INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONES: 224281/2/3/4/5/6
CABLE: CENTRATOM - TELEX 460392-1

SMR/90 - 16

COLLEGE ON MICROPROCESSORS:

TECHNOLOGY AND APPLICATIONS IN PHYSICS

7 September - 2 October 1981

MICROPROCESSOR INTERFACING

J-D. NICOUD

LAMI
Ecole Polytechnique Fédérale
de Lausanne
Chemin de Bellerive 16
CH-1007 Lausanne
Switzerland

MICROPROCESSOR INTERFACING

J.D. Nicoud,
LAMI EPFL, Bellerive 16, CH-1007 Lausanne

0. Introduction

This paper introduces the general concepts related to data transfers within and between systems, and presents some specific solutions and applications in this rapidly expanding field. Microprocessors themselves are not explained, but programmable interfaces are approached, and the software of interfacing is partly covered.

1. Parallel transfers

1.1. General problem

Information to be transferred from one device (e.g. a computer) to another device (e.g. a peripheral) is usually split into a set of binary words of adequate length (multiple of 8 bits). These words are transferred in parallel or serially (one bit at a time), and some synchronisation is required in order to recognize the arrival of each word and detect the end of the message.

The devices of a simple system are frequently connected in a star manner, with the computer in the center and a peripheral device on each branch. Within a computer, the processor, the memory boards, the interfaces cannot be connected in a star manner, with all the direct connections between each pair of devices.

A common set of lines, the bus is a shared resource each pair of devices have to use when communicating. Since the bus is short, it can be fast, and multiplex easily the transfers occuring in the system.

Increasing the number of paths within the system is very expensive, and is not considered here.

The links established between computers form a complex network. Adequate software and node interfaces allow the communications within this large system.

Fig. 1.1 shows these three levels of communications, for which experience has shown that bus, star and network connections were the best. The bus concept is however of an increasing importance, due to its use in instrumentation for distances up to 20 m (e.g. HP-IB/IEEE 488/IEC 625) and in office automation for distances up to 2 km (e.g. Ethernet/IEEE 802).

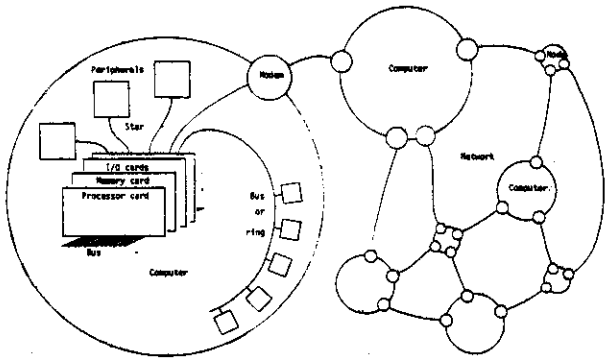

Fig. 1.1. Hierarchy and topology in a system

1.1.1. Bussed systems

The common resource of the bus implies a strict hierarchy in the system. Devices are either masters (they can activate a transfer) or slaves (they answer to sollicitations). Since there is a single transfer path, one master at a time, the commander can exchange information with one (optionally more) selected slave, the responder (fig. 1.2).

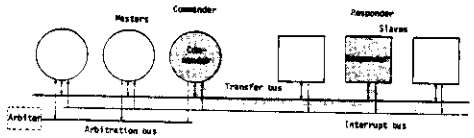

Fig. 1.2. Transfer in a bussed system

A transfer moves information from one or a group of data cells from one or a group of data cells (memory cell, I/O register, processor register) to another. These cells arew numbered and an unique address allows their access.

A transfer can be initiated by a master only if it has got access to the bus, and an arbitration mechanism must decide who will get the bus next, and notify it. Arbiter can be distributed or centralized, as it will be explained later.

The commander selects the slave with which he has information to transfer, according to some selection or addressing scheme. Transfers may then be performed as one or a succession of elementary transfers, each one moving a word of a maximum width fixed by the number of bus data lines.

Elementary transfers, or cycles are combined to form block transfers, read-modify-write cycles or read-after-write cycles.

Broadcast cycles write information simultaneously to several responders. Broad-collect cycles read information from several responders, taking care of the fact that a given bus line cannot be simultaneously assigned to be a "one" and to be a "zero" by two different devices.

Slaves needing to be serviced can signal it through an interruption mechanism, otherwise it would not be efficient to have the masters to continuously worry about the possible transfer needs of slaves.

Systems can be as simple as one master and few slaves (e.g. MKDS kits), but they tend to consist mostly of combined master/slave devices, and may be split into several sub-systems having each one its bus, with another bus and adequate interconnection between all these sub-systems.

1.1.2. Transfer

As seen earlier, a transfer consists of three consecutive phases:

- arbitration for getting access of the bus; the master having control of the bus is the commander
- selection of one or many implicitely or explicitely addressed slaves. The selected slaves are the responders
- data transfer (read, write, etc.).

The bus width and bandwidth requirements for these three phases are rather different.

Arbitration must be done usually between 3 to 30 masters. If arbitration requests are encoded, this means only 5 bits plus associated control. A new arbitration must occur each time the commander changes, which is highly application dependant.

Selection of slaves, plus precise location of memory or I/O register inside the slave, require a 16 to 32-bit address. One single selection is sufficient for a long data transfer consisting of ordered informations.

Data transfer is done with present microprocessors 8 or 16 bits at a time, this value being often mentionned to be the "width" of the bus.

In simple systems, the three transfer phases occur sequentially on three separated groups of lines (fig. 1.3a).

Pipelining, that is starting the next operation before the present operation is finished, can be done (fig. 1.3b); present microprocessor busses tend to pipeline the arbitration, but not the selection which would be too complex for the time saved (fig. 1.3c).

Due to the cost of the transfer lines and their associated driving electronic, most recent busses tend to multiplex arbitration selection and data transfer on the same line (fig. 1.3d). This is of course the case on 1-bit wide serial busses. Selection and data transfers are very frequently multiplexed in microprocessor systems (fig. 1.3e), and many combinations exist on 8-bit microprocessors, depending if the low address (SC/MP), the high address (8085, 6801) or control (8080), is multiplexed with data.

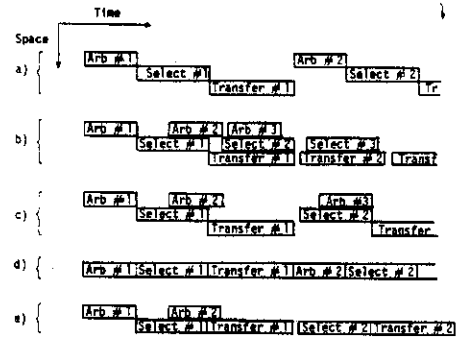

Fig. 1.3. Pipelining and multiplexing on a bus.

1.2. Transaction type

Data transfers on a microprocessor bus are addressed, and consist of a set of control and data exchanges between the commander and the responder called a transaction, consisting of consecutive transfer cycles which may overlap if there exist parallel paths for transferring the selection, data and control information.

Address cycles are special broadcast write cycles: all slaves check for the address, but a single one, or few, are selected.

Data cycles are write or read if a single responder has been selected, broadcast (write) or broad-collect (read) if multiple responders have been selected (fig. 1.4a). If the bus is not multiplexed, address and data can be provided at the same time by the commander but the responder must first be selected before it can process the data (fig. 1.4b).

Read-modify-write transfers consist of one address cycle followed by one data read and one data write cycle, the whole being indivisible in order to prevent any access to the data cell while its information is being modified by a commander (fig. 1.4c).

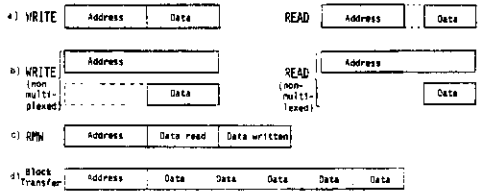

Fig. 1.4. Transaction types

Read-after-write transfers are similar and must also be indivisible. They allow to check that the written information has been correctly stored.

Block transfers consist of one address cycle followed by n data cycles. First data is transferred with the given address, next data being transferred with subsequent addresses, similarly to a post-incremented addressing mode (fig. 1.4d).

A special category of transfers is split transfers. If the requested information is temporarily unaccessible or very slow to get out (e.g. from a disk), it is stupid to wait for it and keep the bus during all this time. In this case, the responder can return an error code saying it will answer later. The commander can also specify by writing in a special register the information he requests to the responder. That one will call back as soon as information will be valid, playing that time the role of a commander.

1.3. Data cycles

1.3.1. Unidirectional transfers

An elementary information transfer cycle between two devices requires information lines, control lines and a protocol specifying when information lines are valid and can be removed.

The device which sends the information is the source, the device which receives it is the destination. If the source takes always the initiative of the transfer, there are only write cycles in the system, and no read cycles. If the commander takes the initiative of the transfer, we have as before both read and write cycles.

1.3.2. Full handshaked write protocol

Transferring information is not just putting binary words on a set of lines. Explicit or implicit information must allow both the destination and the source to be synchronized, so that no information is lost. A typical handshake write protocol is given in fig. 1.5.

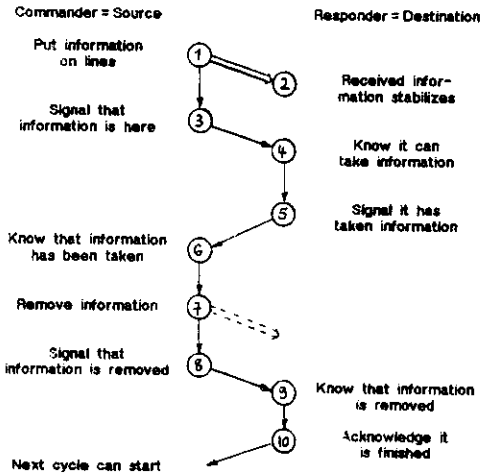

Fig. 1.5. Write cycle protocol with full handshake

Other simpler protocols will be studied later. This one has a simple implementation using two control lines in addition to the information lines. These two lines are named VAL ("valid", issued by the source) and AK ("acknowledge", issued by the destination). VAL is active in states 3 to 7, AK in states 5 to 9.

A timing diagram allows a clear expression of a transfer protocol. Multiple information lines are represented by double lines crossing each other when the information changes. Control signals are shown with a high level when the function suggested by the name of the signal is true. The timing diagram of this first protocol is given in fig. 1.6.

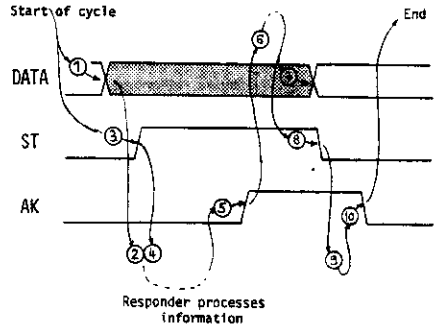

Fig. 1.6. Timing diagram for full handshake write cycle

The technological implementation of this protocol requires drivers between the two concerned devices and the transmission line, which can be long or use a non-electric media (fig. 1.7).

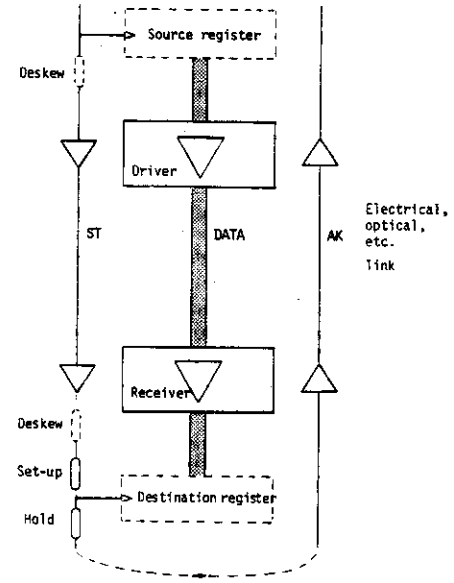

Fig. 1.7. Interface schematic for full handshaked protocol

In the basic case of the transfer of the content of one register A into a distant register B, the clock signal which has loaded a new information in register A can be used as the VAL control signal, but it must be delayed either at source or at destination, in order to take care of differences in propagation time (skew) between the parallel lines and their drivers. Another delay, usually included in the previous one, guaranties the set-up time on the receiving register, before receiving its load pulse. Information must be valid for some hold time after the load pulse, implying a delay which is usually provided by the 4 levels of drivers which have to switch before the information is removed.

1.3.3. Other write protocols

In the protocol of fig. 1.5 and 1.6, it is clear that time slots 1, 3 and 5 are the important ones. Removing the information and checking this being recognised is wasted time, if it is sure the destination is immediately ready for a new transfer. Since we need for a simple interface a full pulse in each cycle, and since either polarity of signal can be used, there are several possible protocols, shown in fig. 1.8.

Scheme a) is the previously seen full handshaked protocol.
Scheme b) provides a short pulse for data valid, and a short pulse for data acknowledge. This assumes a maximum transfer speed fixed by the width of the pulses.
Scheme c) provides an immediate acknowledge that has been selected and is busy handling the information.
The fastest protocol (fig. 1.8d) considers pairs of cycles, synchronized alternatively by positive and negative edges (FASTBUS [1] block transfers). It is faster since the bandwidth of the synchronisation signals and the information is the same.
The last scheme e) has no acknowledge. The responder has to take the information when the strobe (clock) signal is active. This is a synchronous transfer of a predefined speed, all the other transfers a) to d) being asynchronous due to the handshake.

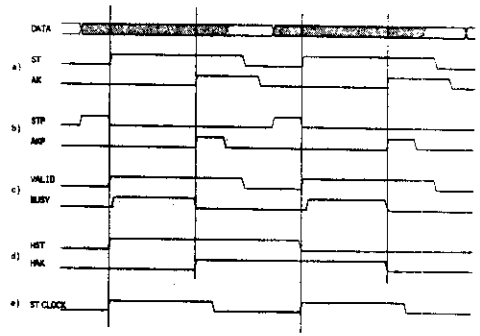

Fig. 1.8. Write protocols.

1.3.4. Example: the Centronics interface

An interface to a simple printer is a typical example of a transfer requiring a write handshaked protocol. Information transferred is 7-bit ASCII characters and each manufacturer has proposed his protocol. Centronics seems to be now the de-facto standard. The transfer protocol is shown in fig. 1.9, with the usual timing constraints (they vary from one manufacturer to another). This protocol is not looking for high speed, due to the low printer speed. The redundant BUSY signal is active when the transfer is slown down because the printer is busy. When the transfer is done in the printer buffer, ACK is promptly given and BUSY is not activated.

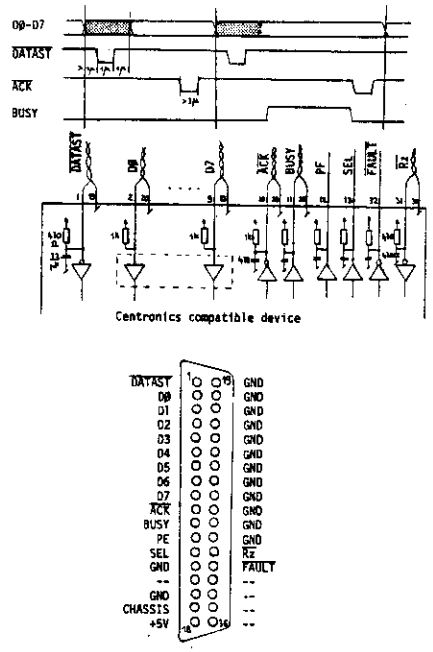

Fig. 1.9. Centronics interface protocol and pinout

It can be noticed that Centronics interface implies an input register, the information being allowed to desappear 1 μs after the data strobe signal. It can also be noticed that signals are inverted on the transfer lines, as it is usually the case with TTL technology. Open collector drivers are specified, with 470 ohm pull-up on receiving side, with a 33 pF shunt capacitor for filtering high frequency. Schmitt trigger gates are recommended. The use of twisted pairs allows cables of 20 meters with a great reliability.

A typical interface problem is to adapt a write protocol, e.g. connect a device using the full-handshaked protocol described in section 1.2.2 with a Centronics printer. The block diagram and the timing diagram of the interface is given in fig. 1.10. Assumption is made that the next data byte will not be requested as long as the previous one is not processed, that is no pipelining occurs.

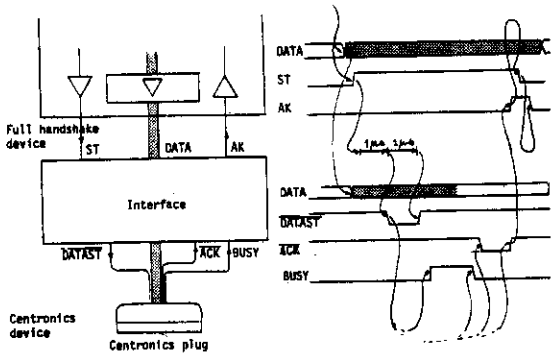

Fig. 1.10. Centronics interface: block and timing diagram

Even in this very simple example, many implementations are possible. Fig. 1.11 shows two schematics for the control logic. The first one (fig. 1.11a) uses one-shots for generating the 1 µs pulse. The second one (fig. 1.11b) uses delays.

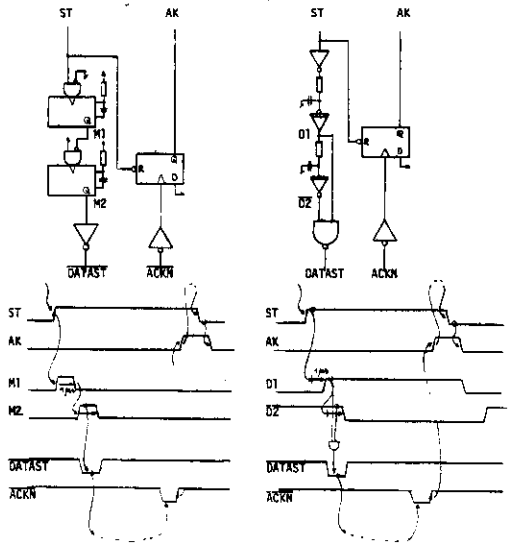

Fig. 1.11. Possible interface schematics for a Centronics interface

1.3.5. Full handshaked read protocol

In a read cycle, the commander requests information from the source device, and takes it when available. A full handshaked read protocol is given in fig. 1.12.

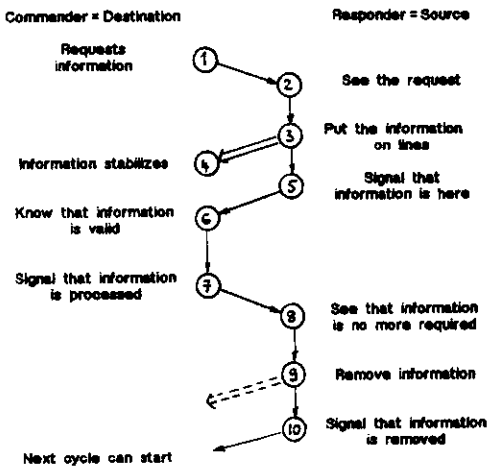

Fig. 1.12. Protocol for full handshaked read cycles

The corresponding timing diagram is given in fig. 1.13. It is rather similar to the write diagram, but it can be noticed that one more edge (1) is very important for the synchronisation of transfer, unless the source provides the information before that one is requested (cycle start at 6).

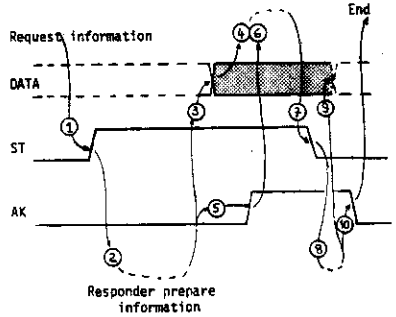

Fig. 1.13. Full handshake read protocol

1.3.6. Other read protocols

As for the write protocol, there are several read protocol, asynchronous and synchronous. No new point appears at that level.

1.4. Broadcast transfers

1.4.1. Broadcast write protocol

Sending information simultaneously to several responders requests to know when all responders have accepted the information. An AND function is required, (fig. 1.14 a) and can be performed with open-collector gates (wired-AND) (fig. 1.14 b).

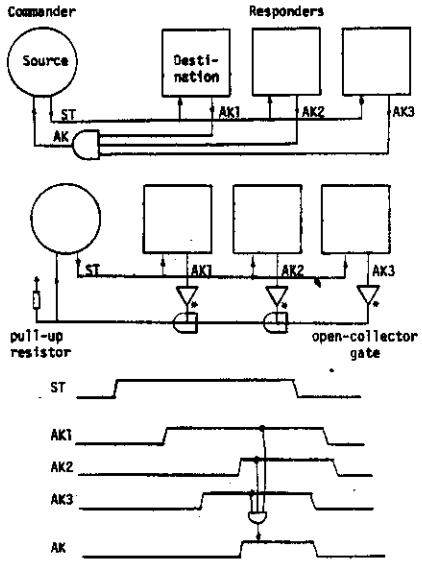

Fig. 1.14. Simple broadcast protocol

The protocol of fig. 1.14 has however a problem if the responder is slow deactivating its read line, and if the commander and the other reponders are fast. A new cycle may start and be acknowledged. Hence an additional line is required in order to know that all devices are ready for receiving new information. This has also the advantage that no transfer will start if some devices are not ready. It accelerates also the transfer, since the information is accepted as soon as the input registers of the responder have been loaded, and the bus may be prepared for the next transfers that will occur as soon as responders are ready.

It can be noticed that the OR function is implemented by inverting the signals and using the wired-OR scheme, equivalent to the wired-AND due to de Morgan theorem. Hence, BUSY* (BUSY, active low) is carried on the bus.

1.4.2. IEEE 488/IEC 625 protocol

The IEEE 488 / IEC 625 bus is a nice instrumentation bus getting now widely accepted. The transfer protocol is exactly as described in fig. 1.15, but the ST signal is inverted and named DAV* (data valid, low). The AK signal is named DAC (data accepted) or NDAC* (not data accepted, low). The BUSY* signal is named NRFD* (not ready for data, low).

The 488 protocol is frequently represented by a flow chart (fig. 1.16).

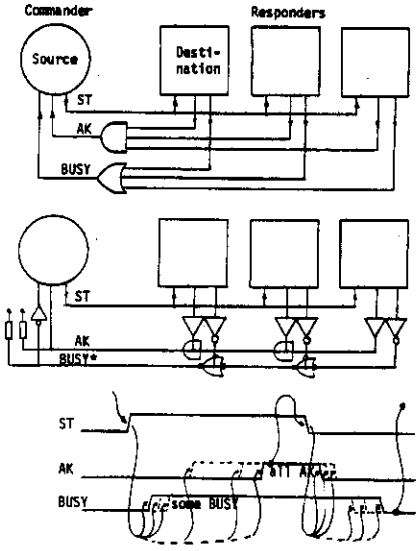

Fig. 1.15. Three-line full handshake broadcast protocol

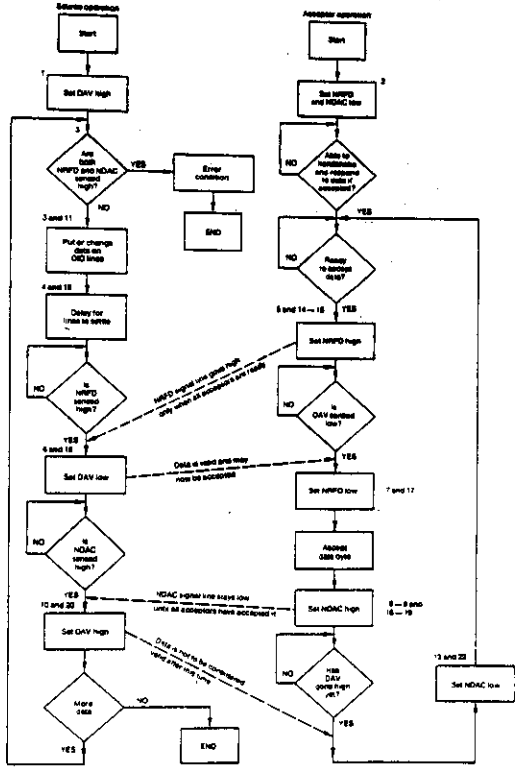

Fig. 1.16. IEEE 488/IEC 625 transfer protocol

The 488 transfers are simple since there is no arbitration and no selection. The commander (called the controller) assigns a source (called the talker) and one or several destinations (called the listeners) in a preliminary phase. Transfer is initiated by the source, and the addressing is implicit, the information being consecutive time dependant values, or formatted blocs with the address informations imbedded.

The not yet mentionned bus lines are

ATN Attention request	Activated by the controller for assigning new roles and new modes
EOI End of input	Activated by the Commander for stopping a transfer
REN Remote Enable	Activated by the Commander for enabling the control panel of each instrument
SRQ Service Requets	Activated by any device for interrupting the process
IFC Interface clear	General reset of the system

1.4.3. Broad-collect protocole

The synchronisation of the collection of information is similar to broadcast. A new problem is due to the fact that the information lines must either be splitted in groups given to all possible responders, or used in a wired-or manner with the zero having the priority over the ones. Broad-collect is used is some interrupt mechanisms, to get the interrupt vector (simplified 8080/8085/Z80 scheme) or to know the source of the request, each line of the bus being assigned to a possible source of request; it is hence possible to collect in a single cycle the status of many devices (used in FASTBUS).

1.5. Bidirectional transfers

1.5.1. Transfer on separate groups of lines

Using different groups of lines for the information written and read is simple, but expensive.
A possible implementation is shown in fig. 1.17, and is close from what has been used in the first S100 implementations.

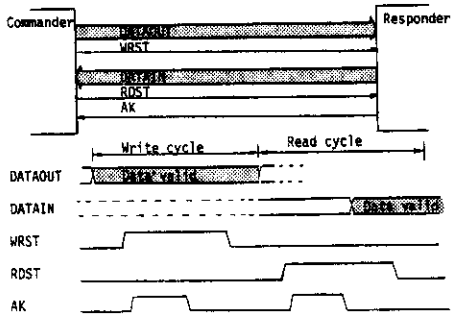

Fig. 1.17. Bidirectional transfer on separate groups of lines

1.5.2. Transfer on bidirectional lines

The base use of the information lines in the previous figure encourages a more efficient scheme, shown in fig. 1.18. The WR (write) line defines the direction of the transfer. The ST (strobe) and AK (acknowledge) lines are defined as before, and follow one of the handshake protocol, or the synchronous protocol without AK.

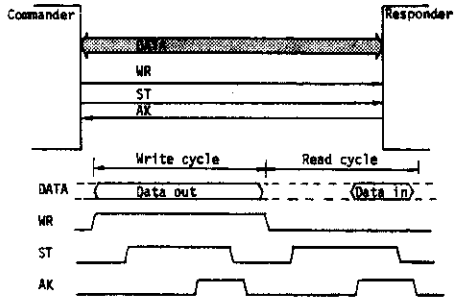

Fig. 1.18. Bidirectional transfer

Instead of the WR-ST pair, one can have, as on fig. 1.17, a WRST (write strobe) and RDST (read strobe). The ackowledge can also be duplicated.

Bidirectional transfers imply an adequate control of the bidirectional lines, in order to have them alternatively conducting in one or other direction. Open-collector or three-state gates have to be used, as shown in fig. 1.19.
Pull-up resistors must be added if open collector.
Impedance adapting networks must exist at both ends if cable length is important.

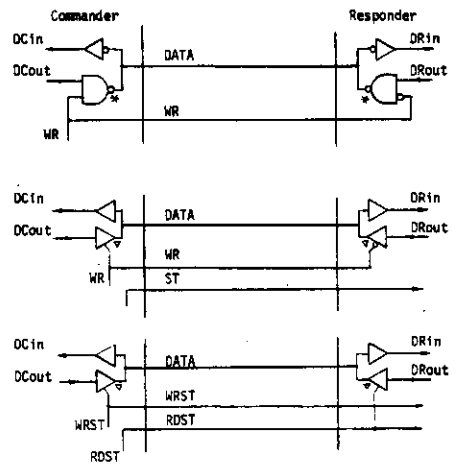

Fig. 1.19. Bidirectional line using open-collector and three-state gates

1.6. Example : bidirectional transfers using programmable interfaces.

Programmable interfaces can be programmed to be either inputs or outputs, and they have dedicated handshake lines, sometimes with several protocols.

The Intel 8255 has a bidirectional mode with four handshake lines. The Motorola 6821 has no bidirectional mode, but can be programmed to use PA alternatively as input and output, use CA1 and CA2 for handshake and any PB bit as a direction indication.

1.7. Selection

Before the transfer, the commander must have selected the responder with which it has information to exchange.

This can be done :

- implicitely : it has been assigned previously, e.g. by a controller as in IEEE 488
- by proximity : the closest responder interested in data transfer will be selected (see 1.8.2)
- by geographical address : each slot of the bus has an address and the responder inserted in a given slot (if there is one) will answer
- by data cell address : each data cell has an unique address intentifier within the system. The data cells of a same module have consecutive addresses or are in a same group of addresses, in order to recognize easily an address of the module, and a data cell sub-address within the module.

The last scheme is the most frequently used in computer systems and requires an address of 12 to 32 bits to be presented to all slaves, so the concerned responder is activated (fig. 1.10).

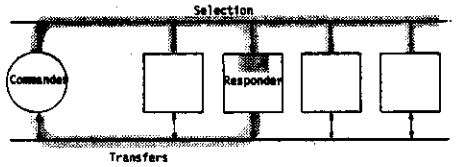

Fig. 1.20. Selection with an address

Selection is a broadcast write cycle and can be handled like this as a special first information cycle. This is the case with microprocessors having a multiplexed bus (TMS 9900, 8086, Z8000). Few 8-bit microprocessors have it, or they have it only partly because an 8-bit address is not sufficient (8085, 6803, COSMAC). Future 32-bit microprocessors (if any have that width on the bus level) will surely be multiplexed.

If there is no need for saving lines, the simplest is to keep the address during the complete transfer, and define combined selection/transfer cycles which are only partly handshaked, that is a pair of control signal edges are not associated with each change in the information (address and data) lines.

1.8. Arbitration

Arbitration occurs before selection, when several masters contend for getting the common resource of the bus lines. The arbiter can be centralized or distributed. In both cases there are critical problems to solve, specially of one wants to be fast.

1.8.1. Central arbiter

The simplest and more reliable way of designing an arbiter is to build a sequential clocked system which scans all the masters in turn, and gives the bus to the first one found having a request. This is however slow, and asynchronous arbitrations are attractive.

If there are only two contenders, the scheme of fig. 1.21 gives the priority to the first one. If they are absolutely at the same time, the circuit may hesitate for some time (metastable state) before giving the priority to one or the other.

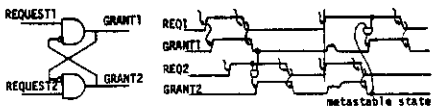

Fig. 1.21. Priority lock for two requests and timing diagram

The metastable problem is a difficult one in all arbiter designs, since requests come asynchronously with respect of each other, and with respect with the clock if the scheme is synchronised.

Metastable state last for a duration which follows probability rules. More than 100 µs may be considered as highly improbable with flip-flops of TTL or N-MOS technology.

1.8.2. Distributed arbiter

The arbitration process may be distributed along the bus; this provides a greater flexibility of design and usually a lower cost, but it is slower.

Two distributed arbitration schemes are of a great interest. The daisy-chain (fig. 1.22) is frequently used, and is just a selection scheme which selects the first requesting device, in a geographical order depending on the wiring.

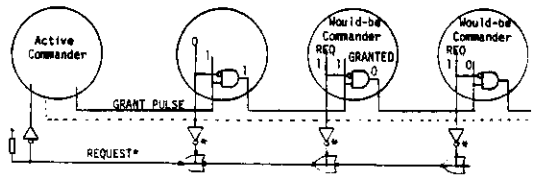

Fig. 1.22. Priority daisy chain

The would-be commanders set their requests on the bus, using a wired-OR scheme. The active commander, when finished, issues then a grant pulse which is gated by each master on the bus. It is clear from the schematic that the first would-be commander on the bus will get the pulse, and will cut the chain.

If the chain is closed, the successive commanders will be in rotating positions. With "old" computer systems, there is often a permanent master at the beginning of the chain, and the control of the bus is passed back to it by each commander.

A more symetrical scheme is the self-selection arbiter used in Fastbus , in IEEE-P696 (S-100) , and in P896 . It makes use of a set of wired-or priority lines, and compares the values on the lines with the value which settles on the bus.

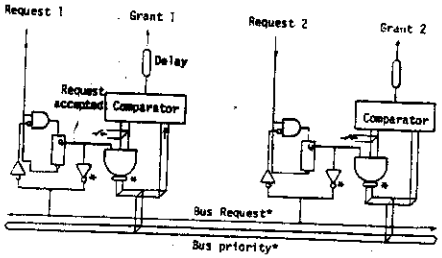

Fig. 1.23. Self-selection arbitration scheme

The priority vectors are encoded or not. If encoded, comparison is done on a bit per bit basis.

A common request line is activated each time an arbitration phase starts. As soon that request has been recognized, no new priority vector is allowed.

A delay or a clock allowsto know when the comparison phase is over in each module; the winning master assigns then a "Bus Busy" line, letting everybody know there is a new commander in the system.

1.9. Interrupt

Interruption is simple in a system having a single master. The interrupt request is directed to that master using a wired-or scheme. The master then starts when it want an identification procedure which can be either software driven (polling) as with simple microprocessor systems, or use some additional hardware.

Hardware identification of the source of interrupt can be done by :

- broad-collect cycle : requesting devices activate the data bus line(s) they have been attributed to
- proximity addressing : a daisy chain allows the requesting device which is the closest from the commander to put an identification word called vector on the lines. This vector can be the address of the interrupting device, or the address (direct or preferably indirect) of the routine which must be executed for servicing the interrupt
- multiple interrupt request lines : each requesting device is attached to a different line, and the arbitration between the requests, according some priority mechanism, is done by the processor or some associated circuitry.

2. Serial transfers

This incomplete chapter will only consider asynchronous serial transfer, using the RS232/V24 standard.

2.1. Asynchronous transfer

The asynchronous transfer provides both bit and byte synchronisation by a start bit followed by the serial information sent at a fixed known rate, often named the Baudrate instead of the bit rate. One stop bit at least must follow the 8-bit transfer (9-bit if parity is included), in order to be able to recognize the next start bit.

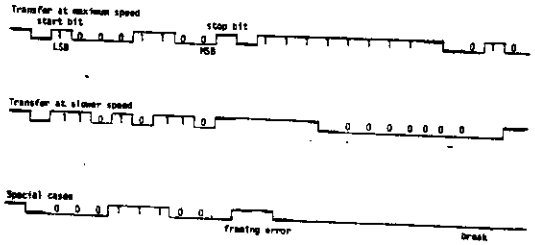

Fig. 2.1. Typical serial transfer timing diagram.

Asynchronous transfer used in the past a 20 mA loop, closed for logical 1, open for logical 0 state. The "break character", a continuous stream of zeros, correspond to an open line, and is still used to signal an error or abort a communication.

The RS232 standard is voltage driven. Logical 0 is +3 to +15 V, logical 1 is -3 to -15 V. RS422 uses lower voltages on balanced lines and can be faster.

RS232 uses a 25-pin plug and a simple convention: Data Communication Equipement (DCE) have a female plug with Data Out on pin 3 and Data In on pin 2; all Data Terminal Equipment (DTE) have a male plug with Data In on pin 3 and Data Out on pin 2.

Few control lines are for modem control and allow some handshake on the transmission, but this handshake may be delayed by few characters because of the double or triple buffering found in most serial programmable interfaces.

A special box called null-modem must be used if a terminal must be connected on a male plug planned to be linked to a modem. It is a passive box in which the data lines and several control lines are exchanged. Similarly, a null-terminal is used if a modem is to be connected on a female plug planned to be linked to a terminal (fig. 2.2).

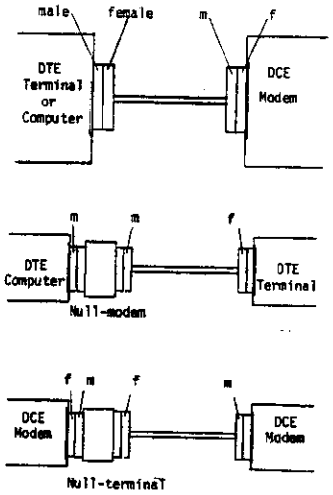

Fig. 2.2. RS232 plug conventions, null-modem and null-terminals

Unhappily these clean rules are not followed by many manifacturers, and one should double check where data and control lines are connected to.

3. Microprocessor interfacing

All microprocessors have a very similar architecture, with an address bus and a data bus, often partly multiplexed, and three groups of control lines for transfer control, interrupt and bus mastership control.

3.1. Transfers

No microprocessors have block transfer, and only few have read-modify-write instructions. Hence, a combined selection and transfer scheme is adequate, and full handshake is not used due to the synchronous operation of the rpocessors and the technological assumptions made by all manufacturers in order to simplify the logic and speed-up as much as possible.

3.1.1. Synchronous transfers

The simplest possible read and write timing is used in the M6800 (fig.3.1a). A single signal called E or ϕ2 is used to strobe the transfers. No handshake occurs; the responder has to take or provide information on time. If the responder is too slow, the only possibility is to slow the E=ϕ2 signal, which is the clock for the system.

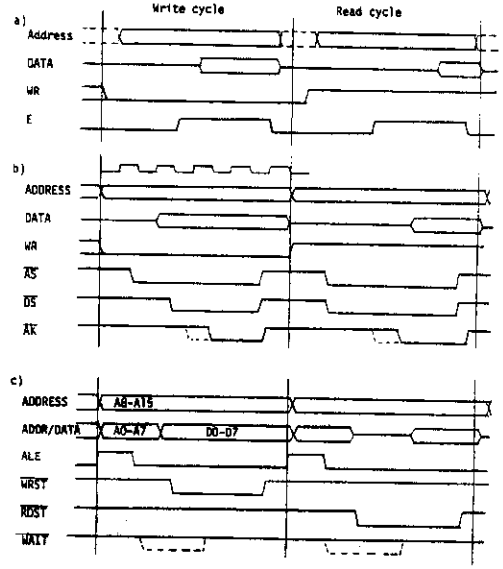

Fig. 3.1. Microprocessor transfer timings
a)6800 b)68000 c)8085

3.1.2. Synchronized handshake

A better scheme is used in the M68000, which first has two strobes due to the possible RMW cycles : AS (address strobe) and DS (data strobe), and an AK (acknowledge) signal, which acknoledges the transfer of the data, but not the selection of the address (fig. 3.1b). The DS-AK sequence is very similar to the one of the full handshaked transfers, but is synchronised by the clock: the clock is high frequency (4 or 5 clock pulses for a read or write cycle), and the AK signal is sampled periodically in the clock cycles following AS active ($\overline{AS}$ low). As long AK is not active ($\overline{AK}$ low), "empty" slots of duration of one clock cycle are inserted, in order to allow the responder to process or prepare information. The continuation of the cycle, with DS being deactivated, occurs on one of the next clock cycles following the deactivation of AK.

A very similar scheme is used on the 8085 (fig. 3.1c), but instead of an acknowledge signal, there is a WAIT signal, which has to be "quickly" activated (technological constraint) if empty cycles have to be inserted. The advantage of the WAIT line is that if all responders are fast enough (which is often the case), that line can be ignored (that is connected to a pull-up resistor)

3.2. Selection

Selection of a data cell is done from the address, and immediately the transfer starts, as shown in fig.3.2.

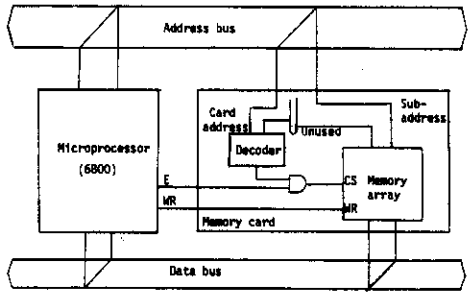

Fig. 3.2. Selection of a responder (e.g. a static memory)

This very simple scheme supposes that the address is stable long enough before the strobe pulse E, so that both the card address decoder, and the data cell decoder in the memory array must be stable before before the pulse, in order to avoid transient selections.

Both the processor and the memory use three-state outputs in order to allow bidirectional transfers. No adapting resistor is required on the local microprocessor bus; the processor output would be unable to drive it anyway.

3.3. Example : selection of mixed peripherals

A typical microprocessor system is built with a microprocessor, its memory and basic I/O on a board, and its extension connector for additional I/O, MUBUS-26 is very simple in its I/O-only non-handshaked implementation. Its 26-pin connector carries the signals shown on fig.3.3. The P signal is generated from the E signal and a group of addresses decoded on the board. An additional decoder defines 8 subgroups of 4 consecutive addresses, either read or write since the W line is coming on an address input of the decoder.

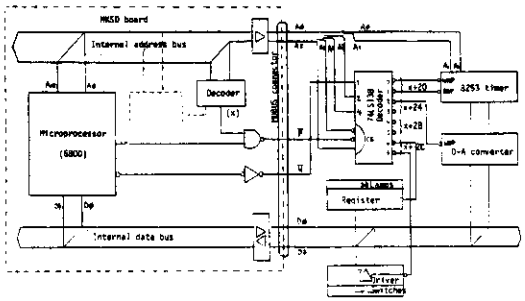

Fig. 3.3. Schematic for selection of typical peripherals

Write and read selection pulses, at the Intel way, are generated by the decoder, which is exactly what is required for loading registers or enabling input data drivers.

The example shown allows to select an Intel 8253 timer (see 3.5.5), which internally decodes the two low address bits, a DA converter (equivalent to a register), a register controlling for instance a set of 8 lamps, and an 8-bit driver allowing to read 8 switches.

It can be noticed that since A1A0 are not used for the selection of the D-A converter, for instance, this one will respond to the 4 consecutive addresses. By adding a 4-output decoder, one could select 4 different D-A converters, register-equivalent devices, but one should be aware that all these decoders add delays, and both the processor and the responders have maximum limits, since no handshake occurs.

The memory map, that is the software model of this interface, is given in fig.3.4. It shows the addresses assigned by the hardware to each I/O cell, the partly decoded area, and the free space. It can be refined to show the meaning of each bit.

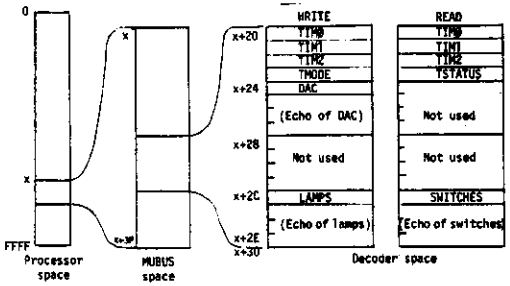

Fig. 3.4. Memory map of the interface

3.4. Programmed transfers

Synchronization of transfers has been shown earlier at the hardware level. If now the software must know when new information must be transferred with the interface, a semaphore must be used. The semaphore is a flip-flop which is set by the interface, when new information must be transferred between the interface and the processor, and is reset by the processor (that is by an instruction of the program, when executed by the processor). The semaphore can be regularly checked by the processor (programmed transfers) or may create an interruption.

3.4.1. Output programmed transfer

A simple write interface is given in fig. 3.5. The register is loaded by the registers and simultaneously a Centronics compatible strobe pulse is generated. When the printer sends its acknowledge, the READY flip-flop is activated, and the processor can read this value to know the status of the transfer. The READY flip-flop is cleared when the processor writes the next information for the printer.

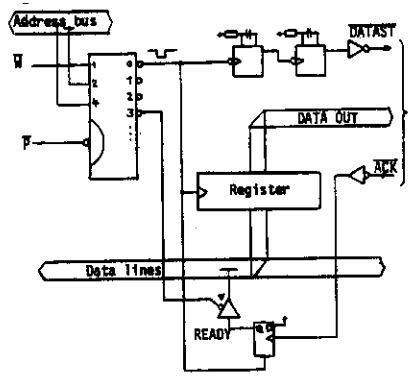

Fig. 3.5. Output interface with handshake

The initialisation of the system (Reset) usually activates Ready; the printer is indeed ready to receive the first character; the program is not forced to send it, even if READY = 1.

The software routine for sending a character is simply

```
WRBYTE  WAIT UNTIL READY
        WRITE DATA
```

In assembly language, one should declare the address of the status register which has to be read for getting the READY bit, and one should declare the address of the READY bit in the byte (frequently the most significant bit is used for semaphores for saving one instruction).

3.5. Input programmed transfer

Input transfers are handled very similarly to output transfers. The schematic is given in fig.3.6.

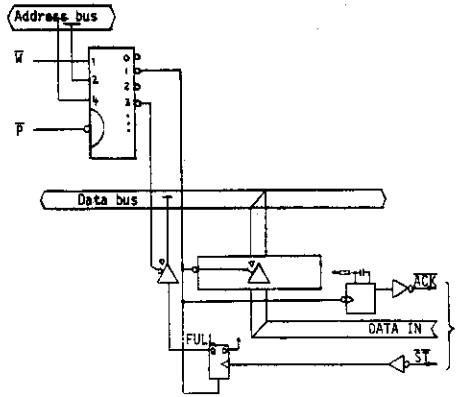

Fig. 3.6. Input interface with handshake

The FULL semaphore is cleared by Reset and each time the output drivers are read. The strobe of a keyboard, paper tape reader, etc. set FULL active. Processor can know (by reading the status register) that a new information is waiting, and the paper reader will not send the next character as long as FULL is active. The software routine for getting characters is

```
RDBYTE  WAIT UNTIL FULL
        READ DATA
```

It can of course be noticed that when the processor is in that routine waiting for FULL, it can only do what is in the wait loop.

Synchronization of serial transfers is done in a very similar manner

3.6. Interrupt

Instead of having the program calling the RDBYTE routine each time it needs or may get a character, the hardware may call the routine for it when the character is actually here. That is the FULL flag interrupt directly the processor (fig.3.7) and forcing the call of a routine at a predefined address if interrupt is enabled (that is interrrupt mask is cleared).

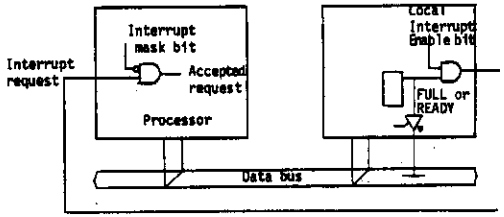

Fig. 3.7. Principle of interruption

Interrupt is automatically desabled when the interrupt routine is terminated by a special return instruction, in order to reactivate the interrupt before returning to the main program.

Interrupt line is usually a wired-or line, which can be activated by several sources. The processor in this case must check all the full and ready semaphores in the system, by reading the corresponding status registers, and servicing them.

In complex systems, the interrupt request of each device is enabled on the bus by a mode bit (local interrupt enable flip-flop). This allows the programmer to define who is able to interrupt at a given instant of time.

Multiple interrupt lines or vectored interrupt simplify the programming since each interrupt is more quickly directed toward the corresponding service routine.

3.7. Direct memory access

Present microprocessors are not really oriented toward multimaster systems as explained in section 1. They can lend the bus to another temporary commander, usually called a DMA device, because this device will then get a direct access to memory, instead to have to be read by the processor. DMA is handled on the processor side by two lines: a HR line (Hold Request) and a HA line (Hold Acknowledge) output meaning the processor has stopped and bus is free for the DMA requesting device.

A DMA interface is usually made of an post-incrementing register and a counter for creating an interrupt when the transfer is terminated. DMA cycles can be interleaved, between the processor cycles or contiguous, depending on the application.

3.8. Programmable interfaces

Present technology allows to multiply the number or registers and complicate the logic in a chip.

For saving pins and still giving the flexibility for the user, different functions can be programmed on the same pins. For instance, the input/output direction of a parallel port can be modified by adding a direction mode bit (fig. 3.8). If the direction bit is active, the I/O pin is an output.

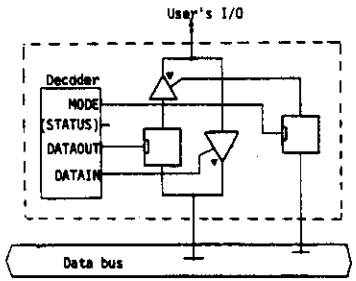

Fig. 3.8. Direction control in a programmable interface

The direction of a group of 8 lines may be controlled by a single mode bit, or each line may be individually controlled as the PIA 6820.

Several mode bits may be associated with a single line to multiplex it on several functions (parallel, serial, timer, etc.). At the limit, a single chip could do anything!

3.8.1. Addressing schemes

Several addressing schemes are used in programmable interfaces, for reducing the number of address selection lines or simplifying the handling of the many registers by the program.

- Direct addressing, as explained in section 3.2, is used in devices like the M6850, M6840, I8255.
- The sub-address of a group of registers can also be prepared in a control register. This is the case of the PIA 6821, where the selection of the data or direction register depends on a bit of the mode register. This is also the case of the M6845 (18 control registers addressed by a 5-bit sub-address in a control register).
- The address can be a part of the written data word. This is not applicable to interrupt registers, but allows in place of storing one 8-bit word, to store two 7-bit words, four 6-bit words, eight 5-bit words, etc. This is used for instance in the I8253 and the AMD 9519.
- The last basic scheme is to have the address in a counter. Consecutive transfers access consecutive registers. A bit of the control register allows to reset the counter and restart the adressing from a known state.

Programmable interfaces are getting as complex as microprocessors. Most of the interfacing problem is already concentrated in them.