0
               +  Y1N *  Y1 * ~Y0N *  Y0 +  Y1N *  Y1 * Y0N * ~Y0;
  UX          := X0 * ~X1N + ~X0 * X1N;
  UY          := Y0 * ~Y1N + ~Y0 * Y1N;
END NSPAL4A.
```

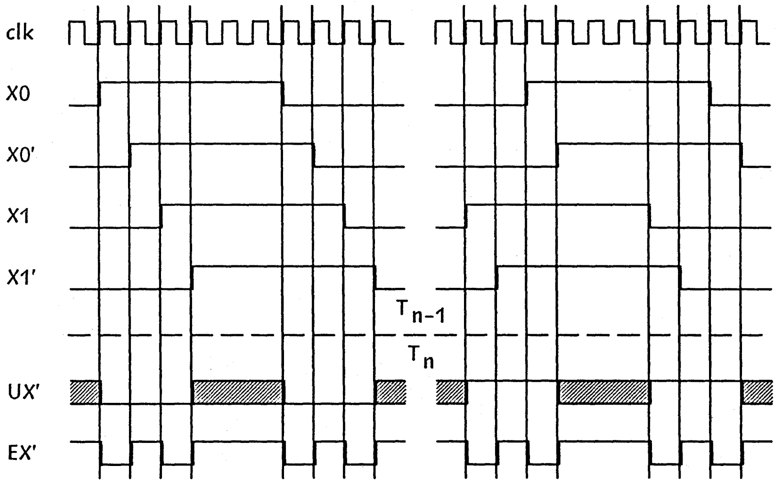


Figure B.2 Timing diagram of the mouse interface.

```
PAL NSPAL6A: 16L8; (* Display Controller Address Decoder *)
  IF TRUE MCS     := AV * A23 * A22 * A21 * ~A20 * ~A19 * ~A18;
  IF TRUE DPSTAT  := AV * A23 * A22 * A21 *  A20 *  A19 *  A18 * A17 * A16 * A15 * A14 * A13
                     * A12 * A11 * ~A10 * A9;
END NSPAL6A.
```

```
PAL NSPAL7B: 16L8; (* Byte Enable & Other Glue Logic *)
  IF CPUGNT BE0  :=  MMUMAC * ~A1 + ~MMUMAC * CPUBE0 + RD;
  IF CPUGNT BE1  :=  MMUMAC * ~A1 + ~MMUMAC * CPUBE1 + RD;
  IF CPUGNT BE2  :=  MMUMAC *  A1 + ~MMUMAC * CPUBE2 + RD;
  IF CPUGNT BE3  :=  MMUMAC *  A1 + ~MMUMAC * CPUBE3 + RD;
  IF TRUE GD     :=  CPUGNT * DBE * ~MMUMAC + CPUGNT * DBE * ~A1;
  IF TRUE GW     :=  CPUGNT * DBE *  MMUMAC * A1;
  IF TRUE ~r'/w  :=  RD;              IF TRUE ~r/w'   := ~RD;
END NSPAL7B.
```

C Test Programs

Global Variables

```
VAR
   A, B, C: ARRAY [0..255] OF INTEGER;
   head: NodePtr;
```

Local Variables

```
VAR
   i, j, k, m: INTEGER;
   I, J, K: LONGINT;
   r0, r1, r2: REAL;
   p: NodePtr;
```

Empty Loop

```
k := 1000;                          MOVW     1000     k(FP)
REPEAT                          L:  ADDQW    -1       k(FP)
   k := k - 1                       CMPQW    0        k(FP)
UNTIL k = 0;                        BNE      L
```

INTEGER Arithmetic

```
j := 0;                             MOVQW    0        j(FP)
k := 1000;                          MOVW     1000     k(FP)
REPEAT                          L:  ADDQW    -1       k(FP)
   k := k - 1;                      ADDQW    1        j(FP)
   j := j + 1;                      MOVW     k(FP)    R7
   i := (k*3) DIV (j*5)             MULW     3        R7
UNTIL k = 0;                        MOVW     j(FP)    R6
                                    MULW     5        R6
                                    DIVW     R6       R7
                                    MOVW     R7       i(FP)
                                    CMPQW    0        k(FP)
                                    BNE      L
```

LONGINT Arithmetic

```
J := 0D;                            MOVQD    0        J(FP)
K := 1000D;                         MOVD     1000D    K(FP)
REPEAT                          L:  ADDQD    -1       K(FP)
   K := K - 1;                      ADDQD    1        J(FP)
   J := J + 1;                      MOVD     K(FP)    R7
   I := (K*3D) DIV (J * 5D)         MULD     3        R7
UNTIL K = OD;                       MOVD     J(FP)    R6
                                    MULD     5        R6
                                    DIVD     R6       R7
```

```
                                            MOVD       R7        I(FP)
                                            CMPQD      0         K(FP)
                                            BNE        L
```

REAL Arithmetic

```
k := 1000;                                  MOVW       1000      k(FP)
r1 := 7.28;                                 MOVF       7.28E00   r1(FP)
r2 := 34.8;                                 MOVF       3.48E01   r2(FP)
REPEAT                                  L:  ADDQW      -1        k(FP)
   k := k - 1;                              MOVF       r1(FP)    F6
   r0 := (r1 * r2) / (r1 + r2)              MULF       r2(FP)    F6
UNTIL k = 0;                                MOVF       r1(FP)    F4
                                            ADDF       r2(FP)    F4
                                            DIVF       F4        F6
                                            MOVF       F6        r0(FP)
                                            CMPQW      0         k(FP)
                                            BNE        L
```

Array Indexing

```
k := 1000;                                  MOVW       1000      k(FP)
 i := 0;                                    MOVQW      0         i(FP)
B[0] := 73;                                 MOVW       73        B(SB)
REPEAT                                  L:  MOVZWD     i(FP)     R7
   A[i] := B[i]; B[i] := A[i];              MOVZWD     i(FP)     R6
   k := k - 1                               MOVW       B(SB) [R6:W]  A(SB) [R7:W]
UNTIL k = 0;                                MOVZWD     i(FP)     R7
                                            MOVZWD     i(FP)     R6
                                            MOVW       A(SB) [R6:W]  B(SB) [R7:W]
                                            ADDQW      -1        k(FP)
                                            CMPQW      0         k(FP)
                                            BNE        L
```

Procedure Call

```
PROCEDURE Q(x, y, z, w: INTEGER);       Q:  ENTER      []        0
BEGIN                                       EXIT       0
END Q;                                      RET        0         16

k := 1000;                                  MOVW       1000      k(FP)
REPEAT                                  L:  MOVZWD     i(FP)     TOS
   Q(i, j, k, m);                           MOVZWD     j(FP)     TOS
   k := k - 1                               MOVZWD     k(FP)     TOS
UNTIL k = 0;                                MOVZWD     m(FP)     TOS
                                            BSR        Q
                                            ADDQW      -1        k(FP)
```

```
                                                  CMPQW    0           k(FP)
                                                  BNE      L
```

Copying Arrays

```
k := 1000;                                        MOVW     1000        k(FP)
REPEAT                                        L:  ADDQW    -1          k(FP)
   k := k - 1;                                    ADDR     B(SB)       R1
   A := B; B := C; C := A                         ADDR     A(SB)       R2
UNTIL k = 0;                                      MOVZBD   128         R0
                                                  MOVSD    0
                                                  ADDR     C(SB)       R1
                                                  ADDR     B(SB)       R2
                                                  MOVZBD   128         R0
                                                  MOVSD    0
                                                  ADDR     A(SB)       R1
                                                  ADDR     C(SB)       R2
                                                  MOVZBD   128         R0
                                                  MOVSD    0
                                                  CMPQW    0           k(FP)
                                                  BNE      L
```

Pointer Handling

```
k := 1000;                                        MOVW     1000        k(FP)
REPEAT p := head;                            L1:  MOVD     head(SB)    p(FP)
   REPEAT p := p↑.next UNTIL p = NIL;        L2:  MOVD     p(FP)+4     p(FP)
   k := k - 1                                     CMPQD    0           p(FP)
UNTIL k = 0;                                      BNE      L2
(* head points to a list of 100 nodes *)          ADDQW    -1          k(FP)
                                                  CMPQW    0           k(FP)
                                                  BNE      L1
```

References

References to datasheets are listed separately.

[Backus 78]
J. Backus. Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs. Comm. of the ACM, Vol. 21, No. 8, 1978, pp. 613-641.

[Baecker 79]
R. Baecker. Digital Video Display Systems and Dynamic Graphics. Computer Graphics, Vol. 13, No. 2, 1979, pp. 48-56.

[Bechtolsheim 80]
A. Bechtolsheim, F. Baskett. High-Performance Raster Graphics for Microcomputer Systems. Computer Graphics, Vol. 14, No. 3, 1980, pp. 43-47.

[Borrill 85]
P. L. Borrill. 32-Bit Buses - An Objective Comparison. IEE Computing and Control Division, Colloquium on "IEEE 896 Futurebus - The Ultimate Backplane Bus Standard ?", No. 1986/20, January 1986.

[CG&A 86]
i) M. Asal, G. Short, T. Preston, R. Simpson, D. Roskell, K. Guttag. The Texas Instruments 34010 Graphics System Processor.
ii) C. Carinalli, J. Blair. National's Advanced Graphics Chip Set for High-Performance Graphics.
iii) G. Shires. A New VLSI Graphics Coprocessor - The Intel 82786. IEEE Computer Graphics and Applications, Vol. 6, No. 10, 1986.

[Chen 74]
R. C. Chen. Bus Communications Systems. Ph.D. Thesis, Carnegie-Mellon University, 1974.

[Cohen 81]
D. Cohen. On Holy Wars and a Plea for Peace. IEEE Computer, Vol. 14, No. 10, 1981, pp. 48-54.

[Corso 86]
D. Del Corso, H. Kirrmann, J. D. Nicoud. Microcomputer Buses and Links. Academic Press, London, 1986.

[Färber 84]
G. Färber. Bussysteme: Parallele und serielle Bussysteme in Theorie und Praxis. R. Oldenbourg Verlag, München, Wien, 1984.

[Gustavson 84]
D. B. Gustavson. Computer Buses - A Tutorial. IEEE Micro, Vol. 4, No. 4, 1984, pp. 7-22.

[Gutknecht 83]
J. Gutknecht. System Programming in Modula-2: Mouse and Bitmap Display. Institut für Informatik, ETH Zurich, Report No. 56, September 1983.

[Hayes 79]
J. P. Hayes. Computer Architecture and Organization. McGraw-Hill, Singapore, 1979.

[Heiz 87]
W. Heiz. Modula-2 auf einem RISC: Realisierung und Vergleich. Ph.D. Thesis, ETH Zurich, 1987. (To be published)

[Ingalls 81]
D. H. Ingalls. The Smalltalk Graphics Kernel. Byte, Vol. 6, No. 8, 1981, pp. 168-194.

[Kirrmann 83]
H. Kirrmann. Data Format and Bus Compatibility in Multiprocessors. IEEE Micro, Vol. 3, No. 4, 1983, pp. 32-47.

[Knudsen 83]
S. E. Knudsen. A Modula-2 Oriented Operating System for the Personal Computer Lilith. Ph.D. Thesis Nr. 7346, ETH Zurich, 1983.

[Kohen 85]
E. Kohen. An Interactive Method for Middle Resolution Font Design on Personal Workstations. Int. Comp. Symp., ACM European Region, Florence, 1985.

[Kronfeld 85]
C. D. Kronfeld. Architectural Elements for Bitmap Graphics. Xerox Palo Alto Research Center, Report CSL-85-2, 1985.

[Lampson 80]
B. W. Lampson, K. A. Pier. A Processor for a High-Performance Personal Computer. The 7th Int. Symp. on Comp. Arch., ISCA-7, IRISA, La Baule, May 1980, pp. 146-160.

[Levy 78]
J. V. Levy. Buses, The Skeleton of Computer Structures. In Computer Engineering: A DEC View of Hardware Systems Design by C. G. Bell, J. C. Mudge, J. E. McNamara. Digital Press, 1978, pp. 269-299.

[Lyon 85]
T. Lyon, J. Skudlarek. All the Chips that Fit. USENIX Conf. Proc., June 1985, pp. 557-561.

[Newman 79]
W. M. Newman, R. F. Sproull. Principles of Interactive Computer Graphics. McGraw Hill, New York, 2nd Ed., 1979.

[Ohran 84]
R. S. Ohran. Lilith: A Workstation Computer for Modula-2. Ph.D. Thesis No. 7646, ETH Zurich, 1984.

[Peschel 87]
F. Peschel, M. Wille. Porting Medos-2 onto the Ceres Workstation. Institut für Informatik, ETH Zurich, Report No. 78, April 1987.

[Pike 83]
R. Pike. Graphics in Overlapping Bitmap Layers. ACM Trans. on Graphics, Vol. 2, No. 2, 1983, pp. 135-160.

[Pike 85]
R. Pike, B. Locanthi, J. Reiser. Hardware/Software Trade-offs for Bitmap Graphics on the Blit. Software-Practice and Experience, Vol. 15, No. 2, 1985, pp. 131-151.

[Pinkham 83]
R. Pinkham, M. Novak, C. Guttag. Video RAM Excels at Fast Graphics. Electronic Design, Vol. 31, No. 17, 1983, pp. 161-182.

[Seck 83]
J. P. Seck, M. Courvoisier, J. C. Geffroy. Contrôle de l'accès a un bus partagé: les arbitres. R.A.I.R.O. Automatique/Systems Analysis and Control, Vol. 17, No. 4, 1983, pp. 359-403.

[Taub 84]
D. M. Taub. Arbitration and Control Acquisition in the Proposed IEEE 896 Futurebus. IEEE Micro, Vol. 4, No. 4, 1984, pp. 28-41.

[Thacker 79]
C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, D. R. Boggs. Alto: A Personal Computer. Xerox Palo Alto Research Center, Report CSL-79-11, 1979.

[Thurber 72]
K. J. Thurber, E. D. Jensen, L. A. Jack, L. L. Kinney, P. C. Patton, L. C. Anderson. A systematic approach to the design of digital bussing structures. AFIPS Conf. Proc., Vol. 41, Part II, 1972 FJCC, pp. 719-740.

[Thurber 78]
K. J. Thurber, G. M. Masson. Distributed-Processor Communication Architecture. Lexington Books, Lexington MA, Toronto, 1978.

[Wanner 84]
J. Wanner. Assembler und Rasteroperationen für den NS16000. Diploma thesis, Institut für Informatik, ETH Zurich, September 1984.

[Weicker 84]
R. P. Weicker. Dhrystone: A Synthetic Systems Programming Benchmark. Comm. of the ACM, Vol. 27, No. 10, 1984, pp. 1013-1030.

[Whitton 84]
M. C. Whitton. Memory Design for Raster Graphics Displays. IEEE Computer Graphics, Vol. 4, No. 3, 1984, pp. 48-65.

[Wirth 81a]
N. Wirth. The personal computer Lilith. Institut für Informatik, ETH Zurich, Report No. 40, April 1981.

[Wirth 81b]
N. Wirth. The personal computer Lilith. Proc. of the 5th Intern. Conf. on Software Engineering, IEEE Computer Society Press, San Diego, 1981.

[Wirth 82]
N. Wirth. Programming in Modula-2. Springer-Verlag, Heidelberg, New York, 1982.

[Wirth 86a]
N. Wirth. Microprocessor architectures: a comparison based on code generation by a compiler. Comm. of the ACM, Vol. 29, No. 10, 1986, pp. 978-990.

[Wirth 86b]
N. Wirth. A Fast and Compact Compiler for Modula-2. Institut für Informatik, ETH Zurich, Report No. 64, July 1986.

[Wirth 87a]
N. Wirth. An Extensible System and a Programming Tool for Workstation Computers. 4th South African Comp. Symp., Pretoria, July 1987.

[Wirth 87b]
N. Wirth. Hardware Architectures for Programming Languages and Programming Languages for Hardware Architectures. Proc. 2nd Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-II), Palo Alto, October 1987, pp. 2-8.

Data Sheets

[AMD 80]
Am9519A Universal Interrupt Controller. Datasheet, Advanced Micro Devices, 1980.

[AMD 84]
Am9519A Universal Interrupt Controller. Technical Manual, Advanced Micro Devices, 1984.

[AMD 85]
Bus Interface Product Specifications. Databooklet, Advanced Micro Devices, October 1985.

[Hitachi 1986]
IC Memory Products. Databook, 10-2 R, 1986, pp. 382-404.

[Intel 84]
Multibus II Bus Architecture Specification Handbook. Revision C, 146077-C, Intel, 1984.

[Intel 86]
80386 High Performance 32-Bit CHMOS Microprocessor with Integrated Memory Management. Datasheet, Intel, 1986.

[MEM 84]
M3002 Real Time Clock Circuit. Datasheet, Microelectronic-Marin, 1984.

[Motorola 85a]
VMEbus Specification Manual. Revision C.1, Motorola, 2nd Ed., 1985.

[Motorola 85b]
MC68020 32-Bit Microprocessor User's Manual. Prentice-Hall, Englewood Cliffs, NJ, 2nd Ed., 1985.

[MMI 78]
Programmable Logic Array Handbook. Monolithic Memories, 5th Ed., 1978.

[NCR 85]
NCR 5386 SCSI Protocol Controller. Datasheet, NCR, 1985.

[NS]
NS32000 Series User Information. National Semiconductor.

[NS 83]
Interface, Bipolar LSI, Bipolar Memory, Programmable Logic. Databook, National Semiconductor, 1983.

[NS 84a]
Series 32000 Instruction Set Reference Manual. National Semiconductor, 1984.

[NS 84b]
The Specifics of 32-Bit Architecture and Implementation. National Semiconductor, 1984.

[NS 86a]
Series 32000 Databook. National Semiconductor, 1986.

[NS 86b]
Series 32300 Datasheets: NS32332 32-Bit Advanced Microprocessor with Virtual Memory, NS32382 Memory Management Unit, NS32381 Floating Point Unit, NS32301 Timing Control Unit. Datasheets, National Semiconductor, 1986.

[Philips 83]
SCN2681 Dual Asynchronous Receiver/Transmitter (DUART). Datasheet, Philips, May 1983.

[Philips 84]
FAST TTL Logic series. Databook IC15N, Philips, 1984.

[SMC 85]
Data Catalog 1985, Standard Microsystems Corporation, 1985.

[TI 77]
The Interface Circuits Data Book. Databook, Texas Instruments, 1977.

[TI 83a]
Nubus Specifications. TI-2242825-0001, Texas Instruments, 1983.

[TI 83b]
TMS4161 65536 Bit Multiport Memory. Datasheet, Texas Instruments, July 1983.

[TI 83c]
The TTL Data Book: Advanced Low-Power Schottky, Advanced Schottky. Volume 3, Texas Instruments, 1984.

[VTI]
VL16160 "Raster Op" Graphics/Boolean Operation ALU. Datasheet, VTI VLSI Technology.

[WD 83]
WD1002-05/HDO Winchester/Floppy Disk Controller. OEM Manual, Western Digital, July 1983.

[Zilog 82a]
Z8530 and Z8030 SCC Initialization: A Worksheet and an Example. Zilog, September 1982.

[Zilog 82b]
Z8530/Z8030 Serial Communications Controller. Technical Manual, Advanced Micro Devices, 1982.

[Zilog 85]
Z8530 SCC Serial Communications Controller. Datasheet, Zilog, August 1985.

Curriculum Vitae

I was born on August 18, 1959 in the city of Zurich, Switzerland. From 1966 until 1972 I attended primary school in Glattbrugg. In 1972 I entered the Kantonsschule Oerlikon where I graduated in 1978 with a Matura Typ B.

In 1979 I began my studies in electrical engineering at the Swiss Federal Institute of Technology (ETH) in Zurich. I obtained the diploma in electrical engineering in 1984 .

Since January 1984 I have worked as an assistant at the Institut für Informatik of ETH Zurich in the research group of Prof. N. Wirth.